Ecole Nationale Supérieure des Télécommunications
Département Traitement du Signal et des Images
Ecole doctorale Edite

Universidad del País Vasco
Euskal Herriko Unibertsitatea
University of the Basque Country
Computer Engineering Faculty
Intelligent Systems Group

Thèse de Doctorat
Spécialité Signal et Images

Inexact Graph Matching Using Estimation of Distribution Algorithms

Mise en Correspondance Inexacte de Graphes par Algorithmes d'Estimation de Distributions

Endika Bengoetxea

Directeurs de Thèse : Isabelle Bloch
Pedro Larrañaga

19 décembre 2002

Acknowledgements

There are some people to whom I am deeply indebted for their help in the process that ends with this PhD dissertation. I take this opportunity to acknowledge their never faltering encouragement and support.

First of all, I wish to thank my two PhD supervisors Isabelle Bloch and Pedro Larrañaga for offering me the opportunity to do this PhD in collaboration between their laboratories. Special thanks also for their wise supervision, guidance, patience, and friendly encouragement.

I also owe a debt of gratitude to my colleagues of the Intelligent Systems Group of the Department of Computer Science and Artificial Intelligence of the University of the Basque Country in San Sebastian, as well as to the members of the Image Processing and Interpretation Group from the Department of Signal and Image Processing at the Ecole Nationale Supérieure des Télécommunications for their help on the development of this thesis, their friendly welcome, and their patience with my spoken and written French.

My gratitude also to Roberto Cesar, from the University of São Paulo in Brazil, for his helpful collaboration and dedication in many of the image processing parts of this thesis as well as in providing me the data required for a successful completion of the experiments. I would like to thank Jose Miguel and Alex Mendiburu from the Department of Computer Architecture and Technology for their support and contribution in the parallel programming section, as well as Aymeric Perchant, Claudia Boeres, and Jérémie Pescatore for their useful advise and contribution to this thesis.

Special thanks also for the members of the Deanship of the Computer Engineering Faculty at the University of the Basque Country, specially to the Dean Iñaki Morlan and the Vicedean Teresa Miquélez, for their understanding and confidence on me while I was working in the Deanship and on this PhD thesis at the same time. They, as well as many other colleagues, family and friends have *suffered* the stress of carrying out this PhD thesis in three years and at the same time being working as a lecturer, Erasmus coordinator, and Academic Secretary in my home Faculty.

Also, I would like to thank a lot the interest and the useful comments of the two reviewers of the PhD dissertation Jean-Michel Jolion and Michele Sebag, as well as the rest of the members of the jury Olivier Cappe, Yves Sorel and Jose Antonio Lozano for the time dedicated for the evaluation of this PhD.

Finally, the author would like to acknowledge the financial help and the support of these institutions which have supported partially this dissertation:

- The Spanish Ministry for Science and Education, with project HF1999-0107
- The French Ministry for Education, Research and Technology with project Picasso-00773TE.
- The Spanish Ministry for Science and Technology with project TIC2001-2973-C05-03.
- The University of the Basque Country, with the projects UPV140.226-EB131/99 and 9/UPV/EHU 00140.226-12084/2000
- The Department of Education Universities and Research of the Basque Government, with project PI 1999-40

Résumé

Motivation

L'objectif de cette thèse est de contribuer au développement de méthodes de reconaissance d'objets dans les images, s'appuyant sur des informations de haut niveau, sur la structure de la scène (topologie, organisation spatiale des différents objets dans l'image).

Au cours des dernières années les graphes ont été utilisés pour représenter mathématiquement la connaissance structurelle. L'intérêt de ce type de représentation va en augmentant dans la communauté scientifique grâce à la possibilité d'utiliser cette représentation en combinaison avec des algorithmes de mise en correspondance de graphes (*graph matching* en anglais), car cette représentation permet de représenter les variations et les différences structurelles entre objets différents.

Différentes applications ont été proposées en utilisant ce type de représentation pour réaliser la reconnaissance automatique des images. Dans ce type d'applications, la connaissance sur les variations structurelles possibles des objets qui apparaissent dans les images (personne, visage, image médicale, etc.) est exprimée à travers un modèle que l'on représente comme un graphe (ou atlas, comme on l'appelle dans certains cas). La cartographie, la reconnaissance de caractères, et la reconnaissance de structures cérébrales à partir d'images de résonance magnétique sont des exemples de domaines dans lesquels ce type de représentation a été utilisé et publié. Ce dernier exemple est illustré figure 1.

Le *graphe modèle* est souvent construit en utilisant un nœud pour chacune des régions de l'objet à reconnaître (Ex: dans le cas d'images d'un cerveau humain, il y aura un nœud pour représenter le cervelet, un autre pour chaque ventricule...) et les arêtes pour représenter les relations entre ces régions. On utilise des attributs pour exprimer les propriétés de chaque nœud ou arête, et pour pouvoir les identifier ainsi que les distinguer les uns des autres. Ce graphe est souvent construit de manière supervisée.

Après la construction d'un graphe modèle, et pour pouvoir réaliser la reconnaissance d'images à travers lui, il est nécessaire de construire un graphe à partir de chacune des images à reconnaître. Ces graphes, qui sont dénommés dans la littérature *graphes de données* ou *graphes d'entrée*, sont automatiquement construits par l'ordinateur sans l'assistance de l'utilisateur. Dans ce processus de construction du graphe de données correspondant à une image, une étape de segmentation de l'image précède la construction du graphe et est fondamentale : le graphe de données est formée en utilisant un nœud pour représenter chacun des segments obtenus. Les mêmes attributs que ceux du graphe modèle sont calculés pour les nœuds et les arêtes du graphe de données.

Mise en correspondance de graphes

La reconnaissance des structures de l'image est réalisée en recherchant la meilleure correspondance entre les graphes modèle et de données. Il s'agit d'affecter une étiquette à chacune des régions de l'image à reconnaître, les étiquettes étant définies par les régions du graphe modèle. La figure 2 illustre le principe de la mise en correspondance.

En notant $G_M = (V_M, E_M)$ le graphe modèle et $G_D = (V_D, E_D)$ le graphe de données, le problème consiste à trouver un homomorphisme $h : V_D \rightarrow V_M$ tel que $(u, v) \in E_D$ si et seulement si $(f(u), f(v)) \in E_M$. Ces homomorphismes sont évalués en calculant la similarité entre les attributs des nœuds et des arêtes mis en correspondance. On cherche alors un meilleur homomorphisme au sens d'un certain critère combinant ces similarités.

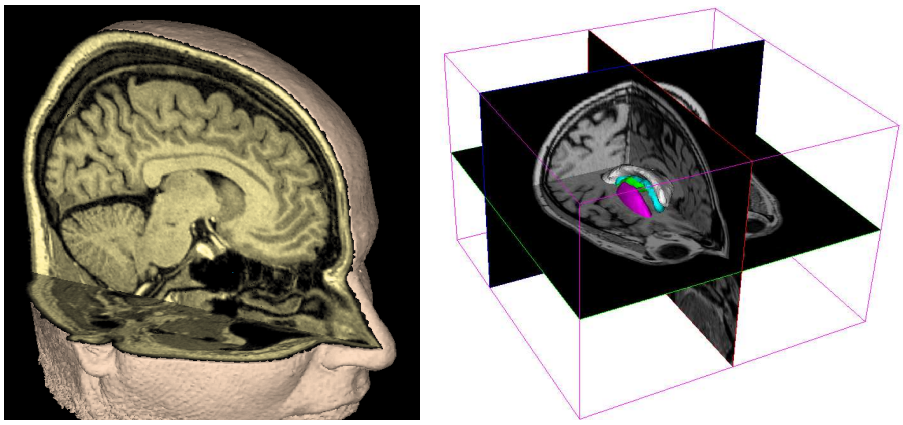


Figure 1: Exemple d'application de la mise en correspondance de graphes appliquée à la reconnaissance d'images. L'objectif est de reconnaître les différentes parties d'un cerveau humain à partir d'une image 3D comme celle de gauche.

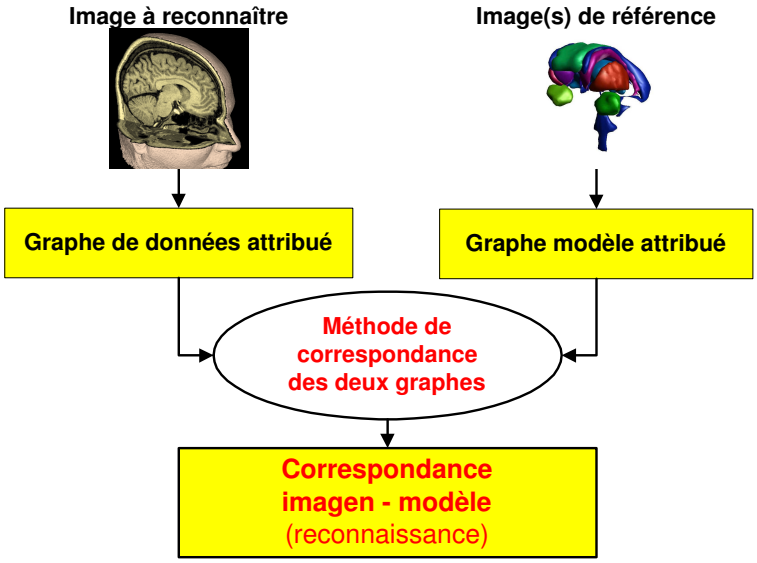


Figure 2: Principe de la mise en correspondance des graphes modèle et de données pour la reconnaissance de structures cérébrales.

Une grande partie de la littérature est consacrée à la recherche d'isomorphismes de graphes, ce qui suppose en particulier que $|V_M| = |V_D|$. On parle alors de correspondance exacte. Cependant, dans beaucoup de problèmes, surtout pour ceux où la segmentation s'effectue automatiquement, la condition d'isomorphisme est trop stricte et le nombre de nœuds du graphe de données est supérieur à celui du modèle ($|V_M| < |V_D|$). On parle alors de *correspondance inexacte de graphes*, et la complexité de ces problèmes est supérieure à celle des précédents. C'est à ce type de problème que nous nous intéressons ici.

La figure 3 illustre les différents types de problèmes de mise en correspondance de graphes. Dans les problèmes de mise en correspondance inexacte, on utilise souvent :

- un nœud nul ou *dummy* représentant la non-reconnaissance (par exemple pour une

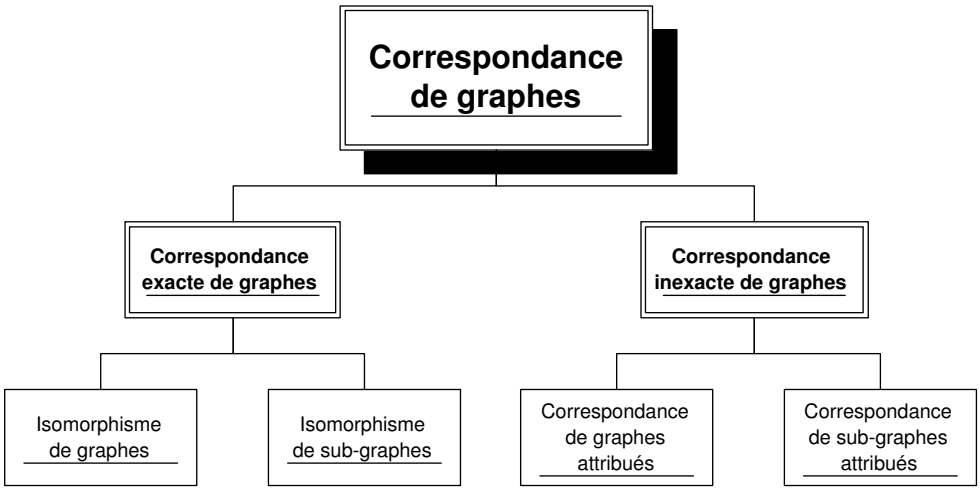


Figure 3: Classement des problèmes de mise en correspondance de graphes en deux classes principales.

région de l'image correspondant à une structure non représentée dans le modèle) ;

- des correspondances multiples entre nœuds : ces types de problèmes sont les plus complexes, car il est possible qu'un segment de l'image à reconnaître fasse en même temps partie de plus d'une région de l'objet à traiter. Bien que ce type de problèmes puisse exister, leur résolution implique plus de combinaisons et la recherche pour trouver la solution optimale peut être de complexité trop élevée pour pouvoir être traitée.

Beaucoup de techniques très différentes ont été développées pour la mise en correspondance de graphes : arbres de décision [Messmer and Bunke, 1999], réseaux de neurones [Riviere et al., 2002], algorithme EM [Cross and Hancock, 1998], relaxation probabiliste [Christmas et al., 1995], heuristiques et métaheuristiques [Pelillo et al., 1999], algorithmes génétiques [Wilson and Hancock, 1997], et programmation évolutive [Singh and Chaudhury, 1997]. Beaucoup de ces techniques restent restreintes aux correspondances exactes de graphes. Cependant, dans beaucoup d'autres exemples, surtout en ce qui concerne les applications aux images réelles, la condition d'isomorphisme est trop forte, cependant les approches pour résoudre les problèmes de correspondances inexactes sont actuellement moins répandues.

Les problèmes considérés dans cette thèse sont des problèmes de mise en correspondance inexacte de graphes, sans ajouter de nœud nul. En outre, dans le but de ne pas augmenter la complexité de problèmes qui sont déjà complexes, il a été décidé de réaliser une sursegmentation de l'image à reconnaître pour éviter la possibilité de correspondances multiples entre graphes. La méthode proposée pour réaliser la mise en correspondance repose sur les algorithmes d'estimation de distributions (EDAs).

Formalisation de la mise en correspondance de graphes comme un problème d'optimisation combinatoire

A cause de la grande complexité du problème de mise en correspondance inexacte, nous nous sommes tournés vers des algorithmes d'optimisation combinatoire sous contrainte. Cette thèse analyse les aspects et mécanismes qui doivent s'appliquer pour pouvoir formuler le

problème de cette manière et ensuite appliquer des algorithmes tels que les Algorithmes d'Estimation de Distributions (EDAs).

Les caractéristiques suivantes doivent être définies :

- **une représentation des individus** ou des solutions, de manière à représenter chacun des points dans l'espace de recherche ;
- **une fonction objectif à optimiser** qui affecte une valeur à chaque solution possible ou individu, pour exprimer la validité et la qualité de l'individu comme solution au problème.

Chaque individu est composé d'un vecteur de valeurs qui peuvent être discrètes ou continues. Les EDAs permettent d'utiliser les deux types d'individus contrairement à d'autres algorithmes. Cette thèse présente la façon de travailler avec les deux types d'individus et de représentations afin de les appliquer dans des problèmes de mise en correspondance de graphes.

Une représentation possible des individus consiste à associer à chaque nœud de G_D un nœud de G_M. Donc, la taille des individus est déterminée dans ce cas par $n = |V_D|$ variables[1], $X_i \in \boldsymbol{X}\ i = 1, \ldots, |V_D|$, où chaque variable contient une valeur entre 1 et $|V_M|$. La valeur de chaque variable dans l'individu a la signification suivante : $X_i = k$ avec $1 \leq i \leq |V_D|$ et $1 \leq k \leq |V_M| \Leftrightarrow$ le i-ème nœud de G_D est identifié comme le k-ème nœud du graphe modèle G_M.

Cette thèse propose deux autres représentations d'individus dans le domaine discret, ainsi qu'une représentation pour le domaine continu. En outre, et avec l'intention de montrer la robustesse de la méthode proposée, une contrainte supplémentaire que doivent accomplir les individus pour pouvoir être pris en considération est ajoutée :

$$\forall a_M \in V_M,\ \exists a_D \in V_D \ \mid \ h(a_D) = a_M$$

On impose ainsi que toute structure du modèle soit identifiée dans l'image. Cette restriction augmente ainsi la complexité des problèmes et celle des différents mécanismes présentés dans cette thèse. Elle permet de montrer comment on peut tenir compte des restrictions quand on applique les EDAs.

Enfin, dans le but d'évaluer la qualité de l'individu comme solution possible à un problème, on définit deux fonctions $c_N(a_D, a_M)$ et $c_E(e_D, e_M)$ qui mesurent respectivement la similarité entre deux nœuds $a_D \in V_D$ et $a_M \in V_M$, et entre deux arêtes $e_D \in E_D$ et $e_M \in E_M$. Nous avons défini plusieurs fonctions objectif dépendant de ces fonctions de similarité.

Algorithmes d'Estimation de Distributions

Les Algorithmes d'Estimation de Distributions (EDAs) sont une approche récente dans la famille des algorithmes évolutifs appliqués aux problèmes d'optimisation. Ces algorithmes se caractérisent par un procédé d'évolution d'un ensemble de solutions. A cause de la similitude avec des populations d'êtres vivants, on appelle d'habitude *individu* chacune des solutions, *population* l'ensemble d'individus, et *génération* chacune des populations qui se développent successivement. La différence la plus significative entre les EDAs et les Algorithmes Génétiques (GAs), qui sont parmi les plus connus et utilisés dans ce domaine, réside

[1]Dans le domaine des algorithmes génétiques on considère que les individus sont composés de gènes, mais dans le domaine des EDAs on parle plutôt de variables probabilistes.

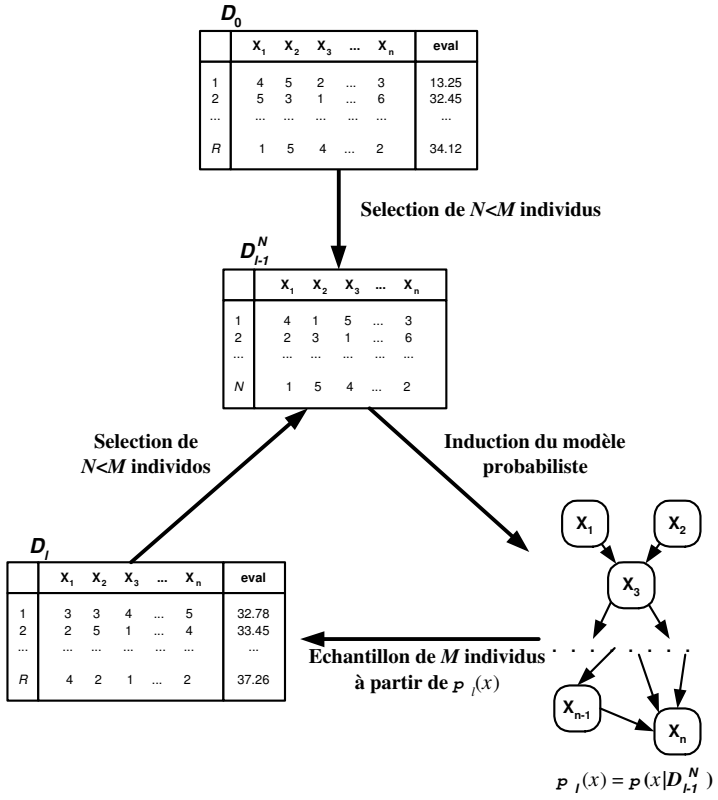


Figure 4: Illustration de l'approche EDA en optimisation.

dans le processus d'évolution d'une génération à la suivante : elle est réalisée dans le cas des GAs par des opérations de mutation et de croisement, et dans les EDAs par des techniques fondées sur la théorie des probabilités et plus particulièrement sur l'apprentissage et la simulation de réseaux bayésiens ou de réseaux gaussiens. Cette idée est illustrée figure 4.

La figure 5 présente le pseudocode générique des EDAs qui suit essentiellement les étapes suivantes :

1. La population initiale D_0 formée par R individus est créée. La génération de ces R individus est souvent réalisée en supposant une distribution uniforme pour chaque variable. Une fois que les individus sont engendrés, ils sont évalués selon la fonction objectif.

2. Pour faire évoluer la i-ème population D_{l-1} vers la suivante D_l, N individus sont sélectionés dans D_{l-1} ($N < R$) en suivant un certain critère. Nous appelons D_{l-1}^N l'ensemble des N individus sélectionnés dans la génération numérotée $l-1$.

3. Le modèle probabiliste graphique n–dimensionnel qui représente le mieux les dépendances entre les n variables est déterminé. C'est l'étape d'*apprentissage*, qui est cruciale dans les EDAs à cause de l'importance de la prise en compte de toutes les dépendances entre les variables pour assurer une évolution satisfaisante vers des individus meilleurs.

4. La nouvelle population D_l est constituée avec R nouveaux individus obtenus après avoir simulé la distribution de probabilité apprise dans l'étape précédente. On utilise souvent une approche élitiste, et ainsi le meilleur individu de la population D_{l-1}^N est

EDA

$D_0 \leftarrow$ Engendrer R individus (la population initiale D_0) au hasard

Répéter pour $l = 1, 2, \ldots$ jusqu'*à* satisfaire un critère d'arrêt

$D_{l-1}^N \leftarrow$ S $N < R$ individus de D_{l-1} en suivant
une méthode de selection déterminée

$\rho_l(\boldsymbol{x}) = \rho(\boldsymbol{x}|D_{l-1}^N) \leftarrow$ Estimer la distribution de probabilité
qu'un individu se trouve entre les individus selectionnés

$D_l \leftarrow$ Echantillonner R individus, la nouvelle population, *à* partir de $\rho_l(\boldsymbol{x})$

Figure 5: Pseudocode générique des EDAs.

gardé dans la nouvelle population D_l. De cette manière, dans chaque génération un total de $R-1$ nouveaux individus sont créés au lieu de R.

Les étapes 2, 3 et 4 sont répétées jusqu'à satisfaire une condition d'arrêt. Des exemples de critères d'arrêt sont : arriver à un nombre de génération maximal, atteindre un nombre maximal d'individus analysés, uniformité dans la population, ou le fait de ne pas obtenir un individu avec une valeur de fonction objectif meilleure après un certain nombre de générations.

Un grand nombre d'algorithmes qui font partie des EDAs a été proposé dans la littérature. On peut les classer en trois grands groupes selon la complexité du type de dépendances entre variables considéré :

- **Sans dépendance entre variables :** la structure du réseau bayésien (ou gaussien si nous travaillons dans le domaine continu) est fixe et ne contient pas d'arcs. En d'autres termes, cela signifie que toutes les variables de cet individu sont considérées indépendantes entre elles.

  Comme exemple d'algorithmes du domaine discret qui appartiennent à ce groupe nous pouvons mentionner UMDA (Univariate Marginal Distribution Algorithm) [Mühlenbein, 1998] où l'estimation de la distribution de probabilité est réalisée de la manière suivante :

  $$p(\boldsymbol{x}) = \prod_{i=1}^{n} p(x_i)$$

  Un autre exemple est l'algorithme connu comme UMDA$_c$ (Univariate Marginal Distribution Algorithm - continuous) [Larrañaga et al., 2000], qui est l'équivalent de l'UMDA dans le domaine continu.

- **Dépendances entre paires de variables :** dans la structure du réseau bayésien ou gaussien chacune des variables peut avoir au plus un parent. Cela nécessite donc une étape d'estimation de la meilleure structure.

  Un exemple d'EDAs pour le domaine discret appartenenant à ce groupe est MIMIC (Mutuel Information Maximization for Input Clustering) [de Bonet et al., 1997] qui propose de réaliser la factorisation de probabilité suivante :

  $$p(\boldsymbol{x}) = p(x_{i_1} \mid x_{i_2}) \cdot p(x_{i_2} \mid x_{i_3}) \cdots p(x_{i_{n-1}} \mid x_{i_n}) \cdot p(x_{i_n})$$

MIMIC recherche la permutation $\pi = (i_1, i_2, \ldots, i_n)$ qui minimise la divergence de Kullback-Leibler entre la distribution estimée $\hat{p}_\pi(\boldsymbol{x})$ et la distribution réelle $p(\boldsymbol{x})$.

De nouveau, il existe une version continue de MIMIC nommée MIMIC$_c$ (Mutuel Information Maximization for Input Clustering - continuous) [Larrañaga et al., 2000].

- **Dépendances multiples entre variables :** cette fois toutes les probabilités conditionnelles peuvent intervenir, et la structure du réseau bayésien ou gaussien est quelconque. Cette caractéristique demande une recherche exhaustive de la meilleure structure probabiliste graphique parmi toutes celles qui sont possibles, et donc ces algorithmes sont plus chers en termes de temps d'exécution que ceux des groupes précédents. Leur avantage est qu'ils sont capables d'apprendre une structure probabiliste graphique qui considère les relations entre les différentes variables des individus plus fidèlement.

  Citons comme exemple dans le domaine discret l'agorithme EBNA (Estimation of Bayesian Networks Algorithm) [Etxeberria and Larrañaga, 1999], et EGNA (Estimation of Gaussian Networks Algorithm) [Larrañaga et al., 2000, Larrañaga and Lozano, 2001] dans le domaine continu.

  L'agorithme EBNA utilise un *score* BIC (Bayesian Information Criterion) [Schwarz, 1978] qui mesure l'aptitude d'une structure à représenter les dépendances entre les individus. Le réseau bayésien qui maximise cette mesure est estimé par la méthode de l'Algorithme B [Buntine, 1991]. La factorisation de la probabilité s'écrit alors de la façon suivante :

$$p(\boldsymbol{x}) = \prod_{i=1}^{n} p(x_i | \boldsymbol{pa}(x_i))$$

  où $\boldsymbol{pa}(x_i)$ est l'ensemble des parents de la variable x_i dans le réseau bayésien.

Les algorithmes et approches d'EDAs qui sont proposés dans la littérature sont nombreux, bien que pour le moment il y n'ait que peu d'articles qui en montrent tout le potentiel en comparaison avec d'autres paradigmes plus connus comme par exemple les GAs et les Stratégies Evolutives (ES). Cette thèse vise à combler ce manque dans le cas de la mise en correspondance de graphes pour la reconnaissance d'objets dans les images. Un autre des aspects innovants de cette thèse est précisément le fait d'appliquer pour la première fois les EDAs aux problèmes de mise en correspondance de graphes, tout en tenant compte des deux domaines discret et continu.

Dans ce cas, les distributions estimées dans les EDAs représentent les dépendances entre les différentes correspondances possibles entre nœuds de G_D par rapport aux nœuds de G_M.

EDAs parallèles

Une des principales limites des EDAs pour les problèmes traités ici est le temps de calcul important qu'ils nécessitent pour aboutir à la solution. Nous avons donc étudié les possibilités de parallélisation de ces algorithmes afin de réduire ces temps d'exécution.

Nous avons analysé différents EDAs en fonction des besoins de calcul pour chacune de leurs étapes. Cette analyse a été réalisée avec l'outil GNU *gprog*. Ces études soulignent que les algorithmes les plus chers en calcul sont ceux du troisième groupe, et cet effet est précisément dû à l'étape d'apprentissage. Dans le cas de la reconnaissance de structures

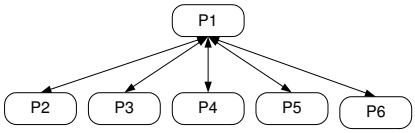


Figure 6: Schéma d'exécution maître-esclave, où un processus joue le rôle de *maître* de tâches et les autres réalisent des parties d'un travail commun.

cérébrales, cet apprentissage avec EBNA prend 85,7% du temps d'exécution total. Nous avons décidé de paralléliser l'algorithme EBNA (troisième groupe et domaine discret).

L'apprentissage dans EBNA repose sur la mesure BIC, qui demande presque tout le temps de calcul de cet apprentissage. La mesure $BIC(S, D)$, où S est la structure et D les données à partir desquelles est réalisé l'apprentissage, peut être divisée en composantes $BIC(i, S, D)$ qui expriment la mesure locale BIC pour une variable X_i :

$$BIC(S, D) = \sum_{i=1}^{n} BIC(i, S, D) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} \log \frac{N_{ijk}}{N_{ij}} - \frac{1}{2}(r_i - 1)q_i$$

Pour paralléliser le programme on utilise le modèle d'exécution maître-esclave illustré figure 6, dans lequel chacun des esclaves calcule les termes $BIC(i, S, D)$ correspondant aux différentes variables X_i $i = 1, \ldots, n$.

Pour essayer des techniques distinctes de parallélisme, cette thèse utilise des librairies pthreads et MPI qui permettent de tester l'intérêt d'utiliser la *mémoire partagée* et le *passage des messages* respectivement, avec des configurations différentes des ordinateurs.

Les essais réalisés montrent qu'avec des threads et avec du MPI nous pouvons obtenir des réductions significatives du temps de calcul, allant jusqu'à 60%.

Résultats expérimentaux

Nous avons traité différents exemples d'application des EDAs à des problèmes de correspondance inexacte de graphes. Des exemples synthétiques obtenus par simulations permettent d'illustrer différentes caractéristiques des EDAs et des techniques proposées :

- Comparaison entre EDAs, algorithmes génétiques et stratégies évolutives pour des problèmes de complexités très différentes.
- Analyse de quatre méthodes adaptées aux EDAs pour satisfaire les contraintes imposées aux problèmes de correspondance de graphes.
- Etude de l'évolution des structures graphiques probabilistes de génération en génération pour les EDAs discrets ou continus.
- Application et mesure de rendement des techniques de parallélisme proposées.

Des applications réelles ont ensuite été traitées, portant sur la reconnaissance de structures cérébrales à partir d'images 3D de résonance magnétique, et de structures faciales (il s'agit de trouver le nez, la bouche, etc.). Les résultats obtenus pour ce dernier problème pour les algorithmes ssGA (algorithme génétique), UMDA et EBNA sont illustrés figure 7.

Dans tous les exemples réels traités, les erreurs de reconnaissance que peuvent commettre les techniques de correspondance entre graphes se divisent en trois types, illustrés figure 8.

(a) ssGA (b) UMDA (c) EBNA

Figure 7: Exemple d'un problème réel de reconnaissance de structures faciales. Les résultats sont obtenus en utilisant l'algorithme génétique SSGA et les EDAs UMDA et EBNA. Ces résultats montrent que la reconnaissance est meilleure dans le cas des EDAs.

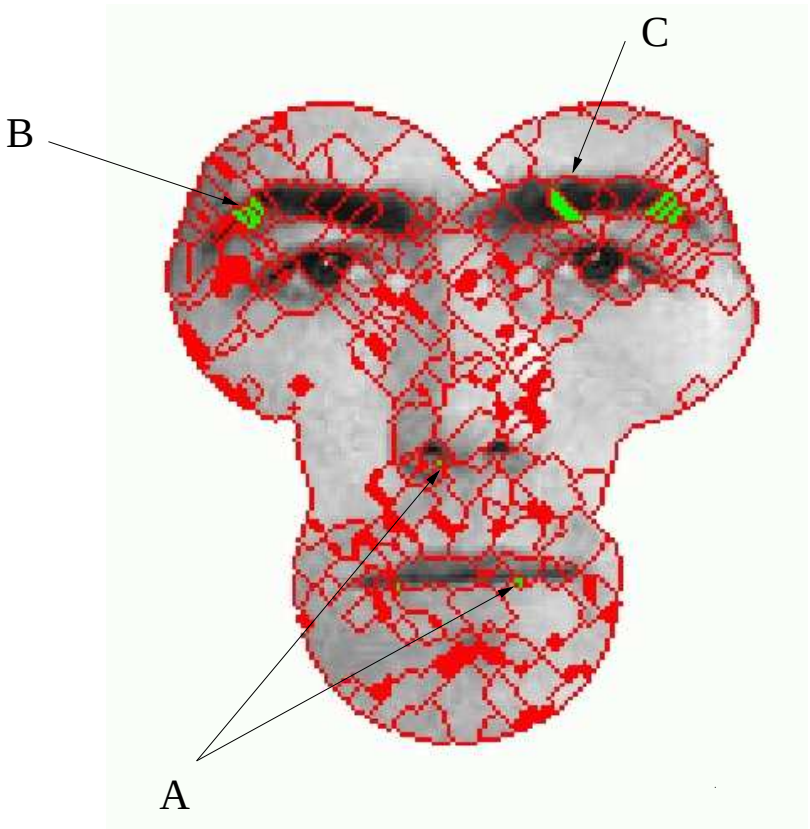


Figure 8: Exemple de quelques erreurs typiques de reconnaissance. A: segments de taille trop petite pour pouvoir être identifiés, même visuellement. B: segments situés à proximité des limites de deux régions ou plus, trop ambigüs pour pouvoir être classés. C: vraies erreurs de reconnaissance.

Conclusions et perspectives

Cette thèse formalise les problèmes de mise en correspondance de graphes comme des problèmes d'optimisation combinatoire sous contraintes. Un des apports les plus innovants de cette thèse consiste à utiliser pour la première fois les EDAs dans ce type de problèmes.

Quant à la comparaison des EDAs avec des GAs, la conclusion des essais réalisés est la suivante : pour des problèmes qui ne sont pas très complexes les GAs donnent des résultats similaires à ceux des EDAs et nécessitent moins de temps d'exécution ; cependant, pour des problèmes complexes les EDAs donnent toujours de meilleurs résultats que les GAs, qui sont plus susceptibles de tomber dans des extrema locaux. En outre, les EDAs du domaine continu se comportent généralement mieux que ceux du domaine discret quant à la qualité de la solution obtenue, bien qu'avec un coût plus grand en termes de temps d'exécution.

Différents types de fonctions objectif ont également été proposés, fondés sur des paradigmes différents comme la théorie des ensembles flous et la théorie des probabilités. Les résultats montrent aussi l'importance de choisir une bonne façon de générer le graphe modèle et de définir les attributs.

Finalement, les résultats expérimentaux montrent que la parallélisation des EDAs contribue de façon satisfaisante à la réduction de temps de calcul.

Les perspectives de ce travail sont les suivantes :

Modélisation : considérer les différences d'importance entre les régions, engendrer des modèles à partir de plus d'une image.

Fonctions objectif : réaliser une comparaison de rendement entre elles, tester d'autres représentations d'individus et de fonctions objectif.

Vérifier la validité de la méthode sur des séquences d'images et exploiter la continuité d'une image à la suivante pour améliorer la reconnaissance.

Amélioration des EDAs : autres techniques de génération de populations initiales, essais avec d'autres modèles graphiques probabilistes.

Parallélisation des EDAs : paralléliser d'autres EDAs, combinaison entre algorithmes en parallèle, appliquer d'autres techniques de parallélisme.

Publications des travaux réalisés pendant la thèse

Cette thèse a donné lieu à plusieurs publications dans des revues et congrès internationaux.

2002

- Bengoetxea, E., Larrañaga, P., Bloch, I., Perchant, A., and Boeres, C. (2002). Learning and simulation of Bayesian networks applied to inexact graph matching. Pattern Recognition 35(12):2867-2880.
- Cesar R., Bengoetxea E., Bloch I. Inexact graph matching using stochastic optimization techniques for facial feature recognition. In International Conference on Pattern Recognition (ICPR 2002). Quebeck, Canada.
- Cesar R., Bengoetxea E., Bloch I., Larrañaga, P. Inexact graph matching for facial feature recognition. International Journal of Imaging Systems and Technology (submitted).

- Mendiburu A., Bengoetxea, E., Miguel J. Paralelización de algoritmos de estimación de distribuciones. In XIII Jornadas de Paralelismo, pages 37-41. ISBN: 64-8409-159-7.

2001

- Bengoetxea, E., Larrañaga, P., Bloch, I., and Perchant, A. (2001). Estimation of Distribution Algorithms: A New Evolutionary Computation Approach For Graph Matching Problems. Lecture notes in Computer Science 2134. Third International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition (EMMCVPR-2001). M. Figueiredo, J. Zerubia and A. K. Jain (eds.), pages 454-468, Sophia Antipolis, France.

- Bengoetxea, E., Larrañaga, P., Bloch, I., and Perchant, A. (2001). Image Recognition with Graph Matching Using Estimation of Distribution Algorithms. Proceedings of the Medical Image Understanding and Analysis (MIUA 2001), 89-92, Birmingham, UK.

- Bengoetxea, E., Larrañaga, P., Bloch, I., Perchant, P. (2001). Solving graph matching with EDAs using a permutation-based representation. Estimation of Distribution Algorithms. A new tool for Evolutionary Computation, Larrañaga, P. and Lozano, J.A. (eds.). Kluwer Academic Publishers.

- Larrañaga, P., Bengoetxea,E., Lozano, J.A., Robles, V., Mendiburu, A., and de Miguel, P. (2001). Searching for the Best Permutation with estimation of Distribution Algorithms. Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 2001). Workshop on Stochastic Search Algorithms, pages 7-14, Seattle, USA.

- Bengoetxea, E., Miquélez, T., Lozano, J. A., and Larrañaga, P. (2001). Experimental results in function optimization with EDAs in continuous domain. Estimation of Distribution Algorithms. A new tool for Evolutionary Computation, Larrañaga, P. and Lozano, J.A. (eds.). Kluwer Academic Publishers.

- Larrañaga, P., Lozano, J. A., and Bengoetxea, E. (2001). Estimation of Distribution Algorithms based on multivariate normal and Gaussian networks. Technical Report KZZA-IK-1-01 of the Department of Computer Science and Artificial Intelligence, University of the Basque Country, Spain.

- Miquélez, T., Bengoetxea, E., Morlán, I., and Larrañaga, P. (2001). Obtention of filters for Image Restauration Using Estimation of Distribution Algorithms. CAEPIA 2001, Spanish Society for the Artificial Intelligence. (in Spanish).

2000

- Bengoetxea, E., Larrañaga, P., Bloch, I., Perchant, A., and Boeres, C. (2000). Inexact graph matching using learning and simulation of Bayesian networks. An empirical comparison between different approaches with synthetic data. Proceedings of CaNew workshop, ECAI 2000 Conference, ECCAI. Berlin, Germany.

Summary in Spanish – Resumen en castellano

Motivación

Desde hace ya varias décadas se han utilizado las computadoras para trabajar con imágenes. Los ejemplos más clásicos son aquellos en los que la computadora mejora una imagen con ruido o que está borrosa. Sin embargo, este tipo de trabajos utiliza información que contiene la imagen que se podría considerar de *bajo nivel*, es decir información tal que nivel de gris de los pixels, segmentación, e incluso formas geométricas de segmentos de una imagen. Sin embargo, también encontramos en las imágenes otro tipo de información que podríamos denominar de alto nivel, como son por ejemplo la estructura de la escena en la imagen, la topología, y organización espacial de los diferentes objetos en la imagen. Este último tipo de información la que resulta más difícil de procesar para una computadora, y es precisamente parte del objetivo de esta tesis: presentar un procedimiento novedoso para facilitar el reconocimiento automático de imágenes.

En los últimos años se ha propuesto la utilización de grafos para representar el conocimiento estructural utilizando técnicas matemáticas, y poder así utilizar esta representación para facilitar la comprensión a las computadoras. El interés de este tipo de representaciones va en aumento entre la comunidad científica debido a la posibilidad de utilizar esta representación junto con algoritmos de correspondencia de grafos[2], ya que esta representación es muy versátil y permite también representar las variaciones y diferencias estructurales entre diferentes objetos.

Asimismo se han propuesto diversas aplicaciones directas de este tipo de representación para realizar el reconocimiento automático de imágenes. En este tipo de aplicaciones, el conocimiento sobre posibles variaciones estructurales del objeto que aparece en las imágenes (una persona, un rostro, una imagen médica, etc.) es expresado por medio de un modelo que se representa como un grafo (o atlas, como también se denomina en ciertos casos). Ejemplos de áreas de reconocimiento de imágenes en las cuales este tipo de representaciones han sido utilizadas y publicadas son cartografía, reconocimiento de caracteres, y reconocimiento de estructuras cerebrales a partir de imágenes de resonancia magnética. Este último ejemplo se ilustra en la Figura 9.

El *grafo modelo* se construye habitualmente utilizando un vértice para expresar cada una de las regiones del objeto a reconocer (Ej.: en el caso de imágenes de un cerebro humano, habrá un vértice para representar el cerebelo, otro para el cuerpo calloso...) y aristas para representar la interrelación entre estas regiones. Asimismo, se utilizan atributos para expresar las propiedades de vértices y aristas para poder así identificarlos y distinguir los unos de los otros. Este grafo modelo contiene atributos en los vértices y aristas, y muchas veces es necesaria la supervisión de un experto (Ej. un neurólogo para construir el modelo que representa un cerebro humano) para asegurar la veracidad e idoneidad del modelo construido.

Tras la construcción de un grafo modelo, y para poder realizar el reconocimiento de imágenes a través del mismo, es necesario construir un grafo a partir de cada una de las imágenes a reconocer. Estos grafos que se denominan en la literatura *grafos de datos* o *grafos de entrada*, son construidos a menudo automáticamente por la computadora sin asistencia del usuario. En este proceso de construcción del grafo de datos correspondiente a una imagen, uno de los pasos más significativos en cuanto al rendimiento del método es el de la segmentación de la propia imagen previo a la construcción del propio grafo: el grafo de datos se genera utilizando un vértice para representar cada uno de los segmentos en los que

[2]La correspondencia de grafos se denomina en inglés *graph matching*

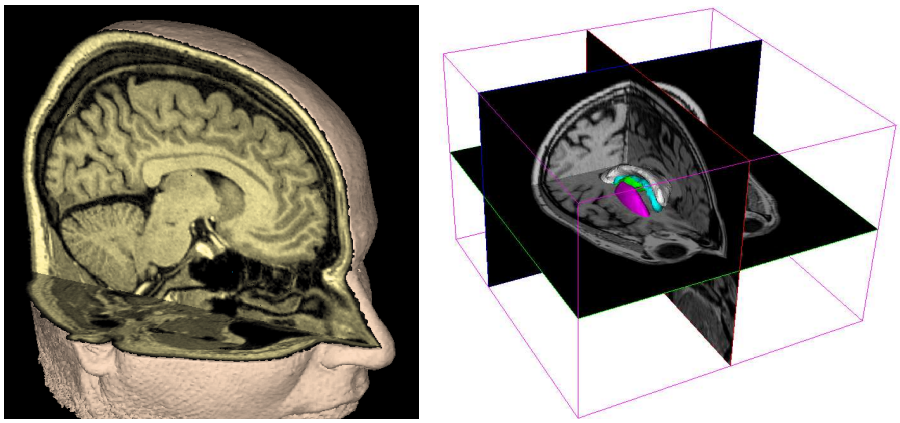


Figure 9: Ejemplo de aplicación de la correspondencia entre grafos al reconocimiento de imágenes. El objetivo en este problema en concreto es el de identificar las diferentes partes del cerebro a partir de la imagen en 3D de la izquierda.

se ha dividido la imagen (de forma similar al grafo modelo), y es precisamente debido a esto que la segmentación debe hacerse con especial atención para asegurar un número y tamaño adecuado de los segmentos.

Correspondencia de grafos

El reconocimiento de la imagen se realiza mediante la correspondencia entre los grafos modelo y de datos[3]. Este proceso es equivalente a asignar una etiqueta a cada una de las regiones de la imagen a reconocer, de manera que se designa cada región de la imagen como perteneciente a una sección concreta del objeto. Esta asignación de etiquetas se efectúa vértice a vértice de entre los del grafo de datos, asignando a cada uno un vértice del grafo modelo. Este proceso se ilustra en la Figura 10.

Formalizando lo anterior, se definen dos grafos en problemas de reconocimiento de patrones basados en modelos, –el grafo modelo G_M y el grafo de datos G_D– y el procedimiento de buscar correspondencias entre ellos se basa en analizar las semejanzas entre ellos según sus vértices, aristas y atributos. Por lo tanto, podemos plantear el problema de correspondencia entre grafos de la siguiente manera: dados dos grafos $G_M = (V_M, E_M)$ y $G_D = (V_D, E_D)$, el problema consiste en encontrar un homomorfismo $h : V_D \rightarrow V_M$ tal que $(u, v) \in E_D$ si y sólo si $(f(u), f(v)) \in E_M$.

Se han planteado varios problemas en la literatura para poder efectuar la correspondencia entre grafos y demostrar la validez de las distintas técnicas. En los problemas más simples se asume que el número de vértices en ambos grafos es el mismo, por lo que la técnica empleada en buscar la mejor correspondencia o solución debe devolver un isomorfismo. Este tipo de problemas se engloban dentro de *correspondencias exactas de grafos*, y la propiedad que se cumple en estos es $|V_M| = |V_D|$. Sin embargo, en muchos otros problemas (sobre todo en aquellos en los que la segmentación se ha realizado automáticamente) la condición de isomorfismo es demasiado estricta para poder satisfacerla y el número de vértices del grafo de datos es habitualmente superior al del modelo. En este otro tipo de problemas, cada posible solución es un homomorfismo h en el que se asigna a cada vértice del grafo de datos

[3] A este proceso de correspondencia entre grafos también se le denomina en castellano *macheo de grafos*.

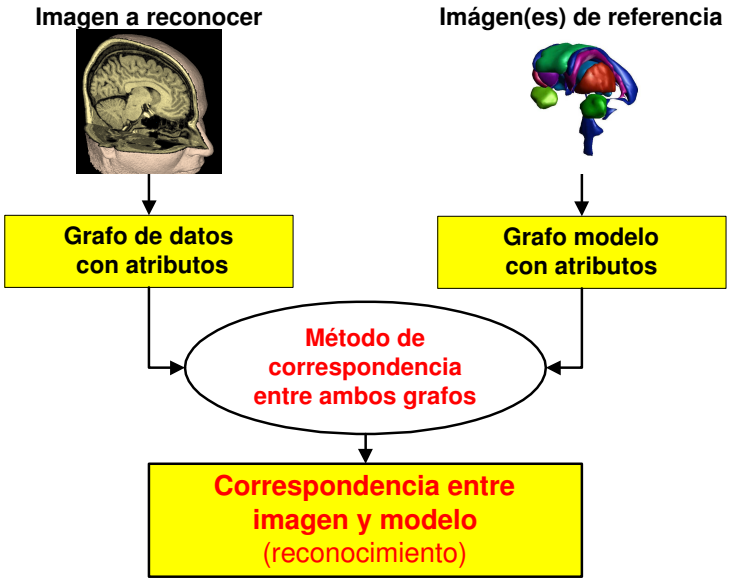


Figure 10: Ejemplo de cómo se realiza el reconocimiento de imágenes a partir de los grafos modelo y de datos.

una etiqueta correspondiente a un vértice. Estos últimos son conocidos como problemas de *correspondencia inexacta de grafos*, y su complejidad es mucho mayor que la de los anteriores. En este último caso se cumple la propiedad $|V_M| < |V_D|$.

Existen también otras posibles formas de clasificar los problemas de correspondencia de grafos. En cualquier caso, la clasificación basada en problemas de correspondencia de grafos exactos e inexactos es la más extendida. La Figura 11 muestra la clasificación más difundida en la literatura.

Se han propuesto muchos tipos diferentes de problemas de correspondencia de grafos en la literatura, y en muchos de ellos se utilizan los siguientes enfoques:

- Nodo nulo o *dummy*: son problemas en los que se utiliza un nodo adicional para tener una correspondencia añadida y poder dejar de considerar regiones en la imagen que no correspondan a partes del objeto a reconocer.

- Correspondencias múltiples entre nodos: este tipo de problemas son mucho más complejos de los comentados hasta ahora ya que es posible que un segmento de la imagen a reconocer sea a la vez parte de más de una región del objeto a analizar. Aunque este tipo de problemas pueden plantearse, su resolución conlleva más combinaciones y el espacio de búsqueda para encontrar la solución óptima puede llegar a ser intratable computacionalmente.

Se pueden aplicar muchas técnicas diferentes para encontrar la correspondencia óptima en un problema concreto, incluso procedentes de paradigmas muy diversos. Ejemplos de artículos mostrando métodos aplicados a correspondencia entre grafos son los que proponen árboles de decisión [Messmer and Bunke, 1999], redes neuronales [Riviere et al., 2002], algoritmo EM [Cross and Hancock, 1998], relajación probabilística [Christmas et al., 1995], heurísticos y metaheurísticos [Pelillo et al., 1999], algoritmos genéticos [Wilson and Hancock, 1997], y programación evolutiva [Singh and Chaudhury, 1997]. Muchas de estas técnicas están

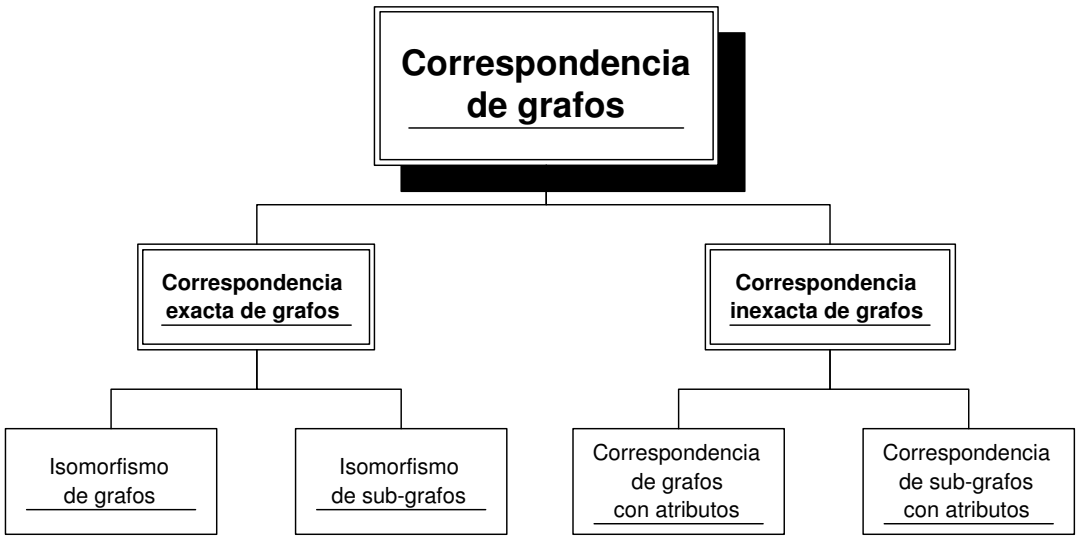


Figure 11: Clasificación de problemas de correspondencias entre grafos en dos clases principales.

orientadas a correspondencias exactas de grafos y por lo tanto devuelven el isomorfismo que representa la mejor correspondencia entre el grafo modelo y el de datos generado a partir de la imagen original a ser reconocida. Sin embargo, en muchos otros (sobre todo en aquellos aplicados a problemas con imágenes reales) la condición de isomorfismo es demasiado fuerte y por lo tanto se pueden encontrar en la literatura numerosas aproximaciones para resolver problemas de correspondencias inexactas de grafos.

Los problemas que se han considerado en esta tesis doctoral son todos problemas de correspondencia inexacta de grafos, y en ningún caso se ha considerado la posibilidad de añadir nodos nulos. Además, con el objetivo de no aumentar la complejidad de problemas que ya de por sí son complejos, se ha decidido realizar una sobre-segmentación de la imagen a reconocer para evitar la posibilidad de correspondencias múltiples entre grafos. De esta forma el número de vértices de los grafos de nodos es mayor en nuestros problemas. El método que proponemos para realizar la correspondencia entre grafos es el de Algoritmos de Estimación de Distribuciones (EDAs), tema que se estudia en profundidad en esta tesis. Así, se puede resumir el objetivo y la motivación de esta tesis como la presentación de un nuevo paradigma para hacer frente a correspondencias inexactas de grafos aplicados al dominio de la visión por computador y reconocimiento de patrones.

Correspondencia de grafos planteado como un problema de optimización combinatorial

El problema de correspondencia inexacta de grafos es NP-duro, tal y como demuestran varios trabajos en la literatura. Debido a esta complejidad tan grande, el uso de algoritmos que proporcionan una aproximación a la mejor solución se muestra necesario. Así, con el objetivo de poder aplicar este tipo de algoritmos, se puede plantear el problema de correspondencias entre grafos como un problema de optimización combinatorial con restricciones. Esta tesis analiza los aspectos y mecanismos que deben aplicarse para poder plantear el problema de esta manera y poder aplicar posteriormente algoritmos como los denominados Algoritmos de Estimación de Distribuciones (EDAs).

Para poder plantear un problema cualquiera como un utilizando técnicas este tipo de algoritmos se necesita esencialmente definir las siguientes características:

- **Una representación de los individuos** o soluciones, de manera que podamos expresar cada uno de los puntos en el espacio de búsqueda.
- **Una función objetivo a optimizar**, que asigna un valor a cada solución posible (o individuo) para expresar cómo es de adecuado el individuo analizado como solución para el problema.

Los individuos se expresan como vectores de valores que pueden ser discretos o continuos. Algunos algoritmos están diseñados para trabajar únicamente con individuos discretos o continuos, aunque otros como por ejemplo los EDAs permiten utilizar ambos tipos de individuos. Esta tesis presenta la forma de trabajar con ambos tipos de individuos y diferentes representaciones dentro de los dominios discretos y continuos para aplicarlas en problemas de correspondencia de grafos.

Un ejemplo de una de las representaciones que se ha aplicado en varios problemas expuestos en esta tesis consiste en asociar a cada vértice de G_D un vértice de G_M. Por lo tanto, el tamaño de los individuos en este caso concreto es de $n = |V_D|$ variables[4], $X_i \in \boldsymbol{X}$ $i = 1, \ldots, |V_D|$, donde cada variable contiene un valor entre 1 y $|V_M|$. El valor de cada variable en el individuo tiene el siguiente significado: $X_i = k \quad 1 \leq i \leq |V_D|, \quad 1 \leq k \leq |V_M| \Leftrightarrow$ el i-ésimo vértice de G_D es identificado como el k-ésimo vértice de G_M.

Esta tesis propone otras dos representaciones más en el dominio discreto, así como una representación de individuos para el dominio continuo. Además, y con la intención de mostrar la robustez del método propuesto, se ha añadido una restricción a los problemas que deben de cumplir los individuos para poder tenerse en consideración. Esta restricción es la siguiente:

$$\forall a_M \in V_M \; \exists a_D \in V_D \;\; | \;\; h(a_D) = a_M$$

De esta forma, se espera que la solución devuelta por el método haya encontrado cada una de las partes representadas en el modelo al menos a uno de los segmentos de la imagen a reconocer. Esta restricción aumenta así la complejidad del problema, y esta tesis presenta varios mecanismos que pueden utilizarse para tener en cuenta restricciones mediante la aplicación de EDAs.

Finalmente, y con el objetivo de evaluar la bondad del individuo como posible solución a un problema, se definen dos funciones $c_N(a_D, a_M)$ y $c_E(e_D, e_M)$ que miden respectivamente la semejanza entre dos vértices $a_D \in V_D$ y $a_M \in V_M$, y entre dos aristas $e_D \in E_D$ y $e_M \in E_M$. Se definen en esta tesis un total de cinco funciones objetivo basadas en estas dos medidas de semejanza para poder medir la conveniencia de la solución que representa cada individuo para un problema determinado.

Algoritmos de Estimación de Distribuciones

Los Algoritmos de Estimación de Distribuciones (o EDAs como se les conoce en la literatura) es un tópico reciente dentro de la familia de la computación evolutiva aplicada a problemas de optimización. Otros ejemplos dentro del dominio de la computación evolutiva son los Algoritmos Genéticos (GAs).

[4]Cuando se aplican algoritmos genéticos se dice que los individuos están formados por *genes*, mientras que en otros paradigmas como en el caso de los EDAs se dice que tienen *variables*. En esta tesis se considera que los individuos constan de variables.

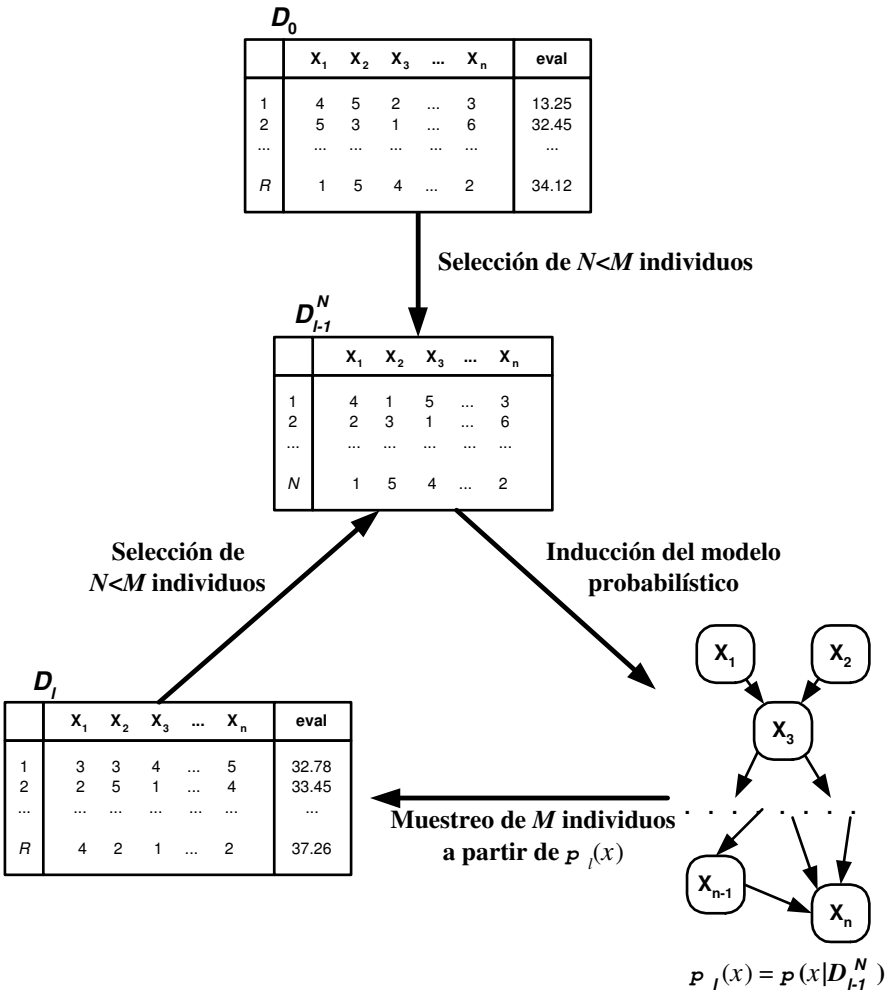


Figure 12: Ilustración de la aproximación EDA en procesos de optimización.

La computación evolutiva se engloba dentro del tipo de técnicas no deterministas, y los algoritmos de computación evolutiva se caracterizan por mantener un conjunto de posibles soluciones que se hace evolucionar progresivamente. Debido al símil con poblaciones de seres vivos, se denomina habitualmente *individuo* a cada una de las soluciones, *población* al conjunto de individuos, y *generación* a cada una de las poblaciones que evolucionan sucesivamente una tras otra. La diferencia más significativa entre los GAs (que son sin duda los más conocidos y utilizados dentro de la computación evolutiva) y los EDAs es que la evolución desde una generación a la siguiente se realiza en el caso de los GAs mediante operaciones de mutación y cruce, mientras que en los EDAs esta evolución se realiza utilizando técnicas basadas en la teoría de la probabilidad y más concretamente mediante el aprendizaje y la simulación de redes Bayesianas o redes Gausianas. Esta idea se ilustra en la Figura 12.

La Figura 13 muestra el pseudocódigo genérico de los EDAs, que sigue esencialmente los siguientes pasos:

1. Primeramente, se genera la población inicial D_0 formada por R individuos. La creación de estos R individuos se realiza a menudo asumiendo una distribución uniforme en cada variable. Tras generar los individuos, estos se evalúan mediante la aplicación de

EDA

$D_0 \leftarrow$ Generar R individuos (la población inicial D_0) al azar

Repetir para $l = 1, 2, \ldots$ hasta satisfacer un criterio de parada

$D_{l-1}^N \leftarrow$ Seleccionar $N < R$ individuos de D_{l-1} siguiendo un método de selección determinado

$\rho_l(\boldsymbol{x}) = \rho(\boldsymbol{x}|D_{l-1}^N) \leftarrow$ Estimar la distribución de probabilidad de que un individuo se encuentre entre los individuos seleccionados

$D_l \leftarrow$ Muestrear R individuos (la nueva población) a partir de $\rho_l(\boldsymbol{x})$

Figure 13: Pseudocódigo genérico de los EDA.

la función objetivo.

2. Segundo, para evolucionar la $l-1$-ésima población D_{l-1} hacia la siguiente D_l, se seleccionan N individuos ($N < R$) de D_{l-1} siguiendo un criterio. Denominamos D_{l-1}^N al conjunto de los N individuos seleccionados de la generación número $l-1$.

3. Tercero, se induce el modelo gráfico probabilístico n–dimensional que mejor representa las interdependencias entre las n variables. Este paso es conocido como el del *aprendizaje*, y es el más crucial de los EDA debido a la importancia de tener en cuenta todas las dependencias entre variables para asegurar una evolución satisfactoria hacia individuos más válidos.

4. Finalmente, la nueva población D_l se constituye con R nuevos individuos obtenidos tras *simular* la distribución de probabilidad aprendida en el paso previo. A menudo se utiliza una aproximación elitista, manteniendo así el mejor individuo de la población D_{l-1}^N e la nueva población D_l. En este último supuesto, se crean cada generación un total de $R-1$ nuevos individuos en vez de R.

Los pasos 2, 3 y 4 se repiten hasta satisfacer una condición de parada concreto. Ejemplos de criterios de parada son: llegar a un número de generación máxima, alcanzar un número máximo de individuos analizados, uniformidad en la población recién generada, o el hecho de no obtener un individuo con un valor de función objetivo mejor tras un cierto número de generaciones.

Se han propuesto una gran variedad de algoritmos en la literatura que son parte de los EDA, los cuales pueden clasificarse en tres grandes grupos dependiendo de la complejidad del tipo de dependencias entre variables que tienen en cuenta:

- **Sin interdependencias entre variables:** estos EDAs se basan únicamente en distribuciones univariantes $\rho(x_i)$. Esto significa que la estructura en forma de red Bayesiana (o Gausiana si trabajamos en el dominio continuo) es fija y no contiene arcos. En otras palabras, esto significa que todas las variables del individuo se consideran independientes entre sí.

  Como ejemplo de algoritmos pertenecientes a este grupo tenemos en el dominio discreto el llamado UMDA (Univariate Marginal Distribution Algorithm) [Mühlenbein, 1998]

donde la estimación de la distribución de probabilidad se realiza de la siguiente manera:

$$p(\boldsymbol{x}) = \prod_{i=1}^{n} p(x_i)$$

Otro ejemplo es el conocido como UMDA$_c$ (Univariate Marginal Distribution Algorithm - continuous)[Larrañaga et al., 2000], el cual es el equivalente a UMDA aunque en este caso corresponde al dominio continuo.

- **Dependencias a pares entre variables:** los EDAs pertenecientes a este grupo están basados en distribuciones univariantes –$\rho(x_i)$– y también en condicionales de segundo orden –$\rho(x_j \mid x_i)$. La diferencia más significativa con respecto al grupo anterior es que la estructura de la red Bayesiana (o Gausiana) puede ser diferente, aunque cada una de las variables puede tener como mucho un padre. Esto requiere un paso previo de selección de la mejor estructura que no existía en los anteriores.

  Un ejemplo de EDAs discretos pertenecientes a este grupo es MIMIC (Mutual Information Maximization for Input Clustering) [de Bonet et al., 1997], que propone realizar la siguiente factorización de la probabilidad:

  $$p(\boldsymbol{x}) = p(x_{i_1} \mid x_{i_2}) \cdot p(x_{i_2} \mid x_{i_3}) \cdots p(x_{i_{n-1}} \mid x_{i_n}) \cdot p(x_{i_n})$$

  MIMIC se basa en buscar la permutación $\pi = (i_1, i_2, \ldots, i_n)$ que minimiza la divergencia de Kullback-Leibler entre la estimación $\hat{p}_\pi(\boldsymbol{x})$ y la distribución real $p(\boldsymbol{x})$.

  De nuevo, existe una versión continua de MIMIC llamada MIMIC$_c$ (Mutual Information Maximization for Input Clustering - continuous) [Larrañaga et al., 2000].

- **Dependencias múltiples entre variables:** los EDAs pertenecientes a este grupo consideran tanto distribuciones univariantes como condicionales orden dos o superior, y por lo tanto las estructuras de redes Bayesianas o Gausianas no tiene ninguna restricción en el número de arcos que pueden contener. Esta característica requiere una búsqueda exhaustiva de la mejor estructura gráfica probabilística entre todas las posibles, y por lo tanto estos algoritmos son más costosos en tiempo de ejecución que los de los grupos anteriores, aunque también son capaces de aprender modelos que reflejan más fielmente las interrelaciones entre las diferentes variables que forman parte de los individuos.

  En el dominio discreto, como un ejemplo de EDAs pertenecientes a este grupo tenemos el conocido como EBNA (Estimation of Bayesian Networks Algorithm) [Etxeberria and Larrañaga, 1999]. Un ejemplo del dominio continuo es EGNA (Estimation of Gaussian Networks Algorithm) [Larrañaga et al., 2000, Larrañaga and Lozano, 2001], que sigue una aproximación similar a EBNA.

  En EBNA se define un *score* o medida basada en la máxima verosimilitud penalizada conocida como BIC (Bayesian Information Criterion) [Schwarz, 1978] que mide la idoneidad de una estructura para representar las interdependencias entre los individuos. Utilizando esta medida, se busca la red Bayesiana que lo maximiza, y para ello los autores proponen el método conocido como Algoritmo B [Buntine, 1991]. Una vez definida la estructura, la factorización de la probabilidad se realiza de la siguiente forma:

  $$p(\boldsymbol{x}) = \prod_{i=1}^{n} p(x_i | \boldsymbol{pa}(x_i))$$

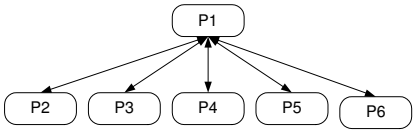


Figure 14: Esquema de ejecución maestro-esclavo, donde un proceso hace de *gestor* de tareas y los demás realizan partes de un trabajo común.

donde $\boldsymbol{pa}(x_i)$ es el conjunto de padres de la variable x_i en la red Bayesiana.

Son varios los algoritmos y aproximaciones de EDAs que se han propuesto en la literatura, aunque por el momento no hay demasiados artículos demostrando todo su potencial en comparación con otros paradigmas más conocidos como por ejemplo los GAs y Estrategias evolutivas (ES). Esta tesis viene a cubrir este espacio para la aplicación concreta de reconocimiento de objetos en imágenes. Otro de los aspectos novedosos de esta tesis es precisamente el hecho de aplicar EDAs por primera vez a problemas de correspondencia de grafos, y esta tarea se realiza para dominios discretos y continuos. Asimismo se han desarrollado en esta tesis métodos orientados a los EDAs para gestionar las restricciones que puedan existir en problemas, aplicables incluso en aquellos problemas que no sean de correspondencia de grafos.

En el caso de aplicar EDAs a correspondencia de grafos, las distribuciones y estructuras probabilísticas que se estiman en los EDA representan las dependencias entre las diferentes posibilidades de correspondencias entre vértices de G_D con respecto a vértices de G_M.

Uno de los mayores inconvenientes que se han encontrado al aplicar EDAs a correspondencias entre grafos es el hecho de que, debido al gran tamaño de los vértices y de los atributos a tener en cuenta en problemas reales, se requiere mucho tiempo de cálculo para que los EDAs evolucionen. Este inconveniente es especialmente evidente en los EDAs del tercer grupo dado que para la búsqueda de la mejor estructura gráfica probabilística se requiere analizar muchas posibles estructuras. Debido a esto, esta tesis propone técnicas de paralelismo para EDAs orientadas a reducir estos tiempos de ejecución.

Paralelización de EDAs

Esta tesis presenta un estudio que analiza las necesidades de CPU de los diferentes EDAs. Este análisis se ha realizado con la herramienta GNU *gprog*. En estos estudios se evidencia que los algoritmos más costosos computacionalmente son los del tercer grupo, y es precisamente el paso del aprendizaje el que mayor peso conlleva. En el caso concreto del problema de reconocimiento de estructuras cerebrales, el aprendizaje con EBNA supone el 85,7% del tiempo de ejecución total. Se decidió paralelizar el algoritmo EBNA debido a que los dominios discretos están más extendidos entre los EDAs y que los EBNA pertenecen al tercer grupo de los EDAs.

El aprendizaje en EBNA se basa en la medida BIC, y es precisamente esta medida la que requiere casi todo el tiempo de cálculo del aprendizaje. Una importante propiedad de BIC –$BIC(S, D)$ donde S es la estructura y D los datos a partir de los cuales se ha realizado el aprendizaje– es que se puede descomponer en componentes $BIC(i, S, D)$ que expresan la

(a) ssGA (b) UMDA (c) EBNA

Figure 15: Ejemplo de un problema real de reconocimiento de estructuras faciales utilizando correspondencia entre grafos. Se ilustran los resultados conseguidos utilizando algoritmos genéticos y los EDAs UMDA y EBNA. Estos resultados muestran que el reconocimiento en este caso es superior en el caso de ambos EDAs.

medida local BIC para la variable X_i:

$$BIC(S,D) = \sum_{i=1}^{n} BIC(i,S,D) = \sum_{i=1}^{n}\sum_{j=1}^{q_i}\sum_{k=1}^{r_i} N_{ijk}\log\frac{N_{ijk}}{N_{ij}} - \frac{1}{2}(r_i-1)q_i$$

Para paralelizar el programa se utilizó el modelo de ejecución maestro-esclavo ilustrado en la Figura 14, de manera que cada uno de los esclavos computan los términos $BIC(i,S,D)$ correspondientes a las diferentes variables X_i $i = 1,\ldots,n$.

Para probar distintas técnicas de paralelismo se utilizaron las librerías pthreads y MPI, las cuales permitieron comparar la posibilidad de utilizar memoria compartida y paso de mensajes respectivamente con diferentes configuraciones de máquinas.

Los experimentos realizados mostraron que tanto con threads como con MPI se obtienen reducciones sustanciales en el tiempo de cálculo, llegando en algunas ocasiones a reducir el tiempo de cálculo inicial en un 60%.

Ejemplos experimentales

Esta tesis muestra diferentes ejemplos de aplicación de EDAs a problemas de correspondencia inexacta de grafos. De entre los ejemplos se presentan tanto problemas creados artificialmente para mostrar varias características de los EDAs y técnicas propuestas, así como problemas reales. Los estudios que se han realizado con ejemplos artificiales son los siguientes:

- Comparativa entre EDAs, GAs y ES para problemas de complejidad muy diferente.
- Análisis de cuatro métodos adaptados a los EDAs para controlar el cumplimiento de restricciones en problemas de correspondencia de grafos.
- Estudio de la evolución de las estructuras gráficas probabilísticas de generación en generación durante una búsqueda con EDAs discretos o continuos.
- Aplicación y medición de rendimiento de las técnicas de paralelismo propuestas.

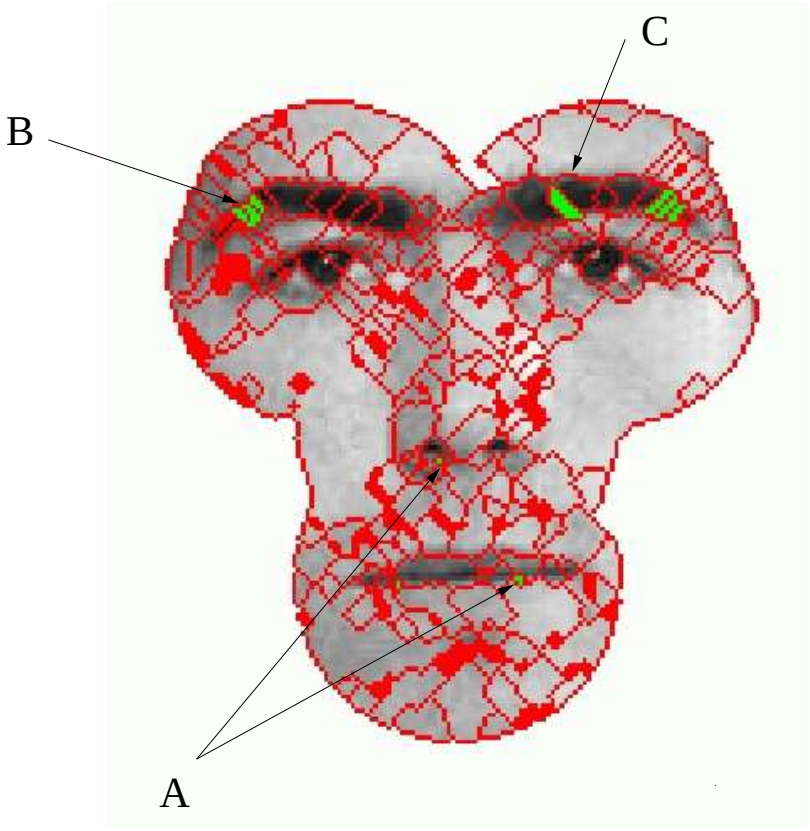


Figure 16: Ejemplo de algunos errores típicos en regiones. A: segmentos muy pequeños, de tamaño tan pequeño que es difícil incluso para una persona identificarlas propiamente. B: segmentos situados en cercanía de límites entre dos o más regiones, tan cercanos que crean dudas para poder clasificarse adecuadamente. C: verdaderos errores de reconocimiento.

Además de todos estos estudios, se muestran en esta tesis aplicaciones reales, orientadas sobre todo a reconocimiento de estructuras cerebrales a partir de imágenes en 3D de resonancia magnética, y de reconocimiento de estructuras faciales (se trata de encontrar la nariz, boca, etc.). Los resultados conseguidos para este último problema para los algoritmos con *steady state* (un algoritmo genético), UMDA y EBNA se muestran en la Figura 15.

En todos los ejemplos reales que se han utilizado en esta tesis se ha comprobado que los posibles errores de reconocimiento al utilizar técnicas de correspondencia entre grafos se dividen en tres tipos. Estos tres tipos se ilustran en la Figura 16.

Conclusiones y perspectivas

Esta tesis plantea los problemas de correspondencia entre grafos como problemas de optimización combinatorial con restricciones. Una de las aportaciones más novedosas de esta tesis consiste en utilizar EDAs por primera vez en este tipo de problemas.

De la comparación de EDAs con GAs la conclusión de los experimentos realizados es la siguiente: en problemas no muy complejos los GAs encuentran resultados similares a los de los EDA requiriendo menos tiempo de ejecución; sin embargo, en problemas complejos los EDAs siempre consiguen mejores resultados que los GAs, siendo estos últimos además muy

susceptibles a caer en máximos locales. Además, los EDA continuos muestran generalmente un mejor rendimiento que los discretos a la hora de devolver una solución, aunque a costa de un mayor tiempo de ejecución.

Se proponen asimismo varios tipos diferentes de funciones objetivo basados en diferentes paradigmas como por ejemplo lógica difusa y teoría de la probabilidad. Los resultados muestran también la importancia de la generación del grafo modelo y la definición de atributos.

Finalmente, los resultados experimentales muestran que la paralelización de los EDA contribuye satisfactoriamente a la reducción de tiempos de cálculo.

Referente a líneas de trabajo futura, se pueden mencionar las siguientes áreas y posibles ideas:

* **Modelización:** considerar importancias diferentes entre diferentes regiones, generar modelos a partir de más de una imagen...

* **Funciones objetivo:** realizar una comparación de rendimiento entre ellas, pruebas con otras representaciones de individuos y funciones objetivo...

* **Comprobar la validez del método en secuencias de imágenes**

* **Mejoras en los EDA:** otras técnicas de generación de poblaciones iniciales, pruebas con otros modelos gráficos probabilísticos...

* **Paralelización de EDAs:** paralelizar otros EDAs, combinación entre algoritmos en paralelo, aplicar otras técnicas de paralelismo...

Publicaciones de esta tesis doctoral

Durante el desarrollo de esta tesis doctoral se han publicado varios trabajos en revistas, libros y congresos, sobre todo a nivel internacional. La lista completa de publicaciones es la siguiente:

2002

- Bengoetxea, E., Larrañaga, P., Bloch, I., Perchant, A., and Boeres, C. (2002). Learning and simulation of Bayesian networks applied to inexact graph matching. Pattern Recognition 35(12):2867-2880.

- Cesar R., Bengoetxea E., Bloch I. Inexact graph matching using stochastic optimization techniques for facial feature recognition. In International Conference on Pattern Recognition (ICPR 2002). Quebeck, Canada.

- Cesar R., Bengoetxea E., Bloch I., Larrañaga, P. Inexact graph matching for facial feature recognition. International Journal of Imaging Systems and Technology (submitted).

- Mendiburu A., Bengoetxea, E., Miguel J. Paralelización de algoritmos de estimación de distribuciones. In XIII Jornadas de Paralelismo, pages 37-41. ISBN: 64-8409-159-7.

2001

- Bengoetxea, E., Larrañaga, P., Bloch, I., and Perchant, A. (2001). Estimation of Distribution Algorithms: A New Evolutionary Computation Approach For Graph Matching Problems. Lecture notes in Computer Science 2134. Third International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition (EMMCVPR-2001). M. Figueiredo, J. Zerubia and A. K. Jain (eds.), pages 454-468, Sophia Antipolis, France.

- Bengoetxea, E., Larrañaga, P., Bloch, I., and Perchant, A. (2001). Image Recognition with Graph Matching Using Estimation of Distribution Algorithms. Proceedings of the Medical Image Understanding and Analysis (MIUA 2001), 89-92, Birmingham, UK.

- Bengoetxea, E., Larrañaga, P., Bloch, I., Perchant, P. (2001). Solving graph matching with EDAs using a permutation-based representation. Estimation of Distribution Algorithms. A new tool for Evolutionary Computation, Larrañaga, P. and Lozano, J.A. (eds.). Kluwer Academic Publishers.

- Larrañaga, P., Bengoetxea,E., Lozano, J.A., Robles, V., Mendiburu, A., and de Miguel, P. (2001). Searching for the Best Permutation with estimation of Distribution Algorithms. Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 2001). Workshop on Stochastic Search Algorithms, pages 7-14, Seattle, USA.

- Bengoetxea, E., Miquélez, T., Lozano, J. A., and Larrañaga, P. (2001). Experimental results in function optimization with EDAs in continuous domain. Estimation of Distribution Algorithms. A new tool for Evolutionary Computation, Larrañaga, P. and Lozano, J.A. (eds.). Kluwer Academic Publishers.

- Larrañaga, P., Lozano, J. A., and Bengoetxea, E. (2001). Estimation of Distribution Algorithms based on multivariate normal and Gaussian networks. Technical Report KZZA-IK-1-01 of the Department of Computer Science and Artificial Intelligence, University of the Basque Country, Spain.

- Miquélez, T., Bengoetxea, E., Morlán, I., and Larrañaga, P. (2001). Obtention of filters for Image Restauration Using Estimation of Distribution Algorithms. CAEPIA 2001, Spanish Society for the Artificial Intelligence. (in Spanish).

2000

- Bengoetxea, E., Larrañaga, P., Bloch, I., Perchant, A., and Boeres, C. (2000). Inexact graph matching using learning and simulation of Bayesian networks. An empirical comparison between different approaches with synthetic data. Proceedings of CaNew workshop, ECAI 2000 Conference, ECCAI. Berlin, Germany.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

'A journey of a thousand miles begins with a single step.'

Confucius

The automatic recognition of images is an objective which is difficult to achieve. This issue has been analyzed and tackled in many different ways. Moreover, the automatic recognition of images is regarded as a complex field within the pattern recognition domain, as many questions such as the way of representing the knowledge and the way of adapting the knowledge representation to the changes in the images are essential for successful result and have to be analyzed with care.

Graph representations are widely used for dealing with structural information in different domains such as networks, psycho-sociology, image interpretation, pattern recognition, and many others. In a typical graph representation, regions of the image are represented by vertices in the graph. These vertices are related to each other by edges, which express structural relationships between objects. Vertices and edges are usually attributed.

One important problem to be solved when using such representations is graph matching, for instance to recognize image regions with the help of a model. The problem consists in searching for the best homomorphism between two graphs: the one that represents the model –the model graph[1]– and the other that represents the image –the data graph. The best graph homomorphism is determined regarding both the attributes of vertices and edges.

In order to achieve a good correspondence between the model and the data graphs, the most usual way is to search for a graph isomorphism. A lot of work is dedicated to the search for an exact isomorphism between two graphs or subgraphs. However, in many cases the bijective condition is too strong. Because of the schematic aspect of the model and of the difficulty to segment accurately the image into meaningful entities, no isomorphism can usually be expected between both graphs. Such problems call for inexact graph matching.

This type of problem occurs in different model-based image recognition domains. For instance, in the case of the recognition of brain images, the model (i.e. the anatomic atlas) is usually build as a representation of the regions that appear in a healthy human brain, and the data images are obtained using Magnetic Resonance images (MRi) of patients. In other application types such as cartography, the model is constructed from maps of the region of study, while the data images are obtained from aerial or satellite photographs. A last application to which this type of abstraction is applied is the recognition of facial features.

[1]The model is sometimes also called *atlas* or *map* depending on the type of problem. Some authors in the literature also use the term *pattern graph* to refer to the model graph.

As the number of features in the image increases, the size of graphs increases too, and the matching process becomes more complex. This has been proved to be NP-hard. As a result, many different techniques have been applied to perform the matching between graphs, and among them we can find the use of relaxation techniques, the EM algorithm, combinatorial optimization techniques, etc.

This thesis proposes to solve this optimization problem using a class of evolutionary computation techniques called Estimation of Distribution Algorithms (EDAs), which are based on learning and simulation of probabilistic graphical models (i.e. Bayesian or Gaussian networks). The main drawback of other evolutionary computation techniques, such as Genetic Algorithms (GAs) in the discrete domain and Evolutionary Strategies (ESs) in the continuous domain, is that their behavior depends to a large extent on tuning appropriately associated parameters, and for this purpose the researcher requires experience in the resolution and use of these algorithms. EDAs do not have that many parameters and can be applied easily to complex problems. Furthermore, EDAs have already shown a better performance than GAs in many problems, specially in complex ones, but their use in graph matching has not been proposed previously. In addition, this thesis proposes the definition of a new type of attributes for creating the model and data graphs as well as related fitness functions, which are based on probability theory. Moreover, the parallelization of EDAs in the discrete domain is designed and tested in order to reduce the CPU cost of the most promising methods. Finally, two real problems are described and tested: the recognition of human brain structures and of facial features from images. The latter application has been developed in collaboration with Roberto Cesar from the University of São Paulo in Brazil.

This thesis is organized as follows. This introductory chapter summarizes the context, the motivation, and the main contributions of this PhD thesis. Chapter 2 explains the graph matching problem and its different types as found in the literature. Chapter 3 proposes to solve these problems as combinatorial optimization ones with constraints, and analyzes the modifications and adaptations that are required in any evolutionary computation technique when applied to graph matching problems. Chapter 4 explains the theoretical background behind the evolutionary computation approach of estimation of distribution algorithms for both the discrete and continuous domains. Chapter 5 reviews all the aspects that need to be considered for parallelizing EDAs in order to reduce the computation time that these require, and constitutes an overview of the main aspects that need to be taken into account for applying up-to-date parallelization techniques. Chapters 6 and 7 describe the experiments carried out with synthetic data and real problems respectively, and the performance and behavior of the different algorithms and the results obtained are presented. Finally, Chapter 8 presents the conclusions of this PhD thesis.

Chapter 2

The graph matching problem

'Imagination is more important than knowledge. Knowledge is limited. Imagination encircles the world.'

Albert Einstein

This chapter explains the graph matching problem in detail. We first introduce some notation and terminology. Next, a classification of the different graph matching types is presented: this PhD thesis concentrates on inexact graph matching problems, but this chapter summarizes other types of graph matching too. The complexity of the different graph matching problems is also analyzed. Finally, the state of the art is presented. The amount of this type of work shows that the interest on the field is increasing with the years.

2.1 Basic notation and terminology

A graph $G = (V, E)$ in its basic form is composed of vertices and edges. V is the set of vertices (also called *nodes* or *points*) and $E \subset V \times V$ (also defined as $E \subset [V]^2$ in the literature) is the set of edges (also known as *arcs* or *lines*) of graph G. The difference between a graph G and its set of vertices V is not always made strictly, and commonly a vertex u is said to be in G when it should be said to be in V.

The *order* (or *size*) of a graph G is defined as the number of vertices of G and it is represented as $|V|$ and the number of edges as $|E|$[1].

If two vertices in G, say $u, v \in V$, are connected by an edge $e \in E$, this is denoted by $e = (u, v)$ and the two vertices are said to be *adjacent* or *neighbors*. Edges are said to be undirected when they have no direction, and a graph G containing only such types of graphs is called *undirected*. When all edges have directions and therefore (u, v) and (v, u) can be distinguished, the graph is said to be *directed*. Usually, the term *arc* is used when the graph is directed, and the term *edge* is used when it is undirected. In this dissertation we will mainly use directed graphs, but graph matching can also be applied to undirected ones[2]. In addition, a directed graph $G = (V, E)$ is called *complete* when there is always an edge $(u, u') \in E = V \times V$ between any two vertices u, u' in the graph.

[1] In some references in the literature the number of vertices and edges are also represented by $|G|$ and $||G||$ respectively.

[2] In this thesis we will use the terms *vertex* and *edge* for graphs representing knowledge. Later in Chapter 4 probabilistic graphical models such as Bayesian and Gaussian networks are introduced, and the terms *node* and *arc* will be used for these in order to distinguish them.

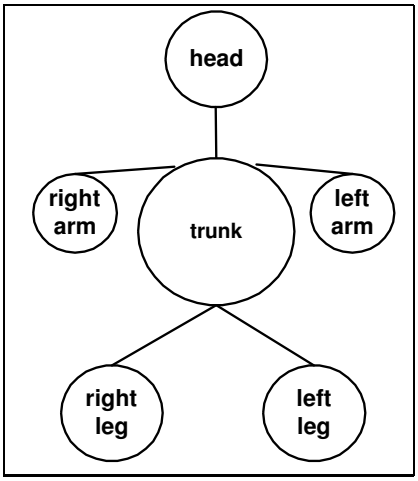


Figure 2.1: Illustration of how the physical parts of a human body can be represented using a graph.

Graph vertices and edges can also contain information. When this information is a simple label (i.e. a name or number) the graph is called *labelled graph*. Other times, vertices and edges contain some more information. These are called vertex and edge *attributes*, and the graph is called *attributed graph*. More usually, this concept is further specified by distinguishing between *vertex-attributed* (or *weighted graphs*) and *edge-attributed* graphs[3].

A *path* between any two vertices $u, u' \in V$ is a non-empty sequence of k different vertices $< v_0, v_1, \ldots, v_k >$ where $u = v_0, u' = v_k$ and $(v_{i-1}, v_i) \in E,\ i = 1, 2, \ldots, k$. Finally, a graph G is said to be *acyclic* when there are no cycles between its edges, independently of whether the graph G is directed or not.

2.2 Definition and classification of graph matching problems

Many fields such as computer vision, scene analysis, chemistry and molecular biology have applications in which images have to be processed and some regions have to be searched for and identified. When this processing is to be performed by a computer automatically without the assistance of a human expert, a useful way of representing the knowledge is by using graphs. Graphs have been proved as an effective way of representing objects [Eshera and Fu, 1986].

When using graphs to represent objects or images, vertices usually represent regions (or features) of the object or images, and edges between them represent the relations between regions. As an example, we can use a graph to represent a person using the graph shown in Figure 2.1: here all the main physical parts that one expects in a photograph or drawing of a person are shown in the form of vertices in a graph, while edges represent adjacency between the vertices. In this work, we will consider model-based pattern recognition problems, where the model is represented as a graph (the model graph, G_M), and another graph (the data graph, G_D) represents the image where recognition has to be performed. The latter graph is built from a segmentation of the image into regions. The graph in Figure 2.1 could serve as the model graph in a graph matching problem.

Similar graphs can be used for representing objects or general knowledge, and they can be

[3]Attributed graphs are also called *labelled graphs* in some references, and therefore these definitions are also known as *vertex-labelled* and *edge-labelled* graphs.

either directed or undirected. When edges are undirected, they simply indicate the existence of a relation between two vertices. On the other hand, directed edges are used when relations between vertices are considered in a not symmetric way. Note that the graph in Figure 2.1 is undirected, and therefore the attributes on each edge (u, u') are not specified to be from u to u' or vice-versa.

2.2.1 Exact and inexact graph matching

In model-based pattern recognition problems, given two graphs –the model graph G_M and the data graph G_D– the procedure of comparing them involves to check whether they are similar or not. Generally speaking, we can state the graph matching problem as follows: Given two graphs $G_M = (V_M, E_M)$ and $G_D = (V_D, E_D)$, with $|V_M| = |V_D|$, the problem is to find a one-to-one mapping $f : V_D \rightarrow V_M$ such that $(u, v) \in E_D$ iff $(f(u), f(v)) \in E_M$. When such a mapping f exists, this is called an isomorphism, and G_D is said to be isomorphic to G_M. This type of problems is said to be *exact graph matching*.

The term *inexact* applied to some graph matching problems means that it is not possible to find an isomorphism between the two graphs to be matched. This is the case when the number of vertices is different in both the model and data graphs[4]. This may be due to the schematic aspect of the model and the difficulty to segment accurately the image into meaningful entities. Therefore, in these cases no isomorphism can be expected between both graphs, and the graph matching problem does not consist in searching for the *exact* way of matching vertices of a graph with vertices of the other, but in finding the *best* matching between them. This leads to a class of problems known as inexact graph matching. In that case, the matching aims at finding a non-bijective correspondence between a data graph and a model graph. In the following we will assume $|V_M| < |V_D|$.

The interest of inexact graph matching has been recently increased in the last years due to the application of computer vision to areas such as cartography, character recognition, and medicine. In these areas, automatic segmentation of images results in an over-segmentation and therefore in the data graph containing more vertices than the model graph. That is why applications on these areas do usually require inexact graph matching techniques. In cartography, the typical example is when a graph is used to represent the knowledge extracted from of a map storing all the features. The matching with an image consists in identifying structures in the image with the help of the map. In character recognition, a model in the form of a graph is generated for each character and the objective is to find which is the model that best suits the analyzed image of a character. In medical images, graphs can be used to represent an anatomical atlas. As a concrete case of the latter, in brain imaging internal brain structures can be recognized with the help of a graph where each vertex represents a brain structure in the atlas, and edges represent spatial relationships between these structures. In the case of the recognition of human facial features from images the regions in the model represent each of the features to be recognized such as mouth, eyes and eyebrows. Experiments carried out in this thesis and presented in Chapter 7 are focused on these two real problems.

[4]A graph matching problem can be considered to be inexact when both graphs to be matched do not contain the same number of vertices and edges. However, it is important to note that in the case of some attributed graph matching problems, the fact of even having the same number of vertices and edges does not imply the existence of an isomorphism, and in the latter case that would also be an inexact graph matching problem. In this PhD thesis we will not consider the latter type of problems when referring to *inexact graph matching*.

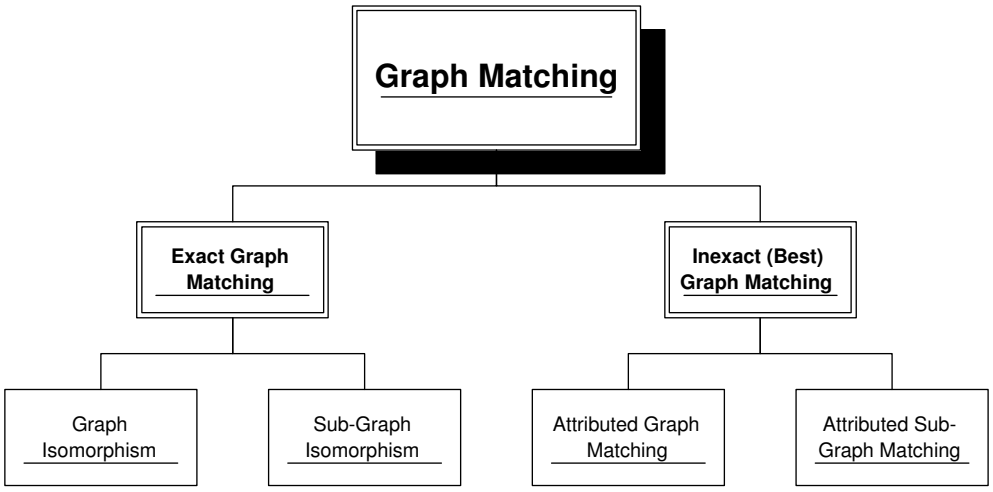


Figure 2.2: Classification of all the graph matching types into two main classes: exact graph matching and inexact graph matching (in which the best among all the possible non necessarily bijective matchings has to be found).

The best correspondence of a graph matching problem is defined as the optimum of some objective function which measures the similarity between matched vertices and edges. This objective function is also called *fitness function*[5].

In an *inexact graph matching problem* such as the ones described as examples, since we have $|V_M| < |V_D|$, the goal is to find a mapping $f' : V_D \rightarrow V_M$ such that $(u, v) \in E_D$ iff $(f(u), f(v)) \in E_M$. This corresponds to the search for a small graph within a big one. An important sub-type of these problems are *sub-graph matching problems*, in which we have two graphs $G = (V, E)$ and $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$, and in this case the aim if to find a mapping $f' : V' \rightarrow V$ such that $(u, v) \in E'$ iff $(f(u), f(v)) \in E$. When such a mapping exists, this is called a subgraph matching or subgraph isomorphism.

Exact and inexact graph matching are the terms that we will use in this thesis to differentiate these two basic types of graph matching problems. However, in the literature this type of graph matching problems are also called *isomorphic* and *homomorphic* graph matching problems respectively.

2.2.2 Graph matching using dummy vertices

In some inexact graph matching problems, the problem is still to find a one-to-one, but with the exception of some vertices in the data graph which have no correspondence at all. Real graph matching problem examples of the latter case can be found in [Finch et al., 1998b]. Similar examples can be found for medical applications, as in case of pathologies for instance. This case also happens when the over-segmentation procedure used for the data graph construction has been performed automatically and many regions appear to be components not present in the model or they are simply part of the background.

Examples can be found in the case of recognizing human structures in brain images. The anatomic atlas that the model graph represents is designed according to a healthy brain. However, if the data image that has to be recognized contains pathologies such as a tumor,

[5] In the literature fitness functions also receive the name of *energy functions*, and due to this graph matching algorithms are also regarded as *energy minimization algorithms*.

the graph matching procedure will not be able to match the vertices corresponding to these objects satisfactorily to any of the brain regions to be recognized. Similar examples can be found in satellite images for instance, when the model graph has been extracted from an image obtained some years before the actual photograph to be analyzed and new roads and buildings are present on the place.

More formally, given two graphs $G_M = (V_M, E_M)$ and $G_D = (V_D, E_D)$, the problem consists in searching for a homomorphism $h : V_D \rightarrow V_M \bigcup \{\emptyset\}$, where $\emptyset$ represents the null value, meaning that when for a vertex $a \in V_D$ we have $h(a) = \emptyset$ there is no correspondence in the model graph for vertex a in the data graph. This value $\emptyset$ is known in the literature as the *null vertex* or the *dummy vertex*.

Note that in these cases the fitness function that measures the fitness of the homomorphism h has to be designed taking into account that it should encourage the graph matching algorithm to reduce the number of vertices $a_1, a_2, \ldots, a_n \in V_D$ satisfying the condition $h(a_i) = \emptyset, i = 1, 2, \ldots, n$. In other words, nothing avoids that the homomorphism h for which $\forall a_i \in V_D \; h(a_i) = \emptyset$ is valid, but this solution would not represent any satisfactory result.

It is important to note that the use of the dummy vertex can be regarded as an additional vertex only for the model graph. However, in some problems the equivalent opposite case is also happening: using the same example of the human brain, the graph matching algorithm could face the case of having to recognize the data image of a patient that has undergone a lobotomy, and therefore a lobe is missing in the image. In this case, the graph matching problem has the particularity that there would not be a correspondence in the optimal solution for all the model vertices, but it does require the use of dummy vertices. This aspect is also common in other image recognition problems such as the aerial image recognition one, in which ancien buildings and factories can leave space in a newer photograph to new parks or roads.

2.2.3 Graph matching allowing more than one correspondence per vertex

Some other graph matching problems allow many-to-many matches, that is, given two graphs $G_M = (V_M, E_M)$ and $G_D = (V_D, E_D)$, the problem consists on searching for a homomorphism $f : V_D \rightarrow W$ where $W \in \mathcal{P}(V_M) \backslash \{\emptyset\}$ and $W \subseteq V_M$. In case of using also dummy vertices, W can take the value $\emptyset$, and therefore $W \in \mathcal{P}(V_M)$.

This type of graph matching problems are more difficult to solve, as the complexity of the search for the best homomorphism has much more combinations and therefore the search space of the graph matching algorithm is much bigger. These types of graph matching problems make sense when the segmentation step in the image does not satisfy the condition of having segmented all the regions completely, and therefore some of the automatically segmented regions are at the same time part of two or more model regions.

An important difficulty in graph matching problems that allow more than a single matching per data vertex is the design of a fitness function to measure the quality of each of the possible homomorphisms. Again, the number of model vertices matched to each single data vertex needs to be kept as low as possible so that the graph matching algorithm is forced to return more concrete results. These aspects are very dependent on each particular problem.

2.3 Complexity of graph matching

Graph matching is considered to be one of the most complex problems in object recognition in computer vision [Bienenstock and von der Malsburg, 1987]. Its complexity is due to its

combinatorial nature. Following the classification of graph matching problems explained in Section 2.2, as the nature of each of them is different, we will analyze their complexity separately.

2.3.1 Exact graph matching: graph isomorphism

This whole category of graph matching problems has not yet been classified within a particular type of complexity such as P or NP-complete. Some papers in the literature tried to prove its NP-completeness when the two graphs to be matched are of particular types or satisfy some particular constraints [Basin, 1994, Garey and Johnson, 1979], but it still remains to be proved that the complexity of the whole type remains within the NP-completeness at most.

On the other hand, for some types of graphs the complexity of the graph isomorphism problem has been proved to be of polynomial type. An example is the graph isomorphism of planar graphs, which has been proven in [Hopcroft and Wong, 1974] to be of polynomial complexity, although the cost of the leading constant also appears to be quite large.

As a result, it can be said that this issue remains as an interesting open theoretical problem, although it also encourages researchers to try to find polynomial time solutions for this type of graph matching problems.

2.3.2 Exact sub-graph matching: sub-graph isomorphism

This particular type of graph matching problems has been proven to be NP-complete [Garey and Johnson, 1979]. However, some specific types of graphs can also have a lower complexity. For instance, the particular case in which the big graph is a forest and the small one to be matched is a tree has been shown to be of a polynomial complexity [Garey and Johnson, 1979, Reyner, 1977].

2.3.3 Inexact graph matching: graph and sub-graph homomorphisms

In inexact graph matching, where we have $|V_M| \leq |V_D|$, the complexity is proved in [Abdulkader, 1998] to be NP-complete. Similarly, the complexity of the inexact sub-graph problem is equivalent in complexity to the largest common sub-graph problem, which is known to be also NP-complete.

2.4 State of the art in the literature

This section is a review of the literature on graph matching. All the references commented correspond to graph matching problems and methods, although some of them do not correspond with the concrete types of problems addressed in this dissertation and are included for the interested reader so that he can have an idea on the different subjects and groups working nowadays on the field. The interested reader can additionally find in [Jolion and Kropatsch, 1998, Kropatsch and Jolion, 1999, Jolion et al., 2001] works on practically all aspects discussed on this chapter: subgraph transformations for inexact matching, isomorphism between strong fuzzy relational graphs, a framework for region segmentation algorithms, image sequence segmentation, image analysis with a graph-based parallel computing model, and so on.

Even if this chapter has been focused on the graph matching types that can be found in the literature, this section focuses also in other related aspects such as the image processing techniques, the way of building the attributes, the different algorithms and formalizations of these problems, and their applications. The outline of this section is as follows: Section 2.4.1 reviews image processing techniques for graph matching purposes, as well as the generation of the graphs from images. Section 2.4.2 refers to the different ways of representing the information in graphs, and records references on geometrical properties of graph edit distances, and graph metrics. Finally, Sections 2.4.3, 2.4.4, 2.4.5, 2.4.6, and 2.4.7 explain the different graph matching problems that have been proposed based on genetic algorithms, probability theory-based approaches, decision trees, neural networks, and clustering techniques respectively. Each section comments examples of applications in which these methods have been applied. Finally, Section 2.4.8 discusses the advantages and disadvantages of the main approaches analyzed in the previous sections.

2.4.1 Image processing and graph construction techniques for graph matching

In most of the graph matching problems for image recognition, vertices in graphs represent regions of images, and the division in regions is the result of a segmentation procedure. This segmentation procedure is sometimes performed with the aid of an expert, but other times automatic segmentation and graph construction techniques are applied to create the graphs that are to be matched.

Attributed graph representation using fuzzy set theory

Fuzzy set theory has been used in the literature as a means to create vertex and edge attributes to be applied to graph matching. At a theoretical level there are many key works for these type of attributes, such as the representation of distances in images [Bloch, 1999b] and the representation of relative positions between objects [Bloch, 1999a]. Based on this idea, we can find many references in the literature using this type of attributes for inexact graph matching [Perchant et al., 1999, Perchant and Bloch, 1999], and for sub-graph matching [Hwan, 2001].

Fuzzy attributed graph models are proposed for very different image type representations, such as fingerprint verification [Fan et al., 2000] as a mechanism to structure the knowledge captured in conceptual models, and face detection from color images using a fuzzy pattern matching method [Wu et al., 1999].

Morphological graph matching and elastic graph matching

Elastic graph matching is a graph representation and matching technique that takes into account the possible deformation of objects to be recognized. Elastic graph matching usually consists of two consecutive steps, namely a matching with a rigid grid, followed by a deformation of the grid, which is actually the elastic part. The deformation step is introduced in order to allow for some deformation, rotation, and scaling of the object to be matched.

An example of the application of this type of representation is the identification and tracking of cyclones [Lee and Liu, 1999, 2000]. In the past decades, satellite interpretation was one of the vital methods for the determination of weather patterns all over the world, especially for the identification of severe weather patterns such as tropical cyclones as well

as their intensity. Due to the high variation and complexity of cloud activities for the tropical cyclone patterns, meteorological analysts all over the world so far are still relying on subjective human justification for cyclone identification purposes. Elastic graph matching techniques have been proposed for automatic detection of this problem.

Another usual application of this technique is the authentication of human faces, where the deformation patterns of the face are represented as graph attributes [Duc et al., 1999, Kotropoulos et al., 2000a,b, Tefas et al., 2002]. The different facial expression and the rotation patterns of the human head are intended to be represented in the form of graphs. Wavelet transforms are also used for creating the elastic face graph model, such as in [Ma and Xiaoou, 2001] where the face graph model is constructed using the discrete wavelet transform, and in [Duc et al., 1997, Lyons et al., 1999, Wiskott, 1999], where different 2D Gabor wavelet representation are used. Another related problem is the recognition of facial regions such as mouth and nose, which is also solved using these methods [Herpers and Sommer, 1998].

Morphological graph matching applies hyperplanes or deformable spline-based models to the skeleton of non-rigid discrete objects [di Ruberto and Dempster, 2001, Ifrah, 1997, Kumar et al., 2001, Rangarajan et al., 2001, Tefas et al., 2001], the graph being built from skeleton characteristic points. An approach to interactively define features and subsequently recognize parts regarding shape features using a sub-graph matching algorithm is proposed in [Sonthi, 1997]. An illustrative example of mathematical morphology applied to curve fitting and not to skeleton points can be found in [Bakircioglu et al., 1998], where curves on brain surfaces are matched by defining distances between curves regarding their speed, curvature, and torsion. Also, in [Dorai, 1996] another framework for representation and recognition of 3D free-form objects is shown, where the surface representation scheme describes an object concisely in terms of maximal surface patches of constant shape index. Finally, the application of planar surfaces to the modelling the data graph can be solved with probabilistic relaxation as shown in [Branca et al., 2000].

Another geometrical approach is introduced in [Shams, 1999], using generalized cylinders called *geons* as visual primitives to represent object models. Geons constitute a structural level intermediate between local features and whole objects, used here as a basis for powerful generalization between different view points. This work makes use of graph matching techniques and cross-correlation methods to find 2D projections of geons as partial matches between several images.

Examples of other applications of these techniques are: symmetry-based indexing of image databases [Sharvit et al., 1998], shape recognition from large image libraries [Huet and Hancock, 1999], recognition of shape features via multiple geometric abstractions [Sonthi, 1997], shape recognition using probability theory-based methods [Khoo and Suganthan, 2002, Shao and Kittler, 1999], structural matching by discrete relaxation [Wilson and Hancock, 1997], representation and recognition of 3D free-form objects [Cheng et al., 2001, Dorai, 1996], and hand posture recognition [Triesch and von der Malsburg, 2001].

Multiple graph matching

Some graph matching problems are based on the idea of having more than one model, and on performing graph matching to a database of models so that the model that best approaches the characteristics of the data graph is selected. Therefore, the aim here is to recognize a model rather than going deeply to recognize each of the segments of the data image. An illustrative example is found in [Messmer and Bunke, 1999] in which decision trees are used

for graph and subgraph isomorphism detection in order to match a graph to the best of a dictionary of graphs.

Due to the computation cost of using databases and to perform image processing on each, several references can be found about the subject of how to perform efficiently graph matching to all the models. For instance, in [Berretti et al., 2001] the usage of an indexing metric to organize large archives of graph models and for (sub)graph error correcting isomorphism problem is proposed. [Sharvit et al., 1998] proposes an approach for indexing pictorial databases based on a characterization of the symmetry in edge maps.

From a more theoretical point of view, [Williams et al., 1997] describes the development of a Bayesian framework for multiple graph matching. The starting point of this proposal is the Bayesian consistency measure developed by Wilson and Hancock [Wilson and Hancock, 1996] which is generalized from matching graph pairs to multiple graphs. In [Huet and Hancock, 1999] a graph-matching technique for recognizing line-pattern shapes in large image databases is described. The methodological contribution of the paper is to develop a Bayesian matching algorithm that uses edge-consistency and vertex attribute similarity. Recognition is performed by selecting from the database the image with the largest a posteriori probability.

One of the problems faced most commonly as multiple graph matching is the human face recognition [Kotropoulos et al., 2000a,b]. The particularity of [Mariani, 2000] is that the images of the database are all taken from the same person viewed from different angles, which aim is to estimate the orientation of the head. Following the same problem, [Hancock et al., 1998] compares the performance of two computer-based face database identification systems based on human ratings of similarity and distinctiveness, and human memory performance. Multiple graph matching has also been applied to many other problems such as the comparison of saliency map graphs [Shokoufandeh et al., 1999].

2.4.2 Distance measures, conceptual graphs, and graph edit distances and metrics

The graph edit distance between two graphs is defined as the number of modifications that one has to undertake to arrive from one graph to be the other. The distance between two graphs is defined as the weighted sum of the costs of edit operations (insert, delete, and relabel the vertices and edges) to transform one graph to the other. The fact of applying these concepts and removing vertices or edges in graphs is analyzed in many works, as removal will lead to smaller graphs and therefore the graph matching problem can be reduced in complexity.

[Fernandez and Valiente, 2001] proposes a way of representing attributed relational graphs, the maximum common subgraph and the minimum common supergraph of two graphs by means of simple constructions, which allow to obtain the maximum common subgraph from the minimum common supergraph, and vice versa. A distance measure between pairs of circular edges and relations among them is introduced in [Foggia et al., 1999]. This measure is to be applied in domains with high variability in the shape of the visual patterns (i.e. where a structural approach is particularly useful). In [Bunke, 1997] the relation between graph edit distance and the maximum common subgraph is analyzed, showing that under a fitness function introduced in this paper, graph edit distance computation is equivalent to solving the maximum common subgraph problem. Other related theoretical subjects are addressed in [Wang et al., 1995], in which two variants of the approximate graph matching problem are considered and analyzed: the computation of the distance between the graphs G_M and G_D, and the definition of the minimum distance between G_M and G_D

when subgraphs can be freely removed from G_D.

Other interesting approach of spectral type to solve graph matching translate this problem to the search of the maximal clique in the association graph. Examples on this are [Doob et al., 1980, Massaro and Pelillo, 2001].

As steps forward in this field, in [Myers et al., 2000] a framework for comparing and matching corrupted relational graphs is introduced. The authors develop the idea of the *edit-distance* and show how this can be used to model the probability distribution for structural errors in the graph-matching problem. This probability distribution is aimed at locating matches using maximum a posteriori label updates. The resulting graph-matching algorithm is compared with the one of [Wilson and Hancock, 1996], where the use of edit-distances is presented as an alternative to the exhaustive compilation of label dictionaries. In addition, [Ing-Sheen and Kuo-Chin, 2001] presents a region-based color image retrieval system using geometric properties, in which relational distance graph matching between two spatial relational graphs is performed to find the best matches with the minimum relational distance. Then, shape matching is applied to obtain the best match with the minimum geometric distance. Finally, in [Jolion, 2003] a new model for graph decimation is proposed with the aim of reducing the computational complexity of algorithms used for clustering, matching, feature extraction and so on.

Many useful applications of these techniques can be found in the literature. An illustrative example is shown in [Haris et al., 1999], where a method for extraction and labelling of the coronary arterial tree is proposed. In [Geusebroek et al., 1999], distance graph matching is applied to the segmentation of tissues, by basing the characterization of tissues on the topographical relationship between the cells. The neighborhood of each cell in the tissue is modelled by the distances to the surrounding cells, and comparison with an example or prototype neighborhood reveals topographical similarity between tissue and model. Optimal video clip ordering on a network in order to reduce computer network traffic in multimedia equipments is solved in [Ng and Shum, 1998] by representing the network as a graph and using graph edit distances and graph matching. The recognition of handwritten digits coming from a standard character database is also solved using this method [Foggia et al., 1999]. Chinese character recognition regarding stroke order and character recognition by analyzing graph distances have been proposed [Liu, 1997a]. Other work applying this approach involving clustering with distance measures is shown in [Gold et al., 1999].

Error correction graph matching

In error-correcting graph matching (or *error-tolerant graph matching* as it is also called) one considers a set of graph edit operations, and defines the edit distance of two graphs G_1 and G_2 as the shortest (or least cost) sequence of edit operations that transform G_1 into G_2. Error-correcting graph matching is a powerful concept that has various applications in pattern recognition and machine vision, and its application is focused on distorted inputs. It constitutes a different approach very similar to other graph matching techniques. In [Bunke and Shearer, 1998] this topic is addressed and a new distance measure on graphs that does not require any particular edit operations is proposed. This measure is based on the maximal common subgraph of two graphs. A general formulation for error-correcting subgraph isomorphism algorithms is presented in [Llados et al., 2001] in terms of adjacency graphs, and [Bunke, 1999] presents a study on the influence of the definition of fitness functions for error correcting graph matching.which reveals guidelines for defining fitness functions for optimization algorithms in error correcting graph matching. In addition, in [Messmer and

Bunke, 1998b] an algorithm for error-correcting subgraph isomorphism detection from a set of model graphs to an unknown input graph is introduced.

Applications of this technique are manyfold. [Fuchs and Men, 2000] present a procedure to compute error-correcting subgraph isomorphism in order to encode a priory knowledge for application to 3D reconstruction of buildings for cartography. Other examples are: relating error-correction and decision trees [Messmer and Bunke, 1998a], related to indexing of graph models [Berretti et al., 2001], crystallographic map interpretation [Oldfield, 2002], and construction of specific frameworks for error-tolerant graph matching [Bunke, 1998].

Conceptual graphs

Conceptual graphs have been used to model knowledge representations since their introduction in the early 80's. The formalism of conceptual graphs introduced in [Sowa, 1984] is a flexible and consistent knowledge representation with a well-defined theoretical basis. Moreover, simple conceptual graphs are considered as the kernel of most knowledge representation formalisms built upon Sowa's model. This formalism can capture semantics in the representation of data, and it offers some useful constructs which makes it a likely platform for a knowledge-based system. An extension of this concept to graph matching is introduced in [Baget and Mugnier, 2002], where reasoning in Sowa's model can be expressed by a graph homomorphism called projection. This paper presents a family of extensions of this mode, based on rules and constraints, keeping graph homomorphism as the basic operation. The use of conceptual graphs is also extended to some other works [Chen, 1996, Emami, 1997, Finch et al., 1997, Lai et al., 1999].

Apart from this type of graphs, graph matching has also been proposed, from both a theoretical or a practical view point, in combination with: matching graphs [Eroh and Schultz, 1998], minimal condition subgraphs [Gao and Shah, 1998], finite graphs [Bacik, 2001], weighted mean of a pair of graphs [Bunke and Gunter, 2001], median graphs [Jiang et al., 2001], and decomposition approaches [Messmer and Bunke, 2000]. Furthermore, different ways of representing patterns are analyzed in terms of symbolic data structures such as strings, trees, and graphs in [Bunke, 2001].

2.4.3 Graph matching using genetic algorithms

The fact of formulating complex graph matching problems as combinatorial optimization ones is not novel, and many references applying different techniques in this field can be found in the literature. Genetic algorithms are just an example of this.

Some of the works in the literature concentrate firstly on the type of crossover and mutation operators that are most suitable for graph matching problems. [Khoo and Suganthan, 2002] present a comparison between different genetic operators, and compares the performance of genetic algorithms when using two different types of individual representation. Furthermore, in [Singh and Chaudhury, 1997] an evolutive algorithm without crossover operators suitable for genetic algorithms is presented in order to obtain faster convergence to the solution. The authors also illustrate methods to parallelize their algorithm.

Another important aspect concerns the use of Bayesian measures. [Cross et al., 1997] describe a framework for performing relational graph matching using genetic search. The authors cast the optimization process into a Bayesian framework using the fitness measure previously introduced in [Wilson and Hancock, 1996]. This study shows that such Bayesian consistency measure could be efficiently optimized using a hybrid genetic search procedure

that incorporates a local search strategy using a hill-climbing step. The authors demonstrate analytically that this hill-climbing step accelerates convergence significantly. This idea is extended in [Cross et al., 2000], which is also a convergence analysis for the problem of attributed graph matching using genetic search. [Myers and Hancock, 2001] constitutes an extension of the Bayesian matching framework introduced in [Wilson and Hancock, 1997] by taking into account the cases with ambiguities in feature measurements. This paper also develops an evolutionary optimization framework that is applied on a genetic algorithm adapted for this search. Finally, the main idea in [Myers and Hancock, 2000] is that attributed graph matching problems (or *consistent labelling problems* as called by the authors) usually have more than one valid and satisfactory solution, and the aim of the authors is to propose a method to obtain different solutions at the same time during the genetic search, using an appropriately modified genetic algorithm.

As an example of real applications, works on the field on the human brain images recognition example can be found in [Boeres et al., 1999, Boeres, 2002, Perchant et al., 1999, Perchant and Bloch, 1999].

2.4.4 Graph matching using techniques based on probability theory

We can find in the literature many techniques applying probability theory to graph matching problems. A review on general purpose probabilistic graph matching can be found in [Farmer, 1999], where different types of probabilistic graphs, different techniques for their manipulation, and fitness functions appropriated to use for these problems are presented.

The first works applying probability theory to graph matching [Hancock and Kittler, 1990, Kittler et al., 1993] use an iterative approach using a method called probabilistic relaxation, and only take into account binary relations and assuming a Gaussian error. In these papers the use of binary relations is justified to be enough for defining the whole structure fully. Later a Bayesian perspective is used to account for both unary and binary attributes [Christmas et al., 1995, Gold and Rangarajan, 1996, Wilson and Hancock, 1996, 1997]. Later in [Williams et al., 1999] a comparative study of various deterministic discrete search-strategies for graph-matching is presented, which is based on the previous Bayesian consistency measure reported in [Wilson and Hancock, 1996, 1997]. Tabu search is proposed in this article as a graph matching algorithm. Finally, in [Finch et al., 1998a] a new fitness function is developed for graph matching based on a mixture model that gauges relational consistency using a series of exponential functions of the Hamming distances between graph neighborhoods. The effective neighborhood potentials are associated with the mixture model by identifying the single probability function of zero Kullback-Leibler divergence. This fitness function is simply a weighted sum of graph Hamming distances. Unfortunately, all these works do not consider dependencies between more than two vertices, although an attempt to consider more complex dependencies can be found in [Farmer, 1999] based on a new representation using association probabilistic graphs.

Other papers that can be classified under this section are mentioned next. [Shams et al., 2001] is a comparison between some conventional algorithms for solving attributed graph matching to the mutual information maximization method and an adaptation of multi-dimensional Gabor wavelet features. [Osman et al., 2000] constitutes an approach to based on modelling human performance for mental representation of objects by a machine learning and matching system based on inductive logic programming and graph matching principles. Finally, in [Liu et al., 2000] a method for recognition of Chinese characters is illustrated where both model and input characters are represented as complete relational graphs. The

graph-matching problem is solved with the Hungarian method.

Applying probabilistic relaxation to graph matching

Probabilistic relaxation is also used for solving the graph matching problem when formulated in the Bayesian framework for contextual label assignment [Christmas et al., 1995], and the same idea is applied in [Wilson and Hancock, 1997] combining several popular relational distance measures and an active process of graph-editing. More recently, [Wilson and Hancock, 1999] improves this Bayesian framework for matching hierarchical relational models.

Other articles related to this field are the application of planar surfaces to generate the data graph, and recognition performing graph matching with probabilistic relaxation in [Branca et al., 2000], application to representation and recognition of shapes in objects [Shao and Kittler, 1999], a generalization of the probabilistic relaxation labelling model by applying the relaxation to a graph of labels rather than to a pair of labels combined with random graphs to model scenes [Skomorowski, 1999], and a new relaxation scheme for graph matching in computer vision formulated as a process of eliminating unlikely candidates rather than finding the best match directly [Turner and Austin, 1998]. In the latter, a Bayesian development is used leading to an algorithm which can be implemented on a neural network architecture.

Applying the EM algorithm to graph matching

Another important approach is the EM algorithm. [Cross and Hancock, 1998] and [Finch et al., 1998b] are examples of this, in which two similar EM frameworks are proposed for two different graph matching problems. An algorithm for inexact graph matching that concentrates only on the connectivity structure of the graph and does not draw on vertex or edge attributes (structural graph matching) is proposed in [Luo and Hancock, 2001a]. An extension of the latter work is given in [Luo and Hancock, 2001b]. Also, [Kim and Kim, 2001] present a hierarchical random graph representation for handwritten character modelling in which model parameters of the hierarchical graph are estimated automatically from the training data by EM algorithm and embedded training.

Applying other probability theory methods to graph matching

Other examples of probability theory-based techniques are found in references commented on this thesis, on the following subjects: discrimination of facial regions using simulated annealing [Herpers and Sommer, 1998], multi modal person authentication using probabilistic relaxation [Duc et al., 1997, Mariani, 2000], Bayes theory applied to multiple graph matching [Huet and Hancock, 1999], multiple graph matching using Bayesian inference [Williams et al., 1997], and complex problems in proteonics using simulated annealing [Fariselli and Casadio, 2001].

2.4.5 Applying decision trees to graph matching

Decision trees have also been applied to graph matching. An example of this is [Shearer et al., 2001], in which decision trees are used for solving the largest common subgraph problem instead of applying queries to a database of models. Another example can be found in [Messmer and Bunke, 1998a] where decision trees are applied as a fast algorithm for the computation of error-correcting graph isomorphisms. Decision trees are also been applied to

multiple graph matching [Messmer and Bunke, 1999]. The decision tree is created using a set of a priori known model graphs generated from exact subgraph isomorphism detection.

2.4.6 Graph matching using neural networks

Neural networks have also extensively been applied to many graph matching problems. Very different types of neural networks have been tested trying to find the most suitable for each particular graph matching problem. Examples of this are the use of a mean field annealing neural network as a constraint satisfaction network for 3D object recognition [Lyul and Hong, 2002], a congregation of neural networks for the automatic recognition of cortical sulci of human brains in MRi images [Riviere et al., 2000, 2002], hybrid RBF network track mining techniques combined with integrated neural oscillatory elastic graph matching for tropical cyclone identification and tracking [Lee and Liu, 2000], dynamic link matching –i.e. a neural network with dynamically evolving links between a reference model and a data image– for face authentication with Gabor information on deformable graphs [Duc et al., 1999], and a 3D Hopfield neural network for sign language recognition [Huang and Huang, 1998]. On the other hand, in [Rangarajan et al., 1999b] a theoretical study of the convergence properties of a particular neural network is presented.

Other examples of the use of neural networks are: frontal face authentication problems [Kotropoulos et al., 2000a], overlapped shape recognition [Suganthan et al., 1999], recognition of hand printed Chinese characters [Suganthan and Yan, 1998], and a Bayesian development implemented as a neural network architecture [Turner and Austin, 1998].

2.4.7 Graph matching using clustering techniques

The use of clustering techniques has also been applied in graph matching, and many examples can be found in the literature such as [Carcassoni and Hancock, 2001] where clustering techniques are applied for attributed graph-matching. In this paper authors demonstrate how to improve method robustness to structural differences by adopting a hierarchical approach. Another example is [Sanfeliu et al., 2000] where a new type of graphs called *function-described graphs* are defined to represent an ensemble of attributed graphs for structural pattern recognition. The unsupervised synthesis of function-described graphs is studied here in the context of clustering a set of attributed graphs and obtaining a function-described graph model for each cluster. The idea is therefore to take into account the matching for groups of vertices instead of individually.

Other clustering-based approaches have also been applied to automatically recognize form documents [Fan et al., 1998], clustering with distance measures [Gold et al., 1999], automatic satellite interpretation of tropical cyclone patterns [Lee and Liu, 1999], and an eigen decomposition approach [Umeyama, 1988].

2.4.8 Discussion

In this Section, we discuss the pros and cons of the main classes of approaches found in the literature.

The complexity of the graph matching approaches make it very difficult to take into account all types of dependencies between the vertices and edges in the model and data graphs. That is why in most existing frameworks defined for this problem (e.g. Bayesian, morphological and EM frameworks described in the previous sections) only unary and binary

relations between vertices are taken into account. The advantage of such simplifications is that the complexity of graph matching is reduced when tackling the original problem, and this allows the use of techniques that require evaluating a lot of individuals through the search for the best solution. On the other hand, these approaches are ignoring many relationships that can be decisive when searching for a satisfactory matching. Due to this, and depending on the type of problem, these simplifications could lead to non satisfactory results.

Regarding the application of genetic algorithms in graph matching problems, the performance that they present is remarkable, specially when they are combined with dedicated frameworks such as Bayesian ones. In addition, the number of evaluations that they require to reach a solution is very low compared to other stochastic heuristic searches. However, one of their drawbacks is that their performance is very dependent on the high number of input parameters that have to be set. This dependence is specially dependent on the type of cross-over and mutation operators selected. The literature presents plenty of different GAs which are more appropriated for particular types of problems, and the user who wants to apply these methods needs to have a lot of experience in order to tune properly the algorithm and obtain satisfactory results. Another important drawback of GAs is the fact that when the fitness function contains many local maxima these algorithms get easily stuck on them. Moreover, in the definitions of similarities proposed for building the fitness functions for GAs, again only unary and binary relationships are taken into account. This simplification is understandable since the number of evaluations that GAs require to reach the optimum solution in such complex problems is high, but important relationships can be missed out.

Regarding the use of neural networks, they have shown in the last years a good performance in solving problems as complex as graph matching ones. The ability of neural networks to take into account particular restrictions of problems has also been proved experimentally. However, regarding the way of representing the knowledge, the main drawback of this approach is that this information is encoded in a kind of *black box* that does not allow easily to infer how the network reasons. Other approaches based on probabilistic structures (such as Bayesian networks) have the ability to express in a format directly comprehensible for researchers how the knowledge is represented.

2.5 Graph matching problem types selected for this thesis

In this thesis, we address the problem of inexact graph matching. This choice has been done taking into account the NP-complete complexity of this problem, which is more difficult to solve than the exact graph matching problem. One of the objectives of this PhD thesis is to show the validity of the EDA paradigm when applied to graph matching problems, and therefore the inexact sub-graph matching category has been chosen. However, as graph and sub-graph inexact matching problems are of equivalent category, we will simply use the term *inexact graph matching* to refer to both of them. In addition, we have set an additional constraint that makes the problem even more restrictive: in order to consider a homomorphism as valid, all the vertices in the model graph will have at least a vertex in the data graph that has been matched to it. Examples on synthetic data and on two real applications are explained in more detail in Chapters 6 and 7.

Chapter 3

Graph matching as a combinatorial optimization problem with constraints

'The beginning of knowledge is the discovery of something we do not understand.'

Frank Herbert

3.1 Introduction

Let us introduce some notations. We will call $G_M = (V_M, E_M)$ the model graph and $G_D = (V_D, E_D)$ the data graph which contains the image segments that have to be matched against G_M, where V_i is the set of vertices and E_i is the set of edges of graph G_i $(i = M, D)$. In inexact graph matching problems G_D is assumed to contain more segments than G_M, as that is the case when generated from an over-segmented image. Usual constraints for the matching are that each vertex of G_D is matched with exactly one vertex of G_M (which assumes that no unexpected objects are present on the image data) and that each vertex of G_M is matched at least with one vertex of G_D (which assumes that all model objects are indeed present in the image)[1]. In Section 3.2, we summarize how such constraints are taken into account in existing methods.

In order to solve any problem using combinatorial optimization techniques, it is necessary to find a means to represent each of the solutions to the problem as a vector of values (usually this is known as an *individual*), as well as a *fitness function* to evaluate each of these solutions so that the algorithm can distinguish among good and bad solutions. In Section 3.3, we address the problem of representing solutions as individuals. Constraints are introduced either in the representation itself, or as an elimination process of incorrect individuals. In Section 3.4 we deal with the definition of fitness functions.

[1]This implies in particular that the method is not directly applicable to incomplete models or to images with pathologies (in the case of medical images).

3.2 Graph matching problems with special constraints in the literature

Some real graph matching problems contain more specific constraints that any valid homomorphism has to satisfy apart from the ones commented so far. These constraints are very specific for each graph matching problem, but they usually taken into account when generating the graph attributes. [Feder and Vardi, 1999] propose a framework based on group theory for constraint satisfaction in NP-complete problems such as attributed graph isomorphism. Authors claim that this framework can also be applied to any other type of graph matching problems. Also, the complexity of rules and constraints in conceptual graphs is analyzed in [Baget and Mugnier, 2002]. In [Blake, 1994] the idea of partitioning the graph matching problem into sub-problems under the control of constraints is considered. Additionally, in [Liu, 1997b] a method that provides a design advisory system for mechanical components is developed by building a scheme for modelling constraints by defining a new type of graphs called *constraint graphs*. In this work, the representation and management of constraints are elaborated using a graph-based approach in the form of constraint graphs. The product design optimization is formulated as a graph matching problem and solved by integer programming techniques.

However, regarding graph matching algorithms and methods to find correct solutions, one of most applied mechanisms for obtaining only valid solutions at the end of the search in problems with constraints is the use of a particular type of neural networks called Hopfield networks [Suganthan et al., 1995, Suganthan and Yan, 1998, Suganthan et al., 1999]. This approach encodes the constraints of the problem in a way that a fitness function does not have to take them into account. This is achieved due to the ability of the network to learn the constraint parameter adaptively, as the adaptation scheme eliminates the need to specify the constraint parameter empirically and generates valid and better quality mappings. Another approach is proposed in [Gangnet and Rosenberg, 1993], which is an illustration on how to apply constraints using the method proposed for constraint solving in [Freeman-Benson et al., 1990]. The authors claim that their approach is simple and efficient for constraint resolution when applied to graph matching problems. Finally, a mean field annealing neural network is proposed as a constraint satisfaction network in [Lyul and Hong, 2002].

Examples of real graph matching problems with constraints introduced in the literature are frontal face authentication [Ma and Xiaoou, 2001, Tefas et al., 2001], and topographical constraints in face recognition [Wiskott, 1999].

3.3 Representations of graph matching solutions by means of individuals

A solution to a graph matching problem is an association between vertices of G_M and vertices of G_D satisfying the required constraints of the particular problem. Regarding the way of representing a solution by means of individuals, the type of values that these can contain allow us to classify the different individual formats or representations as

- discrete individuals: all the values in the individual are discrete,
- continuous individuals: all the values in the individual are continuous.

The type of individual representation used is also a factor that is determined by the type of combinatorial optimization technique to be applied[2]. For instance, Genetic Algorithms (GAs) are usually applied only to discrete individuals, while other techniques such as Evolutionary Strategies (ESs) can only be applied to continuous individuals. EDAs can be applied to both types of individuals, as under this paradigm there are many different algorithms available that will be described in the next chapter.

It is important to note that, independently of using discrete or continuous individuals, different individual representations can be used for a same problem. When selecting an individual representation it should be taken into account that this choice is an important factor that will condition the evolution of the search process. Just as an example on the type of factor that should be considered, it is convenient not to use ambiguous individual representations in which two different individuals represent a same solution for the problem. However, in some cases the individual representation used allows the existence of ambiguities but this aspect is controlled deliberately.

3.3.1 Individual representations in the discrete domain

We denote by $V_M = \{u_M^1, u_M^2, \ldots, u_M^{|V_M|}\}$ and $V_D = \{u_D^1, u_D^2, \ldots, u_D^{|V_D|}\}$ the set of vertices of G_M and G_D respectively. There are different ways of representing individuals with discrete values for inexact graph matching, two examples of which are as follows:

Representation 1: Individuals with $|V_M| \cdot |V_D|$ genes or variables c_{ij}, that only take values 0 and 1.

For $1 \leq i \leq |V_M|$ and $1 \leq j \leq |V_D|$, $c_{ij} = 1$ means that the vertex u_D^j of G_D is matched with the vertex u_M^i of G_M.
This is the representation used for instance in [Boeres et al., 1999].

Representation 2: Individuals with $|V_D|$ genes or variables, X_i $i = 1, \ldots, |V_D|$, where each of them that can take any value between 1 and $|V_M|$.

For $1 \leq k \leq |V_M|$ and $1 \leq i \leq |V_D|$, $X_i = k$ means that the vertex u_D^i of G_D is matched with the u_M^k vertex of G_M.

The latter is the representation of individuals that we have selected for EDAs and GAs for reasons that will be explained in Section 3.3.3. Therefore, in this case individuals contain as many variables as vertices are in G_D ($|V_D|$ vertices), and each of the variables can take as many values as vertices are in G_M ($|V_M|$ values).

The biggest drawback of using any of these two representations is that some individuals can represent solutions that are not acceptable for the problem, that is, that do not satisfy a set of constraints defined beforehand. In this thesis we will deal with very particular constraints. These will be discussed later in Section 3.3.3.

Representation 3: Individual representation based on a permutation of values. The individual can also be represented as a permutation of discrete values. A permutation is a list of numbers in which all the values from 1 to n have to appear in an individual of size n. In this case, the values of the individual do not represent directly the matching of each

[2]It must be said that even in the case of using a continuous individual representation we will still have a combinatorial optimization problem since in our approach continuous individuals are transformed to a permutation of discrete values. This procedure is explained later in this chapter.

G_D, but the order in which each vertex of G_D will be matched following a predefined procedure. Permutation-based individual representations have been typically applied to problems such as the Travelling Salesman Problem [Flood, 1956] or the Vehicle Routing Problem [Fiala, 1978]. An illustrative example of applying permutations to genetic algorithms for solving these problems can be found in [Freisleben and Merz, 1996]

For the particular graph matching problems in this thesis, we have applied individual representations 2 and 3 for the discrete domain. In both cases, discrete individuals have a length of $|V_D|$ variables, where the number of values that each variable can take are $|V_M|$ or $|V_D|$ for Representations 2 and 3 respectively. In addition, due to the type of graph matching problems that this thesis deals with, we have decided to add constraints that have to be satisfied by any valid solution (independently of the representation used), which are defined in Section 3.3.3

From the permutation to the solution it represents

Once having the permutation, the individual has to be translated to the solution it symbolizes so that it can be evaluated. Because the evaluation of an individual is executed many times by any graph matching algorithm, it is important that this translation is performed by a fast and simple algorithm.

Evaluating a solution requires to compare vertices of G_M and vertices of G_D. If we have a permutation, a solution can be evaluated by comparing the vertices in the order given by the permutation and deciding which is the most similar by means of a similarity function, $\varpi(i,j)$, defined to compute the similarity between vertices of G_D. The similarity measures used so far in the literature have been applied to two vertices [Perchant et al., 1999, Perchant and Bloch, 1999, 2000b, Perchant, 2000, Perchant and Bloch, 2000a], one from each graph, and their goal has been to help in the computation of the fitness of a solution, that is, the final value of a fitness function.

Figure 3.1 shows a procedure that could be used in order to translate a permutation of discrete values –an individual $\boldsymbol{x}= (x_1, \ldots, x_{|V_M|}, x_{|V_M|+1}, \ldots, x_{|V_D|})$– to the solution that it represents. This procedure follows an idea inspired on the partitional clustering algorithms proposed in [McQueen, 1967] and specially in [Forgy, 1965], and it is divided in two main steps as follows:

- In the first step the values x_1 to $x_{|V_M|}$ are directly matched to vertices of G_M from 1 to $|V_M|$ respectively.
- For each of the next values of the individual, $x_{|V_M|+1}$ to $x_{|V_D|}$, and again following the order given in the permutation, the most similar vertex of G_D based on the similarity measure $\varpi(i,j)$ will be selected, and its previously matched vertex of G_M will be also chosen as the matching for the new vertex.

Permutation-based representations can be used for any graph matching problem. A detailed example of this method, as well as a deeper explanation of it can be found in Appendix A and in [Bengoetxea et al., 2001c,d].

Another important aspect of using a permutation-based approach is the fact that the cardinality of the search space is $|V_D|!$, which is different from the $|G_M|^{|G_D|}$ cardinality of

From permutations to the solution

Definitions

$|V_M|$: number of vertices in the model graph G_M
$|V_D|$: number of vertices in the data graph G_D (where $|V_D| > |V_M|$)
Size of the individual (the permutation): $|V_D|$
$\boldsymbol{x}= (x_1, \ldots, x_{|V_D|})$: individual containing a permutation
$x_i \in \{1, \ldots, |V_D|\}$: value of the i^{th} variable in the individual
$PV_i = \{x_1, \ldots, x_{i-1}\}$: set of values assigned in the individual to the variables $X_1, \ldots, X_{i-1}$. (PV = previous values)
$\varpi(i, j)$: similarity function that measures the similarity between vertex i and vertex j (with $i, j \in V_D$)

Procedure

Phase 1

For $i = 1, 2, \ldots, |V_M|$
(first $|V_M|$ values in the individual, treated in order)
Match vertex $x_i \in V_D$ of data graph G_D
with vertex $i \in V_M$ in model graph G_M

Phase 2

For $i = |V_M| + 1, \ldots, |V_D|$
(remaining values in the individual, treated in this order)
Let $k \in PV_i$ be the most similar vertex to x_i from
all the vertices of PV_i ($k = \arg\max_{j=1\ldots i-1} \varpi(i, j)$)
Match vertex $x_i \in V_D$ of data graph G_D
with the vertex that is matched to vertex k of G_M

Figure 3.1: Pseudocode to compute the solution represented by a permutation-based individual.

the previously described individual representation. Moreover, the fact of using permutations and a similarity measure $\varpi(i, j)$ leads to redundancies in the solutions, as two different permutations may correspond to the same solution. An example of this is also shown again in Appendix A and in [Bengoetxea et al., 2001a,c].

Definition of the similarity

The definition of the similarity function $\varpi(i, j)$ is very important in the translation procedure from a permutation-based individual to the solution it symbolizes. There are three main aspects to be taken into account when defining this function for its use in the second step of the translation procedure:

1. **Which vertices have to be compared.** The two vertices to compare can be of graph G_D, or both from graph G_D (e.g. $\varpi(i, j)$ has two parameters, vertex i and vertex j, we know that $i \in V_D$, the choice is either $j \in V_M$ or $j \in V_D$). Other approaches are also possible, for instance combining the similarity of vertices from both G_M and G_D

and assigning them a weight, or also by having a fitness function capable of returning a value for individuals that are not correct permutations.

2. **Recalculating or not the similarity measure as the individual is being generated.** The function $\varpi(i,j)$ could be constant or not for any two vertex values. In the latter case, the similarity value can be assumed to vary depending on the vertices that have already been matched while the individual is being instantiated. In that case, each time that a variable is instantiated, the values of $\varpi(i,j)$ will vary depending on the effect of the new and the previous matchings. In other words, this means that in the second step of the translation an extra clustering procedure would be required, such as the cluster analysis proposed in [Forgy, 1965], in order to update the function $\varpi(i,j)$.

3. **Selection of the attributes of both vertices and edges that will be used for measuring the similarity.** This aspect is specific for each particular problem. This choice will determine to an important degree the behavior of the algorithm.

In [Bengoetxea et al., 2001c,d] we propose a similarity measure $\varpi(i,j)$ that is used to measure the similarity between vertices of the same graph G_D, which has fixed similarity values and therefore no variations apply during the instantiation of individuals.

3.3.2 Individual representations in the continuous domain

Continuous EDAs provide different algorithms for the continuous domain that could be more suitable for some problems (see Section 4.4). Nevertheless, the representation of individuals has to be defined in the most appropriated way to obtain the best performance with this approach.

Individuals in this approach consist of a continuous value in $\Re^n$. The main goal is to find a representation of individuals and a procedure to obtain an univocal solution to the matching from each of the possible permutations.

For this case we propose as an example a strategy based on the mechanism of the previous section, trying to translate the individual in the continuous domain to a correct permutation in the discrete domain. This translation will give us a permutation of discrete values, and we would proceed as explained in Section 3.3.1 in order to obtain the solution that the individual symbolizes.

Again, this new representation of individuals does not give a direct meaning of the solution it represents. This new type of representation can also be regarded as a way to change the search from the discrete to the continuous world, which allows us to apply techniques to estimate densities that are completely different from the ones used in discrete domains.

As this procedure to translate from the continuous world to the discrete one has to be performed for each individual as an additional step to the method introduced in the previous section, this process has to be fast and simple enough in order to reduce computation time.

Taking these aspects into account, we show as an example a method that can easily be applied to graph matching problems. The individual size is set to $|V_D|$ as before, where each of the variables of the individual can take any value following a Gaussian distribution. In order to obtain a translation to a discrete permutation, we propose to order the continuous values of the individual, and to set its corresponding discrete values by assigning to each $x_i \in \{1,\ldots,|V_D|\}$ the respective order in the continuous individual. The procedure described in this section is shown as pseudocode in Figure 3.2.

From a continuous value in $\Re^n$ to a permutation

Definitions

$n = |V_D|$: size of the individual, which is the number of vertices in data graph G_D (the permutation)

$\boldsymbol{x^C} = (x^C_1, \ldots, x^C_{|V_D|})$: individual containing continuous values (the input)

$\boldsymbol{x^D} = (x^D_1, \ldots, x^D_{|V_D|})$: individual containing a permutation of discrete values (the output)

$x^D_i \in 1, \ldots, n$: value of the i^{th} variable in the individual

Procedure

Order the values $x^C_1, \ldots, x^C_{|V_D|}$ of individual $\boldsymbol{x^C}$ using any fast sorting algorithm such as Quicksort

Let k_i be position in which each value x^C_i $(1 \leqslant i \leqslant |V_D|)$ occupies after ordering all the values

The values of the individual $\boldsymbol{x^D}$ will be set in the following way: $\forall i = 1, \ldots, |V_D|$ $x^D_i = k_i$

Figure 3.2: Pseudocode to translate from a continuous value in $\Re^n$ to a discrete permutation composed of discrete values.

3.3.3 Conditions for a correct individual in a graph matching problem

Each of the different image recognition problems that are solved using graph matching techniques has particularities that have to be taken into account by any acceptable solution. Due to this, the way of considering the constraints in a graph matching approach is an important aspect that has to be considered attentively. A useful way of dealing implicitly with constraints is to select an individual representation that does not allow the possibility for invalid individuals to appear. Unfortunately, this representation does not always exist, and satisfaction of the constraints need to be tackled using explicit mechanisms.

In order to show how problem specific constraints can be tackled explicitly, independently of the representation of individuals of choice, we will consider that any individual to represent a *correct solution* for graph matching examples in this thesis satisfies all the following 3 conditions:

1. All the vertices in G_D must have a corresponding match with a vertex in G_M. For some type of images such as the ones in cartography this condition is not necessary: sometimes when comparing a new photograph with a previous map of the same area additional objects such as new roads and new houses that cannot be matched could appear. For these cases, the use of a dummy vertex is advised (this technique is described in Section 2.2.2). In our graph matching problems in this thesis we do not consider such cases and assume that all of our segments correspond at least to a vertex in G_M, and as a result no dummy vertex has to be defined (i.e. $\emptyset$ label has to be added).

2. Each vertex in G_D can have at most a vertex matched in G_M. It is not acceptable that a segment in the image corresponds to more than a segment in the atlas. This aspect is analyzed in general terms for graph matching problems in Section 2.2.3. This condition is satisfied in the human brain structure recognition problem [Perchant and Bloch, 1999] by applying over-segmentation techniques to the image, making sure that an object in the model appears always properly divided in the segmented image. None of the individual representations defined described in Section 3.3.1 do not allow matching a vertex of G_D against two or more segments in G_M, and thus using any of those representations this condition would not need to be controlled in the graph matching algorithms (Section 4.2.2 explains more details about controlling the generation of individuals).

3. All the vertices in G_M must have at least a matched vertex of G_D. We have decided to add this additional assumption to the graph matching problems in this thesis, because it also needs to be satisfied in some real graph problems such as the one that is introduced in Section 7.2 for the recognition of human brain images.

It is important to ensure that the final solution of any graph matching algorithm corresponds to a correct individual. Moreover, whichever the representation of individuals chosen, any acceptable individual obtained with any of the methods proposed in GAs, EDAs, or any other graph matching approach in order to be considered as acceptable for our problem.

The reason for considering constraints only for vertices and not for the edges is that when using graph matching for image recognition in vertices represent image regions, and the only constraints that considered here are about the properties that the best solution must satisfy. In this sense, the arcs are used for representing information about relations between the regions, but the final solution simply shows the matching vertex by vertex. Constraints such as taking into account only some specific edges are also considered in these problems, but these are applied when defining the fitness functions. The main difference between many fitness functions proposed in Section 3.4 are actually the edges that are considered or ignored.

The choice of a particular representation of individuals reviewed in Section 3.3.1 is important to reduce the effort of controlling the satisfaction or not of these three conditions. If we compare Representations 1 and 2 all the requirements to check whether an individual is correct or not can be summarized as follows (the need to analyze the first condition is not required since we have decided not to use dummy vertices in our problems):

1. Every vertex of G_D must be matched with one and only one vertex of G_M (2 first conditions).

- **Representation 1:**

$$\sum_{i=1}^{|V_M|} c_{ij} = 1 \quad \forall j \in \{1, \ldots, |V_D|\}$$

- **Representation 2:**

  This condition is inherent in the representation, there is no need to check this condition.

2. Every vertex of G_M must be matched at least with one vertex of G_D (third condition).

1	1	1	2	2	2	2	2	3	4	5	6	7	8

(a)

1	1	1	1	2	2	2	2	3	3	3	3	4	4

(b)

Figure 3.3: Representation of a correct individual (a), and an incorrect individual (b) for our graph matching problem example, for the case that G_M (the model graph) contains 8 vertices and G_D (the data graph representing the segmented image) contains 14 vertices.

- **Representation 1:**

$$\forall i \in \{1, \ldots, |V_M|\} \quad \exists j \quad c_{ij} = 1$$

- **Representation 2:**

$$\forall i \in \{1, \ldots, |V_M|\} \quad \exists j \quad X_j = i$$

This shows that the Representation 2 of individuals only requires to check actively the third constraint. This is the reason why this representation will be used in our experiments on this thesis. Note that the constraints we impose are a simplification of more general inexact graph matching problems. The interest is that they restrict the set of possible solutions.

The case of Representation 3 of individuals of Section 3.3.1 is not comparable with the other two ones. The reason for this is that the individuals themselves are not the ones that have to satisfy these three conditions, and it is the solution that they symbolize the one that needs to satisfy them. In fact, following the procedure described in Section 3.3.1 we ensure that any individual will satisfy these constraints. As a result, we do not need to worry about the correctness of the particular solutions when using this third representation of individuals. However, the single constraint that these individuals must satisfy is that they must contain a correct permutation, which becomes in practice a more restrictive property to satisfy than the three conditions introduced in this section.

In order to illustrate the meaning of these constraints, we will consider as an example a model graph G_M containing 8 vertices (labelled from 1 to 8) and a data graph G_D representing the segmented image which contains 14 vertices (labelled from 1 to 14). If we use the Representation 2 of individuals of Section 3.3.1 for our graph matching problem, the individual in Figure 3.3a shows an example of a correct matching, where the first three vertices of G_D are matched with the vertex number 1 of G_M, the next five vertices of G_D are matched with the vertex number 2 of the graph G_M, and so on. In such a representation, when generating new individuals the result can be an invalid solution (i.e. it does not satisfy our constraint). The individual in Figure 3.3b is an example of an invalid solution for our problem. In this case, the individual represents a matching for some of the vertices of G_M, but there are still other vertices from this model graph (vertices 5, 6, 7 and 8) that have no match at all.

3.3.4 What to do with incorrect individuals?

It is important to decide what to do with these incorrect individuals. In the literature many papers can be found in the domain of GAs where the existence of incorrect individuals is allowed hoping that these individuals can lead to the generation of fitter correct individuals. In many other articles [Richardson et al., 1989, Smith and Tate, 1993], individuals not representing a valid solution for the problem are either corrected or penalized. Here, we consider all these possibilities and compare them to each other (see Section 4.5). EDAs have been applied to problems with constraints in very few cases [Bengoetxea et al., 2000, 2001a,b], and this dissertation shows and introduces many ways on which EDA approaches can be adapted for this type of problems taking as an example a combinatorial optimization problem with constraints such as inexact graph matching.

3.4 Fitness functions applied to graph matching

The objective of the fitness function is to have a means to evaluate each of the possible individuals so that the search algorithm can compare the different solutions and act in consequence to find the best solution. Therefore, it is important to define appropriately this function in order to assess the search of any graph matching algorithm. The behavior of the fitness function is also very dependent on the individual representation selected and both elements are very linked to each other. The influence of the fitness function definition in finding the best matching between graphs has also been subject of research [Bunke, 1999].

This section discusses the different possibilities for defining a fitness function. The decision of which fitness function to use is very important when using metaheuristics, as important as the individual representation that is used, because it will also determine how difficult is the search for an optimal individual that will lead to a correct recognition.

An ideal fitness function should verify the following properties (assuming that the best fitness is the highest):

1. It should be monotonic, that is, the closer an individual is to the optimum solution the larger its fitness has to be. This assumption is very strong since local maxima are very often present, but these should be avoided as much as possible. However, above all, the most important is that the optimum solution has the largest value.

2. It has to avoid ambiguities, and therefore different individuals should have a different fitness value when they are evaluated.

3. The fitness function has to take into account only the appropriate vertex and edge attributes, the ones that are more meaningful in the search for the optimum solution. In many cases, it is also important to assign a different weight to vertex and edge similarities, giving more importance to the edge attributes as these are the ones that take best into account relationships between regions and not yet the regions themselves.

4. As a secondary aim, it has to be easy and fast to compute, as this is executed many times and it could otherwise delay too much the execution of the whole algorithm.

Whichever the fitness function selected, it is important to take into account that the fitness function is always very particular for each problem, and that the fitness function itself influences the behavior and the final results obtained by the graph matching algorithm. In

addition, main differences appear between fitness functions defined for discrete and continuous domains. In the next sections fitness functions for discrete representations are analyzed, while in the last one we focus on fitness functions for the continuous domain.

3.4.1 Graph attributes and similarities

Both the model and data graphs are attributed, and the recognition process is based on similarities between these attributes in the form of fitness functions. The type of attributes that can be used for the computation of the fitness function and the similarity between graphs can be based on different paradigms. Examples of this are the use of fuzzy set theory following the theoretical background developed in [Bloch, 1999a,b]. For instance, many references on fuzzy set theory applied to graph matching concentrate on building dedicated fitness functions. In [Perchant et al., 1999, Perchant and Bloch, 1999] the fitness function is based on the fuzzy attributes of each of the model and data graphs and the fitness value of each solution is computed from comparing directly the similarity between these, whereas in [Suganthan et al., 1998] the fitness function is computed by fusing the information encoded in the attributes using nonlinear fuzzy information aggregation operators. In both cases the learning of the different matching for each of the attribute-pairs is formulated as an optimization problem, although in the former a genetic algorithm is used and in the latter a learning procedure based on gradient projection algorithm is applied.

Apart from using fuzzy set theory for graph matching, other alternatives are the use of vector-type and geometric attributes, and attributes based on probability theory using distribution divergences and likelihood values. We will provide examples of these attribute types.

Once having defined vertex and edge attributes, similarities between attributes of the model and of the data are defined. The similarity between any two vertices $a_D \in V_D$ and $a_M \in V_M$, denoted by $c_N(a_D, a_M)$, is defined in terms of vertex attributes and their semantics, and we assume to be normalized by returning a value in $[0, 1]$ where a higher value represents a higher similarity. Analogously, the similarity between two edges $e_D = (a_D^i, a_D^j)$ of E_D and $e_M = (a_M^k, a_M^l)$ of E_M is denoted by $c_E(e_D, e_M)$, is defined in terms of edge attributes and their semantics, and we also assume it to be normalized in $[0, 1]$. Both $c_N(a_D, a_M)$ and $c_E(e_D, e_M)$ are very particular to the problem and are defined after selecting the type of attributes to be used.

The next step is to define a fitness function to evaluate each of the solutions generated by the graph matching algorithm in order to find the best homomorphism h which satisfies the conditions $h : V_D \rightarrow V_M$ and $\forall a_M \in V_M \; \exists a_D \in V_D \;\mid\; h(a_D) = a_M$ as well as structural and similarity constraints.

3.4.2 f_1: only taking into account the matched vertices

Based on the definitions of the vertex similarity $c_N(a_D, h(a_D))$ and the edge similarity $c_E((a_D^i, a_D^j), (h(a_D^i), h(a_D^j)))$, the first proposal for a fitness function is to define the global similarity of a homomorphism h as:

$$f_1(h) = \frac{\alpha}{|V_D|} \sum_{a_D \in V_D} c_N(a_D, h(a_D)) + \frac{1-\alpha}{|E_D|} \sum_{(a_D^i, a_D^j) \in E_D} c_E((a_D^i, a_D^j), (h(a_D^i), h(a_D^j))) \tag{3.1}$$

where $0 \leq \alpha \leq 1$ is a parameter used for tuning the relative importance of vertex and edge similarity.

3.4.3 f_2: taking into account all the vertices and similarities

The fitness function described in the previous section only considers similarity between vertices (or edges) that are mapped by the homomorphism h, but it does not take into account the fact that unmapped vertices could also have a high similarity which may be not desirable. Authors in [Perchant and Bloch, 1999] adopt this idea and propose a fitness function for the graph matching problem applied to medical images of the brain that has been used latter in [Perchant et al., 1999] and [Boeres et al., 1999]. This second proposal for a global similarity function accounting for such situations is presented in the following way:

$$f_2(h) = \frac{\alpha}{|V_D||V_M|} \sum_{(a_D,a_M)\in V_D\times V_M} [1 - |\theta_N(a_D,a_M) - c_N(a_D,a_M)|] + \frac{1-\alpha}{|E_D||E_M|} \sum_{(a_D^i,a_D^j)\in E_D,(a_M^k,a_M^l)\in E_M} [1 - |\theta_E((a_D^i,a_D^j),(a_M^k,a_M^l)) - c_E((a_D^i,a_D^j),(a_M^k,a_M^l))|], \tag{3.2}$$

where

$$\theta_N(a_D,a_M) = \begin{cases} 1 & \text{if } a_M = h(a_D), \\ 0 & \text{otherwise,} \end{cases}$$

$$\theta_E((a_D^i,a_D^j),(a_M^k,a_M^l)) = \begin{cases} 1 & \text{if } a_M^i = h(a_D^k) \text{ and } a_M^j = h(a_D^l), \\ 0 & \text{otherwise,} \end{cases}$$

and $0 \leq \alpha \leq 1$ is a parameter used to adapt the weight of vertex and edge correspondences in $f_2(h)$. The fitness function can be easily understood by having a look to the two main terms: the first one measures the correspondence between vertices of the model and data graphs, and the second the correspondence between edges of both graphs. The term $\theta_N(a_D,a_M)$ is used to check whether two vertices are matched in the solution or not. If the two vertices are very similar ($c_N(a_D,a_M) \approx 1$) and the matching is not present in the solution that is being evaluated ($\theta_N(a_D,a_M) = 0$) then we will obtain for the matching a low value for this part of the represented solution ($[1 - |\theta_N(a_D,a_M) - c_N(a_D,a_M)|] \approx 0$). On the other hand, if this correspondence is present in the solution ($\theta_N(a_D,a_M) = 1$), we obtain a good value for this particular matching of the two vertices ($[1 - |\theta_N(a_D,a_M) - c_N(a_D,a_M)|] \approx 1$). The second term follows an analogous approach analyzing all the edges of both graphs.

3.4.4 f_3: not considering edges of vertices in G_D matched to the same vertex in G_M

Finally, a last improvement to the fitness function can be done by not taking into account edges of G_D between vertices that have been matched to the same vertex in G_M since the attributes of these edges are not meaningful and do not have to be compared to any edge of G_M. This idea to improve fitness functions is proposed and tested experimentally in [Boeres, 2002]. For that purpose, denote by $E_D^* = \{e_D = (a_D^i,a_D^j) \in E_D \mid h(a_D^i) \neq h(a_D^j)\} \subset E_D$ the set of edges that will only be considered for computing the global similarity. As a result, a third global similarity function is proposed:

$$f_3(h) = \frac{\alpha}{|V_D||V_M|} \sum_{(a_D,a_M)\in V_D\times V_M} [1 - |\theta_N(a_D,a_M) - c_N(a_D,a_M)|] +$$

$$\frac{1-\alpha}{|E_D^*||E_M|} \sum_{(a_D^i,a_D^j)\in E_D^*,(a_M^k,a_M^l)\in E_M} [1-|\theta_E((a_D^i,a_D^j),(a_M^k,a_M^l))-c_E((a_D^i,a_D^j),(a_M^k,a_M^l))|], \tag{3.3}$$

where α, $\theta_N(a_D, a_M)$ and $\theta_E((a_D^i, a_D^j), (a_M^k, a_M^l))$ are defined as in the previous section.

3.4.5 f_4: function based on the divergence between distributions

We propose the use of attributes based on probability theory, as these have not yet been applied to our graph matching problems. These require again to construct a completely different model and to represent it in a graph format using probabilistic parameters that will allow comparison of similarities with a data graph. Later on this chapter, two new types of fitness functions based on probability theory will be introduced.

The attributes that we will use can be classified in vertex attributes (unary attributes) and edge attributes (binary attributes) in the same way as described in Section 3.4.1.

The unary attribute that we consider in our examples for each of the vertices is the grey level distribution of the region represented by the vertex. Apart from this, there are also other three vertex attributes that will be used for other purposes but for measuring the similarity directly. These attributes are:

- size of the region (in pixels),
- coordinates x and y of the center of gravity of the region,
- super-region number in which the region is located, which is an attribute given by the tracking procedure used to find approximate landmarks of interesting features.

Analogously, the binary (edge) attributes that we will consider are:

- distance,
- relative position.

However, edge attributes will be represented in the form of vectors considering the center of gravity of the destination, exactly as done in [Cesar et al., 2002b]. Vectors will be computed from all the points of the origin to that destination-center of gravity, and we will calculate the mean and variance of the x and y components of all these vectors. Using this type of representation will implicitly record information of the distance and relative position.

Foundations of the new fitness functions based on probability theory

In order to consider attributes using probability, we will assume that each attribute of a region, either if it is a vertex or edge attribute, is modelled by a random variable that follows a normal distribution. For computing the similarity, we propose that both the model and data graphs are complete ones[3].

A vertex attribute such as the grey level can usually be fitted by a normal distribution (although this fact depends on the type of the input image). However, attributes such

[3]The generic definition of a complete graph is a graph $G = (V, E)$ such that $\forall a, a' \in V \ \exists e = (a, a') \in E$, and usually the condition $a \neq a'$ is also assumed. Therefore the edges from a vertex to itself are not considered. Here we assume that a complete graph does not contain such edges, with each vertex in a complete graph containing $|V| - 1$ edges.

as distance, relative position, and in general, others more dependent on the shape of the region, could lead to problems when searching for a good solution. The modelization of these attribute values by normal distributions is more suitable when regions have a rounded shape.

However, when evaluating a solution, regions matched to a same model vertex can be considered as *fused* in a sense, and this fused region will often not have a rounded shape. As a result, and in order to allow a satisfactory approximation to normal distributions, it is very important to analyze carefully how to model edge attributes in both the model and data graphs.

The fact of using a representation based on vectorial components of the edge attributes is a solution to this problem. Both the distance and relative position attributes are implicit in a vectorial representation, as the distance corresponds to the modulus and the relative position to the angle of the vector.

In previous attributes based on fuzzy set theory that can be found in the literature such as [Perchant, 2000], these edge attributes are modelled using the necessity, and possibility (i.e. roughly equivalent to the minimum and the maximum). In our case, we will assume that attributes can be modelled by normal distributions, and therefore we will use the mean and variance.

The edges will be assumed to always finish in the center of gravity of the destination region. Therefore, the center of gravity of every region in both the model and data images will need to be computed. Each edge will be stored using their x and y components. As both the model and data graphs are complete ones, we will have $|V_i| - 1$ $i = M, D$ edges for each region respectively, where we will use two attributes for each.

As a result, for each vertex in both graphs we will consider the following attributes:

1. One unary attribute: the grey level distribution. As this attribute is assumed to follow a normal distribution, in a region a it will be represented as $\mathcal{N}_{a;g}(\mu_{a;g}, \sigma^2_{a;g})$, where μ is the mean and σ^2 is the variance.

2. Two edge attributes for each edge starting in the region a, and arriving at region k: $\mathcal{N}_{a,k;x}(\mu_{a,k;x}, \sigma^2_{a,k;x})$ and $\mathcal{N}_{a,k;y}(\mu_{a,k;y}, \sigma^2_{a,k;y})$. As in a complete graph with n vertices we have $n-1$ edges from each vertex, we have a total of $2n-2$ edge attributes for each vertex. Note that these definitions assume implicitly that the x and y components are independent, and therefore the variance-covariance matrix will be of the form $\begin{pmatrix} \sigma^2_{a,k;x} & 0 \\ 0 & \sigma^2_{a,k;y} \end{pmatrix}$. Another possibility is to consider a binary attribute by means of a 2-dimensional normal distribution, in which case the variance-covariance matrix would not contain zeroes.

Computing the attributes of each region

As attributes of the regions will be modelled by normal distributions, each attribute will be characterized by its mean and variance values. Therefore, in an image with n regions, a region a will be represented as an d-dimensional normal distribution, $\mathcal{N}_a(\boldsymbol{\mu}_a, \Sigma_a)$ (one vertex

attribute and $2n-2$ edge ones, which makes $d = 2n-1$):

$$\begin{aligned}\mathcal{N}_a(\boldsymbol{\mu}_a, \Sigma_a) = \quad (\quad & \mathcal{N}_{a;g}(\mu_{a;g}, \sigma^2_{a;g}), \\ & \mathcal{N}_{a,1;x}(\mu_{a,1;x}, \sigma^2_{a,1;x}), \mathcal{N}_{a,1;y}(\mu_{a,1;y}, \sigma^2_{a,1;y}), \quad \ldots, \\ & \mathcal{N}_{a,a-1;x}(\mu_{a,a-1;x}, \sigma^2_{a,a-1;x}), \mathcal{N}_{a,a-1;y}(\mu_{a,a-1;y}, \sigma^2_{a,a-1;y}), \\ & \mathcal{N}_{a,a+1;x}(\mu_{a,a+1;x}, \sigma^2_{a,a+1;x}), \mathcal{N}_{a,a+1;y}(\mu_{a,a+1;y}, \sigma^2_{a,a+1;y}), \quad \ldots, \\ & \mathcal{N}_{a,n;x}(\mu_{a,n;x}, \sigma^2_{a,n;x}), \mathcal{N}_{a,n;y}(\mu_{a,n;y}, \sigma^2_{a,n;y}) \quad) \end{aligned} \tag{3.4}$$

This representation will be used for regions in the model and data graphs. It is important to take into account that the number of vertices in the model graph ($|V_M|$) is smaller than the number of vertices in the data graph ($|V_D|$). Each vertex in the model graph has $2|V_M|-1$ attributes and each vertex in the data graph has $2|V_D|-1$ attributes.

Cases in which mixtures of normal distributions are required

Mixtures of normal distributions are a way of representing a new distribution composed by a weighted sum of many normal distributions. Mixtures are used when the composition of two normal distributions cannot be approximated by a single new normal distribution. The later case occurs typically when the means of the different normal distributions are very far from each another, and therefore a weight is given to each of the distributions in order to express the contribution of each of them to the global combined distribution.

The weights of each original normal distribution are computed using algorithms such as the EM [Dempster et al., 1977]. This procedure is obviously very CPU expensive. In addition, comparing a mixture of normal distributions and a normal distribution (i.e. when comparing the solution proposed by an individual and a region in the model) is a lot more complicated than comparing two normal distributions in terms of number of operations required.

In our particular example, as proved on the following section, the fact of representing edge attributes as the two components (x and y) of a vector will allow us to approximate a combination of normal distributions to a new normal distribution, without having the need to use mixtures.

Attributes of the fused regions modelled by normal distributions

Let us consider as an example the case illustrated in Figure 3.4. This figure shows on the left two regions (A and B) of a model image, and on the right we have a typical result of an over-segmentation procedure, where the regions detected on a data image have lead to a data graph with more subregions for each model region. Therefore each correspondence of the regions in the model appears usually subdivided in the data image (the region A in the model has been divided in three subregions after the automatic over-segmentation procedure, regions 1, 2, and 3, and region B has been subdivided in two, regions 11 and 12).

In order to evaluate a solution, we would require to *fuse* regions 1, 2, and 3 in the data image, and then compare the similarity to the model region A. Similarly, data image regions 11 and 12 should be fused and afterwards compared to the model region B. The fusion of attributes of vertices in G_D such as the grey level (i.e. the one selected for our problems) represented as a normal distribution can be approximated to a new normal distribution for evident reasons, and it does not require further discussion.

Figure 3.4: Illustration of two regions in the model graph (left), and the typical result after following an over-segmentation process on an image to be recognized (right). This figure also illustrates the centers of gravity of each of the regions. These will be used as a destination point representative of the whole vector from any point of the origin to the destination.

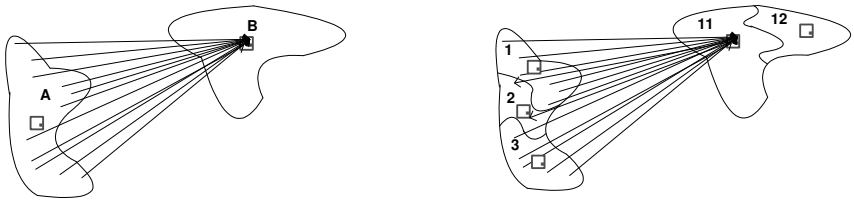


Figure 3.5: Example showing how the edge attributes are computed in both the model and data graphs.

On the other hand, as already explained, the edge attributes that are of our interest for comparing the similarity between regions are the distance and relative position. Given a homomorphism h, and in order to evaluate it, before starting with any comparison the regions in the data image that are matched to a same vertex in the model in h have to be fused, as they are supposed to be part of a same region. Figure 3.4 shows which centers of gravity that will be used for each region, and Figure 3.5 is an illustrative example on how this computation is done.

From the previous figures it is obvious that in the data graph we have edge attributes that are computed between subregions, but the equivalent regions in the model have a different center of gravity, and therefore the solution that can be obtained is also different. The proof that remains to be performed is illustrated in Figure 3.6, which essentially consists in showing that it is possible to compute the edge attributes between the model regions A and B starting from the edge attributes in the data graph of the data regions 1, 2, 3, 11 and 12. Next, we prove that this computation is possible and how it can be performed. In addition, we show that the *fusion* of edge attributes in the regions 1 2 and 3 into one, as well as such of the regions 11 and 12 into another, can effectively be modelled by a normal distribution using the vector components as representative data.

We will take as an example for this proof the regions and their centers of gravity of Figure 3.4. Let S_B be the size (in pixels) of region B in the model. Analogously, let S_{11} and S_{12} be the sizes of regions 11 and 12 respectively in the data graphs. As these two regions in the data graph are the equivalent as region B in the model graph, we assume that $S_B \equiv S_{11}+S_{12}$. Using geometrical properties it can easily be proved the x and y coordinates of the center of gravity of model region B –$B^B = \left(B_x^B, B_y^B\right)$– can be computed from the

Figure 3.6: Summary of the problem of obtaining the edge attributes between regions A and B in the model graph, knowing the values of the edge attributes in the data graph.

centers of gravity of data image regions 11 and 12 –$B^{11} = \left(B_x^{11}, B_y^{11}\right)$ and $B^{12} = \left(B_x^{12}, B_y^{12}\right)$.

This result can be easily generalized for a model region R which is divided in m smaller subregions $r_1, r_2, \ldots, r_m$ in the data image with respective sizes in pixels $S_{r_1}, S_{r_2}, \ldots, S_{r_m}$. Then, it can be proved that the fused center of gravity of region R can be computed as

$$B^R = \left(B_x^R, B_y^R\right) \equiv \left(\frac{\sum_{i=1}^{m} B_x^{r_i} \cdot S_{r_i}}{\sum_{i=1}^{m} S_{r_i}}, \frac{\sum_{i=1}^{m} B_y^{r_i} \cdot S_{r_i}}{\sum_{i=1}^{m} S_{r_i}} \right) \tag{3.5}$$

Similarly, we can also prove that the edge attributes of any model region can also be computed starting from the edge attributes of the equivalent data image regions. Given a destination model region R divided in m smaller subregions $r_1, r_2, \ldots, r_m$ in the data graph with respective sizes $S_{r_1}, S_{r_2}, \ldots, S_{r_m}$, knowing that the fused center of gravity is the one described previously, and that $S_B \approx S^* = \sum_{i=1}^{m} S_{r_i}$, it can be easily shown that from any point $p_1 = (x_1, y_1)$ within the origin regions the following is satisfied:

$$\overrightarrow{v_{p_1 B^B}} \approx \sum_{i=1}^{m} \frac{S_{r_i}}{S^*} \cdot \overrightarrow{v_{p_1 B^{r_i}}} \tag{3.6}$$

This results is a linear combination of normal distributions. As such a combination of normal distributions is also a normal distribution, then we have proved that the when fusing the edge attributes of the $r_1, r_2, \ldots, r_m$ regions leads also to a region which edge attributes follow a normal distribution.

Evaluating a solution using the new approach

If individual regions of the data graph are to be compared to regions in the model graph, we could use one of these methods:

- The distance –or divergence– between both distributions could be used for comparison.
- Another possibility is to use the likelihood function to evaluate the similarity between regions.

We tested the use of both methods and compared them. Next, all the explanations and further formalization of these two methods are presented.

This fourth proposal of fitness function is based on the Kullback-Leibler divergence [Kullback and Leibler, 1951], which measures the difference between two different distributions. The Kullback-Leibler divergence is defined for any two types of distributions, although it can be written in a simple form if the two distributions to be compared are normal ones.

As explained in previous section, we will assume that all the attributes of all the regions in both the model and data graph follow normal distributions, and this assumption will help us to perform simpler operations to compute the divergence.

However, it is compulsory for this divergence that the dimensions of both normal distributions to be compared are the same. In fact, each of the regions of both the atlas and data graphs will be represented as a $(2n-1)$-dimensional normal distribution, where this dimension is $(2|V_M|-1)$ and $(2|V_D|-1)$ for each vertex of the model or data graphs respectively. Therefore, given a homomorphism, the question is how to find a way of comparing model and data vertices taking into account these restrictions for comparison. This comparison will be used as a similarity value.

In essence, when evaluating a solution given by a homomorphism h, each of the attributes of the regions in the data graph that are matched to the same vertex in the homomorphism (i.e. regions $a_D^i, a_D^j, \ldots, a_D^z \in V_D | h(a_D^i) = h(a_D^j) = \cdots = h(a_D^z) = a_M^r \in V_M$) have to be combined in a way that the combination of the normal distributions representing each attribute is approximated to a new normal distribution. This combination of regions will be done by *fusing* the regions which are labelled with the same model graph, in other words, the k^{th} fused region a_k^F is defined in the following way: $a_k^F = \{a_D^l \in V_D, k \in V_M | h(a_D^l) = a_M^k\}$.

It is important to note that after fusion of the data image regions and their attributes the result is a new region with $2|V_M|-1$ attributes.

After fusing all the data image regions to their corresponding *fused* region a_i^F $i = 1, \ldots, |V_M|$, their attributes are also combined, and this results in $2|V_M|-1$ attributes. Therefore, any fused region a^F can be represented as a new d-dimensional normal distribution $\mathcal{N}^F(\boldsymbol{\mu}^F, \Sigma^F)$ with $d = 2|V_M|-1$ (since there are 1 vertex attribute and $2|V_M|-2$ edge attributes) following the definition of $\mathcal{N}_a(\boldsymbol{\mu}_a, \Sigma_a)$ given in Equation 3.4 for any region a.

Afterwards, in order to compare how similar are the regions of the model and those of the data image we will only need to compare the d-dimensional normal distribution of the fused region a_k^F and the d-dimensional normal distribution of the model region a_M^k, with $d = (2|V_M|-1)$ in both graphs and $k = 1, \ldots, |V_M|$.

The main requirement of this method to be applied is to prove that the vertex and edge attributes of the fused regions can be satisfactorily approximated to normal distributions, knowing that the attributes of each of the vertices in V_D follow a normal distribution. This proof was shown in Section 3.4.5.

The Kullback-Leibler divergence. Kullback and Leibler introduced a divergence measure that is known as the Kullback-Leibler divergence [Kullback and Leibler, 1951]. The idea behind a divergence is to have a measure to quantify the information quantity given by data to discriminate between one or another probability distributions. In the discrete domain, having a finite set of values $S = \{a_1, a_2, \ldots, a_n\}$, and if the P and Q probability distributions are given by $P(a_i) = p_i$ and $Q(a_i) = q_i$ $i = 1, 2, \ldots, n$, then the Kullback-Leibler divergence is expressed by

$$D_{K-L}(P, Q) = \sum_{i=1}^{n} p_i \log \frac{p_i}{q_i} \tag{3.7}$$

Regarding the continuous case, given two density functions $f(x)$ and $g(x)$ for the space χ, we have that $\frac{dP}{dQ}(x) = \frac{f(x)}{g(x)}$, and therefore the Kullback-Leibler divergence is given by

$$D_{K-L}(P, Q) = \int_{\chi} f(x) \log \frac{f(x)}{g(x)} dx \tag{3.8}$$

In the case of having any two normal distributions of dimension d denoted by $\mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1)$ and $\mathcal{N}(\boldsymbol{\mu}_2, \Sigma_2)$ respectively, the Kullback-Leibler divergence is computed in [Kullback and Leibler, 1951] as

$$\begin{aligned} D_{K-L}\left(\mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1), \mathcal{N}(\boldsymbol{\mu}_2, \Sigma_2)\right) &= \frac{1}{2}\left[(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^t \Sigma_2^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)\right] \\ &+ \frac{1}{2}\left(trace\left(\Sigma_2^{-1}\Sigma_1 - I\right) + \log\frac{|\Sigma_2|}{|\Sigma_1|}\right) \end{aligned} \tag{3.9}$$

where I is the unary matrix. This expression is further simplified when the elements of both d-dimensional normal distributions are independent. In the latter case, the matrices $\Sigma_j \;\; j = 1, 2$ of dimensions $d \times d$ would be of the form $\begin{pmatrix} \sigma_{j,1}^2 & 0 & 0 & \dots & 0 \\ 0 & \sigma_{j,2}^2 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \sigma_{j,d}^2 \end{pmatrix}$. It can be proved that under these conditions the Kullback-Leibler distance is given by:

$$\begin{aligned} D_{K-L}(\mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1), \mathcal{N}(\boldsymbol{\mu}_2, \Sigma_2)) &= \frac{1}{2}\left[\sum_{i=1}^{d} \frac{(\mu_1^i - \mu_2^i)^2}{\sigma_2^i}\right] \\ &+ \frac{1}{2}\left(\sum_{i=1}^{d}\left(\frac{\sigma_1^i}{\sigma_2^i} - 1\right) + \log\left(\prod_{i=1}^{d}\frac{\sigma_2^i}{\sigma_1^i}\right)\right) \end{aligned} \tag{3.10}$$

In our case, given a solution, for each of the model regions a_M^k two d-dimensional normal distributions with $d = 2|V_M| - 1$ will be compared: The normal distribution of the model vertex itself and the one obtained after fusing all the data regions associated to the k^{th} fused region a_k^F. Note also that the Kullback-Leibler divergence is not symmetrical, and that $D_{K-L}(P, Q) \neq D_{K-L}(Q, P)$. Therefore, as a reference has to be defined, we decided to choose as the reference the data graph vertex that is compared to each of the vertices of the model graph (note that the fitness function is also defined as $f_4(h) : V_D \rightarrow V_M$). Therefore, we consider the $D_{K-L}\left(\mathcal{N}\left(\boldsymbol{\mu}_{a_M^k}, \Sigma_{a_M^k}\right), \mathcal{N}\left(\boldsymbol{\mu}_{a_k^F}, \Sigma_{a_k^F}\right)\right)$ values for all the model regions.

Definition of the fitness function. Taking all the previous explanations into account, we propose a first fitness function definition based on the Kullback-Leibler distance. In this definition, denoted as f_4, when computing a solution proposed, we first *fuse* the regions in the data graph, and then we compare each of the vertex and edge attributes of the fused region to the vertex of the matched region. Taking the illustrative example of the previous section, the similarity between region A in the model and the combination of regions 1, 2, and 3 in the data graph will be computed using their respective d-dimensional normal distributions $\mathcal{N}(\boldsymbol{\mu}_A, \Sigma_A), \mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1), \mathcal{N}(\boldsymbol{\mu}_2, \Sigma_2), \mathcal{N}(\boldsymbol{\mu}_3, \Sigma_3)$, with the sizes in pixels of each of the regions being S_A, S_1, S_2, and S_3 respectively. For this, the last three distributions have to be combined. Knowing that $\Sigma_j \;\; j = 1, 2, 3$ would be of the form $\begin{pmatrix} \sigma_{j,1}^2 & 0 & 0 & \dots & 0 \\ 0 & \sigma_{j,2}^2 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \sigma_{j,d}^2 \end{pmatrix}$, we will denote the resultant distribution by $\mathcal{N}(\boldsymbol{\mu}_{123}, \Sigma_{123})$, having:

$$\boldsymbol{\mu}_{123} = \frac{S_1 \cdot \boldsymbol{\mu}_1 + S_2 \cdot \boldsymbol{\mu}_2 + S_3 \cdot \boldsymbol{\mu}_3}{S_1 + S_2 + S_3}$$

$$\Sigma_{123} = \begin{pmatrix} \tau^2_{123,1} & 0 & 0 & \dots & 0 \\ 0 & \tau^2_{123,2} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \tau^2_{123,d} \end{pmatrix} \tag{3.11}$$

where $\tau^2_{123,i} = \frac{S_1 \cdot \sigma^2_{1,i} + S_1 \cdot (\mu_{123,i} - \mu_{1,i})^2 + S_2 \cdot \sigma^2_{2,i} + S_2 \cdot (\mu_{123,i} - \mu_{2,i})^2 + S_3 \cdot \sigma^2_{3,i} + S_3 \cdot (\mu_{123,i} - \mu_{3,i})^2}{S_1 + S_2 + S_3}$ $i = 1, \dots, d$, and $\boldsymbol{\mu}_{123} = (\mu_{123,1}, \mu_{123,2}, \dots, \mu_{123,d})$. Note that the dimension of both d-dimensional normal distributions is $d = 2|V_M| - 1$.

The component of this region in the proposed fitness function will be defined as follows:

$$\begin{aligned} f_4(h)_{a_M} &= D_{K-L}(\mathcal{N}(\boldsymbol{\mu}_{a_M}, \Sigma_{a_M}), \mathcal{N}(\boldsymbol{\mu}_{123}, \Sigma_{123})) \\ &= \frac{1}{2}\left[\sum_{i=1}^{d} \frac{(\mu_{a_M,i} - \mu_{123,i})^2}{\sigma^2_{123,i}}\right] + \frac{1}{2}\left(\sum_{i=1}^{d}\left(\frac{\sigma^2_{a_M,i}}{\sigma^2_{123,i}} - 1\right) + \log\left(\prod_{i=1}^{d} \frac{\sigma^2_{123,i}}{\sigma^2_{a_M,i}}\right)\right) \end{aligned} \tag{3.12}$$

And finally, the global fitness function will be computed as

$$f_4(h) = \sum_{a_M \in V_M} f_4(h)_{a_M} \tag{3.13}$$

Generalizing this fitness function for a region a^k_M of the model that is matched to the equivalent m smaller subregions $r_1, r_2, \dots, r_m$ in the data graph (i.e. $h(r_1) = h(r_2) = \dots = h(r_n) = R$) with respectively sizes $S_{r_1}, S_{r_2}, \dots, S_{r_m}$, we have firstly that the combination of all the subregions in the data graph is given by the d-dimensional normal distribution $\mathcal{N}(\boldsymbol{\mu}_{fused_{a^k_M}}, \Sigma_{fused_{a^k_M}})$, knowing that

$$\boldsymbol{\mu}_{fused_{a^k_M}} = \frac{\sum_{i=1}^{m} S_{r_i} \cdot \boldsymbol{\mu}_{r_i}}{\sum_{i=1}^{m} S_{r_i}}$$

$$\Sigma^2_{fused_{a^k_M}} = \begin{pmatrix} \tau^2_{fused_{a^k_M},1} & 0 & 0 & \dots & 0 \\ 0 & \tau^2_{fused_{a^k_M},2} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \tau^2_{fused_{a^k_M},d} \end{pmatrix} \tag{3.14}$$

where

$$\tau^2_{fused_{a^k_M},i} = \frac{\sum_{j=1}^{d}\left(S_{r_j} \cdot \sigma^2_{r_j,i} + S_{r_j} \cdot \left(\mu_{fused_{a^k_M},j} - \mu_{r_i,j}\right)^2\right)}{\sum_{j=1}^{d} S_{r_j}}$$

for $i = 1, \dots, d$, and $\boldsymbol{\mu}_{fused_{a^k_M}} = \left(\mu_{fused_{a^k_M},1}, \mu_{fused_{a^k_M},2}, \dots, \mu_{fused_{a^k_M},d}\right)$. Finally, the fitness function is defined as

$$f_4(h) = \sum_{a^k_M \in V_M} f_4(h)_{a^k_M} = \sum_{a^k_M \in V_M} D_{K-L}\left(\mathcal{N}\left(\boldsymbol{\mu}_{a^k_M}, \Sigma_{a^k_M}\right), \mathcal{N}\left(\boldsymbol{\mu}_{fused_{a^k_M}}, \Sigma_{fused_{a^k_M}}\right)\right) \tag{3.15}$$

As this fitness function expresses the divergence of probability distributions, in this case the graph matching algorithm is committed to return the individual that minimizes this expression, and not to maximize as in the previous three fitness functions.

3.4.6 f_5: function based on likelihood

This fifth proposal of fitness function is based on the likelihood of the data given the model. In this case, the regions in the model are also represented as a d-dimensional normal distribution, for a dimension for each of the vertex and edge attributes, as well as for the previous fitness function $f_4(h)$: the mean and variance of each of the attributes for all the regions of the model are computed as a previous step, and these are used for computing the likelihood. Regarding the data graph, a similar representation is assumed, and also the data information required to compute the likelihood is stored. This information is later used to compute each of the homomorphisms.

For this approach, we select at random a set of N pixels from each of the regions in the data graph. We decided to choose this number N as a constant for all type of regions independently of their size, although when fusing the regions in the data image only a number of pixels proportional to the size of the fused region are taken later into account. The number N of pixels selected has to be big enough to ensure representation of each of the regions in the data image. We denote the representation of pixels for the k^{th} region of the data image by x_1^k, x_2^k, ..., x_N^k , with $k = 1, \ldots, |V_D|$. The total size of this sample is $M = N|V_D|$.

The next step is to compute the attribute values of the selected pixels in each region. In the case of the vertex attributes, we consider again the grey level and the texture value. For the edge attributes, we compute again the binary attributes using as a reference the pointed vertex the center of gravity of the regions that have to be fused in the homomorphism as explained in the previous sections. As a result, the number of attributes to be recorded by each sample pixel is $2|V_D| - 1$. The attributes for the i^{th} pixel of the k^{th} region is denoted by $\boldsymbol{y}_i^k = \left(y_{i,1}^k, y_{i,2}^k, \ldots y_{i,2|V_D|-1}^k\right)$.

Computing the likelihood of the data regarding each attribute. Given the probability distribution of a single attribute j of a particular region k of the model, $\mathcal{N}(\mu_{k,j}, \sigma_{k,j}^2)$, the likelihood is a measurement with N possible values $y_{1,j}^k, y_{2,j}^k, \ldots, y_{N,j}^k$ defined as:

$$\begin{aligned} L_j(y_{1,j}^k, y_{2,j}^k, \ldots, y_{N,j}^k; \mathcal{N}(\mu_{k,j}, \sigma_{k,j}^2)) &= \prod_{i=1}^{N} \left[\frac{1}{\sqrt{2\pi}\ \sigma_{k,j}} \exp\left(-\frac{1}{2\sigma_{k,j}^2} (y_{i,j}^k - \mu_{k,j})^2 \right) \right] \\ &= \left(\frac{1}{\sqrt{2\pi}\ \sigma_{k,j}} \right)^N \exp\left(-\frac{1}{2\sigma_{k,j}^2} \sum_{i=1}^{N} (y_{i,j}^k - \mu_{k,j})^2 \right) \end{aligned} \tag{3.16}$$

In order to evaluate a given homomorphism h, we proceed in a similar way as in the previous sections: firstly, the regions in the data image following the labels given by the homomorphism h are identified, and $|V_M|$ fused regions are obtained. Secondly, the centers of gravity of these fused regions are computed, and the vertex and edge attributes are recomputed. As a result, each of the fused regions contains 1 vertex attribute and $2|V_M| - 2$ edge attributes. Thirdly, a set of N pixels are selected from each of the fused regions. Fourthly, the likelihood of the d-dimensional probability distribution representing a model region $a_M^k \in V_M$, with $d = 2|V_M| - 1$, is computed as:

$$L\left(\boldsymbol{y}_1^k, \boldsymbol{y}_2^k, \ldots, \boldsymbol{y}_N^k; \mathcal{N}\left(\boldsymbol{\mu}_{a_M^k}, \Sigma_{a_M^k}\right)\right) = \prod_{j=1}^{2|V_M|-1} L_j(y_{1,j}^k, y_{2,j}^k, \ldots, y_{N,j}^k; \mathcal{N}(\mu_{k,j}, \sigma_{k,j})) \tag{3.17}$$

where L_j is defined in Equation 3.16. The result obtained in Equation 3.17 is used in $f_5(h)$ to compute the similarity between region k of the model and the k^{th} fused region in the data image.

Evaluation of a homomorphism: definition of the fitness function. Following the definitions of the previous section, a homomorphism h can be evaluated by means of the likelihood of the models for every vertex in the model graph regarding the pixels sampled in the data image. For that, the similarities of all the vertices of the model graph with such of the corresponding fused region in the data image are considered to define the global similarity –GS– of a given homomorphism h as follows:

$$\begin{aligned} GS &= \prod_{k=1}^{|V_M|} L\left(\boldsymbol{y}_1^k, \boldsymbol{y}_2^k, \ldots, \boldsymbol{y}_N^k; \mathcal{N}\left(\boldsymbol{\mu}_{a_M^k}, \Sigma_{a_M^k}\right)\right) \\ &= \prod_{k=1}^{|V_M|} \prod_{j=1}^{2|V_M|-1} L_j(y_{1,j}^k, y_{2,j}^k, \ldots, y_{N,j}^k; \mathcal{N}(\mu_{a_M^k,j}, \sigma_{a_M^k,j})) \\ &= \prod_{k=1}^{|V_M|} \prod_{j=1}^{2|V_M|-1} \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\ \sigma_{a_M^k,j}} \exp\left(-\frac{1}{2\sigma^2_{a_M^k,j}}(y_{i,j}^k - \mu_{a_M^k,j})^2\right) \\ &= \left[\prod_{k=1}^{|V_M|} \prod_{j=1}^{2|V_M|-1} \left(\frac{1}{\sqrt{2\pi}\ \sigma_{a_M^k,j}}\right)^N\right] \exp\left(\sum_{k=1}^{|V_M|} \sum_{j=1}^{2|V_M|-1} \sum_{i=1}^{N} \left(-\frac{(y_{i,j}^k - \mu_{a_M^k,j})^2}{2\sigma^2_{a_M^k,j}}\right)\right) \end{aligned} \tag{3.18}$$

where $\mathcal{N}\left(\boldsymbol{\mu}_{a_M^k}, \Sigma_{a_M^k}\right)$ $k = 1, \ldots, |V_M|$ are the ones defined in the previous sections, and all the $\boldsymbol{y}_1^k, \boldsymbol{y}_2^k, \ldots, \boldsymbol{y}_N^k$ are the attribute values of the N pixels chosen at random from the fused region k, which is formed by the $a_D \in V_D$ such that $h(a_D) = k$.

However, in order to define the fitness function $f_5(h)$ using this similarity definition, the expression in Equation 3.18 can be simplified, since a proportional fitness function is valid for our purpose. A simpler yet proportional definition of this equation is described below:

$$GS = C\ \exp\left(\sum_{k=1}^{|V_M|} \sum_{j=1}^{2|V_M|-1} \sum_{i=1}^{N} \left(-\frac{(y_{i,j}^k - \mu_{a_M^k,j})^2}{2\sigma^2_{a_M^k,j}}\right)\right) \propto \sum_{k=1}^{|V_M|} \sum_{j=1}^{2|V_M|-1} \sum_{i=1}^{N} \frac{(y_{i,j}^k - \mu_{a_M^k,j})^2}{-2\sigma^2_{a_M^k,j}} \tag{3.19}$$

where $C = \left[\prod_{k=1}^{|V_M|} \prod_{j=1}^{2|V_M|-1} \left(\frac{1}{\sqrt{2\pi}\ \sigma_{a_M^k,j}}\right)^N\right]$. As a result, the fitness function $f_5(h)$ can be defined as the simple expression

$$f_5(h) = \sum_{k=1}^{|V_M|} \sum_{j=1}^{2|V_M|-1} \sum_{i=1}^{N} \frac{(y_{i,j}^k - \mu_{a_M^k,j})^2}{-2\sigma^2_{a_M^k,j}} \tag{3.20}$$

In this case, as well as with $f_1(h)$, $f_2(h)$, and $f_3(h)$, the graph matching algorithm is committed to return the individual that maximizes this expression.

3.5 Conclusion

This chapter presents the graph matching problem as a combinatorial optimization one, analyzing the main aspects that have to be taken into account for that: the definition of an individual representation and its associated fitness function. Three different individual representations have been presented in the discrete domain, as well as five different fitness functions.

For the continuous domain, an individual representation based on permutations has been introduced. This representation allows to apply the fitness functions defined for the discrete domain for specific graph matching algorithms in the continuous domain.

It is important to note that all the definitions given in this chapter can be applied by any combinatorial optimization algorithm. In the next chapters algorithms such as estimation of distribution algorithms, genetic algorithms, and evolutionary strategies apply these fitness functions to search for the best homomorphism.

Chapter 4

Estimation of distribution algorithms

'That is what learning is. You suddenly understand something you've understood all your life, but in a new way. '

Doris Lessing

4.1 Introduction

Generally speaking, all the search strategy types can be classified either as complete or heuristic strategies. The difference between them is that complete strategies perform a systematic examination of all possible solutions of the search space whereas heuristic strategies only concentrate on a part of them following a known algorithm.

Heuristic strategies are also divided in two groups: deterministic and non-deterministic strategies [Pearl, 1984]. The main characteristic of deterministic strategies is that under the same conditions the same solution is always obtained. Examples of this type are forward, backward, stepwise, hill-climbing, threshold accepting, and other well known algorithms, and their main drawback is that they have the risk of getting stuck in local maximum values. Non-deterministic searches are able to escape from these local maxima by means of the randomness [Zhigljavsky, 1991] and, due to their stochasticity, different executions might lead to different solutions under the same conditions.

Some of the stochastic heuristic searches such as simulated annealing only store one solution at every iteration of the algorithm. The stochastic heuristic searches that store more than one solution every iteration (or every generation as each iteration is usually called in these cases) are grouped under the term of population-based heuristics, an example of which is evolutionary computation. In these heuristics, each of the solutions is called *individual*. The group of individuals (also known as *population*) evolves towards more promising areas of the search space while the algorithm carries on with the next generation. Examples of evolutionary computation are Genetic Algorithms (GAs) [Goldberg, 1989, Holland, 1975], evolutionary strategies (ESs) [Rechenberg, 1973], evolutionary programming [Fogel, 1962] and genetic programming [Koza, 1992]. See [Bäck, 1996] for a review on evolutionary algorithms.

The behavior of evolutionary computation algorithms such as GAs depends to a large extent on associated parameters like operators and probabilities of crossing and mutation,

size of the population, rate of generational reproduction, the number of generations, and so on. The researcher requires experience in the resolution and use of these algorithms in order to choose the suitable values for these parameters. Furthermore, the task of selecting the best choice of values for all these parameters has been suggested to constitute itself an additional optimization problem [Grefenstette, 1986]. Moreover, GAs show a poor performance in some problems[1] in which the existing operators of crossing and mutation do not guarantee that the building block hypothesis is preserved[2].

All these reasons have motivated the creation of a new type of algorithms classified under the name of Estimation of Distribution Algorithms (EDAs) [Larrañaga and Lozano, 2001, Mühlenbein and Paaß, 1996], trying to make easier to predict the movements of the populations in the search space as well as to avoid the need for so many parameters. These algorithms are also based on populations that evolve as the search progresses and, as well as genetic algorithms, they have a theoretical foundation on probability theory. In brief, EDAs are population-based search algorithms based on probabilistic modelling of promising solutions in combination with the simulation of the induced models to guide their search.

In EDAs the new population of individuals is generated without using neither crossover nor mutation operators. Instead, the new individuals are sampled starting from a probability distribution estimated from the database containing only selected individuals from the previous generation. At the same time, while in other heuristics from evolutionary computation the interrelations between the different variables representing the individuals are kept in mind implicitly (e.g. building block hypothesis), in EDAs the interrelations are expressed explicitly through the joint probability distribution associated with the individuals selected at each iteration. In fact, the task of estimating the joint probability distribution associated with the database of the selected individuals from the previous generation constitutes the hardest work to perform. In particular, the latter requires the adaptation of methods to learn models from data that have been developed by researchers in the domain of probabilistic graphical models.

The underlying idea of EDAs will be introduced firstly for the discrete domain, and then it will be reviewed for the continuous domain. As an illustrative example, we will consider the problem that arises in supervised classification known as *feature subset selection* (FSS) [Inza et al., 2000, 2001]. Given a file of cases with information on n predictive variables, $X_1, X_2, \ldots, X_n$, and the class variable C to which the case belongs, the problem consists in selecting a subset of variables that will induce a classifier with the highest predictive capacity in a test set. The cardinality of the search space for this problem is 2^n.

Figure 4.1 shows a generic schematic of EDA approaches, which follow essentially the following steps:

1. Firstly, the initial population D_0 of R individuals is generated. The generation of these R individuals is usually carried out by assuming a uniform distribution on each variable, and next each individual is evaluated.

2. Secondly, in order to make the $l-1^{th}$ population D_{l-1} evolve towards the next D_l one, a number N ($N < R$) of individuals are selected from D_{l-1} following a criterion. We denote by D_{l-1}^N the set of N selected individuals from generation $l-1$.

[1] Problems in which GAs behave worse than simpler search algorithms are known as *deceptive problems*, in which the GAs get usually stuck in local minima and return worse results.

[2] The building block hypothesis [Holland, 1975] states that GAs find solutions by first finding as many building blocks as possible, and then combining them together to give the highest fitness. Following this hypothesis, we can search more effectively by exploiting similarities in the solutions.

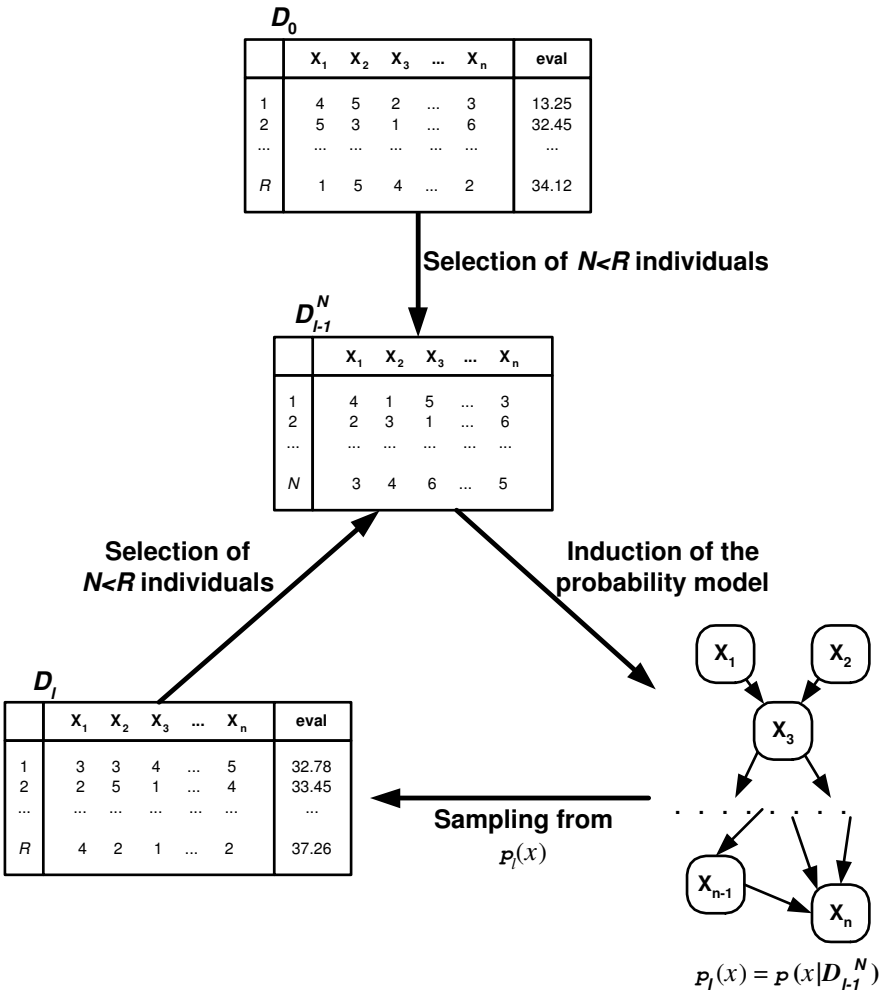


Figure 4.1: Illustration of EDA approaches in the optimization process.

3. Thirdly, the n–dimensional probabilistic model that better represents the interdependencies between the n variables is induced. This step is also known as the *learning* procedure, and it is the most crucial one, since representing appropriately the dependencies between the variables is essential for a proper evolution towards fitter individuals.

4. Finally, the new population D_l constituted by R new individuals is obtained by carrying out the simulation of the probability distribution learned in the previous step. Usually an elitist approach is followed, and therefore the best individual of population D_{l-1}^N is kept in D_l. In this latter case, a total of $R-1$ new individuals is created every generation instead of R.

Steps 2, 3 and 4 are repeated until a stopping condition is verified. Examples of stopping conditions are: achieving a fixed number of populations or a fixed number of different evaluated individuals, uniformity in the generated population, and the fact of not obtaining an individual with a better fitness value after a certain number of generations.

4.2 Probabilistic graphical models

4.2.1 Bayesian networks

This section will introduce the probabilistic graphical model paradigm [Howard and Matheson, 1981, Lauritzen, 1996, Pearl, 1988] that has extensively been used during the last decade as a popular representation for encoding uncertainty knowledge in expert systems [Heckerman and Wellman, 1995]. Only probabilistic graphical models of which a structural part is a directed acyclic graph will be considered, as these adapt properly to EDAs. The following is an adaptation of the paper [Heckerman and Geiger, 1995], and will be used to introduce Bayesian networks as a probabilistic graphical model suitable for its application in EDAs.

Let $\boldsymbol{X} = (X_1, \ldots, X_n)$ be a set of random variables, and let x_i be a value of X_i, the i^{th} component of $\boldsymbol{X}$. Let $\boldsymbol{y} = (x_i)_{X_i \in \boldsymbol{Y}}$ be a value of $\boldsymbol{Y} \subseteq \boldsymbol{X}$. Then, a probabilistic graphical model for $\boldsymbol{X}$ is a graphical factorization of the joint generalized probability density function, $\rho(\boldsymbol{X} = \boldsymbol{x})$ (or simply $\rho(\boldsymbol{x})$). The representation of this model is given by two components: a structure and a set of local generalized probability densities.

The structure S for $\boldsymbol{X}$ is a directed acyclic graph (DAG) that describes a set of conditional independences[3] [Dawid, 1979] about the variables on $\boldsymbol{X}$. $\boldsymbol{Pa}_i^S$ represents the set of parents –variables from which an arrow is coming out in S– of the variable X_i in the probabilistic graphical model which structure is given by S. The structure S for $\boldsymbol{X}$ assumes that X_i and its non descendants are independent given $\boldsymbol{Pa}_i^S$, $i = 2, \ldots, n$. Therefore, the factorization can be written as follows:

$$\rho(\boldsymbol{x}) = \rho(x_1, \ldots, x_n) = \prod_{i=1}^{n} \rho(x_i \mid \boldsymbol{pa}_i^S). \tag{4.1}$$

Furthermore, the local generalized probability densities associated with the probabilistic graphical model are precisely the ones appearing in Equation 4.1.

A representation of the models of the characteristics described above assumes that the local generalized probability densities depend on a finite set of parameters $\boldsymbol{\theta}_S \in \boldsymbol{\Theta}_S$, and as a result the previous equation can be rewritten as follows:

$$\rho(\boldsymbol{x} \mid \boldsymbol{\theta}_S) = \prod_{i=1}^{n} \rho(x_i \mid \boldsymbol{pa}_i^S, \boldsymbol{\theta}_i) \tag{4.2}$$

where $\boldsymbol{\theta}_S = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n)$.

After having defined both components of the probabilistic graphical model, and taking them into account, the model itself can be represented by $M = (S, \boldsymbol{\theta}_S)$.

In the particular case of every variable $X_i \in \boldsymbol{X}$ being discrete, the probabilistic graphical model is called *Bayesian network*. If the variable X_i has r_i possible values, $x_i^1, \ldots, x_i^{r_i}$, the local distribution, $p(x_i \mid \boldsymbol{pa}_i^{j,S}, \boldsymbol{\theta}_i)$ is an unrestricted discrete distribution:

$$p({x_i}^k \mid \boldsymbol{pa}_i^{j,S}, \boldsymbol{\theta}_i) = \theta_{x_i^k \mid \boldsymbol{pa}_i^j} \equiv \theta_{ijk} \tag{4.3}$$

where $\boldsymbol{pa}_i^{1,S}, \ldots, \boldsymbol{pa}_i^{q_i,S}$ denotes the values of $\boldsymbol{Pa}_i^S$, that is the set of parents of the variable X_i in the structure S; q_i is the number of different possible instantiations of the parent variables

[3] Given $\boldsymbol{Y}, \boldsymbol{Z}, \boldsymbol{W}$ three disjoints sets of variables, $\boldsymbol{Y}$ is said to be conditionally independent of $\boldsymbol{Z}$ given $\boldsymbol{W}$ when for any $\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{w}$ the condition $\rho(\boldsymbol{y} \mid \boldsymbol{z}, \boldsymbol{w}) = \rho(\boldsymbol{y} \mid \boldsymbol{w})$ is satisfied. If this is the case, then we will write $I(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{W})$.

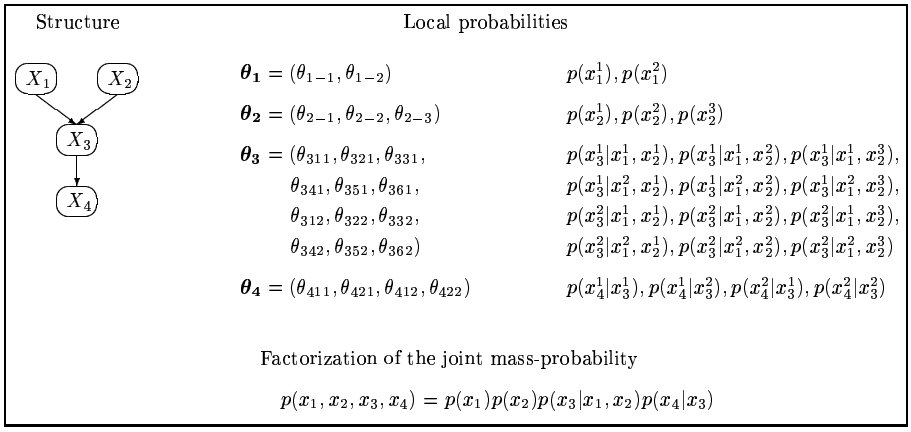


Figure 4.2: Structure, local probabilities and resulting factorization in a Bayesian network for four variables (with X_1, X_3 and X_4 having two possible values, and X_2 with three possible values).

of X_i. Thus, $q_i = \prod_{X_g \in \boldsymbol{Pa}_i^S} r_g$. The local parameters are given by $\boldsymbol{\theta}_i = ((\theta_{ijk})_{k=1}^{r_i})_{j=1}^{q_i})$. In other words, the parameter θ_{ijk} represents the conditional probability that variable X_i takes its k^{th} value, knowing that its parent variables have taken their j^{th} combination of values. We assume that every θ_{ijk} is greater than zero.

Figure 4.2 contains an example of the factorization of a particular Bayesian network with $\boldsymbol{X} = (X_1, X_2, X_3, X_4)$ and $r_2 = 3$, $r_i = 2 \;\; i = 1, 3, 4$. From this figure we can conclude that in order to define and build a Bayesian network the user needs to specify:

1.- a structure by means of a directed acyclic graph that reflects the set of conditional independencies among the variables,

2.- the prior probabilities for all root nodes (nodes with no predecessors), that is $p(x_i{}^k \mid \emptyset, \boldsymbol{\theta}_i)$ (or θ_{i-k}), and

3.- the conditional probabilities for all other nodes, given all possible combinations of their direct predecessors, $p(x_i{}^k \mid \boldsymbol{pa}_i^{j,S}, \boldsymbol{\theta}_i)$ (or θ_{ijk}).

4.2.2 Simulation in Bayesian networks

The simulation of Bayesian networks can be regarded as an alternative to exact propagation methods that were developed to reason with networks. This method creates a database with the probabilistic relations between the different variables previous to other procedures. In our particular case, the simulation of Bayesian networks is used merely as a tool to generate new individuals for the next population based on the structure learned previously.

Many approximations to the simulation of Bayesian networks have been developed in recent years. Examples of these are *the likelihood weighting method* developed independently in [Fung and Chang, 1990] and [Shachter and Peot, 1990], and later analyzed in [Shwe and Cooper, 1991], the *backward-forward sampling method* [Fung and del Favero, 1994], the *Markov sampling method* [Pearl, 1987], and the *systematic sampling method* [Bouckaert, 1994]. [Bouckaert et al., 1996] is a good comparison of the previous methods applied to different random Bayesian network models using the average time to execute the algorithm and the average error of the propagation as comparison criteria. Other approaches can be

PLS
    Find an ancestral ordering, $\boldsymbol{\pi}$, of the nodes in the Bayesian network
    For $j = 1, 2, \ldots, R$
        For $i = 1, 2, \ldots, n$
            $x_{\pi(i)} \leftarrow$ generate a value from $p(x_{\pi(i)} \mid \boldsymbol{pa}_i)$

Figure 4.3: Pseudocode for the Probabilistic Logic Sampling method.

also found in [Chavez and Cooper, 1990, Dagum and Horvitz, 1993, Hryceij, 1990, Jensen et al., 1993].

The method used in this report is the Probabilistic Logic Sampling (PLS) proposed in [Henrion, 1988]. Following this method, the instantiations are done one variable at a time in a forward way, that is, a variable is not sampled until all its parents have already been so. This requires previously to order all the variables from parents to children –any ordering of the variables satisfying such a property is known as ancestral ordering. We will denote $\boldsymbol{\pi} = (\pi(1), \ldots, \pi(n))$ an ancestral order compatible with the structure to be simulated. The concept of forward means that the variables are instantiated from parents to children. For any Bayesian network there is always at least one ancestral ordering since cycles are not allowed in Bayesian networks. Once the values of $\boldsymbol{pa}_i$ –the parent values of a variable X_i– have been assigned, its values are simulated using the distribution $p(x_{\pi(i)} \mid \boldsymbol{pa}_i)$. Figure 4.3 shows the pseudocode of the method.

4.2.3 Gaussian networks

In this section we introduce one example of the probabilistic graphical model paradigm that assumes the joint density function to be a multivariate Gaussian density [Whittaker, 1990].

An individual $\boldsymbol{x} = (x_1, \ldots, x_n)$ in the continuous domain consists of a continuous value in $\Re^n$. The local density function for the i^{th} variable X_i can be computed as the linear-regression model

$$f(x_i \mid \boldsymbol{pa}_i^S, \boldsymbol{\theta}_i) \equiv \mathcal{N}(x_i; m_i + \sum_{x_j \in \boldsymbol{pa}_i} b_{ji}(x_j - m_j), v_i) \tag{4.4}$$

where $\mathcal{N}(x_i; \mu_i, \sigma_i^2)$ is a univariate normal distribution with mean μ_i and variance $v_i = \sigma_i^2$ for the i^{th} variable.

Taking this definition into account, an arc missing from X_j to X_i implies $b_{ji} = 0$ in the former linear-regression model. The local parameters are given by $\boldsymbol{\theta}_i = (m_i, \boldsymbol{b}_i, v_i)$, where $\boldsymbol{b}_i = (b_{1i}, \ldots, b_{i-1i})^t$ is a column vector. A probabilistic graphical model built from these local density functions is known as a *Gaussian network* [Shachter and Kenley, 1989].

The components of the local parameters are as follows: m_i is the unconditional mean of X_i, v_i is the conditional variance of X_i given $\boldsymbol{Pa}_i$, and b_{ji} is a linear coefficient that measures the strength of the relationship between X_j and X_i. Figure 4.4 is an example of a Gaussian network in a 4–dimensional space.

In order to see how Gaussian networks and multivariate normal densities are related, the joint density function of the continuous n–dimensional variable $\boldsymbol{X}$ is by definition a multivariate normal distribution iff:

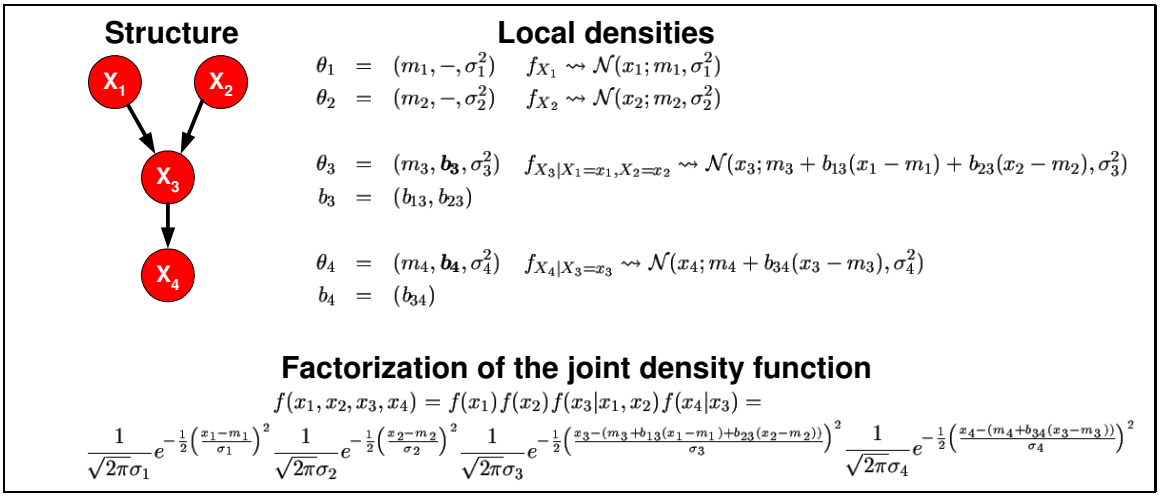


Figure 4.4: Structure, local densities and resulting factorization for a Gaussian network with four variables.

$$f(\boldsymbol{x}) \equiv \boldsymbol{\mathcal{N}}(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma) \equiv (2\pi)^{-\frac{n}{2}} |\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^t \Sigma^{-1} (\boldsymbol{x}-\boldsymbol{\mu})} \tag{4.5}$$

where $\boldsymbol{\mu}$ is the vector of means, Σ is covariance matrix $n \times n$, and $|\Sigma|$ denotes the determinant of Σ. The inverse of this matrix, $W = \Sigma^{-1}$, which elements are denoted by w_{ij}, is known as the precision matrix.

This density can also be written as a product of n conditional densities using the chain rule, namely

$$f(\boldsymbol{x}) = \prod_{i=1}^{n} f(x_i \mid x_1, \ldots, x_{i-1}) = \prod_{i=1}^{n} \mathcal{N}(x_i; \mu_i + \sum_{j=1}^{i-1} b_{ji}(x_j - \mu_j), v_i) \tag{4.6}$$

where μ_i is the unconditional mean of X_i, v_i is the variance of X_i given $X_1, \ldots, X_{i-1}$, and b_{ji} is a linear coefficient reflecting the strength of the relationship between variables X_j and X_i [de Groot, 1970]. This notation allows us to represent a multivariate normal distribution as a Gaussian network, where for any $b_{ji} \neq 0$ with $j < i$ this network will contain an arc from X_j to X_i.

Extending this idea it is also possible to generate a multivariate normal density starting from a Gaussian network. The unconditional means in both paradigms verify that $m_i = \mu_i$ for all $i = 1, \ldots, n$, [Shachter and Kenley, 1989] describe the general transformation procedure to build the precision matrix W of the normal distribution that the Gaussian network represents from its $\boldsymbol{v}$ and $\{b_{ji} \mid j < i\}$. This transformation can be done with the following recursive formula for $i > 0$, and $W(1) = \frac{1}{v_1}$:

$$W(i+1) = \begin{pmatrix} W(i) + \frac{\boldsymbol{b}_{i+1} \boldsymbol{b}_{i+1}^t}{v_{i+1}}, & -\frac{\boldsymbol{b}_{i+1}}{v_{i+1}} \\ -\frac{\boldsymbol{b}_{i+1}^t}{v_{i+1}}, & \frac{1}{v_{i+1}} \end{pmatrix} \tag{4.7}$$

where $W(i)$ denotes the $i \times i$ upper left submatrix, $\boldsymbol{b}_i$ is the column vector $(b_{1i}, \ldots, b_{i-1i})^t$ and $\boldsymbol{b}_i^t$ is its transposed vector.

For instance, taking into account the example in Figure 4.4 where $X_1 \equiv \mathcal{N}(x_1; m_1, v_1)$, $X_2 \equiv \mathcal{N}(x_2; m_2, v_2)$, $X_3 \equiv \mathcal{N}(x_3; m_3 + b_{13}(x_1 - m_1) + b_{23}(x_2 - m_2), v_3)$ and $X_4 \equiv \mathcal{N}(x_4; m_4 + b_{34}(x_3 - m_3), v_4)$, the procedure described above results in the following precision matrix W:

$$W = \begin{pmatrix} \frac{1}{v_1} + \frac{b_{13}^2}{v_3}, & \frac{b_{13}b_{23}}{v_3}, & -\frac{b_{13}}{v_3}, & 0 \\ \frac{b_{23}b_{13}}{v_3}, & \frac{1}{v_2} + \frac{b_{23}^2}{v_2}, & -\frac{b_{23}}{v_3}, & 0 \\ -\frac{b_{13}}{v_3}, & -\frac{b_{23}}{v_3}, & \frac{1}{v_3} + \frac{b_{34}^2}{v_4}, & -\frac{b_{34}}{v_4} \\ 0, & 0, & -\frac{b_{34}}{v_4}, & \frac{1}{v_4} \end{pmatrix}. \tag{4.8}$$

The representation of a multivariate normal distribution by means of a Gaussian network is more appropriated for model elicitation and understanding rather than the standard representation, as in the latter it is important to ensure that the assessed covariance matrix is positive–definite. In addition, the latter requires to check that the database D with N cases, $D = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, follows a multivariate normal distribution.

4.2.4 Simulation in Gaussian networks

In [Ripley, 1987] two general approaches for sampling from multivariate normal distributions were introduced. The first method is based on a Cholesky decomposition of the covariance matrix, and the second, known as the conditioning method, generates instances of $\boldsymbol{X}$ by sampling X_1, then X_2 conditionally to X_1, and so on. This second method is analogous to PLS, the sampling procedure introduced in Section 4.2.2 for Bayesian networks, but with the particularity of being designed for Gaussian networks.

The simulation of a univariate normal distribution can be carried out by means of a simple method based on the sum of 12 uniform variables. Traditional methods based on the ratio-of-uniforms [Box and Muller, 1958, Brent, 1974, Marsaglia et al., 1976] could also be applied alternatively.

4.3 Estimation of distribution algorithms in discrete domains

4.3.1 Introduction

This section introduces the notations that will be used to describe EDAs in discrete domains. It also constitutes a review of the EDA approaches for combinatorial optimization problems that can be found in the literature.

Let X_i $(i = 1, \ldots, n)$ be a random variable. A possible instantiation of X_i will be denoted x_i. $p(X_i = x_i)$ –or simply $p(x_i)$– will denote the probability that the variable X_i takes the value x_i. Similarly, $\boldsymbol{X} = (X_1, \ldots, X_n)$ will represent an n–dimensional random variable, and $\boldsymbol{x} = (x_1, \ldots, x_n)$ one of its possible realizations. The mass probability of $\boldsymbol{X}$ will be denoted by $p(\boldsymbol{X} = \boldsymbol{x})$ –or simply $p(\boldsymbol{x})$. The conditional probability of the variable X_i given the value x_j of the variable X_j will be written as $p(X_i = x_i \mid X_j = x_j)$ (or simply $p(x_i \mid x_j)$). D will denote a data set, i.e., a set of R instantiations of the variables $(X_1, \ldots, X_n)$.

Figure 4.5 shows the pseudocode of EDAs in combinatorial optimization problems using the notation introduced, where $\boldsymbol{x} = (x_1, \ldots, x_n)$ will represent the individuals of n genes, and D_l will denote the population of R individuals in the l^{th} generation. Similarly, D_l^N will represent the population of the selected N individuals from D_{l-1}. In EDAs the main task is to estimate $p(\boldsymbol{x} \mid D_{l-1}^N)$, that is, the joint conditional probability over one individual $\boldsymbol{x}$ being

EDA

$D_0 \leftarrow$ Generate R individuals (the initial population) randomly

Repeat for $l = 1, 2, \ldots$ until a stopping criterion is met

$D_{l-1}^N \leftarrow$ Select $N < R$ individuals from D_{l-1} according to a selection method

$p_l(\boldsymbol{x}) = p(\boldsymbol{x} \mid D_{l-1}^N) \leftarrow$ Estimate the probability distribution of an individual being among the selected individuals

$D_l \leftarrow$ Sample R individuals (the new population) from $p_l(\boldsymbol{x})$

Figure 4.5: Pseudocode for EDA approaches in discrete domains.

among the selected individuals. This joint probability must be estimated every generation. We will denote by $p_l(\boldsymbol{x}) = p_l(\boldsymbol{x} \mid D_{l-1}^N)$ the joint conditional probability at the l^{th} generation.

The most important step is to find the interdependencies between the variables that represent one point in the search space. The basic idea consists in inducing probabilistic models from the best individuals of the population. Once the probabilistic model has been estimated the model is sampled to generate new individuals (new solutions), which will be used to generate a new model in the next generation. This procedure is repeated until a stopping criterion is satisfied. Moreover, the most difficult step for EDAs is actually to estimate satisfactorily the probability distribution $p_l(\boldsymbol{x})$, as the computation of all the parameters needed to specify the underlying probability model becomes impractical. That is why several approximations propose to factorize the probability distribution according to a probability model.

The next sections introduce EDA approaches that can be found in the literature. All the algorithms and methods are classified depending on the maximum number of dependencies between variables that they can account for (maximum number of parents that a variable X_i can have in the probabilistic graphical model). The reader can find in [Larrañaga and Lozano, 2001] a more complete review of this topic.

4.3.2 Without interdependencies

All methods belonging to this category assume that the n–dimensional joint probability distribution factorizes like a product of n univariate and independent probability distributions, that is $p_l(\boldsymbol{x}) = \prod_{i=1}^{n} p_l(x_i)$. This assumption appears to be inexact due to the nature of any difficult optimization problem, where interdependencies between the variables will exist to some degree. Nevertheless, this approximation can lead to an acceptable behavior of EDAs for some problems like the ones on which independence between variables can be assumed.

There are several approaches corresponding to this category that can be found in the literature. Examples are Bit-Based Simulated Crossover –BSC– [Syswerda, 1993], the Population-Based Incremental Learning –PBIL– [Baluja, 1994], the compact Genetic Algorithm [Harik et al., 1998], and the Univariate Marginal Distribution Algorithm –UMDA– [Mühlenbein, 1998].

As an example to show the different ways of computing $p_l(x_i)$, in UMDA this task is

done by estimating the relative marginal frequencies of the i^{th} variable within the subset of selected individuals D_{l-1}^N. We describe here this algorithm in more detail as an example of approaches on this category.

UMDA –Univariate Marginal Distribution Algorithm

This algorithm assumes all the variables to be independent in order to estimate the mass joint probability. More formally, the UMDA approach can be written as:

$$p_l(\boldsymbol{x};\boldsymbol{\theta}^l) = \prod_{i=1}^{n} p_l(x_i;\boldsymbol{\theta}_i^l) \tag{4.9}$$

where $\boldsymbol{\theta}_i^l = \left(\theta_{ijk}^l\right)$ is recalculated every generation by its maximum likelihood estimation, i.e. $\widehat{\theta}_{ijk}^l = \frac{N_{ijk}^{l-1}}{N_{ij}^{l-1}}$, N_{ijk}^l is the number of cases in which the variable X_i takes the value x_i^k when its parents are on their j^{th} combination of values for the l^{th} generation, with $N_{ij}^{l-1} = \sum_k N_{ijk}^{l-1}$. The latter estimation can be allowed since the representation of individuals chosen assumes that, all the variables are discrete, and therefore the estimation of the local parameters needed to obtain the joint probability distribution $-\widehat{\theta}_{ijk}^{\,l}-$ is done by simply calculating the relative marginal frequencies of the i^{th} variable within the subset of selected individuals D_{l-1}^N in the l^{th} generation.

4.3.3 Pairwise dependencies

In an attempt to express the simplest possible interdependencies among variables, all the methods in this category propose that the joint probability distribution can be estimated well and fast enough by only taking into account dependencies between pairs of variables. Figure 4.6 shows examples of graphical models where these pairwise dependencies between variables are expressed.

Algorithms in this category require therefore an additional step that was not required in the previous one, which is the construction of a structure that best represents the probabilistic model. In other words, the parametric learning of the previous category –where the structure of the arc-less model remains fixed– is extended to a structural one.

An example of this second category is the greedy algorithm called MIMIC (Mutual Information Maximization for Input Clustering) proposed in [de Bonet et al., 1997], which is explained in more detail below. Other approaches in this group are the ones proposed in [Baluja and Davies, 1997] and the one called BMDA (Bivariate Marginal Distribution Algorithm) [Pelikan and Mühlenbein, 1999].

MIMIC –Mutual Information Maximization for Input Clustering

MIMIC is an EDA proposed for the first time in [de Bonet et al., 1997]. The main idea is to describe the true mass joint probability as closely as possible by using only one univariate marginal probability and $n-1$ pairwise conditional probability functions.

Given a permutation $\pi = (i_1, i_2, \ldots, i_n)$, we define the class of probability functions, $\mathcal{P}_\pi(\boldsymbol{x})$, as

$$\mathcal{P}_\pi(\boldsymbol{x}) = \{p_\pi(\boldsymbol{x}) \mid p_\pi(\boldsymbol{x}) = p(x_{i_1} \mid x_{i_2}) \cdot p(x_{i_2} \mid x_{i_3}) \cdot \ldots \cdot p(x_{i_{n-1}} \mid x_{i_n}) \cdot p(x_{i_n})\} \tag{4.10}$$

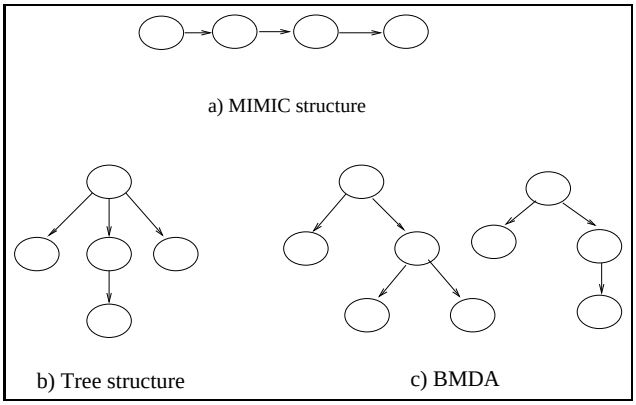


Figure 4.6: Graphical representation of proposed EDA in combinatorial optimization with pairwise dependencies (MIMIC, tree structure, BMDA).

where $p(x_{i_n})$ and $p(x_{i_j} \mid x_{i_{j+1}})$, $j = 1, \ldots, n-1$, are estimated by the marginal and conditional relative frequencies of the corresponding variables within the subset of selected individuals D_{l-1}^N in the l^{th} generation. The goal for MIMIC is to choose the appropriate permutation π^* such that the associated $p_{\pi^*}(\boldsymbol{x})$ minimizes the Kullback-Leibler information divergence between the true probability function, $p(\boldsymbol{x})$, and the probability functions, $p_\pi(\boldsymbol{x})$, of the class $\mathcal{P}_\pi(\boldsymbol{x})$. More formally,

$$D_{K-L}(p(\boldsymbol{x}), p_\pi(\boldsymbol{x})) = \mathsf{E}_{p(\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x})}{p_\pi(\boldsymbol{x})}\right] = \sum_{\boldsymbol{x}} p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{p_\pi(\boldsymbol{x})}. \tag{4.11}$$

This Kullback-Leibler information divergence can be expressed using the Shanon entropy of a probability function, $h(p(\boldsymbol{x})) = -\mathsf{E}_{p(\boldsymbol{x})}[\log p(\boldsymbol{x})]$, in the following way:

$$\begin{aligned} D_{K-L}(p(\boldsymbol{x}), p_\pi(\boldsymbol{x})) = -h(p(\boldsymbol{x})) + h(X_{i_1} \mid X_{i_2}) + \\ h(X_{i_2} \mid X_{i_3}) + \ldots + h(X_{i_{n-1}} \mid X_{i_n}) + h(X_{i_n}) \end{aligned} \tag{4.12}$$

where $h(X \mid Y)$ denotes the mean uncertainty in X given Y, that is:

$$h(X \mid Y) = \sum_y h(X \mid Y = y) p_Y(y) \tag{4.13}$$

and

$$h(X \mid Y = y) = -\sum_x p(X = x \mid Y = y) \log p_{X|Y}(x \mid y) \tag{4.14}$$

expresses the uncertainty in X given that $Y = y$.

The latter equation can be rewritten by taking into account that $-h(p(\boldsymbol{x}))$ does not depend on π. Therefore, the task to accomplish is to find the sequence π^* that minimizes the expression

$$J_\pi(\boldsymbol{x}) = h(X_{i_1} \mid X_{i_2}) + \ldots + h(X_{i_{n-1}} \mid X_{i_n}) + h(X_{i_n}). \tag{4.15}$$

In [de Bonet et al., 1997] the authors prove that it is possible to find an approximation of π^* avoiding the need to search over all $n!$ permutations by using a straightforward greedy algorithm. The proposed idea consists in selecting firstly X_{i_n} as the variable with the smallest estimated entropy, and then in successive steps to pick up the variable –from the set of

MIMIC - Greedy algorithm to obtain π^*

(1) $i_n = \arg\min_j \widehat{h}(X_j)$

–search for the variable with shortest entropy

(2) $i_k = \arg\min_j \widehat{h}(X_j \mid X_{i_{k+1}})$

$j \neq i_{k+1}, \ldots, i_n \quad k = n-1, n-2, \ldots, 2, 1$

–every step, from all the variables not selected up to that step, look for the variable of shortest entropy conditioned to the one before

Figure 4.7: MIMIC approach to estimate the mass joint probability distribution.

variables not chosen so far– such that its average conditional entropy with respect to the previous one is the smallest.

Figure 4.7 shows the pseudocode of MIMIC.

4.3.4 Multiple interdependencies

Several other EDA approaches in the literature propose the factorization of the joint probability distribution to be done by statistics of order greater than two. Figure 4.8 shows different probabilistic graphical models that are included in this category. As the number of dependencies between variables is greater than in the previous categories, the complexity of the probabilistic structure as well as the task of finding the best structure that suits the model is bigger. Therefore, these approaches require a more complex learning process.

The following is a brief review of the most important EDA approaches that can be found in the literature within this category:

- The FDA (Factorized Distribution Algorithm) is introduced in [Mühlenbein et al., 1999]. This algorithm applies to additively decomposed functions for which, using the running intersection property, a factorization of the mass-probability based on residuals and separators is obtained.

- In [Etxeberria and Larrañaga, 1999] a factorization of the joint probability distribution encoded by a Bayesian network is learnt from the database containing the selected individuals in every generation. The algorithm developed is called EBNA (Estimation of Bayesian Networks Algorithm), and it makes use of the Bayesian Information Criterion (BIC) score as the measure of the quality of the Bayesian network structure together with greedy algorithms that perform the search in the space of models. This algorithm is explained in more detail later in this section as an example of its category.

- In [Pelikan et al., 1999] the authors propose an algorithm called BOA (Bayesian Optimization Algorithm) which uses a Bayesian metric –the Bayesian Dirichlet equivalent (BDe) [Heckerman et al., 1995]– to measure the goodness of every structure found. A greedy search procedure is also used for this purpose. The search starts in each generation from scratch.

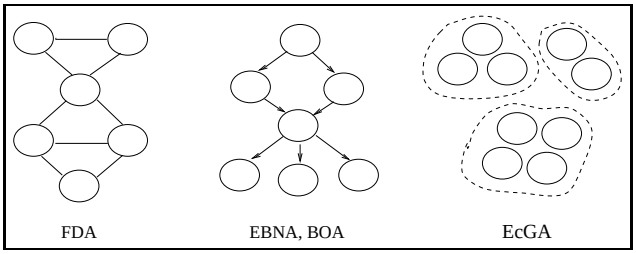


Figure 4.8: Graphical representation of proposed EDA in combinatorial optimization with multiply dependencies (FDA, EBNA, BOA and EcGA).

- The LFDA (Learning Factorized Distribution Algorithm) is introduced in [Mühlenbein and Mahning, 1999], which follows essentially the same approach as in EBNA.
- The Extend compact Genetic Algorithm (EcGA) proposed in [Harik, 1999] is an algorithm of which the basic idea consists in factorizing the joint probability distribution as a product of marginal distributions of variable size.

EBNA –Estimation of Bayesian Network Algorithm

EBNA is an EDA proposed in [Etxeberria and Larrañaga, 1999] that belongs to the category of algorithms that take into account multiple interdependencies between variables. This algorithm proposes the construction of a probabilistic graphical model with no restriction in the number of parents that variables can have.

In brief, the EBNA approach is based on a score+search method: a measure is selected to indicate the adequacy of any Bayesian network for representing the interdependencies between the variables –the score– and this is applied in a procedure that will search for the structure that obtains a satisfactory score value –the search process.

Scores for Bayesian networks.

In this algorithm, given a database D with N cases, $D = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, a measure of the success of any structure S to describe the observed data D is proposed. This measure is obtained by computing the maximum likelihood estimate $-\widehat{\boldsymbol{\theta}}-$ for the parameters $\boldsymbol{\theta}$ and the associated maximized log likelihood, $\log p(D \mid S, \widehat{\boldsymbol{\theta}})$. The main idea in EBNA is to search for the structure that maximizes $\log p(D \mid S, \widehat{\boldsymbol{\theta}})$ using an appropriate search strategy. This is done by scoring each structure by means of its associated maximized log likelihood.

Using the notation introduced in Section 4.3.1, we obtain

$$\begin{aligned} \log p(D \mid S, \boldsymbol{\theta}) &= \log \prod_{w=1}^{N} p(\boldsymbol{x}_w \mid S, \boldsymbol{\theta}) \\ &= \log \prod_{w=1}^{N} \prod_{i=1}^{n} p(x_{w,i} \mid \boldsymbol{pa}_i^S, \boldsymbol{\theta}_i) \\ &= \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} \log(\theta_{ijk})^{N_{ijk}} \end{aligned} \tag{4.16}$$

EBNA$_{BIC}$

$M_0 \leftarrow (S_0, \boldsymbol{\theta}_0)$
$D_0 \leftarrow$ Sample R individuals from M_0
For $l = 1, 2, \ldots$ until a stop criterion is met
$D_{l-1}^N \leftarrow$ Select N individuals from D_{l-1}
$S_l^* \leftarrow$ Find the structure which maximizes $BIC(S_l, D_{l-1}^N)$
$\boldsymbol{\theta}^l \leftarrow$ Calculate $\{\theta_{ijk}^l = \frac{N_{ijk}^{l-1}+1}{N_{ij}^{l-1}+r_i}\}$ using D_{l-1}^N as data set
$M_l \leftarrow (S_l^*, \boldsymbol{\theta}^l)$
$D_l \leftarrow$ Sample R individuals from M_l using PLS

Figure 4.9: Pseudocode for EBNA$_{BIC}$ algorithm.

where N_{ijk} denotes the number of cases in D in which the variable X_i has the value x_i^k and $\boldsymbol{Pa}_i$ is instantiated as its j^{th} value, and $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$.

Knowing that the maximum likelihood estimate for θ_{ijk} is given by $\widehat{\theta}_{ijk} = \frac{N_{ijk}}{N_{ij}}$, the previous equation can be rewritten as

$$\log p(D \mid S, \widehat{\boldsymbol{\theta}}) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} \log \frac{N_{ijk}}{N_{ij}}. \tag{4.17}$$

For the case of complex models, the sampling error associated with the maximum likelihood estimator might turn out to be too big to consider the maximum likelihood estimate as a reliable value for the parameter –even for a large sample. A common response to this difficulty is to incorporate some form of penalty depending on the complexity of the model into the maximized likelihood. Several penalty functions have been proposed in the literature. A general formula for a penalized maximum likelihood score could be

$$\sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} \log \frac{N_{ijk}}{N_{ij}} - f(N) dim(S) \tag{4.18}$$

where $dim(S)$ is the dimension –i.e. the number of parameters needed to specify the model– of the Bayesian network following the structure given by S. This dimension is computed as $dim(S) = \prod_{i=1}^{n} q_i(r_i - 1)$. The penalization function $f(N)$ is a non-negative one. Examples of values given to $f(N)$ in the literature are the Akaike's Information Criterion (AIC) [Akaike, 1974] –where it is considered as a constant, $f(N) = 1$– and the Jeffreys-Schwarz criterion which is also known as the Bayesian Information Criterion (BIC) [Schwarz, 1978] –where $f(N) = \frac{1}{2} \log N$.

Following the latter criterion, the corresponding BIC score –$BIC(S, D)$– for a Bayesian network structure S constructed from a database D and containing N cases is as follows:

$$BIC(S, D) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} \log \frac{N_{ijk}}{N_{ij}} - \frac{\log N}{2} \sum_{i=1}^{n} (r_i - 1) q_i \tag{4.19}$$

where N_{ijk} and N_{ij} and q_i are defined as above.

EBNA$_{K2}$

$M_0 \leftarrow (S_0, \boldsymbol{\theta}_0)$
$D_0 \leftarrow$ Sample R individuals from M_0
For $l = 1, 2, \ldots$ until a stop criterion is met
$D_{l-1}^N \leftarrow$ Select N individuals from D_{l-1}
$S_l^* \leftarrow$ Find the structure which maximizes $K2(S_l, D_{l-1}^N)$
$\boldsymbol{\theta}^l \leftarrow$ Calculate $\{\theta_{ijk}^l = \frac{N_{ijk}^{l-1}+1}{N_{ij}^{l-1}+r_i}\}$ using D_{l-1}^N as data set
$M_l \leftarrow (S_l^*, \boldsymbol{\theta}^l)$
$D_l \leftarrow$ Sample R individuals from M_l using PLS

Figure 4.10: Pseudocode for EBNA$_{K2}$ algorithm.

On the other hand, by assuming that all the local probability distributions θ_{ijk} in EBNA follow a Dirichlet distribution with the hyperparameters $\alpha_{ijk} = 1$, these are calculated every generation using their expected values as obtained in [Cooper and Herskovits, 1992]:

$$\mathsf{E}[\theta_{ijk}^l \mid S, D_{l-1}^N] = \frac{N_{ijk}^{l-1} + 1}{N_{ij}^{l-1} + r_i}. \tag{4.20}$$

The whole approach is illustrated in Figure 4.9, which corresponds to the one developed originally in [Etxeberria and Larrañaga, 1999]. In this paper, the authors use the penalized maximum likelihood as the score to evaluate the goodness of each structure found during the search. In particular, they propose the use of the BIC score. Due to the application of scores other than BIC, the original EBNA that is illustrated in Figure 4.9 is commonly known as EBNA$_{BIC}$.

Another score that has also been proposed in the literature is an adaption of the K2 algorithm [Cooper and Herskovits, 1992], which is also known as EBNA$_{K2}$. Given a Bayesian network, if the cases occur independently, there are no missing values, and the density of the parameters given the structure is uniform, then the authors show that

$$p(D \mid S) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}!. \tag{4.21}$$

EBNA$_{K2}$ assumes that an ordering on the variables is available and that, a priori, all structures are equally likely. It searches, for every node, the set of parent nodes that maximizes the following function:

$$g(i, \boldsymbol{Pa}_i) = \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}!. \tag{4.22}$$

Following this definition, the corresponding K2 score –$K2(S, D)$– for a Bayesian network structure S constructed from a database D and containing N cases is:

$$K2(S, D) = \sum_{i=1}^{n} g(i, \boldsymbol{Pa}_i) = \sum_{i=1}^{n} \left(\prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}! \right) \tag{4.23}$$

where N_{ijk} and N_{ij} and q_i are defined as above.

EBNA$_{K2}$ is proposed as a *greedy* heuristic. It starts by assuming that a node does not have parents, then in each step it adds incrementally that parent whose addition most increases the probability of the resulting structure. EBNA$_{K2}$ stops adding parents to the nodes when the addition of a single parent can not increase this probability. Obviously, as well as with EBNA$_{BIC}$, this approach does not guarantee to obtain the structure with the highest probability.

Search methods.

Regarding the search method that is combined with the score, in order to obtain the best existing model all possible structures must be searched through. Unfortunately, this has been proved to be NP-hard [Chickering et al., 1994]. Even if promising results have been obtained through global search techniques [Etxeberria et al., 1997a,b, Larrañaga et al., 1996a,b,c], their computation cost makes them impractical for our problem. As the aim is to find a satisfactory model as good as possible –even if not the optimal– in a reasonable period of time, a simpler search method that avoids analyzing all the possible structures is preferred. An example of the latter is the so called B Algorithm [Buntine, 1991]. The B Algorithm is a greedy search heuristic which starts from an arc-less structure and adds iteratively the arcs that produce maximum improvement according to the BIC approximation –although other measures could also be applied. The algorithm stops when adding another arc would not increase the score of the structure.

Local search strategies are another way of obtaining good models. These start from a given structure, and every step the addition or deletion of an arc that improves most the scoring measure is performed. Local search strategies stop when no modification of the structure improves the scoring measure. The main drawback of local search strategies is their heavy dependence on the initial structure. Nevertheless, as [Chickering et al., 1995] showed that local search strategies perform quite well when the initial structure is reasonably good, the model of the previous generation could be used as the initial structure when the search is based on the assumption that $p(\boldsymbol{x} \mid D_l^N)$ will not differ very much from $p(\boldsymbol{x} \mid D_{l-1}^N)$

The initial model M_0 in EBNA is formed by a structure S_0, which is an arc-less DAG, and the local probability distributions given by the n unidimensional marginal probabilities $p(X_i = x_i) = \frac{1}{r_i}$, $i = 1, \ldots, n$ –that is, M_0 assigns the same probability to all individuals. The model of the first generation –M_1– is learnt using Algorithm B, while the rest of the models are learnt by means of a local search strategy which take the model of the previous generation as the initial structure.

4.4 Estimation of distribution algorithms in continuous domains

4.4.1 Introduction

This section complements the previous two ones, as it introduces the EDA algorithms for their use in optimization in continuous domains. The continuous EDAs will be discussed following the same layout as in the previous sections. All the continuous EDA approaches will be also classified using an analogous comparison based on the complexity of the estimation of the probability distribution. In this case, as we are in the continuous domain, the density function will be factorized as a product of n conditional density functions.

The notation for continuous EDAs does not vary significantly from the one presented for discrete EDAs, as continuous EDAs follow essentially equivalent steps to approximate the best solution, and therefore Figure 4.5 is still valid to describe the continuous EDA approach (it would be enough with substituting $p_l(\boldsymbol{x})$ by $f_l(\boldsymbol{x})$). Nevertheless, regarding the learning and simulation steps, EDAs in the continuous domain have some characteristics that make them very particular.

Let $X_i \in \boldsymbol{X}$ be a continuous variable. Similarly as in the discrete domain, a possible instantiation of X_i will be denoted x_i, and D will denote a data set, i.e., a set of R instantiations of the variables $(X_1, \ldots, X_n)$. In the continuous domain, again $\boldsymbol{x} = (x_1, \ldots, x_n)$ represents an individual of n variables, D_l denotes the population of R individuals in the l^{th} generation, and D_l^N represents the population of the selected N individuals from D_l, and as for the discrete case, the main task is still to estimate the joint density function at every generation. We will denote by $f_l(\boldsymbol{x} \mid D_{l-1}^N)$ the joint conditional density function at the l^{th} generation.

The most difficult step in here is also the search of interdependencies between the different variables. Again, in continuous EDAs probabilistic models are induced from the best N individuals of the population. Once the structure is estimated, this model is sampled to generate the R new individuals that will form the new generation. As the estimation of the joint density function is a tedious task, approximations are applied in order to estimate the best joint density function according to the probabilistic model learned at each generation.

As with the discrete domain, all the continuous EDA approaches can be divided in different categories depending on the degree of dependency that they take into account. Following the classification in [Larrañaga and Lozano, 2001], we will divide all the continuous EDA in three main categories.

4.4.2 Without dependencies

This is the category of algorithms that do not take into account dependencies between any of the variables. In this case, the joint density function is factorized as a product of n one-dimensional and independent densities. Examples of continuous EDAs in this category are the Univariate Marginal Distribution Algorithm for application in continuous domains (UMDA$_c$) [Larrañaga et al., 2000], Stochastic Hill-Climbing with Learning by Vectors of Normal Distributions (SHCLVND) [Rudlof and Köppen, 1996], Population-Based Incremental Learning for continuous domains (PBIL$_c$) [Sebag and Ducoulombier, 1998], and the algorithm introduced in [Servet et al., 1997]. As an example of all these, UMDA$_c$ is shown more in detail.

UMDA$_c$

The Univariate Marginal Distribution Algorithm for application in continuous domains (UMDA$_c$) was introduced in [Larrañaga et al., 2000]. In this approach, every generation and for every variable some statistical tests are performed to obtain the density function that best fits the variable. In UMDA$_c$ the factorization of the joint density function is given by

$$f_l(\boldsymbol{x}; \boldsymbol{\theta}^l) = \prod_{i=1}^{n} f_l(x_i, \boldsymbol{\theta}_i^l). \tag{4.24}$$

Unlike UMDA in the discrete case, UMDA$_c$ is a structure identification algorithm meaning that the density components of the model are identified with the aid of hypothesis tests.

UMDA$_c$

** learning the joint density function **
for $l = 1, 2, \ldots$ until the stopping criterion is met
  for $i = 1$ to n do
    (i) select via hypothesis test the density function $f_l(x_i; \boldsymbol{\theta}_i^l)$ that
      best fits D_{l-1}^{N,X_i}, the projection of the selected individuals over the ith variable
    (ii) obtain the maximum likelihood estimates for $\boldsymbol{\theta}_i^l = (\theta_i^{l,k_1}, \ldots, \theta_i^{l,k_i})$

Each generation the learnt joint density function is expressed as:
$f_l(\boldsymbol{x}; \boldsymbol{\theta}^l) = \prod_{i=1}^{n} f_l(x_i, \widehat{\boldsymbol{\theta}}_i^{\,l})$

Figure 4.11: Pseudocode to estimate the joint density function followed in UMDA$_c$.

Once the densities have been identified, the estimation of parameters is carried out by means of their maximum likelihood estimates.

If all the univariate distributions are normal distributions, then for each variable two parameters are estimated at each generation: the mean, μ_i^l, and the standard deviation, σ_i^l. It is well known that their respective maximum likelihood estimates are:

$$\widehat{\mu_i}^l = \overline{X_i}^l = \frac{1}{N}\sum_{r=1}^{N} x_{i,r}^l; \qquad \widehat{\sigma_i}^l = \sqrt{\frac{1}{N}\sum_{r=1}^{N}\left(x_{i,r}^l - \overline{X_i}^l\right)^2} \tag{4.25}$$

This particular case of the UMDA$_c$ is called UMDA$_c^G$ (Univariate Marginal Distribution Algorithm for Gaussian models).

Figure 4.11 shows the pseudocode to learn the joint density function followed by UMDA$_c$.

4.4.3 Bivariate dependencies

MIMIC$_c^G$

This algorithm was introduced in [Larrañaga et al., 2000] and is basically an adaptation of the MIMIC algorithm [de Bonet et al., 1997] to the continuous domain. In this, the underlying probability model for every pair of variables is assumed to be a bivariate Gaussian.

Similarly as in MIMIC, the idea is to describe the underlying joint density function that fits the model as closely as possible to the empirical data by using only one univariate marginal density and $n-1$ pairwise conditional density functions. For that, the following theorem [Whittaker, 1990] is used:

Theorem 4.1: [Whittaker, 1990, pp. 167] Let $\boldsymbol{X}$ be a n–dimensional normal density function, $\boldsymbol{X} \equiv \boldsymbol{\mathcal{N}}(\boldsymbol{x}; \boldsymbol{\mu}, \sum)$, then the entropy of $\boldsymbol{X}$ is

$$h(\boldsymbol{X}) = \frac{1}{2}n(1 + \log 2\pi) + \frac{1}{2}\log|\sum|\,. \tag{4.26}$$

MIMIC_c^G

Choose $i_n = \arg\min_j \ \hat{\sigma}^2_{X_j}$

for $k = n-1, n-2, \ldots, 1$

Choose $i_k = \arg\min_j \ \hat{\sigma}^2_{X_j} - \frac{\hat{\sigma}^2_{X_j X_{i_{k+1}}}}{\hat{\sigma}^2_{X_{i_{k+1}}}}$

$j \neq i_{k+1}, \ldots, i_n$

Figure 4.12: Adaptation of the MIMIC approach to a multivariate Gaussian density function.

When applying this result to univariate and bivariate normal density functions to define MIMIC_c^G, we obtain that

$$h(X) = \frac{1}{2}(1 + \log 2\pi) + \log \sigma_X \tag{4.27}$$

$$h(X \mid Y) = \frac{1}{2}\left[(1 + \log 2\pi) + \log\left(\frac{\sigma^2_X \sigma^2_Y - \sigma^2_{XY}}{\sigma^2_Y}\right)\right] \tag{4.28}$$

where $\sigma^2_X(\sigma^2_Y)$ is the variance of the univariate $X(Y)$ variable and σ_{XY} denotes the covariance between the variables X and Y.

The learning of the structure in MIMIC_c^G is shown in Figure 4.12. It follows a straightforward greedy algorithm composed of two steps. In the first one, the variable with the smallest sample variance is chosen. In the second step, the variable X with the smallest estimation of $\frac{\sigma^2_X \sigma^2_Y - \sigma^2_{XY}}{\sigma^2_Y}$ regarding the variable Y chosen in the previous iteration is selected, and X is linked to Y in the structure.

4.4.4 Multiple interdependencies

Algorithms in this section are approaches of EDAs for continuous domains in which there is no restriction on the number of interdependencies between variables to take into account on the density function learnt at every generation. In the first example introduced, the density function corresponds to a non restricted multivariate normal density that is learned from scratch at each generation. The next two examples are respectively an adaptation and an improvement of this first model. Finally, this section also introduces edge exclusion test approaches to learn from Gaussian networks, as well as two score+search approaches to search for the most appropriated Gaussian network at each generation.

EMNA_{global}

This approach performs the estimation of a multivariate normal density function at each generation. Figure 4.13 shows the pseudocode of EMNA_{global} (Estimation of Multivariate Normal Algorithm – global). In EMNA_{global}, at every generation we proceed as follows: the vector of means $\boldsymbol{\mu}_l = (\mu_{1,l}, \ldots, \mu_{n,l})$, and the variance–covariance matrix Σ_l are computed. The elements of the latter are represented as $\sigma^2_{ij,l}$ with $i, j = 1, \ldots, n$. As a result, at every generation all the $2n + \begin{pmatrix} n-1 \\ 2 \end{pmatrix}$ parameters need to be estimated: n means, n variances

EMNA$_{global}$

$D_0 \leftarrow$ Generate R individuals (the initial population) at random

for $l = 1, 2, \ldots$ until the stopping criterion is met

$D_{l-1}^N \leftarrow$ Select $N < R$ individuals from D_{l-1} according to the selection method

$f_l(\boldsymbol{x}) = f(\boldsymbol{x} \mid D_{l-1}^N) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_l, \Sigma_l) \leftarrow$ Estimate the multivariate normal density function from the selected individuals

$D_l \leftarrow$ Sample R individuals (the new population) from $f_l(\boldsymbol{x})$

Figure 4.13: Pseudocode for the EMNA$_{global}$ approach.

and $\begin{pmatrix} n-1 \\ 2 \end{pmatrix}$ covariances. This is performed using their maximum likelihood estimates in the following way:

$$\hat{\mu}_{i,l} = \frac{1}{N} \sum_{r=1}^{N} x_{i,r}^l \quad i = 1, \ldots, n$$

$$\hat{\sigma}_{i,l}^2 = \frac{1}{N} \sum_{r=1}^{N} (x_{i,r}^l - \overline{X}_i^l)^2 \quad i = 1, \ldots, n$$

$$\hat{\sigma}_{ij,l}^2 = \frac{1}{N} \sum_{r=1}^{N} (x_{i,r}^l - \overline{X}_i^l)(x_{j,r}^l - \overline{X}_j^l) \quad i, j = 1, \ldots, n \quad i \neq j. \tag{4.29}$$

At a first glance the reader could think that this approach requires much more computation than in the other cases in which the estimation of the joint density function is done with Gaussian networks. However, the mathematics on which this approach is based are quite simple. On the other hand, approaches based on edge exclusion tests on Gaussian networks also require the computation of as many parameters as the ones needed by this approach in order to carry out a hypothesis test on them. Moreover, the second type of Gaussian network approaches based on score+search methods also require a lot of extra computation, as the searching process needs to look for the best structure over the whole space of possible models. The reader can find more details about EMNA$_{global}$ in [Larrañaga et al., 2001].

EMNA$_a$

EMNA$_a$ (Estimation of Multivariate Normal Algorithm – adaptive) is an adaptive version of the previous approach.

The biggest particularity of this algorithm is the way of obtaining the first model, $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_1, \Sigma_1)$, the parameters of which are estimated from the best individuals selected in the initial population. After this step, EMNA$_a$ behaves as a steady–step genetic algorithm. The pseudocode for EMNA$_a$ is given in Figure 4.14.

Every iteration, an individual from the current multivariate normal density model is sampled. Next, the goodness of this simulated individual is compared to the worst individual

EMNA$_a$

$D_0 \leftarrow$ Generate R individuals (the initial population) at random
Select $N < R$ individuals from D_0 according to the selection method
Obtain the first multivariate normal density $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_1, \Sigma_1)$
for $l = 1, 2, \ldots$ until the stopping criterion is met
  Generate an individual $\boldsymbol{x}_{ge}^l$ from $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_l, \Sigma_l)$
  if $\boldsymbol{x}_{ge}^l$ is better than the worst individual, $\boldsymbol{x}^{l,N}$, then
    1.- Add $\boldsymbol{x}_{ge}^l$ to the population and drop $\boldsymbol{x}^{l,N}$ from it
    2.- Obtain $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_{l+1}, \Sigma_{l+1})$

Figure 4.14: Pseudocode for the EMNA$_a$ approach.

of the current population. If the fitness value of the new individual has a better value, then the new individual replaces the worst one in the population. In the latter case, it is also necessary to update the parameters of the multivariate normal density function.

The updating of the density function is done using the following formulas that can be obtained by means of simple algebraic manipulations [Larrañaga et al., 2001]:

$$\boldsymbol{\mu}_{l+1} = \boldsymbol{\mu}_l + \frac{1}{N}\left(\boldsymbol{x}_{ge}^l - \boldsymbol{x}^{l,N}\right) \tag{4.30}$$

$$\begin{aligned}
\sigma_{ij,l+1}^2 &= \sigma_{ij,l}^2 - \frac{1}{N^2}\left(x_{ge,i}^l - x_i^{l,N}\right)\cdot\sum_{r=1}^{N}\left(x_j^{l,r} - \mu_j^l\right) - \frac{1}{N^2}\left(x_{ge,j}^l - x_j^{l,N}\right)\cdot\sum_{r=1}^{N}\left(x_i^{l,r} - \mu_i^l\right) \\
&+ \frac{1}{N^2}\left(x_{ge,i}^l - x_i^{l,N}\right)\left(x_{ge,j}^l - x_j^{l,N}\right) - \frac{1}{N}\left(x_i^{l,N} - \mu_i^{l+1}\right)\left(x_j^{l,N} - \mu_j^{l+1}\right) \\
&+ \frac{1}{N}\left(x_{ge,i}^l - \mu_i^{l+1}\right)\left(x_{ge,j}^l - \mu_j^{l+1}\right)
\end{aligned} \tag{4.31}$$

where $\boldsymbol{x}_{ge}^l$ represents the individual generated in the lth iteration. Note also that in the EMNA$_a$ approach the size of the population remains constant every generation independently of the fitness of the individual sampled.

EMNA$_i$

EMNA$_i$ (Estimation of Multivariate Normal Algorithm – incremental) is a new approach following a similar idea of EMNA$_a$, as both algorithms generate each generation a simple individual which fitness value is compared to the worst individual in the current population. However, the biggest difference between them is what happens with the worst individual when its fitness value is lower than the new individual: in EMNA$_i$ the worst individual remains in the population, and therefore the population increases in size in those cases. Its main interest is that the rules to update the density function are simpler than in EMNA$_a$.

The reader is referred to [Larrañaga et al., 2001] for more details about this algorithm.

EGNA$_{ee}$, EGNA$_{BGe}$, EGNA$_{BIC}$

For $l = 1, 2, \ldots$ until the stopping criterion is met

$D_{l-1}^{N} \leftarrow$ Select N individuals from D_{l-1}

(i) $\hat{S}_l \leftarrow$ Structural learning via:
edge exclusion tests $\rightarrow$ EGNA$_{ee}$
Bayesian score+search $\rightarrow$ EGNA$_{BGe}$
penalized maximum likelihood + search $\rightarrow$ EGNA$_{BIC}$

(ii) $\hat{\boldsymbol{\theta}}^{l} \leftarrow$ Calculate the estimates for the parameters of $\hat{S}_l$

(iii) $M_l \leftarrow (\hat{S}_l, \hat{\boldsymbol{\theta}}^{l})$

(iv) $D_l \leftarrow$ Sample R individuals from M_l using the continuous version of the PLS algorithm

Figure 4.15: Pseudocode for the EGNA$_{ee}$, EGNA$_{BGe}$, and EGNA$_{BIC}$ algorithms.

EGNA$_{ee}$, EGNA$_{BGe}$, EGNA$_{BIC}$

The optimization in continuous domains can also be carried out by means of the learning and simulation of Gaussian networks. An example of this is the EGNA approach (Estimation of Gaussian Networks Algorithm), which is illustrated in Figure 4.15. This approach has three versions: EGNA$_{ee}$ [Larrañaga et al., 2000, Larrañaga and Lozano, 2001], EGNA$_{BGe}$ and EGNA$_{BIC}$ [Larrañaga et al., 2001]. The basic steps of these algorithms each iteration are as follows:

1. Obtain the Gaussian network structure by using one of the different methods proposed, namely edge–exclusion tests, Bayesian score+search, or penalized maximum likelihood+search.

2. Computation of estimates for the parameters of the learned Gaussian network structure.

3. Generation of the Gaussian network model.

4. Simulation of the joint density function expressed by the Gaussian network learned in the previous steps. An adaptation of the PLS algorithm to continuous domains is used for this purpose.

The main difference between all these EGNA approaches is the way of inducing the Gaussian network: in EGNA$_{ee}$ the Gaussian network is induced at each generation by means of edge exclusion tests, while the model induction in the EGNA$_{BGe}$ and EGNA$_{BIC}$ is carried out by score+search approaches. EGNA$_{BGe}$ makes use of a Bayesian score that gives the same value for Gaussian networks reflecting identical conditional (in)dependencies, and EGNA$_{BIC}$ uses a penalized maximum likelihood score based on the Bayesian Information Criterion (BIC). In both EGNA$_{BGe}$ and EGNA$_{BIC}$ a local search is used to search for good structures. These model induction methods are reviewed in the next subsection.

Applications of these EGNA approaches can be found in Cotta et al. (2001), Bengoetxea et al. (2001a), Lozano and Mendiburu (2001) and Robles et al. (2001).

Next, three different methods that can be applied to induce Gaussian networks from data are introduced. The first of them is based on edge exclusion tests, while the other two are score+search methods.

Edge exclusion tests

Dempster (1972) introduced a type of graphical Gaussian models on which the structure of the precision matrix is modelled rather than the variance matrix itself. The aim of this is to simplify the joint n–dimensional normal density by checking if a particular element w_{ij} with $i = 1, \ldots n-1$ and $j > i$ of the $n \times n$ precision matrix W can be set to zero. In [Wermuth, 1976] it was shown that fitting these models is equivalent to check the conditional independence between the corresponding elements of the n–dimensional variable $\boldsymbol{X}$. [Speed and Kiiveri, 1986] showed that this procedure is equivalent to check the possibility of deleting the arc connecting the nodes corresponding to X_i and X_j in the conditional independence graph. That is why these tests are commonly known as edge exclusion tests. As the fact of excluding any edge connecting X_i and X_j is analogous to accepting the null hypothesis $H_0 : w_{ij} = 0$ with the alternative hypothesis $H_A : w_{ij}$ unspecified, many graphical model selection procedures have a first step performing the $\binom{n}{2}$ single edge exclusion tests. In this first step, likelihood ratio statistic is evaluated and compared to a χ^2 distribution. However, the use of this distribution is only asymptotically correct. In [Smith and Whittaker, 1998] the authors introduced an alternative to these tests based on the likelihood ratio test that is discussed next.

The likelihood ratio test statistic to exclude the arc between X_i and X_j from a graphical Gaussian model is defined as $T_{lik} = -n \log(1 - r^2_{ij|rest})$, where $r_{ij|rest}$ is the sample partial correlation of X_i and X_j adjusted for the rest of the variables. This can be expressed in terms of the maximum likelihood estimate for every element of the precision matrix as $r_{ij|rest} = -\hat{w}_{ij}(\hat{w}_{ii}\hat{w}_{jj})^{-\frac{1}{2}}$ [Whittaker, 1990].

In [Smith and Whittaker, 1998] the density and distribution functions of the likelihood ratio test statistic is obtained through the null hypothesis. These expressions are of the form:

$$f_{lik}(t) = g_{\mathcal{X}}(t) + \frac{1}{4}(t-1)(2n+1)g_{\mathcal{X}}(t)N^{-1} + \bigcirc(N^{-2})$$

$$F_{lik}(x) = G_{\mathcal{X}}(x) + -\frac{1}{2}(2n+1)xg_{\mathcal{X}}(x)N^{-1} + \bigcirc(N^{-2}) \tag{4.32}$$

where $g_{\mathcal{X}}(t)$ and $G_{\mathcal{X}}(x)$ are the density and distribution functions of a $\mathcal{X}^2_1$ variable respectively.

Score+search methods

The idea behind this other approach consists in defining a measure to evaluate each candidate Gaussian network (i.e. the *score*) and to use a method to search in the space of possible structures the one with the best score (i.e. the *search*).

All the search methods discussed for Bayesian networks can also be applied for Gaussian networks. In Section 4.3.4 two scores called BIC and K2 were introduced, as well as two search procedures called B Algorithm and local search. Variants of these two search strategies could also be applied for Gaussian networks.

Regarding the score metrics for the continuous domain, different score metrics can be found in the literature in order evaluate how accurately a Gaussian network represents data dependencies. We will discuss the use of two types of them: the penalized maximum likelihood metric and Bayesian scores.

Penalized maximum likelihood: If $L(D \mid S, \boldsymbol{\theta})$ is the likelihood of the database $D = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ given a Gaussian network model $M = (S, \boldsymbol{\theta})$, then we have that:

$$L(D \mid S, \boldsymbol{\theta}) = \prod_{r=1}^{N} \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi v_i}} e^{-\frac{1}{2v_i}(x_{ir} - mi - \sum_{x_j \in \boldsymbol{Pa}_i} b_{ji}(x_{jr} - m_j))^2}. \tag{4.33}$$

The maximum likelihood estimates for $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n)$, namely $\widehat{\boldsymbol{\theta}} = (\widehat{\boldsymbol{\theta}}_1, \ldots, \widehat{\boldsymbol{\theta}}_n)$, are obtained either by maximizing $L(D \mid S, \boldsymbol{\theta})$ or also by maximizing the expression $\ln L(D \mid S, \boldsymbol{\theta})$. The definition of the latter is as follows:

$$\ln L(D \mid S, \boldsymbol{\theta}) =$$

$$\sum_{r=1}^{N} \sum_{i=1}^{n} [-\ln(\sqrt{2\pi v_i}) - \frac{1}{2v_i}(x_{ir} - mi - \sum_{x_j \in \boldsymbol{Pa}_i} b_{ji}(x_{jr} - m_j))^2] \tag{4.34}$$

where $\widehat{\boldsymbol{\theta}} = (\widehat{\boldsymbol{\theta}}_1, \ldots, \widehat{\boldsymbol{\theta}}_n)$ are the solutions of the following equation system:

$$\begin{cases} \frac{\partial}{\partial m_i} \ln L(D \mid S, \boldsymbol{\theta}) = 0 & i = 1, \ldots, n \\ \frac{\partial}{\partial v_i} \ln L(D \mid S, \boldsymbol{\theta}) = 0 & i = 1, \ldots, n \\ \frac{\partial}{\partial b_{ji}} \ln L(D \mid S, \boldsymbol{\theta}) = 0 & j = 1, \ldots, i-1 \text{ and } X_j \in \boldsymbol{Pa}_i \end{cases} \tag{4.35}$$

As proved in [Larrañaga et al., 2001], the maximum likelihood estimates for $\boldsymbol{\theta}_i = (m_i, b_{ji}, v_i)$ with $i = 1, \ldots, n$ and $j = 1, \ldots, i-1$ and $X_j \in \boldsymbol{Pa}_i$ are obtained as follows:

$$\hat{m}_i = \overline{X}_i$$

$$\hat{b}_{ji} = \frac{S_{X_j X_i}}{S^2_{X_j}}$$

$$\hat{v}_i = S^2_{X_i} - \sum_{X_j \in \boldsymbol{Pa}_i} \frac{S_{X_j X_i}}{S^2_{X_j}} + 2 \sum_{X_j \in \boldsymbol{Pa}_i} \sum_{X_k \in \boldsymbol{Pa}_i k > j} \frac{S_{X_j X_k} S_{X_j X_i} S_{X_k X_i}}{S^2_{X_j} S^2_{X_k}} \tag{4.36}$$

where $\overline{X}_i = \frac{1}{N} \sum_{r=1}^{N} x_{ir}$ is the sample mean of variable X_i, $S^2_{X_j} = \frac{1}{N} \sum_{r=1}^{N} \left(x_{jr} - \overline{X}_j\right)^2$ denotes the sample variance of variable X_j, and $S_{X_j X_i} = \frac{1}{N} \sum_{r=1}^{N} \left(x_{jr} - \overline{X}_j\right)\left(x_{ir} - \overline{X}_i\right)$ denotes the sample covariance between variables X_j and X_i. Note that in the case of $\boldsymbol{Pa}_i = \emptyset$, the variance corresponding to the i^{th} variable becomes only dependent on the value $S^2_{X_i}$, as all the rest of the terms in the formula become 0.

As stated before, a general formula for a penalized maximum likelihood score is

$$\sum_{r=1}^{N}\sum_{i=1}^{n}\left[-\ln(\sqrt{2\pi v_i})-\frac{1}{2v_i}\left(x_{ir}-mi-\sum_{x_j\in \boldsymbol{Pa}_i} b_{ji}(x_{jr}-m_j)\right)^2\right]-f(N)dim(S). \tag{4.37}$$

The number of parameters, $dim(S)$, required to fully specify a Gaussian network model with a structure given by S can be obtained using the following formula:

$$dim(S)=2n+\sum_{i=1}^{n}\mid \boldsymbol{Pa}_i \mid . \tag{4.38}$$

In fact, for each variable X_i we need to compute its mean, μ_i, its conditional variance, v_i, and its regression coefficients, b_{ji}. The comments on $f(N)$ in Section 3.3.2 are also valid here.

Bayesian scores: The Bayesian Dirichlet equivalent metric (BDe) [Heckerman et al., 1995] has a continuous version for Gaussian networks [Geiger and Heckerman, 1994] called Bayesian Gaussian equivalence (BGe). This metric has the property of being score equivalent. As a result, two Gaussian networks that are isomorphic –i.e. they represent the same conditional independence and dependence assertions– will always obtain the same score.

The metric is based upon the fact that the normal–Wishart distribution is conjugate with regard to the multivariate normal. This fact allows us to obtain a closed formula for the computation of the marginal likelihood of the data once given the structure.

[Geiger and Heckerman, 1994] proved that the marginal likelihood for a general Gaussian network can be computed using the following formula:

$$L(D\mid S)=\prod_{i=1}^{n}\frac{L\left(D^{X_i\cup \boldsymbol{Pa}_i}\mid S_c\right)}{L\left(D^{\boldsymbol{Pa}_i}\mid S_c\right)} \tag{4.39}$$

where each term is on the form given in Equation 4.40, and where $D^{X_i\cup \boldsymbol{Pa}_i}$ is the database D restricted to the variables $X_i\cup \boldsymbol{Pa}_i$.

Combining the results provided by the theorems given in [de Groot, 1970, Geiger and Heckerman, 1994] we obtain:

$$L(D\mid S_c)=(2\pi)^{-\frac{nN}{2}}\left(\frac{\nu}{\nu+N}\right)^{\frac{n}{2}}\frac{c(n,\alpha)}{c(n,\alpha+N)}|T_0|^{\frac{\alpha}{2}}|T_N|^{-\frac{\alpha+N}{2}} \tag{4.40}$$

where $c(n,\alpha)$ is defined as

$$c(n,\alpha)=\left[2^{\frac{\alpha n}{2}}\pi^{\frac{n(n-1)}{4}}\prod_{i=1}^{n}\Gamma\left(\frac{\alpha+1-i}{2}\right)\right]^{-1}. \tag{4.41}$$

This result yields a metric for scoring the marginal likelihood of any Gaussian network.

The reader is referred to [Geiger and Heckerman, 1994] for a discussion on the three components of the user's prior knowledge that are relevant for the learning in Gaussian

networks: (1) the prior probabilities $p(S)$, (2) the parameters α and ν, and (3) the parameters $\boldsymbol{\mu}_0$ and T_0.

4.5 Estimation of distribution algorithms for inexact graph matching

After having introduced the notation for EDAs in Sections 4.2.1 and 4.3.1, we will define the way of solving the inexact graph matching problem using any of the EDA approaches introduced so far.

4.5.1 Discrete domains

The representation of individuals that will be used in this section is the one proposed in the second representation of individuals in Section 3.3. The permutation-based representation will be used when applying continuous EDAs, as this representation is more suited for them –a deeper explanation of this is given in Section 4.5.2. When using discrete EDAs, a permutation-based representation will add an extra step for translating the individual to the solution it symbolizes before the individual can be evaluated applying the fitness function, with the consequent lack of performance. In the second representation of Section 3.3 we have individuals that contain directly the solution they symbolize, and therefore the fitness function is applied directly.

Let $G_M = (V_M, E_M)$ be the model graph, and $G_D = (V_D, E_D)$ the data graph obtained from a segmented image. The size of the individuals will be $n = |V_D|$, that is, $\boldsymbol{X} = (X_1, \ldots, X_{|V_D|})$ is a n-dimensional variable where each of its components can take $|V_M|$ possible values (i.e. in our case $r_i = |V_M|$ for $i = 1, \ldots, |V_D|$). We denote by $x_i^1, \ldots, x_i^{|V_M|}$ the possible values that the i^{th} variable, X_i, can have.

In the same way, for the unrestricted discrete distributions θ_{ijk}, the range of i, j and k for graph matching using the representation of individuals proposed is as follows: $i = 1, \ldots, |V_D|$, $k = 1, \ldots, |V_M|$, and $j = 1, \ldots, q_i$, where $q_i = |V_M|^{npa_i}$ and npa_i denotes the number of parents of X_i.

4.5.1.1 Estimating the probability distribution

We propose three different EDAs to be used to solve inexact graph matching problems in the discrete domain. The fact that the difference in behavior between algorithms is to a large extent due to the complexity of the probabilistic structure that they have to build, these three algorithms have been selected so that they are representatives of the many different types of discrete EDAs. Therefore, these algorithms can be seen as representatives of the three categories of EDAs introduced in Section 4.3: (1) UMDA [Mühlenbein, 1998] is an example of an EDA that considers no interdependencies between the variables (i.e. the learning is only parametrical, not structural). This assumption can be allowed in the case of inexact graph matching depending on the complexity of the problem, and mostly depending on $|V_M|$ and $|V_D|$. It is important to realize that a balance between cost and performance must be achieved, and therefore, in some complex problems the use of UMDA can be justified when trying to shorten the computation cost. Nevertheless, other algorithms that do not require such assumptions should return a better result if fast computation is not essential. (2) MIMIC [de Bonet et al., 1997] is an example that belongs to the category of pairwise

dependencies. Therefore, there is an extra task in MIMIC, which is the construction of the probabilistic graphical model that represents the pairwise dependencies. As a result, when applying this algorithm to graph matching, the computation time for the structural learning is proportional to the number of vertices of the graphs ($|V_M|$ and $|V_D|$). This shows again that due to the higher cost that MIMIC has UMDA can be used in order to obtain best results in a fixed period of time. (3) EBNA [Etxeberria and Larrañaga, 1999] is an example of the category of EDAs where multiple interdependencies are allowed between the variables, on which the structural learning is even more complex than in the two previous algorithms. The models constructed with EDAs describe the interdependencies between the different variables more accurately than those of MIMIC (and, obviously, better than with UMDA). Nevertheless, as the size of the graphs to match has a direct influence on the complexity of the Bayesian network to be build (the complexity and the computation time increase exponentially with $|V_M|$ and $|V_D|$), the use of other simpler probabilistic graphical models –which restrict the number of parents that a variable can have in the probabilistic structure– such as the two proposed above is justified.

Finally, a last remark about the fact that within the same algorithm there are sometimes different techniques to find the probabilistic structure that best suits the n–dimensional probabilistic model. In the case of Bayesian networks for instance, there are two ways of building the graph: by detecting dependencies, and through score and search [Buntine, 1996, de Campos, 1998, Heckerman, 1995, Krause, 1998, Sangüesa and Cortés, 1998]. These different possibilities can also have an influence on the structure estimated at each generation to represent the model. In the case of EBNA a score+search method using the *BIC* score is used, although any other score could also be used.

4.5.1.2 Adapting the simulation scheme to obtain correct individuals

Section 3.3.3 introduces the conditions that have to be satisfied to consider an individual as *correct* for the particular graph matching problems that we are considering in this thesis. In the same section we also showed that, in order to consider a solution as correct, the only condition to check is that all the vertices in graph G_M contain at least a matching, that is, that every vertex of G_M appears in the individual at least once. If we include a method in EDAs to ensure that this condition will be satisfied by each individual sampled, this would prevent the algorithm from generating incorrect individuals like the one in Figure 3.3b.

There are at least three ways of facing the problem of the existence of incorrect individuals in EDAs: controlling directly the simulation step, correction a posteriori, and changing the fitness function. The first technique can only be applied to EDAs, as it is based on modifying the previously estimated probability distribution and therefore it cannot be put into practice on other types of algorithms. The last two techniques can be applied to EDAs and other heuristics that deal with constraints. In [Michalewicz, 1992, Michalewicz and Schoenauer, 1996] we can find examples of GAs applied to problems where individuals must satisfy specific constraints for the problem.

Next, we discuss some examples of these techniques to control the generation of the individuals. Even if all of them are presented as a solution to graph matching, they can equally be applied to any other problem where individuals must satisfy any constraints.

Controlling directly the simulation step

Up to now in most of the problems where EDAs have been applied no constraints had to be taken into account. This is the reason why very few articles about modifying the simulation

step for this purpose can be found –some of them are for instance [Bengoetxea et al., 2000, 2002a] and [Santana and Ochoa, 1999]. In our inexact graph matching problem the nature of the individuals is special enough to require a modification of this simulation step. Two different ways of modifying the simulation step are introduced in this section.

However, it is important to note that altering the probabilities at the simulation step, whichever the way, implies that the learning of the algorithm is also denaturalized somehow. It is therefore very important to make sure that the manipulation is only performed to *guide* the generation of potentially incorrect individuals towards correct ones.

- **Last Time Manipulation (LTM): forcing the selection of values still to appear only at the end.** This method consists in not altering the simulation step during the generation of the individual until the number of vertices of G_M remaining to match and the number of variables to be simulated in the individual are equal. For instance, this could happen when three vertices of G_M have not been matched yet and the values of the last three variables have to be calculated for an individual. In this case, we will force the simulation step so that only these three values could be sampled in the next variable.

  In order to force the next variable of the individual to take only one of the values that have not still appeared, the value of the probabilities that are used to perform the simulation will be changed. In this way, we will set the probability of all the values already appeared in the individual to 0, and the probabilities of the values not still appeared will be modified accordingly. More formally, the procedure to generate an individual will follow the order $\boldsymbol{\pi} = (\pi(1), \pi(2), \ldots, \pi(|V_D|))$, that is, all the variables will be instantiated in the following order: $\left(X_{\pi(1)}, X_{\pi(2)}, \ldots, X_{\pi(|V_D|)}\right)$. If we are instantiating the m^{th} variable (i.e. we are sampling the variable $X_{\pi(m)}$), the following definitions apply: let be $VNO(V_M)^m = \{u_M^i \in V_M \mid \nexists X_j \in \boldsymbol{X} \;\; j \in \{\pi(1), \ldots, \pi(m-1)\}, \;\; X_j = i\}$ the set that contains all the vertices of G_M not yet matched in the individual in the previous $m-1$ steps (VNO stands for Vertices Not Obtained), $vns^m = |V_D| - m$ is the number of variables still to be simulated, $\theta_{\pi(m)lk}$ is the probability of the variable $X_{\pi(m)}$ to take the value $x^k_{\pi(m)}$ (its k^{th} value) knowing that its parents are already on their l^{th} possible combination of values (as $\boldsymbol{\pi}$ follows an ancestral ordering, we know that the parent variables of $X_{\pi(m)}$ have already been instantiated in the previous $m-1$ steps), and $P^m_{Indiv} = \sum_{k \mid u^k_M \in V_M \backslash VNO(V_M)^m} \theta_{\pi(m)lk}$. With these definitions, following this method we will only modify the $\theta_{\pi(m)lk}$ values when the condition $|VNO(V_M)^m| = vns^m$ is satisfied. When this is the case, the probability for the value $x^k_{\pi(m)}$ to appear next in the variable $X_{\pi(m)}$ of the individual knowing that its parents are in the l^{th} combination of values, $\theta^*_{\pi(m)lk}$, will be adapted as follows:

$$\theta^*_{\pi(m)lk} = \begin{cases} \theta_{\pi(m)lk} \cdot \frac{1}{1-P^m_{Indiv}} & \text{if } u_M^{\pi(m)} \in VNO(V_M)^m \\ 0 & \text{otherwise.} \end{cases} \tag{4.42}$$

  Once the probabilities have been modified, it is guaranteed that the only values assigned to $X_{\pi(m)}$ will be one of the vertices of V_M not still obtained (a vertex from $VNO(V_M)^m$), as the probability to obtain next any vertex from $V_M \backslash VNO(V_M)^m$ has been set to 0. This modification of the $\theta_{\pi(m)lk}$ has to be repeated for the rest of the variables of the individual, but taking into account that in the next step there is one more value which probability has to be set to 0, and thus P^m_{Indiv} must be computed again. Following this

Adaptation 1 of PLS: Last Time Manipulation (LTM)

Definitions

vns^m: number of variables not simulated before the m^{th} step

$VNO(V_M)^m$: set of vertices of V_M not matched by simulation before the m^{th} step

P^m_{Indiv}: sum of the probabilities to appear next in the individual any vertex of V_M already matched before the m^{th} step

θ_{ijk}: probability to appear the value k next in the variable X_i knowing that the values defined in the individual for its parents are on their j^{th} possible combination of values

Procedure

Find an ancestral ordering, $\boldsymbol{\pi}$, of the nodes in the Bayesian network

For $m = \pi(1), \pi(2), \ldots, \pi(|V_D|)$ (number of variables, to sample following the ancestral ordering $\boldsymbol{\pi}$)

If $(|VNO(V_M)^m| == vns^m)$

For $k = 1, 2, \ldots, |V_M|$ (number of values for each variable)

Modify the probabilities: the modified probability for the value k to appear next in the variable $X_{\pi(m)}$ of the individual for the l^{th} combination of values of its parents, $\theta^*_{\pi(m)lk}$, is

$$\theta^*_{\pi(m)lk} = \begin{cases} \theta_{\pi(m)lk} \cdot \frac{1}{1-P^m_{Indiv}} & \text{if } u_M^{\pi(m)} \in VNO(V_M)^m \\ 0 & \text{otherwise} \end{cases}$$

where $P^m_{Indiv} = \sum_{k \mid u^k_M \in V_M \setminus VNO(V_M)^m} \theta_{ilk}$, and l is the combination of values of the parent variables of $X_{\pi(m)}$ (which have been previously instantiated in the previous $m-1$ iterations)

$X_{\pi(m)} \leftarrow$ generate a value from $\theta^*_{\pi(m)lk} = p\left(X_{\pi(m)} = k \mid \boldsymbol{pa}^l_{\pi(m)}\right)$

Else

$X_{\pi(m)} \leftarrow$ generate a value from $\theta_{\pi(m)lk} = p\left(X_{\pi(m)} = k \mid \boldsymbol{pa}^l_{\pi(m)}\right)$

Figure 4.16: Pseudocode for Last Time Manipulation (LTM).

method, at the last step ($m = |V_D|$), only one value v will have its probability set to $\theta^*_{\pi(m)lv} = 1$ and for the rest of the values $\theta^*_{\pi(m)lw} = 0 \ \ \forall w \neq v$. Therefore, the only value that will be assigned to the variable $X_{|V_D|}$ will be v.

This technique does not modify the probabilities of the variables in any way until $|VNO(V_M)^m| = vns^m$. Therefore, the simulation step will remain as it is, without any external manipulation unless the latter condition is satisfied. However, when that condition is satisfied the method will modify the actual distribution that the values will follow.

Figure 4.16 shows the pseudocode of this first adaptation of PLS. A detailed example of LTM can also be found in Appendix B.

- **All Time Manipulation (ATM): increasing the probability of the values not appeared from the beginning.** This second technique is another way of manipulating the probabilities of the values for each variable within the individual, but this

time the manipulation takes place not only at the end but from the beginning of the generation of the individual. The value of the probabilities remains unaltered only after all the possible values of the variables have already appeared in the individual (that is, when $VNO(V_M) = \emptyset$).

For this, again the order of sampling the variables $\boldsymbol{\pi}$ will be followed, instantiating them in the same order $\left(X_{\pi(1)}, X_{\pi(2)}, \ldots, X_{\pi(|V_D|)}\right)$. Every step the probabilities of a variable will be modified before its instantiation. The required definitions for the m^{th} step (the sampling of the variable $X_{\pi(m)}$) are as follows: let $|V_D|$ be number of variables of each individual, let also be $VNO(V_M)^m$, vns^m, and $\theta_{\pi(m)lk}$ as defined before. The latter probability will be modified with this method obtaining the new $\theta^*_{\pi(m)lk}$ as follows:

$$\theta^*_{\pi(m)lk} = \begin{cases} \theta_{\pi(m)lk} \cdot \frac{K - P^m_{Indiv}}{K \cdot \left(1 - P^m_{Indiv}\right)} & \text{if } u^i_M \in VNO(V_M)^m \text{ and } |VNO(V_M)^m| \neq vns^m \\ \frac{\theta_{\pi(m)lk}}{K} & \text{if } u^i_M \notin VNO(V_M)^m \text{ and } |VNO(V_M)^m| \neq vns^m \\ \theta_{\pi(m)lk} \cdot \frac{1}{1 - P^m_{Indiv}} & \text{if } u^i_M \in VNO(V_M)^m \text{ and } |VNO(V_M)^m| = vns^m \\ 0 & \text{if } u^i_M \notin VNO(V_M)^m \text{ and } |VNO(V_M)^m| = vns^m \end{cases} \tag{4.43}$$

where $K = \left\lceil \frac{N - vns^m}{vns^m - |VNO(V_M)^m|} \right\rceil$, and $P^m_{Indiv} = \sum_{u^k_m \in V_M \setminus VNO(V_M)^m} \theta_{\pi(m)lk}$.

In fact, the two last cases are defined only to avoid the division by zero problem, but these two cases can also be calculated when using limits, as the $|VNO(V_M)^m| = vns^m$ case can be understood as the limit when $K \longrightarrow \infty$:

$$\begin{aligned} |VNS(V_D)| &= |VNO(V_M)| \Rightarrow \\ \theta^*_{ijk} &= \lim_{K \to \infty} \left(\theta_{ijk} \cdot \frac{K - P_{Indiv}}{K \cdot (1 - P_{Indiv})} \right) = \\ &= \theta_{ijk} \cdot \lim_{K \to \infty} \left(\frac{1 - \frac{P_{Indiv}}{K}}{1 - P_{Indiv}} \right) = \theta_{ijk} \cdot \frac{1}{1 - P_{Indiv}} \end{aligned}$$

and

$$\begin{aligned} |VNS(V_D)| &= |VNO(V_M)| \Rightarrow \\ \theta^*_{ijk} &= \lim_{K \to \infty} \left(\frac{\theta_{ijk}}{K} \right) = 0 \end{aligned}$$

The reason to modify the probabilities in such a manner is that at the beginning, when vns^m is much bigger than $|VNO(V_M)^m|$, there is still time for all the values to appear in the individual, and thus the probabilities are not modified very much. Only when $|VNO(V_M)^m|$ starts to be very close to vns^m will the effect of the manipulation be stronger, meaning that there are not much variables to be instantiated regarding the values not appeared yet on the individual. Finally, when $|VNO(V_M)^m| = vns^m$, there is no chance to leave the probabilities as they are, and only the values not appeared yet have to be selected. For this, the probabilities of the values already appeared are set to 0, and the other ones are modified in the same way as in the previous method.

Adaptation 2 of PLS: All Time Manipulation (ATM)

Definitions

vns^m: number of variables not simulated before the m^{th} step.

$VNO(V_M)^m$: set of vertices of V_M not matched by simulation before the m^{th} step

P^m_{Indiv}: sum of the probabilities to appear next in the individual any vertex of V_M already matched before the m^{th} step

θ_{ijk}: probability to appear the value k next in the variable X_i knowing that the values defined in the individual for its parents are on their j^{th} possible combination of values.

Procedure

Find an ancestral ordering, $\boldsymbol{\pi}$, of the nodes in the Bayesian network

For $m = \pi(1), \pi(2), \ldots, \pi(|V_D|)$ (number of variables, to sample following the ancestral ordering $\boldsymbol{\pi}$)

If $(|VNO(V_M)^m| > 0)$

For $k = 1, 2, \ldots, |V_M|$ (number of values for each variable)

Modify the probabilities: the modified probability for the value k to appear next in the variable $X_{\pi(m)}$ of the individual for the l^{th} combination of values of its parents, $\theta^*_{\pi(m)lk}$, is

$$\theta^*_{\pi(m)lk} = \begin{cases} \theta_{\pi(m)lk} \cdot \frac{K - P^m_{Indiv}}{K \cdot \left(1 - P^m_{Indiv}\right)} & \text{if } u^i_M \in VNO(V_M)^m \text{ and } |VNO(V_M)^m| \neq vns^m \\ \frac{\theta_{\pi(m)lk}}{K} & \text{if } u^i_M \notin VNO(V_M)^m \text{ and } |VNO(V_M)^m| \neq vns^m \\ \theta_{\pi(m)lk} \cdot \frac{1}{1 - P^m_{Indiv}} & \text{if } u^i_M \in VNO(V_M)^m \text{ and } |VNO(V_M)^m| = vns^m \\ 0 & \text{if } u^i_M \notin VNO(V_M)^m \text{ and } |VNO(V_M)^m| = vns^m \end{cases}$$

where $K = \left\lceil \frac{N-m}{m-n} \right\rceil$, l is the combination of values of the parent variables of $X_{\pi(m)}$ (which have been previously instantiated in the previous $m-1$ iterations), and $P^m_{Indiv} = \sum_{k \mid u^k_M \in V_M \setminus VNO(V_M)^m} \theta_{ilk}$

$X_{\pi(m)} \leftarrow$ generate a value from $\theta^*_{\pi(m)lk} = p\left(X_{\pi(m)} = k \mid \boldsymbol{pa}^l_{\pi(m)}\right)$

Else

$X_{\pi(m)} \leftarrow$ generate a value from $\theta_{\pi(m)lk} = p\left(X_{\pi(m)} = k \mid \boldsymbol{pa}^l_{\pi(m)}\right)$

Figure 4.17: Pseudocode for All Time Manipulation (ATM).

Figure 4.17 shows the pseudocode of this second adaptation of the simulation. A detailed example of ATM can also be found in Appendix B.

This second technique modifies the probabilities nearly from the beginning, giving more chance to the values not already appeared, but it also takes into account the probabilities learned by the Bayesian network in the learning step. It does not modify the probabilities in any way when $|VNO(V_M)^m| = 0$, that is, when all the values have

already appeared in the individual.

Correction a posteriori

This technique is completely different from the ones proposed before, as it is not based on modifying the probabilities generated by the algorithm at all: the idea is to correct the individuals that do not contain an acceptable solution to the problem after they have been completely generated. In order to do this correction, once the individual has been completely generated and has been identified as not correct ($|VNO(V_M)^{|V_D|}| > 0$), a variable which contains a value that appears more than once in the individual is chosen randomly and substituted by one of the missing values. This task is performed $|VNO(V_M)^{|V_D|}|$ times, that is, until the individual is correct.

The fact that no modification is done at all in the learned probabilities means that this method does not demerit the learning process, and thus the learning process is respected as when using PLS. As the generation of the individuals is not modified at all with respect to PLS, the only manipulation occurs on the wrong individuals, and the algorithm can be supposed to require less generations to converge to the final solution. Furthermore, this method can also be used with other evolutionary computation techniques such as GAs.

Changing the fitness function

This last method is not based neither on modification of the probabilities during the process of the generation of the new individuals nor in adapting them later before adding them to the population. Instead, the idea is completely different and consists in applying a penalization in the fitness value of each individual.

The penalization has to be designed specifically for each problem in order to avoid unexpected results. For instance, on the experiments carried out with this technique and commented later in Section 6.2, the penalization has been defined as follows: if $f(\boldsymbol{x})$ is the value obtained by the fitness function for the individual $\boldsymbol{x}= (x_1, \dots x_{|V_D|})$, and if $|VNO(V_M)^{|V_D|}|$ is the number of vertices of G_M not present in the individual, the modified fitness value, $f^*(\boldsymbol{x})$ will be changed as follows:

$$f^*(\boldsymbol{x}) = \frac{f(\boldsymbol{x})}{|VNO(V_M)^{|V_D|}| + 1}. \tag{4.44}$$

Another important difference regarding the other methods to control the generation of individuals explained so far is that the penalization does allow the generation of incorrect individuals, and therefore these will still appear in the successive generations. This aspect needs to be analyzed for every problem when a penalization is applied. Nevertheless, as these incorrect individuals will be given a lower fitness value, it is expected that their number will be reduced in future generations. It is therefore important to ensure that the penalization applied to the problem is strong enough. On the other hand, the existence of these individuals can be regarded as a way to avoid local maxima, expecting that, starting from them, fittest correct individuals would be found.

4.5.2 Continuous domains

Similarly as in Section 4.5.1, we will define the graph matching problem using a particular notation for continuous EDAs.

The selected representation of individuals for continuous domains was introduced previously in Section 3.3.2. These individuals are formed of only continuous values, and as

already explained no combination of values from both the discrete and continuous domains is considered in this thesis. As explained in that section, in order to compute the fitness value of each individual, a previous translation step is required from the continuous representation to a permutation-based discrete one (as shown in Figure 3.2), and finally in a second step from there to a representation in the discrete domain like the one explained in Section 3.3 –the latter procedure has already been described in Figure 3.1. All these translations have to be performed for each individual, and therefore these operations result in an overhead in the execution time which makes the evaluation time longer for the discrete domain when using this method. It is important to note that the meaning of the values for each variable is not directly linked to the solution the individual symbolizes, and therefore we will not refer directly to graphs G_M and G_D in this section.

Section 3.3.2 defined that the size of these continuous individuals is $n = |V_D|$, that is, there are $\boldsymbol{X} = \left(X_1, \ldots, X_{|V_D|}\right)$ continuous variables, each of them having any value in a range of (-100, 100) for instance. We denote by x_i the values that has the i^{th} variable, X_i.

Estimating the density function

We propose different continuous EDAs belonging to different categories to be used in inexact graph matching. As explained in the previous sections, these algorithms are expected to increase their complexity when more complex learning algorithms are used and when more complicated structures are allowed to be learned. Again, these algorithms should be regarded as representatives of their categories introduced in Section 4.4.1: (1) UMDA$_c$ [Larrañaga et al., 2000, 2001] as an example of an EDA that considers no interdependencies between the variables; (2) MIMIC$_c$ [Larrañaga et al., 2000, 2001] which is an example that belongs to the category of pairwise dependencies; and finally, (3) EGNA$_{BGe}$ and EGNA$_{BIC}$ [Larrañaga et al., 2000, Larrañaga and Lozano, 2001] as an example of the category of EDAs where multiple interdependencies are allowed between the variables, where no limits are imposed on the complexity of the model to learn. It is important to note that any other algorithm mentioned in Section 4.4.1 could perfectly have been chosen as a representative of its category instead of the ones we have selected.

It is important to note that even if EGNA$_{BGe}$ is expected to obtain better results, the fact of not setting limits to the number of interdependencies that a variable can have could also mean that the best structure is a simple one. For instance, a structure learned with EGNA$_{BGe}$ could perfectly contain at most pairwise dependencies. This fact is completely dependent on the complexity of the graph matching problem applied in our case. This means that in such cases the difference in results obtained with EGNA$_{BGe}$ and MIMIC$_c$ would not show significant differences, while the computation time in EGNA$_{BGe}$ will be significantly higher.

Adapting the simulation scheme to obtain correct individuals

In the discrete domain the need to introduce adaptations in the simulation arises because of the additional constraint in our proposed graph matching problems for all the solutions to contain all the possible values of V_M.

However, in the case of the continuous domain no such restriction exists, and the fact of using a procedure to convert each continuous individual to a permutation of discrete values ensures that the additional constraint is always satisfied for any continuous individual. Therefore, all the possible continuous individuals in the search space do not require to any

adaptation in the simulation step, and therefore no measures need to be introduced at this point.

Chapter 5

Parallel estimation of distribution algorithms

'I do not fear computers. I fear the lack of them.'

Isaac Asimov

5.1 Introduction

The reduction in the execution time is a factor that becomes very important when using many applications nowadays. In spite of the enhancement on computer hardware, this problem is always happening, as the increase in power is always related to the application of new methods that were also too time consuming in previous computer systems. Parallel programming techniques provide feasibility of solving new problems or longer size ones.

The computation time is sometimes forgotten or avoided in some research works, as the validity of new methods or algorithms among others is in many times the most important point to be considered in the first stages. However, if one wants to apply parallelism techniques in a real life application (or even to commercialize it) the fact of reducing the computation time of the program becomes an important aspect to take into account. When willing to make faster the execution of a program, specially for algorithms that are very computationally consuming, we have four main choices:

1. We can optimize the code so that it does not repeat tasks or that minimizes the time of accessing slow functions such as disk access.

2. We can improve the hardware on which the algorithm is executing. This is done sometimes by recording the whole program in a ROM-type chip, but also by buying a faster processor, adding more processors to the computer, or by increasing the size of RAM memory of the computer.

3. Compilers that will convert automatically a sequential source program into a parallel one have also been proposed, but nowadays they mostly work for easily parallelizable programs such as vector and matrix operations.

4. We can rewrite the code making use of parallelization techniques.

From all these solutions, the first one can improve considerably the computation time of a program, but its main drawbacks are that this reduction has a limit and that highly

optimized code is normally very difficult to maintain. The second solution is always possible, but it requires an important economic cost depending on how much we want the hardware to be optimized. In addition, it is important to analyze which is the hardware resource that makes the program go slower, as many times users tend to invest money in adding more and faster processors when the fact is that most of the time the bottleneck is the lack of RAM memory[1]. The third solution implies the use of high performance compilers that are able to provide parallel versions of a sequential source code in order to allow execution in different processors at the same time. These compilers use techniques to detect loops within the code where matrix or vector operations are performed, and they apply special techniques for parallelization. Unfortunately, these compilers are not *intelligent* enough to parallelize every sequential program and to detect data communication and computation parts within them that could be parallelized, and as a result these will not be a solution in our case. Furthermore, these compilers are usually very hardware dependent and appear to be very expensive. The interested reader can find more information about this subject in [Polaris, 1994, Wolfe, 1996].

Taking all the aforementioned reasons into account, we will concentrate on the last solution, the application of parallelization techniques. These are very powerful and effective for most of the cases, and they are often very easy to be applied to existing sequential programs. In addition, the proliferation of dedicated parallel libraries makes the application of parallelization techniques to be quite easy for a beginner on this field. This chapter intends to explain the basics of the existent parallelism techniques, the state of the art of the parallelization field, as well as to give an introduction to the most important ones using as an example the EDA program itself.

5.2 Sequential programs and parallelization

5.2.1 The design of parallel programs

Different techniques and methodologies can be applied for creating parallel solutions. However, the best parallel solution that can be developed is usually quite different from the sequential one. It is of fundamental importance to choose the proper methodology for parallel design in order to obtain the best parallel approach, in which concurrency aspects are taken into account in depth before focusing on machine-dependent issues.

In [Foster, 1995] one of the many possible parallel design methodologies is introduced. This methodology contains four distinct stages that are performed one after another: partitioning, communication, agglomeration, and mapping. During the first and second stages the programmer focuses on concurrency and scalability, while in the third and fourth stages the attention is on locality and other performance-related issues. These four stages are illustrated in Figure 5.1, and are explained as follows:

Partitioning: the parts on the problem that can be parallelized are identified and selected. This task is performed independently of the hardware type available, and issues such as the number of processors available are not taken into account at this stage.

[1]The lack of enough RAM memory forces the computer to work into the hard disk. The disk access time is measured on the order of milliseconds, while RAM memory is in the order of nanoseconds. Therefore this factor implies a considerable delay on the overall execution time of the whole program and the operating system.

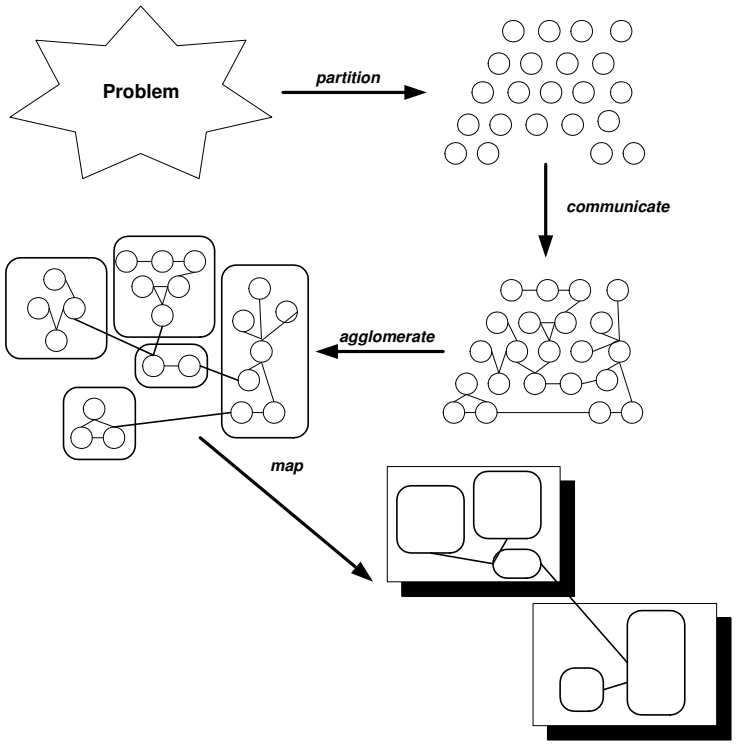


Figure 5.1: Parallel design methodology when parallelizing programs, as described in [Foster, 1995]. The four stages are illustrated: starting from a problem specification, (1) the problem is partitioned in smaller tasks that are divided in processes, (2) communication requirements between the different processes are determined, (3) processes are agglomerated, and finally (4) processes are mapped to processors.

Communication: the communication that is required to coordinate all the processes is identified and analyzed. Communication data structures and proper communication protocols are defined.

Agglomeration: this stage focuses on performance requirements and implementation costs. Some of the processes are combined into larger groups in order to improve performance and implementation costs.

Mapping: processes are organized regarding the number of processors available. The main objectives are to balance the work load of each process and to minimize communication costs.

Usually, parallel programs will create and destroy dynamically processes in order to obtain a balance in work load on amount of processes between processors. The design of parallel algorithms is presented here as sequential work, but often considerations at some stages require reconsidering previously designed aspects. Next, these four steps will be discussed in more detail.

Partitioning the problem

The main objective of the partition step is to divide the problem in the smallest possible tasks, in order to obtain what is called a *fine-grained* decomposition of the problem[2].

Partition is applied to both the whole computation job and the data associated to the processes. Usually, programmers focus first on the data partition (i.e. the way of partitioning the data is determined) and finally processes are associated to the partitioned data. This partitioning technique is known as *domain decomposition*.

Another approach is to focus first in partitioning the global job in processes and to work out afterwards which is the best way to partition the data. This is called *functional decomposition*.

These two techniques are complementary, and they can be applied to different parts of a single problem or even both can be applied at the same time to the whole problem in order to obtain different parallel algorithms to solve a single problem.

Process communication schemes in parallel programs

After partitioning the problem in sub-tasks assigned to processes, the programmer has to determine the way in which all these working processes will be coordinated and organized. As processes will require to receive information associated to another processes, the information flow between all the processes has to be analyzed with care. This task is performed in the communication phase of the parallel program design. For this, it is essential to take into account all the synchronization and communication aspects between all the processes.

A general idea to understand how to coordinate the different processes collaborating for a global job is the use of a producer-consumer scheme, which is a general approach of how two types of processes can be organized. The general case is the one in which there are a set of processes playing the role of the producers, where they produce elements that are stored in a buffer or an intermediate work pool. The rest of the processes will play the role of the consumer, which will read elements from the buffer or work pool and will use them for some determined task. Figure 5.2 illustrates this approach.

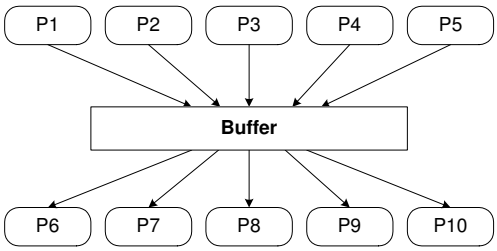


Figure 5.2: The producer-consumer approach.

However, there are also many other ways to organize the communication and synchronization between all the processes. In any case, it is always important to determine the nature of the communication channel to select the best way of communicating processes. In [Foster, 1995], these aspects are focused on the communication, of which different types are classified taking into account four orthogonal axes:

[2]A fine-grained decomposition represents the greatest flexibility possible to parallelize programs, as it divides the whole problem in the maximum number of tasks.

Local/Global: in local communication each process communicates with a small set of processes. In global communication it communicates with the rest of them.

Structured/Unstructured: in structured communication, all the processes form a regular structure when communicating. Regular structures can be a tree, a pipeline, or a grid for instance. On the other hand, in unstructured communication arbitrary graphs are formed.

Static/Dynamic: the identity of the communicating partner processes is always the same in static communication. When it is dynamic, the identity of the communicating partners can be different each time.

Synchronous/Asynchronous: in synchronous communication, there are producers and consumers of messages that execute in a coordinated way, with producer/consumer pairs cooperating in data transfer operation. In asynchronous communication, a consumer can obtain its data without the collaboration of the producer.

There are many working schemes that could be chosen for coordinating the execution of all the processes. None of them is better than the others for all the practical cases, and the programmer has to decide which is the one that best fits the organization of each parallel program.

Next, a short review of five typical and basic working schemes for organizing processes in parallel programming is given. Each of them is applied in those tasks in which their use has been shown to be the most effective. The programmer should be aware of all these possibilities and choose the one that he considers to be the optimum, as this choice as well as the information flow is of fundamental importance to obtain a satisfactory performance on parallel programs. A bad design could lead to a dramatic worsen on the performance of the whole program, resulting in some cases in even worse execution times than with a sequential program.

Generally speaking, the different basic working schemes are the following: phase parallel, divide and conquer, pipeline, master-slave, and work pool.

Phase parallel. A parallel program that follows this method will be divided in a series of two main steps. In the computation step, each of the processes will perform an independent computation in parallel. In the following step, all the processes will synchronize (either by using lock variables, semaphores or any other blocking communication method, see Section C.3 in Appendix C) and all the results will be gathered. Figure 5.3 shows this approach.

Divide and conquer. This algorithm for parallel programming is very similar to its sequential homologous as it can be appreciated in Figure 5.4. A parent process divides its computing weight in many smaller parts an it assigns them to a number of child processes. The children processes will proceed similarly, and then they will gather their children's results and send them back to their parent. This set of division on the job and gathering and sending of results is made recursively. The main drawback of this method is that a balance is required for an equitable division of tasks among processes. When using this scheme it is important to consider the dynamic nature of the problem, as well as to analyze the possibility of fully parallelizing it.

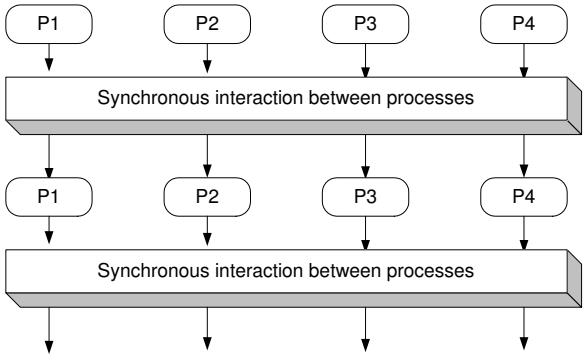


Figure 5.3: The phase parallel approach.

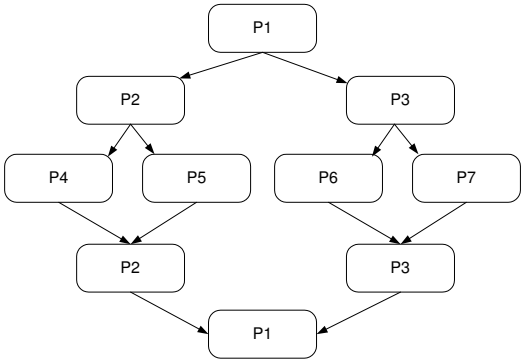


Figure 5.4: The divide and conquer approach.

Pipeline. In this approach, all the processes form a chain as shown in Figure 5.5. This chain is feed with a continuous flow of data, and processes execute the computations associated to each different step on the pipeline, one after another for each data-unit, but all work at the same time on different data-units. This behavior is similar to the execution of instructions in a segmented processor, in which it is important to take into account the dependence types of the input data.

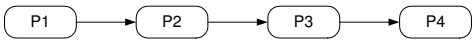


Figure 5.5: The pipeline approach.

Master-slave. As already explained, this working scheme –also known as *manager-worker* or *process farm*– is among the most applied. A process takes the role of the master or manager, executes parts of the global job that cannot be parallelized, and divides and sends to the rest of slave or worker processes the part of the global job that can be executed in parallel. This approach is illustrated in Figure 5.6. When a slave or worker process finishes its task, it sends back to the master the results obtained. Afterwards, the master is then sending more work to the slave. The main drawback is that the master process coordinates the whole information exchange, which in some cases results in a bottleneck.

In some particular problems, the master also performs part of the job instead of simply waiting for the rest of the workers, and therefore it also plays at the same time the role of a worker.

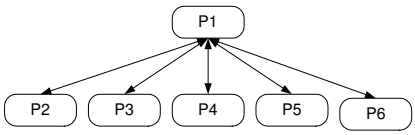


Figure 5.6: The master-slave approach.

Work pool. This method is often used with the model of shared variables. Figure 5.7 shows that this approach requires a global data-structure that is used as a work pool. Initially some basic amount of work has to be added to this pool. The main difference between this model and the previous master-slave one is that in this case all the processes are of the same type, as there is no master process on the scheme. All the processes that take part in the job can access it and produce: (1) no work, (2) a part of the work that will be placed on the pool, or (3) many parts of the work that will also be placed on the pool.

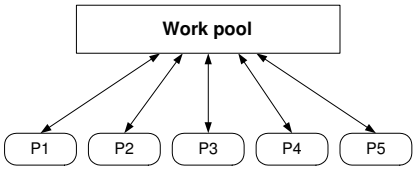


Figure 5.7: The work pool approach.

The parallel part of the program finishes when the work pool is empty. This method makes easier to balance the work load between all the processes. However, it is not easy to implement it when the message passing model is to be used (see Section C.3.5 in Appendix C) as this model does not allow an efficient access to the shared structure. The pool can be implemented as an unordered set, or an ordinary queue with or without priority.

Agglomeration

The agglomeration phase focuses on the performance of the system that has been designed so far: for instance, the fact of having many more processes than processors is highly inefficient in terms of execution time, as most of the time the system will not be able to keep executing all the processes at the same time. When agglomerating, we move from the theoretical design phase towards the realistic one, where the hardware resources available will be taken into account. The agglomeration is therefore the opposite step of the partitioning phase, in which the fine-grained solution is revised and the number of global tasks reduced. In addition to the reduction in the number of processes, in the agglomeration phase the replication of the data between different tasks will be determined to be worthwhile or not in terms of efficiency. In brief, the two objectives for the agglomeration phase is firstly to reduce the number of processes (by combining them and creating larger ones), and secondly to provide an appropriated number of them so that the processors can deal with, and to reduce communication costs.

Even if the number of tasks will be reduced in this step, it is usually the case that at the end more processes than processors will be scheduled. The reason for this is that processes will make use of other resources such as input/output devices: processes that are waiting for input/output devices will leave their processor free for a time, and in the meanwhile another process could make a profit and advance in its job.

Mapping

Finally, the mapping phase will distribute the processes among the available processors. The mapping phase does not arise on single-processor or on shared-memory computers that provide automatic task scheduling.

The goal of mapping algorithms is to minimize execution time. Two strategies are used for achieving this goal:

1. Processes that are able to execute independently are executed in different processors, in order to increase concurrency.

2. Processes that communicate frequently are assigned to the same processor, in order to increase locality.

Unfortunately, these two strategies sometimes conflict between them. In addition, the fact that usually there are more processes than processors is another factor that makes the mapping problem more complex.

There are several algorithms, such as load-balancing algorithms, that provide solutions to the mapping problem. This problem is known to be NP-hard, and discussion on this topic is out of the scope of this thesis. The interested reader is referred to [Foster, 1995] for more information on this topic.

5.2.2 Parallelizing an already existing sequential program

The algorithm to design parallel programs introduced in the previous section is suitable when an algorithm has to been redesigned from scratch. However, in many cases the programmer has already a sequential program to solve the problem that is executing too slowly just because of specific bottlenecks at different stages of the algorithm.

When a sequential version of the program is available, and we have access to its source code, it is not necessary to apply the four stages of the previous design model. Instead, a common practice is to identify the parts in the code that represent the most important bottlenecks in the program and to apply parallelization mechanisms to these parts.

Identifying bottlenecks in programs can be a difficult task if we do not know exactly how the program behaves. In addition, sometimes different input data could lead to very different CPU-time requirements of the different routines of the sequential algorithm. Hopefully, there are many tools to perform an analysis of the execution time rate of each of the routines on the program. An example of these tools is the *gprof*, which is a GNU tool that records all the required information for an exhaustive analysis. This tool is used together with the *gcc* ANSI C++ compiler, and an example of its application is shown later in Section 5.5.2. Such a tool helps the programmer to determine the routines that constitute the main bottlenecks in the program, which should be parallelized.

Once the routines to be parallelized have been identified, we could use one of the many communication and synchronization paradigms available. Many of these are described in

Section C.3.6 in Appendix C. For these, the task that the selected routines performs is divided in subtasks and, as well as in the previous section, communication and synchronization between them is determined and implemented.

An example of a way of coordinating the different processes collaborating for a global job is the use of a producer-consumer scheme, which was introduced in Section 5.2.1. Very often this approach is used for the case of a single producer and one or more consumers. It is important to mention that the client-server scheme is also a particular case of the producer-consumer: we could assume that the producers are clients that basically *produce* requests and store them in a structure such as a buffer, and the server will play the role of *consuming* requests by attending them.

Section 5.5 shows an example on how to apply these techniques for the case of having already a sequential program in which routines that represents bottlenecks are identified and parallelized.

5.3 Parallel architectures and systems

5.3.1 Parallel computer architectures

As the performance of any parallel program is heavily dependent on the hardware available, it is important to have a general understanding of the existing parallel machines in order to select the best one for our purpose. Obviously, one can always execute parallel programs on an ordinary PC or workstation with a single CPU and its particular memory, but the fact of having more than 2 or 4 processes or threads will lead to a competition between all of them for the use of the CPU rather than to the desired collaboration between them. The drastic reduction in computer prices in the last years has made possible for more users to acquire machines with more than a processor. Moreover, the existence of very fast local area networks allows single processor computers to communicate with each other with rates similar as communication within an internal bus of a machine.

We present in this section a classification of the different parallel systems. The classification is done based on the number of processors and the arrangement of CPUs and memory within the different parallel systems. One of the first classifications is the one based on Flynn's specification [Flynn, 1972]. An illustration of this classification is shown in Figure 5.8. Following this general classification, the main difference between computer systems is done regarding the number of processors and their type in the next way:

Single Instruction Single Data (SISD): This is an ordinary workstation system, where there is a single CPU and a single address map accessible by the CPU. This is not considered as a parallel system.

Single Instruction Multiple Data (SIMD): This type of systems can be considered as a first approach to parallel computing, and they are nowadays disappearing. In a typical SIMD machine we can have hundreds of CPUs, even thousands, all of them with a small private memory space. They all execute at the same time the same instruction over different data (at least, if the instruction makes it possible). These systems were thought for parallel computation of vector and matrix operations. When a CPU requires data stored in another's memory address map an explicit communication procedure has to be executed before. The main problems of these machines are their inflexibility as well as their high dependence on synchronization between all the CPUs of the system.

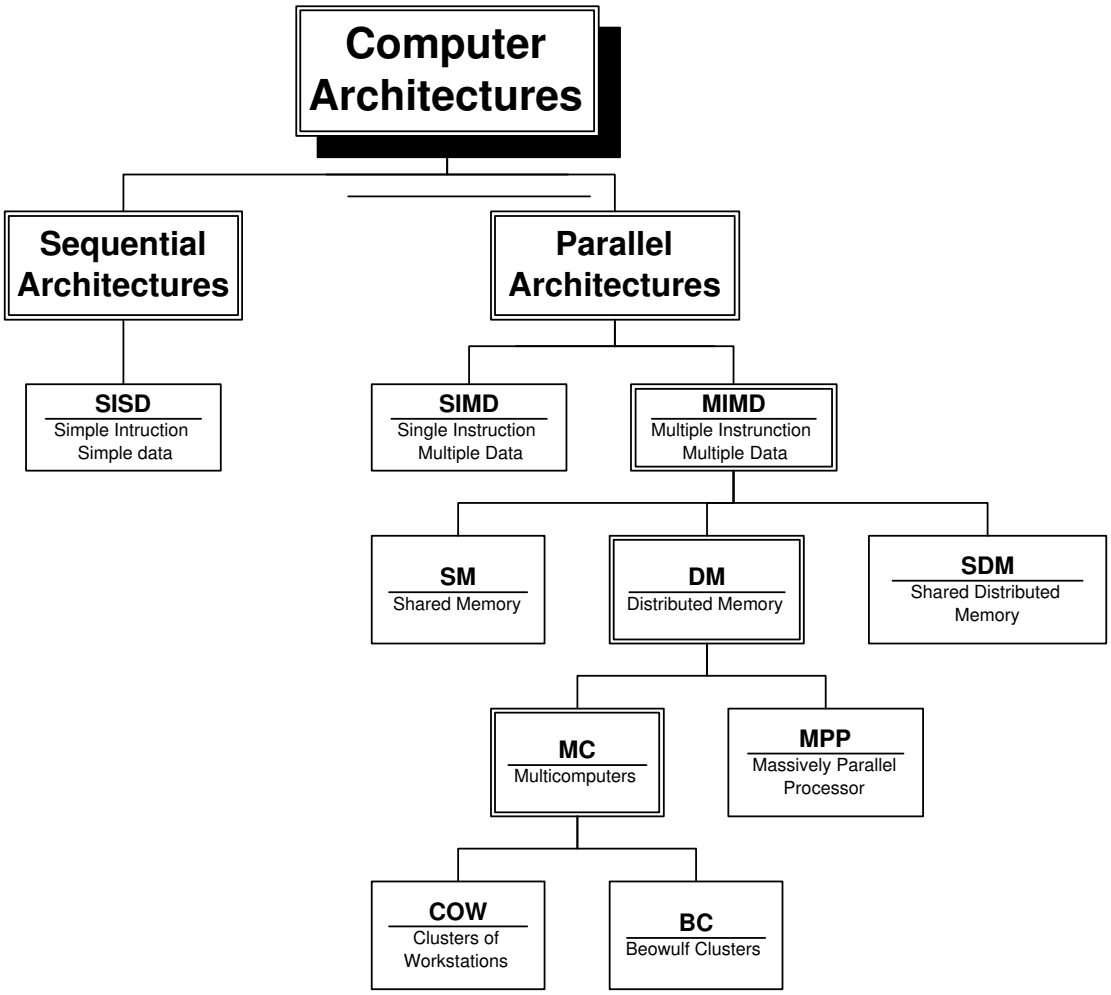


Figure 5.8: Illustration of all the different computer architectures. Here all the possible forms are shown, and many of them exist today. In some of these systems the number of processors is just one, but in other there can be thousands of them.

Multiple Instruction Multiple Data (MIMD): In this approach with many CPUs, all the processors have their particular memory space too, but the way of execution is very asynchronous and each processor can execute a different instruction, or even a different program. There is a complete independence in execution between all the CPUs. These systems are very convenient for nowadays's parallel systems. A diagram of the most used memory models is also presented in Figure 5.9. MIMD systems' performance shows a high dependence on the memory architecture, and depending on it we can classify these systems as follows:

Shared memory. The system has a single memory space, so that any computer can access any local or remote memory in the system, independently of the process that is the owner. Having a single memory space allows having shared memory for communication between all the executing workers or tasks, which makes it be an efficient communication mechanism. This is very convenient for parallel programs where the amount of data to process is very big, as the data requires no copying in order for all the CPUs to access it at the same time. In other

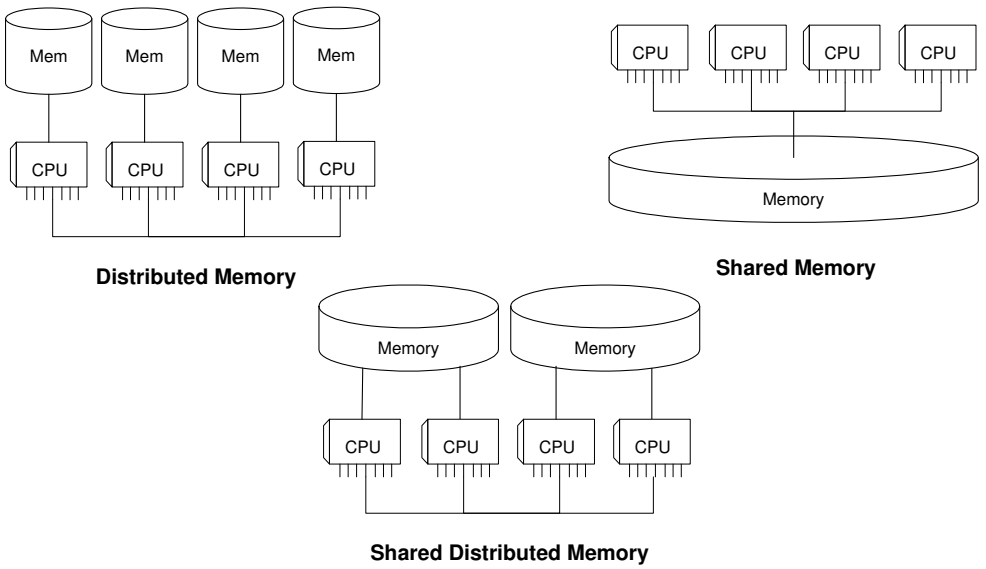


Figure 5.9: Illustration of the most used memory models for parallel computer architecture taxonomy.

cases, all this data should be exchanged between CPUs' address maps during the execution of the parallel program with the consequent lack on efficiency. An aspect to remember is that the disadvantage of using shared memory access is that shared variables lead to the existence of race conditions between parallel programs or routines, and therefore synchronization mechanisms have to be included in our parallel programs.

Distributed memory. In this case each processor has a particular address map that the rest cannot access at all. It is therefore essential that a process sends all the data to another if they have to collaborate to compute something. This communication operations are explicit (i.e. they have to be programmed). These systems have the advantage of being very easily scalable, but the lack of a shared address map adds a time penalty for each inter-process communication. An example of this type of computer systems are the so called Massively Parallel Processors (MPP).

Shared distributed memory. This hybrid model has the goal of having the advantages of both systems (i.e. the easy communication mechanism of shared memory and the scalability of distributed systems). These systems would contain their own memory address map, but the architecture is designed so that any node can access the memory of another with a slight time penalty. This type of architecture is a current trend on multiprocessor architectures.

In the recent years the development of two main fields in computing, such as the faster processors and the improvements of networks in communication speed, have also made available some new computer systems within the distributed memory MIMD model that could be regarded as *virtual parallel computers*. It is important to distinguish properly between three different possible machines that follow the MIMD model:

Network of workstations: in this model many very fast separated computers are con-

nected through a fast Local Area Network (LAN). Each computer has its own console, can work as an independent machine, and occasionally they can all collaborate in a common task. This type of systems are known as NOW (Network Of Workstations).

Cluster: clusters are made of different machines, but there is a single console to control all the computers. However, in order to have full control of each of the nodes, it is necessary to do a connection through the console.

Multicomputers: in multicomputers all the different computers behave as a single one, and the user or programmer can control the whole execution of all the CPUs from a single node. No connection is necessary to a single node in order to control what is going on in the multicomputer.

This type of MIMD model machines are mainly built with clusters of workstations, where each workstation in the cluster acts as a separate computer. Processes on these systems cannot use shared memory, as the memory space of every workstation is physically separated from the others, and therefore message passing primitives are used for inter-process communication between processes executing in different workstations. In some special cases, these systems also are able to use virtual shared memory.

However, there are also special types of computer clusters composed of ordinary hardware architectures (i.e. ordinary PCs) with public domain software (i.e. Linux OS and dedicated libraries designed for fast message passing). A node plays the role of the server and controls the whole cluster, serving files to the rest of the nodes (i.e. the client-nodes). In addition, in some cases a node can also be a shared memory system (a multiprocessor). This particular type of clusters are known as Beowulf Clusters (BC) [Beowulf, 1994].

The main advantage of NOWs is their lower cost compared to other parallel machines, scalability, and code portability. The main drawback is the lack of available software specially designed for this type of systems in order to make the cluster behave as a single virtual machine. However, the existence of specialized public domain libraries such as Parallel Virtual Machine (PVM) and implementations of the Message Passing Interface (MPI) makes programmers easier to work with them (later in Section 5.3.3 these two libraries are analyzed in detail).

Finally, it is worth mentioning how typically all these parallel systems are combined with the different communication and synchronization methods. Typically, when shared memory is available the selected communication approach is to use shared buffers and variables that all execution units can access using as a result an explicit synchronization mechanism such as mutex semaphores. On the other hand, in distributed systems only message passing primitives are possible for communication between execution units, and synchronization is implicit on these primitives. Table 5.1 illustrates the way of combining communication paradigms (shared memory and message passing) with different architectures (multiprocessors and multicomputers). This table shows that the native communication models for multiprocessors and multicomputers are shared memory and message passing respectively, but that multiprocessors can also use message passing (which also can be efficient) and multicomputers can use shared memory mechanisms (although this latter solution in practice is not so efficient).

5.3.2 Comparison of parallel programming models

As explained before, parallel architectures as well as parallel software provide a way of dividing a task into smaller subtasks, each of them being executed on a different processor.

	Multiprocessors	Multicomputers
Shared memory	native	virtual shared memory library
Message passing	implemented over shared memory	native

Table 5.1: Table showing the combination of communication models and parallel architecture models. The native communication models for multiprocessors and multicomputers are shared memory and message passing respectively.

When programming this subdivision of tasks we can use several concurrent models, as described in Section 5.2.1, such as client-server or producer-consumer. Whichever the type of model chosen, two are the aspects that should be taken into account in order to compare two different parallel implementations of a computationally expensive sequential program:

Granularity: This is the relative size of each of the computation units (i.e. the amount of work for each of the workers) that execute in parallel. This concept is also known as coarseness, or fineness of task division.

Communication: This is relative to the way that execution units communicate to each other and how they synchronize their work.

It is important to take into account that the number of processes or workers is also an important factor to consider every time that the parallel program will be executed. This is very commonly a parameter of parallel programs. One might think that it is better to use as many processes as possible so that the workload of each is very low and each of them needs less time to complete its subtask, leading to a shorter time to complete the whole job. However, it is important to take into account that the number of CPUs is a limiting factor, as the maximum number of processes that can be running at the same time is equal to the number of CPUs (the rest will be in an idle state waiting for an executing process to pass to a blocked state and to take ownership of the freed CPU). In addition, it is also important to note that creating a new process is a procedure that also requires some additional time by the operating system, as well as the communication or synchronization procedures to coordinate them all. The latter is specially costly when processes are executing in different workstations connected through a network. That is why the number of processes has to be carefully chosen trying to find a balance between workload and cost for creating, communicating, and synchronizing these.

5.3.3 Communication in distributed systems: existing libraries

In the recent years some alternatives have been created in order to provide a programming interface when writing programs for multicomputers. Three of the most known ones are the following:

- Parallel Virtual Machine (PVM): PVM [Parallel Virtual Machine, 1989] is a software system that allows a heterogeneous network to be seen as a parallel computer. As a result, when a parallel program is to be executed, this is done over a virtual parallel

machine –hence the name. PVM is maintained by the so called *Heterogeneous Network Computing research* project. This project is a collaboration of the Oak Ridge National Laboratory, the University of Tennessee and Emory University.

- OpenMP[3] application program interface: OpenMP [OpenMP, 1997] is a portable, scalable model that gives shared-memory parallel programmers a simple and flexible interface for developing parallel applications for platforms ranging from the desktop to the supercomputer. The design of OpenMP is based on threads that share a common memory space.

- The Message Passing Interface (MPI): MPI [Message Passing Interface, 1993] has been designed and standardized by the so called *MPI forum* which is formed by academic and industrial experts and is thought to be used in very different types of parallel computers. Nowadays there are many implementations of the standard available for many platforms. MPI is used now by the most important parallel computer vendors, as well as in universities and commercial companies.

PVM and OpenMP are standards where both the interface and implementation are defined (they are closed standards regarding these aspects), while in MPI only the interface is designed and many different implementations exist. These parallel programming alternatives were designed with the main objective of providing a solution to the problems of lack of standardization in parallel programming, that resulted in a lack of portability. Actually, different versions of these libraries exist for many different operating systems and architectures, which make them be portable enough for most of the existing computer systems.

5.4 Parallelism techniques in the literature applied to graph matching

Due to the complexity of many real graph matching problems, long execution times are required for dealing with all the search space as well as with all the data to evaluate each solution. As a result, parallelism techniques have been proposed in the literature to parallelize graph matching algorithms of very different types.

Many different parallelism techniques have been applied in the literature for faster computation of graph matching problems. Among them we have the use of linear combination and parallel graph matching techniques for 3D polyhedral objects representable by 2D line-drawings [Wang, 1999], and the use of a communication scheduling framework for communication algorithms [Bhat et al., 1999].

The subject of using parallel techniques to graph matching is also analyzed as a whole subject in some references. Examples of this are the description of the basic combinatorial, algebraic, and probabilistic techniques required for the development of fast parallel algorithms for graph matching problems and for closely related combinatorial problems [Karpinski and Rytter, 1998, Reif, 1993], and a study of different parallel algorithms as well as a proposal of a new one for attributed exact graph matching [Abdulkader, 1998].

On the other hand, there are also examples on applying graph matching techniques to improve parallelism in computer networks and operating systems. An example of this is tackling the problem of finding an optimal allocation of tasks onto processors of a distributed computing system [Tom and Murthy, 1999].

[3]The MP in OpenMP stands for *Multi Processing*.

All these references concentrate on the design of the parallel program, and in analyzing the complexity of the different graph matching approaches in order to obtain a simpler one. In addition, they are mainly based in shared memory approaches and threads, and practically no examples on combining fast networks and message passing standards for graph matching problems can be found.

5.5 Parallelization of sequential EDA programs

5.5.1 Computation needs in EDAs

Experiments applying EDAs to complex problems such as inexact graph matching –see Chapters 6 and 7– show that some of the EDA algorithms are very time consuming [Bengoetxea et al., 2000, 2001a,b,c,d, 2002a]. The execution time is directly proportional to the number of dependencies (i.e. maximum number of parents that each node can have) that the learning algorithm takes into account, as well as to the number of vertices of both the model and data graphs (i.e. the number of regions to recognize). This is specially evident in the case of EBNA for the discrete domain and EGNA for the continuous domain. Unfortunately, in real problems these graphs contain a lot of vertices due to imprecisions in image acquiring techniques, and therefore these two EDAs can even require many days to fulfill the stopping criterion.

Parallel programming techniques can be applied to improve these execution times by executing processes in parallel. As a first step before starting to modify the code, it is important to identify the steps on the algorithm that are suitable for parallelization, as some of its steps are inherently sequential and cannot be parallelized. An example of a step that one can think of to be easily parallelized is the simulation step. This step is used to create the R individuals that will form the next generation after the learning step in the probabilistic graphical model. These individuals can be created in parallel, as the simulation of each of these individuals has to be done without taking into account the generation of the previous ones.

Another important step that can be parallelized is the learning step in EDAs. EBNA and EGNA algorithms are among the most complex EDAs as they try to take into account all types of dependencies between the variables, and therefore the learning steps of these two are specially time consuming. Due to this reason, we will concentrate in parallelizing these two algorithms. However, before proceeding to any parallelization design, it is essential to know how the learning is performed on these. The learning procedures in EBNA and EGNA have been reviewed in Sections 4.3.4 and 4.4.4 respectively. The techniques described in Section 5.2.1 were applied for this purpose.

Whichever the procedure selected for parallelization, it is important firstly to analyze in the whole problem which of these steps is the one that consumes most of the CPU time of the overall execution time. This is an important factor since parallelizing a function that only supposes for instance a 5% in the global execution of the program will not have much influence in terms of reduction of execution time. The next section is a study of execution steps and procedures on EDAs.

5.5.2 Analysis of the execution times for the most important parts of the sequential EDA program

It is important to realize that the simple procedure of creating a new execution unit (either a process or a thread) requires some additional execution time, and therefore further study is required in order to be sure that the amount of work per process is big enough to justify the extra time-overhead of creating the process itself.

The different EDAs described so far for both the discrete and continuous domains have been executed and the time required for each of the internal steps and procedures has been measured. The tool used for this measurement is the widely known *gprof*, which is a GNU tool that records all the required information that we need. As already explained in Section 5.2.2, this tool is used together with the *gcc* ANSI C++ compiler. Table 5.3 shows the execution times in absolute values to obtain these results after execution in a two processor Ultra 80 Sun computer under Solaris version 7 with 1 Gb of RAM. It is important to note that these values do not correspond to ordinary execution times because the compilation options required for doing this analysis are different and debugging flags are active, thus making the execution much slower. After all, the results are given in statistical terms such as percentages of use of CPU, information that is enough for our purposes. That is also why execution in another machine such as a PC with Linux would return similar results in terms of execution time percentages.

Three experiments were carried out using the inexact graph matching problem with graphs generated randomly for Study 1 (the 10 & 30, 30 & 100 and 30 & 250 examples), and using the fitness function defined in Section 3.4.3. Some of the results obtained with these experiments are shown in Table 5.2.

In order to understand the results shown in Table 5.2, it is important to have a better understanding on the purpose of the procedures in the source code. There is a fitness function that is used to compute the value of each individual. This function is represented in the table as *Fitness Func.* and is the one defined in Equation 3.2. The fitness function is the same for both discrete and continuous EDA experiments, although in the latter algorithms there is an additional step in the continuous case as described in Section 3.3.2. These procedures and the way of carrying out the experiments are described in detail in Section 3.4.3 as well as in [Bengoetxea et al., 2001c,e, Mendiburu et al., 2002].

There is also a procedure that performs the learning for each of the EDAs, the learning of the probabilistic graphical model (Bayesian network or the Gaussian network in discrete and continuous domains respectively) expressed in the table as *Learning* for the discrete and continuous EDAs. This procedure constitutes the main difference between the different EDAs, and the relative execution time among the different EDAs of this procedure depends essentially on the complexity of the chosen algorithm. In the continuous domain, there are also two procedures called *Means and Covariances* and *Matrix operations* that are also part of the learning process of the algorithms (i.e. their times are included on the *Learning* part on continuous EDAs) that are on the table in order to show their relative significance in the execution time.

Finally, the last procedure to mention is *Simulation*, which finality is the generation of the R individuals of the next generation. This procedure is different in the discrete and continuous cases, but essentially they perform the same task.

The times given in Table 5.2 only reflect the time in percentages. This information shows clearly that in the EBNA$_{BIC}$ and EGNA$_{BIC}$ cases the fact of parallelizing the simulation step would not reduce drastically the overall execution time of the whole program, as in

EDA Algorithm	Procedure	Execution Time (%)	Procedure	Execution Time (%)
UMDA	Fitness Func.	9.6	Fitness Func.	81.9
	Simulation	7.4	Simulation	9.4
MIMIC	Learn Bayesian N.	7.7	Learn Bayesian N.	24.2
	Fitness Func.	12.89	Fitness Func.	63.7
	Simulation	6.4	Simulation	8.0
$EBNA_{BIC}$	Learning (BIC)	47.3	Learning (BIC)	85.7
	Fitness Func.	18.1	Fitness Func.	12.3
	Simulation	4.0	Simulation	1.4
$UMDA_c$	Means and Covariances	24.6	Means and Covariances	24.6
	Fitness Func.	14.3	Fitness Func.	34.7
	Simulation	2.1	Simulation	0.9
$MIMIC_c$	Learning	23.6	Learning	23.1
	Fitness Func.	13.7	Fitness Func.	34.1
	Means and Covariances	23.5	Means and Covariances	22.6
	Simulation	4.4	Simulation	3.5
$EGNA_{BGe}$	Learning	5.6	Learning	55.0
	Means and Covariances	21.4	Means and Covariances	7.2
	Fitness Func.	12.2	Fitness Func.	10.9
	Simulation	1.9	Simulation	0.3
$EGNA_{BIC}$	Learning (BIC)	53.1	Learning (BIC)	(*)
	Means and Covariances	3.5	Means and Covariances	(*)
	Fitness Func.	1.6	Fitness Func.	(*)
	Simulation	0.6	Simulation	(*)
$EGNA_{ee}$	Learning	34.2	Learning	(*)
	Means and Covariances	15.9	Means and Covariances	(*)
	Matrix operations	17.2	Matrix operations	(*)
	Fitness Func.	9.2	Fitness Func.	(*)
	Simulation	5.3	Simulation	(*)
$EMNA_{global}$	Learning	40.4	Learning	(*)
	Matrix operations	33.7	Matrix operations	(*)
	Means and Covariances	6.8	Means and Covariances	(*)
	Fitness Func.	5.4	Fitness Func.	(*)
	Simulation	10.3	Simulation	(*)

Table 5.2: Time to compute for two graph matching problems synthetically generated with sizes 10 & 30 (first column) and 50 & 250 (second column). All the figures are given in relative times, i.e. 100% = full execution time. The values with the symbol (*) would require more than a month of execution time to be properly computed.

the big case (last column) this step does not represent a significant execution time of the algorithm (only the 0.6% and 4% of the execution time). These two experiments show clearly that the most time consuming function is the *BIC* function, the one that is used to evaluate the different Bayesian and Gaussian networks in their respective versions.

It is also important to realize from the tables presented that the relative significance of the learning procedures in the execution time is more important when the size of the problem is also bigger. These data also show that the learning is not linear regarding the size of the problem, showing the already known NP-hard nature.

EDA Algorithm	10 & 30 example: execution time	50 & 250 example: execution time
UMDA	00:02:22	04:28:25
MIMIC	00:02:25	05:27:33
EBNA_{BIC}	00:04:44	37:01:45
UMDA_c	00:28:52	33:37:51
MIMIC_c	00:29:42	33:25:03
EGNA_{BGe}	00:34:14	146:43:30
EGNA_{BIC}	03:52:04	(*)
EGNA_{ee}	00:44:34	(*)
EMNA_{global}	01:46:34	(*)

Table 5.3: Time to compute the analysis in Table 5.2 for the 10 & 30 and 50 & 250 examples (hh:mm:ss). Again, the values with the symbol (*) required more than a month of execution time to be properly computed.

As discrete representations are mainly used in real graph matching problems we decided to parallelize the most CPU expensive discrete EDA that we defined: the EBNA_{BIC}. Therefore, this thesis concentrates mainly on parallelizing the BIC score, which implementation appears to be very important in the total execution time required by this EDA medium size problems.

5.5.3 Interesting properties of the BIC score for parallelization

Looking at the time required to execute algorithms such as EBNA in the discrete domain and EGNA in the continuous domain, it appears clear that parallel programming and concurrency techniques need to be applied in order to obtain shorter execution times. Some parallel algorithms have already been proposed in the literature for similar purposes [Freitas and Lavington, 1999, Sangüesa et al., 1998, Xiang and Chu, 1999], and also more concretely for the EBNA algorithm [Lozano et al., 2001].

In EBNA_{BIC} and EGNA_{BIC} the learning of the probabilistic graphical model is usually done by starting with an arc-less structure and by adding or removing step by step the arc that most increases the BIC score. This process is repeated until a stopping criterion is met, and the final result is a probabilistic graphical structure that reflects the interdependencies between the variables. As a result, both EBNA_{BIC} and EGNA_{BIC} are based on a score+search approach.

The BIC score is based on the penalized maximum likelihood. It can be written as:

$$BIC(S,D) = \sum_{i=1}^{n} BIC(i,S,D) \tag{5.1}$$

$$BIC(i,S,D) = \sum_{j=1}^{q_i}\sum_{k=1}^{r_i} N_{ijk}\log\frac{N_{ijk}}{N_{ij}} - \frac{1}{2}(r_i-1)q_i. \tag{5.2}$$

An important property of the BIC score is that it is *decomposable*. This means that the score can be calculated as the sum of the separate local BIC scores of each of the variables. Therefore, each variable X_i has associated a local BIC score –$BIC(i,S,D)$– as defined in Equation 5.2.

As a result, this allows us to compute the component that each variable adds on the global BIC score of the structure separately. It is important to remember that the arc adding or removing has to ensure that the structure will still be a DAG (Directed Acyclic Graph) in order to be accepted.

The structural learning algorithm has to find the best arc addition or removal in order to improve the BIC score. This task is accomplished by computing the corresponding BIC score for every arc modification. Therefore, as there are $n(n-1)$ possible arc modifications in a structure with n nodes, there are also $n(n-1)$ possible gains on the BIC scores to calculate each step arc modifications. The arc modification that maximizes the BIC score, whilst maintaining the DAG structure, is applied to S. Also, if the arc (j,i) is modified (i.e. added or removed), only the component $BIC(i,S,D)$ is affected. In the next step the rest of the $BIC(k,S,D)$ $k \neq i$ do not change, and therefore only $n-2$ terms have to be computed.

All the computation of the different possible arc modifications can be computed separately. This task can be distributed to different processes that can be computing in parallel all the $BIC(i,S,D)$ in different processors.

5.5.4 Parallel techniques applied in this thesis

Regarding the different parallel architectures and systems shown in Section 5.3 where a parallel program can run and in order to offer the two possibilities of using shared memory or message passing, we have developed two different parallel versions of EBNA$_{BIC}$: one is based on using threads and shared memory, suitable for SM and SDM or multiprocessor machines in general, and was implemented using the *pthreads* library. The other is based on processes communicating using message passing, suitable for NOW, clusters, or multicomputers, implemented using the MPI interface. The reason for choosing these two parallelization standards is their proved performance, but also their portability and availability for different operating systems such as Windows, Solaris and Linux. Appendix D shows the main parts of the source code in EBNA, while an example of parallelizing the EBNA program in the way described is presented later in Section 6.4. Experimental results and conclusions are shown in the latter section for both using threads and MPI. These techniques are applied to the parallelization of the BIC procedure due to its computation cost in complex problems.

Chapter 6

Experiments with synthetic examples

'Learning is not attained by chance, it must be sought for with ardor and attended to with diligence. '

Abigail Adams

6.1 Introduction

This chapter introduces three different studies leading to demonstrate the validity of the EDA approach and its behavior for the application to inexact graph matching problems. Different aspects already considered but not shown with experimental tests are described in these studies.

In order to avoid at the maximum the influence of a particular real problem on the behavior of EDAs and other algorithms, random or synthetic attributed graphs have been created and will be used as a starting point in all the experiments in this chapter. The aim of the different sections is as follows: Study 1 analyzes and compares the performance of EDAs and other evolutionary computation techniques such as GAs. Study 2 is an illustration of what happens inside complex EDAs and tries to make clearer the way of approaching better solutions. For this, the best structure that the probabilistic graphical models adopts for better representing the selected individuals of each generation is shown. Finally, in Study 3 the parallelization process of a complex algorithm such as EBNA is carried out, and an easily adaptable method for other similar complex approaches such as EGNA is introduced and justified with experimental data showing the considerable improvement in execution time that the proposed method obtains.

6.2 Study 1: measurement of the performance

6.2.1 Design of the experiment

Three different synthetic examples of graphs have been created randomly using different sizes of model graphs –G_M– and data graphs –G_D. The sizes of these graphs are as follows from the smallest to the biggest: in the first example the model graph G_M contains 10 vertices and 15 edges, and the data graph G_D has 30 vertices and 39 edges. For the second example the graph G_M contains 30 vertices and 39 edges, and the graph G_D 100 vertices and 247

edges. Finally, in the third example G_M contains 50 vertices and 88 edges, and the graph G_D has 250 vertices and 1681 edges. The sizes of these graphs have been chosen carefully. Firstly, the size for the graphs of the first –small– example has been selected based on the explanations given for a real problem in [Boeres et al., 1999, Boeres, 2002], which shows a reduced example of the inexact graph matching problem of graphs extracted from healthy human brain images introduced in [Perchant et al., 1999] and [Perchant and Bloch, 1999]. The size of the graphs for the third example are similar to the ones introduced in [Perchant et al., 1999], in which a model graph G_M that contains 43 vertices and 336 edges is matched against another graph G_D that contains 245 vertices and 1451 edges. Finally, the graphs for the second example are half way between the first and the third examples' ones. In what follows, we will call 10 & 30 example, 30 & 100 example, and 50 & 250 example to the small, second, and big graph matching cases respectively.

The number of edges chosen for all these graphs were selected knowing that the value returned by this fitness function does not depend on $|E_M|$ and $|E_D|$. Following the classification of graphs between sparse and dense introduced in [Larrañaga et al., 1997], the number of edges have been chosen to be the median of the sparse graphs of that size.

As in every optimization problem, a fitness function has to be defined in order to evaluate the goodness of any of the possible solutions. The fitness function $f_2(h)$ introduced in Section 3.4 (Equation 3.2) was selected for the experiments in this study just as an example because of its previous use in real graph matching problems [Boeres, 2002, Perchant and Bloch, 1999, Perchant et al., 1999].

For all the three cases, both the model and data graphs G_M and G_D have been generated randomly from scratch. The fact that no image processing is performed in the graph construction avoids the dependence of the image processing techniques in the final result. Obviously, as these graphs are generated randomly, they do not represent neither any knowledge nor any common segments, and we do not have previous knowledge about which is the optimum matching between vertices of G_M and G_D. As well as both graphs, $c_N(a_D, a_M)$ and $c_E(e_D, e_M)$ were also generated randomly, and α is assigned a value of 0.8 because the best results are obtained with this value in [Boeres et al., 1999] for their particular application. This value is taken as an example for these experiments too.

As the aim of the experiments with these three synthetic examples is to test the performance of EDAs in general, we have selected three discrete EDAs introduced in Section 4.5.1, as well as four continuous EDAs introduced in Section 4.5.2. Because the main difference between EDAs in both domains is the number of dependencies between variables that they can take into account, the fact of having graphs with different sizes will influence parameters such as the best solution obtained after a number of generations, the time to compute the algorithm, and the evolution of the algorithm itself through the search. This section describes the experiments and the results obtained[1]. The three discrete EDA algorithms are also compared to three broadly known GAs: canonic basic (cGA) [Holland, 1975], elitist (eGA) [Michalewicz, 1992] and steady state (ssGA) [Whitley and Kauth, 1988]. The two first GAs evolve from a population to another by applying crossover operations to some individuals in the population, and the difference between them is that the eGA always includes in the new population the best individual of the previous one, whereas cGA does not. The ssGA approach is somehow different, as it only generates one individual at each iteration, replacing only the worst individual of the population when its fitness value is worse than the

[1] A review of this work focusing only on the discrete domain can also be found in [Bengoetxea et al., 2001a, 2002a].

one of the new individual.

EDAs and GAs were implemented in ANSI C++ language, and the experiments were executed on a two processor Silicon Graphics machine SGI-Origin200 under IRIX OS version 64-Release 6.5 with 500 Mbytes of RAM.

In the discrete case, all the programs were designed to finish the search when the whole population is formed by the same repeated individual or when a maximum of 100 generations was reached. GAs were programmed to generate the same number of individuals as with discrete EDAs, and therefore 100 generations were executed for all of them. The ssGA is a special case because of generating a single individual at each iteration, but it was also programmed in order to generate the same number of individuals by allowing more iterations. In the continuous case, the algorithms were designed to finish when the 150^{th} generation was reached.

The initial population for all the discrete algorithms was generated using the same random generation procedure based on a uniform distribution for all the possible values, and in the case where the correction of the individuals applies, both discrete EDAs and GAs were programmed using the same correction procedures. In the same way, the fitness function used in all the algorithms is exactly the same. In the continuous case, the generation of the first generation was also done following a similar procedure for generating continuous values.

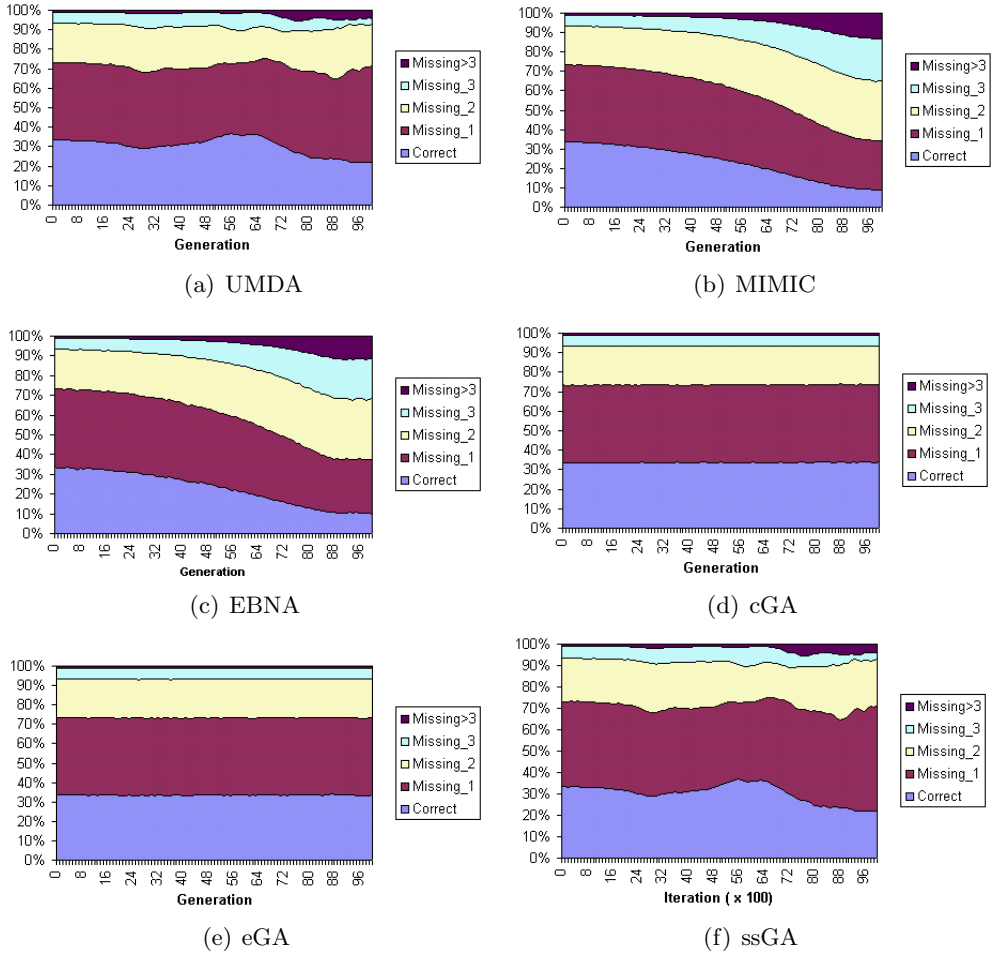
In EDAs of both domains, the following parameters were used: a population of 2000 individuals ($R = 2000$), from which the best 1000 are selected ($N = 1000$) to estimate the probability, and the elitist approach was selected (that is, always the best individual is included for the next population and 1999 individuals are simulated). In GAs a population of 2000 individuals was also selected, with a mutation probability of $1.0/|V_D|$ and a crossover probability of 1.

6.2.2 The need to obtain correct individuals

As one of the aims of this study is to analyze the behavior of EDAs in ordinary problems, we decided to add extra constraints to the problem. For this, three conditions have been introduced in Section 3.3.3 that need to be satisfied for any solution. It is important to realize that both in GAs and discrete EDAs the individuals generated every generation could contain an incorrect solution (that is, the solution might not satisfy the three conditions introduced in Section 3.3.3). That is why some techniques to avoid the presence of incorrect individuals have been introduced in Section 4.5.1. Continuous EDAs do not have such a problem, as the individual representation chosen avoids it completely.

Nevertheless, as using the techniques to obtain correct individuals introduced in Section 4.5.1 implies a computational cost as well as a direct and permanent manipulation on the population itself, it is important to check whether the percentage of incorrect individuals for the different algorithms is high enough to justify such a correction. We will take our 30 & 100 case as an example to confirm whether the additional manipulation process is required or not.

Figure 6.1 shows the percentages of correct and incorrect individuals during the search process for the three discrete Estimation of Distribution Algorithms (UMDA, MIMIC and EBNA), as well as for the cGA, eGA and ssGA in the 30 & 100 example without applying any technique to correct the wrong individuals (PLS only). Similarly, Figure 6.2 shows the results of applying penalization under the same conditions. These graphics illustrate the mean results of 20 executions for each of the algorithms. For each graphic the x axis represents the generation number, and the y axis the percentage of individuals that are



(a) UMDA (b) MIMIC

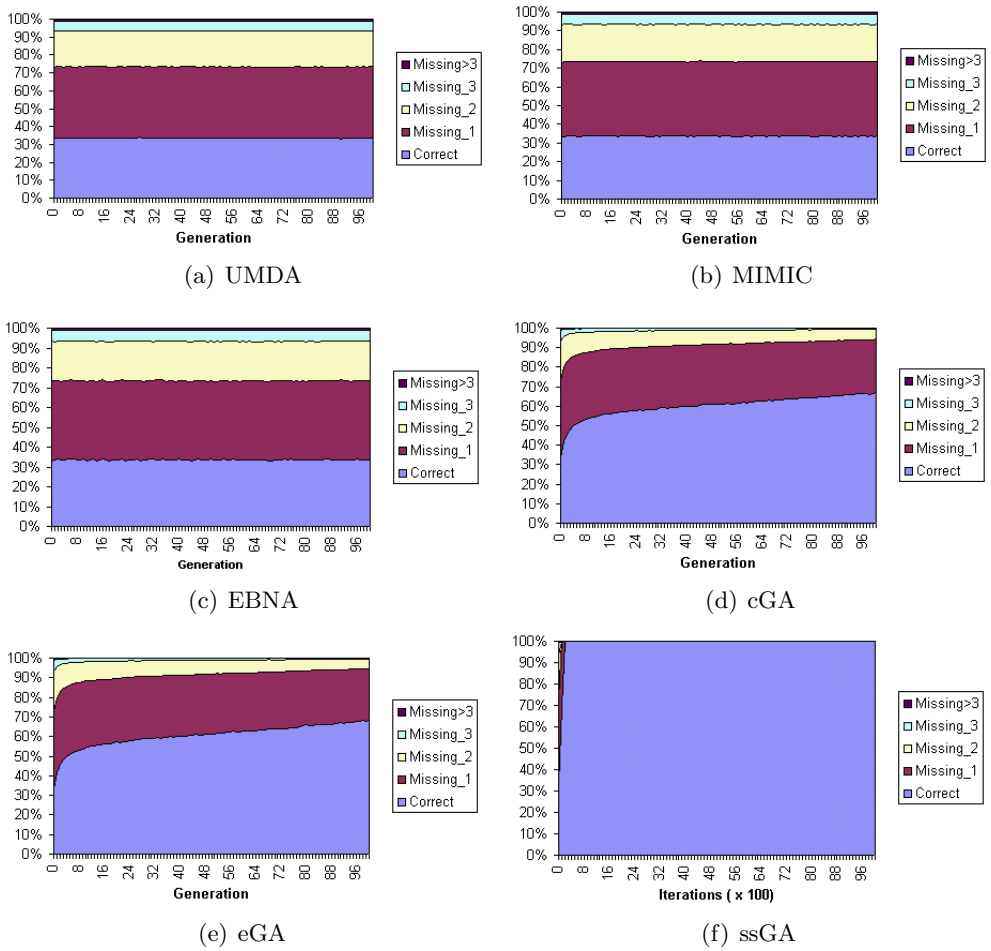
(c) EBNA (d) cGA

(e) eGA (f) ssGA

Figure 6.1: Figures for the correctness of the UMDA, MIMIC and EBNA (discrete EDAs) as well as for the cGA, eGA and ssGA (GAs), applied to the second example of 30 & 100 vertices without correction.

correct (i.e. all the vertices in V_M have been assigned in the matching), the ones where a value is missing (i.e. when only one vertex of V_M has not been assigned), the ones where two values are missing (i.e. when two vertices of V_M have not been assigned), when three values are still to be assigned, and finally the individuals in which more than three vertices have not been assigned to a data vertex. These graphics illustrate that at the final generation of all the algorithms practically none of the individuals is acceptable, but the percentage of correct individuals decreases sooner when increasing the size of the graphs. It is important to note that in this 30 & 100 example the individuals have a total size of 100 variables or genes, and each of them have to be assigned to a value between 1 and 30 ($|V_M| = 30$ and $|V_D| = 100$) has to be assigned to each of them, thus it is more probable that at least one of the vertices in V_M has not been assigned in the variables or genes than in the 10 & 30 example.

From the results we can conclude that for graphs of complexity similar or higher than in this 30 & 100 example, the number of correct individuals gradually decreases every generation for all the algorithms. Furthermore, the percentage of individuals that have one vertex of V_M not matched in the last generation appears to be very high for all the algorithms. These



(a) UMDA (b) MIMIC

(c) EBNA (d) cGA

(e) eGA (f) ssGA

Figure 6.2: Figures for the correctness of the UMDA, MIMIC and EBNA discrete EDAs as well as for the cGA, eGA and ssGA GAs, applied to the second example of 30 & 100 vertices and using penalization.

graphs show clearly the behavior of the different algorithms, as well as the nature of the fitness function selected. As a result, we can conclude that without any mechanism to correct or guide the generation of individuals the percentage of correct ones will decay quite fast. This decay can be appreciated in both discrete EDAs and GAs, but for the case of EDAs the problem appears to be much more important, as when applying PLS simulation alone none of the three EDA contains any correct individuals at the last generation of the search. On the other hand, the three types of GAs do also contain a high proportion of incorrect individuals at the last generation, and therefore this does not ensure that the final solution returned by the algorithm will be a correct one.

Another conclusion about the penalization procedure can also be obtained from these results: the penalization of the incorrect individuals is the only correction method that does not manipulate the learning and simulation steps (the others are LTM and ATM as explained in Section 4.5.1), but whichever the penalization weight to the incorrect individuals there will always be the possibility of finding incorrect individuals in the final population. In fact, the proportion of incorrect individuals for penalization in UMDA, MIMIC, EBNA, cGA, eGA and ssGA were of 33.66%, 33.69%, 33.69%, 66.81%, 68.32% and 100% respectively. A

stronger penalization is required to be applied to individuals if this method is to be selected for graphs with as many vertices as in these three examples, but it would never ensure 100% of correct individuals in the population.

6.2.3 Discrete domain

Experimental results by combining correction methods and algorithms

Once proved the need to control the generation of the individuals in every population, the three methods described in Section 4.5.1 for discrete domains were combined with the three discrete EDA algorithms. In the case of the GAs, the last two methods described in the same section were used for cGA eGA and ssGA, as the ones based on the modification of the probability in the simulation step do not apply in GAs which do not perform such a step.

The results obtained from the different executions of the algorithms are shown in this section. For each algorithm and example the mean values of the fitness value of the best individual at the last generation, the number of generations to reach the final solution, and the computation time are shown.

The computation time presented in these experimental results is actually the CPU time of the process from the beginning to the end, and therefore it is not dependent on the variations on the multiprogramming level during the execution time. This computation time is presented as a measure to illustrate the different computation complexity of all the algorithms. It is important also to note that all the operations for the estimation of the distribution, the simulation, and the evaluation of the new individuals are carried out through memory operations.

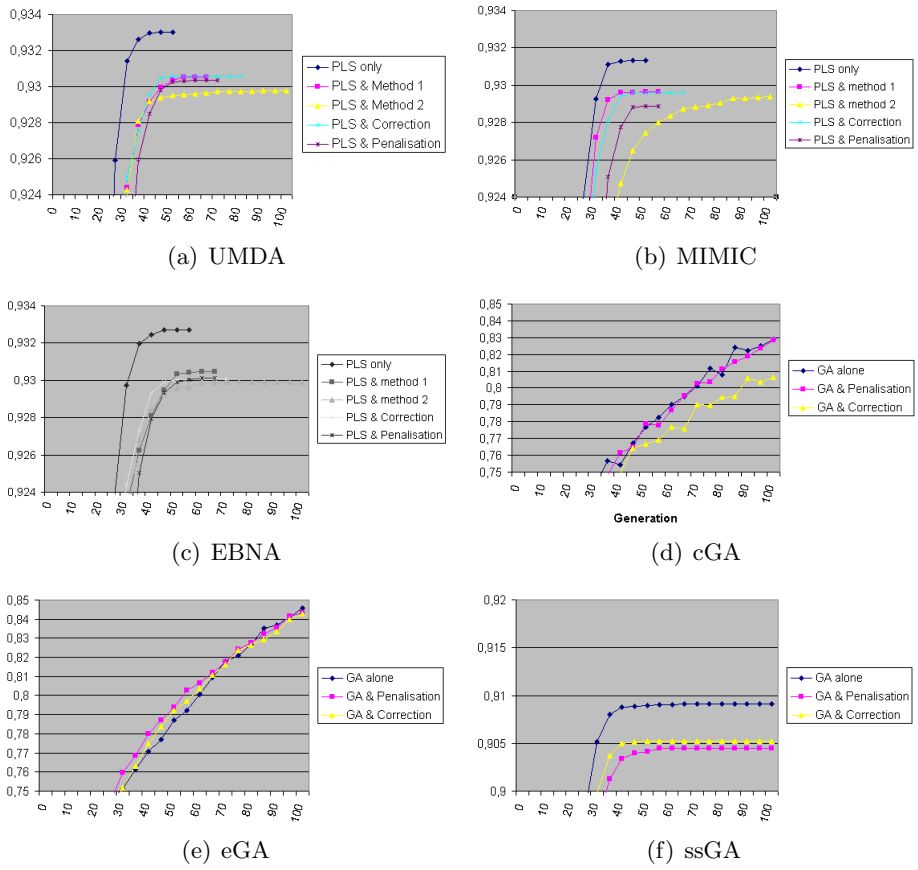
The null hypothesis of the same distribution densities was tested for each of the different algorithms and for each of the correction methods to control the generation of new individuals. The non-parametric test of Kruskal-Wallis was used [Kruskal and Wallis, 1952][2]. This task was carried out with the statistical package S.P.S.S. release 9.00.

Discrete domain: best individual, number of generations required, and computation time

The results of the 10 & 30 example are shown in Figure 6.3 and Tables 6.1, 6.2, and 6.3. Figure 6.3 is done with the mean of 20 executions for all the algorithms, showing their different behaviors depending on the correction method employed. The reader is reminded that the *PLS only* and *Penalization* methods do not ensure a population of only correct individuals. Shorter lines indicate that the algorithm finishes requiring less generations. Tables 6.1, 6.2, and 6.3 show also the mean values as well as the results of applying the Kruskal-Wallis test to the different parameters (fitness value, number of generations required, and execution time required).

In an analogous way, results for the 30 & 100 example are shown in Figure 6.4 and Tables 6.4, and 6.5. Results for the 50 & 250 example are found in Figure 6.5 and Tables 6.6, and 6.7. These tables show also the mean values showing their behavior when changing the correction method employed as well as the results of applying the Kruskal-Wallis test to the different parameters (metric and execution time required). The number of generations reached for the 10 & 30 and 30 & 100 examples was of 100 for all the algorithms.

[2]The interested reader is referred to [Siegel, 1956] for a deep explanation on non-parametric tests.



(a) UMDA (b) MIMIC

(c) EBNA (d) cGA

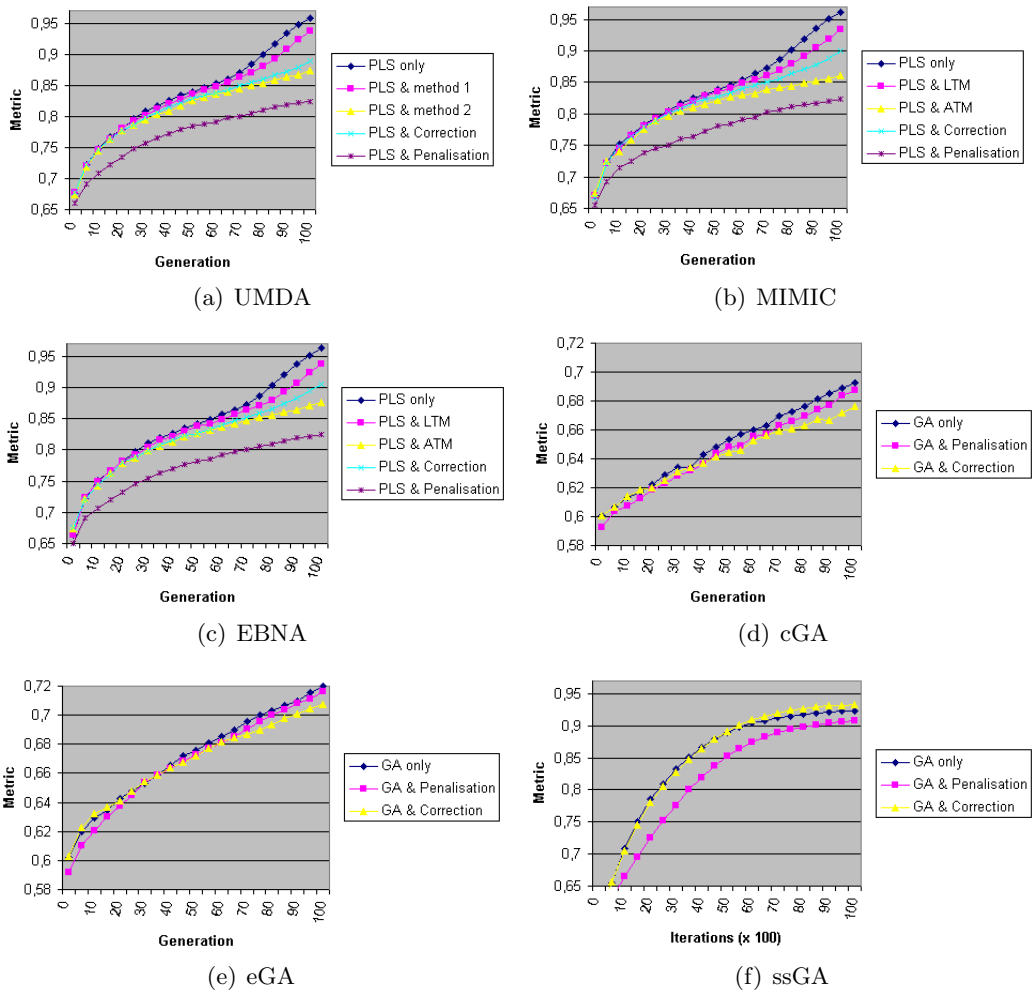
(e) eGA (f) ssGA

Figure 6.3: Graphs showing the best individual at each generation of the searching process for the algorithms UMDA, MIMIC, EBNA, cGA, eGA, and ssGA for the case of the 10 & 30 vertex graphs. Note the different scales between discrete EDAs and GAs.

At the light of the results obtained in the fitness values, we can conclude that from the three GAs used, ssGA appears clearly as the one that obtains the best results. Furthermore, the computation time to generate the final solution is also less than the one required by the other two GAs.

There is little difference in the best individual obtained by the different discrete EDAs: even if with some correction methods EBNA obtains the best results, in some cases such as in LTM and penalization, UMDA returned the best results. As explained before, EBNA was expected to return the best result due to its ability to estimate more accurately the probability, in spite of the higher computational cost. Nevertheless, as the differences do not appear to be significant, we could not conclude that any algorithm is superior to the rest simply based on these experimental results. Even if in EBNA no restrictions are set to the structure to learn, the results obtained could indicate that the most appropriate structure for this problem could be a structure with at most pairwise dependencies. It is also important to note that both graphs G_M and G_D have been created at random and that they do not show any knowledge, which makes the matching process even more dependent on the fitness function used.

Regarding the difference between discrete EDAs and GAs, it seems clear that EDAs obtain better results for any of the correction methods applied to the individuals. It is



(a) UMDA (b) MIMIC

(c) EBNA (d) cGA

(e) eGA (f) ssGA

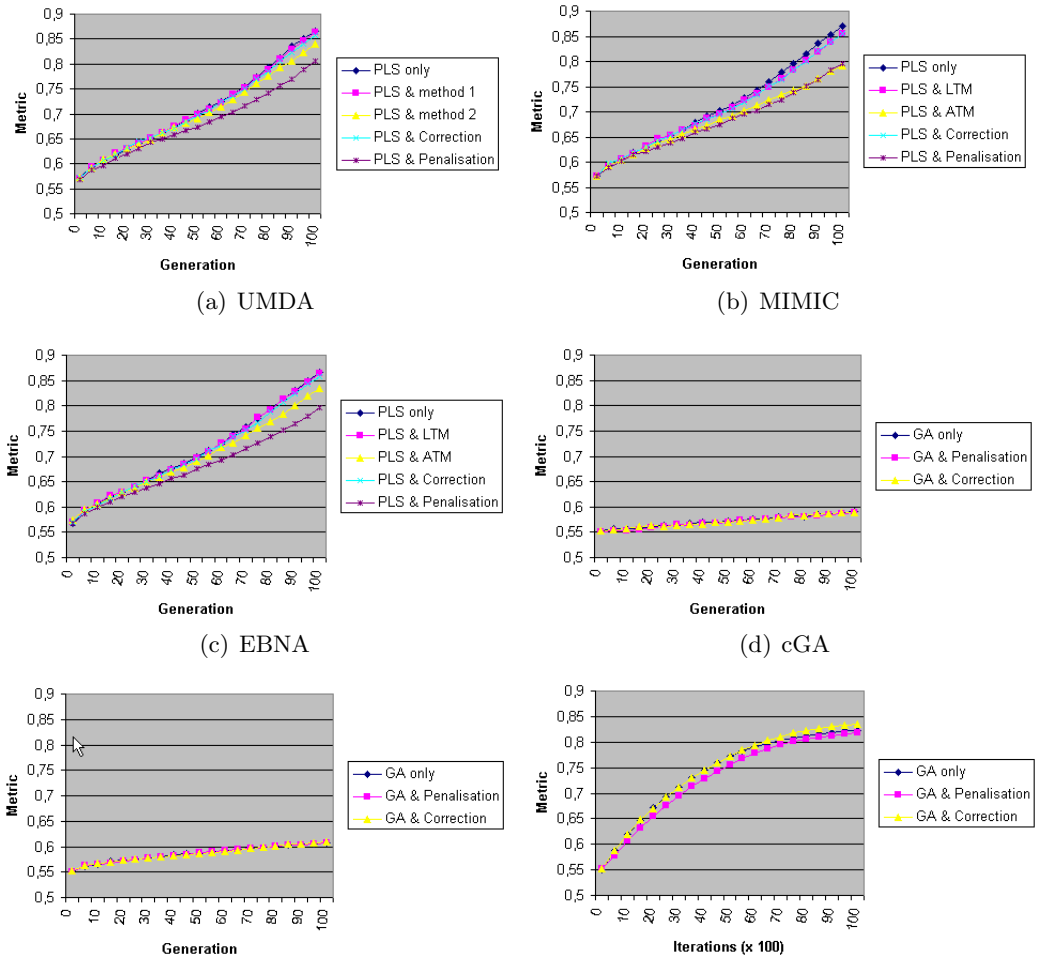
Figure 6.4: Graphs showing the best individual at each generation of the searching process for the algorithms UMDA, MIMIC, EBNA, cGA, eGA, and ssGA for the case of the 30 & 100 vertex graphs. Note the different scales between discrete EDAs and GAs.

important therefore to note that ssGA obtains similar results as EDAs.

Additional results. Additionally, the Kruskal-Wallis test was also applied to the correction methods between discrete EDAs only, and the non-parametric test of Mann-Whitney [Mann and Whitney, 1947] was carried out for GAs only. The results are shown in Table 6.8.

Another important aspect to remember is the control of the correctness of the individuals. As the LTM and ATM imply the manual modification of the learned model by changing the probabilities, the learning itself is somehow manipulated. The correction of individuals does also modifies at random some of the individuals of the population. The penalization of the incorrect individuals is the only correction method that does not manipulate the learning and simulation steps, but whichever the penalization weight to the incorrect individuals there will always be the possibility of containing incorrect individuals in the final population. A stronger penalization could improve these values, but it would never ensure 100% of correct individuals in the population.

Regarding the fitness values of the best individuals obtained at the end of the search



(a) UMDA (b) MIMIC

(c) EBNA (d) cGA

(e) eGA (f) ssGA

Figure 6.5: Graphs showing the best individual at each generation of the searching process for the algorithms UMDA, MIMIC, EBNA, cGA, eGA, and ssGA for the case of the 50 & 250 vertex graphs. Note the different scales between discrete EDAs and GAs.

processes, we can conclude that from the three GAs used, ssGA appears clearly as the one that obtains the best results. Furthermore, the computation time to generate the final solution is also less than the one required by the other two GAs. The best individuals obtained using the different EDAs are very similar: even if with some correction methods EBNA obtains the best results, in some cases such as in LTM and penalization UMDA performs better. As explained before, EBNA is expected to return better results due to its ability to estimate more accurately the probability distribution every generation, in spite of a higher computational cost. Nevertheless, the small differences between EDAs do not appear to be significant for this example regarding Table 6.8. This effect can be explained by the fact that both graphs have been created at random and that they should not reflect any dependence between variables, and as a result EBNA cannot find more dependencies than other simpler EDAs. On the other hand, when comparing EDAs and GAs, it appears clearly that EDAs obtain better results using any of the correction methods applied to the individuals. It is important to note however that only ssGA obtains nearly as good results as EDAs.

	Method 1	Method 2	Correction	Penalization	Statistical Significance
UMDA	0.9305	0.9297	0.9305	0.9303	$p = 0.016$
MIMIC	0.9296	0.9293	0.9296	0.9289	$p = 0.242$
EBNA	0.9304	0.9299	0.9301	0.9301	$p = 0.320$
cGA	–	–	0.8065	0.8285	$p < 0.001$
eGA	–	–	0.8428	0.8431	$p = 0.766$
ssGA	–	–	0.9053	0.9045	$p = 0.636$
Statistical Significance	$p < 0.001$	$p = 0.283$	$p < 0.001$	$p < 0.001$	

Table 6.1: Best fitness values for the 10 & 30 example (mean results of 20 runs).

	Method 1	Method 2	Correction	Penalization	Statistical Significance
UMDA	50.85	67.00	49.90	53.65	$p = 0.023$
MIMIC	42.00	92.70	46.90	47.40	$p < 0.001$
EBNA	52.00	63.40	51.20	54.55	$p = 0.088$
cGA	–	–	100	100	$p = 1.000$
eGA	–	–	100	100	$p = 1.000$
ssGA	–	–	100	100	$p = 1.000$
Statistical Significance	$p < 0.001$	$p < 0.001$	$p < 0.001$	$p < 0.001$	

Table 6.2: Number of required generations for the 10 & 30 example (mean results of 20 runs).

	Method 1	Method 2	Correction	Penalization	Statistical Significance
UMDA	00:58	01:25	01:01	01:32	$p < 0.001$
MIMIC	00:56	02:39	01:02	01:32	$p < 0.001$
EBNA	03:37	04:25	03:38	04:34	$p < 0.001$
cGA	–	–	01:00	01:00	$p = 0.100$
eGA	–	–	01:01	01:01	$p = 0.100$
ssGA	–	–	01:09	01:09	$p = 0.747$
Statistical Significance	$p < 0.001$	$p < 0.001$	$p < 0.001$	$p < 0.001$	

Table 6.3: Time to compute for the 10 & 30 example (mean results of 20 runs, in mm:ss format).

6.2.4 Continuous domain

In an analogous way as in the discrete domain, continuous EDAs were also tested in order to check their performance in graph matching problems. The same three examples were taken and were executed 20 times each. The results of the experiment are shown in Figures 6.6, 6.7 and 6.8, and Table 6.9.

This table shows that the differences between the algorithms in the discrete and continu-

	Method 1	Method 2	Correction	Penalization	Statistical Significance
UMDA	0.940502	0.874684	0.892806	0.825850	$p < 0.001$
MIMIC	0.936400	0.859538	0.898960	0.824063	$p < 0.001$
EBNA	0.936739	0.875429	0.905114	0.823836	$p < 0.001$
cGA	–	–	0.674490	0.687297	$p = 0.004$
eGA	–	–	0.706609	0.712994	$p = 0.160$
ssGA	–	–	0.932318	0.911038	$p < 0.001$
Statistical Significance	$p = 0.773$	$p < 0.001$	$p < 0.001$	$p < 0.001$	

Table 6.4: Best fitness values for the 30 & 100 example (mean results of 20 runs).

	Method 1	Method 2	Correction	Penalization	Statistical Significance
UMDA	00:11:50	00:12:56	00:11:59	00:13:05	$p < 0.001$
MIMIC	00:17:19	00:18:24	00:17:29	00:18:50	$p < 0.001$
EBNA	03:18:06	03:19:06	03:18:13	03:19:16	$p < 0.001$
cGA	–	–	00:09:07	00:09:08	$p < 0.001$
eGA	–	–	00:09:07	00:09:07	$p = 1.000$
ssGA	–	–	00:09:03	00:09:04	$p = 0.317$
Statistical Significance	$p < 0.001$	$p < 0.001$	$p < 0.001$	$p < 0.001$	

Table 6.5: Time to compute for the 30 & 100 example (mean results of 20 runs, in hh:mm:ss format).

	Method 1	Method 2	Correction	Penalization	Statistical Significance
UMDA	0.863936	0.840546	0.856377	0.805633	$p < 0.001$
MIMIC	0.854939	0.792449	0.855580	0.796515	$p < 0.001$
EBNA	0.863677	0.833811	0.858611	0.795378	$p < 0.001$
cGA	–	–	0.587868	0.588509	$p = 0.725$
eGA	–	–	0.608552	0.607220	$p = 0.725$
ssGA	–	–	0.835702	0.818191	$p < 0.001$
Statistical Significance	$p = 0.050$	$p < 0.001$	$p < 0.001$	$p < 0.001$	

Table 6.6: Best fitness values for the 50 & 250 example (mean results of 20 runs).

	Method 1	Method 2	Correction	Penalization	Statistical Significance
UMDA	01:47:34	01:51:16	01:47:54	01:49:26	$p < 0.001$
MIMIC	02:45:43	02:50:07	02:45:45	02:47:26	$p < 0.001$
EBNA	53:01:35	53:08:04	53:03:35	52:59:49	$p = 0.001$
cGA	–	–	01:40:23	01:40:15	$p = 0.297$
eGA	–	–	01:40:35	01:40:34	$p = 0.925$
ssGA	–	–	01:40:50	01:40:39	$p = 0.180$
Statistical Significance	$p < 0.001$	$p < 0.001$	$p < 0.001$	$p < 0.001$	

Table 6.7: Time to compute for the 50 & 250 example (mean results of 20 runs, in hh:mm:ss format).

First Experiment(10 & 30)

	between EDAs only	between GAs only
Correction	Fitness value: $p = 0.139$ Generations: $p < 0.001$ Time: $p < 0.001$	Fitness value: $p < 0.001$ Generations: $p = 1.000$ Time: $p < 0.001$
Penalization	Fitness value: $p < 0.001$ Generations: $p < 0.001$ Time: $p < 0.001$	Fitness value: $p < 0.001$ Generations: $p = 1.000$ Time:$p < 0.001$

Second Experiment(30 & 100)

	between EDAs only	between GAs only
Correction	Fitness value: $p = 0.164$ Time: $p < 0.001$	Fitness value: $p < 0.001$ Time: $p < 0.001$
Penalization	Fitness value: $p = 0.471$ Time: $p < 0.001$	Fitness value: $p < 0.001$ Time:$p < 0.001$

Third Experiment(50 & 250)

	between EDAs only	between GAs only
Correction	Fitness value: $p = 0.886$ Time: $p < 0.001$	Fitness value: $p < 0.001$ Time: $p = 0.012$
Penalization	Fitness value: $p = 0.787$ Time: $p < 0.001$	Fitness value: $p < 0.001$ Time:$p = 0.025$

Table 6.8: Particular non-parametric tests for the 10 & 30, 30 & 100 and 50 & 250 examples. The cases where the generations in GAs are $p = 1.000$ indicate that all GAs executed during 100 generations. These are the mean results of 20 runs for each algorithm.

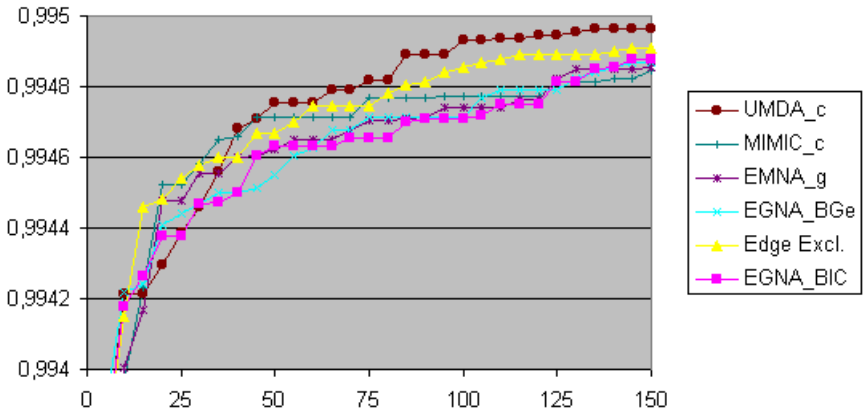


Figure 6.6: Graphs showing the best individual of the 10 & 30 case at each generation of the searching process for the continuous EDAs UMDA$_c$, MIMIC$_c$, EGNA$_{BGe}$, EGNA$_{BIC}$, EGNA$_{ee}$, and EMNA$_{global}$.

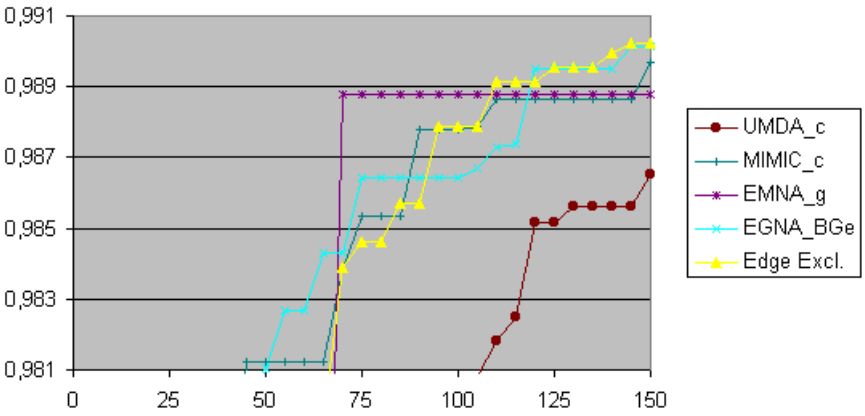


Figure 6.7: Graphs showing the best individual of the 30 & 100 case at each generation of the searching process for the continuous EDAs UMDA$_c$, MIMIC$_c$, EGNA$_{BGe}$, EGNA$_{ee}$, and EMNA$_{global}$.

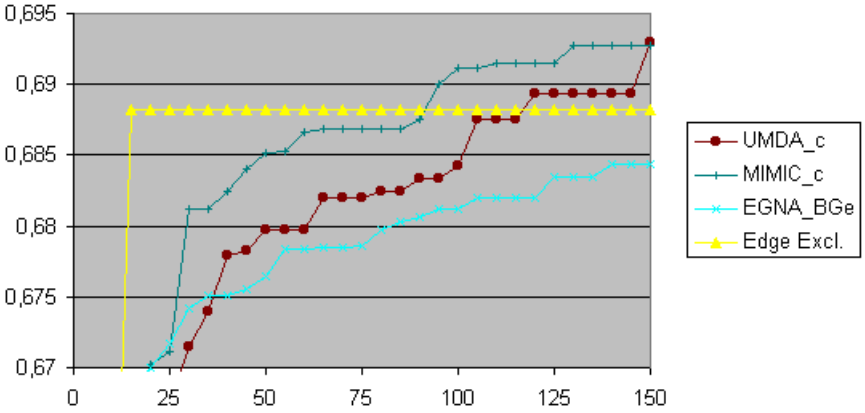


Figure 6.8: Graphs showing the best individual of the 50 & 250 case at each generation of the searching process for UMDA$_c$, MIMIC$_c$, EGNA$_{BGe}$, and EGNA$_{ee}$.

	10 & 30 best	10 & 30 time	30 & 100 best	30 & 100 time	50 & 250 best	50 & 250 time
$UMDA_c$	0.994964	00:10:36	0.986516	01:33:29	0.693013	10:13:14
$MIMIC_c$	0.994844	00:11:09	0.989696	01:35:29	0.692722	10:18:56
$EGNA_{ee}$	0.994909	00:14:29	0.990219	07:26:56	0.688189	67:08:11
$EGNA_{BGe}$	0.994887	00:13:23	0.990141	03:09:57	0.684401	461:59:00
$EGNA_{BIC}$	0.994878	02:29:41	–	–	–	–
$EMNA_{global}$	0.994855	00:32:09	0.988799	97:45:25	–	–

Table 6.9: Figures of the 3 cases of Study 1 for the continuous EDAs, obtained as the mean values after 20 executions of the continuous EDAs. The *best* column corresponds to the best fitness value obtained through the search.

ous domains are significant for all the algorithms analyzed. The null hypothesis of the same distribution densities was tested (non-parametric tests of Kruskal-Wallis and Mann-Whitney) for each of them with the statistical package S.P.S.S. release 9.00. These tests confirmed the significance of the differences in the results regarding the value of the best solution obtained. It is important to remember that all the solutions obtained by the continuous representation are correct, and therefore these results can be compared directly to any of the correction methods described for the discrete case. In many continuous versions of the EDA algorithms fitter results were obtained at the end of the search than their respective discrete versions, and it was only on the 50 & 250 example, where the results obtained by continuous EDAs are worse than the discrete EDAs. The main drawback of continuous EDAs is the longer execution time they require, which is extremely larger for the case of more complex continuous EDAs such as $EGNA_{BGe}$. In $EGNA_{BIC}$ and $EMNA_{global}$ the execution time was so high that after 500 hours of execution time the processes were aborted. These results show clearly that the behavior of selecting a discrete learning algorithm or its equivalent in the continuous domain is very different regarding all the parameters analyzed.

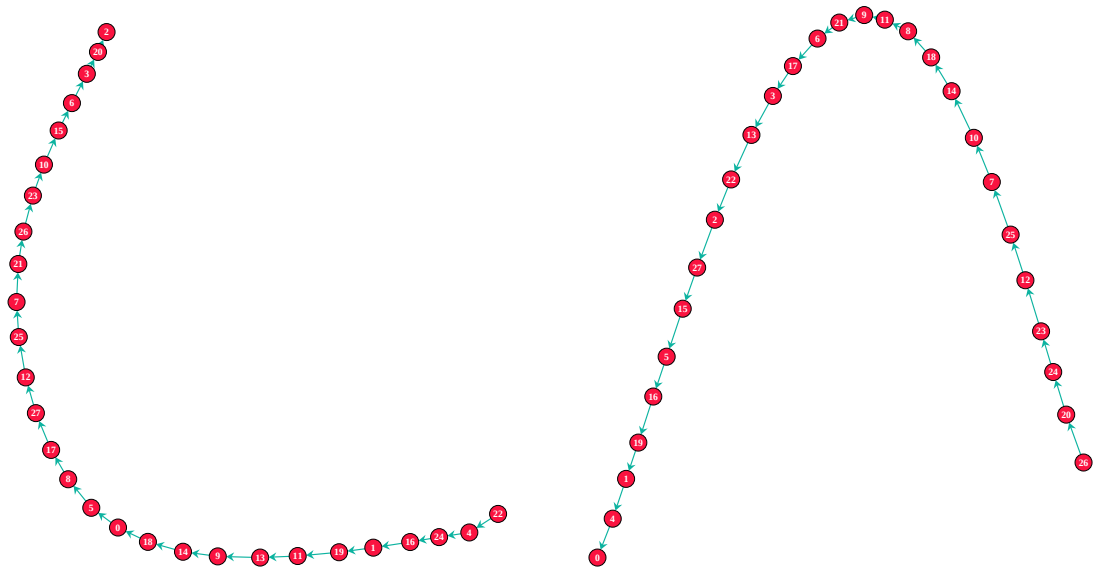
It is important to note that the number of evaluations was different as the ending criteria for the discrete and continuous domains have been set to be different. In all the cases, the continuous algorithms obtained a fitter individual, but the CPU time and number of individuals created was also bigger.

At the light of the results obtained in the fitness values, we can conclude the following: generally speaking, continuous algorithms perform better than discrete ones, either when comparing all of them in general or when only with algorithms of equivalent complexity.

6.3 Study 2: evolution of probabilistic graphical structures

The aim of this study is to analyze the evolution of the probabilistic graphical model complexity (the Bayesian network in the discrete case and the Gaussian network in the continuous one) so that the reader can have an idea of the complexity of the graph matching problem and the behavior of each of the different EDAs.

Due to the difficulty of visualizing a structure with as many nodes as shown in the graphs of the previous study, a smaller real example has been chosen. This example is taken from an image of human muscle cells, where the model graph G_M contains a vertex for each of the cells in the image. This image was over-segmented and the data graph G_D was obtained from it. In this particular example the graph G_M contains 14 vertices and 66



(a) 1^{st} generation (b) 21^{st} generation

Figure 6.9: Figures showing the structures learned during the MIMIC search in discrete EDAs.

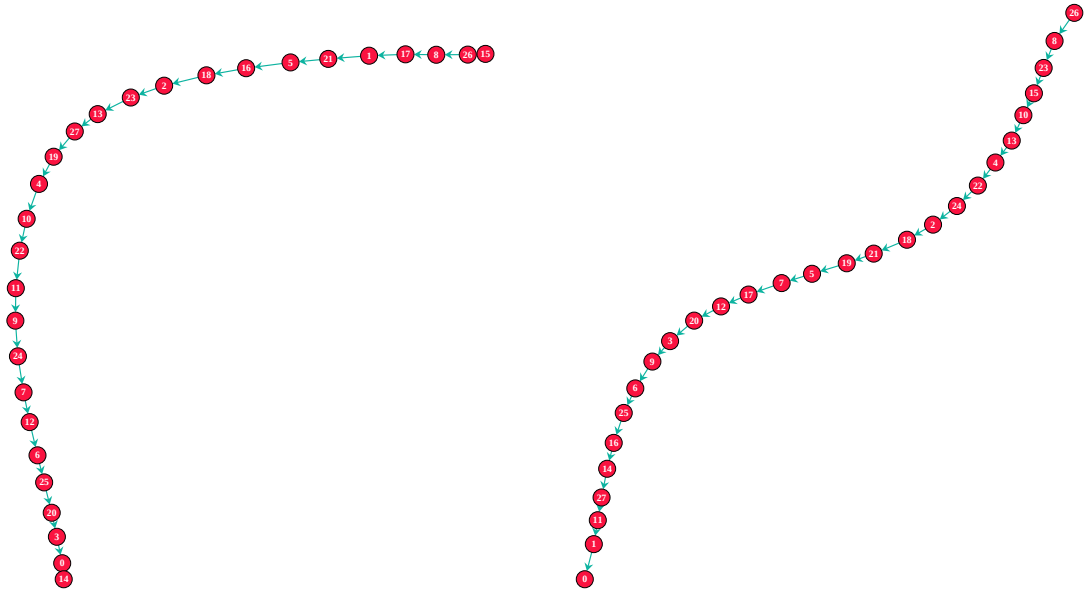
edges, and the data graph G_D contains 28 vertices and 124 edges. EDAs were applied to this example using individuals with a size of 28 variables ($|V_D| = 28$) in both discrete and continuous domains. Therefore, all the probabilistic graphical structures generated during the successive generation in the EDA approach using this representation of individuals is of 28 vertices. The number of edges of these structures symbolizes the number of dependencies between the different regions of the data image that the algorithm detects.

The discrete UMDA example is not shown in any figure, as it does consider all the variables as having no interdependencies. The assumed structure is the same as for UMDA$_c$, and it is shown in Figure 6.13a.

With discrete EDAs we obtain structures such as the ones illustrated in Figure 6.9 (for the MIMIC approach) and Figure 6.11 (for EBNA). These two examples show clearly that the algorithm is learning a structure according to the complexity we expected: MIMIC takes into account pairwise dependencies and generates a structure in the form of a chain in every generation, and only the order of the variables changes during the search. The EBNA algorithm imposes no restrictions to the number of dependencies that a variable can have, and therefore there is no limitation in the number of arcs that a node can have in every generation.

With continuous EDAs we can appreciate the analogous behavior: the continuous MIMIC$_c$ case is illustrated in Figure 6.10, where a behavior similar to Figure 6.9 can be seen. Again, this was expected as the discrete MIMIC does also consider pairwise dependencies. Nevertheless, the estimation of the distribution is performed using different methods in both algorithms according to the domain of the variables (MIMIC generated a Bayesian network and MIMIC$_c$ a Gaussian network).

With continuous EDAs we obtain structures such as the ones illustrated in Figure 6.10 (for the MIMIC$_c$ approach) and Figure 6.12 (for EGNA$_{ee}$). These two examples show clearly that the algorithm is learning a structure according to the complexity we expected: MIMIC$_c$



(a) 1^{st} generation (b) 150^{th} generation

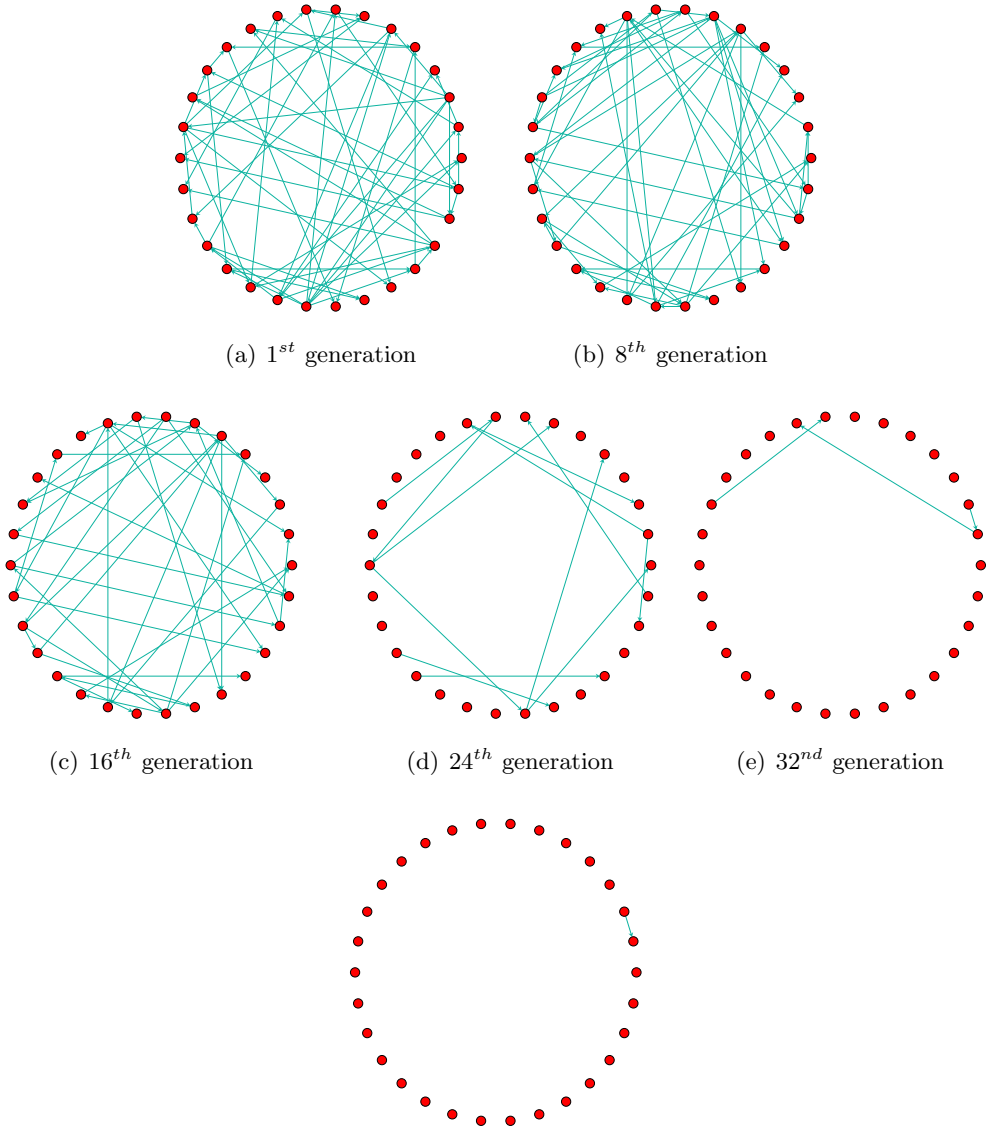
Figure 6.10: Figures showing the structures learned during the MIMIC$_c$ continuous EDA.

takes into account pairwise dependencies and generates a structure in the form of a chain in every generation, and the EGNA$_{ee}$ algorithm imposes no restrictions to the number of dependencies that a variable can have (there is no limitation in the number of arcs that a node can have in every generation).

Figure 6.13 is a special case, as the structures of both the UMDA$_c$ algorithm and EMNA$_{global}$ are always fixed during the whole search process (i.e. the estimation of the probabilities does not imply the learning of a structure, this is fixed), and therefore the structure is considered to exist as a fixed one.

We can appreciate in these experiments as well as in others such as the ones mentioned in [Bengoetxea et al., 2001c,e] that EBNA and EGNA algorithms, although they are analogous in complexity in their domains, they have different tendencies. Both algorithms do not set any restriction to the number of dependencies that variables can have (i.e. the probabilistic structures can have any number of arcs for each node). In EBNA, the algorithm tends to finish with an arc-less structure, which is influenced by the fact that in the last generations the best individual appears many times in the population, and therefore the algorithm finds the same value in a variable too often to detect dependencies regarding the rest of the variables –see Figure 6.11. In EGNAs, values are continuous and cannot be repeated as easily as in the discrete domain, and therefore as values are different the dependencies can also be found and represented as arcs in the structure. This is why at the last generations of the search EGNAs show structures with a lot of arcs. This effect can be appreciated for both EBNA and EGNA in Figure 6.14. In addition, there are also some additional factors that may influence this result:

- The complexity of the problem is not as high as expected, and therefore in this case UMDA and EBNA would return similar results.

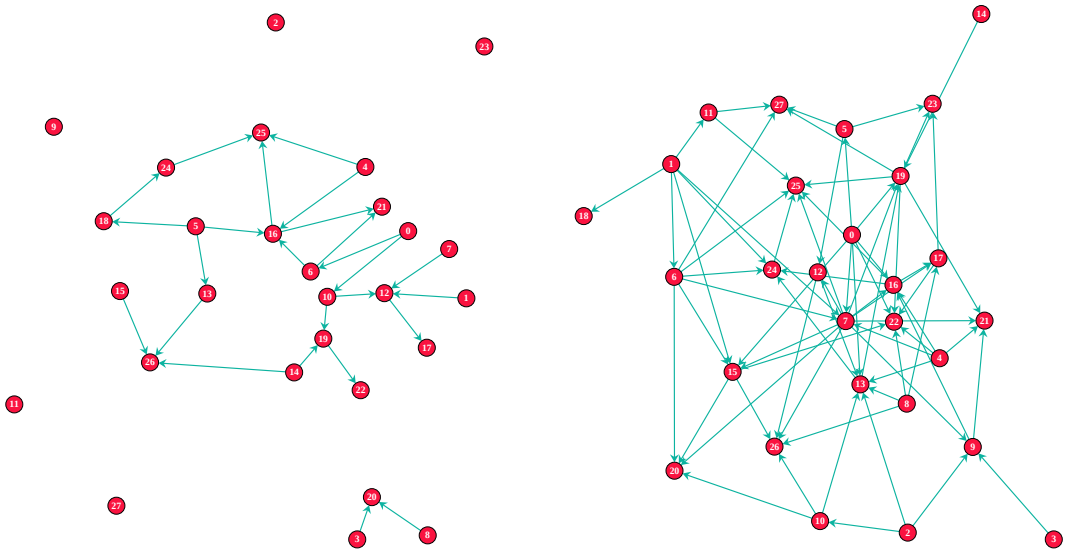


(a) 1^{st} generation (b) 8^{th} generation

(c) 16^{th} generation (d) 24^{th} generation (e) 32^{nd} generation

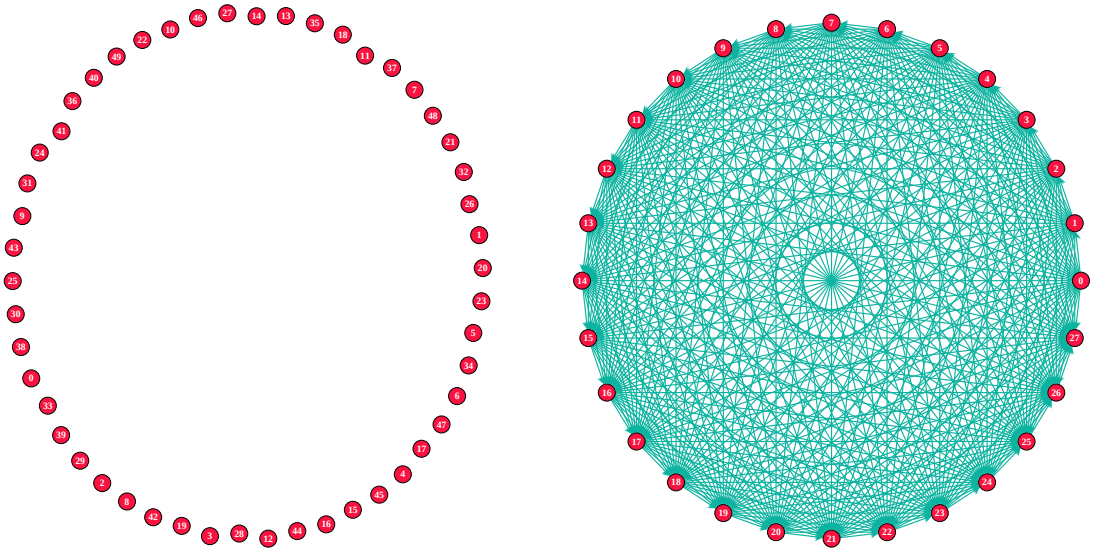
(f) 37^{th} generation

Figure 6.11: Graphs showing the evolution of the Bayesian network in a EBNA search, illustrating clearly that the number of arcs of the probabilistic structure decreases gradually from the first generation to the last ones. A circular layout has been chosen in order to show the same nodes in the same position. The number of arcs decreases as follows respectively: 57, 56, 40, 12, 3, and 1. After the 37^{th} generation and until the last (the 43^{th} one) the Bayesian network does not contain any arc.



(a) 1^{st} generation (b) 150^{th} generation

Figure 6.12: Figures showing the structures learned during the Edge Exclusion EGNA$_{ee}$ continuous EDA.

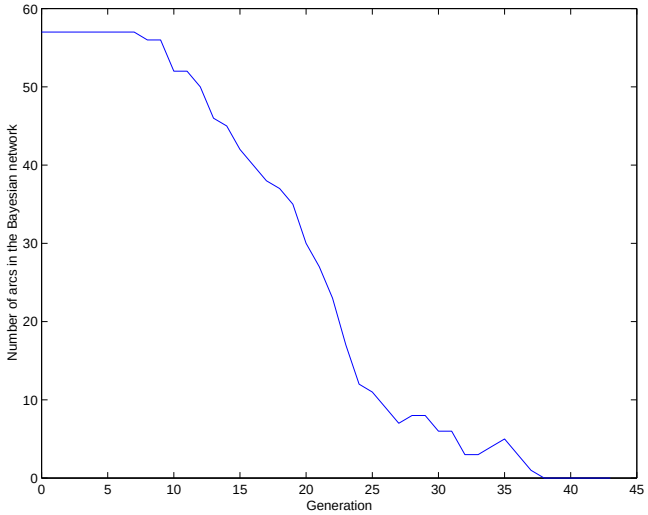


(a) UMDA$_c$ (b) EMNA$_{global}$

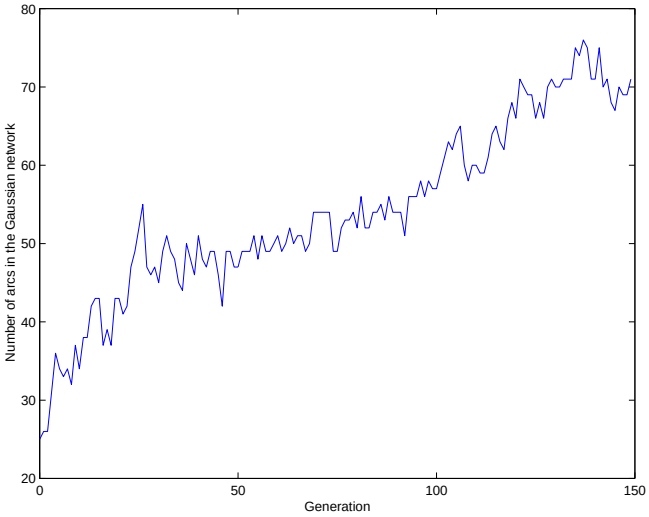
Figure 6.13: Figures showing the structures learned during the UMDA$_c$ and EMNA$_{global}$ continuous EDAs.

- The fact that using a representation with that many values per variable requires a bigger population per generation so that more complex dependencies can be analyzed.

It is important to note that in the former case, even if the best results obtained are similar, the execution time for EBNA would be much higher. This is caused by the first part of



(a) EBNA



(b) EGNA$_{ee}$

Figure 6.14: Graphs showing the evolution in number of arcs of the respective probabilistic structures for EBNA and EGNA$_{ee}$. The two different tendencies are illustrated: EBNA tends to a structure with less arcs when the search goes on, while EGNA-type algorithms tend to a structure with more arcs.

the learning process of EBNA that requires searching for the best structure to model the probability distribution, as in UMDA there is no such a step.

6.4 Study 3: parallelization

This study concentrates on the parallelization issues concerning EDA algorithms. Following the techniques and explanations given in Chapter 5, this section concentrates in the implementation and in obtaining experimental results to show the behavior of the different communication methods available in communication for parallel programs: the use of shared

memory and threads, and the use of message passing and processes in different machines. This section is subdivided in two parts regarding both paradigms.

6.4.1 Parallelizing EBNA using threads and shared memory

Taking into account all the properties of the BIC score described in Section 4.3.4 and analyzed in Section 5.5.3, we perform a basic modification on the sequential program in order to allow to compute in parallel all the arc modifications. For this task, the broadly known *pthreads* standard library is used, which allows threads to communicate through memory and to synchronize them using semaphores.

The score+search procedure is therefore organized in a very different way, as now threads have to divide the work: a thread plays the role of the *manager* that distributes the work among the rest, and the others are the *workers* that have to compute all the possible arc modifications for a same number of nodes.

Reorganizing the execution of the program

The fact of using a multithread program allows us to use shared memory for communication between them. This is implemented by creating a single process which contains many threads executing within its memory space. Global variables are defined, and these are used for direct communication between threads. However, the use of shared variables for communication leads to the existence of race conditions within the program. Therefore, a synchronization mechanism is required to ensure exclusive access to the critical sections in the program. The critical sections have to be identified by the programmer, and it is also his responsibility to integrate the synchronization primitives within the code.

In order to accomplish the parallelization task, the first thing to do is to choose a multithread standard library to program. The *pthreads* library is commonly available in many operating systems, so we decided to select it.

Once having decided this, the next step is to select the working scheme that all the parallel program will use for organizing the work of all the threads. EDA programs do have several parts that can only be executed sequentially, and the only part that we intend to parallelize is the computation of the BIC score each generation. Therefore the use of a master-slave scheme is very suitable for this case: a first thread will be executing the sequential parts of the EDA program, and when it reaches the step of estimating the probability distribution it will divide the work and send it to the workers. The worker threads will compute the BIC score.

An important aspect to consider is the number of worker-threads that will be created at the same time. We could think at a first glance that creating as many threads as possible is the best to finish the job, but it is important to take into account that creating a thread also has a cost associated, and that each thread has to be given enough work in order to justify its generation time. In addition, another limiting factor to decide how many simultaneous threads can be working is the number of CPUs of our system: having too many workers will lead to a system with workers competing between them to take ownership of CPUs instead of having them cooperating. For this reason, a semaphore is used to limit the number of threads created at any time. This semaphore is initialized to the maximum number of threads that can exist. In our case we have a two processor computer, and therefore this limit was set to 4 threads[3].

[3]In fact, since in our particular case we have only two processors in our computer, we could limit the

EDA Algorithm	10 & 30 ex. sequential exec. time	10 & 30 ex. parallel exec. time	50 & 250 ex. sequential exec. time	50 & 250 ex. parallel exec. time
$EBNA_{BIC}$	00:04:34	00:03:20	52:59:49	28:00:04
$EGNA_{BGe}$	00:13:23	00:16:35	461:59:00	67:48:01
$EGNA_{BIC}$	02:29:41	01:59:31	(*)	(*)

Table 6.10: Execution time for the 10 & 30 and 50 & 250 examples using the $EBNA_{BIC}$, $EGNA_{BGe}$ and $EGNA_{BIC}$ algorithms regarding their sequential and parallel versions of computing the BIC score (hh:mm:ss). The values with the symbol (*) required more than a month of execution time to be properly computed.

Experimental results of the multithread BIC procedure using shared memory

As the parallel BIC algorithm does not follow a new algorithm, the best fitness values obtained with the new parallel version of the EBNA approach are exactly the same as the sequential version. The only differences that can be expected are just in the execution time. Table 6.10 shows the effect of applying the parallel algorithm on the 50 & 250 example in both the $EBNA_{BIC}$ and $EGNA_{BIC}$ algorithms for their sequential and parallel versions.

The results show clearly that the use of threads reduces considerably the execution time for the 30 & 100 and 50 & 250 examples. A special mention is for the 10 & 30 example in the $EGNA_{BGe}$ case, as these particular results show that the parallel version requires longer time than the sequential program. This is the result of parallelizing the learning step: in Table 5.2 we can see clearly that the percentage of computing time for such an small example is of 5.6 %, while in the 50 & 250 example we obtain a computation percentage of 55 %. This illustrates that parallel programming techniques can improve the overall execution time of the program, but that the cost of creating new processes or threads has also to be taken into account, as already explained in the previous sections. In the small example with the $EGNA_{BGe}$ algorithm the relative weight of the function is so small in the whole execution time that each of the worker-threads do not have enough processing tasks in order to justify their creation, while in the big example the learning step is the one requiring the most computation time and the fact of being computed in different threads gives very satisfactory results.

For the rest of the cases and EDAs the final result was a considerable improvement in execution time in both the small and big examples, and the application of parallel programming techniques such as the use of the library *pthreads* appears to be very advisable.

In brief, the results obtained could be summarized as follows: multithread libraries applied on multiprocessors are a very powerful tool for programs such as EDAs that require a big amount of CPU time, but it is necessary to perform previously an analysis on the relative time consumed by each of its functions in order to be sure to be applying them correctly.

6.4.2 Parallelizing EBNA using processes and MPI

MPI is a message passing interface introduced in Section 5.3.3. As its name indicates, MPI has been designed to use message passing as the communication mechanism for inter-process

number of threads to 2. However, any program has always some small periods of time where a thread is blocked waiting for a operating system job, and another thread could use the CPU in the meanwhile. This is the reason why it is convenient to create some more threads than processors.

communication. Threads that are attached to the same process usually communicate through global variables in shared memory. However, any two threads from two different processes (either if the processes are on the same computer or connected through a network) cannot use shared memory and therefore message passing is the only possible mechanism. MPI provides an efficient mechanism for threads from different processes, and it can also been applied even when shared memory is available.

MPI has been designed as an interface for communicating processes that could even be executing in different computers at the same time. We have chosen the MPI implementation called MPICH for our experiments. The reasons for choosing both MPI and this particular implementation MPICH is their portability, good performance and availability for operating systems such as Windows, Linux and Solaris. MPICH even contains versions for very fast computer network configurations such as Myrinet, which allows us for further reduction in execution times.

As in MPI the communication is based on message passing primitives, the synchronization between sender and receiver is done implicitly. In addition, using MPI allows us to use a cluster formed by many 2-processor simple architecture PCs under Linux connected by a very fast local area network (LAN) to collaborate and cooperate each other by creating processes in all the machines without having to change the program. This means that in this case we can use the same master-slave working scheme as with the parallel version of the *pthreads* library, but this time workers would be processes that could be executing in different CPUs and even in different computers at the same time.

However, in the particular example of the EDA program, the fact that processes cannot share any memory among them forces the manager to send all the data structures required to compute the BIC function to each of the workers. Afterwards when all the workers have completed their part of the job, each one will have to send a message to the master with the amount of work done. In addition, as in MPI processes are created and not threads, it is important to take into account that the creation of a process requires more time that creating a thread. Moreover, if processes are to be created in other computers within a cluster this operation will take even longer.

All the latter consideration make us modify partly the structure of the EDA program. In addition, the fact that in MPI the master and the slaves have to execute the same program and that processes cannot be created dynamically is also another reason for a deep restructure of the whole sequential program.

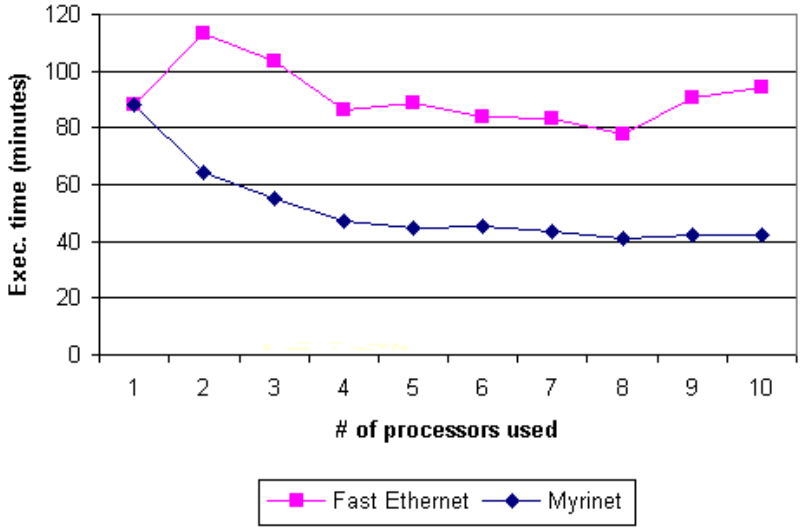
Experimental results of the parallel BIC procedure using MPI

The implementation of the EBNA$_{BIC}$ parallel version using MPI was tested on a different machine than the one used in the multithread program. The reason for this is that in this case for an efficient use of MPI a cluster of workstations is more suitable rather than a single machine with fast processors. In our case, we tested the parallel program based on MPI using a cluster formed by 5 computers with 2 Intel Pentium II processors at 350 MHz, 512 KBytes cache, and 128 MBytes of RAM each. The operating system used is GNU-Linux. These computers are connected by two different local area networks: one is a Fast Ethernet and Myrinet. The different between them is that Myrinet has a bigger bandwidth, provides shorter latency times for communication, but it is much more expensive. The decision of using a network or another does not imply any modification on the source code, and it does only affect the compilation options of the MPI distribution. Execution times with both types of local area networks (LANs) were tested and are presented in this section.

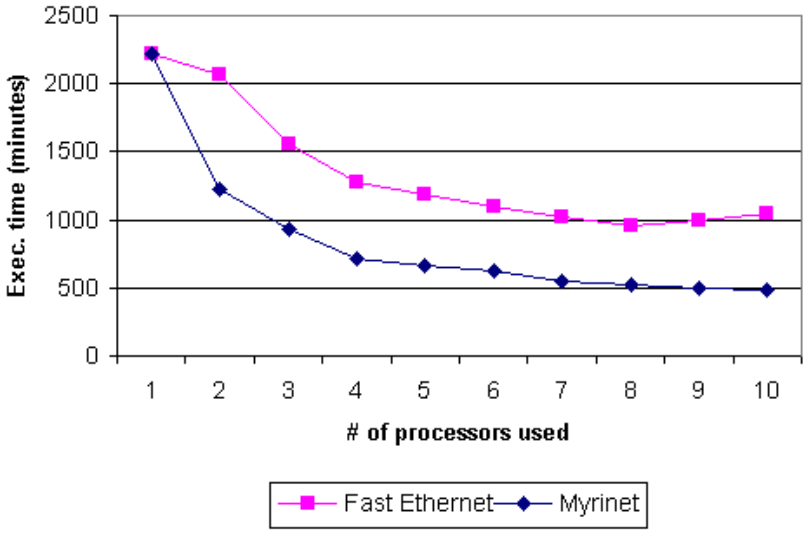
Different experiments were performed using both the 30 & 100 and 50 & 250 examples of Study 1. Both networks were tested using different number of processors (i.e. number of worker-processes created). Figure 6.15 shows the evolution of the execution time for the two examples and in the two networks. The case of the 30 & 100 example with Fast Ethernet illustrated in Figure 6.15a is specially illustrative, as it shows the typical case in which in two moments the fact of increasing the number of workers also increases the execution time required. The explanation of this is as follows: the first and initial increase in execution time is due to the added computation required to implement the communication between workers and the manager, while the latter at the end is a result of dividing too much the work load per worker and therefore requiring more time to create each of them rather than having them working. In between both moments, we have a part in which the parallelization techniques give the expected results of reducing the execution time. In this example but for the Myrinet network this does not happen because the network is faster and therefore the overhead created by the communication is also smaller.

The 50 & 250 example presents acceptable figures for both the Fast Ethernet and Myrinet networks. Both graphics in Figure 6.15 show also the differences in data transfer speed between the Fast Ethernet and Myrinet networks. On the other hand, with smaller examples the increase in execution time could also happen even when using a fast network such as Myrinet. These results are also shown in numeric format in Table 6.11. Therefore, our experiments show the expected effects when parallelizing complex and CPU intensive programs, and these also serve as an idea of the minimum size that the problems need to have in order to obtain shorter execution times. This table also shows that in both the Fast Ethernet and Myrinet networks, the optimum number of workers is 7 (1 manager process + 7 worker processes), and for the Myrinet case the higher the number of processes the shorter the execution time (we arrived until a total of 9 worker processes), although in the Myrinet case we would find a moment after which the execution time would also show the increase when augmenting too much the number of workers. However, these results are only valid for the particular configuration of the cluster where the experiments were carried out, and will have considerable differences depending on the RAM memory available, and the number and working frequency of the processes in each computer that forms the cluster.

Figure 6.16 is an example of the types of traces that we can obtain using the MPICH implementation of MPI. These traces are used to identify bottlenecks in MPI programs, but here are presented for illustrating the communication requirements that the evolution of each generation requires. The state and MPI primitive executed by each of the processes are shown using different colors. These figures demonstrate that while the manager is executing the rest of the workers are simply waiting, and therefore no parallelization is occurring during this time. Later, there is a phase in which the manager is executing *sending* operations and workers are mostly executing receiving ones. The latter is the time in which the work is being distributed to and executed by the workers while the manager is administrating and coordinating all the job. This repeats once and again until the total number of generations have been completed. Obviously, in order to achieve a considerable reduction in execution time the parallel phase needs to occupy most of the time in the diagram, which has been shown to happen in our case when having graphs of big sizes. The speed of the network is also another factor to consider, and this fact is shown in the different times illustrated in the figure.



(a) 30 & 100 example



(b) 50 & 250 example

Figure 6.15: Illustration of the evolution in execution time when using MPI and depending on the number of processes.

6.5 Conclusions of the studies on synthetic problems

Very different studies have been described in this chapter, from which different results and conclusions for the application of these techniques on real graph matching problems. We can summarize them as follows:

Study 1: the main conclusion that is obtained from this study is that when additional constraints are present in real problems a mechanism is available in EDAs to ensure that the final solution will satisfy all of them. Specially in graph matching problems that have a complexity similar in size to the ones obtained in real images, the control of

Number of Processes	Total Time Fast Eth.	Speed Up Fast Eth.	Total Time Myrinet	Speed Up Myrinet
1	01:28:07	1	01:28:07	1
2	01:53:03	0.779	01:04:28	1.368
3	01:43:40	1.850	00:54:59	1.602
4	01:26:04	1.023	00:47:19	1.862
5	01:28:55	0.991	00:44:47	1.967
6	01:23:47	1.052	00:45:13	1.949
7	01:23:02	1.061	00:43:33	2.023
8	01:17:57	1.130	00:41:02	2.147
9	01:30:22	0.975	00:42:18	2.083
10	01:34:20	0.934	00:42:02	2.096

(a) 30 & 100 vertices

Number of Processes	Total Time Fast Eth.	Speed Up Fast Eth.	Total Time Myrinet	Speed Up Myrinet
1	37:00:09	1	37.00:09	1
2	34:24:55	1,075	20:21:01	1.818
3	25:54:22	1.428	15:29:36	2.388
4	21:16:22	1.739	11:50:02	3.126
5	19:50:30	1.864	11:06:42	3.330
6	18:15:10	2.027	10:20:06	3.580
7	16:59:37	2.177	09:10:19	4.034
8	15:52:58	2.329	08:42:15	4.251
9	16:30:14	2.242	08:21:53	4.423
10	17:22:47	2.129	08:10:07	4.529

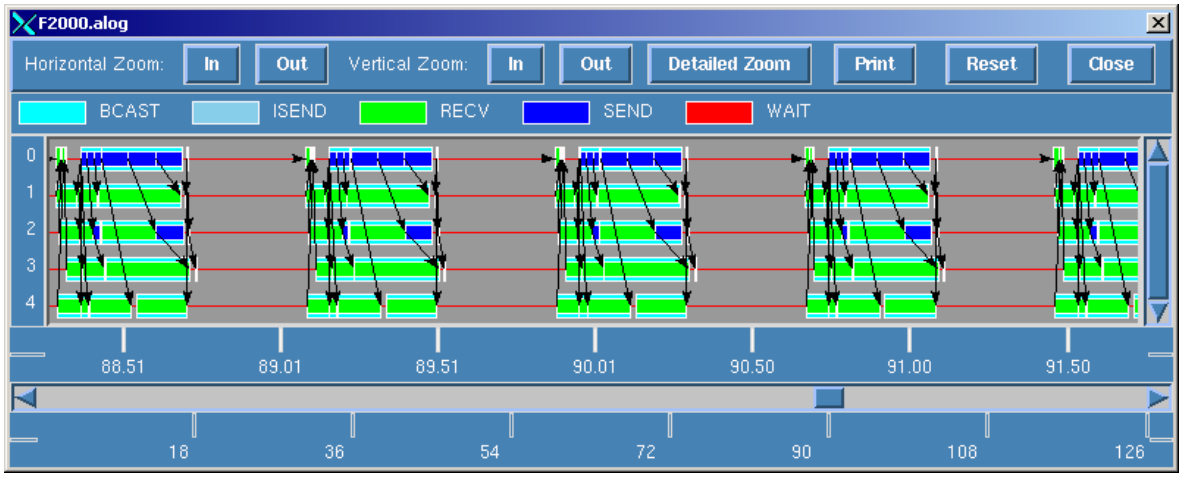
(b) 50 & 250 vertices

Table 6.11: Execution times obtained when increasing the number of processes (processors used) for the medium-sized example with 30 & 100 vertices (above) and for the big example with 50 & 250 vertices (below). Times are presented in *hh:mm:ss* format.

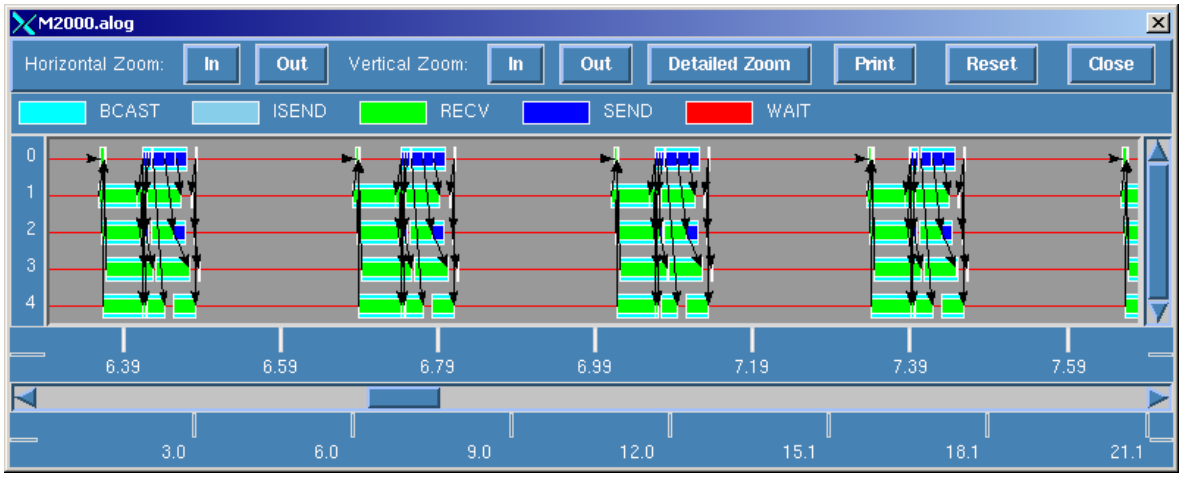
constraints can be performed applying to EDAs the methods described in Section 4.5.1.

In addition, this study presents a comparison of the performance of the different EDAs and other evolutionary computation algorithms which illustrate their differences depending on the complexity of the graph matching problem. In addition, the same graphics show the evolution in best solution obtained per generation, which can be used to select an algorithm over the rest regarding the stopping criterion of the search. In any case, EDAs show on the whole a better performance than other evolutionary computation techniques for both the discrete and continuous domains.

It is important to note that the behavior of the algorithms is always very dependent on the fitness function selected. Fitness functions are defined for each particular problem, and algorithms also adapt in a different way to the type of fitness functions. From the experiments carried out we can see that the different algorithms have different tendency to fall on local maxima (or local minima if the fitness function has to be minimized)



(a) Trace of execution in a Fast Ethernet network



(b) Trace of execution in a Myrinet network

Figure 6.16: Figures for the communication mechanisms and other MPI primitives on the parallel version of EBNA$_{BIC}$ for the particular configuration of our cluster.

or to scape from them. In this study, EDAs showed a better propensity to avoid local maxima, although in fitness function with less local maxima GAs obtain the optimum in a shorter time.

Study 2: this study illustrates the behavior of the different EDAs during the search process. This behavior is specially interesting in case of EBNA and EGNA, as they take into account all the possible dependencies. This study illustrates the evolution of the probabilistic graphical models on EDAs and explains why they converge to solution at the end of the search process.

Study 3: from the results presented in Study 3, we can conclude that the mechanism used in this thesis reduces considerably the execution time required for CPU intensive programs such as EBNA$_{BIC}$. The two versions of the parallel program, the multithread and the MPI ones, can be applied to multiprocessors and clusters respectively according to the hardware availability. In addition, results of two types of LANs are presented showing

the relevance of the network speed on the execution time.

In both cases, the paradigm applied is a master-slave scheme as described in Section 5.6. This mechanism has been designed for its easy adaptation to other EDAs or evolutionary computation algorithms. The source code of the multithread and MPI version including detailed explanations about the communication and synchronization implementations is presented in Appendix D.

However, another important conclusion that is obtained from these results is that the number of workers is an important parameter that is very dependent on the characteristics of the computers that has to be chosen with care in order to obtain a satisfactory reduction in execution time without increasing the overhead of the communication between the workers and the master.

Chapter 7

Experiments with real examples

" The most exciting phrase to hear in science, the one that heralds new discoveries, is not 'Eureka! ' but 'That's funny ...' "

Isaac Asimov

7.1 Introduction

After the experiments carried out with synthetic data in the previous sections, this chapter presents experimental examples carried out with real data. These real problems present different characteristics and restrictions that have to be taken into account by the graph matching algorithm.

The first application is the recognition of healthy human brain MR images in three dimensions. Secondly, the inexact graph matching example of the recognition of facial features on human faces is tackled.

7.2 Recognition of brain images

7.2.1 Motivation

The recognition of internal structures of the human brain in magnetic Resonance images (MRi) using anatomical atlas presents nowadays an increasing interest, and many publications can be found in the literature on applications derived from this field in areas such as morphometry, localization of pathologies regarding abnormal structures in the brain, study of the evolution of pathologies, and support for functional studies of the brain. In all these applications the individual anatomy and the recognition of the internal structures is a preliminary step. Moreover, the step of automatically recognizing brain images by computers is a goal that could improve considerably the diagnosis and work of neurologists and radiologists.

The main difficulties of such a recognition are the differences in the orientation of the images and in gray level distributions between machines, but mainly the differences are due to the structural particularities of healthy individuals (the many variations from a person to another in composition, size and orientation on these different regions), and the presence of possible pathologies. Most of the commercial programs that nowadays perform a similar recognition procedure are based on the analysis of the grey levels of the image. Others, more sophisticated, are able of identifying regions under variation of grey level distributions on

images, although this always require the help of the physician to ensure that the detection is satisfactory enough.

More recently digital anatomical atlases have been proposed as a valid generic support for representing the different brain regions and their variability.

The use of graphs tries to encapsulate all the anatomical knowledge of a healthy brain independently of these factors. In the considered application, the atlas (or model) is represented as an attributed graph, and the images to be recognized undergo an automatic segmentation procedure after which its characteristics are also represented in the form of a data graph. The whole recognition process is therefore proposed as a graph matching problem.

Regarding the input image from which the data graph is generated, it is a common practise to apply over-segmentation to the data image previously [Perchant et al., 1999, Perchant and Bloch, 1999] to ensure that all the boundaries between regions in the model appear also in the data image. This makes graph matching easier as it ensures that a vertex in G_D is matched only against a single vertex in G_M. Unfortunately, as a consequence of this procedure even more segments are created by further subdividing the input image, which at the same time results in an increase on the number of vertices in the data graph and hence in the complexity of the problem itself. Due to this reason, the isomorphism condition is too restrictive and therefore this problem is actually regarded as an inexact graph matching one.

7.2.2 Construction of the atlas and data graphs

The atlas graph[1] that we use as a general reference has been constructed with the aid of medical doctors, and it contains a vertex for each of the brain regions such as cerebellum, ventricles, and corpus callosum with all the attributes of each of them such as approximate size, approximate position in the brain, and grey level distribution in MR images. Our particular model contains 43 regions, and therefore the model graph G_M has 43 vertices. In our model the graph is not complete and only edges between neighboring regions are considered. The model graph that we used as an example contains 336 edges.

The size of the data graph G_D used in our experiments contains 245 vertices and 1451 edges. Figure 7.1 illustrates how both the atlas and data graphs are generated.

The vertex and edge attributes that we use in our particular example have been obtained from the problem described in [Perchant, 2000, Perchant and Bloch, 2000b]. The particular attributes that are considered for both the atlas and data graphs are the grey level distribution for the vertices, and the relative position and distance for the edges. These attributes are founded on fuzzy set theory, and all the information that is required for comparing each of them is stored in the form of the necessity and possibility values. The authors also define a method to combine these fuzzy attributes to create similarity measures between vertex and edge attributes of the atlas and data graphs, and these vertex and edge similarities are the basis for creating fitness functions on their example. This method provides a means to obtain the vertex similarity value $c_N(a_D, a_M)$ between any two vertices $a_D \in V_D$ and $a_M \in V_M$, as well as the edge similarity value $c_E(e_D, e_M)$ between any two edges $e_D = (a_D^i, a_D^j) \in E_D$ and $e_M = (a_M^k, a_M^l) \in E_M$. Both $c_N(a_D, a_M)$ and $c_E(e_D, e_M)$ are normalized in $[0, 1]$.

[1]The model graph is usually known as the *atlas graph* in the literature on this particular problem.

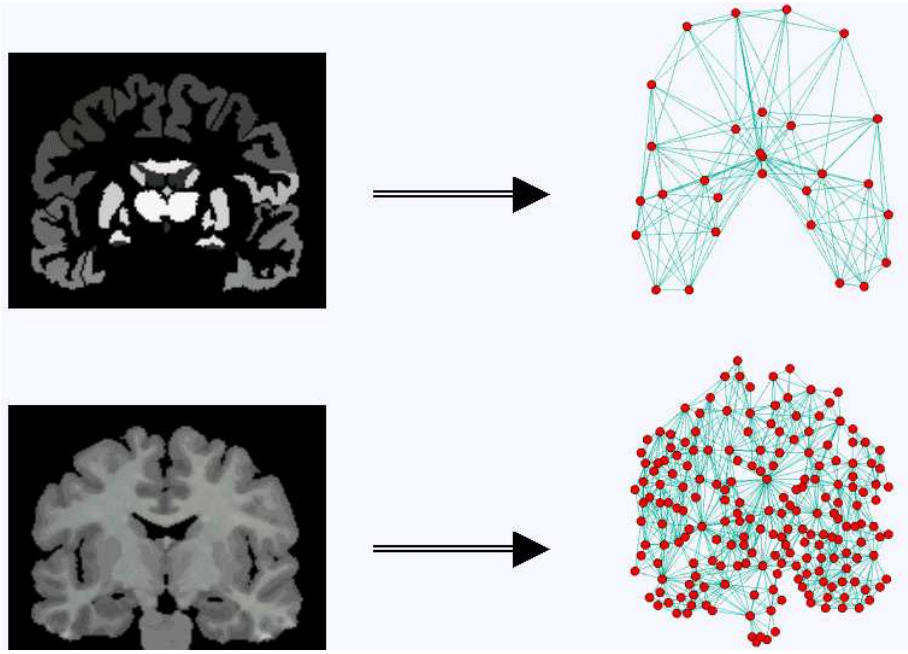


Figure 7.1: Illustration of the generation of the atlas (above) and the data graph (below). The atlas is created from a real 3D MR image of a healthy human brain manually segmented as we can see above. The data graph G_D is created segmenting the input image to be recognized. All the images are always in 3D, although here we show only a coronal view of them. Both graphs on the right show the complexity of the problem. The graph matching procedure has to assign a model vertex for each of the vertices in the data graph.

7.2.3 Description of the problem and the graph matching approach

Figure 7.2 shows the matching procedure of this problem and the meaning of the results. This figure illustrates how the successful recognition of the caudate nucleus (one of the brain regions) is performed using graph matching. As the image is acquired in 3D, two columns corresponding to an axial and a sagittal view are presented. The over-segmentation is also evident and can be appreciated in the data graph compared to the atlas graph. In the input image, the caudate nucleus is represented by two different segments in this particular example, although depending on the segmentation process this also could have been divided in even more segments (or may be in just a single one).

In the atlas (above) one vertex represents the caudate nucleus, and edges from this vertex to the neighboring vertices represent spatial relationships between them. The images below show the corresponding parts identified as the caudate nucleus once the atlas and data graph have been matched satisfactorily.

Any successful graph matching method is expected to obtain as a final result the matching illustrated in the row below for the case of the caudate nucleus. Similar examples could be given for the rest of the brain segments. However, it is important to point out that the search process relies on the fitness function and the individual representation, and therefore the proper choice of these two aspects will be responsible to a large extent for obtaining a satisfactory result at the end. The fitness function that we have selected for our experiments with this example has been obtained from the literature in [Perchant et al., 1999] and it

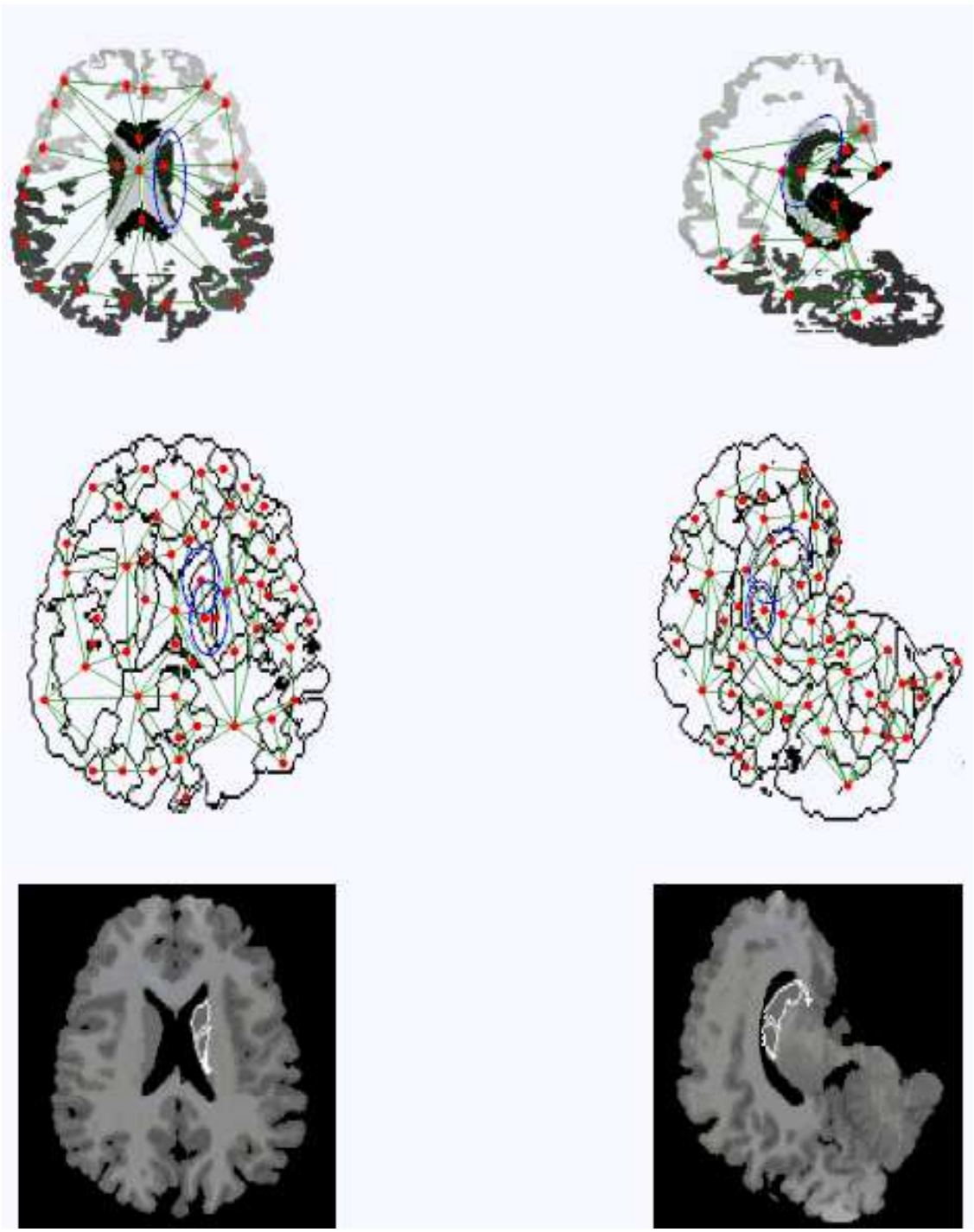


Figure 7.2: Example of the graph matching process applied to the recognition of structures in MR human brain images. In each row we have an axial and a sagittal view of the model graph, data graph, and a result respectively. All these images concentrate on the recognition of a particular segment of the brain among all the ones to be identified: the caudate nucleus.

follows the idea of the one introduced in Equation 3.2 in Section 3.4.3. On the other hand, the individual representations used here for applying discrete and continuous EDAs are the ones described in Sections 3.3.1 and 3.3.2 respectively. The procedure to evaluate individuals in the continuous domain with the same fitness function is the one explained also in Section 3.3.2 and in Appendix A. In both the discrete and continuous domains, $\alpha = 0.4$ was considered following the conclusions of [Perchant et al., 1999].

This real problem has also specific constraints that have to be satisfied for any solution in order to accept it as valid. The conditions of this problem are exactly the same as in the synthetic case analyzed in Section 6.2: no dummy vertices are allowed, only a matching is possible for each vertex in the data graph, and there is an additional constraint so that there must exist at least a region in the input image assigned to every vertex in the model. The latter condition is justified since we are dealing only with healthy brains and therefore all the brain regions have to be identified. All these aspects have already been discussed and formalized in Section 3.3.3.

7.2.4 Experimental results

Description of the experiment

In this section we compare the performances of EDA algorithms to each other and to a broadly known GA, the GENITOR [Whitley and Kauth, 1988] –a steady state (ssGA) type algorithm. Both EDAs and GENITOR are implemented in ANSI C++ language, and the experiment was executed on a two processor Ultra 80 Sun computer under Solaris version 7 with 1 GByte of RAM.

The initial population for all the algorithms was created using the same random generation procedure based on a uniform distribution. In the discrete case, all the algorithms were set to finish the search when a maximum of 100 generations or when uniformity in the population was reached. GENITOR, because of being a ssGA algorithm, only generates an individual at each iteration, but it was programmed in order to generate the same number of individuals as in discrete EDAs by allowing more iterations (201900 individuals). In the continuous case, the ending criterion was to reach 301850 evaluations (i.e. number of different individuals generated).

In EDAs, the following parameters were used: a population of 2000 individuals ($R = 2000$), from which a subset of the best 1000 are selected ($N = 1000$) to estimate the probability, and the elitist approach was chosen (that is, always the best individual is included for the next population and 1999 individuals are simulated). In GENITOR a population of 2000 individuals was also set, with a mutation rate of $p_m = \frac{1}{|V_D|}$ and a crossover probability of $p_c = 1$. The crossover and mutation operators used in GENITOR are CX [Oliver et al., 1987] and EM [Banzhaf, 1990] respectively.

Results

Results such as the best individual obtained, the CPU time, and the number of evaluations to reach the final solution were recorded for each of the experiments. The CPU time is given as a measure to illustrate the different computation complexity of all the algorithms.

Each algorithm was executed 10 times. The non-parametric tests of Kruskal-Wallis [Kruskal and Wallis, 1952] and Mann-Whitney [Mann and Whitney, 1947] were used to

	Best fitness value	Execution time	Number of evaluations
UMDA	0.7186	00:53:29	85958
MIMIC	0.7027	00:57:30	83179
EBNA_{BIC}	0.7167	01:50:39	85958
UMDA_c	0.7450	03:01:05	301850
MIMIC_c	0.7480	03:01:07	301850
EGNA_{BGe}	0.7469	04:13:39	301850
ssGA	0.6936	07:31:26	201900
	$p < 0.001$	$p < 0.001$	$p < 0.001$

Table 7.1: Mean values of experimental results after 10 executions for each algorithm for the inexact graph matching problem of the human brain structure recognition.

test the null hypothesis of the same distribution densities for all –or some– of them[2]. This task was done with the statistical package S.P.S.S. release 9.00. The results for the tests applied to all the algorithms are shown in Table 7.1. The study of particular algorithms gives the following results:

- Between algorithms of similar complexity only:
  - UMDA vs. UMDA_c. Fitness value: $p < 0.001$; CPU time: $p < 0.001$; Evaluations: $p < 0.001$.
  - MIMIC vs. MIMIC_c. Fitness value: $p < 0.001$; CPU time: $p < 0.001$; Evaluations: $p < 0.001$.
  - EBNA vs. EGNA. Fitness value: $p < 0.001$; CPU time: $p < 0.001$; Evaluations: $p < 0.001$.

  These results show that the differences between EDAs in the discrete and continuous domains are significant in all the cases analyzed, meaning that the behavior of selecting a discrete learning algorithm or its equivalent in the continuous domain leads to very different results. It is important to note that the number of evaluations was expected to be different, as the ending criteria for the discrete and continuous domains are also different. In all the cases, continuous EDAs obtained a fitter individual, but the CPU time and number of individuals created was also bigger.

- Between discrete algorithms only:
  - Fitness value: $p < 0.001$. CPU time: $p < 0.001$. Evaluations: $p < 0.001$.

  In this case statistically significant different results are also obtained in fitness value, CPU time, and number of evaluations. The discrete algorithm that obtained the best result was UMDA, closely followed by EBNA. The differences in the CPU time are also according to the complexity of the learning algorithm they apply. Finally, the results show that MIMIC required significantly less individuals to converge (to reach the uniformity in the population), whereas the other two EDA algorithms require nearly the same number of evaluations to converge. The genetic algorithm GENITOR is far behind the performance of EDAs. The computation time is also a factor to

[2]The interested reader is referred to [Siegel, 1956] for further explanations on non-parametric tests.

consider: the fact that GENITOR requires about 7 hours for each execution shows the complexity of the graph matching problem.

- Between continuous algorithms only:

  – Fitness value: $p = 0.342$. CPU time: $p < 0.001$. Evaluations: $p = 1.000$.

  Differences in fitness values between all the continuous EDAs appear to be not significant. As expected, the CPU time required for each of them is according to the complexity of the learning algorithm. On the other hand, the fact of having the same number of evaluations is due to the same ending criterion. Speaking about the differences in computation time between discrete and continuous EDA algorithms, it is important to note that the latter ones require all the 300000 individuals to be generated before they finish the search. The computation time for the continuous algorithms is also longer than their discrete equivalents as a result of several factors: firstly, due to the higher number of evaluations they perform each execution, secondly because of the longer individual-to-solution translation procedure that has to be done for each of the individuals generated[3], and lastly, as a result of the longer time required to learn the model in continuous spaces.

We can conclude from these results that generally speaking continuous algorithms perform better than discrete ones, either when comparing all of them in general or only with algorithms of equivalent complexity.

7.2.5 Computational complexity and parallelization of the problem

The complexity of the human brain recognition problem is high due to the inherent complexity of the brain structures and the fact of having 3D images [Bengoetxea et al., 2002b]. Unfortunately, such a high complexity results in very long computation time if a satisfactory matching is to be found. This section focuses on this aspect to demonstrate the need and usefulness of applying parallelization techniques.

Analysis of the execution times for the most important parts of the sequential EDA program

Section 5.5.2 underlines the need of analyzing the potential for parallelization of a program by profiling it. That section showed how to perform this step for the specific case of EBNA$_{BIC}$ using the GNU gprof tool. Table 7.2 summarizes the most salient results for this particular problem.

The BIC function computes the BIC score –i.e. the goodness of the probabilistic structure to represent the data. This function is executed many times each generation, until the best Bayesian network is found. This score constitutes essentially the whole learning of the Bayesian network. Table 7.2 shows clearly that in EBNA$_{BIC}$ the most time consuming function is BIC, the one that is used to evaluate the different Bayesian networks for representing the interdependencies between the variables.

We exploit the use of the two parallel versions of the EBNA$_{BIC}$ algorithm for this problem, the first based on threads and shared memory, and the second on processes and message passing.

[3]See Section 3.3.2 for a further explanation of this individual-to-solution procedure.

EDA type	Internal function	Relative CPU time
$EBNA_{BIC}$	BIC (Learn probabilistic model)	85.7 %
	Evaluation of individuals (Fitness function)	12.3 %
	Simulation	1.4 %

Table 7.2: Time of computing for the human brain recognition graph matching problem solved with $EBNA_{BIC}$. All the figures in the last column are given in relative times, i.e. 100 % = full execution time.

	sequential	parallel 2 workers	parallel 4 workers	parallel 6 workers	parallel 8 workers
Machine 1	26:49:48	17:40	16:55	17:08	16:11
Machine 2	10:54:15	7:14	6:56	6:50	6:57

Table 7.3: Execution times for the human brain recognition problem using the $EBNA_{BIC}$ algorithm regarding its sequential and shared memory based parallel *pthread* version for computing the BIC score (hh:mm). Results show important improvements in execution times.

Experimental results using threads and shared memory

The parallel version of the BIC function using threads does not compute any different algorithm regarding the sequential BIC function. For this reason, the best fitness values obtained with the parallel version of the $EBNA_{BIC}$ approach are exactly the same obtained with the sequential program. The only difference is just the execution time required. Two small SMP machines with different hardware configurations have been used to evaluate the performance of a parallel pthreads-based version of the $EBNA_{BIC}$ program, which characteristics are as follows: the first computer, *Machine 1*, contains 2 Intel Pentium II processors at 350 MHz, 512 KB cache, and 128 MB of RAM; the second computer, *Machine 2*, has 2 Intel Pentium III processors at 1 GHz, 256 KB cache, and 512 MB of RAM. Both machines use the operating system GNU-Linux and its programming environment. Table 7.3 compares execution times for sequential and parallel versions of $EBNA_{BIC}$. Note that parallelism does not come for free: the parallel master-worker paradigm can be applied to a single worker –thus, achieving no parallelism at all. Using Machine 2 with a single-worker parallel program, the execution time is 12:08:49, i.e., more than 1 hour slower than the original, sequential version. This cost comes from spawning the worker thread and synchronizing it with the master.

Of course, real improvement can only be apparent when two or more worker threads run in parallel. Table 7.3 shows that, with two workers runs are much shorter. We tried using more than two worker threads: additional gains are achieved, but not very significant. Generally speaking, the results obtained could be summarized as follows: parallel programming techniques based on shared memory appear to be an effective acceleration tool for very CPU-consuming programs such as EDAs. Programmers need to familiarize themselves with libraries such as *pthreads* (or equivalent ones) to take full advantage of the potential of parallel machines.

Experimental results using message passing

A similar experiment was performed using the parallel version of the $EBNA_{BIC}$ algorithm based on MPI. The program was executed in the cluster which characteristics were explained

	parallel 2 workers	parallel 4 workers	parallel 6 workers	parallel 8 workers	parallel 10 workers
Fast Ethernet	27:04	19:21	17:23	15:59	17:25
Myrinet	17:36	13:04	11:41	10:53	10:42

Table 7.4: Execution times for the human brain recognition problem using the EBNA$_{BIC}$ algorithm regarding its sequential and message passing based parallel MPI version for computing the BIC score (hh:mm). The execution time of the sequential version on one of the machines is 26 hours and 49 minutes. Results show the validity of the parallel system.

in Section 6.4.2. As well as in that section, the Fast Ethernet and Myrinet computer networks were tested. The results are shown in Table 7.4. The results obtained also show considerable reductions in the computation time, specially for the Myrinet case. Similar results as in the previous section are obtained, although the performance of MPI is still better than such of the pthreads one.

7.3 Recognition of human facial features

7.3.1 Motivation

Face analysis and recognition have received intense and growing attention from the computer vision community, partially because of the many applications such as human-computer interaction, model-based coding, teleconferencing, security and surveillance. A particularly important task that arises in different problems of face recognition is the location and segmentation of *facial feature regions*, such as eyebrows, iris, lips, and nostrils [Pantic and Rothkrantz, 2000]. Some of the most popular methods such as techniques based on template matching, integral projections, and so on are also discussed in [Pantic and Rothkrantz, 2000].

We propose the use of graph matching to solve this problem. In this approach, we have again one graph representing a model of a face and an image for which recognition has to be performed. Both the model and data graphs are built from regions and relationships between regions, and there are vertex and edge attributes in both of them too. The idea is to model basic knowledge about the features in a face as a graph. Based on an over-segmentation, image information is also represented as a graph. More specifically, an attributed relational graph is used to represent each image. Therefore, the recognition procedure expects to find a suitable matching between both graphs. The introduced technique can be applied to both static images and to video sequences.

7.3.2 Construction of the model and data graphs

The model graph

The easiest way of obtaining the model graph $G_M = (V_M, E_M)$ is by manually segmenting a face image selected specifically for this purpose. The attributes of the model graph are later computed regarding data extracted from that image. Figure 7.3 illustrates the way of generating the model graph from a selected image. Firstly, the facial landmarks are located by a tracking method [Feris and Cesar, 2001] and used to define a limited region. The model is manually obtained from a reference image. Figure 7.3c shows the face image that has been manually segmented to define the face model, and that has been used in the preliminary

experiments reported in this section. A more robust approach is to segment a set of face images and to calculate the graph attributes from different images. This approach allows the inclusion of vertices corresponding to specific facial features that may not be present in the selected model image, such as teeth or beard. The associated vertex and edge attributes are calculated only from the images where they are effectively present. Complete graphs are used, and even edges from and to the same vertex are considered. As a result, if $|V_M| = n$ then $\forall a_M^i \in V_M$ there are n edges in E_M so that $(a_M^i, a_M^j) \in E_M$ with $j = 1, 2, \ldots, n$.

The model graph is generated so that it contains vertices associated to each facial feature of interest, e.g. for each eyebrow, iris, nostril, mouth and the skin. The proposed approach allows additionally inclusion of other facial features non always included such as glasses, teeth and beard.

It is important to note that in the model of Figure 7.3c, some single facial features have been subdivided deliberately. An example of this are the eyebrows, which are subdivided in three parts each. This has been done because the adopted vertex and edge attributes are calculated based on average measures over the segmented image regions. Therefore, model attributes extracted from large regions tend to be less representative because such regions often present larger variability with respect to the attributes. That is why some facial features have been subdivided in order to circumvent such a potential problem. In addition, the fact that the skin is not a well-localized facial feature (in contrary to pupils and nostrils) presents an additional difficulty for introducing structural relations between skin and the other features. As an example, while it is possible to define structural relations such as *the pupil is above the nostrils*, it would be more difficult to define a similar relation with respect to the skin. As a solution to this problem, alternative types of structural relations such as *surrounded by* are avoided. Instead, we have adopted the approach of further dividing the skin into sub-segments as shown in Figure 7.3c, which allows us to use the same relational attributes between all facial features. An additional reason for over-segmenting the input image is that –as well as in the previous example of the human brain– this procedure will ensure that each of the segments will correspond to only one model region.

It is worth emphasizing that subdividing the model regions implies increasing the number of graph vertices and edges and thus it increases the complexity of the whole problem. Therefore, there is a trade-off between quality of model attributes and computation time that should be carefully considered when designing the graph model.

Generation of the data graph

The initial step to segment the facial feature regions is to locate the face in the image, which can be done both in photograph images and in video sequences. The latter requires detection of the face in a frame and to track it in the subsequent frames. In approaches such as [Cesar and Bloch, 2001, Cesar et al., 2002a], these location and tracking steps are performed using the Gabor Wavelet Network (GWN) [Kruger and Sommer, 1999]. The GWN acts as a rigid model that provides *approximative landmarks* which are located near the facial features to be segmented [Feris and Cesar, 2001] (e.g. eyes, nose and mouth). These landmarks are used in two different ways in order to make our approach more efficient: firstly, only certain regions around the landmarks are considered, and secondly, the landmark information is used by the optimization algorithm to constrain the solution search space. The reader is referred to [Feris and Cesar, 2001] for further details on the GWN approach. We will call *super-regions* to the 4 landmarks that we consider on our particular example, which are centered each on the left pupil, right pupil, the nose, and the mouth.

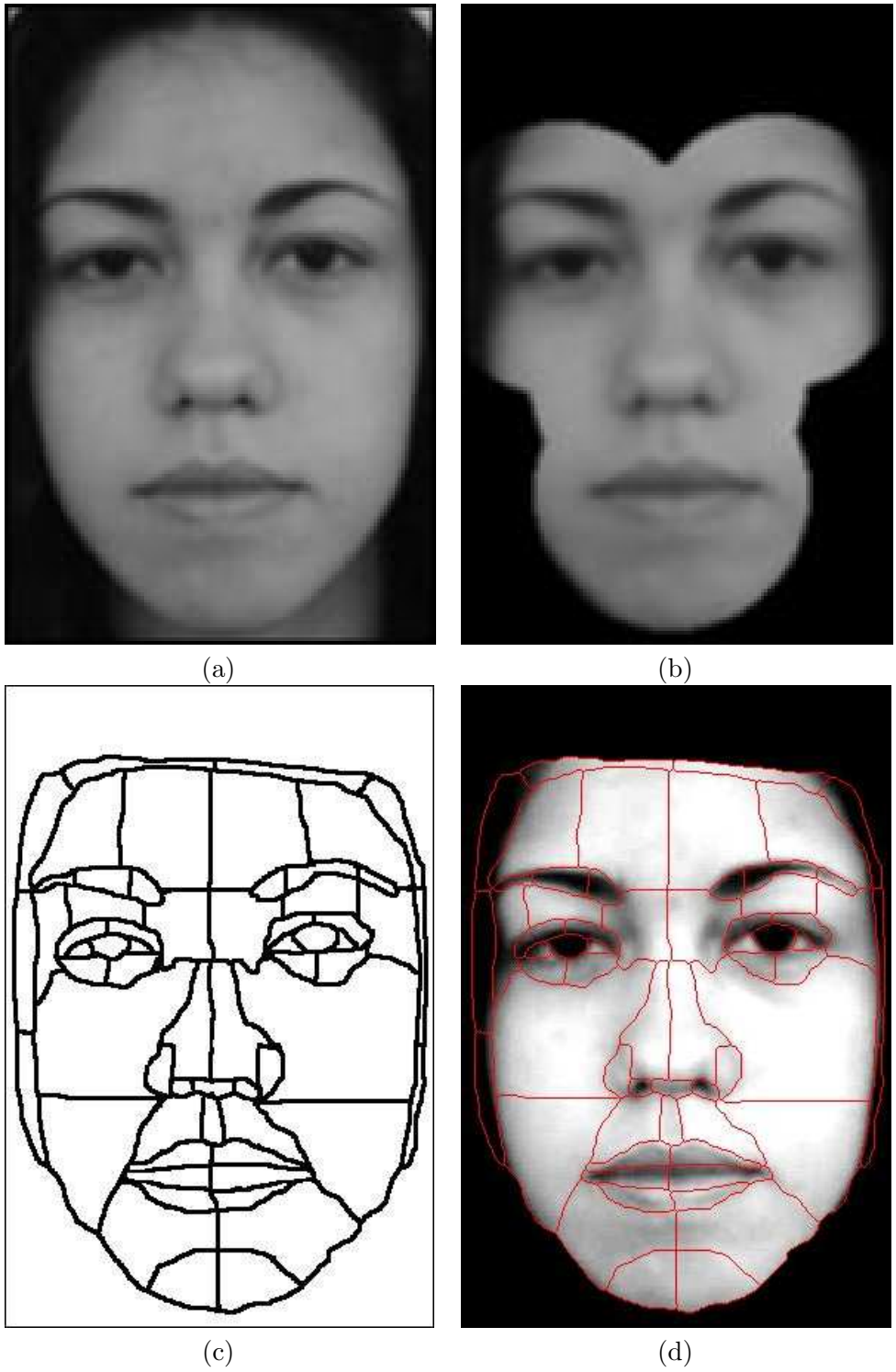


Figure 7.3: Illustration of the process to generate the model graph: (a) an original image is selected to extract the model from; (b) a masked version which contains only the regions of interest around the landmarks is obtained; (c) the face model is obtained by manual segmentation; (d) image where the model is superimposed to the face image (just for explanatory purposes).

Figure 7.4: Example of an over-segmented image after applying the watershed algorithm.

The input image is over-segmented using a watershed algorithm based on the approximative landmarks of the facial feature regions (the four super-regions in our case), which results in an over-segmented image such as the one shown in Figure 7.4. The over-segmentation is performed since it is important that the edges that define the facial regions to be segmented are also present after the segmentation process –similarly as in the brain recognition problem. The resulting over-segmented image is the one that is used next to generate the attributed data graph G_D, which is later matched against a model graph through a graph matching algorithm. As with the model graph, each vertex of G_D is adjacent to all other G_D vertices, and it is also adjacent to itself. Under these conditions no bijective correspondence can be expected because of the over-segmented nature of the image regarding the model, and as a result this problem calls for inexact graph matching.

Vertex and edge attributes and similarities

The knowledge about face features is represented in the attributed graphs by the unary (vertex) and binary (edge) attributes of both the model and data graphs. Unary attributes are calculated from the segmented region of the corresponding vertex, while relational attributes are based on the spatial disposition of the regions. We have based our decision on which type of attributes to use regarding the work described on this field such as [Cesar and Bloch, 2001, Cesar et al., 2002a,b], where a detailed description of vertex and edge attributes for this particular graph matching problem can be found. Following these references, the attributes and their similarities where selected as follows:

Unary attributes: Let $G = (V, E)$ be the graph (either the model or data graph), then $\eta(a) = (g(a), w(a), g_{min}(a), l(a))$ with $a \in V$ is the unary attribute of the region cor-

responding to vertex a in the segmented image, where $g(a)$ denotes the average grey level, $w(a)$ is computed as a texture index from wavelet coefficients, $g_{min}(a)$ denotes the average grey-level of the 15% darkest pixels of the image region associated to vertex a, and $l(a)$ is a super-region number assigned by the tracking procedure regarding approximative landmarks as described in the previous section. Both $g(a)$ and $g_{min}(a)$ are normalized between 0 and 1 with respect to the minimum and maximum possible grey-levels, and the value of 15% used to calculate $g_{min}(a)$ is a parameter that may be changed depending on the nature of the input image.

Regarding the definition of the vertex similarity, the vertex attributes $g_{min}(a)$ and $l(a)$ are included for supporting the recognition process, but only the vertex attributes $g(a)$ and $w(a)$ are considered in the computation of the similarity between vertices: $g_{min}(a)$ is included in order to facilitate the recognition of textured regions composed of light and dark pixels (e.g. the eyebrows and in some cases the mouth), and the super-region label $l(a)$ provided by the tracking procedure is only taken into account to restrict the search space. As a result, the similarity between any two vertices $a_D \in V_D$ and $a_M \in V_M$, denoted by $c_N(a_D, a_M)$, is defined as:

$$c_N(a_D, a_M) = 1 - (\ \beta\ |g(a_D) - g(a_M)|\ +\ (1-\beta)|w(a_D) - w(a_M)|\ )$$

where $0 \leq \beta \leq 1$ is a parameter for tuning the relative importance between the vertex attributes, and $c_N(a_D, a_M)$ is normalized to $[0, 1]$ with a higher value representing a higher similarity between the two vertices.

Binary attributes: Let $a^i, a^j \in V$ be any two vertices of G. The relational attribute $\nu(a^i, a^j)$ of the edge $(a^i, a^j) \in E$ is defined as the vector $\overrightarrow{v}_{a^i,a^j} = \frac{\overrightarrow{p_{a^i} p_{a^j}}}{2d_{max}}$, where d_{max} is the largest distance between any two points of the masked face regions, and p_{a^i} and p_{a^j} are the centers of gravity of the respective image regions. Following this particular definition of binary attributes, we have clearly that, $\nu(a^i, a^j) = -\nu(a^j, a^i)$, and as a result edges are directed. Note also that we allow $a^i = a^j$, and therefore the edge $(a^i, a^i) \in E$ is also considered, as well as its attribute $\nu(a^i, a^i)$. The latter are considered as a reference for comparing edges of regions $a1_D, a2_D \in V_D$ when the homomorphism h satisfies that $h(a1_D) = h(a2_D) = a1_M$ (in that case we would compare the edges $(a1_D, a2_D)$ and $(a1_M, a1_M)$). Note that this comparison is not considered in all the fitness functions proposed in Section 3.4.1.

Analogously as for vertex similarity, the similarity between any two edges $e_D = (a^i_D, a^j_D) \in E_D$ and $e_M = (a^k_M, a^l_M) \in E_M$ is defined as:

$$c_E(e_D, e_M) = 1 - \left| \overrightarrow{v}(a^i_D, a^j_D) - \overrightarrow{v}(a^k_M, a^l_M) \right|$$

where $\overrightarrow{v}(a, a')$ is the vector that goes from the center of gravity of region a to such of region a'. Again, we will only consider for a global similarity value those $c_E(e_D, e_M)$ that satisfy the condition $a^k_M = h(a^i_D), a^l_M = h(a^j_D) \in V_M$. This similarity is also normalized to $[0, 1]$.

7.3.3 Description of two face feature recognition problems and the graph matching approach

This section illustrates the facial feature recognition problem using a first reduced example (only taking into account the segments of an eye) and a second one representing a whole

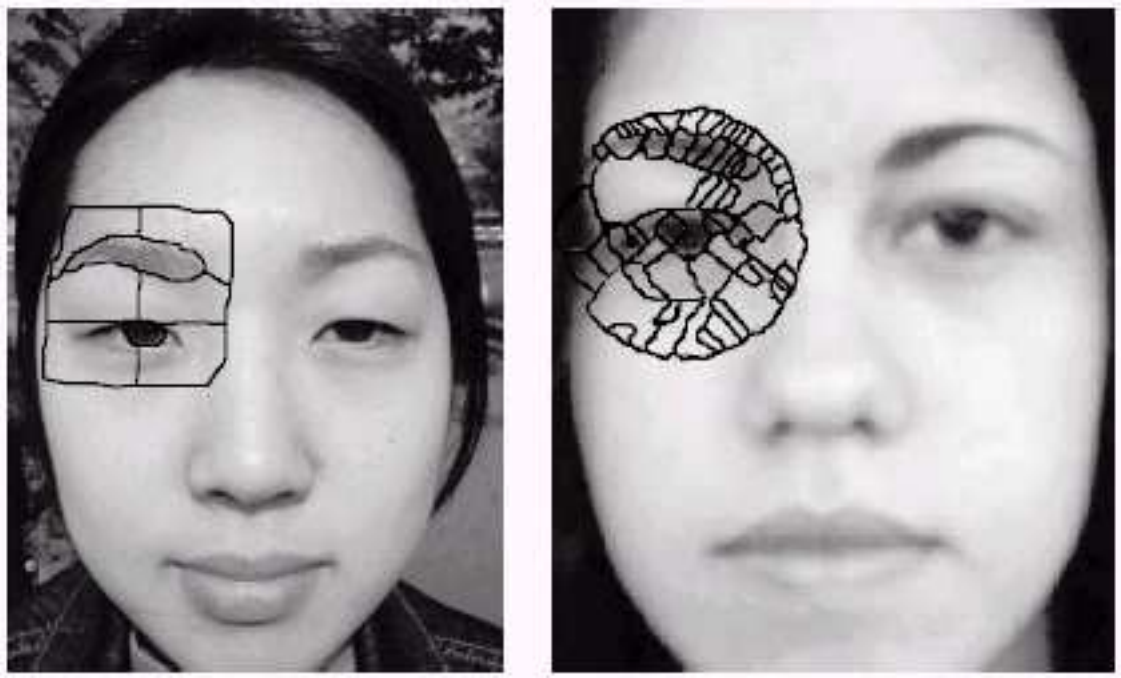


Figure 7.5: Illustration of the first (reduced) example of the facial feature recognition problem. These images show the small parts selected from the images to create the model graph G_M (left) and the data graph (right).

human face. In these examples, both the model and data graphs are attributed, and the recognition will rely on similarity functions between attributes. We use in both examples the model and data graph generation techniques described in the previous section.

The best homomorphism is found by optimizing a fitness function based on object and relational attributes defined on the graphs. Discrete EDAs and GAs are applied in both examples.

Problem 1: a reduced example for only an eye

This problem is proposed as a combinatorial optimization one. The fitness function that we have selected for our experiments is the $f_1(h)$ one described in Section 3.4.2, with $\alpha = 0.4$ and $\beta = 0.7$. On the other hand, only discrete EDAs will be applied and the individual representations used here is the one described in Section 3.3.1. This problem does not contain any specific constraints such as the ones required in the human brain recognition problem, although in the final solution no dummy vertices are allowed and only a matching is possible for each vertex in the data graph. Figure 7.5 illustrates the whole problem by showing the model and data images. In this example, the model contains 13 regions and the data image 75 ($|V_M| = 13$ and $|V_D| = 75$).

Problem 2: the whole face example

This second example analyzes other aspects in whole face examples. The model used for this case is again the one shown in Figure 7.3, which has been already applied in the literature [Cesar et al., 2002a,b] and was obtained using a standard digital camera. Input face

Algorithm	$\min_h(f_1(h))$	# of errors
UMDA	0.9455	5 (6.6%)
MIMIC	0.9455	5 (6.6%)
EBNA_{BIC}	0.9455	5 (6.6%)
EBNA_{K2}	0.9455	5 (6.6%)
eGA	0.9094	54 (71.0%)
ssGA	0.9454	7 (9.3%)

Table 7.5: Summary of the results for the small facial feature recognition problem.

images to which recognition is to be performed have been obtained using the same camera, but some images have also been downloaded from standard public databases available on-line[4]. The input or target images have been selected in order to show the robustness of the method regarding images acquired in different conditions (i.e. geometry, illumination, distance from the camera, etc.).

Four different face images were analyzed, and Table 7.6 shows for each of them the number of segments and edges after the automatic over-segmentation procedure. The model used is shown in Figure 7.3 and it contains 62 vertices and 3844 edges.

7.3.4 Experimental results

Problem 1

The results obtained with the reduced example of the eye regions for all the algorithms are summarized in Table 7.5. The number of incorrectly labelled regions is computed based on an optimum solution drawn by labelling a manually segmented data image. All EDAs provide the same solution. Although their computation time is higher, they lead to a better optimization compared to GAs. The 5 errors that are present in the returned solution are also included in the solutions provided by the other algorithms. The ssGA and specially the eGA approaches performed somewhat worse.

Figure 7.6 illustrates the same result obtained for all the EDAs. The most important features to be identified in the surrounding region of only one eye are the pupil, and the eyebrow. This figure shows these relevant parts following the final graph matching solution obtained with EDAs. The segments encircled in white are the ones matched satisfactorily as pupil or eyebrow. The ones encircled in red correspond to the 5 matching errors. Among these 5 errors, 3 correspond to very tiny regions, for which even the manual result given as the optimal cannot be considered reliable. Another error corresponds to a region in the hair, and exhibits one of the limits of the proposed approach, in cases where the data image contains objects that are not represented in the model. An extension of this approach could be to allow for no recognition in such cases (i.e allowing the use of dummy vertices). Finally, the last error corresponds to a region at one extremity of the eyebrow, which is labelled as skin. The reason is that the contour between the eyebrow and the skin was not detected by the watershed algorithm and therefore this region contains also some skin.

The skin is divided in the model deliberately in several regions in order to avoid geo-

[4]Examples of face images can be found for instance on the web site at the University of Bern, available at `http://iamwww.unibe.ch/~fkiwww/staff/achermann.html` and from the University of Stirling, available at `http://pics.psych.stir.ac.uk/`.

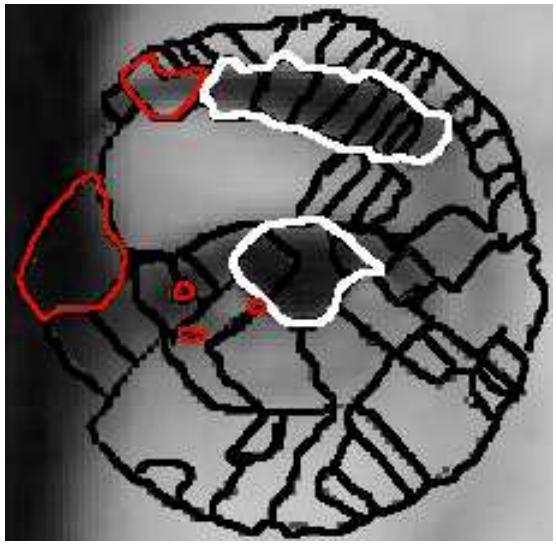


Figure 7.6: Illustration of the solution obtained with the different EDAs for the first (reduced) example of the facial feature recognition problem. The regions encircled in white correspond to the segments matched as pupil and eyebrow satisfactorily, while the ones in red represent the errors in the solution.

	deise	f014	f041	m036
vertices	112	176	183	228
edges	12544	30976	33489	51984

Table 7.6: Figures of the 4 cases that are analyzed in the second example, illustrating the number of vertices and edges that are considered. These values are illustrated for showing the difference in complexity for each of the examples.

metrical properties such as *surrounded by*. These skin section appear in the final solution to be correctly recognized. This satisfactory result illustrates the influence of edge attributes when vertex attributes are too similar for differentiation of regions.

Problem 2

We present firstly some results obtained applying only the $f_1(h)$ fitness function described in Section 3.4.2. These results are shown in Figures 7.7 to 7.10. Each of these figures shows the obtained segmentation and recognition of the eyebrows, nostrils and lips using the following search algorithms: (a) UMDA, (b) MIMIC, (c) EBNA$_{BIC}$, and (d) ssGA.

The results illustrated in Figures 7.7 to 7.10 are discussed in detail next. We executed 5 times each of the algorithms for each of the examples. Table 7.7 shows the results in the form of the mean fitness value of the best individual at the last generation, the number of different individuals created during the search, and the CPU time. The latter computation time is presented as a measure to note the difference in computation complexity of all the algorithms. The machine in which all the executions were performed is a two processor Ultra 80 Sun computer under Solaris version 7 with 1 Gb of RAM.

The null hypothesis of the same distribution densities was tested (non-parametric tests of Kruskal-Wallis and Mann-Whitney) for each of the examples and algorithm executions with

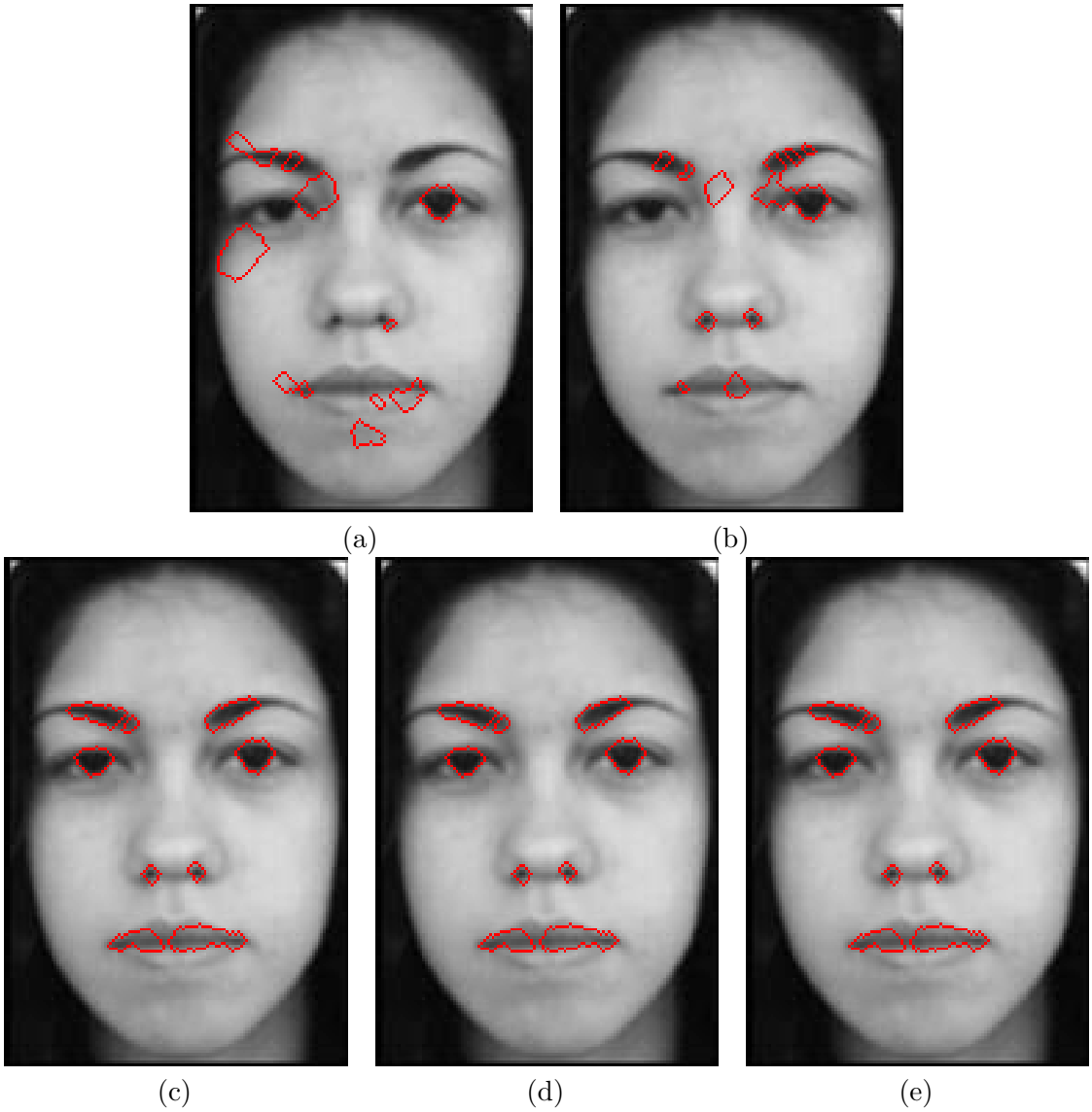


Figure 7.7: Segmentation and recognition of facial features for the *deise* example using (a) eGA, (b) ssGA, (c) UMDA, (d) MIMIC and (e) EBNA$_{BIC}$. As the model has been extracted from this image, the target and the model image are the same for this case.

the statistical package S.P.S.S. release 10.1. The results of these tests are shown in Table 7.8, and they confirm the significance of the differences of all the algorithms regarding the value of the best solution obtained of EDAs and GAs. They also show that differences between the different EDAs in the best individual obtained are not statistically significant, but these are significant among eGA and ssGA. In all the examples the EDAs obtained better results than the GAs, and these differences appear to be statistically significant regarding Table 7.8. Also, the differences in execution time appear to be significant in all the algorithms, and EDAs required in all cases more time. As a result, regarding Tables 7.7 and 7.8, we can conclude that the results are much better in EDAs at the expense of a higher computation time, but EDAs arrive to a more satisfactory final individual by having to evaluate less

Figure 7.8: Segmentation and recognition of facial features for the example *f014* using (a) eGA, (b) ssGA, (c) UMDA, (d) MIMIC and (e) EBNA$_{BIC}$.

individuals than GAs. This fact is important to take into account if the computation of the fitness function is more complex (i.e. if it requires more CPU time) than the one selected for our experiments.

Table 7.9 shows the errors obtained for some images in the form of a ground-truth-based assessment (total number of misclassified regions and percentage with respect to the total number of regions in the segmented image) for the different optimization algorithms. As it can also be seen, the ssGA and eGA algorithms lead to much poorer results with respect to the recognized facial features and the best solution obtained. These results could be partially understood by the fact that these two GAs are both general purpose algorithms. The use of GAs specially thought for our problem could lead to better results[5]. However, it must also be said that the EDAs applied are also general purpose ones.

In the light of the results obtained for the fitness values, we can sum them all up as follows: generally speaking, EDAs obtained in all the executions a fitter individual than GAs, but

[5]After the many years that genetic algorithms are part of the literature, very different versions and adaptations have been proposed. Therefore, other versions of GAs or even a different choice of crossover and mutation operators could lead to a better behavior.

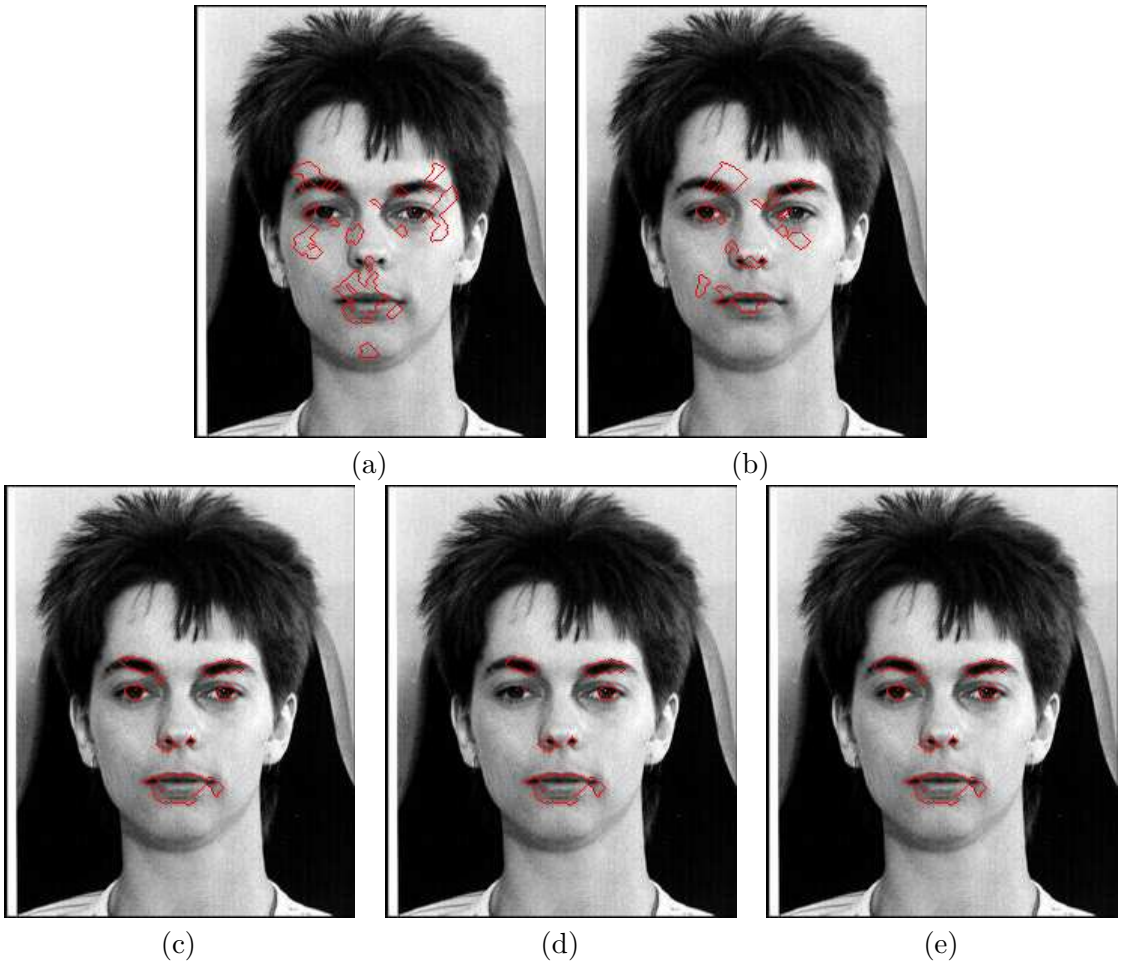


Figure 7.9: Segmentation and recognition of facial features for example *f041* using (a) eGA, (b) ssGA, (b) UMDA, (c) MIMIC and (d) EBNA$_{BIC}$.

although the number of individuals created is lower, the CPU time required was bigger. Besides the important problem of assessing the optimization algorithms with respect to the criterion function and execution time, it is also important to analyze the obtained results with respect to the problem context, i.e. recognition of the facial features of interest. In order to perform this task, we have generated a ground-truth for some faces by manually labelling the over-segmented images, i.e. by obtaining a human solution for the problem. Of course, such ground-truth is prone to some subjectivity, since a human operator decides the label of each region of the over-segmented image with respect to the model (nearby regions around the facial features are generally difficult to be classified, being often labelled differently depending on the operator). Then, the ground-truth is compared to each automatically labelled image to obtain the number of errors, i.e. number of misclassified regions (with respect to the ground-truth). Some of the error regions are shown in Figure 7.11. This figure shows 3 types of errors found in our experiments:

- Very small regions nearby the facial features (indicated by A in Figure 7.11), which have generally been missed by the operator while producing the ground-truth.

Figure 7.10: Segmentation and recognition of facial features for the example *m036* using (a) ssGA, (b) UMDA, (c) MIMIC and (d) EBNA$_{BIC}$.

- Regions in the outer portion of the eyebrows (indicated by B in Figure 7.11), which have been left out during the classification procedure because of the above explained reasons.
- True errors such as matching to a wrong region (indicated by C in Figure 7.11).

As it can be seen, the proposed method is able to correctly recognize the facial features of interest, being robust to substantial differences between the model (Figure 7.3d) and the target image. A problem that has been identified in our experiments is that the outer portions of the eyebrows in the model contain several skin pixels, leading to misclassifications near it. Therefore, only the two inner portions that compose each eyebrow in the model have been identified as *eyebrows* by the graph matching algorithms. Because of the structural constraints in the fitness function $f_1(h)$, the outer portions of the eyebrows in the obtained results have not been included, which is a drawback that we intend to circumvent in future work.

It is interesting to note that the facial features that have been recognized could be used for matching the model over the input face, being suitable for updating the model parameters with respect to the target image. The latter would be useful for instance to apply this technique for face and facial feature tracking in video sequences, thus avoiding the need for

	deise			f014		
	best	Time	eval.	best	time	eval.
UMDA	0.6896	00:13:54	143909	0.6827	00:58:26	184894
MIMIC	0.6894	00:16:53	129008	0.6827	01:07:46	167117
$EBNA_{BIC}$	0.6898	01:19:20	153924	0.6830	04:40:16	185908
eGA	0.5656	00:33:00	202000	0.5357	01:30:35	202000
ssGA	0.6429	00:25:54	202000	0.6120	01:07:50	202000
	f041			m036		
	best	time	eval.	best	time	eval.
UMDA	0.6813	00:58:09	196702	0.6794	01:41:22	201900
MIMIC	0.6813	01:09:51	181910	0.6790	02:08:22	201900
$EBNA_{BIC}$	0.6813	05:15:19	1951	0.6794	09:11:07	201900
eGA	0.5374	01:35:02	202000	–	–	–
ssGA	0.6107	01:08:58	202000	0.598	01:53:35	202000

Table 7.7: Figures of the 4 cases that we analyzed, illustrating the mean values after 5 executions of each of the algorithms. The *best* column corresponds to the mean best fitness value obtained through the search, and the differences between the algorithms are evident as EDAs obtain the best results for all the examples. The *time* column shows the CPU time required for the search, and the *eval.* one shows the number of individuals that had to be evaluated in order to end the search.

		GAs-EDAs	among GAs	among EDAs
deise	best	$p < 0.001$	$p = 0.008$	$p = 0.078$
	eval.	$p < 0.001$	$p = 1.000$	$p = 0.068$
	time	$p = 0.121$	$p = 0.008$	$p < 0.001$
f014	best	$p < 0.001$	$p = 0.008$	$p = 0.105$
	eval.	$p < 0.001$	$p = 1.000$	$p = 0.064$
	time	$p = 0.495$	$p = 0.008$	$p = 0.002$
f041	best	$p < 0.001$	$p = 0.008$	$p = 0.811$
	eval.	$p < 0.001$	$p = 1.000$	$p = 0.012$
	time	$p = 0.643$	$p = 0.008$	$p < 0.002$
m036	best	$p < 0.001$	–	$p = 0.085$
	eval.	$p < 0.001$	–	$p = 1.000$
	time	$p = 0.306$	–	$p = 0.002$

Table 7.8: Statistical significance for all the 4 examples and algorithms after 5 executions of each of the algorithms, by means of the results of the non-parametric tests of Kruskal-Wallis and Mann-Whitney. The first column shows the result of the test comparing all EDAs with all GAs, the second is the test for comparing eGA and ssGA, and the third shows the comparison between the three EDAs.

the GWN detection and tracking every sequence [Cesar et al., 2002b, Costa and Cesar, 2001].

7.4 Conclusions of the experiments on real problems

Different approaches for brain image recognition and for facial feature segmentation based on graph homomorphisms have been proposed. In this context, we have defined graph-based model representations of a human brain and a face model respectively, as well as the proce-

	deise		f014		f041		m036	
	Errors	%	Errors	%	Errors	%	Errors	%
UMDA	1	0.89	12	6.82	7	3.83	12	5.26
MIMIC	1	0.89	12	6.82	8	4.37	15	6.58
$EBNA_{BIC}$	1	0.89	11	6.25	5	2.73	15	6.58
eGA	21	18.75	36	20.45	46	25.14	50	21.93
ssGA	35	31.25	63	35.80	57	31.15	-	-

Table 7.9: Table showing the number of misclassified regions in each test image for each algorithm. The *Errors* column indicates the number of misclassified regions, while the column *%* shows the percentage with respect to the total number of regions in the image.

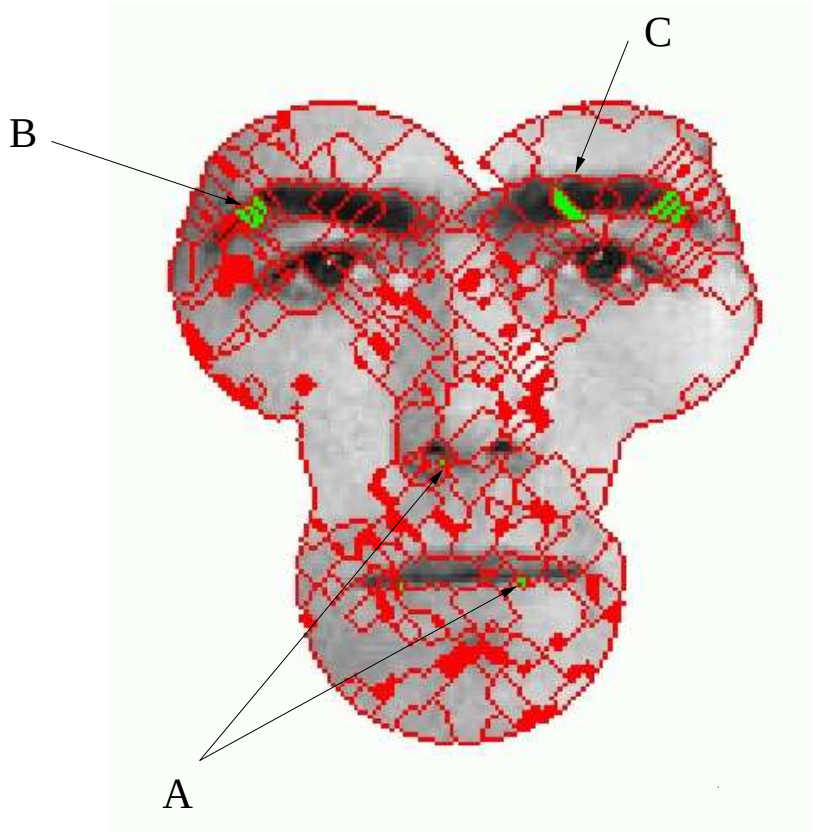


Figure 7.11: Example of some typical error regions. A: two very small regions nearby facial features, which are too small to be properly identified. B: regions in the outer portion of the eyebrows that could be ambiguous to classify as eyebrow or skin, in this case recognized as skin. C: true matching error, in this case eyebrow region recognized as part of the pupil.

dures for automatic segmentation of input images for these real problems. Fitness functions and individual representations for these problems have been tested with GAs and EDAs. The image processing techniques for the facial feature recognition problem are more sophisticated, since the facial features are segmented taking advantage of tracking information provided by a GWN technique. Our ongoing work aims at improving the method robustness and at generalizing it in a number of different ways, e.g. using fuzzy morphisms [Perchant

and Bloch, 2002], developing other object and relational attributes, and taking advantage of the homomorphism found in the previous frame in a video sequence when searching for the one in the current frame. These image processing techniques (except the temporal aspect) remain also to be tested for the human brain recognition problem.

Furthermore, the above definition of graph homomorphism implies that all vertices in G_D are mapped to G_M and if the input face presents features not known by the model, they will be classified. In the case of the facial feature recognition problem, if the input face has glasses and the model graph does not include them, the glass regions will be classified as skin or some other facial feature. Two possible solutions to this problem can be designed. The first one is to leave some of the G_D vertices unmapped, thus relaxing the homomorphism approach. The second approach is to define a *dummy vertex* in the model graph, to which the unclassified input regions should be mapped. Both approaches present specific difficulties and are currently being considered in our research. Also, other fitness functions are currently under investigation. Finally, we aim at applying our graph-based face model to face and facial expression recognition problems. In particular, we would like to analyze if temporal changes in the relational attributes could be used for facial expression recognition, because such trajectories are important for perceiving changes in facial expressions.

Chapter 8

Conclusions and future work

'Our imagination is the only limit to what we can hope to have in the future.'

Charles F. Kettering

8.1 Conclusions

In this thesis, we addressed the problem of recognition of structures in images using graph representations and inexact graph matching. One of the main contributions of our work is to express this task as a combinatorial optimization problem with constraints, and to propose methods to solve it based on EDAs and their parallelization.

A discussion on different representations of individuals has been provided. In particular, we proposed representations in both the discrete and continuous domains. Some of the constraints imposed to the matching could be introduced directly in the representations.

Different types of fitness functions have been presented. Our contribution here is twofold. First an experimental comparison of their behavior has been performed, and second new fitness functions based on probability theory have been designed.

The main focus of our thesis was on the optimization itself. A new approach based on estimation of distribution algorithms was introduced for solving the graph matching problem. Its foundations rely on an evolutionary computation paradigm that applies learning and simulation of probabilistic graphical models (i.e. Bayesian networks in the discrete domain and Gaussian networks in the continuous one) as an important part of the search process. Our contribution in this part was to adapt these algorithms to the inexact graph matching problem with constraints, which to our knowledge have never been addressed before. In particular we proposed original solutions to take the constraints into account. This contribution can certainly be exploited in other combinatorial optimization problems with constraints, thereby enlarging the potential application field of EDAs.

Finally another contribution relies in the parallelization of EDAs. Up-to-date parallelization techniques have been applied to these algorithms, resulting in two different programs suitable for execution on multiprocessors with shared memory and cluster of workstations under windows or GNU-Linux systems. The use of shared memory libraries with threads –using *pthreads*– as well as high-level parallelization libraries based on message passing –such as MPI– have been analyzed in detail. The particular case of EBNA$_{BIC}$ has been detailed, and each of its steps has been analyzed in terms of parallelization and computation costs. A parallel version of this algorithm is proposed for the BIC metric. This contribution allows now to use EDAs to solve problems with higher complexity.

From an experimental point of view, our contribution lies in the comparison of the performance of EDAs in both discrete and continuous domains with other evolutionary computation techniques such as genetic algorithms and evolutionary strategies. These experiments were performed for the different types of individual representations, different types of fitness functions, and applied to synthetic and real graph matching problems. Results show that our approach obtains better results and that converge to a solution by having to evaluate less individuals than other more usual evolutionary computation methods such as genetic algorithms. These differences in the results have been proved to be statistically significant after applying non-parametric tests.

8.2 Future work

Many different adaptations, tests, and experiments have been left for the future due to lack of time (i.e. the experiments with real data are usually very time consuming, requiring even days to finish a single run). Future work concerns deeper analysis of particular mechanisms, new proposals to try different methods, or simply curiosity.

There are some ideas that I would have liked to try during the description and the development of the fitness functions in Chapter 3. This thesis has been mainly focused on the use of EDAs for graph matching, and most of the fitness functions used to find the best result where obtained from the literature of adapted from these, leaving the study of fitness functions outside the scope of the thesis. The following ideas could be tested:

1. It could be interesting to consider the regions in the model and data images with different importance, depending on their size or their specific meaning with respect to the recognition process. This mechanism would for instance aid to distinguish in very complex problems which are the regions that are essential to be found, the ones that sometimes appear, and the ones that rarely do.

2. The way the model is constructed could be also changed: instead of using one typical image (prototype), it could be based on different images, in order to provide some information on the variability among the different images, and introduce it in the attributes. Unfortunately, in the type of images that we have taken as real examples the construction of a model from each image is a tedious task and no further study in this direction could be performed.

Obviously, the use of other types of individual representations and fitness functions could be investigated since they have an important influence on the results obtained at the end. New approaches in this direction can be induced from techniques described in the literature such as [Bloch, 1999a,b, Rangarajan et al., 1999a, Sanfeliu and King-Sun, 1983].

The performances of all the fitness functions described in Section 3.4.3 have not been compared on a same problem. The main reason was that some fitness functions are very complex to compute and require a considerable execution time to evaluate each individual. Parallelization techniques have been applied to the learning step in EDAs, but not for the evaluation of individuals, and such a mechanism could help at reducing execution times. Nevertheless, we are already designing and running experiments to compare the performance of our newly proposed probability theory-based fitness function $f_4(h)$ and $f_5(h)$ to such of the fitness functions defined previously in this section. The preliminary results of these experiments do not seem to be satisfactory, and further study is still required in order to understand the behavior of these two fitness functions and improve it.

Concerning the results for both applications (brain and facial features), we can also expect to improve them by having richer graphs, with more attributes.

In the definition of the EDAs in Chapter 4, there are also many ideas that could be exploited to try to obtain a most effective convergence towards the best solution. An example of this is the use of a mechanism that could be understood as a learning depending on the fitness value of the individual: in the learning proposed for EDAs all the selected individuals are used for the learning equally regardless of their fitness value. This means that the fitness value is just considered for selecting the best individuals, but differences between the values among these individuals are not considered in the learning process. A similar idea to this is proposed in the Bit-Based Simulated Crossover algorithm (BSC) [Syswerda, 1993], but this idea could be extended to any EDA. One of the disadvantages that this new type of learning can have is that by accelerating the convergence the search is too focused to the main individuals, and therefore EDAs could lead to local maxima. However, this idea is still a possibility that could be analyzed in the future to check whether local maxima are avoided or not and how to improve it for specific problems such as inexact graph matching.

The initial population in all EDAs has been built using a uniform distribution. Other methods could be also tested, as sometimes a pre-processing step could be added so that the search can also start with some specific individuals. Also, other types of statistical initializations such as greedy probabilistic methods could help at directing the search from the beginning, leading to less evaluations.

Regarding the application of parallelism to EDAs, an extension for the near future is the use of more powerful multicomputers in order to improve the parallelization: the computers we used had at most only 2 processors, and therefore no more than 4 workers were created per computer so that all the workers do not compete for CPU use with the corresponding thrashing problem. An additional task to perform is the parallelization of other algorithms such as EGNA_{ee} and EMNA, which are also susceptible of being parallelized due to the high number of tasks that can be performed in parallel on different processors.

Appendix A

Example on how to use a permutation-based approach in EDAs

Section 3.3.1 explains the use of permutation-based individual representations for EDAs, and Figure 3.1 shows a procedure to translate a permutation-based representation to the matching solution it symbolizes. This appendix illustrates this procedure as well as some problems to be taken into account with an example.

A.1 Example of translating from a permutation to the solution it symbolizes

In order to demonstrate the representation of individuals containing permutations and the procedure for translating them to a point in the search space, we make use of the example shown in Figure A.1. In this example we are considering an inexact graph matching problem with a data graph G_D of 10 vertices ($|V_D| = 10$) and a model graph G_M of 6 vertices ($|V_M| = 6$). We also use a similarity measure for the example (the $\varpi(i, j)$ function), the results of which are shown in the same figure. This similarity function does not always have to be symmetrical, and just as an example of this we have chosen a non-symmetrical one (see Section 3.3.1 for a discussion on this topic). The translation has to produce individuals of the same size (10 variables, that is, 10 nodes in the Bayesian network), but each of their variables may contain a value between 1 and 6, that is, the number of the vertices of V_M with which any vertex of G_D can be matched in the solution.

Figure 3.1 shows the procedure for both phases 1 and 2. Following the procedure for phase 1, the first 6 vertices will be matched, and we will obtain the first matches for the three individuals in Figure A.1.

In the second phase, generation of the solution will be completed by processing one by one all the remaining variables of the individual. For that, we will consider the next variable that is still not treated, the 7^{th} in our example. Here, the first individual in the example has the value 7 in its 7^{th} variable (i.e. position), which means that vertex 7 of G_D will be matched next. Similarly, the vertices of G_D to be assigned to the 7^{th} position for the other two example individuals are vertices 10 and 4 respectively.

Next, in order to decide which vertex of G_M we have to assign to our vertex 7 of G_D

Individuals:

1	2	3	4	5	6	7	8	9	10

5	8	7	1	6	9	10	3	4	2

10	9	8	7	6	5	4	3	2	1

Similarity Function:

$\varpi(i,j)$	1	2	3	4	5	6	7	8	9	10
1	1.00	0.87	0.67	0.80	0.77	0.48	0.88	0.80	0.75	0.89
2	0.03	1.00	0.96	0.13	0.73	0.90	0.15	0.66	0.74	0.92
3	0.20	0.42	1.00	0.63	0.05	0.22	0.20	0.51	0.31	0.50
4	0.52	0.50	0.88	1.00	0.49	0.88	0.08	0.91	0.38	0.47
5	0.19	0.90	0.85	0.71	1.00	0.15	0.24	0.51	0.97	0.80
6	0.47	0.87	0.67	0.80	0.77	1.00	0.88	0.80	0.75	0.87
7	0.03	0.96	0.35	0.13	0.73	0.90	1.00	0.66	0.74	0.92
8	0.20	0.42	0.93	0.63	0.05	0.22	0.20	1.00	0.31	0.50
9	0.52	0.50	0.89	0.53	0.49	0.88	0.08	0.91	1.00	0.47
10	0.19	0.90	0.85	0.71	0.18	0.15	0.24	0.51	0.97	1.00

Figure A.1: Example of three permutation-based individuals and a similarity measure $\varpi(i,j)$ between vertices of the data graph ($\forall i,j \in V_D$) for a data graph with $|V_D| = 10$.

1	2	3	4	5	6	–	–	–	–
1	2	3	4	5	6	7	8	9	10

4	–	–	–	1	5	3	2	6	–
1	2	3	4	5	6	7	8	9	10

–	–	–	–	6	5	4	3	2	1
1	2	3	4	5	6	7	8	9	10

Figure A.2: Result of the generation of the individual after the completion of phase 1 for the example in Figure A.1 with G_D containing 6 vertices ($|V_M| = 6$).

according to the first individual, we analyze the similarity of vertex 7 of V_D and each of the previously matched vertices of G_D (vertex 1 to 6). This similarity measure is given by the function ϖ shown in Figure A.1. If we look at the 7^{th} line in this table we see that in columns 1 to 6, the highest value is 0.96, in column 2. Therefore, following the algorithm in phase 2, we match to vertex 7 of G_D the same vertex of G_M assigned to vertex 2 of G_D. As

we can see in Figure A.2, for the first individual, vertex 2 in G_D was matched to the vertex 2 of G_M, and therefore we will also assign vertex 2 of G_M to the 7^{th} vertex of G_D.

Similarly, for the second individual, the 7^{th} variable of the individual is also processed. This has the value 10, so vertex 10 of G_D is therefore the next to be matched. We will compare this vertex with the vertices matched previously, i.e. vertices 5, 8, 7, 1, 6 and 9. The highest similarity value for these is $\varpi = 0.97$, in column 9. Therefore the most similar vertex is 9, and vertex 10 of G_D will be matched to the same vertex of G_M as vertex 9 of G_D was. Looking at Figure A.2, this is 6^{th} vertex of G_M. Following the same process for the third individual, we obtain that vertex 4 of G_D is matched with vertex 3 of G_M. Figure A.3 shows the result of this first step of phase 2.

Continuing this procedure of phase 2 until the last variable, we obtain the solutions shown in Figure A.4.

1	2	3	4	5	6	2	–	–	–
1	2	3	4	5	6	7	8	9	10

4	–	–	–	1	5	3	2	6	6
1	2	3	4	5	6	7	8	9	10

–	–	–	3	6	5	4	3	2	1
1	2	3	4	5	6	7	8	9	10

Figure A.3: Generation of the solutions for the example individuals in Figure A.1 after the first step of phase 2 ($|V_M| = 6$).

1	2	3	4	5	6	2	3	3	3
1	2	3	4	5	6	7	8	9	10

4	2	2	2	1	5	3	2	6	6
1	2	3	4	5	6	7	8	9	10

1	3	3	3	6	5	4	3	2	1
1	2	3	4	5	6	7	8	9	10

Figure A.4: Result of the generation of the solutions after the completion of phase 2.

Note that each of the vertices of G_D is assigned to a variable between 1 and $|V_M| = 6$. Note also that every vertex of G_M is matched to at least one vertex of G_D, and that a value is given to every vertex of G_D, giving a matching value to each of the segments in the data image (all the segments in the data image are therefore recognized with a structure of the model).

A.2 The redundancy problem on permutation-based representations

An important aspect of this individual representation based on permutations is that the cardinality of the search space is $n!$ This cardinality is higher than that of the traditional individual representation, but it is tested for its use with EDAs in graph matching for the first time here. In addition, it is important to note that a permutation-based approach can create redundancy in the solutions, as two different permutations may correspond to the same solution. An example of this is shown in Figure A.5, where two individuals with different permutations are shown and the solution they represent is exactly the same.

Individual 1:

1	2	3	4	5	6	7	8	9	10

Individual 2:

1	2	3	4	5	6	7	9	8	10

Solution they represent:

1	2	3	4	5	6	2	3	3	3
1	2	3	4	5	6	7	8	9	10

Figure A.5: Example of redundancy in the permutation-based approach. The two individuals represent the same solution shown at the bottom of the figure.

Appendix B

Example of correcting individuals

This appendix gives an example of how an individual is generated using either the LTM and ATM methods as explained in Section 4.5.1. Just as an example, we will assume that we have a case where the model and data graphs contain respectively 8 and 10 vertices ($|V_M| = 8$ and $|V_D| = 10$). Therefore, the length of an individual will be of 10 variables.

Both LTM and ATM are methods that act directly on the simulation step of EDAs. Therefore, we need to perform first the learning step following the EDA that we have chosen. The learning step will return a Bayesian network (in this case with a size of 10 nodes) as well as the estimation of the distribution $p_l(\boldsymbol{x})$ obtained from the N selected individuals, being the latter in the form of the different conditional probabilities θ_{ijk} as defined in Equation 4.3.

We will assume that the learned Bayesian network structure is the one shown in Figure B.1. As explained before, the Bayesian network shows interdependencies between the variables (e.g. in this case this structure is showing that the value taken by variable 1 is dependent on the value of variables 2 and 9, that is, the matching assigned to vertex 1 of the data graph G_D is dependent on the matching assigned to vertices 2 and 9 of the same graph G_D, while the latter vertices can be matched to any vertex of G_M independently of the rest of matches). Following this Bayesian network, it is important to have a look at the different combination of values of the parent-variables: in the case of nodes 2, 5 and 9, they do not have parents, so they are considered as independent. Nodes 3, 4, 6, 7 and 10 have a single parent, and therefore the possible combination of values for the parents is $|V_M|$=8 (i.e. the number of values that the only parent can take). Finally, nodes 1 and 8 have two parents each, and therefore the number of possible combinations of values of the two parents is $|V_M|^2 = 64$. Having all this into account, the probabilities that we will have to compute are just the following: θ_{2-k}, θ_{9-k}, $\theta_{31k} \ldots \theta_{38k}$, $\theta_{11k} \ldots \theta_{1(64)k}$, $\theta_{41k} \ldots \theta_{48k}$, $\theta_{61k} \ldots \theta_{68k}$, $\theta_{71k} \ldots \theta_{78k}$, $\theta_{(10)1k} \ldots \theta_{(10)8k}$, θ_{5-k}, and $\theta_{81k} \ldots \theta_{8(64)k}$, where $k = 1 \ldots |V_M|$ in all the cases.

Just as an example for our purposes, we will assume that the value of these probabilities is uniform (this is not normally the case, we just do it for simplicity).

At this stage, we will start the simulation step in order to create the new R individuals of the next generation. Each individual $\boldsymbol{x}= (x_1, \ldots, x_{|V_D|})$ has to be generated by instantiating each of the variables one after another. For this we will use the PLS method in which an ancestral ordering $\boldsymbol{\pi}$ of the nodes in the Bayesian network is followed as explained in Section 4.2.2. An ancestral ordering is any ordering in which any variable is placed after all its parent variables on the Bayesian network. A possible ancestral ordering for the Bayesian network in Figure B.1 is $(2, 9, 3, 1, 6, 4, 7, 10, 5, 8)$, but others such as $(2, 9, 5, 3, 4, 6, 1, 7, 10, 8)$ or $(9, 1, 7, 10, 5, 8, 2, 3, 4, 6)$ could also be considered. Any of these could be used, but we will

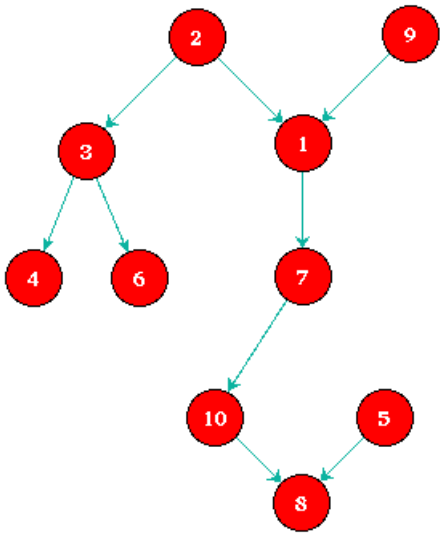


Figure B.1: Example of a Bayesian network structure.

select the first one: $\boldsymbol{\pi} = (2, 9, 3, 1, 6, 4, 7, 10, 5, 8)$.

Once the ancestral ordering has been found, we will start generating individuals. We will instantiate the variables or each individual following the ancestral ordering, $\boldsymbol{\pi}$. We will proceed similarly at the beginning either for LTM and ATM, where initially $VNO(V_M)^1 = \{1, 2, 3, 4, 5, 6, 7, 8\}$ and $vns^1 = |V_D| = 10$.

B.1 Simulation with LTM

Following $\boldsymbol{\pi}$, we will start with variable X_2 ($\pi(1) = 2$). Variable X_2 is independent from the rest, and the probability to take any of its possible values is the same ($\forall k = 1 \ldots |V_M|, \theta_{2-k} = \frac{1}{|V_M|}$). As the condition $|VNO(V_M)^1| = vns^1$ is not satisfied no modifications are to be done on the probabilities, so we will select a value at random and we will assign it to variable X_2. Let us imagine that this value is 1. Variable X_2 in the individual is set with this value, which in other words means that in the solution represented by this individual we are matching vertex 2 of G_D with vertex 1 of G_M.

The next variable to instantiate is X_9 ($\pi(2) = 9$) which is also independent from the rest, and its situation is the same ($\forall k = 1 \ldots |V_M|, \theta_{9-k} = \frac{1}{|V_M|}$). This time we have that $VNO(V_M)^2 = \{2, 3, 4, 5, 6, 7, 8\}$ and $vns^2 = 9$, and therefore $|VNO(V_M)^2| = vns^2$ is not satisfied. Therefore we will select a value at random. Let us again imagine that this value is 1, then we will assign the value 1 to variable X_9.

So far, after finishing this second step we have the following individual:

	1							1	

The following variable to work with is X_3 ($\pi(3) = 3$). Following the Bayesian network, this variable is dependent of variable X_2, which has already been instantiated. We said before that its probabilities are equal, and therefore we have that $\forall j, k = 1 \ldots |V_M|, \theta_{3jk} = \frac{1}{|V_M|}$. As $VNO(V_M)^3 = \{1, 3, 4, 5, 6, 7, 8, 10\}$ and $vns^3 = 8$, we have again that $|VNO(V_M)^3| \neq vns^3$, following its distribution we select a value at random. Let us now imagine that this value is 4, thus so we assign the value 4 to variable X_3.

In a similar way, we will now consider next variable X_1 ($\pi(4) = 1$), which is dependent on variables X_2 and X_9. As it depends in two parent-variables instead on in one as before, in this case $\forall k = 1 \ldots |V_M| \;\; j = 1 \ldots |V_M|^2, \theta_{12k} = \frac{1}{|V_M|}$ and $\theta_{1jk} = \frac{1}{|V_M|}$. We also have that $VNO(V_M)^4 = \{1, 2, 3, 5, 6, 7, 8\}$ and $vns^4 = 7$, $|VNO(V_M)^4| \neq vns^4$. Therefore following the distribution of this value, we select a value at random, and let us imagine that we obtain one more time the value 1. So far we have the following individual:

1	1	4						1	

Next, the variable X_4 is treated ($\pi(3) = 4$). Variable X_4 is dependent only on variable X_3, which has already been instantiated. In this case we have a single parent, and therefore $\forall j, k = 1 \ldots |V_M|, \theta_{6jk} = \frac{1}{|V_M|}$. If we were in an ordinary PLS simulation approach, a value would have been chosen at random. However, this time we have that $VNO(V_M)^5 = \{2, 3, 5, 6, 7, 8\}$ and $vns^5 = 6$, that is, the condition $|VNO(V_M)^5| = vns^5$ is satisfied. As a result, following the LTM approach, we have to modify the θ_{6jk} probabilities before instantiation so that values already appeared in previous steps do not appear again. We do this because the number of variables to instantiate equals the number of values still not appeared in the individual. Following Equation 4.42 we modify the probabilities as follows:

$$\theta_{6j1} = 0, \theta_{6j4} = 0, \theta_{6jk} = \frac{1}{|V_M| - 2} \quad \forall j = 1 \ldots |V_M|, \quad k = 2, 3, 5, 6, 7, 8, 9, 10 \tag{B.1}$$

In other words, we set the probabilities for the values already appeared to 0, avoiding them to appear for this variable, and we normalize the rest of probabilities. Doing it so, we make sure that the next value that will be instantiated will not be neither 1 nor 4.

As with this last case, in the successive variables to simulate, the condition $|VNO(V_M)^m| = vns^m \;\; m = 6, 7, 8, 9, 10$ will be satisfied, and therefore for each of these variable to instantiate a value not yet appeared will be assigned. Therefore LTM will ensure that all the vertices in G_M will have at least a vertex from G_D to which are matched.

Note that the procedure followed with LTM is basically the same as PLS until the condition $|VNO(V_M)^m| = vns^m$ is satisfied. If random values would be different, the latter condition was never satisfied, and the LTM procedure will behave as an ordinary PLS approach.

B.2 Simulation with ATM

ATM is somehow more complex than LTM in the sense that all the θ_{ijk} probabilities are manipulated even before the condition $|VNO(V_M)^m| = vns^m$ is satisfied.

In ATM the probabilities change in relation to a value $K = \left\lceil \frac{N - vns^m}{vns^m - |VNO(V_M)^m|} \right\rceil$. This will be used for adapting the probabilities when the condition $|VNO(V_M)^m| = vns^m$ is satisfied. The finality is to give more probability to values not appeared yet and to lower the rest.

Following our example, we will assume that the value for N is 1000 and that the ancestral ordering of choice is the same as in the LTM example. Following it, variable X_2 will be treated first ($\pi(1) = 2$). As the condition $|VNO(V_M)^1| = vns^1$ is not satisfied, following the ATM approach we compute the values K and P_{Indiv}:

$$K = \left\lceil \frac{N - vns^1}{vns^1 - |VNO(V_M)^1|} \right\rceil = \left\lceil \frac{1000 - 10}{10 - 8} \right\rceil = 495$$

$$P^1_{Indiv} = \sum_{k \mid u^k_M \in V_M \setminus VNO(V_M)^1} \theta_{2-k} = 0.$$

As we have at the beginning that all vertices of G_M are in $VNO(V_M)^1$, then all the probabilities will be changed following Equation 4.43 by multiplying then by the following factor:

$$\frac{K - P^m_{Indiv}}{K \cdot \left(1 - P^m_{Indiv}\right)} = \frac{495 - 0}{495 \cdot 1}.$$

This means that all the probabilities will not be changed. As before a value will be obtained for variable X_2 at random following its distribution ($\forall k = 1 \ldots |V_M|, \theta_{2-k} = \frac{1}{|V_M|}$). Let us imagine that this value is 1 as in the LTM example .

The second variable to treat is X_9 ($\pi(2) = 9$). We know that $|VNO(V_M)^2| = vns^2$ is not satisfied, so following the ATM approach we have that:

$$K = \left\lceil \frac{N - vns^2}{vns^2 - |VNO(V_M)^2|} \right\rceil = \left\lceil \frac{1000 - 9}{9 - 7} \right\rceil = 496$$

$$P^1_{Indiv} = \sum_{k \mid u^k_M \in V_M \setminus VNO(V_M)^1} \theta_{2-k} = \frac{1}{|V_M|} = \frac{1}{8} = 0.125.$$

Following Equation 4.43 we will now have a slight change on the probabilities:

$$\theta^*_{9-k} = \begin{cases} \theta_{9-k} \cdot \frac{496 - 0.125}{496 \cdot (1 - 0.125 = \theta_{9-k} \cdot 1.14)} & \text{if } k = 2,3,4,5,6,7,8 \\ \frac{\theta_{9-k}}{496} & \text{if } k = 1 \ . \end{cases}$$

This example shows that as $|VNO(V_M)^m|$ and vns^m are more similar the effect on the probabilities will be stronger. This change in the probabilities is applied to all the variables to instantiate until the condition $|VNO(V_M)^m| = vns^m$ is satisfied, and then ATM behaves like LTM. The effect on this method is that learned probabilities are manipulated even more than with LTM, and therefore values not yet appeared are more possible to appear.

Appendix C

On processes and threads: synchronization and communication in parallel programs

C.1 Sequential and parallel programs: main differences

In all our programs, there are some execution phases in which our processes are taking some time in an I/O operation or in a synchronization operation waiting for the system or another process to finish a task. During these phases, our programs are not progressing on their work, it is rather the operating system or some other process running on the computer the ones which are taking all the valuable CPU time. This is the reason why programs that abuse too much in using file read/write operations are so slow.

Processes that need to wait for an operating system service do usually leave the CPU free for other processes. This waiting is referred to as a process to be in the state of *blocked*. In the case of a sequential problem such as the EDA program, the whole algorithm is being executed as a single task, and the fact that our process is waiting one of these *blocking* operations makes the algorithm to be completely stuck during these periods of time. During the time spent waiting in the blocked state, another process takes ownership of the CPU and the execution of our algorithms does not progress at all.

Parallel programs provide a means for another process involved in the same calculation to make use of the CPU and continue with another part of the common job while our process is blocked (i.e. due to be waiting for an I/O or a synchronization operation). This happens both in single-processor and multiprocessor machines.

It is also important to note that when we write a sequential program (i.e. a non-parallel algorithm), our program will not be able under any circumstance of making use of more than a single CPU, even if we have more than one available. Only compilers that will convert automatically a sequential source program into a parallel one could help, but as explained before in the introduction these can reach these objective only in special cases. Therefore, if we want to use more than a CPU at the same time, we need to apply some kind of parallelization technique manually (writing specific instruction in the source code) when writing our program.

Parallel programs are based on dividing the job in smaller pieces so that more than

a single process work together[1] and collaborate between them to obtain the same result hopefully in less time. Doing it so, during the time that the I/O device takes to perform its task, the parallel program is able to make progress with another part of the job concurrently. Moreover, the fact of having a parallel program also allows us to make use of up to all the available CPUs at the same time.

Unfortunately, dividing the job in different processes that could be executed independently requires to rewrite some parts of the sequential program, or even to design a parallel version of it from scratch. This is not always an easy task, as several aspects need to be taken into account: the selection of the inter-process communication mechanism that will be used (which also consumes some extra execution time), the selection of the inter-process synchronization mechanism, and the identification of the parts of the sequential program that are candidate to be parallelized. The latter is important, as in most sequential programs there are always some parts that require to be done sequentially and cannot be executed in parallel. A simple example of the later is the moment when all the results of each of the working processes are to be gathered and a final result given. In many cases the preparation of the final result has to be done by a single process, and the rest of them cannot do anything during this period apart from waiting for more working instructions.

C.2 Processes and threads

In traditional operating systems, the concept of *process* is described as the instance of an executing program. In these, a process can only contain a running program where there is a single execution flow, that is, the program will be executing a single instruction at any time. This means in other words that in this traditional systems a process is represented as a single execution unit and that it only can be executed in a single CPU at any particular time.

In addition, the memory space assigned to a process in these traditional operating systems is defined to be *private*: no other process is allowed to access another process' memory space. In these systems no shared memory is available between processes, and therefore the only inter-process communication mechanism available for programmers is the use of message passing primitives –another possible mechanism is also the use of disk files, but this is very inefficient in terms of time performance. Only the operating system could make use of shared variables for communication between its internal parts.

In the recent years, some operating systems such as UNIX have included particular system calls[2] in order to allocate and to free shared memory from the user's address space in order to make possible for processes to share memory between them and to use shared variables (e.g. a shared buffer) as a inter-process communication mechanism. However, this mechanism is nowadays misused in UNIX in favor of more efficient and user-friendly mechanisms.

A next step forward in the operating systems history is the arrival of *multithreading*

[1] Term *process* is not the most appropriated in some cases, and *execution unit* should be used instead. However, the term *process* will be used in this thesis due to the fact that it is the one of choice in many parallelism books.

[2] *System calls* are standard programmable functions that any operating system provides as an interface for programmers to use system's resources and services. In the Windows family the term *Application Program Interface* or simply *API* is used instead. Any program running in any operating system can only access hardware by using these system calls, and all of them form the formal interface between the operating system and the executing programs. There are many types of system calls. Modern operating systems do also provide specific ones for inter-process synchronization.

operating systems. These systems included a new concept known as *thread*[3] that allowed a program to have more than an internal function running at the same time within the same memory space of a single process. As a result, a process could have more than a execution flow, and therefore more than a single line of the same program could be executing in parallel in two different CPUs simultaneously. Multithreading operating systems provide dedicated system calls for creating and managing execution threads as well as for creating and managing processes. In this systems, the memory address space of every process is still private for the rest of the processes, and as execution threads are attached to a unique process, they can use its whole memory space. In a multithread operating system two threads attached to a same process can use global shared variables within the process's memory space for communication. However, two threads attached to different process could not share any memory space and message passing is the only communication mechanism for them[4].

Nowadays, all commercial and general purpose operating systems are multithreading ones. When we write a sequential (i.e. non-parallel) program, the compiler will create an executable file that will execute in a traditional way, with a single thread or execution unit per process. However, we can use the specialized system calls that our operating systems provide, at a lower level than the programming language, and compile a program that will create more than a thread per process. Unfortunately, there is no a standard for system call interfaces applied to thread management that can be used in all the operating systems. Even within the UNIX family there are many different and incompatible interfaces available. In the recent years the POSIX standard's *pthread* interface library appears to be the most broadly used, and even an implementation of this library for the windows operating system family can be found on the Internet.

Whichever the selected parallel programming approach, either by making use of threads or processes, in both cases the communication and synchronization elements available are the same. The only restriction is that in the case of the processes there cannot be any communication mechanism based on shared memory, due to the privacy of the memory address spaces of them[5]. However, some books try to avoid distinction between processes and threads when speaking about parallelism, and the term *task* is used. In our case, we will use the term *process* to refer to both processes and threads, as this is done commonly in the literature.

C.3 Communication and synchronization between processes or threads

In any operating system, processes compete for accessing shared resources or they co-operate within a single application to communicate information to each other. Both situations are managed by the operating system by using synchronization mechanisms that allow exclusive

[3]Threads are also known as *light weight processes* or *subprocesses*. These are in fact execution flows or execution units within the same memory address space of a process. Operating systems that allow more than a thread to be running at the same time within a process (i.e. more than an instruction within the same process is executing at the same time) are called *multithreading operating systems*. Threads share all the global variables and functions within their process, and therefore they can use shared memory and variables for communication. However, synchronization mechanisms are required to avoid race conditions.

[4]Hard disk files could also be used for inter-process communication or inter-thread communication, but this mechanism is very inefficient. In parallel systems this communication solution is avoided, specially for the case of threads.

[5]Unless you use specific mechanisms that are only available in some operating systems, such as in UNIX.

access to shared resources and communication elements in a coordinated way.

In multithreading operating systems, the address space within any single process is shared between its threads, and therefore communication is performed by means of data structures within this shared memory space. This mechanism is quite efficient in terms of execution time, but on the other hand explicit communication mechanisms are required to ensure exclusive access to these shared structures. If shared buffers or queues are defined, and if exclusive access to them is provided, elaborated communication schemes such as the producer-consumer can be used. The client-server scheme can be regarded as a particular case of the producer-consumer where clients *produce* requests and servers *consume* them. When tasks (either processes or threads) communicate using a client-server approach, they do not share the address space. In this case, message passing primitives that provide implicit synchronization are required, and as a result they simplify the programming of the communication between processes.

In general terms, in parallel programming the term *communication* refers to the information exchange between processes. The communication between processes requires some kind of *synchronization*, either implicit or explicit, in the form of a norm or a protocol, that makes possible the information exchange between the communicating processes.

This section focuses on the different communication and synchronization mechanisms available for processes at a user level, as well as in the problems that programmers have to face when programming parallel applications.

C.3.1 The model of race conditions and critical sections

Let us consider two processes P_i and P_j that form part of a parallel program where these produce results for other processes not shown in this example. P_i and P_j store their results in a shared buffer buf. In order to find out which is the next free position in buf to store a result, there is also a shared pointer *inp*. If a process wants to store a result in buf, it has to call the following function:

```
void StoreResult(char item) {
    int p;
    ...
    p = inp;
    buf[p]= item;
    inp++;
    ...
}
```

This function seems to be correct, and no apparent problems can be noticed at a first glance. However, let us imagine the case in which the parallel execution progresses as follows:

```
          Pi                          Pj
    (1) p = inp;
                              (2) p = inp;
                              (3) buf[p]= item;
    (4) buf[p]= item;
    (5) inp++;
                              (6) inp++;
```

As we can see, this particular execution order will lead to storing the second value *item* at the same position of the buffer as the first one. P_j will overwrite its value over P_i's one. This problem is known as the *race condition*. Any program accessing shared resources can have this problem, however it may not be present in all the runs, as it depends on the order in which different processes execute their instructions. Any part of a program that can create race conditions is called a *critical section*.

Critical sections are present in general terms whenever a resource (i.e. a buffer or, simply, a variable) is shared among many concurrent processes. Programmers can use the operating system primitives to avoid race conditions. For this, it is important firstly that critical sections are identified in our programs. The programmer could afterwards use the appropriated system calls for communication or synchronization that will provide exclusive access to the critical section. Doing it so, the programmer can be sure that at most one single process can be executing a critical section at any time. In addition, the programmer can also use one of the many specialized libraries available for parallel programming: these libraries are installed just above the operating system, they hide the system calls interface, and usually they provide a simpler and more powerful way of avoiding race conditions.

C.3.2 Exclusive access to critical sections

Let assume that we have n processes, $\{P_1, P_2, ..., P_n\}$ where for each process P_i, $i = 1, \ldots, n$ there is a program section (critical section, CS) which accesses or manipulates shared information. In order to provide exclusive access to the CS, we have to ensure that when a process P_i executes the CS no other process can enter into it until P_i exits from it.

The way of solving this problem is to use a protocol to access critical sections, and for that purpose we will introduce two generic primitives, `enter_CS()` and `exit_CS()`, as shown next:

```
...
enter_CS()  /* if nobody in CS, continue, otherwise wait */

[the Critical Section ]

exit_CS()   /* now another process can enter the CS */
...
```

When the CS is free and some processes are waiting to enter it, the protocol has to specify which of them enters, forcing the rest to keep waiting. Usually, a First Come First Serve (FCFS) ordering is desirable, but the selection of the next process could also be based on the priorities of processes or in another aspect.

Critical sections are also very common in concurrent applications. Just as an example, we will use the producer-consumer example (Figure C.1) to illustrate the critical section problem and its possible solutions. This example was originally thought only for 1 producer-process and 1 consumer-process. A process is producing elements which are stored in a shared buffer calling a function `store_element()`. The full buffer condition is controlled using a shared variable `counter`. Analogously, a consumer process obtains elements from this buffer by using a function `obtain_element()` that also requires a critical section, therefore decreasing the counter in an unit. Apart from the access to the buffer, the producer-consumer example has another critical section on the access to the counter variable, which is also a shared

```
int counter= 0;

producer() {
    int element;

    while (true) {
        produce_element(&element);
        while (counter == N) NOP;
        store_element(element);
        counter=counter+1;
    }
}

consumer() {
    int element;

    while (true) {
        while (counter == 0) NOP;
        obtain_element(&element);
        counter=counter-1;
        consume_element(element);
    }
}
```

Figure C.1: Producer-consumer example with race conditions.

resource. The code in Figure C.1 has race conditions in the access to the buffer when it is either full or empty.

The solution to the problem of the exclusive access to critical sections is fundamental for the field of inter-process communication, as any other concurrent or parallel problem can be expressed by its means.

C.3.3 Conditions for critical sections

There are many possible implementations to the primitives `enter_CS()` and `exit_CS()`, but whichever implementation we choose it has to satisfy the following conditions as stated in [Dijkstra, 1965]:

Mutual exclusion. There cannot be more than a process executing the CS simultaneously.

No deadlock. No process blocked outside the CS can avoid any other to enter the CS.

Bounded waiting. A process cannot be waiting indefinitely for entering the CS.

Hardware independence. No supposition can be done about the number of processors or the relative processing speed of the processes.

Any protocol that implements the two primitives `enter_CS()` and `exit_CS()` has to fulfill these four conditions if it is to be regarded as a valid communication mechanism. In

order to compare the performance of two different implementations of this protocol, we will always have to do an initial assumption: the primitives `enter_CS()` and `exit_CS()` need to be atomic and are executed before entering and after exiting the critical section respectively.

C.3.4 Communication primitives

Inter-process communication solutions are normally classified following the synchronization method that they use, explicit or implicit. Doing it so, we have two main solution groups:

1. Using shared variables (it requires explicit synchronization)

   Active waiting :

   - Software: lock variables, Dekker's and Petterson's algorithms, Lamport's algorithm...
   - Hardware: interrupt inhibition, specific machine language instructions...

   Blocked waiting :

   - Basic primitives: sleep and wake up, semaphores...
   - High-level constructions: critical regions, monitors...

2. Using message passing (implicit synchronization)

Shared variables can be often mapped in memory. When this is the case, these are an adequate communication mechanism for synchronizing for instance execution-threads within a process. Disk files could also be used for communicating independent processes, but this solution is much less efficient. The message passing mechanism (explained in Section C.3.5) makes also use of shared objects for communication purposes, and its primitives guarantee exclusive access to these. Generally speaking, two processes communicate using a message passing mechanism, which is regarded as a very efficient and high level mechanism.

A.- Active waiting

Lock variables

If there is no specific synchronization mechanism to access a common resource, a possibility is to access critical sections by means of an active waiting algorithm that uses shared variables. These variables will be used to control the access to critical sections. But it is important to check the validity of the implementation, as the intuitive use of *lock variables* does not ensure exclusive access to the critical section, as shown here:

```
int lock=0;

Enter_CS: while (lock) NOP;    /* active waiting */
          lock= 1;

Exit_CS: lock= 0;
```

It is easy to check that this implementation of the `enter_CS()` and `exit_CS()` protocol does not satisfy the mutual exclusion property. The fact of requiring two separated instructions for both read the value of `lock` and activate it generates a new critical section in these

two instructions: another race condition is created between the first and the second line of `enter_CS()`.

The only existing software solutions are complex and computationally very time consuming –the algorithms of Peterson and Lamport are possible solutions, and they can be found for instance in [Silberschatz et al., 2000].

Inhibition and activation of interrupts

Some operating systems also provide the inhibition and activation of interrupt levels as a mechanism to solve the critical section problem. This solution is only valid for single-processor computers as it is highly restrictive. This method is based on the idea of stopping interrupt signals to reach the processor, and therefore on inhibiting the system from preemption[6] the process that is executing. It is implemented as follows:

```
Enter_CS: s = inhibit()  /* Inhibits all the interrupt levels */

Exit_CS:  activate(s)    /* Activates interrupts */
```

This mechanism of manipulating the interrupt levels is very restrictive and could create many additional drawbacks. Firstly, it does not satisfy the hardware independence condition, as it is dependent on hardware availability for this type of mechanism. On the other hand, if we use such a mechanism for a critical section that requires a long execution time, absolutely all the interrupts will be inhibited for a long time, leading to additional problems depending on the type of interrupts that should have been activated on that time. The inhibition of interrupts does not distinguish between types of interrupts, and as a result even very critical interrupt levels that would not threaten the integrity of the critical section will be inhibited.

B.- Blocked waiting

Every active waiting mechanism has some important drawbacks due to the fact that the protocol keeps on waiting by consultation to the lock variable. The main problems that this type of mechanisms carry are:

- During the waiting time, the CPU is in use preventing other processes to have access to it.
- A bad scheduler can lead all the processes to block to each other[7]: let us consider as an example that an operating system using two priorities for its processes, high (H) and low (L) levels, and two processes, P_H of high priority and P_L with low priority. Then if P_L is in the critical section and P_H wants to enter it, the scheduler will always decide to choose P_H giving P_L no chance to keep with its execution and therefore keeping P_H in active waiting indefinitely. This situation constitutes what it is called a *deadlock*. Also, this particular example shows that the active waiting mechanisms behave differently depending on their implementation in the operating system.

[6] *Preemption* is an operating system concept related to process planning. It is the result of a process being stopped in its execution on the CPU and another being executed in its place. This happens for instance when a higher priority process than the one in the CPU is ready to continue its execution: as a result the higher priority process replaces the lower one in the CPU. If lower priority processes are executing a critical section and if interrupts and inhibited no other process will change its status and thrown it away from the CPU. The interested reader can find more information in [Silberschatz et al., 2000].

[7] A scheduler is the internal routine of the operating system that selects will pass next to occupy a freed CPU.

The alternative for the active waiting is to block a process when it wants to enter the critical sections that it is busy. This section reviews some of the basic techniques for blocked waiting.

Sleep* & *wake-up

This two primitives allow to block a process P_i when executing `sleep()`. Another process P_j will wake up the sleeping process by calling the primitive `wake-up(` P_i `)`.

```
Enter_CS (for a process Pi):   /* if CS busy */ sleep();

Exit_CS (for a process Pj):    wake-up(Pi)
```

These two primitives are not able to provide exclusive access to critical sections by themselves, but they are to be used by any mechanisms that implement primitives for the protocol to access critical sections.

As with the rest of mechanisms reviewed so far, the sleep and wake-up primitives do also have some problems. If the process P_i is not sleeping, the primitive `wake-up(Pi)` will not have any effect, and therefore its call will be ignored and forgotten. Unfortunately, this situation can create race conditions in schemes such as the producer-consumer as it is written in Figure C.2 applying sleep & wake-up primitives, where the condition of full (or empty) buffer and the action of sleeping do again constitute a new critical section. This problem can be seen in the following example: if the producer has detected the full buffer function and just before executing the `sleep()` primitive if it is preempted from the CPU, the consumer could consume an element and execute the `wake-up(producer)` primitive before the system returns the control to the producer, and therefore the posterior call will be ignored. When the producer reaches again the CPU, it will not check again whether the buffer is full or not, and as a result it will go to sleep with no possibility of the consumer to execute again the waking up primitive. This situation constitutes again a deadlock state.

Semaphores

A more general abstraction is the *semaphore* [Dijkstra, 1965], which is built over the sleep & wake-up primitives. Semaphores keep record of all the sleeping and waking up primitives, waking up a single blocked process later when it is required.

Every semaphore contains a queue of processes that are blocked waiting to a waking-up event to arrive, as well as a counter for waking up signals already received. These features allow us to use semaphores both for inter-process synchronization as well as for managing resources. Semaphores are used for these purposes inside the operating system and at user level. Once defined a semaphore `s`, two basic atomic operations are allowed:

```
wait(s)   --also: p(s), down(s), sem_wait(s) ...

signal(s) --also: v(s), up(s), sem_post(s) ...
```

When a process executes the primitive `wait(s)` over a semaphore s, if the counter associated to s is strictly bigger than zero the process will continue and the counter will be decreased in a unit; otherwise, the process will be blocked (i.e. sent to sleeping by executing `sleep()`). When a process executes the primitive `signal(s)` the counter is incremented by one unit; however, if there are blocked processes, one of them will also be awakened.

```
int counter= 0;

producer() {
    int element;

    while (1) {
        produce_element(&element);
        if (counter == N) sleep();
        store_element(element);
        counter=counter+1;
        if (counter == 1) wake-up(consumer);
    }
}

consumer() {
    int element;

    while (1) {
        if (counter == 0) sleep();
        obtain_element(&element);
        counter=counter-1;
        if (counter == N-1) wake-up(producer);
        consume_element(element);
    }
}
```

Figure C.2: Example of the producer-consumer example solved with sleep & wake-up primitives. The solution proposed here is only valid for one producer and one consumer.

Initially a value is assigned to the semaphore's counter by means of an initialization primitive: `ini_semaphore(s,value)` which assigns the `value` to the counter associated to `s` and also inititializes the associated queue as an empty one.

Semaphores are a very common synchronization mechanism, and they are available in most of the modern operating systems. These systems provide the *wait*, *signal* and *ini_semaphore* operations in the form of system call primitives that can directly be used in our programs. The next section provides a revision of the many uses of the semaphores in order to show a general idea of their usefulness.

Use of semaphores. Semaphores can be used for instance for long term mutual exclusion management, as they only affect the processes willing to access the critical section at a particular period of time, without having influence on the rest of processes running on the system. The critical section problem can be solved with semaphores by initializing the semaphore to one[8], and then used in the following way:

```
Enter_CS: wait(mutex); /* if (CS busy) the process is blocked*/
```

[8]These semaphores initialized to one that are used for addressing the mutual exclusion property are often called *mutex* or *binary semaphores*.

```
struct semaphore_t mutex, holes, items;

ini_semaphore(mutex, 1); ini_semaphore(holes, N);
ini_semaphore(items, 0);

producer() {
    int element;

    while (1) {
        produce_element(&element);
        wait(holes);
        wait(mutex);
        store_element(element);
        signal(mutex);
        signal(items);
    }
}

consumer() {
    int element;

    while (1) {
        wait(items);
        wait(mutex);
        obtain_element(&element);
        signal(mutex);
        signal(holes);
        consume_element(element);
    }
}
```

Figure C.3: Example of solving the producer-consumer problem using semaphores.

```
Exit_CS:  signal(mutex);
```

Semaphores are often used as a more general synchronization tool, and they can also be applied for assigning shared resources between variables. Given n units of a shared resource that any process can use, the semaphore R for the resource is initialized,

```
ini_semaphore(R, n);
```

and a process will use the resource following the next protocol:

```
wait(R);

    /* use the shared resource */

signal(R);
```

Please, note that $n = 1$ is the particular case of the mutual exclusion. The producer-consumer problem, in which a process produces items and store them in a shared buffer while the consumer process reads them from the buffer and consumes them, is taken as an example of the use of semaphores for mutual exclusion and resource management (Figure C.3). A semaphore is used for the mutual exclusion, the one called `mutex`, in order to solve the exclusive access on the operations for storing and obtaining elements on the shared *buffer* resource[9]. Two other semaphores are also defined, one for blocking the producer when the buffer is full (*holes*) and another for blocking the consumer when the buffer is empty (*items*). Note that the initialization of all the semaphores is done in the example of Figure C.3. It is also worth saying that this solution of the producer-consumer problem is also valid for the more general case of having n producers and m consumers, $n, m > 1$.

Semaphores can also be used for communication purposes. For instance, in a client-server scheme the client will use a semaphore to synchronize with the end of server work, while the server will use a semaphore as a event for waiting clients' requests.

Finally, there are other more complex and typical synchronization problems such as the problem of the readers-writers that can be solved using semaphores –more information about this problem can be found in [Stallings, 2000] and [Andrews, 1991].

Implementation of semaphores. A semaphore is a predefined structure that allows only one of the three basic operations `wait`, `signal`, and `ini_semaphore` shown before. Once again, the implementation of these primitives has to be atomic, although in the following example the specification of critical sections has been omitted. The semaphores as well as their basic operations can be implemented in C language as follows:

```
struct semaphore_t {
  int counter;
  struct queue q;
}

void ini_semaphore(struct semaphore_t *sem, int val) {
  sem->counter= val;
  ini_queue(sem.q);
}

void wait(struct semaphore_t *sem) {
  if (sem->counter == 0) {
    put_process_in_queue(sem->q, my_process);
    sleep(my_process);     /* sleep the process to make it wait */
  }
  else --sem->counter;
}

void signal(struct semaphore_t *sem) {
  if (empty_queue(sem->q)) ++sem->counter;
  else wake-up(first(sem->q)); /* wake up the first in the queue */
}
```

[9] A more efficient implementation would use another short-term mutual exclusion mechanism such as lock variables instead of mutex semaphores for this simple case.

Barriers

Barriers are synchronization elements used in parallel programming to synchronize a finite set of n processes between them. A barrier will stop the first processes arriving to the synchronization primitive until all the n processes have executed the call. The generic synchronization primitive is executed as follows:

```
struct barrier_t bar; int n; ... barrier (bar, n) ...
```

where `bar` is the name of the barrier that we are applying and `n` is the number of processes that want to synchronize with it. The necessary condition continue the execution of a process executing the primitive `barrier` is that n processes have called this function.

Barriers are available in many parallel programming libraries and standards, and due to its simplicity, this mechanism is used very usually for synchronization purposes. When an operating system or a selected parallel programming library does not provide barriers, these can be easily implemented using either semaphores or lock variables.

C.3.5 Message passing

In all the solutions described so far we have assumed the existence of shared elements between execution threads (shared memory or shared files) where the shared variables for communication would be mapped in memory. In case of using threads within a single process, the use of global variables in memory is direct and intuitive, but they require explicit synchronization mechanisms using one of the many mechanisms already described.

Another alternative for inter-process communication, which is also suitable when these processes are executing on different computers communicating through a network, is to use *message passing* primitives. These message passing primitives make use directly or indirectly of specific communication elements[10] that behave as First In First Out (FIFO) queues. Message passing primitives ensure exclusive access to the communication channel, and therefore their use does no require explicit synchronization to be programmed when reading or writing a message.

The two generic communication primitives available in any message passing system are defined as follows:

```
send    --also called: write_message,
receive --also called: read_message,
```

Message passing communication types

There are many communication types available when message passing:

- direct communication

  ```
  send(process_id, message)
  receive(process_id, message)
  ```

[10]These communication elements receive many names in the literature: communication links, channels, or communication ports. In this dissertation we call them communication channels or just *channels*.

In this communication type, processes that want to communicate to each other need to know their id numbers, although sometimes an extension of this is used so that a process can send messages to any receiver or to receive messages from any sender (*broadcasting*):

```
send(ALL, message)
receive(ANY, message)
```

- Communication using mailboxes

```
send (mailbox, message)
receive (mailbox, message)
```

A mailbox is a named FIFO-type communication channel. The processes need to know the mailbox's name if they want to communicate to each other, but they do not need to know the other's id number. In general terms, there will be many processes sending messages to the mailbox and also many other reading from it.

In this section we will consider a message to be a character string. This is typically the case also in networking protocols. The length of the message can be either fixed or variable depending on the message passing implementation.

Synchronization with message passing primitives

In message passing, it is important to understand how the implicit synchronization behaves when two processes are communicating. The primitive `receive` will block the process if the communication channel is empty of messages, and otherwise the first message will be returned and the process will continue its execution. The behavior of the `send` operation can be different from an implementation to another, and this is dependent on many characteristics of the communication channel: alternatives such as being buffered or not buffered, or to be synchronous or asynchronous, will determine the behavior of the two primitives at a great extent. Another of these main characteristics is the *capacity*. The capacity of the communication channel determines the way of synchronizing the processes when accessing the mailbox for writing. Depending on this characteristic the communication channels can have:

Unlimited capacity: this is the ideal communication channel. The sending process will never, under any condition, be blocked.

Limited capacity: the sender-process will be blocked only when the communication channel is full.

Null capacity: this possibility makes sense only in direct communication. As no message can be stored, the two processes have to synchronize to each other every time they want to communicate: the first of them that is ready for sending or receiving is blocked until the other is ready for the opposite operation. This mechanism is also known as *rendezvous*. This procedure is similar to the use of barriers by two processes, but in this case the communication of information is also included.

```
producer() {
    int element;
    char message[50];

    while (1) {
        produce_element(&element);
        compose_message (&message, element);
        send(MBX, message);
    }
}

consumer() {
    int element;
    char message[50];

    while (1) {
        receive(MBX, &message);
        decompose_message (message, &element);
        consume_element(element);
    }
}
```

Figure C.4: Example of the producer-consumer scheme using message passing.

Except from the case of null capacity, the message passing mechanism leads to an asynchronous model for processes to work. This means that the sender can usually continue its execution even if the receiver has not still received the message. As an example of using message passing primitives, Figure C.4 shows how to solve the producer-consumer problem using a mailbox called `MBX`.

C.3.6 Communication and synchronization paradigms

The exclusive access to critical sections is a fundamental problem in inter-process communication and synchronization. The previous sections have reviewed the basic mechanisms that can be used to solve this problem in our programs. The paradigms described on this section are typical communication and synchronization situations (the producer-consumer one is just an example of them) that can be solved by means of these basic mechanisms, once they usefulness on the critical section problem has been proved. These paradigms are different from the ones introduced in Section 5.2.1 in the sense that the ones introduced here are typical basic applications that are used for measuring the performance of the different communication and synchronization mechanisms for different types of problems. Some other more complex paradigms are also briefly described in this section.

The typical communication and synchronization paradigms are the following:

Readers & **writers** : The access of shared information in disk files, databases, and other types of shared resources is usually asymmetric, that is, most of the accesses are for reading and much less for writing. In these cases, the definition of critical sections for exclusive access is too restrictive and inefficient, as in most of the cases n reader

processes could be allowed to be accessing the critical section at the same time without restrictions. Only when the information needs to be updated is required that the critical section is free before the writer enters it. This problem can be solved in different ways using for instance semaphores. This solution is shown in [Stallings, 2000].

Assigning multiple resources : The basic problem of assigning n resources has been introduced in Section C.3.4, as semaphores can be applied in an easy and simple way. When the resources are of different types assignments can be done to different processes at the same time, the problem acquires a very complex nature, and usually falls on unbounded waiting and deadlock problems. There are many solutions for this problem to avoid the deadlock state that can be found in [Dijkstra, 1965].

Client-server : This paradigm is a particular case of the producer-consumer problem, and it is applied in many situations (i.e. general applications, inside the operating system, in networking...) to control the use of resources. A process plays the role of the server, and it centralizes the requests to access a resource (or many resources) requested by the client-processes. In the producer-consumer case, clients and server communicate with each other by a shared buffer (a communication channel or a mailbox if we use a message passing interface) to store clients' requests. The server will treat them sequentially, avoiding any conflict in accessing resources between clients. The client-server scheme is used extensively in concurrent applications, and it is well suited for implementing services between processes, specially for processes in different computers communicating through a network.

Appendix D

Analysis and parallelization of the source code of EBNA$_{BIC}$

D.1 Short review of the source code of the sequential EDA program

Next, the parts of the code in the sequential EDA program that are more promising for parallelizing are described. This section is shown as an example of how parallelization has to be done and which are the main implications of it.

We will concentrate on the discrete case, more precisely in the EBNA algorithm and its way to compute the BIC score. As every node in the candidate Bayesian network does contribute to the overall BIC score of the structure, the score is computed first for each of the nodes. However, for our purpose we are not interested in the overall BIC score, but in finding Bayesian network that optimizes it. That is why we use a matrix `A`, where for every two nodes i and j of the Bayesian network, `A[i,j]` will record the change in the BIC score for the arc from i to j when it is added (if there is not an arc from i to j) or deleted (when there is that arc) in the Bayesian network. Other global variables in the sequential program that take part in the computation of the BIC score are the following:

IND_SIZE: the number of variables of the individuals. Therefore, this is also the number of nodes in the probabilistic graphical structure.

SEL_SIZE: the number of selected individuals from which the probabilistic graphical structure is to be learned.

cases: this matrix stores the selected individuals from which the learning is to be done. Its size is *SEL_SIZE* $\times$ *IND_SIZE*.

parents: this boolean matrix has a size of *IND_SIZE* $\times$ *IND_SIZE* and it is used to represent the Bayesian network: if i and j are two nodes, then when `parent[i,j]` is true it means j is the parent of i in the Bayesian network.

STATES: this integer vector of size IND_SIZE tell us the number of values that each variable can take.

The parts of the sequential program that perform the learning in EBNA are the following:

```
void CalculateA(int **&cases) {
  for(int i=0;i<IND_SIZE;i++)
        CalculateANode(i,cases);
}

void CalculateANode(int node, int **&cases) {
    double old_metric, new_metric;
    old_metric = BIC(node,cases);

    for(int i=0;i<IND_SIZE;i++)
      if (i!=node)
        {
            //change the j->i arc in the Bayesian network
            parents[node][i] = !parents[node][i];
            new_metric = BIC(node,cases);

            //restore the j->i arc in the Bayesian network
            parents[node][i] = !parents[node][i];

            //Compute difference
            A[node][i] = new_metric - old_metric;
        }
        else A[node][i] = INT_MIN;
}
```

And the actual BIC function is computed with the following function:

```
double BIC(int node, int **&cases) {
    int j,k;

    // Calculate the number of combinations of parent values.
    int no_j = 1;
    for(j=0;j<IND_SIZE;j++)
        if(parents[node][j]) no_j *= STATES[j];

    // Allocate memory for all nijk-s and initialize them.
    int ** nijk = new int*[no_j];
    for(j=0;j<no_j;j++)
    {
        nijk[j] = new int[STATES[node]];
        for(k=0;k<STATES[node];k++) nijk[j][k] = 0;
    }

    // Calculate all nijk-s.
    for(j=0;j<SEL_SIZE;j++)
    {
        // Find the parent configuration for the j-th case.
        int parent_configuration = 0;
        for(int parent=0;parent<IND_SIZE;parent++)
```

```
            if(m_parents[node][parent])
            {
                parent_configuration *= STATES[parent];
                parent_configuration += cases[j][parent];
            }

        // Update the corresponding nijk.
        nijk[parent_configuration][cases[j][node]]++;
    }

    // Calculate the BIC value.
    double bic = 0;
    for(j=0;j<no_j;j++)
    {
        int nij = 0;
        for(k=0;k<STATES[node];k++)
            nij += nijk[j][k];
    }

    bic -= log(SEL_SIZE)*no_j/2;

    // Free the memory allocated for the nijk-s.
    for(j=0;j<no_j;j++)
        delete [] nijk[j];
    delete [] nijk;

    return bic;
}
```

D.2 Parallelization using threads

The score+search procedure will be parallelized by making threads to divide the work: a thread plays the role of the *manager* that distributes the work among the rest, and the others are the *workers* that have to compute all the possible arc modifications for a same number of nodes (i.e each worker will execute the procedure `CalculateANode` for a total of `IND_SIZE/number_of_workers` nodes).

D.2.1 New adaptation on the source code for threads

The parallel program will use shared memory in the form of defined global variables for communication between the different cooperating threads. However, the use of shared variables for communication leads to the existence of race conditions within the program, and therefore a synchronization mechanism is required to ensure exclusive access to the critical sections in the program. The multithreading standard library selected is *pthreads*.

As already explained in Section 6.4.1, we apply a manager-slave working scheme. In our case, we decided to apply a manager-slave working scheme, where a thread plays the role of the manager and the rest of workers wait for a node number whose particular BIC score has

to be computed. The worker threads will compute the BIC score by making use of common resources that are present in shared memory such as the global variables `cases`, `parents`, and `STATES`, and they will finally write their results in the shared matrix `A`.

In order to control the maximum number of threads that can be working at the same time a semaphore called `SemMaxChildren` is defined. This semaphore is initialized to the maximum number of threads that can exist. In our case we have a two processor computer, and therefore this limit was set to 4 threads.

In addition a table is created in order to store the thread identification for each worker-thread in the program. This one-dimensional table is called `WorkerId`, and it is used by the primitives to create and wait for threads.

Primitives to create and organize threads.

Having all this aspects in mind, some functions were defined in order to create an easier to use abstraction layer of the underlying low level pthreads interface. The calls to native *pthreads* primitives are easy to identify in the source code, as they all have the prefix `pthread_` on their names. These functions are intended to respect the maximum limit of threads established:

*** Creation of a thread.** This function allows a program to create a new thread within the process that executes it. If a call to the function `CreateThread` is done and there are already too many threads created, the function will block the call until one of the workers finishes, although this control is supposed to be done by the manager before trying to create a new thread. The last parameters of this function is the worker number of the thread, and the id of the newly created thread is stored in the table `WorkerId`. The function returns the number of the thread that is created, or -1 in case of error:

```
unsigned long CreateThread(void * (*my_func)(void *),
                           void *arglist, int NumWorker)
{
  pthread_t NumChildThread;
  pthread_attr_t thread_attr;
  int ThreadNr;
  void *(*functname)(void *) = (void *(*)( void * )) my_func;

  //making sure that there are not too many
  //threads already created
  sem_wait( &SemMaxChildren );
  num_threads++;

  status = pthread_attr_init (&thread_attr);
  if (status != 0)
      cerr << "Error " << status << ": Create attr" << endl;

  //Create a detached thread: this is required in order to
  //use more than a single CPU by a process
  status = pthread_attr_setdetachstate (&thread_attr,
                                  PTHREAD_CREATE_DETACHED);
```

```
  if (status != 0)
      printf("Error %d: Set detach", status);

  status = pthread_create (&NumChildThread, &thread_attr,
                           functname, arglist);
  if (status!=0) {
      printf("Error: the new thread could not be created!");
      return (-1);
  }

  //Set the worker number in the table
  WorkerId[NumWorker] = NumChildThread;

  return ((unsigned long) NumChildThread);
}
```

* **End of a thread.** When a thread finishes its work it is destroyed by calling the function `EndThread`. However, if a thread is blocked waiting the end of another to be created, this function will unblock the call to `CreateThread`.

  As the *pthreads* library does not have any primitive to manage the end of the threads in a way that we need, we created a queue to store the end of threads within our application. Each time a thread finishes, the returned value is stored in the queue until a thread that is waiting for the end of another thread receives the information. This queue is a shared resource in memory, and therefore all the lines of code accessing it constitute a critical section. This critical section is managed with a mutex called `MutexQueue`. As this queue is a very typical example of a structure is memory that stores elements, the access to it has been represented in these two primitives: `insert_in_queue` and `read_from_queue`.

```
void EndThread(void *value_ptr)
{
  sem_post( &SemMaxChildren );

  //Record the end of the thread event to synchronize it with
  //the WaitingThread primitive
  wait( &MutexQueue);
  insert_in_queue(value_ptr, &EndedThreads);
  num_threads--;
  signal( &MutexQueue);

  sem_post( &SemWaitingAnyThread);

  pthread_exit(value_ptr);
  return;
}
```

* **Kill another thread.** This function can be used to kill another thread. The id of the thread to kill must be provided. This function also stores a value in the queue to

control the end of processes, but in this case an error code of -1 is sent:

```
void KillThread(unsigned long ThreadNumber)
{
  pthread_kill( (pthread_t) ThreadNumber, SIGKILL);

  //Record the end of thread event
  wait( &MutexQueue);
  insert_in_queue(-1, &EndedThreads);//-1: code for errors
  num_threads--;
  signal( &MutexQueue);

  sem_post( &SemWaitingAnyThread);

  }

  return;
}
```

* **Waiting for a single thread.** This function is used to make threads wait for another's end. The only parameter required is the id of the thread that we are waiting for. If this parameters has the value 0, the function will wait for any of the existing threads. If no other thread is created within the process' environment, a error value of -1 is returned. It is also important to see that the tread will not be destroyed until another is ready to receive the end signal. This is controlled by synchronizing the ending thread with the someone waiting for a thread end by means of the semaphore `SemWaitingAnyThread` that is also used in the `EndThread` primitive described previously.

```
int WaitThread(unsigned long ThreadNumber)
{
  int ReturnValue;

  //If there are no threads left return an error
  if (num_threads==0) return(-1);


  sem_wait( &SemWaitingAnyThread);

  wait( &MutexQueue);
  if (ThreadNumber==0) {
      //The waiting thread is synchronized with the end
      //of someone else, and the returned value is given
      read_first_from_queue(&ReturnValue, &EndedThreads);
      return (int) ReturnValue;
  }
  else {
      signal( &MutexQueue );
      return thr_join(ThreadNumber, NULL, NULL);
```

```
  }
}
```

* **Waiting for all threads.** This function is used to implement more easily the manager-slave scheme. It is used by the manager in order to keep waiting for the ending of all of the worker threads. This function has only a parameter which is the number of worker threads that were created, and it makes use of the table `Workerid` to obtain the id number of the next thread to wait for.

```
void WaitForAllThreads(int number_worker_threads)
{
  int i;

    //wait for all threads. If there is no left, return.
    if (m_num_threads==0) return;

    for ( i = 0; i < number_worker_threads; i++){
      pthread_join(m_Workerid[i], NULL);
      sem_wait (&SemWaitingAnyThread);
    }

    //This function does not use the queue, so empty it now
    wait( &MutexQueue);
    initialize_queue_to empty();
    signal( &MutexQueue);

    return;
}
```

The code of the manager and the workers.

Having defined all these primitives, we have now to change the sequential program and adapt it for the manager-slave working scheme. For this, two new functions are defined: `ParallelBIC` will be the one executing the master-thread, and `ParallelBICWorker` will be the one executing all the worker-threads. The function `ParallelBIC` will have two arguments: which are the number of nodes that the Bayesian network has (this is also the number of tasks to do in parallel) and the number of the node that we are computing each time. In fact, the function in the sequential program where the job is divided is `CalculateANode` in which all the relationships with the rest of the nodes are computed using the BIC score, and the function `ParallelBIC` will divide the whole task in pieces giving each worker the corresponding amount of work. We will use some global variables for communication between the manager and the slaves (`CurrentTask`, `MaxNumTasks`, `Node`, `OldMetric` and `JobSize`). In addition, a mutex called `MutexComm` is used for ensuring the mutual exclusion when accessing critical variables used for communication.

```
void ParallelBIC(int NumTasks, int node)
//NumTasks is the total amount of nodes
//Node is the node number that we are processing in parallel.
```

```
{
   //Initialize global variables
   CurrentTask =0; //we start from the node 0
   Node = node; //number of the node that we are treating
   JobSize = ((NumTasks-1) / MAX_THREADS)+1;
   MaxNumTasks = NumTasks;

   //The actual value of the metric is computed
   //so that the workers compare it with their
   //work. This is a sequential task that
   //cannot be parallelized.
   OldMetric = BIC(Node, cases);

   // Creation of the parallel crew by using threads
   for(int i=0;i<NumTasks;i++)
   {
       WorkerNum=num_threads;
       CreateThread((void * (*)(void *)) ParallelWorkerBIC, NULL);
   }

   //Wait until all the last threads have finished their work.
   WaitForAllThreads(MAX_THREADS);

   return;
}

void ParallelWorkerBIC(void) {
   int FirstJob, LastJob;
   double new_metric;

   //Calculate the part of the work to do - critical section
   wait( &MutexQueue);
   FirstJob = (CurrentTask) * (JobSize);
   LastJob = ((CurrentTask + 1) * (JobSize)) -1;
   CurrentTask++;
   signal( &MutexQueue);

   //Check that the end of the work is not reached
   if (LastJob > MaxNumTasks) LastJob = MaxNumTasks;

   for(int i=FirstJob;i<LastJob;i++) {
     if (i!=Node) {

         //a new function is used to compute the difference of
         //changing the arc i-> node. Here we cannot change
         //the variable parents as this is shared with the rest
         // of the threads
```

```
          new_metric = deltaBIC(Node,i, cases);

          A[Node][i] = new_metric - OldMetric;
      }
      else A[Node][i] = INT_MIN;
    }

    EndThread(NULL);
    return;
}
```

The new function `deltaBIC` that is used by the workers to compute the difference in the BIC score if we modify the arc that in the nodes *nparent-¿node* by adding it (if it does not exist in the Bayesian network) or removing it (if it does exist in the Bayesian network). Thereof the name of the function, which is very similar to the BIC function, essentially it does only change the consideration of the nparent value on the computation. However, here we compute the difference in the partial BIC score of the node `node` without changing the value of the global variable `parents`. This last aspect is very important as the variable `parents` is shared among all the working-threads and any change on it will also corrupt the computation for the rest of the working-threads:

```
double deltaBIC(int node, int nparent, int **&cases) {int j,k;

    // Calculate the number of parent configurations.
    int no_j = 1;
    for(j=0;j<IND_SIZE;j++) {
      //If we are analyzing nparent, consider it as changed
      if (j!=nparent) {
        if(parents[node][j]) no_j *= STATES[j];
      }
      else {
        if(!(m_parents[node][nparent])) no_j *= STATES[nparent];
      }
    }

    // Allocate memory for all nijk-s.
    int ** nijk = new int*[no_j];
    for(j=0;j<no_j;j++)
    {
        nijk[j] = new int[STATES[node]];
        for(k=0;k<STATES[node];k++) nijk[j][k] = 0;
    }

    // Calculate all nijk-s.
    for(j=0;j<SEL_SIZE;j++)
    {
        // Find the parent configuration for the j-th case.
        int parent_configuration = 0;
        for(int parent=0;parent<IND_SIZE;parent++) {
```

```
        //If we are analyzing nparent, consider it as changed
        if (parent!=nparent) {
          if(m_parents[node][parent])
          {
              parent_configuration *= STATES[parent];
              parent_configuration += cases[j][parent];
          }
        }
        else {
          if(!(m_parents[node][nparent]))
          {
              parent_configuration *= STATES[nparent];
              parent_configuration += cases[j][nparent];
          }
        }
      }

      // Update the corresponding nijk.
      nijk[parent_configuration][cases[j][node]]++;
  }

  // Calculate the BIC value.
  double deltabic = 0;
  for(j=0;j<no_j;j++)
  {
      int nij = 0;
      for(k=0;k<STATES[node];k++) nij += nijk[j][k];
  }

  deltabic -= log(SEL_SIZE)*no_j/2;

  // Free the memory allocated for the nijk-s.
  for(j=0;j<no_j;j++)
      delete [] nijk[j];
  delete [] nijk;

  return deltabic;
}
```

We also have to initilialize all the structures and shared variables at the beginning of the program in the following way:

```
#include <limits.h> #include <semaphore.h> #include <pthread.h>

#define MAX_THREADS  4  /* Max Number of Worker-threads
                           since we have 2 processors */

... pthread_t thread; int num_threads=0; //this variable is
updated automatically
```

```
                    //in CreateThread and EndThread
int JobSize, CurrentTask, MaxNumTasks, Node; double OldMetric;
          //Variables for communication between master and workers

pthread_mutex_t MutexQueue; pthread_mutex_t MutexComm; sem_t
SemMaxChildren, SemWaitingAnyThread; int WorkerId[MAX_THREADS];
... main() {
  ...
  //Initialise semaphores and mutex
  sem_init(&SemMaxChildren, 0, MAX_THREADS);
  pthread_mutex_init(&MutexQueue, NULL) ;
  pthread_mutex_init(&MutexComm, NULL) ;
  sem_init(&SemWaitingAnyThread, 0, 0);
  ...
}
```

And finally, we have to change the `CalculateANode` function of the sequential version of the program so that it executes the parallel version instead of the sequential one. This time it requires only one parameter, as the `cases` matrix has been converted as a global variable in order to be accessed by all the threads at the same time:

```
void CalculateANode(int node) {
    ParallelBIC(IND_SIZE, node);
}
```

This example shows how to adapt a sequential program to convert it as parallel. We have used the BIC score as an example, but any other scores for the discreet case could also use the same idea and they would easily be adapted following these steps. Furthermore, the continuous version of the EDA program can also be adapted in a similar way. The continuous EDAs EGNA$_{BIC}$ and EGNA$_{BGe}$ where also parallelized following a similar approach.

D.3 Parallelization using MPI

MPI has been designed to use message passing as the communication mechanism for inter-process communication. MPI provides an efficient mechanism for threads from different processes, and it can also been applied even when shared memory is available.

As we have seen in the previous section, the parallel version of EBNA using threads makes use of shared variables not only for communication, but also for reading the data required for the `ParallelWorker` function. The fact of using shared memory also required the use of external synchronization mechanisms such as semaphores.

On the other hand, in the particular example of the EDA program, the fact that processes cannot share any memory among them requires the manager to send all the data structures required to compute the `deltaBIC` function to each of the workers. The data structures that have to be sent by the manager to the workers every time we create a worker-process are the matrices `cases` and `parents`, the vector `STATES`, and the global variables `IND_SIZE` and `SEL_SIZE`. These variables have a length according to the size of the problem, and when applying EDAs for a big example they can increase in size rapidly as well as the amount of information to communicate to the workers.

D.3.1 New adaptation on the source code for MPI

The main function of the source code is organized as follows: MPI creates all the processes at the beginning of the execution, so it is important to make the workers wait and to make the master start the initialization phase and to make it execute all the sequential parts of the program. We have designed a communication protocol for communicating the master and the workers that contains two steps: firstly, the manager will send an integer to all the workers in order to inform them about the next operation they have to perform, where 0 means to compute the BIC score in parallel and 1 means to finish their execution and terminate. If we wanted the workers to parallelize another second part of the EDA program we could also have created a new operation and added another work-code to the protocol. If the program is to be finished, workers will end after receiving the 0 value and then the manager will prepare the final results. If the BIC score is to be computed in parallel, the master will have to send to all the workers all the data matrix, vectors and variables they require for performing their job, and then the manager will receive all the results from all the workers in order to gather all the parts of the `A` matrix computed in parallel by the workers.

The amount of work that each worker has to do will depend on the number of workers and the size of the matrix `A`, and therefore the division on the amount of work for each process will be computed automatically taking these aspects into account. The main function of the source code that carries out this approach is the following:

```
main(int argc, char* argv[]) {
  int i;

  //Initialize the MPI system
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &g_size); //g_size <- number of processes
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank); //my_rank <- process MPI id

  NumWorkers = g_size; //1 manager-worker, n-1 workers

  if (my_rank == 0) {
    //Code for the Master

    //The initilization CommInitializeBICParallel_MPI() is done once
    //the parameters are read on the program

    EDA(argc, argv); // The manager reads the arguments and executes
                     // the part of the program reserved for the
                     // master.

    //At the end of the program the master send the code 0 to
    //indicate the workers that they have finished their work.
    op = 0;
    MPI_Bcast(&op, 1, MPI_INT, 0, MPI_COMM_WORLD);
  }
  else {
    //Code for the Workers
```

```
    //Initializes the variables required for the rest of the work.
    CommInitializeBICParallel_MPI() ;

    while (1){ //Execute once and again...

      //wait until receiving a code of operation to perform
      MPI_Bcast(&op, 1, MPI_INT, 0, MPI_COMM_WORLD);

      switch(op) {  //depending on the operation-code

      case 1: //Compute the BIC score
              CommStartBICParallel_MPI(); //read all the data
                                          //structures required
              //Calculate which is the corresponding part of A
              //for this worker to compute
              for (i=0; i<WorkSize; i++)
                   ParallelWorkerBIC(my_rank*WorkSize+i, i);

              //Send the results
              CommEndBICParallel_MPI();

              break;
      case 0: //exit the program
      default: break;
      }

      if (op==0) break; //end the while.
    }
  }

  if (my_rank == 0) {

    //Write final results
    WriteFinalResults();
  }

  MPI_Finalize();
}
```

The initialization function at the beginning of the program is executed by all the processes, either the manager and the workers, and it is written as follows:

```
void CommInitializeBICParallel_MPI() { int i,j;

  //The manager computes the size of work for each worker
  if (my_rank ==0) {
    WorkSize = IND_SIZE/NumWorkers;
  }
```

```
  //Receive IND_SIZE
  MPI_Bcast(&IND_SIZE, 1, MPI_INT, 0, MPI_COMM_WORLD);
  //Receive SEL_SIZE
  MPI_Bcast(&SEL_SIZE, 1, MPI_INT, 0, MPI_COMM_WORLD);
  //Receive Worksize
  MPI_Bcast(&WorkSize, 1, MPI_INT, 0, MPI_COMM_WORLD);

  //Reserve memory for global variables
  parents_MPI = (bool**) malloc (IND_SIZE*sizeof(bool *));
  A_MPI = (double **) malloc (IND_SIZE*sizeof(double *));
  for (i=0; i<IND_SIZE; i++) {
    parents_MPI[i] = (bool *) malloc (IND_SIZE*sizeof(bool));
    A_MPI[i] = (double *) malloc (IND_SIZE*sizeof(double));
  }
  cases_MPI = (int **) malloc (SEL_SIZE*sizeof(int *));
  for (i=0; i<SEL_SIZE; i++) {
    cases_MPI[i] = (int *) malloc (IND_SIZE*sizeof(int));
  }
  STATES_MPI =  (int *) malloc (IND_SIZE*sizeof(int));

  //Receive STATES
  if (my_rank ==0) { // The manager prepares the matrix STATES to be sent
    for (i=0; i<IND_SIZE; i++)
      STATES_MPI[i] = STATES[i];
  }
  MPI_Bcast(STATES_MPI, IND_SIZE, MPI_INT, 0, MPI_COMM_WORLD);

  return;
}
```

The function that will be executed at the beginning of the parallel BIC operation will send all the required input data that changes from a generation to the next to all the workers. The function executed at the end of the parallel BIC operation will send all the results to the manager. These two functions are implemented as follows:

```
void CommStartBICParallel_MPI() { int i,j;

 //Receive parents
 MPI_Bcast(parents_MPI, IND_SIZE*IND_SIZE, MPI_INT, 0, MPI_COMM_WORLD);

 //Receive cases
 MPI_Bcast(cases_MPI, SEL_SIZE*IND_SIZE, MPI_INT, 0, MPI_COMM_WORLD);

}

void CommEndBICParallel_MPI() { int i,j;

 //Gather the Results
```

```
 MPI_Gather(A_MPI, WorkSize*IND_SIZE, MPI_DOUBLE, A_MPI,
            WorkSize*IND_SIZE, MPI_DOUBLE, 0, MPI_COMM_WORLD);

}
```

Note that these functions are short in source code, but in fact require a high overload of exchanged information, as some of the matrixes that are sent such as the `cases` can have a very big size. Therefore, the time to send them to the workers can be of considerable importance. It is also important to realize that these functions where not required when using threads, as these were shared in memory among all the workers.

All the functions seen up to now where just to exchange data between the master and the slaves. All of them were not required in the parallel version of the BIC program with threads. The following functions are the ones executed by the master and the workers respectively in order to perform the parallel BIC task. The main difference on this new version with MPI is that the master will also play the role of a worker at the same time, and this is done in such a way because the creation of a new process and sending all the input variables to it is very time consuming:

```
void MPI_ParallelBIC(int NumTasks)
//The Manager
{
  int i,j;

  //The manager organizes the work:
  //Send the signal for job to all the workers
  op = 1;
  MPI_Bcast(&op, 1, MPI_INT, 0, MPI_COMM_WORLD);

  CommStartBICParallel_MPI();

  //The master also takes part on the computation
  //of a part of the parallel BIC similarly as
  //the rest of the workers
  for (i=0; i<WorkSize; i++)
       ParallelWorkerBIC(my_rank*WorkSize+i, i);

  //Gather Results from the rest of workers
  CommEndBICParallel_MPI();

  //Now all the results are stored in the matrix A
}


void ParallelWorkerBIC(int node, int TaskNumber) {
    double old_metric, new_metric;
    old_metric = BIC(node);

    for(int i=0;i<IND_SIZE;i++)
```

```
        if(i!=node) {
           new_metric = deltaBIC(node,i);
           A_MPI[TaskNumber][i] = new_metric - old_metric;
        }
        else A_MPI[TaskNumber][i] = INT_MIN;

    return;
}
```

where the `BIC` and `deltaBIC` functions are defined exactly as in the parallel case with threads.

Finally, just a short mention of the inclusion of the following lines on the program

```
#include <limits.h>
#include "mpi.h"
```

Bibliography

A. M. Abdulkader. *Parallel Algorithms for Labelled Graph Matching*. PhD thesis, Colorado School of Mines, 1998.

H. Akaike. New look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.

G. R. Andrews. *Concurrent Programming. Principles and Practice*. The Benjamin-Cummings Publishing Company, 1991.

R. Bacik. *Structure of Graph Homomorphism*. PhD thesis, Simon Fraser University, Canada, 2001.

T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, 1996.

J. Baget and M. Mugnier. Extension of simple conceptual graphs: the complexity of rules and constraints. *Journal of Artificial Intelligence Research*, 16:425–465, 2002.

M. Bakircioglu, U. Grenander, N. Khaneja, and M. Miller. Curve matching on brain surfaces using Frenet distances. *Human Brain Mapping*, 6(5-6):329–333, 1998.

S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical report, Carnegie Mellon Report, CMU-CS-94-163, 1994.

S. Baluja and S. Davies. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. Technical report, Carnegie Mellon Report, CMU-CS-97-107, 1997.

W. Banzhaf. The molecular traveling salesman. *Biological Cybernetics*, 64:7–14, 1990.

D. A. Basin. A term equality problem equivalent to graph isomorphism. *Information Processing Letters*, 54:61–66, 1994.

E. Bengoetxea, P. Larrañaga, I. Bloch, and A. Perchant. Estimation of distribution algorithms: A new evolutionary computation approach for graph matching problems. In M. Figueiredo, J. Zerubia, and A. K. Jain, editors, *Lecture Notes in Computer Science 2134*, pages 454–468, Sophia Antipolis, France, 2001a. Third International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition (EMMCVPR-2001).

E. Bengoetxea, P. Larrañaga, I. Bloch, and A. Perchant. Image recognition with graph matching using estimation of distribution algorithms. In *Proceedings of the Medical Image*

Understanding and Analysis – MIUA 2001, pages 89–92, Birmingham, United Kingdom, 2001b.

E. Bengoetxea, P. Larrañaga, I. Bloch, and A. Perchant. Solving graph matching with EDAs using a permutation-based representation. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pages 243–265. Kluwer Academic Publishers, 2001c.

E. Bengoetxea, P. Larrañaga, I. Bloch, A. Perchant, and C. Boeres. Inexact graph matching using learning and simulation of Bayesian networks. An empirical comparison between different approaches with synthetic data. In *Proceedings of CaNew workshop, ECAI 2000 Conference, ECCAI*, Berlin, 2000.

E. Bengoetxea, P. Larrañaga, I. Bloch, A. Perchant, and C. Boeres. Inexact graph matching using learning and simulation of probabilistic graphical models. Technical Report 2001D017, Ecole Nationale Supérieure des Télécommunications, 2001d.

E. Bengoetxea, P. Larrañaga, I. Bloch, A. Perchant, and C. Boeres. Learning and simulation of Bayesian networks applied to inexact graph matching. *Pattern Recognition*, 35(12): 2867–2880, Dec 2002a.

E. Bengoetxea, J. Miguel, P. Larrañaga, and I. Bloch. Model-based recognition of brain structures in 3D magnetic resonance images using graph matching and parallel estimation of distribution algorithms. *International Journal of Cybernetics and Systems*, 2002b. Submitted.

E. Bengoetxea, T. Miquélez, P. Larrañaga, and J. A. Lozano. Experimental results in function optimization with EDAs in continuous domain. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pages 181–194. Kluwer Academic Publishers, 2001e.

Beowulf. Beowulf parallel worstation project. URL: `http://www.beowulf.org` . Scyld Computing Corporation, 1994.

S. Berretti, A. del Bimbo, and E. Vicario. Efficient matching and indexing of graph models in content-based retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1089–1105, Oct 2001.

P. Bhat, V. Prasanna, and C. Raghavendra. Adaptive communication algorithms for distributed heterogeneous systems. *Journal of Parallel and Distributed Computing*, 59(2): 252–279, Nov 1999.

E. Bienenstock and C. von der Malsburg. A neural network for invariant pattern recognition. *Europhysics Letters*, 4(1):121–126, 1987.

R. E. Blake. Partitioning graph matching with constraints. *Pattern Recognition*, 27(3): 439–446, 1994.

I. Bloch. Fuzzy relative position between objects in image processing: a morphological approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(7):657–664, 1999a.

I. Bloch. On fuzzy distances and their use in image processing under imprecision. *Pattern Recognition*, 32:1873–1895, 1999b.

C. Boeres. *Heurísticas para Reconhecimento de Cenas por Correspondência de Grafos*. PhD thesis, Universidade Federal do Rio de Janeiro, Brazil, Sep 2002.

C. Boeres, A. Perchant, I. Bloch, and M. Roux. A genetic algorithm for brain image recognition using graph non-bijective correspondence. Ecole Nationale Supérieure des Télécommunications, Unpublished manuscript, 1999.

R. R. Bouckaert. A stratified simulation scheme for inference in Bayesian belief networks. In *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–94)*, pages 110–117. Seattle, WA, 1994.

R. R. Bouckaert, E. Castillo, and J. M. Gutiérrez. A modified simulation scheme for inference in Bayesian networks. *International Journal of Approximate Reasoning*, 14:55–80, 1996.

G. E. P. Box and M. E. Muller. A note on the generation of random normal deviates. *Ann. Math. Statist.*, 29:610–611, 1958.

A. Branca, E. Stella, and A. Distante. Qualitative scene interpretation using planar surfaces. *Autonomous Robots*, 8(2):129–139, Apr 2000.

R. P. Brent. A Gaussian pseudo random generator. *Comm. Assoc. Comput. Mach.*, 17: 704–706, 1974.

H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689–694, Aug 1997.

H. Bunke. *Error-tolerant Graph Matching: a Formal Framework and Algorithms*, volume 1451 of *Lecture Notes in Computer Science*. Springer, Berlin, 1998.

H. Bunke. Error correcting graph matching: on the influence of the underlying cost function. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(9):917–922, 1999.

H. Bunke. Recent advances in structural pattern recognition with applications to visual form analysis. In C. Arcelli, L.P. Cordella, and G. Sanniti de Baja, editors, *Visual Form 2001. Proceedings of the Fourth International Workshop on Visual Form IWVF4. Lecture Notes in Computer Science 2059*, pages 11–23, Springer-Verlag, Berlin, Germany, 2001.

H. Bunke and S. Gunter. Weighted mean of a pair of graphs. *Computing*, 67(3):209–224, 2001.

H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3–4):255–259, Mar 1998.

W. Buntine. Theory refinement in Bayesian networks. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pages 52–60, 1991.

W. Buntine. A guide to the literature on learning probabilistic networks from data. *IEEE Transactions on Knowledge and Data Engineering*, 8(2):195–210, 1996.

M. Carcassoni and E. R. Hancock. Weighted graph-matching using modal clusters. In W. Skarbek, editor, *Computer Analysis of Images and Patterns. Proceedings of the 9th International Conference, CAIP 2001. Lecture Notes in Computer Science 2124*, pages 142–151, Warsaw, Poland, 2001.

R. Cesar, E. Bengoetxea, and I. Bloch. Inexact graph matching using stochastic optimization techniques for facial feature recognition. In *International Conference on Pattern Recognition 2002, ICPR 2002*, Québec, Aug 2002a.

R. Cesar, E. Bengoetxea, I. Bloch, and P. Larrañaga. Inexact graph matching for facial feature segmentation and recognition. *International Journal of Imaging Systems and Technology*, submitted, 2002b.

R. Cesar and I. Bloch. First Results on Facial Feature Segmentation and Recognition using Graph Homomorphisms. In *VI Simpósio Ibero-Americano de Reconhecimento de Padrões SIARP 2001*, pages 95–99, Florianapolis, Brazil, Oct 2001.

R. M. Chavez and G.F. Cooper. A randomized approximation algorithm for probabilistic inference on Bayesian belief networks. *Networks*, 20(5):661–685, 1990.

H. Chen. Extracting knowledge from concept-based searching systems using conceptual graphs. Master's thesis, University of Guelph, Canada, 1996.

Y. Cheng, X. Wang, R. Collins, E. Riseman, and A. Hanson. Three-dimensional reconstruction of points and lines with unknown correspondence across images. *International Journal of Computer Vision*, 45(2):129–156, Nov 2001.

D. M. Chickering, D. Geiger, and D. Heckerman. Learning Bayesian networks is NP–hard. Technical report, Microsoft Research, Redmond, Washington, 1994.

D. M. Chickering, D. Geiger, and D. Heckerman. Learning Bayesian networks: Search methods and experimental results. In *Preliminary Papers of the Fifth International Workshop on Artificial Intelligence and Statistics*, pages 112–128, 1995.

W. J. Christmas, J. Kittler, and M. Petrou. Structural matching in computer vision using probabilistic relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):749–64, 1995.

G. F. Cooper and E. A. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

L. F. Costa and R. Cesar. *Shape Analysis and Classification: Theory and Practice*. CRC Press, 2001.

C. Cotta, E. Alba, R. Sagarna, and P. Larrañaga. Adjusting weights in artificial neural networks using evolutionary algorithms. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pages 361–377. Kluwer Academic Publishers, 2001.

A. D. J. Cross and E. R. Hancock. Graph matching with a dual-step EM algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1236–53, 1998.

A. D. J. Cross, R. Myers, and E. R. Hancock. Convergence of a hill-climbing genetic algorithm for graph matching. *Pattern Recognition*, 33(11):1863–1880, Nov 2000.

A. D. J. Cross, R. C. Wilson, and E. R. Hancock. Inexact graph matching using genetic search. *Pattern Recognition*, 30(6):953–970, 1997.

P. Dagum and E. Horvitz. A Bayesian analysis of simulation algorithms for inference in belief networks. *Networks*, 23(5):499–516, 1993.

A. P. Dawid. Conditional independence in statistical theory. *Journal of the Royal Statistics Society, Series B*, 41:1–31, 1979.

J. S. de Bonet, C. L. Isbell, and P. Viola. MIMIC: Finding optima by estimating probability densities. In *Advances in Neural Information Processing Systems*, volume 9. M. Mozer, M. Jordan and Th. Petsche eds., 1997.

L. M. de Campos. Automatic learning of graphical models. I: Basic methods. In *Probabilistic Expert System*, pages 113–140. Ediciones de la Universidad de Castilla–La Mancha, In spanish, 1998.

M. de Groot. *Optimal Statistical Decisions*. McGraw–Hill, New York, 1970.

A. P. Dempster. Covariance selection. *Biometrika*, 32:95–108, 1972.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of Royal Statistical Society*, 39(1):1–38, 1977.

C. di Ruberto and A. Dempster. Attributed skeleton graphs using mathematical morphology. *Electronics Letters*, 37(22):1325–1327, Oct 2001.

E. Dijkstra. *Cooperating Sequential Processes*. Technological University, Eindhoven, The Nederlands, 1965. Reprinted in Great Papers in Computer Science, P. Laplante (eds.), IEEE Press, 1996.

M. Doob, D. Cvetković, and H. Sachs. *Spectra of Graphs: Theory and Application*. Academic Press, 1980.

C. Dorai. *Cosmos: a Framework for Representation and Recognition of 3D Free-form Objects*. PhD thesis, Michigan State University, 1996.

B. Duc, E. Bigun, J. Bigun, G. Maitre, and S. Fischer. Fusion of audio and video information for multi modal person authentication. *Pattern Recognition Letters*, 18(9):835–843, Sep 1997.

B. Duc, S. Fischer, and J. Bigun. Face authentication with Gabor information on deformable graphs. *IEEE Transactions on Image Processing*, 8(4):504–516, Apr 1999.

L. Emami. Conceptual browser: A concept-based knowledge extraction technique. Master's thesis, University of Guelph, Canada, 1997.

L. Eroh and M. Schultz. Matching graphs. *Journal of Graph Theory*, 29(2):73–86, Oct 1998.

M. A. Eshera and K.-S. Fu. An image understanding system using attributed symbolic representation and inexact graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(5):604–618, 1986.

R. Etxeberria and P. Larrañaga. Global optimization with Bayesian networks. In *Special Session on Distributions and Evolutionary Optimization*, pages 332–339. II Symposium on Artificial Intelligence, CIMAF99, 1999.

R. Etxeberria, P. Larrañaga, and J. M. Picaza. Analysis of the behaviour of genetic algorithms when searching Bayesian networks from data. *Pattern Recognition Letters*, 18 (11–13):1269–1273, 1997a.

R. Etxeberria, P. Larrañaga, and J. M. Picaza. Reducing Bayesian networks complexity while learning from data. In *Proceedings of Causal Models and Statistical Learning*, pages 151–168. UNICOM, London, 1997b.

K. Fan, C. Liu, and Y. Wang. A randomized approach with geometric constraints to fingerprint verification. *Pattern Recognition*, 33(11):1793–1803, 2000.

K. Fan, J. Lu, and G. Chen. A feature point clustering approach to the recognition of form documents. *Pattern Recognition*, 31(9):1205–1220, Sep 1998.

P. Fariselli and R. Casadio. Prediction of disulfide connectivity in proteins. *Bioinformatics*, 17(10):957–964, Oct 2001.

S.-J. Farmer. Probabilistic graph matching. Unpublished manuscript. University of York (U.K.), 1999.

T. Feder and M.Y. Vardi. The computational structure of monotone monadic snp and constraint satisfaction: a study through datalog and group theory. *SIAM Journal on Computing*, 28(1):57–104, 1999.

R. S. Feris and R. Cesar. Detection and tracking of facial landmarks using Gabor wavelet networks. In S. Singh, N. Murshed, and W. Kropatsch, editors, *ICAPR'2001 - International Conference on Advances in Pattern Recognition. Lecture Notes in Computer Science 2013*, pages 311–320, Rio de Janeiro, Brasil, Mar 2001. Springer-Verlag Press.

M. Fernandez and G. Valiente. A graph distance metric combining maximum common subgraph and minimum common supergraph. *Pattern Recognition Letters*, 22(6-7):753–758, May 2001.

F. Fiala. Vehicle routing problems. *GMD-Mitteilungen*, 46, 1978.

A. W. Finch, R. C. Wilson, and E. R. Hancock. Matching Delaunay graphs. *Pattern Recognition*, 30(1):123–140, 1997.

A. W. Finch, R. C. Wilson, and E. R. Hancock. An energy function and continuos edit process for graph matching. *Neural Computation*, 10(7):1873–94, Oct 1998a.

A. W. Finch, R. C. Wilson, and E. R. Hancock. Symbolic graph matching with the EM algorithm. *Pattern Recognition*, 31(11):1777–1790, 1998b.

M. M. Flood. The traveling salesman problem. *Operations Research*, 4:61–75, 1956.

M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. Computers*, C-21:948–960, 1972.

L. J. Fogel. Autonomous automata. *Ind. Res.*, 4:14–19, 1962.

P. Foggia, C. Sansone, F. Tortorella, and M. Vento. Definition and validation of a distance measure between structural primitives. *Pattern Analysis and Applications*, 2(3):215–227, 1999.

E. Forgy. Cluster analysis of multivariate data: Efficiency vs. interpretability of classification. *Biometrics*, 21:768–769, 1965.

I. Foster. *Designing and Bbuilding Parallel Programs*. Addison-Wesley Publishing Company, 1995.

B. N. Freeman-Benson, J. Maloney, and A. Borning. An incremental constraint solver. *Communications of the ACM*, 33(1):54–63, 1990.

B. Freisleben and P. Merz. A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. In *International Conference on Evolutionary Computation*, pages 616–621, 1996.

A. A. Freitas and S. H. Lavington. *Mining Very Large Databases with Parallel Processing*. Kluwer Academic Publishers, London, 1999.

F. Fuchs and H. Le Men. Efficient subgraph isomorphism with 'a priori'knowledge. application to 3D reconstruction of buildings for cartography. In F.J. Ferri, J.M. Inesta, A. Amin, and P. Pudil, editors, *Advances in Pattern Recognition. Proceedings of Joint IAPR International Workshops SSPR and SPR 2000. Lecture Notes in Computer Science 1876*, pages 427–436, 2000.

R. M. Fung and K. C. Chang. Weighting and integrating evidence for stochastic simulation in Bayesian networks. In M. Henrion, R. D. Shachter, L. N. Kanal, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, volume 5, pages 209–220, Amsterdam, 1990. Elsevier.

R. M. Fung and B. del Favero. Backward simulation in Bayesian networks. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 227–234. Morgan Kaufmann Publishers, San Francisco, 1994.

M. Gangnet and B. M. Rosenberg. Constraint programming and graph algorithms. *Ann. Math. Artificial Intelligence*, 8(3–4):271–284, 1993.

S. Gao and J. Shah. Automatic recognition of interacting machining features based on minimal condition subgraph. *Computer Aided Design*, 30(9):727–739, Aug 1998.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New-York, 1979.

D. Geiger and D. Heckerman. Learning Gaussian networks. Technical report, Microsoft Advanced Technology Division, Microsoft Corporation, Seattle, Washington, 1994.

J. Geusebroek, A. Smeulders, F. Cornelissen, and H. Geerts. Segmentation of tissue architecture by distance graph matching. *Cytometry*, 35(1):11–22, Jan 1999.

S. Gold and A. Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(4):377–88, 1996.

S. Gold, A. Rangarajan, and E. Mjolsness. Learning with preknowledge: Clustering with point and graph matching distance measures. *Unsupervised Learning Foundations of Neural Computation*, 1(1):355–372, 1999. Ref: 41.

D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, Massachusetts, USA, 1989.

J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):122–128, 1986.

E. R. Hancock and J. Kittler. Edge-labeling using dictionary-based relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):165–181, 1990.

P. Hancock, V. Bruce, and M. Burton. A comparison of two computer-based face identification systems with human perceptions of faces. *Vision Research*, 38(15–16):2277–2288, Aug 1998.

G. Harik. Linkage learning via probabilistic modeling in the EcGA. Technical report, University of Illinois, Urbana, Illinois, USA, 1999. IlliGAL Report No. 99010.

G. Harik, F. G. Lobo, and D. E. Golberg. The compact genetic algorithm. In *Proceedings of the IEEE Conference on Evolutionary Computation*, pages 523–528, 1998.

K. Haris, S. Efstratiadis, N. Maglaveras, C. Pappas, J. Gourassas, and G. Louridas. Model-based morphological segmentation and labeling of coronary angiograms. *IEEE Transactions on Medical Imaging*, 18(10):1003–1015, Oct 1999.

D. Heckerman. A tutorial on learning with Bayesian networks. Technical report, MSR-TR-95-06, Microsoft Advanced Technology Division, Microsoft Corporation, Seattle, Washington, 1995.

D. Heckerman and D. Geiger. Likelihoods and parameter priors for Bayesian networks. Technical report, MSR-TR-95-54, 1995.

D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.

D. Heckerman and M. P. Wellman. Bayesian networks. *Communications of the ACM*, 38: 27–30, 1995.

M. Henrion. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. *Uncertainty in Artificial Intelligence*, 2:149–163, 1988. J. F. Lemmer and L. N. Kanal eds., North-Holland, Amsterdam.

R. Herpers and G. Sommer. Discrimination of facial regions based on dynamic grids of point representations. *International Journal of Pattern Recognition and Artificial Intelligence*, 12(4):381–405, Jun 1998.

J. H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Michigan, 1975.

J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs. In *Sixth ACM Symposium on Theory of Computing*, 1974.

R. Howard and J. Matheson. Influence diagrams. In R. Howard and J. Matheson, editors, *Readings on the Principles and Applications of Decision Analysis*, volume 2, pages 721–764. Strategic Decision Group, Menlo Park, California, 1981.

T. Hryceij. Gibbs sampling in Bayesian networks. *Artificial Intelligence*, 46(3):351–363, 1990.

C. Huang and W. Huang. Sign language recognition using model-based tracking and a 3D Hopfield neural network. *Machine Vision and Applications*, 10(5–6):292–307, Apr 1998.

B. Huet and E. R. Hancock. Shape recognition from large image libraries by inexact graph matching. *Pattern Recognition Letters*, 20(11–13):1259–1269, Nov 1999.

J. Sung Hwan. Content-based image retrieval using fuzzy multiple attribute relational graph. *IEEE International Symposium on Industrial Electronics Proceedings (ISIE 2001)*, 3:1508–1513, 2001.

P. Ifrah. Tree search and singular value decomposition: a comparison of two strategies for point-pattern matching. Master's thesis, McGill University, Canada, 1997.

H. Ing-Sheen and F. Kuo-Chin. Color image retrieval using geometric properties. *Journal of Information Science and Engineering*, 17(5):729–751, Sep 2001.

I. Inza, P. Larrañaga, R. Etxeberria, and B. Sierra. Feature subset selection by Bayesian networks based optimization. *Artificial Intelligence*, 123(1-2):157–184, 2000.

I. Inza, M. Merino, P. Larrañaga, J. Quiroga, B. Sierra, and M. Girala. Feature subset selection by genetic algorithms and estimation of distribution algorithms. A case study in the survival of cirrhotic patients treated with TIPS. *Artificial Intelligence in Medicine*, 23 (2):187–205, 2001.

C. S. Jensen, A. Kong, and U. Kjærulff. Blocking Gibbs sampling in very large probabilistic expert systems. Technical report, R 13-2031, Department of Mathematics and Computer Science, University of Aalborg, Denmark, 1993.

X. Jiang, A. Munger, and H. Bunke. On median graphs: Properties, algorithms, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10): 1144–1151, Oct 2001.

J.-M. Jolion. Stochastic pyramid revisited. *Pattern Recognition Letters*, 24(8):1035–1042, 2003.

J.-M. Jolion and W. G. Kropatsch. Graph based representations. In *First IAPR Int. Workshop on Graph Based Representations*, Vienna, 1998. Springer-Verlag. Computing Suppl. 12.

J.-M. Jolion, W. G. Kropatsch, and M. Vento. Graph based representations. In *Third IAPR Int. Workshop on Graph Based Representations*, Italy, 2001. ISBN: 88-7146-579-2.

M. Karpinski and W. Rytter. *Fast Parallel Algorithms for Graph Matching Problems: Combinatorial, Algebraic and Probabilistic Approach*. The Clarendon Press, Oxford University Press, New York, 1998.

K. Khoo and P. Suganthan. Evaluation of genetic operators and solution representations for shape recognition by genetic algorithms. *Pattern Recognition Letters*, 23(13):1589–1597, Nov 2002.

H. Kim and J. Kim. Hierarchical random graph representation of handwritten characters and its application to hangul recognition. *Pattern Recognition*, 34(2):187–201, Feb 2001.

J. Kittler, W. J. Christmas, and M. Petrou. Probabilistic relaxation for matching problems in computer vision. *IEEE Proceedings of the International Conference on Computer Vision (ICCV93)*, pages 666–673, 1993.

C. Kotropoulos, A. Tefas, and I. Pitas. Frontal face authentication using morphological elastic graph matching. *IEEE Transactions on Image Processing*, 9(4):555–560, Apr 2000a.

C. Kotropoulos, A. Tefas, and I. Pitas. Morphological elastic graph matching applied to frontal face authentication under well-controlled and real conditions. *Pattern Recognition*, 33(12):1935–1947, Dec 2000b.

J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

P. J. Krause. Learning probabilistic networks. Technical report, Philips Research Laboratories, 1998.

W. G. Kropatsch and J.-M. Jolion. Graph based representations. In *Second IAPR Int. Workshop on Graph Based Representations*, 1999. Osterreichchische Computer Gesellschaft. ISBN: 3-85403-126-2.

V. Kruger and G. Sommer. Affine real-time face tracking using a wavelet network. In *ICCV99 Workshop Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems*, pages 141–148, Corfu, Greece, 1999.

W. H. Kruskal and W. A. Wallis. Use of ranks in one-criterion variance analysis. *Journal of American Statistical Association*, 47:583–621, 1952.

S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86, 1951.

S. Kumar, M. Sallam, and D. Goldgof. Matching point features under small nonrigid motion. *Pattern Recognition*, 34(12):2353–2365, Dec 2001.

L. Lai, J. Lee, and S. Yang. Fuzzy logic as a basis for reusing task-based specifications. *International Journal of Intelligent Systems*, 14(4):331–357, Apr 1999.

P. Larrañaga, R. Etxeberria, J. A. Lozano, and J. M. Peña. Optimization in continuous domains by learning and simulation of Gaussian networks. In *Proceedings of the Workshop in Optimization by Building and using Probabilistic Models. A Workshop within the 2000 Genetic and Evolutionary Computation Conference, GECCO 2000*, pages 201–204, Las Vegas, Nevada, USA, 2000.

P. Larrañaga, C. M. H. Kuijpers, R. H. Murga, and Y. Yurramendi. Searching for the best ordering in the structure learning of Bayesian networks. *IEEE Transactions on Systems, Man and Cybernetics*, 41(4):487–493, 1996a.

P. Larrañaga, C. M. H. Kuijpers, M. Poza, and R. H. Murga. Decomposing Bayesian networks: triangulation of the moral graph with genetic algorithms. *Statistics and Computing*, 7:19–34, 1997.

P. Larrañaga and J. A. Lozano. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2001.

P. Larrañaga, J. A. Lozano, and E. Bengoetxea. Estimation of distribution algorithms based on multivariate normal and Gaussian networks. Technical Report KZZA-IK-1-01, Department of Computer Science and Artificial Intelligence, University of the Basque Country, 2001.

P. Larrañaga, R. H. Murga, M. Poza, and C. M. H. Kuijpers. Structure learning of Bayesian networks by hybrid genetic algorithms. In D. Fisher and G.-J. Lenz, editors, *Learning from Data. Artificial Intelligence and Statistics V. Lecture Notes in Statistics 172*, pages 165–174. Springer-Verlag, 1996b.

P. Larrañaga, M. Poza, Y. Yurramendi, R. H. Murga, and C. M. H. Kuijpers. Structure learning of Bayesian networks by genetic algorithms. A performance analysis of control parameters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(9): 912–926, 1996c.

S. L. Lauritzen. *Graphical Models*. Oxford University Press, 1996.

R. Lee and J. Liu. An automatic satellite interpretation of tropical cyclone patterns using elastic graph dynamic link model. *International Journal of Pattern Recognition and Artificial Intelligence*, 13(8):1251–1270, Dec 1999.

R. Lee and J. Liu. Tropical cyclone identification and tracking system using integrated neural oscillatory elastic graph matching and hybrid RBF network track mining techniques. *IEEE Transactions on Neural Networks*, 11(3):680–689, May 2000.

J. Liu. *Online Chinese Character Recognition*. PhD thesis, Chinese University of Hong Kong, 1997a.

J. Liu, K. Ma, W. Cham, and M. Chang. Two-layer assignment method for online chinese character recognition. *IEEE Proceedings on Vision Image and Signal Processing*, 147(1): 47–54, Feb 2000.

T. Liu. *A Coordinated Constraint-based Modeling and Design Advisory System for Mechanical Components and Assemblies*. PhD thesis, University of Massachusetts, 1997b.

J. Llados, E. Marti, and J. Villanueva. Symbol recognition by error-tolerant subgraph matching between region adjacency graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1137–1143, Oct 2001.

J. A. Lozano, R. Sagarna, and P. Larrañaga. Parallel estimation of distribution algorithms. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pages 129–145. Kluwer Academic Publishers, 2001.

B. Luo and E. R. Hancock. A robust eigendecomposition framework for inexact graph matching. In E. Ardizzone and V. di Gesu, editors, *Proceedings 11th International Conference on Image Analysis and Processing*, pages 465–470, 2001a.

B. Luo and E. R. Hancock. Structural graph matching using the EM algorithm and singular value decomposition. *IEEE Transactions on Pattern and Machine Intelligence*, 23(10): 1120–1136, Oct 2001b.

M. Lyons, J. Budynek, and S. Akamatsu. Automatic classification of single facial images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(12):1357–1362, Dec 1999.

L. Yang Lyul and P. Rae Hong. A surface-based approach to 3D object recognition using a mean field annealing neural network. *Pattern Recognition*, 35(2):299–316, Feb 2002.

K. Ma and T. Xiaoou. Discrete wavelet face graph matching. In *Proceedings of the International Conference on Image Processing*, volume 2, pages 217–220, Piscataway, New Jersey, USA, 2001. IEEE Signal Process. Soc.

H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, 18:50–60, 1947.

R. Mariani. Face learning using a sequence of images. *International Journal of Pattern Recognition and Artificial Intelligence*, 14(5):631–648, Aug 2000.

G. Marsaglia, K. Ananthanarayanan, and N. J. Paul. Improvements on fast methods for generating normal random variables. *Information Processing Letters*, 5:27–30, 1976.

A. Massaro and M. Pelillo. A linear complementarity approach to graph matching. In *Proc. of GbR'01, 3rd IAPR TC15 Int. Workshop on Graph based Representations*, pages 160–169, 2001. ISBN: 88 7146 579-2.

J. B. McQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

A. Mendiburu, E. Bengoetxea, and J. Miguel. Paralelización de algoritmos de estimación de distribuciones. In *XIII Jornadas de Paralelismo*, pages 37–41. In Spanish, ISBN: 64-8409-159-7, 2002.

Message Passing Interface. MPI project. MPI official web page. URL: `http://www-unix.mcs.anl.gov/mpi/` . Argonne National Laboratory, USA, 1993.

B. T. Messmer and H. Bunke. Error-correcting graph isomorphism using decision trees. *International Journal of Pattern Recognition and Artificial Intelligence*, 12(6):721–742, Sep 1998a.

B. T. Messmer and H. Bunke. A new algorithm for error-tolerant subgraph isomorphism detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):493–504, May 1998b.

B. T. Messmer and H. Bunke. A decision tree approach to graph and subgraph isomorphism detection. *Pattern Recognition*, 32:1979–1998, 1999.

B. T. Messmer and H. Bunke. Efficient subgraph isomorphism detection: a decomposition approach. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):307–323, Mar 2000.

Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer Verlag, Berlin Heidelberg, 1992.

Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.

H. Mühlenbein. The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5:303–346, 1998.

H. Mühlenbein and T. Mahning. FDA - a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation*, 7(4):353–376, 1999.

H. Mühlenbein, T. Mahning, and A. Ochoa. Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5:215–247, 1999.

H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions i. Binary parameters. In *Parallel Problem Solving from Nature - PPSN IV. Lecture Notes in Computer Science 1411*, pages 178–187, 1996.

R. Myers and E. R. Hancock. Genetic algorithms for ambiguous labelling problems. *Pattern Recognition*, 33(4):685–704, Apr 2000.

R. Myers and E. R. Hancock. Least-commitment graph matching with genetic algorithms. *Pattern Recognition*, 34(2):375–394, Feb 2001.

R. Myers, R. C. Wilson, and E. R. Hancock. Bayesian graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):628–635, Jun 2000.

R. Ng and P. Shum. Optimal clip ordering for multi-clip queries. *VLDB Journal*, 7(4): 239–252, Dec 1998.

T. Oldfield. High-resolution crystallographic map interpretation. *Acta Crystallographica*, 58 (6, Special Iss. 2):963–967, Jun 2002.

J.M. Oliver, D.J. Smith, and J.R.C. Holland. A study of permutation crossover operators on the TSP. In Lawrence Erlbaum, editor, *Genetic Algorithms and Their Applications. Proceedings of the Second International Conference*, pages 224–230, Hillsdale, New Jersey, 1987. Grefenstette, J.J. (Ed.).

OpenMP. OpenMP project. OpenMP official web page. URL: `http://www.openmp.org` . OpenMP Architecture Review Board, 1997.

E. Osman, A. R. Pearce, M. Juettner, and I. Rentschler. Reconstructing mental object representations: A machine vision approach to human visual recognition. *Spatial Vision*, 13(2–3):277–286, 2000.

M. Pantic and L. J. M. Rothkrantz. Automatic analysis of facial expressions: the state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1424–1445, 2000.

Parallel Virtual Machine. PVM project. PVM official web page. URL: `http://www.csm.ornl.gov/pvm` . Oak Ridge National Laboratory, USA, 1989.

J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

J. Pearl. Evidential reasoning using stochastic simulation of causal models. *Artificial Intelligence*, 32(2):245–257, 1987.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, Palo Alto, California, 1988.

M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian optimization algorithm. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99, Orlando FL*, volume 1, pages 525–532, San Francisco, California, 1999. Morgan Kaufmann Publishers.

M. Pelikan and H. Mühlenbein. The bivariate marginal distribution algorithm. In R. Roy, T. Furuhashi, and P. K. Chandhory, editors, *Advances in Soft Computing-Engineering Design and Manufacturing*, pages 521–535, London, 1999. Springer-Verlag.

M. Pelillo, K. Siddiqi, and S. Zucker. Continuous-based heuristics for graph and tree isomorphisms, with application to computer vision. In P. M. Pardalos, editor, *Approximation and complexity in Numerical Optimization: Continuous and Discrete Problems*. Kluwer Academic Publishers, 1999.

A. Perchant. *Morphism of Graphs with Fuzzy Attributes for the Recognition of Structural Scenes*. PhD thesis, Ecole Nationale Supérieure des Télécommunications, Paris, France, Sep 2000. In French.

A. Perchant and I. Bloch. A new definition for fuzzy attributed graph homomorphism with application to structural shape recognition in brain imaging. In *IMTC'99, 16th IEEE Instrumentation and Measurement Technology Conference*, pages 1801–1806, Venice, Italy, May 1999.

A. Perchant and I. Bloch. Graph fuzzy homomorphism interpreted as fuzzy association graphs. In *Proceedings of the International Conference on Pattern Recognition, ICPR 2000*, volume 2, pages 1046–1049, Barcelona, Spain, 2000a.

A. Perchant and I. Bloch. Semantic spatial fuzzy attribute design for graph modeling. In *8th International Conference on Information Processing and Management of Uncertainty in Knowledge based Systems IPMU 2000*, volume 3, pages 1397–1404, Madrid, Spain, Jul 2000b.

A. Perchant and I. Bloch. Fuzzy morphisms between graphs. *Fuzzy Sets and Systems*, 128 (2):149–168, 2002.

A. Perchant, C. Boeres, I. Bloch, M. Roux, and C. Ribeiro. Model-based scene recognition using graph fuzzy homomorphism solved by genetic algorithm. In *GbR'99 2nd International Workshop on Graph-Based Representations in Pattern Recognition*, pages 61–70, Castle of Haindorf, Austria, 1999.

Polaris. POLARIS project. Automatic Parallelization of Real Programs. URL: `http://polaris.cs.uiuc.edu/polaris/polaris.html` . Principal Researchers: D. A. Padua, J. Torrellas, and R. Eigenmann, 1994.

A. Rangarajan, H. Chui, and E. Mjolsness. A new distance measure for non-rigid image matching. In M. Pelillo (Eds.) E. R. Hancock, editor, *Lecture Notes in Computer Science 1654*, pages 237–252, York, UK, 1999a.

A. Rangarajan, H. Chui, and E. Mjolsness. A relationship between spline-based deformable models and weighted graphs in non-rigid matching. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages 8–14, Kauai, HI, USA, 2001. IEEE Computer Soc. Tech. Committee on Pattern Analysis and Machine Intelligence.

A. Rangarajan, A. Yuille, and E. Mjolsness. Convergence properties of the softassign quadratic assignment algorithm. *Neural Computation*, 11(6):1455–1474, Aug 1999b.

I. Rechenberg. *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. Frommann–Holzboog, Stuttgart, 1973.

J. H. Reif. *Synthesis of Parallel Algorithms*. Morgan Kaufmann, San Mateo, California, 1993.

S. W. Reyner. An analysis of a good algorithm for the subtree problem. *SIAM Journal of Computing*, 6(4):730–732, 1977.

J. T. Richardson, M. R. Palmer, G. Liepins, and M. Hilliard. Some guidelines for genetic algorithms with penalty functions. In J. D. Schafferd, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 191–197, San Mateo, California, 1989. Morgan Kaufmann.

B. D. Ripley. *Stochastic Simulation*. John Wiley and Sons, 1987.

D. Riviere, J. Mangin, D. Papadopoulos, J. Martinez, V. Frouin, and J. Regis. Automatic recognition of cortical sulci of the human brain using a congregation of neural networks. *Medical Image Analysis*, 6(2):77–92, Jun 2002.

D. Riviere, J. F. Mangin, D. Papadopoulos, J. M. Martinez, V. Frouin, and J. Regis. Automatic recognition of cortical sulci using a congregation of neural networks. In S. L. Delp,

A. M. di Gioia, and B. Jaramaz, editors, *Proceedings of Medical Image Computing and Computer-Assisted Intervention – MICCAI 2000. Third International Conference. Lecture Notes in Computer Science 1935*, pages 40–49, 2000.

V. Robles, P. de Miguel, and P. Larrañaga. Solving the travelling salesman problem with estimation of distribution algorithms. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pages 211–229. Kluwer Academic Publishers, 2001.

S. Rudlof and M. Köppen. Stochastic hill climbing by vectors of normal distributions. In *Proceedings of the First Online Workshop on Soft Computing (WSC1)*, Nagoya, Japan, 1996.

A. Sanfeliu and F. King-Sun. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(3):353–362, 1983.

A. Sanfeliu, F. Serratosa, and R. Alquezar. Clustering of attributed graphs and unsupervised synthesis of function-described graphs. In A. Sanfeliu, J.J. Villanueva, M. Vanrell, R. Alquezar, J.O. Eklundh, and Y. Aloimonos, editors, *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000. IEEE Comput. Soc*, pages 1022–1025, Los Alamitos, USA, 2000.

R. Sangüesa and U. Cortés. Learning causal networks from data: a survey and a new algorithm for recovering possibilistic causal networks. *AI Communications*, 10:31–61, 1998.

R. Sangüesa, U. Cortés, and A. Gisolfi. A parallel algorithm for building possibilistic causal networks. *International Journal of Approximate Reasoning*, 18:251–270, 1998.

R. Santana and A. Ochoa. Dealing with constraints with estimation of distribution algorithms: the univariate case. In *Second Symposium on Artificial Intelligence. Adaptive Systems. CIMAF 99*, pages 378–384, Havana, Cuba, 1999.

G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 7(2):461–464, 1978.

M. Sebag and A. Ducoulombier. Extending population-based incremental learning to continuos search spaces. In T. Bäck, G. Eiben, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of the 5th Conference on Parallel Problem Solving from Nature – PPSN V*, pages 418–427, Berlin, 1998. Springer-Verlag.

I. Servet, L. Trave-Massuyes, and D. Stern. Telephone network traffic overloading diagnosis and evolutionary techniques. In *Proceedings of the Third European Conference on Artificial Evolution (AE'97)*, pages 137–144, 1997.

R. Shachter and C. Kenley. Gaussian influence diagrams. *Management Science*, 35:527–550, 1989.

R.D. Shachter and M.A. Peot. Simulation approaches to general probabilistic inference on belief networks. In *Uncertainty in Artificial Intelligence 5*, pages 221–234. Elsevier, Amsterdam, 1990.

L. Shams, M. Brady, and S. Schaal. Graph matching vs mutual information maximization for object detection. *Neural Networks*, 14(3):345–354, Apr 2001.

L. B. Shams. *Development of Visual Shape Primitives*. PhD thesis, University of Southern California, California, USA, 1999.

Z. Shao and J. Kittler. Shape representation and recognition based on invariant unary and binary relations. *Image and Vision Computing*, 17(5-6):429–444, Apr 1999.

D. Sharvit, J. Chan, H. Tek, and B. Kimia. Symmetry-based indexing of image databases. *Journal of Visual Communication and Image Representation*, 9(4):366–380, Dec 1998.

K. Shearer, H. Bunke, and S. Venkatesh. Video indexing and similarity retrieval by largest common subgraph detection using decision trees. *Pattern Recognition*, 34(5):1075–1091, May 2001.

A. Shokoufandeh, I. Marsic, and S. Dickinson. View-based object recognition using saliency maps. *Image and Vision Computing*, 17(5-6):445–460, Apr 1999.

M. Shwe and G. Cooper. An empirical analysis of likelihood-weighting simulation on a large multiply connected medical belief network. *Comput. and Biomed. Res.*, 24:453–475, 1991.

S. Siegel. *Nonparametric statistics for the behavioral sciences*. McGraw-Hill, New York, 1956.

A. Silberschatz, P. Galvin, and G. Gagne. *Applied Operating System Concepts*. Addison-Wesley, Reading, Massachusetts, USA, 2000.

M. Singh and A. Chatterjeeand S. Chaudhury. Matching structural shape descriptions using genetic algorithms. *Pattern Recognition*, 30(9):1451–1462, 1997.

M. Skomorowski. Use of random graph parsing for scene labelling by probabilistic relaxation. *Pattern Recognition Letters*, 60:949–956, 1999.

A. E. Smith and D. M. Tate. Genetic optimization using a penalty function. In Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 499–505, San Mateo, California, 1993. Morgan Kaufmann.

P. W. F. Smith and J. Whittaker. Edge exclusion tests for graphical Gaussian models. In *Learning in Graphical Models*, pages 555–574. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

R. Sonthi. *The Definition and Recognition of Shape Features for Virtual Prototyping via Multiple Geometric Abstractions*. PhD thesis, The University of Wisconsin, Wisconsin, USA, 1997.

J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, 1984.

T. P. Speed and H. Kiiveri. Gaussian markov distributions over finite graphs. *Annals of Statistics*, 14:138–150, 1986.

W. Stallings. *Operating Systems. Internals and Design Principles*. Prentice-Hall, 5 edition, 2000.

P. Suganthan, E. Teoh, and D. Mital. Hopfield network with constraint parameter adaptation for overlapped shape recognition. *IEEE Transactions on Neural Networks*, 10(2):444–449, Mar 1999.

P. Suganthan and H. Yan. Recognition of handprinted chinese characters by constrained graph matching. *Image and Vision Computing*, 16(3):191–201, Mar 1998.

P. Suganthan, H. Yan, E. Teoh, and D. Mital. Optimal encoding of graph homomorphism energy using fuzzy information aggregation operators. *Pattern Recognition*, 31(5):623–639, May 1998.

P. N. Suganthan, E. K. Teoh, and D. P. Mital. A self-organising Hopfield network for attributed relational graph matching. *Image and Vision Computing*, 13(1):61–73, 1995.

G. Syswerda. Simulated crossover in genetic algorithms. In *Foundations of Genetic Algorithms*, volume 2, pages 239–255, San Mateo, California, 1993. Morgan Kaufmann.

A. Tefas, C. Kotropoulos, and I. Pitas. Using support vector machines to enhance the performance of elastic graph matching for frontal face authentication. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(7):735–746, Jul 2001.

A. Tefas, C. Kotropoulos, and I. Pitas. Face verification using elastic graph matching based on morphological signal decomposition. *Signal Processing*, 82(6):833–851, Jun 2002.

A. Tom and C. Murthy. Optimal task allocation in distributed systems by graph matching and state space search. *Journal of Systems and Software*, 46(1):59–75, Apr 1999.

J. Triesch and C. von der Malsburg. A system for person-independent hand posture recognition against complex backgrounds. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(12):1449–1453, Dec 2001.

M. Turner and J. Austin. Graph matching by neural relaxation. *Neural Computing and Applications*, 7(3):238–248, 1998.

S. Umeyama. An eigen decomposition approach to weighted graph matching problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):695–703, 1988.

J. T. Wang, K. Zhang, and G. Chirn. Algorithms for approximate graph matching. *Information Sciences*, 82(1-2):45–74, 1995.

P. Wang. Parallel matching of 3D articulated object recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 13(4):431–443, Jun 1999.

N. Wermuth. Model search among multiplicative models. *Biometrics*, 32:253–263, 1976.

D. Whitley and J. Kauth. GENITOR: A different genetic algorithm. In *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*, volume 2, pages 118–130, 1988.

J. Whittaker. *Graphical Models in Applied Multivariate Statistics*. John Wiley and Sons, 1990.

M. Williams, R. C. Wilson, and E. R. Hancock. Multiple graph matching with Bayesian inference. *Pattern Recognition Letters*, 18(11–13):1275–1281, Nov 1997.

M. Williams, R. C. Wilson, and E. R. Hancock. Deterministic search for relational graph matching. *Pattern Recognition*, 32(7):1255–1271, Jul 1999.

R. C. Wilson and E. R. Hancock. Bayesian compatibility model for grach matching. *Pattern Recognition Letters*, 17:263–276, 1996.

R. C. Wilson and E. R. Hancock. Structural matching by discrete relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):634–698, 1997.

R. C. Wilson and E. R. Hancock. Graph matching with hierarchical discrete relaxation. *Pattern Recognition Letters*, 20(10):1041–1052, Oct 1999.

L. Wiskott. The role of topographical constraints in face recognition. *Pattern Recognition Letters*, 20(1):89–96, Jan 1999.

M. Wolfe. *High Performance Compilers for Parallel Computing*. Addison-Wesley, 5 edition, 1996.

H. Wu, Q. Chen, and M. Yachida. Face detection from color images using a fuzzy pattern matching method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21 (6):587–503, 1999.

Y. Xiang and T. Chu. Parallel learning of belief networks in large and difficult domains. *Data Mining and Knowledge Discovery*, 3:315–339, 1999.

A. A. Zhigljavsky. *Theory of Global Random Search*. Kluwer Academic Publishers, 1991.

Citation index

'To finish a work? To finish a picture? What nonsense! To finish it means to be through with it, to kill it, to rid it of its soul, to give it its final blow the coup de grace for the painter as well as for the picture. '

Pablo Picasso