2m11.2977.8

Université de Montréal

Heuristique d'évaporation de pénalités dans une méthode de décomposition pour trouver la plus grande clique d'un graphe

par

Patrick St-Louis

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en informatique option recherche opérationnelle

Avril 2002

©Patrick St-Louis, 2002

QA
76
U54
2002
V.046

Université de Montréal

Faculté des études supérieures

Ce mémoire intitulé:

Heuristique d'évaporation de pénalités dans une méthode de décomposition pour trouver la plus grande clique d'un graphe

présenté par:

Patrick St-Louis

a été évalué par un jury composé des personnes suivantes:

Michael Florian
(président-rapporteur)

Jacques A. Ferland
(directeur de recherche)

Bernard Gendron
(co-directeur)

Michel Gendreau
(membre du jury)

Mémoire accepté le:

20 juin 2002

Sommaire

Un graphe $G = (V, E)$ est composé d'un ensemble de noeuds V et d'arêtes E tels que chaque arête relie deux noeuds dans V. Lorsque deux noeuds sont reliés ensemble par une arête, on dit qu'ils sont adjacents. Une clique est un sous-graphe dont tous les noeuds sont adjacents les uns aux autres. Le problème que nous traitons dans ce mémoire est celui de trouver la clique de plus grande taille dans un graphe (où la taille d'une clique est le nombre de noeuds qui en font partie).

Le problème d'identifier la plus grande clique d'un graphe trouve de nombreuses applications, autant pratiques que théoriques. En pratique, un graphe représente souvent des situations conflictuelles. Les noeuds représentent les entités et les arêtes représentent les conflits entre les différentes entités. Si on veut regouper des entités sans conflit, alors la taille de la plus grande clique du graphe est une borne inférieure sur le nombre de groupes qu'il faut créer. Sur le plan théorique, résoudre ce problème rapidement permettrait de créer de puissants outils d'aide à la décision, et ce dans de nombreux domaines en science.

Pour s'attaquer à ce problème, nous avons développé une méthode de décomposition et une méthode heuristique. La méthode de décomposition peut à la fois servir à améliorer l'efficacité d'une méthode exacte, ou à permettre à une méthode heuristique de mieux explorer le graphe, trouvant ainsi des cliques de taille supérieure. La méthode heuristique d'évaporation de pénalités est, d'une certaine façon, une généralisation continue du principe tabou. En effet, nous remplaçons une variable booléenne (statut tabou) par une variable continue (pénalité). Ce changement permet une flexibilité accrue de la méthode et favorise l'obtention de résultats de meilleure qualité, et ce plus rapidement qu'une méthode tabou.

Sur 80 graphes, la combinaison de nos méthodes réussit à trouver 64 fois la solution optimale, et le taux moyen de différence entre la valeur optimale et celle trouvée par notre combinaison est de 2.2%. Il est intéressant de noter que l'une des meilleures méthodes connues à ce jour trouve 51 solutions optimales parmi les 80 instances, avec un taux moyen de 2.1%. Enfin, ces résultats sont obtenus en utilisant toujours le même ensemble de paramètres précalibrés. Nous pourrions obtenir de meilleurs résultats en ajustant les paramètres pour chaque graphe en fonction de sa taille, de sa densité et de sa forme générale (compacte, allongée, à peine connexe, etc).

Mots-clés : Plus grande clique d'un graphe, Décomposition, Évaporation de pénalités, Méthodes heuristiques, Tabou

---

A graph $G = (V, E)$ consists of a set of vertices V and a set of edges E such that each edge in E connects a pair of vertices in V. When two vertices are connected by an edge, they are said to be adjacent. A clique in G is a subgraph where the vertices are all pairwise adjacent. In this paper we study the maximum clique problem which consists in finding a clique of maximum cardinality.

The maximum clique problem underlies several applications. For example, consider the problem of generating coworker groups for a particular project. To exhibit the conflicting situations, where pairs of coworkers cannot work together, generate a graph where a vertex is associated with each person, and where an edge links each pair of conflicting people. Then the size of a maximum clique in the resulting graph is a lower bound on the number of groups to create in order to avoid any conflicts. Solving this problem rapidly would allow to create powerful decision making tools in several contexts.

To solve this problem, we introduce a general decomposition method and a heuristic method. The decomposition method can both improve the efficiency of an exact method or help a heuristic method to search a graph more thoroughly, thus finding larger cliques. The penalty-evaporation heuristic method can be seen as a generalisation of the tabu

search principle. Indeed, we replace a boolean variable (the tabu status) by a continuous variable (penalty). This change increases the flexibility of the search leading to better results requiring smaller computing time.

To analyse the efficiency of our methods, we use the benchmark instances that were used in the Second DIMACS Implementation Challenge, which took place about 10 years ago. Most of the methods developped since then use these instances as a base for comparisons. The combination of our methods enables to generate an optimal solution for 64 out of the 80 tested instances, and the average percentage of difference between the best known value and the one generated with our method is 2.2%. It is worthy of note that the best known value is reached for 51 out of the 80 problems using one of the best known heuristic procedure with an average percentage of difference equal to 2.1%. These results are found using the same set of parameters for all instances. Better results can be found by ajusting parameters according to the density and shape of the graph.

Keywords : Maximum Clique, Decomposition, Penalty Evaporation, Heuristic, Tabu

Table des matières

Sommaire iii

Table des matières vi

Liste des figures viii

Liste des tableaux ix

Remerciements x

Chapitre 1

Introduction 1

Chapitre 2

Plus grande clique d'un graphe et méthodes de résolution 4

2.1 Introduction à la théorie des graphes 4

2.2 Introduction à la complexité algorithmique 5

2.3 Formulation du problème de la clique de plus grande taille 7

2.4 Principales méthodes de résolution 9

2.5 Méthodes exactes 9

2.5.1 Branch-and-bound 9

2.5.2 Familles de graphes 10

2.6 Méthodes heuristiques 10

2.6.1 Vorace ("Greedy") 10

2.6.2 Recherche par voisinage 12

2.6.3 Algorithmes génétiques 13

2.6.4 Réseaux de neurones 13

2.6.5 Formulation continue 14

2.6.6 Évaporation de Pénalités 14

2.7 Méthodes d'amélioration 17

2.7.1 Parallélisme 17

2.7.2 Décomposition 17

Chapitre 3

A Penalty-Evaporation Heuristic in a Decomposition Method for the Maximum Clique Problem **20**

Chapitre 4

Conclusion **46**

Bibliographie **49**

Liste des figures

2.1 Exemple de graphe non orienté 5

2.2 Clique de taille 5 7

2.3 Grands courants des méthodes de résolution 9

2.4 Contre-exemple pour MIN 11

Liste des tableaux

3.1 Order of magnitude of the penalty factor p 35

3.2 Pairs of penalty and evaporation factors 36

3.3 Tie-breaking criteria combinations 37

3.4 Results for the 80 DIMACS problems(1) 40

3.5 Results for the 80 DIMACS problems(2) 41

3.6 Results for the 80 DIMACS problems(3) 42

3.7 Performance in terms of the number of best solutions generated 43

3.8 Comparing the performance of the methods two by two (1) 43

3.9 Comparing the performance of the methods two by two (2) 44

3.10 Execution times, ordered from fastest to slowest 45

Remerciements

Tout d'abord, je remercie mon directeur, Jacques Ferland, et mon codirecteur, Bernard Gendron, pour l'énorme confiance qu'ils ont eu en mes idées et capacités, ainsi que leur appui incessant tout au long de la recherche et la rédaction de ce mémoire. Merci aussi à Alain Hertz et Pierre McKenzie, qui ont démontré leur intérêt envers mes idées et qui m'ont aidé à croire en moi-même.

Je tiens aussi à souligner l'appui de mes collègues du DIRO, qui ont su prêter une oreille attentive lorsque je tentais d'expliquer certains passages du mémoire, ou simplement quand j'avais besoin de support moral. En particulier, Francis Forget, Catherine Beaulieu et Dominique Tourillon, qui m'ont suivi tout au long de ma recherche et qui ont su maintenir ma bonne humeur.

Enfin, merci à mes parents pour leur soutien moral et financier, sans lequel je n'aurais jamais pu atteindre un niveau d'études aussi élevé, l'un des objectifs les plus importants de ma vie.

Chapitre 1

Introduction

Un graphe $G = (V, E)$ est composé d'un ensemble de noeuds V et d'arêtes E tels que chaque arête relie deux noeuds dans V. Lorsque deux noeuds sont reliés ensemble par une arête, on dit qu'ils sont adjacents. Une clique est un sous-graphe dont tous les noeuds sont adjacents les uns aux autres. Le problème que nous traitons dans ce mémoire est celui de trouver la clique de plus grande taille dans un graphe (où la taille d'une clique est le nombre de noeuds qui en font partie).

Le problème d'identifier la plus grande clique d'un graphe trouve de nombreuses applications, autant pratiques que théoriques. En pratique, un graphe représente souvent des situations conflictuelles. Les noeuds représentent les entités et les arêtes représentent les conflits entre les différentes entités. Si on veut regouper des entités sans conflit, alors la taille de la plus grande clique du graphe est une borne inférieure sur le nombre de groupes qu'il faut créer. Sur le plan théorique, résoudre ce problème rapidement permettrait de créer de puissants outils d'aide à la décision, et ce dans de nombreux domaines en science.

En théorie de la complexité, on a démontré que trouver la plus grande clique d'un graphe est un problème NP-difficile. Ceci implique qu'on ne connaît pas de façon de résoudre le problème en un temps proportionnel à un polynôme défini en fonction de la taille du graphe. On peut aussi dire que résoudre ce problème est aussi complexe que d'énumérer toutes les solutions possibles et de choisir la meilleure. Pour résoudre ce problème, on a donc recours à deux grands types d'approches : exacte ou heuristique. Le but premier des méthodes exactes est de réduire le temps requis par rapport à une énumération explicite des solutions pour trouver la plus grande clique du graphe. Pour y

arriver, on utilise principalement la méthode de Branch-and-bound, souvent combinée à du parallélisme. Il existe aussi plusieurs familles de graphes (ayant les mêmes propriétés) pour lesquelles on sait comment trouver la taille de la plus grande clique rapidement. Du côté des méthodes heuristiques, le domaine est tellement vaste qu'il serait inutile de tenter de le résumer en quelques lignes. Nous reférons le lecteur au chapitre 2 où nous résumons brièvement les principaux types de méthodes heuristiques pour résoudre ce problème.

La première contribution au domaine est une méthode de décomposition réduisant la recherche de cliques à des sous-graphes (de plus petite taille) tout en effectuant une recherche exhaustive dans l'ensemble du graphe. Notre méthode de décomposition permet de réduire le temps de calcul lorsqu'utilisée en conjonction avec des méthodes exactes, et améliore la qualité des solutions trouvées lorsqu'utilisée en conjonction avec des méthodes heuristiques. La seconde contribution est une méthode heuristique basée sur les principes de pénalité et d'évaporation pour trouver rapidement une grande clique dans un graphe. L'idée principale de notre méthode heuristique est de combiner une généralisation du statut tabou à une méthode vorace améliorée. La combinaison de nos deux contributions donne des résultats de meilleure qualité que ceux de la littérature que nous avons répertoriée.

Pour analyser l'efficacité de nos méthodes, nous avons utilisé les mêmes instances que lors du "Second DIMACS Implementation Challenge" [30], qui a eu lieu il y a environ 10 ans. La plupart des méthodes développées depuis lors utilisent ces mêmes instances pour se comparer entre elles. Il était naturel d'en faire autant. Sur 80 graphes, la combinaison de nos méthodes réussit à trouver 64 fois la solution optimale, et le taux moyen de différence entre la valeur optimale et celle trouvée par notre combinaison est de 2.2%. Il est intéressant de noter que l'une des meilleures méthodes connues à ce jour trouve 51 solutions optimales parmi les 80 instances, avec un taux moyen de 2.1%. Enfin, ces résultats sont obtenus en utilisant toujours le même ensemble de paramètres précalibrés. Nous pourrions obtenir de meilleurs résultats en ajustant les paramètres pour chaque graphe en fonction de sa taille, de sa densité et de sa forme générale (compacte, allongée, à peine connexe, etc).

Ce mémoire est composé des chapitres suivants. Le chapitre 2 résume le problème de plus grande clique et les idées les plus populaires pour tenter de le résoudre. En particulier, nous expliquons comment résoudre ce problème à l'aide de notre méthode de décomposition et de notre heuristique d'évaporation de pénalités. Dans le chapitre 3, nous présentons l'article en langue anglaise, soumis pour publication, intitulé "A Penalty-Evaporation Heuristic in a Decomposition Method for the Maximum Clique Problem". Cet article décrit les détails des deux méthodes et présente les résultats numériques obtenus par une implémentation de notre méthode de décomposition en conjonction avec notre méthode d'évaporation de pénalités. Enfin, nous résumons nos travaux en Conclusion et nous discutons des avenues de recherche futures.

Chapitre 2

Plus grande clique d'un graphe et méthodes de résolution

Dans ce chapitre, nous présentons dans son contexte tout ce que dont traite l'article (chapitre 3). Tout d'abord, dans la section 2.1, nous introduisons quelques éléments de la théorie des graphes. Ensuite, dans la section 2.2, nous discutons de la complexité du problème de trouver la plus grande clique d'un graphe. Puis, dans la section 2.3, nous introduisons la formulation du problème linéaire en nombres entiers. Enfin, les sections 2.4 - 2.7 présentent les grandes approches pour résoudre ce problème et introduisent nos méthodes dans leur contexte.

2.1 Introduction à la théorie des graphes

Plusieurs problèmes peuvent être formulés à l'aide d'un graphe non orienté, utilisé pour représenter des relations (arêtes) entre différentes composantes (noeuds) d'un système. Supposons, par exemple, qu'on étudie une espèce en voie d'extinction, comme le panda. On crée un noeud pour chaque panda et on relie par une arête deux pandas qui peuvent se reproduire ensemble (leurs gênes sont suffisamment différents pour qu'il n'y ait pas de risques de consanguinité). On relie aussi ensemble les pandas de même sexe qui se tolèrent (les combats peuvent mener à des blessures fatales qu'on cherche à éviter). Si on devait choisir un sous-ensemble de pandas pour les transporter dans une zone environnementale protégée, lesquels choisirait-on ? La réponse intuitive serait de choisir des pandas qui se tolèrent tous ou qui peuvent se reproduire ensemble (en fait, le problème est plus complexe, mais cette description suffit pour les prochaines

explications).

Ce problème revient à trouver, dans le graphe $G = (V, E)$ où V dénote l'ensemble des noeuds et E l'ensemble des arêtes, un sous-graphe où les noeuds sont tous interreliés les uns aux autres. Un tel sous-graphe est appelé une clique. La solution au problème des pandas correspond à trouver la plus grande clique dans le graphe, et à transporter dans la zone protégée les animaux correspondant aux noeuds de cette clique.

Trouver la plus grande clique d'un graphe est une tâche aisée pour l'intelligence humaine, à la condition qu'on travaille avec de petits graphes. Le graphe de la figure 2.1 possède deux cliques de taille maximale : les sous-graphes comportant respectivement les sous-ensembles de noeuds {1,2,4} et {1,3,4}. En effet, dans ces deux sous-graphes, les sous-ensembles de noeuds sont tous interreliés. Le sous-graphe comportant le sous-ensemble de noeuds {2,3,4} n'est pas une clique puisque le noeud 2 n'est pas relié au noeud 3.

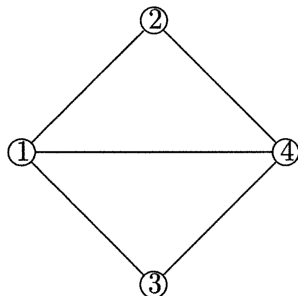

FIG. 2.1 – Exemple de graphe non orienté

2.2 Introduction à la complexité algorithmique

Pour des graphes de plus grande taille (10 noeuds ou plus), l'être humain commence à rencontrer des difficultés pour résoudre le problème de manière optimale. Malheureusement, les problèmes pratiques peuvent facilement compter des milliers de noeuds, ce qui rend la résolution manuelle de ce genre de problèmes impossible par l'être humain. On doit alors se tourner vers la résolution par un algorithme, où on suit étape par étape une séquence d'actions afin d'obtenir le résultat final qui nous intéresse.

La complexité d'un algorithme se caractérise par le nombre maximum d'étapes requises avant de terminer son exécution. La plupart du temps, le nombre d'étapes est une fonction de la taille du problème. Si le problème est basé sur un graphe non orienté, la taille du problème est normalement évaluée en termes du nombre de noeuds dans le graphe, dénoté n.

Tous les problèmes qui se résolvent à l'aide d'un algorithme ayant une complexité s'exprimant comme un polynôme en fonction de la taille du problème font partie de la famille de problèmes P. Un problème est dans NP si c'est un problème de décision (la solution du problème est "oui" ou "non"), et que vérifier le certificat de la solution (ce qui nous permet d'affirmer ou d'infirmer) est un problème dans P. Par exemple, le problème de décision "Est-ce que le nombre X est premier ?" est très difficile à résoudre en général, mais si on possède la liste des diviseurs de X, alors il est facile de répondre à la question. Dans ce cas, la liste des diviseurs est le certificat requis pour vérifier la solution en question.

Un problème est dit NP-complet si tous les problèmes dans NP peuvent se transformer en ce problème en temps polynomial. Ainsi, si on parvient à résoudre un problème NP-complet en temps polynomial, alors tous les problèmes dans NP peuvent aussi être résolus en temps polynomial. Enfin, un problème est NP-difficile si c'est un problème d'optimisation aussi difficile à résoudre qu'un problème quelconque dans NP.

L'un des premiers problèmes à être démontré NP-complet est le suivant : "Est-ce qu'il existe une clique de k noeuds dans le graphe ?" (où $k > 2$ est un nombre entier). Le problème d'optimisation (NP-difficile) correspondant est de trouver la taille de la plus grande clique du graphe.

Le graphe de la figure 2.2 est une clique de taille 5. Tout sous-ensemble de taille $k \leq 5$ forme une clique de taille k. Ainsi, si on sait que la plus grande clique d'un graphe est de taille M, alors le graphe contient au moins une clique de taille $k \leq M$, mais aucune de taille $k > M$. Par conséquent, on peut résoudre le problème NP-complet à partir du problème NP-difficile correspondant, et vice versa. En effet, si on connaît la réponse au problème de décision pour chaque k entre 2 et n, alors la réponse

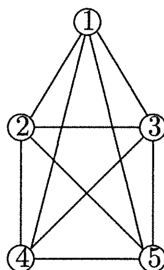

FIG. 2.2 – Clique de taille 5

au problème NP-difficile de déterminer la plus grande clique correspond au plus grand k pour lequel la réponse est "oui".

Tous les algorithmes connus pour trouver la plus grande clique d'un graphe prennent un temps au moins exponentiel par rapport à la taille du problème pour générer une solution. Toutefois, il arrive fréquemment que des problèmes pratiques n'aient pas besoin d'une solution optimale, se satisfaisant d'une simple bonne solution (une clique quelconque). De plus, la plupart des algorithmes exacts utilisent aussi des approximations pour parvenir plus rapidement à une solution optimale. L'étude des bornes inférieures et supérieures sur la plus grande clique d'un graphe est donc très importante, surtout lorsque ces approximations peuvent être trouvées en un temps polynomial par rapport à la taille du problème. Remarquons que l'étude des bornes supérieures, bien que fondamentale pour pouvoir identifier efficacement des solutions optimales, ne fournit pas de solution réalisable, contrairement à celle des bornes inférieures qui permettent d'identifier une clique dans le graphe (dans la plupart des cas).

2.3 Formulation du problème de la clique de plus grande taille

Le modèle mathématique généralement utilisé pour représenter le problème de trouver la plus grande clique d'un graphe est le suivant. Tout d'abord, on associe à chaque noeud i du graphe une variable binaire x_i qui prend la valeur 1 si le noeud fait partie de la solution courante, et 0 sinon. On s'assure ensuite que la solution courante forme

une clique en vérifiant que toutes les paires de noeuds du sous-graphe sont reliées par une arête du graphe. Pour que la condition soit vérifiée, il suffit de s'assurer que, pour toute paire de noeuds qui n'est pas reliée par une arête dans le graphe, au plus un de ceux-ci fait partie de la clique. Enfin, puisqu'on cherche à identifier la plus grande clique du graphe, l'objectif est donc d'affecter la valeur 1 au plus grand nombre de variables possible, ce qui se traduit par maximiser la somme de toutes les variables. Le modèle peut donc être exprimé ainsi :

(MCP)

$$\text{Max} \quad \sum_{i=1}^{|V|} x_i,$$

$$\text{s.t.} \quad x_i + x_j \leq 1 \quad , \forall (i,j) \notin E, \qquad (2.1)$$

$$x_i \in \{0,1\} \quad , \forall i \in V. \qquad (2.2)$$

2.4 Principales méthodes de résolution

Les différentes méthodes de résolution pour déterminer la plus grande clique d'un graphe peuvent se classer parmi les catégories du schéma suivant :

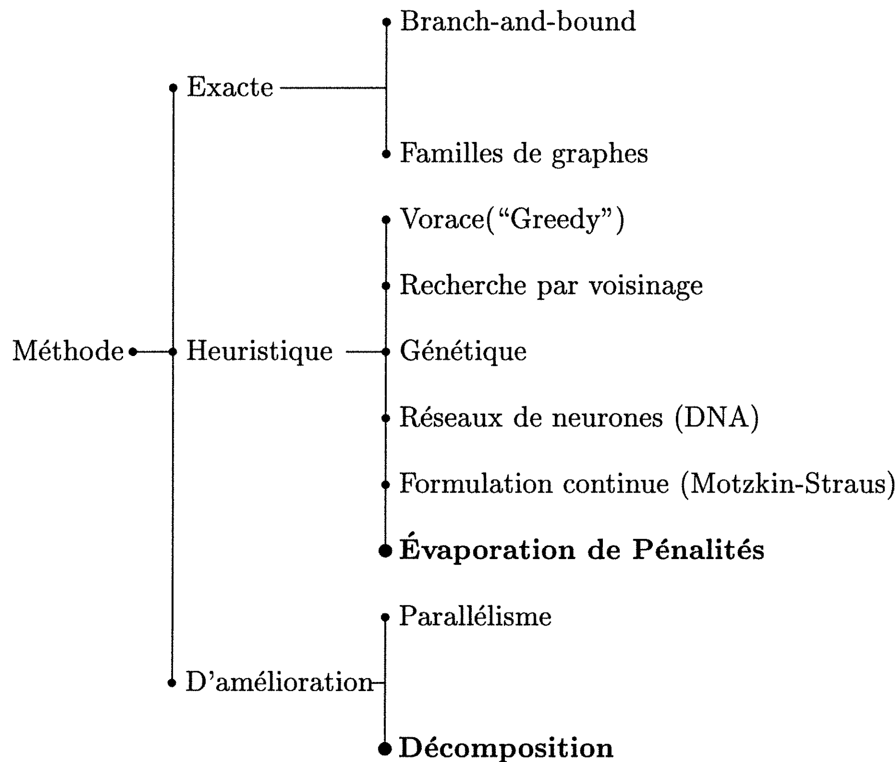

FIG. 2.3 – Grands courants des méthodes de résolution

Dans les sections suivantes, nous présentons à tour de rôle les trois grands types d'approches, soit les méthodes exactes, les méthodes heuristiques et les méthodes d'amélioration.

2.5 Méthodes exactes

2.5.1 Branch-and-bound

Lorsqu'on doit résoudre de manière exacte un problème d'optimisation, utiliser un algorithme de Branch-and-Bound est souvent la seule option disponible. Pour le

problème de trouver la plus grande clique d'un graphe, on utilise souvent des méthodes heuristiques pour déterminer des bornes à chaque itération de l'algorithme. On utilise régulièrement une méthode heuristique de coloration de graphes pour déterminer une borne supérieure, et une méthode heuristique de plus grande clique pour identifier une borne inférieure. Différents algorithmes pour résoudre le problème de plus grande clique de manière exacte se trouvent dans [3, 4, 5, 15, 19, 25, 31, 32, 38, 41, 42].

2.5.2 Familles de graphes

Pour certains graphes, il est plus facile de déterminer une clique de taille optimale. L'exemple trivial est un graphe sans arêtes où la plus grande clique ne comporte qu'un seul noeud. Il existe toute une littérature sur les différentes familles de graphes et leurs propriétés. Sachant qu'un graphe fait partie de l'une de ces familles, il est possible d'en tirer de l'information utile, voire même de résoudre le problème de manière exacte en très peu de temps. Alizadeh [2] a démontré comment calculer le nombre de Lovasz (valeur qui se situe entre la taille de la plus grande clique et le nombre chromatique d'un graphe) rapidement en utilisant une méthode de points intérieurs. Si on sait qu'un graphe est parfait (c'est-à-dire que la taille de sa plus grande clique est égale à son nombre chromatique), alors le nombre de Lovasz nous donne cette valeur en temps polynomial. On retrouve dans [7, 10, 16, 20, 22, 24, 29, 33, 35, 45] d'autres techniques exploitant des familles de graphes.

2.6 Méthodes heuristiques

2.6.1 Vorace ("Greedy")

Lorsque le temps disponible pour rechercher une solution à un problème est réduit, il serait catastrophique en général de tenter de résoudre le problème de manière exacte. On a alors recours à des heuristiques pour trouver une bonne solution en un temps beaucoup plus court. Moins un algorithme remet en question ses décisions précédentes, plus il est

rapide. Ainsi, la famille d'heuristiques la plus rapide est celle des algorithmes voraces. À chaque itération, on choisit un noeud à insérer dans la solution (clique) jusqu'à ce qu'aucun noeud dans le graphe ne puisse augmenter celle-ci. Toutefois, même si l'idée de base reste sensiblement la même, certains algorithmes voraces sont plus efficaces que d'autres. Ainsi, on utilise fréquemment MIN [28], car il est rapide et simple à programmer. À chaque itération, MIN choisit le noeud du graphe résiduel ayant le plus haut degré (nombre de noeuds qui lui sont adjacents) et l'insère dans la solution (clique). Puis, le graphe résiduel est obtenu en supprimant du graphe tous les noeuds qui ne sont pas adjacents au noeud inséré. L'algorithme s'arrête lorsque tous les noeuds ont été considérés.

En général, il est facile de trouver des contre-exemples sur lesquels une méthode vorace ne trouve pas une solution optimale. Pour éviter cela, certains auteurs ajoutent un facteur aléatoire à une telle méthode, ou bien repartent l'exécution de la méthode avec plusieurs solutions initiales différentes. Souvent, ces modifications ne font que déplacer le problème et ne garantissent généralement pas l'optimalité de la meilleure solution trouvée.

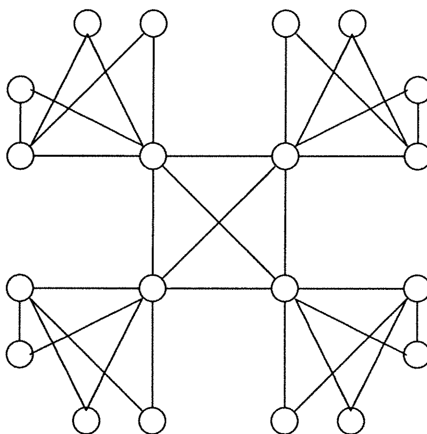

FIG. 2.4 – Contre-exemple pour MIN

Pour le graphe de la figure 2.4, peu importe le noeud choisi à la première itération pour former la clique initiale, MIN ne trouve jamais la clique de taille 4 au centre du graphe. Il est évident que, bien que les algorithmes voraces soient rapides, la qualité des solutions trouvées n'est pas toujours des plus intéressantes, et il est facile de générer des

instances pour lesquelles la performance de ces algorithmes est arbitrairement mauvaise.

2.6.2 Recherche par voisinage

La méthode tabou, le recuit simulé et la méthode de montée sont des exemples de méthodes de recherche par voisinage. Il est à noter que la méthode MIN [28] est une méthode de montée. En général, on définit le voisinage d'une solution comme étant l'ensemble des solutions qu'on peut atteindre en ne modifiant que très légèrement celle-ci, comme en changeant la valeur d'une seule variable, par exemple. Si on se réfère à la formulation (MCP), une solution est obtenue en affectant la valeur 0 ou 1 à chacune des variables (00101110 est un exemple de solution pour un graphe de 8 noeuds). Le voisinage de cette solution serait alors {10101110, 01101110, 00001110, 00111110, 00100110, 00101010, 00101100, 00101111}.

À chaque itération, une méthode de recherche par voisinage se déplace d'une solution courante à une solution voisine (réalisable ou non), tout en cherchant à améliorer la valeur de la fonction économique. Certaines de ces méthodes ont des moyens qui leur sont propres pour éviter de rester prises dans un optimum local. Par exemple, une recherche tabou maintient une liste d'informations relatives aux solutions visitées récemment pour ne pas y retourner et ainsi éviter de cycler (visiter sans arrêt un même sous-ensemble de solutions). Soriano et Gendreau [43] ont développé trois façons différentes de diversifier (changer la région du domaine réalisable où se poursuit la recherche) la recherche d'une méthode tabou, dont l'une d'elles est basée sur des calculs probabilistes. Battiti et Protasi [8] ont proposé une recherche tabou "réactive" (qui change de comportement selon l'état de la recherche) dont l'exécution redémarre périodiquement à différents endroits dans le graphe. Le choix d'un nouveau noeud de départ est basé sur l'analyse des noeuds de départ précédents. Enfin, Fujisawa et Kubo [23] ont introduit une méthode appelée "life span", inspirée d'une recherche tabou, pour tenter de contourner certains aspects négatifs des recherches tabou habituelles.

2.6.3 Algorithmes génétiques

Contrairement aux algorithmes précédents, plusieurs approches travaillent avec une population de solutions plutôt qu'une seule, et tentent d'améliorer la qualité globale de la population. Pour y arriver, ils appliquent généralement des modifications similaires aux phénomènes observés en génétique (mutation, croisement, etc). Malgré la nouveauté de l'approche, la littérature contient déjà plusieurs méthodes de type génétique pour trouver la plus grande clique d'un graphe. Toutefois, on remarque que les algorithmes basés uniquement sur la génétique ne sont pas compétitifs avec les méthodes de recherche par voisinage, alors qu'on obtient souvent d'excellents résultats en combinant les deux approches pour produire des méthodes hybrides. Bui et Eppley [17] ont introduit l'une de ces méthodes hybrides utilisant des optimisations locales. Marchiori [34] a démontré que la force d'un algorithme hybride n'est pas nécessairement liée à la force de chacune de ses composantes, mais plutôt à leur niveau de collaboration. En effet, la combinaison d'une “fitness function” simple et d'une optimisation locale a donné lieu à un excellent algorithme génétique pour trouver la plus grande clique d'un graphe. Aussi, Aggarwal, Orlin et Tai [1] ont démontré comment inclure une procédure d'affectation bipartite qui trouve une clique maximale dans l'union de deux clique à l'intérieur d'un opérateur de croisement optimisé. L'idée a inspiré Balas et Niehaus [6] qui ont étudié des variations de sélection, de remplacement de population et de mutation, et ont ainsi produit un algorithme génétique à état stable efficace.

2.6.4 Réseaux de neurones

Relativement récents dans le domaine, des algorithmes basés sur des réseaux de neurones simulent les propriétés du cerveau humain (adaptation, approximation, reconnaissance de patrons) pour résoudre des problèmes difficiles. La plus grande clique d'un graphe peut être approximée en construisant une suite de réseaux de Hopfield, comme le démontrent Bertoni, Campadelli et Grossi [9]. À chaque itération, la fonction de Lyapunov du réseau précédent est utilisée et on lui ajoute une pénalité associée aux contraintes violées. En 1995, Pelillo [39] a démontré comment résoudre le problème de la

plus grande clique d'un graphe à l'aide d'un réseau d'étiquettage de relaxation ("relaxation labelling network"), une méthode souvent utilisée dans l'étude de la reconnaissance de formes. Cette formulation est basée sur un résultat de Motzkin et Straus [36], l'une des formulations continues les plus populaires (voir la section suivante). Bomze, Budinich, Pelillo et Rossi [12] ont approfondi l'approche de Motzkin-Straus pour en faire un algorithme de recherche locale basée sur la dynamique de réplication. Les mêmes idées ont aussi été appliquées à la version du problème de clique avec poids sur les noeuds [13], ainsi que sur le problème d'isomorphisme de graphe [40].

2.6.5 Formulation continue

La formulation de Motzkin-Straus [36], $\{\max x^T Ax \mid x^T e = 1 \ , \ x \geq 0\}$ où A est la matrice d'adjacence et où la taille de la plus grande clique est $\frac{1}{1-x^T Ax}$, est généralement utilisée pour résoudre la forme continue du problème de trouver la plus grande clique d'un graphe. Inspirés par cette formulation, Gibbons, Hearn et Pardalos [26] ont développé une formulation semblable pour obtenir d'excellents résultats. Gibbons, Hearn, Pardalos et Ramana [27] ont défini un ensemble de caractérisations de la formulation de Motzkin-Straus. Ils en dérivent des critères d'optimalité de premier et second ordre, d'optimalité locale, ainsi que des caractérisations d'optimalité locale stricte. L'une des implémentations les plus performantes est introduite par Busygin [18], qui obtient d'excellents résultats en limitant la recherche à un domaine sphérique à l'intérieur de l'hypercube unitaire.

2.6.6 Évaporation de Pénalités

La méthode d'évaporation de pénalités est l'une des contributions que nous proposons pour le problème de déterminer la plus grande clique d'un graphe. L'idée originale de cette approche part d'un principe similaire à celui utilisé dans la recherche tabou. Rappelons qu'un algorithme tabou progresse d'itération en itération, cherchant la meilleure solution dans un voisinage prédéterminé, en attribuant un statut "tabou" à certaines caractéristiques des solutions récemment rencontrées afin d'éviter de les re-

trouver dans les solutions générées au cours des prochaines itérations. Une fois écoulé un certain nombre d'itérations, ces caractéristiques perdent leur statut tabou et peuvent se retrouver à nouveau dans les solutions générées. On peut s'imaginer sur des sentiers pédestres en forêt où on veut éviter les sentiers qu'on vient d'emprunter récemment pour une durée prédéterminée.

Le but de la méthode d'évaporation de pénalités est de remplacer ce changement brusque de statut tabou par un mécanisme qui soit graduel. Pour y arriver, l'algorithme ajoute une pénalité (valeur réelle donnée en paramètre au début de la résolution) sur son parcours. Plus une portion du parcours est pénalisée, moins elle est intéressante à emprunter. En forêt, ceci correspond à laisser des pierres pointues sur les sentiers où on passe. Ces sentiers restent utilisables, mais deviennent de moins en moins intéressants si on y passe régulièrement. Par contre, l'algorithme réduit à chaque itération une portion de la pénalité sur tous les parcours. Nous appelons ce processus l'"évaporation", et la quantité de pénalité évaporée est une valeur réelle donnée en paramètre au début de la résolution. Ainsi, une portion du parcours qui n'a pas été empruntée depuis maintes itérations redevient progressivement intéressante à emprunter. Ce processus d'évaporation peut être comparé à la situation d'individus qui parcourent les sentiers d'une forêt pour en retirer graduellement les roches les plus pointues, redonnant de l'intérêt aux sentiers ainsi nettoyés.

Il est possible de simuler un statut tabou à l'aide de l'évaporation de pénalités en utilisant une très grande valeur de pénalité et une grande valeur d'évaporation. Ceci revient à laisser, sur les sentiers parcourus, des roches tellement grosses qu'une seule d'entre elles suffit à bloquer le sentier en question. Un sentier redevient donc intéressant uniquement quand toutes les pierres sont retirées. Par exemple, pour simuler un algorithme tabou avec une liste de longueur 10, il suffit de donner une valeur de 10 "grosses pierres" à la pénalité et de l'évaporer à un taux de 1 "grosse pierre" par itération. Il est difficile d'expliquer l'évaporation de pénalités en donnant des valeurs quantitatives plus précises, car chaque problème possède sa propre définition de ce qu'est une "grosse pierre". Toutefois, il suffit d'effectuer quelques tests de calibrage pour arriver à trouver les paramètres optimaux.

L'approche par évaporation de pénalités a été utilisée avec succès dans le contexte de la coloration de graphes par Blöchliger [11] (les résultats sont compétitifs avec les meilleurs algorithmes connus). Dans le contexte de trouver la plus grande clique d'un graphe, nous commençons par trouver une clique arbitraire initiale dans le graphe (choisir un seul noeud est valide, puisqu'il s'agit d'une clique de taille 1). Ensuite, nous cherchons à en augmenter la taille en y insérant un noeud supplémentaire. Toutefois, nous voulons qu'à chaque itération, la solution courante conserve la propriété de clique. Ceci implique qu'à chaque fois que nous insérons un nouveau noeud dans la solution courante, nous en retirons tous les noeuds qui ne lui sont pas adjacents. Nous augmentons alors la pénalité des noeuds qui sont retirés afin de les rendre moins intéressants. Un noeud est intéressant à insérer si une grande quantité de noeuds dans la solution courante lui sont adjacents. Dans le meilleur des cas, lorsque le noeud inséré est adjacent à tous les noeuds de la clique, cette dernière augmente sa taille d'une unité. À partir de ces observations, il est possible d'associer à chaque noeud une valeur, représentant l'avantage de le choisir pour être inséré dans la clique courante, définie comme suit :

$$V(i) = \bar{\delta}(i) - P(i)$$

où $V(i)$ est la valeur du noeud i, $\bar{\delta}(i)$ est le nombre de noeuds dans la clique courante qui lui sont adjacents, et $P(i)$ est la pénalité qui lui est associée.

Ainsi, un noeud i a une grande valeur $V(i)$ si $\bar{\delta}(i)$ est élevé et $P(i)$ est faible. Toutefois, il arrive que plus d'un noeud possède la même valeur. Pour choisir parmi ceux-ci, nous avons utilisé les critères supplémentaires suivants :

- le plus grand degré (nombre de noeuds adjacents) ;
- le plus petit (grand) degré courant (nombre de noeuds adjacents dans la solution courante) ;
- la plus petite (grande) fréquence (nombre de fois qu'un noeud est sélectionné pour entrer dans la solution).

L'efficacité de notre approche repose d'abord sur sa vitesse d'exécution comparable à celle d'un algorithme vorace. On peut donc l'utiliser dans un algorithme de Branch-and-bound pour déterminer une borne inférieure sans pour autant en ralentir la vitesse

d'exécution. Sa seconde force est son mécanisme d'évaporation de pénalités similaire à un statut tabou. Les résultats présentés dans notre article indiquent l'efficacité de notre approche : la qualité des solutions est comparable à celles des meilleurs algorithmes connus et ces solutions sont souvent obtenues beaucoup plus rapidement.

2.7 Méthodes d'amélioration

2.7.1 Parallélisme

Le parallélisme est un aspect souvent mal exploité pour améliorer la vitesse d'algorithmes pour trouver la plus grande clique d'un graphe. Comme son nom l'indique, le parallélisme est une technique permettant d'effectuer en même temps plusieurs calculs indépendants, et ainsi gagner un précieux temps de calcul (voir [37], par exemple). La plupart des algorithmes (exacts ou heuristiques) contiennent une part de calculs indépendants (par exemple, pour calculer le degré de chaque noeud). On pourrait aussi penser résoudre en même temps plusieurs problèmes associés aux feuilles dans un algorithme de Branch-and-Bound, et ainsi rendre son utilisation encore plus efficace en pratique.

2.7.2 Décomposition

Généralement, on peut réduire la difficulté d'un problème en appliquant un procédé *diviser-pour-régner*. Une méthode de décomposition divise le problème en sous-problèmes plus faciles à résoudre, puis recombine les informations pour obtenir une solution globale. Dans le contexte du problème de trouver la plus grande clique, on peut réduire le problème en une série de résolutions sur des sous-graphes (évidemment de taille plus petite que le graphe original), et en extraire une solution pour le graphe entier.

Dans ce mémoire, nous proposons une approche de décomposition pouvant s'appliquer autant en conjonction avec une méthode exacte qu'avec une méthode heuristique,

pour aider ces dernières à trouver la plus grande clique d'un graphe plus efficacement. Le but de notre méthode de décomposition est de réduire progressivement la taille du graphe original en éliminant itérativement des cliques jusqu'à ce que tous les noeuds du graphe soient supprimés. Avant de supprimer une clique, on peut vérifier à l'aide d'une méthode exacte que ses noeuds ne font pas partie d'une clique de taille supérieure. Si le processus de vérification permet d'identifier une telle clique, celle-ci remplace la clique courante. Le processus de vérification s'arrête lorsque tous les noeuds de la clique ont été considérés pour tenter d'identifier une clique de taille supérieure.

L'avantage principal de notre méthode est de réduire la taille des graphes sur lesquels on exécute une méthode exacte. Ceci dit, le gain d'efficacité diminue avec la densité du graphe, mais augmente avec sa taille. Ainsi, plus la taille du graphe est grande ou moins le graphe est dense, plus notre méthode est efficace comparativement à l'utilisation de la méthode exacte à elle seule.

Si notre méthode de décomposition améliore grandement l'efficacité des méthodes exactes, elle peut également améliorer la robustesse des méthodes heuristiques et assurer une fouille plus exhaustive de tout le graphe. Notre étude a porté sur l'utilisation de la décomposition en conjonction avec des méthodes heuristiques pour vérifier si l'amélioration de la qualité des solutions générées justifiait le temps de calcul supplémentaire. En effet, si notre méthode réduit le temps de calcul pour les méthodes exactes, elle a généralement l'effet inverse pour les méthodes heuristiques puisque ces dernières sont généralement très rapides. Nos résultats indiquent qu'il est en effet profitable d'utiliser notre méthode de décomposition, peu importe l'efficacité de la méthode heuristique en conjonction avec laquelle elle est utilisée.

Lorsqu'on utilise une méthode heuristique (à la place d'une méthode exacte) en conjonction avec notre méthode de décomposition pour vérifier si les noeuds de la clique font partie d'une clique de plus grande taille, on n'est jamais certain que les noeuds de la clique supprimée ne font pas partie d'une clique de plus grande taille, voire même optimale. Plutôt que de tenter d'éviter cet effet, nous avons ajouté une étape supplémentaire permettant de reconsidérer à une itération quelconque les noeuds qui

ont été supprimés précédemment. Ainsi, nous pouvons reconstruire une clique optimale à partir de noeuds du graphe résiduel et de noeuds supprimés.

La force principale de notre méthode de décomposition est sa capacité d'améliorer la qualité des résultats obtenus avec une méthode heuristique quelconque. Les résultats que nous présentons dans notre article indiquent bien qu'appliquer la décomposition en conjonction avec MIN améliore sa capacité de trouver une solution optimale (sur 80 graphes, le nombre de solutions optimales trouvées est passée de 18 à 29). Le même taux d'amélioration est observé lorsqu'elle est utilisée en conjonction avec notre algorithme d'évaporation de pénalités (sur les mêmes graphes, le nombre de solutions optimales trouvées est passée de 43 à 64).

Chapitre 3

A Penalty-Evaporation Heuristic in a Decomposition Method for the Maximum Clique Problem

by

Patrick St-Louis

Jacques A. Ferland

Bernard Gendron

Département d'informatique et de recherche opérationnelle

Université de Montréal

P.O. box 6128, Succursale Centre-Ville

Montréal, Québec H3C 3J7

April, 2002

Abstract

In this paper, we introduce two approaches to solve the maximum unweighted clique problem. The first is a general decomposition method restricting the search for a maximum clique to subgraphs, but also performing an exhaustive search of the feasible domain. Any algorithm, exact or heuristic, that provides feasible cliques can be used to search the subgraphs. If the embedded algorithm is exact, then the decomposition algorithm provides an exact solution. If the embedded algorithm is a heuristic method, the decomposition algorithm improves significantly the quality of the solution found by the heuristic method alone. The second approach is a greedy-like heuristic method, based on the concepts of penalty and evaporation, which identifies a large clique in a graph in very short computing time. Numerical results indicate that the heuristic method alone is very strong and reliable, and the gain in quality obtained when embedding it in the decomposition algorithm is worthy of the additional computing time required.

1. Introduction

A graph $G = (V, E)$ consists of a set of vertices V and a set of edges E such that each edge in E connects a pair of vertices in V. When two vertices are connected by an edge, they are said to be adjacent. A clique in G is a subgraph where the vertices are all pairwise adjacent. In this paper we study the maximum clique problem which consists in finding a clique of maximum cardinality.

The maximum clique problem underlies several applications. For example, consider the problem of generating coworker groups for a particular project. To exhibit the conflicting situations, where pairs of coworkers cannot work together, generate a graph where a vertex is associated with each person, and where an edge links each pair of conflicting people. Then the size of a maximum clique in the resulting graph is a lower bound on the number of groups to create in order to avoid any conflicts. Several other applications and a complete recent literature review on the maximum clique problem are presented in [14].

It is well-known that the maximum clique problem is NP-hard. Hence there is no currently known polynomial algorithm to solve it exactly. There exists two main families of methods : exact methods that are often very slow but eventually find the optimal solution for small problems, and heuristic methods that can usually find very good solutions in reasonable time. The exact methods are mostly branch-and-bound algorithms (see [3, 4, 5, 15, 19, 25, 31, 32, 38, 41, 42] for some examples). Hence, they rely on lower and upper bounds which are usually computed using heuristic methods. It is possible to reduce the execution time of an exact algorithm by using parallel computing (see [37] for an example).

In contrast with exact methods that search extensively through all the feasible domain, a heuristic method can be seen as a strategy to search only in promising parts of it. In general, the more exhaustive is the search, the better the quality of the solution found is. In this paper, we introduce a general decomposition method restricting the search for a maximum clique to subgraphs, but also performing an exhaustive search of the feasible domain. Any algorithm, exact or heuristic, that provides feasible cliques

can be used to search the subgraphs. If the embedded algorithm is exact, then the decomposition algorithm provides an exact solution. If the embedded algorithm is a heuristic method, the decomposition algorithm improves significantly the quality of the solution found by the heuristic method alone. We also introduce a greedy-like heuristic method, based on the concepts of penalty and evaporation, which identifies a large clique in a graph in very short computing time. Numerical results indicate that the heuristic method alone is very strong and reliable, and the gain in quality obtained when embedding it in the decomposition algorithm is worthy of the additional computing time required.

We are using the DIMACS benchmark problems (available at `ftp ://dimacs.rutgers.edu/pub/challenge/`) to analyse the effectiveness of our algorithm. The numerical results indicate that our algorithm is competitive with the best heuristic methods. Indeed, the best known value is reached for 64 out of the 80 problems, and the average percentage difference between the best known value and the one generated with our method is 2.2% (note that the best known value is reached for 51 out of the 80 problems using one of the best known heuristic procedure [18] with an average percentage difference equal to 2.1%). These results are found using the same set of parameters for all instances. Better results can be found by ajusting parameters according to the density and shape of the graph. Finally, it is worthy to note that the penalty-evaporation approach has been also used to solve the graph coloring problem, and that the numerical results in [11] indicate that the approach is competitive with the best known heuristic methods for this problem.

The paper is organized as follows. In Section 2, we present a review of the relevant literature. In Section 3, the decomposition algorithm is introduced. Then, we give a detailed description of the penalty-evaporation algorithm in Section 4. Experimental results for the DIMACS benchmark problems are presented and analyzed in Section 5. In particular, we indicate how we calibrate the penalty and evaporation parameters of the approach, and we compare the results generated with this calibrated version to some of the best known heuristics. Finally, Section 6 includes concluding remarks and future research avenues.

The maximum clique problem can be formulated as follows [14] :

(MCP)

$$\text{Max} \quad \sum_{i=1}^{|V|} x_i,$$

$$\text{s.t.} \quad x_i + x_j \leq 1 \quad , \forall (i,j) \notin E, \tag{2.1}$$

$$x_i \in \{0,1\} \quad , \forall i \in V. \tag{2.2}$$

In this formulation, $x_i = 1$ if and only if vertex i belongs to the clique (solution). Hence, due to constraints (2.1), two vertices i and j cannot be in the same solution if they are not adjacent. Furthermore, since the objective function corresponds to the number of vertices in the solution, then the optimal solution is a clique of maximum cardinality.

The continuous relaxation of this problem usually generates non-integer optimal solutions. For example, the solution $x_i = 0.5$, $\forall i \in V$ is a feasible solution for the relaxation, and the objective value is $\frac{|V|}{2}$. Therefore, for every graph having a maximum clique of size less than $\frac{|V|}{2}$, the relaxation of the maximum clique has a non-integer optimal solution.

The feasible domain of problem (MCP) is the set of all cliques in the graph G. Since the size of this set is exponentially large, an enumeration procedure is not practical. Therefore, heuristic procedures are often used to provide near-optimal solutions without exploring every solution of the feasible domain. Their main drawback is that they may be unable to move away from a local maximum to search for a global maximum.

The first popular heuristic solvers were called greedy sequential algorithms. At each iteration, the algorithm chooses a vertex to insert in the solution (clique), until no vertex left in the graph can improve the size of the current solution. One of the most straightforward greedy heuristic procedure, usually called MIN [28], consists in selecting from the remaining subgraph the highest degree vertex, and deleting each vertex that is not adjacent to it. The algorithm stops when every vertex of the graph has been

selected or deleted. When the algorithm stops, the set of selected vertices belongs to a large clique of the original graph.

These greedy algorithms are very fast, but lack the ability to explore the entire graph, thus more often than not missing the optimal solution. To avoid this unwanted characteristic, one would run the greedy algorithm on many different parts of the graph (the term diversification is used to describe similar behavior). Since then, most heuristic methods developed their own adapted diversification process for the same reasons.

Aside from greedy algorithms, several heuristic methods (simulated annealing, tabu search, ascent) share the same basic approach : at each iteration, they move from a current solution (feasible or not) to a nearby solution, trying to reach a feasible solution of better value. For example, the tabu search keeps a history of the last visited solutions in order to avoid cycling (visiting over and over the same subset of solutions). As it happens for most heuristics, the tabu search method can reach a local maximum from which it is unable to escape. Trying to avoid this phenomenon, Soriano and Gendreau [43] studied three different diversification processes, one of them based on probabilistic calculations. Battiti and Protasi [8] proposed a reactive tabu search which periodically restarts its search in different portions of the graph. The selection of the new starting point is affected by an analysis of the history of the previous searches. Another tabu inspired technique, the life span method introduced by Fujisawa and Kubo [23], tries to resolve some of the negative aspects of the standard tabu search method. The name of the method is inspired by their way of evaluating the tabu status which does not require a list of tabu moves. Instead, they keep an array of integers in which they store the number of iterations a vertex is not allowed to be moved (selected).

Other methods use a pool of solutions which they modify using nature-inspired approaches in order to improve the quality of the solutions. Even though these methods are quite recent, the literature contains a surprising amount of different approaches applied mostly in the genetic context. Usually, pure genetic algorithms can hardly compete against local search heuristics, which leads to the development of hybrid genetic algorithms trying to combine the strengths of both approaches. Bui and Eppley [17]

introduced a hybrid genetic algorithm using local optimizations and vertex reordering. Marchiori [34] demonstrated that the strength of a genetic algorithm is not necessarily based on the strengths of each of its components, but rather on their good collaboration. Indeed, the author produced a very good genetic algorithm that combines a simple fitness function and a local optimizer to find maximal cliques. Also, Aggarwal, Orlin and Tai [1] showed how to include a bipartite matching procedure that finds a maximum clique in the union of two cliques into an optimized crossover operation. The idea inspired Balas and Niehaus [6] who studied some variations of selection, population replacement and mutation and produced an effective steady-state genetic algorithm.

Neural network algorithms have also emerged recently, using properties from the human brain (adaptation, approximation, pattern recognition) to solve many difficult problems. The maximum clique problem can be approximated by building a sequence of discrete Hopfield networks, as in Bertoni, Campadelli and Grossi [9]. At each iteration, the Lyapunov function of the previous network is used and a penalty value based on the number of violated constraints is added. In 1995, Pelillo [39] demonstrated how to approximate the maximum clique problem on a relaxation labeling network, a popular method used in pattern recognition. This formulation is based on results found by Motzkin and Straus [36], which is currently one of the most used formulation in recent developments (not only in neural networks). Bomze, Budinich, Pelillo and Rossi [12] developed the Motzkin-Straus approach further in a local search algorithm based on replicator dynamics. The same ideas have been applied successfully on the maximum weighted clique problem [13] and on the graph isomorphism problem [40].

The Motzkin-Straus formulation is mostly used in the continuous form of the integer programming problem. Inspired by this strong formulation, Gibbons, Hearn and Pardalos [26] developed a similar continuous formulation with very good results. Gibbons, Hearn, Pardalos and Ramana [27] also define a set of characterizations for the Motzkin-Straus formulation. They derived first and second order optimality, local optimality and strict local characterizations, and demonstrated their verification in polynomial time. One of the most successful implementations of continuous-based heuristics was introduced by Busygin [18], who obtained very good results by restraining the feasible domain

to a spheral form inside the standard unit hypercube.

Finally, some emerging heuristic methods find and exploit classes of graph structures for which they can find the largest clique in polynomial time. In the early 90's, Alizadeh [2] demonstrated how to compute the Lovasz's number in sublinear parallel time using an interior point method. Since it is known that the Lovasz's number gives the size of the largest clique in perfect graphs, this implies that the size of the largest clique can be found in sublinear parallel time for perfect graphs. Techniques exploiting graph structures can be found in [7, 10, 16, 20, 22, 24, 29, 33, 35, 45].

3. *The Decomposition Algorithm*

The decomposition algorithm is an iterative procedure. At each iteration, we determine a large clique of the graph (included in a residual graph), we verify that its vertices are not part of a larger clique, and then we remove the clique from the graph. The algorithm stops when the residual graph is empty.

Let G be the graph and let G' be the current residual graph after some vertices have been deleted. Let $G'(i)$ (the neighborhood of vertex i in the graph G') be the subgraph of G' induced by vertex i and every vertex in G' that is adjacent to vertex i. Finally, let BC denote the largest clique found by the algorithm so far. The algorithm can be summarized as follows :

$G' = G$, $BC = \phi$

While G' is not empty, iterate :

- 1 : Choose an arbitrary clique C in G'
- 2 : For each vertex i of C, iterate :
  - 2.1 : Find the largest clique C' in $G'(i)$
  - 2.2 : If $|C'| > |C|$ then $C = C'$ and start over at 2
- 3 : If $|C| > |BC|$ then $BC = C$
- 4 : $G' = G' - C$

Theorem : If the algorithm used in Step 2 to generate the largest clique C' in $G'(i)$ is exact (i.e. the algorithm generates an optimal solution), then the decomposition

algorithm generates an optimal solution.

Proof : Let C^* be an optimal clique (solution), and denote by C^k the clique generated by the algorithm once Step 3 is reached at the end of the kth iteration.

Now, assume that at the end of iteration $(k-1)$, the current best clique BC is such that $|BC| < |C^*|$ (i.e. an optimal clique is not identified so far). At iteration k, any vertex i in G' can belong to C^* and C^k only if $|C^k| = |C^*|$ because we are using an exact algorithm to determine C^k in Step 2.1. Hence, if $|C^k| = |C^*|$, then the proof is completed. Otherwise, if $|C^k| \neq |C^*|$, then all vertices in C^k are removed from G' since they cannot belong to C^*.

The argument is repeated inductively until for some iteration k, $|C^k| = |C^*|$. Indeed, at some iteration k, some i in C^* will also belong to C^k (since the algorithm stops when G' is empty). Since none of the other vertices in C^* has been removed from G at the preceding iterations, then the largest clique C' in $G'(i)$ determined with the exact algorithm will be such that $|C'| = |C^*|$. Hence, at the end of iteration k, the current best clique C^k will be such that $|C^k| = |C'| = |C^*|$.

Note that any procedure can be used to determine an arbitrary clique C in G' in Step 1. This clique C can even reduce to any vertex in G'. Using an exact algorithm in Step 2 (to guarantee the optimality of the solution generated with the decomposition algorithm) is an appropriate strategy to deal with large sparse graphs, since it is used to determine the largest clique in small subgraphs.

Now, to deal with large dense graphs, it would be interesting to use the decomposition algorithm with a heuristic method used in Step 2 instead of an exact algorithm. Of course, the preceding theorem does not hold to guarantee the optimality of the clique generated. Indeed, at the end of Step 2 during iteration k, the large clique C^k may include vertices belonging to a larger clique in G'. Thus, removing vertices in C^k may induce removing vertices belonging to an optimal clique and, hence, losing the opportunity of finding this solution in subsequent iterations. To prevent, at least partly, this situation, an additional Step is included at the end of Step 2 of the preceding

decomposition algorithm.

This additional Step includes a verification process allowing to consider vertices that might have been removed in previous iterations or vertices that were not considered by the heuristic procedure used in Step 2.1. The new variant of the decomposition algorithm is summarized as follows :

$G' = G$, $BC = \phi$

While G' is not empty, iterate :

1 : Choose an arbitrary clique C in G'

2 : For each vertex i of C, iterate :

2.1 : Find the largest clique C' in $G'(i)$(using a heuristic method)

2.2 : If $|C'| > |C|$ then $C = C'$ and start over at 2

3 : Let S be the subset of vertices in G that are adjacent to every vertex in C

3.1 : Find the largest clique $C_S \in S$ (using a heuristic or an exact method)

3.2 : $C = C + C_S$

4 : If $|C| > |BC|$ then $BC = C$

5 : $G' = G' - C$

Note that in the additional Step 3, since each vertex in S is adjacent to all vertices in C, the clique can be augmented by using any vertex in S. But to maintain the clique property, whenever several vertices in S are added to C, they must be adjacent to each other. Hence only a clique in S can be added to C, and then the largest clique in S is added to C in order to induce a better solution.

If the heuristic method used in Step 2.1 is effective, then the number of vertices in S should be small. Thus an exact algorithm could be used in Step 3.1 to provide the largest increase of the size of the clique generated.

Finally, it is easy to see that embedding a heuristic method in the decomposition algorithm should, in general, generate better solutions than using the heuristic method alone. Indeed, the decomposition algorithm searches more extensively the entire feasible domain since all parts of the graph are considered. Furthermore the search in any local

part of the graph is greatly intensified (in Steps 2 and 3), thus reducing the risk of missing an optimal solution.

4. *The Penalty-Evaporation Procedure*

The motivation for developing our penalty-evaporation (P-E) algorithm is to provide a greedy-like algorithm that could search a larger portion of the graph than most standard greedy algorithms do, but retaining their computational effectiveness. The numerical results indicate that the solutions generated with our calibrated version of the algorithm are very good (usually much better than MIN), but not always optimal.

The P-E algorithm is a greedy-like algorithm because the current solution is always feasible, and because vertices are added iteratively to the solution. The removal process of our algorithm allows the algorithm to actually "move" from one part of the graph to another. The basic steps of the algorithm are summarized as follows :

1. Find a feasible solution in the graph and use it as the initial solution.
2. Iterate until a specified number of iterations without improvement is reached :
   2.1 : *Insertion.* Choose a vertex i to insert in the current solution.
   2.2 : *Removal.* Remove every vertex from the solution that is not adjacent to i (the current solution remains a clique).

To specify a selection mechanism in Step 2.1, we associate a value $V(i)$ with each vertex i. Intuitively, since the purpose is to find a large clique in the graph, it seems logical to insert the vertex having the highest number of vertices in the current solution adjacent to it. In the best case, the size of the current clique increases by one. Otherwise, since we remove the vertices that are not adjacent to it, this choice induces the smallest reduction of the size of the current clique.

Denote by $\bar{\delta}(i)$ the *current degree* of vertex i equal to the number of vertices in the current solution that are adjacent to i. Since in Step 2.1 we select the vertex i having the largest value $V(i)$, it follows that $V(i)$ should be proportionnal to $\bar{\delta}(i)$; i.e.

$$V(i) \approx \bar{\delta}(i)$$

It is easy to see that if we select the vertex in Step 2.1 only according to the value $\bar{\delta}(i)$, the procedure may cycle (i.e., inserting and removing sequentially a subset of vertices). Hence, a safeguard mechanism is required. Of course, such a mechanism could be to assign a tabu status to each removed vertex forbidding its selection for insertion during a fixed number of successive iterations. This alternative has the inconvenience of forbidding the selection of all removed vertices for the same number of successive iterations. But it is intuitively clear that we should allow earlier selection of removed vertices having larger current degree. Hence we use instead a *penalty factor* p to decrease the value $V(i)$ each time vertex i is removed in Step 3. Furthermore, at each subsequent iteration the *penalty* term $P(i)$ of vertex i evaporates according to an *evaporation factor* e. Thus the penalty $P(i)$ of vertex i is increased by the penalty factor p each time it is removed in Step 3 ; i.e. $P(i) = P(i) + p$ and at each subsequent iteration, it is reduced by the evaporation factor e ; i.e. $P(i) = Max\{0, P(i) - e\}$ (note that the penalty of a vertex is always non-negative).

The values $V(i)$ that are used in our implementation are the following :

$$V(i) = \bar{\delta}(i) - P(i)$$

These values are updated at each iteration. It should be obvious that the tabu status approach can be simulated using a very large penalty factor p and an evaporation factor $e = \frac{p}{nb}$ where nb corresponds to the fixed number of successive iterations where the selection of the removed vertex is forbidden. Note also the similarity with the approach used in ant colony search methods [21]. Indeed these constructive procedures rely on traces to select sequentially the variables and their values in order to generate better solutions. The values of the traces are adjusted according to the quality of the solutions generated, and they evaporate from iteration to iteration. The penalty term $P(i)$ can be seen as a (negative) trace in the selection process that slowly evaporates.

The value of a vertex is the main criterion for choosing the next vertex in Step 2.1. But since the maximal value can be reached for several vertices, additional criteria are used to break ties. This process can be implemented in many ways ; we chose to apply successively each criterion until the tie is broken. Hence, the order in which the

criteria are applied has an impact on the resulting selection. Each ordering induces a different variant of our algorithm. The order in which those criteria are tested is given as a parameter at the beginning of the execution of our algorithm. The criteria are specified in terms of the following properties of each vertex i :

1. $\delta(i)$, the number of vertices adjacent to i in the graph G (i.e. the *degree* of the vertex) ;
2. $\bar{\delta}(i)$, the current degree of vertex i ;
3. $f(i)$, the number of times vertex i has been inserted in the solution since the beginning (i.e. the *frequency* of the vertex).

Recall that one of the most used criterion in greedy procedures is one that selects the vertex having the largest degree $\delta(i)$. Indeed, such a vertex is most likely a member of a large clique. Furthermore, in the context of our P-E algorithm, the probability of removing a vertex i from the current clique should, in general, decrease as $\delta(i)$ increases.

Whenever we face ties in the selection process using $V(i)$, we may have to choose between some vertex having a high current degree that has been selected much recently (i.e. having a high penalty term) and some other vertex having a low current degree that has not been selected much recently (i.e. having a low penalty term). On the one hand, if the strategy is to explore more extensively the graph, then the second vertex should be selected. Hence, the secondary criteria should lead to select the vertex having the smallest current degree and the smallest frequency in order to diversify the search. On the other hand, if the strategy is to intensify the search around some local minima, then the first vertex should be selected. Hence, the secondary criteria should lead to select the vertex having the largest current degree and the largest frequency in order to insert the vertices having the highest potential of increasing the size of the current solution. Now, recall that if the P-E heuristic procedure is embedded into the decomposition algorithm, then the secondary criteria should be selected to intensify the search, since the decomposition algorithm already diversifies the search.

It follows from these comments that the criterion to break ties has to be selected according to the impact that we are seeking. Hence :

1) selecting the vertex with the largest degree drives the search in a part of the graph that is denser;

2) diversification is obtained by selecting the vertex with the smallest current degree or the smallest frequency;

3) intensification is obtained by selecting the vertex with the largest current degree or the largest frequency.

The P-E algorithm is summarized as follows :

{CS, BS are the current and best solutions, respectively;
p, e are the penalty and evaporation parameters, respectively.}

1. Initialisation :
   For each vertex j : $P(j) = 0$
   Choose randomly a vertex i as the initial solution
   $BS = CS = \{i\}$
   For each vertex j :
      If j is adjacent to i : $\bar{\delta}(j) = 1$
      Else : $\bar{\delta}(j) = 0$
2. While a specified number of iterations without improvement is not reached :
   2.1 Evaporate : for each vertex j, $P(j) = Max\{P(j) - e, 0\}$
   2.2 Construct the set of interesting vertices $S = \{j \mid j = Argmax\{V(l) \mid l \notin CS\}\}$
   2.3 Select a vertex i in S using tie-breaking criteria
   2.4 Insert i in CS. Remove from CS vertices that are not adjacent to i and add p to their penalty
   2.5 Update the current degree of each vertex
   2.6 If $|CS| > |BS|$ then $BS = CS$ (we improve the best solution)

In the next section, we analyze how to calibrate the penalty and evaporation values which are given as parameters at the beginning of the execution of the P-E algorithm.

5. Numerical Results

5.1. Parameters of the P-E algorithm

In this section, we present the analysis to calibrate the parameters of our penalty-evaporation algorithm introduced in Section 4. The objective is to identify values for the parameters that seem appropriate for all graphs independently of their shape or their density.

5.1.1 The penalty factor

The penalty factor p is certainly the most important parameter of the procedure. Indeed, a proper value of p should assume that the algorithm spends an appropriate amount of time looking for a large clique in any interesting part of the graph before moving to another one.

On the one hand, if the value of p is too large, then an interesting part of the graph will become uninteresting too fast and the diversification is completed prematurely before identifying an optimal solution in this part of the graph. On the other hand, if the value of p is too small, the search is intensified in only a small part of the graph, thus making the search very dependent on the starting clique in the graph. Furthermore, since each graph has its own shape and its own density, the value of the penalty factor should be fixed accordingly. If the graph is very dense, the algorithm does not need to move a lot throughout the graph; then the diversification is not an important issue, and a low value would be appropriate. But if the graph is sparse, a high value of p is required in order to search the feasible domain of the problem more extensively.

In order to identify an order of magnitude for the value of p, each of the 80 DIMACS problem is solved twice, using different starting solutions for the following values of p : 0,1,2,5,10. In these tests, the value of the evaporation factor e is fixed to 0, and no additional criterion is used to break ties (i.e. the vertex to be inserted is randomly selected among those having the largest value $V(i)$). Each entry in Table 3.1 is equal to the number of times the algorithm generates the best solution for some problem using the corresponding penalty factor p.

Penalty factor p	0	1	2	5	10
Number of times generating the best solution	33	61	47	37	37

TAB. 3.1 – Order of magnitude of the penalty factor p

The results in Table 3.1 indicate that the order of magnitude for p should be 1. This value can be seen as the starting point to identify the proper value of p for any specific graph.

5.1.2 The penalty and evaporation factors

Similar numerical tests are completed to identify good values for the penalty and evaporation factors; i.e. each of the 80 DIMACS instances is solved twice using different starting solutions for the pairs of values exhibited in Table 3.2. Note that no additional criterion is used to break ties, and that each entry in Table 3.2 is equal to the number of times the algorithm generates the best solution for some problem using the corresponding pair of values.

Since the role of the evaporation phase is to diminish a bit of the penalty's impact, it is only logical to think that the optimal penalty value will increase as we set the evaporation factor to a higher value. Hence, since we would be increasing the evaporation factor from 0 (see Section 5.1.1) to a positive value, we decided to increase the initial penalty value to 1.5 instead of 1 (the proposed penalty value found in Section 5.1.1). The initial evaporation value of 0.1 was set arbitrarily.

The results in Table 3.2 indicate that the quality of the results decreases as we move away from the order of magnitude of 1 for the penalty factor even if the value of the evaporation factor is adjusted. The results shown in bold characters in Table 3.2 allow to identify the best value for the evaporation factor associated with the corresponding values of the penalty factor.

5.1.3 Tie-breaking criteria

In this section we refer to the following tie-breaking criteria :

	Penalty								
Evaporation	0.5	0.7	1.1	1.5	1.9	2.3	2.7	3.1	3.5
0.001	64	69	56	53	52				
0.005	73	64	61	57	54				
0.01	**76**	**73**	69	58	54				
0.02	63	**73**	**72**	66	55	49			
0.05	54	65	66	**80**	75	64	58		
0.075		56	69	75	**80**	69	68	66	
0.1		50	60	73	76	**79**	74	75	63
0.125				63	75	73	**75**	**76**	68
0.15				61	71	75	70	68	71
0.2						71	67	74	**74**
0.25						67	68	71	71
0.3								67	67

TAB. 3.2 – Pairs of penalty and evaporation factors

Max δ among the set of vertices considered for insertion, select the one(s) having the largest degree δ.

Max $\bar{\delta}$ among the set of vertices considered for insertion, select the one(s) having the largest current degree $\bar{\delta}$.

Min $\bar{\delta}$ among the set of vertices considered for insertion, select the one(s) having the smallest current degree $\bar{\delta}$.

Max f among the set of vertices considered for insertion, select the one(s) having the largest frequency f.

Min f among the set of vertices considered for insertion, select the one(s) having the smallest frequency f.

The same testing approach is used to analyze the effectiveness of using different tie-breaking criterion or pairs of criteria used successively. Table 3.3 summarizes the results for several combination strategies and for different pairs of values for the penalty and evaporation factors that seem better according to the results in Table 3.2. The entries in bold characters in each row of Table 3.3 correspond to the penalty-evaporation pair

inducing the best performance for the corresponding strategy combination.

1st crit.	2nd crit.	Penalty-evaporation pairs 0.7-0.01	1.1-0.02	1.5-0.05	1.9-0.075	2.3-0.1
Max δ		75	**82**	81	78	78
Max $\bar{\delta}$		72	77	**82**	79	77
Min $\bar{\delta}$		75	**83**	77	78	78
Min f		71	**82**	76	77	78
Max f		74	**79**	73	72	75
Max δ	Max $\bar{\delta}$	73	**83**	81	78	78
Max δ	Min $\bar{\delta}$	76	**81**	**81**	77	77
Max δ	Min f	76	**81**	78	75	77
Max δ	Max f	74	79	**84**	75	75
Max $\bar{\delta}$	Max δ	74	**84**	82	78	76
Max $\bar{\delta}$	Min f	71	**81**	77	76	79
Max $\bar{\delta}$	Max f	77	**81**	74	73	76
Min $\bar{\delta}$	Max δ	77	**84**	82	74	78
Min $\bar{\delta}$	Min f	76	**79**	76	74	74
Min $\bar{\delta}$	Max f	74	**80**	73	74	73
Min f	Max δ	**80**	**80**	79	78	78
Min f	Max $\bar{\delta}$	71	**81**	77	77	79
Min f	Min $\bar{\delta}$	71	**81**	77	78	78
Max f	Max δ	76	**81**	77	77	74
Max f	Max $\bar{\delta}$	74	**80**	73	73	76
Max f	Min $\bar{\delta}$	74	**78**	74	71	72

TAB. 3.3 – Tie-breaking criteria combinations

Refering to Table 3.2, it is interesting to note that there always exists a combination of strategies improving the performance of the algorithm for any pair of values for the penalty and the evaporation factors. Furthermore, for the penalty-evaporation pair (1.1-0.02), all the strategies improve the performance of the algorithm. Also, for any combination of strategies, the performance of the algorithm is generally better using the penalty-evaporation pair (1.1-0.02).

It follows from this analysis that one of the best penalty-evaporation pair to use

is (1.1-0.02) together with the tie-breaking strategy Max $\bar{\delta}$ and then Max δ. In the sequel, we are using these parameters and added Min f as a third tie-breaking criterion (whenever needed).

5.2. Numerical comparisons

In this section, the 80 DIMACS problems are used to compare the performance of our methods with that of the following three methods :

• The MIN algorithm, as mentioned in Section 2, is one of the simplest and fastest heuristic procedure to find a large clique in a graph. Indeed, this algorithm finds a large clique in $O(|E|)$, where $|E|$ is the number of edges in the graph. But it is not very effective since it can determine the optimal solution of only 18 out of the 80 DIMACS instances. Hence we can conjecture that the problems solved to optimality by MIN should be the easiest to solve using more sophisticated algorithms.

• The Qualex algorithm [18] is a continuous-based heuristic method to solve the maximum weighted independent set problem which is equivalent to the maximum weighted clique problem (by replacing edges by anti-edges and vice versa). The problem is formulated as a quadratic programming problem where the feasible domain reduces to a unit hypercube. The edge constraints are embedded in the objective function. The approach proposed by Busygin in Qualex is to restrict the search over a sequence of local spheral domains generated by moving the center of the sphere inside the unit hypercube. Furthermore, the eigenproperties of the matrix are used to set the parameters of the spheres. Qualex is a reliable algorithm, both because of its speed (in $O(a|V|^3)$, where a is the number of tried sphere centers) and because of the quality of the solutions generated (51 out of the 80 DIMACS instances are solved to optimality, and the average proportion between the solution found by Qualex versus the optimal value on the 80 DIMACS instances is equal to 97.9%).

• The Tabu search approach is very effective to solve many difficult problems. Soriano and Gendreau [44] found 3 sets of parameters for which their tabu search algorithm was particularly successful, and they presented them at the DIMACS challenge [30].

We selected, for each graph, the best value obtained by those three versions of the tabu search algorithm. It is worthy to note that this combination generates the optimal solution for 57 out of the 80 DIMACS problems.

The results given by these three procedures (MIN, Qualex and Tabu) are compared with those obtained with the penalty-evaporation procedure (P-E), with the penalty-evaporation procedure embedded in the decomposition algorithm (Decomp. + P-E), and with the MIN procedure embedded in the decomposition algorithm (Decomp. + MIN), respectively. The results are summarized in Tables 3.4 - 3.6, where each entry is equal to the size of the largest clique generated for the corresponding problem using the corresponding method. The second column (Opt) includes the optimal (or best known) values for the DIMACS problems. Furthermore, the values of the parameters in the P-E procedure are fixed to the values identified in Section 5.1. Again, two different starting solutions are used for the P-E algorithm; whenever the size of the largest clique found differs between the two runs, the corresponding values are separated by a comma. Also, for each problem, values identified in bold characters are equal to the optimal (or best known) value. It is interesting to note that for problems c1000_9 and c2000_9, better solutions than the previously best known solutions were generated during the calibration of the parameters for the P-E procedure (this is marked using * in Tables 3.4 - 3.6).

Problem	Opt	MIN	Decomp. + MIN	Qualex	Tabu	P-E	Decomp. + P-E
brock200_1	21	19	19	**21**	**21**	20	20,**21**
brock200_2	12	9	10	**12**	11	10	11
brock200_3	15	13	14	**15**	14	14,13	14,**15**
brock200_4	17	15	15	**17**	16	16	**17**,16
brock400_1	27	22	23	**27**	25	23,24	25
brock400_2	29	21	23	**29**	25	23,24	**29**,25
brock400_3	31	21	23	**31**	25	24	**31**
brock400_4	33	23	23	**33**	25	23,24	25
brock800_1	23	17	19	**23**	21	19	21
brock800_2	24	18	19	**24**	21	19,20	21
brock800_3	25	18	19	**25**	21	19	22,21
brock800_4	26	18	20	**26**	21	20	21
c1000_9	68*	62	62	63	65	64,65	67
c125_9	34	31	33	33	**34**	**34**	**34**
c2000_5	16	13	15	**16**	**16**	15	**16**
c2000_9	77*	67	70	72	74	76,74	76
c250_9	44	42	42	43	**44**	**44**,43	**44**
c4000_5	18	14	16	17	17	16	**18**
c500_9	57	52	52	53	56	56,54	**57**
c-fat200-1	12	**12**	**12**	**12**	**12**	**12**	**12**
c-fat200-2	24	**24**	**24**	**24**	**24**	**24**	**24**
c-fat200-5	58	**58**	**58**	**58**	**58**	**58**	**58**
c-fat500-1	14	**14**	**14**	**14**	**14**	**14**	**14**
c-fat500-10	126	**126**	**126**	**126**	**126**	**126**	**126**
c-fat500-2	26	**26**	**26**	**26**	**26**	**26**	**26**
c-fat500-5	64	**64**	**64**	**64**	**64**	**64**	**64**
dsjc1000_5	15	13	14	14	**15**	14	**15**
dsjc500_5	13	11	12	**13**	**13**	12,**13**	**13**

TAB. 3.4 – Results for the 80 DIMACS problems(1)

Problem	Opt	MIN	Decomp. + MIN	Qualex	Tabu	P-E	Decomp. + P-E
gen200_p0_9_44	44	37	39	39	**44**	**44**	**44**
gen200_p0_9_55	55	38	50	**55**	**55**	**55**	**55**
gen400_p0_9_55	55	48	49	50	54	51	53
gen400_p0_9_65	65	45	48	**65**	**65**	**65**	**65**
gen400_p0_9_75	75	44	52	**75**	**75**	**75**	**75**
hamming10-2	512	**512**	**512**	**512**	**512**	**512**	**512**
hamming10-4	40	36	36	36	**40**	**40**	**40**
hamming6-2	32	**32**	**32**	**32**	**32**	**32**	**32**
hamming6-4	4	**4**	**4**	**4**	**4**	**4**	**4**
hamming8-2	128	**128**	**128**	**128**	**128**	**128**	**128**
hamming8-4	16	**16**	**16**	**16**	**16**	**16**	**16**
johnson16-2-4	8	**8**	**8**	**8**	**8**	**8**	**8**
johnson32-2-4	16	**16**	**16**	**16**	**16**	**16**	**16**
johnson8-2-4	4	**4**	**4**	**4**	**4**	**4**	**4**
johnson8-4-4	14	**14**	**14**	**14**	**14**	**14**	**14**
keller4	11	**11**	**11**	**11**	**11**	**11**	**11**
keller5	27	23	25	26	**27**	26,**27**	**27**
keller6	59	48	53	51	**59**	39,49	**59**
MANN_a27	126	125	125	**126**	125	125	125
MANN_a45	345	342	343	342	342	342	342
MANN_a81	1100	1096	1096	1096	1096	1096	1096
MANN_a9	16	**16**	**16**	**16**	**16**	**16**	**16**
p_hat1000-1	10	9	**10**	**10**	**10**	**10**	**10**
p_hat1000-2	46	43	45	45	**46**	**46**	**46**
p_hat1000-3	68	62	64	65	66	67,**68**	**68**
p_hat1500-1	12	10	11	**12**	11	**12**,11	**12**
p_hat1500-2	65	62	63	64	**65**	**65**	**65**
p_hat1500-3	94	85	92	91	**94**	**94**	**94**

TAB. 3.5 – Results for the 80 DIMACS problems(2)

Problem	Opt	MIN	Decomp. + MIN	Qualex	Tabu	P-E	Decomp. + P-E
p_hat300-1	8	7	7	**8**	**8**	**8**	**8**
p_hat300-2	25	24	**25**	24	**25**	**25**	**25**
p_hat300-3	36	33	34	35	**36**	**36**	**36**
p_hat500-1	9	8	**9**	**9**	**9**	**9**	**9**
p_hat500-2	36	33	35	**36**	**36**	**36**	**36**
p_hat500-3	50	46	48	48	**50**	49,**50**	**50**
p_hat700-1	11	8	9	**11**	**11**	**11**	**11**
p_hat700-2	44	42	**44**	43	**44**	**44**	**44**
p_hat700-3	62	59	**62**	61	**62**	**62**	**62**
san1000	15	10	10	**15**	10	8,9	**15**,10
san200_0_7_1	30	16	**30**	**30**	**30**	17	**30**
san200_0_7_2	18	15	15	**18**	**18**	13,15	**18**
san200_0_9_1	70	47	**70**	**70**	**70**	45,46	**70**
san200_0_9_2	60	38	**60**	**60**	**60**	**60**,43	**60**
san200_0_9_3	44	33	35	40	**44**	36,37	**44**
san400_0_5_1	13	8	**13**	**13**	**13**	8	**13**
san400_0_7_1	40	21	**40**	**40**	**40**	21,20	22,23
san400_0_7_2	30	17	22	**30**	**30**	18,19	**30**
san400_0_7_3	22	15	17	17	18	17	**22**
san400_0_9_1	100	92	**100**	**100**	**100**	54,55	**100**
sanr200_0_7	18	15	17	17	**18**	**18**	**18**
sanr200_0_9	42	37	41	41	**42**	**42**,41	**42**
sanr400_0_5	13	11	12	12	**13**	**13**,12	**13**
sanr400_0_7	21	18	20	20	**21**	**21**,20	**21**

TAB. 3.6 – Results for the 80 DIMACS problems(3)

	Opt	MIN	Decomp. + MIN	Qualex	Tabu	P-E	Decomp. + P-E
TOTAL	80	18	29	51	57	43,41	64,63
(w/o brocks)	68	18	29	39	56	43,41	61,60
Best of the 6 methods	80	19	31	52	59	45,42	67,66

TAB. 3.7 – Performance in terms of the number of best solutions generated

The results in Table 3.7 show the number of problems for which the corresponding methods either find the optimal solutions for the problem (with and without the "brocks" family of graphs) or generate the best solution among the six methods.

	MIN	Decomp. + MIN	Qualex	Tabu	P-E	Decomp. + P-E
MIN	-	50	58	58	53	59
Decomp. + MIN	0	-	36	45	36	48
Qualex	0	6	-	22	22	27
Tabu	0	1	14	-	3	13
P-E	3	9	25	25	-	29
Decomp. + P-E	0	2	9	2	0	-

TAB. 3.8 – Comparing the performance of the methods two by two (1)

In Tables 3.8 and 3.9, each pair of methods are compared. In Table 3.8, each entry is equal to the number of times the method of the corresponding column generates a better solution than the method in the corresponding row. For instance, the method Decomp. + MIN generates a better solution than MIN for 50 problems. For 27 problems, Decomp. + P-E generates better solutions than Qualex, but for 9 others, Qualex generates better solutions than Decomp. + P-E.

In Table 3.9, each entry is equal to the difference of the corresponding entry and the symmetric one in Table 3.8. Hence entry (Decomp. + P-E, Qualex) = 18 and entry (Qualex, Decomp. + P-E) = -18. It follows that symmetric entries having small

	MIN	Decomp. + MIN	Qualex	Tabu	P-E	Decomp. + P-E
MIN	-	50	58	58	50	59
Decomp. + MIN	-50	-	30	44	27	46
Qualex	-58	-30	-	8	-3	18
Tabu	-58	-44	-8	-	-22	11
P-E	-50	-27	3	22	-	29
Decomp. + P-E	-59	-46	-18	-11	-29	-

TAB. 3.9 – Comparing the performance of the methods two by two (2)

values (close to 0) indicate a close performance of the corresponding methods, and the difference in performance grows when the value of the symmetric entries increases. Thus the performance of P-E and Qualex methods is quite similar, but the Decomp. + P-E method seems to be much more effective than the MIN method.

The results in Table 3.9 indicate that the method MIN, even when embedded in the decomposition algorithm, is not competitive with the others as far as the quality of the solutions is concerned. Also, the other methods can be ordered in increasing order of their performance as follows : P-E, Qualex, Tabu, Decomp. + P-E. Note that this ordering is biased in favor of the Tabu search since the result for each problem in Tables 3.4 - 3.6 is the best one among three runs of the method using different parameters. Furthermore, embedding a method in the decomposition approach increases greatly its performance (see the results of Decomp. + MIN versus MIN and of Decomp. + P-E versus P-E).

In Table 3.10, we ordered the methods by execution speed (from fastest to slowest) when applied on the five machine benchmark instances used in the DIMACS challenge. The values in the table are the number of seconds each method used to complete their entire execution on an AMD Thunderbird 1200 MHz with 256M of memory. Each method has been compiled using Microsoft Visual C++ v6.0, except Qualex which has been compiled by Busygin and was graciously given for comparison purposes. Furthermore, we could not include the Tabu search in this table since the length of their

	r100.5	r200.5	r300.5	r400.5	r500.5
MIN	0.0	0.0	0.0	0.0	0.1
P-E	0.0	0.0	0.0	0.1	0.1
Decomp. + MIN	0.1	0.4	1.6	3.4	6.9
Qualex	0	1	3	8	20
Decomp. + P-E	0.2	1.6	4.1	8.6	18.9

TAB. 3.10 – Execution times, ordered from fastest to slowest

execution is a parameter given by the user at the beginning of the execution. Finally, the results in Table 3.10 show that MIN and P-E have similar execution times, as it is for Qualex and Decomp. + P-E. We can also conclude that the P-E algorithm is at least as fast as any other method, and the quality of the results are at least as good as any other method (except when embedded in the decomposition algorithm or when compared to the combination of the three tabu search runs).

6. Conclusion

In this paper, we introduced a decomposition algorithm and a penalty-evaporation approach to solve the maximum clique problem. The numerical results indicate that the effectiveness of any method (MIN or P-E) is greatly improved by embedding it in the decomposition algorithm. Furthermore the P-E method is competitive in terms of quality of the results with Qualex, one of the most effective heuristic procedure to solve the maximum clique problem, while being much faster.

The computing time of the Decomposition algorithm can be improved by using parallelism to complete Step 2.1 for each vertex i of the current largest clique simultaneously. The effectiveness of the penalty-evaporation method can also be improved by adjusting the values of the parameters according to the shape and the density of the graph. Furthermore, it would be interesting to verify the effectiveness of variants of the penalty-evaporation method where a clique is inserted in Step 2.4 instead of a vertex or where the values $V(i)$ are integer (for instance, $V(i) = \bar{\delta}(i) - \lceil P(i) \rceil$) in order to use the tie-breaking criteria more extensively.

Chapitre 4

Conclusion

Le problème de trouver la plus grande clique d'un graphe est un problème important de la théorie des graphes. Sa complexité est telle que résoudre ce problème en temps polynomial en fonction de la taille du graphe donnerait lieu à de puissants outils d'aide à la décision dans tous les domaines des sciences. Pour s'attaquer à ce problème, nous avons développé une méthode de décomposition et une méthode heuristique. La méthode de décomposition peut à la fois servir à améliorer l'efficacité d'une méthode exacte, ou à permettre à une méthode heuristique de mieux explorer le graphe, trouvant ainsi des cliques de taille supérieure. La méthode heuristique d'évaporation de pénalités est, d'une certaine façon, une généralisation continue du principe tabou. En effet, nous remplaçons une variable booléenne (statut tabou) par une variable continue (pénalité). Ce changement permet une flexibilité accrue de la méthode et favorise l'obtention de résultats de meilleure qualité, et ce plus rapidement qu'une méthode tabou.

Les résultats indiquent que nous pouvons améliorer de façon significative l'efficacité de plusieurs méthodes heuristiques en les utilisant en conjonction avec notre méthode de décomposition. Sur 80 graphes, MIN permet de trouver seulement 18 fois une solution optimale, alors qu'en conjonction avec notre méthode de décomposition, il permet d'identifier 29 solutions optimales. De même, notre méthode d'évaporation de pénalités permet de trouver 43 solutions optimales, alors qu'une fois utilisée en conjonction avec notre méthode de décomposition, 64 solutions optimales sont atteintes.

Pour fin de comparaison, parmi les méthodes les plus efficaces connues, nous avons choisi Qualex, un algorithme résolvant une formulation continue du problème et développé par Busygin [18], ainsi qu'une combinaison des meilleurs résultats de trois implé-

mentations d'un algorithme tabou, de Soriano et Gendreau [44]. Qualex trouve 51 solutions optimales sur les 80 graphes, alors que la combinaison des trois implémentations de la méthode tabou en trouve 57. On remarque que notre méthode de décomposition en conjonction avec notre méthode d'évaporation de pénalités donne de meilleurs résultats que ces deux approches, puisqu'elle identifie 64 solutions optimales. Il est toutefois intéressant de noter que les trois implémentations de la recherche tabou obtiennent indépendamment 49, 48 et 49 solutions optimales. À la lumière de ces résultats, on conclut donc que notre approche de décomposition en conjonction avec notre méthode d'évaporation de pénalités est supérieure aux deux autres puisqu'elle trouve au moins 13 solutions optimales de plus.

Nous pouvons proposer plusieurs façons d'améliorer autant la méthode de décomposition que l'heuristique d'évaporation de pénalités. Pour ce travail, plutôt que d'étudier ces améliorations, nous avons préféré fournir une analyse solide des bases de nos méthodes et donner une meilleure idée de l'efficacité des idées de base pour ainsi favoriser les développements futurs. Parmi les améliorations possibles, mentionnons les suivantes. D'abord, si la méthode heuristique utilisée en conjonction avec la méthode de décomposition est efficace, il arrive fréquemment qu'une solution optimale soit trouvée dans les premières itérations. S'il était possible d'obtenir une bonne borne supérieure sur la taille de la plus grande clique du graphe résiduel, on pourrait arrêter l'exécution de la méthode de décomposition lorsque cette valeur est inférieure ou égale à la taille de la plus grande clique trouvée jusqu'ici (BC). De plus, lorsqu'on vérifie si un noeud fait partie d'une clique de plus grande taille à l'aide d'une méthode heuristique, on pourrait aussi calculer une borne supérieure sur la taille de la plus grande clique à laquelle il peut appartenir. Si les valeurs des deux bornes sont égales, alors on est assuré que ce noeud ne peut faire partie d'une clique de taille supérieure. On peut donc supprimer ce noeud et ne jamais le considérer pour réinsertion dans une itération subséquente. Dans le cas où on appliquerait la méthode de décomposition en conjonction avec une méthode exacte, on pourrait utiliser des méthodes heuristiques pour toutes les vérifications, et seulement la méthode exacte lorsque les méthodes heuristiques ne trouvent pas de meilleures cliques. Par exemple, si, pour un noeud donné, sa borne supérieure est inférieure à BC, alors

on peut supprimer ce noeud sans risquer de manquer une clique optimale. Ainsi, pour ce noeud, il n'est pas nécessaire d'appliquer une méthode exacte pour vérifier s'il fait partie d'une clique de plus grande taille. Enfin, une autre façon de réduire le temps de calcul serait d'appliquer le processus de vérification simultanément pour plusieurs noeuds de la clique (à l'étape 2) dans un système en parallèle.

Pour ce qui est de la méthode heuristique d'évaporation de pénalités, on pourrait utiliser une formule beaucoup plus complexe pour la valeur d'un noeud, englobant également des critères secondaires. On pourrait aussi garder la formule courante, mais arrondir à l'entier supérieur. Ceci engendrerait un plus grand nombre de noeuds ayant la même valeur, et donnerait plus d'impact aux critères secondaires. Aussi, puisqu'en fait c'est l'absence d'arêtes entre le noeud que l'on introduit et certains noeuds de la clique courante qui entraîne le rejet de ces noeuds, on pourrait plutôt pénaliser cette absence que les noeuds eux-mêmes. Ainsi, la valeur d'un noeud lorsqu'on considère son introduction dans la clique courante serait calculée en considérant les pénalités dûes à l'absence de certaines arêtes par rapport aux noeuds de la clique courante. La valeur associée à une clique à insérer serait la taille de la clique résultante moins la moyenne des pénalités de ses noeuds.

Bibliographie

[1] AGGARWAL, C., J.B. ORLIN et R. TAI, «Optimized crossover for the independent set problem», *Operations Research*, vol. 45, 1997, pp. 226–234.

[2] ALIZADEH, F., «A sublinear-time randomized parallel algorithm for the maximum clique problem in perfect graphs», *presented at the ACM-SIAM Symposium on Discrete Algorithms 2*, 1991, pp. 188–194.

[3] BABEL, L., «A fast algorithm for the maximum weight clique problem», *Computing*, vol. 52, 1994, pp. 31–38.

[4] BABEL, L. et G. TINHOFER, «A branch and bound algorithm for the maximum clique problem», *ZOR–Methods and Models of Operations Research*, vol. 34, 1990, pp. 207–217.

[5] BALAS, E., S. CERIA, G. CORNUÉJOLS et G. PATAKI, «Polyhedral methods for the maximum clique problem», *in D.S. Johnson and M. Trick (editors). Cliques, Coloring and Satisfiability : Second DIMACS Implementation Challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 26, 1996, pp. 11–28.

[6] BALAS, E. et W. NIEHAUS, «Optimized crossover-based genetic algorithms for the maximum cardinality and maximum weight clique problems», *Journal of Heuristics*, vol. 4, no. 2, 1998, pp. 107–122.

[7] BANG-JENSEN, J., J. HUANG et A. YEO, «Convex-round and concave-round graphs», *SIAM Journal on Discrete Mathematics*, vol. 13, no. 2, 2000, pp. 179–193.

[8] BATTITI, R. et M. PROTASI, «Reactive local search for the maximum clique problem», *presented at WAE'97 : Workshop on Algorithm Engineering, Venice, Italy*, 1997.

[9] BERTONI, A., P. CAMPADELLI et G. GROSSI, «A discrete neural algorithm for the maximum clique problem : Analysis and circuit implementation», *presented at WAE'97 : Workshop on Algorithm Engineering, Venice, Italy*, 1997.

[10] BHATTACHARYA, B.K. et D. KALLER, «An o(m+n log n) algorithm for the maximum clique problem in circular-arc graphs», *Journal of Algorithms*, vol. 25, no. 2, 1997, pp. 336–358.

[11] BLOECHLIGER, I., «A new heuristic for the graph coloring problem», *Research report, Département de Mathématiques Appliquées, EPFL*, 2001.

[12] BOMZE, I.M., M. BUDINICH, M. PELILLO et C. ROSSI, «Annealed replication : A new heuristic for the maximum clique problem», *Discrete Applied Mathematics*, vol. 121, 2002, pp. 27–49.

[13] BOMZE, I., M. PELILLO et V. STIX, «Approximating the maximum weight clique using replicator dynamics», *IEEE Transactions on Neural Networks*, vol. 11, no. 6, 2000, pp. 1228–1241.

[14] BOMZE, I. M., M. BUDINICH, P. M. PARDALOS et M. PELILLO, «The maximum clique problem», *Handbook of Combinatorial Optimization*, vol. 4, 1999.

[15] BOURJOLLY, J.-M., G. LAPORTE et H. MERCURE, «A combinatorial column generation algorithm for the maximum stable set problem», *Operations Research Letters*, vol. 20, 1997, pp. 21–29.

[16] BROERSMA, H., T. KLOKS, D. KRATSCH et H. MULLER, «Independent sets in asteroidal triple-free graphs», *SIAM Discrete Mathematics*, vol. 12, no. 2, 1999, pp. 276–287.

[17] BUI, T.N. et Paul H. EPPLEY, «A hybrid genetic algorithm for the maximum clique problem», *Proceedings of the Sixth International Conference on Genetic Algorithms*, 1995, pp. 478–484.

[18] BUSYGIN, S., «Qualex 2.0 software release notes», *Private communication*, 2001.

[19] CARRAGHAN, R. et P.M. PARDALOS, «An exact algorithm for the maximum clique problem», *Operations Research Letters*, vol. 9, 1990, pp. 375–382.

[20] CICERONE, S. et G. Di STEFANO, «On the extension of bipartite to parity graphs», *Discrete Applied Mathematics*, vol. 95, no. 1-3, 1999, pp. 181–195.

[21] DORIGO, M. et G. Di CARO, «The ant colony optimization meta-heuristic», *New Ideas in Optimization*, 1999, pp. 11–32.

[22] DRAGAN, FF., «Strongly orderable graphs - a common generalization of strongly chordal and chordal bipartite graphs», *Discrete Applied Mathematics*, vol. 99, no. 1-3, 2000, pp. 427–442.

[23] FUJISAWA, K., S. MORITO et M. KUBO, «Experimental analyses of the life span method for the maximum stable set problem», *The Institute of Statistical Mathematics Cooperative Research Report*, vol. 75, 1995, pp. 135–165.

[24] GAVRIL, F., «Maximum weight independent sets and cliques in intersection graphs of filaments», *Information Processing Letters*, vol. 73, no. 5-6, 2000, pp. 181–188.

[25] GENDREAU, M., J.C. PICARD et L. ZUBIETA, «An efficient implicit enumeration algorithm for the maximum clique problem», *Lecture Notes in Economics and Mathematical Systems*, vol. 304, 1988, pp. 70–91.

[26] GIBBONS, L.E., D.W. HEARN et P.M. PARDALOS, «A continuous based heuristic for the maximum clique problem», *in D.S. Johnson and M. Trick (editors). Cliques, Coloring and Satisfiability : Second DIMACS Implementation Challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 26, 1996, pp. 103–124.

[27] GIBBONS, L.E., D.W. HEARN, P.M. PARDALOS et M.V. RAMANA, «Continuous characterizations of the maximum clique problem», *Mathematics of Operations Research*, vol. 22, 1997, pp. 754–768.

[28] HARANT, J., Z. RYJACEK et I. SCHIERMEYER, «Forbidden subgraphs and min-algorithm for independance number», *Discrete Mathematics*, (to appear).

[29] HOCHBAUM, D., «Approximating clique and biclique problems», *Journal of Algorithms*, vol. 29, no. 1, 1998, pp. 174–200.

[30] JOHNSON, D.S. et M. TRICK, *Cliques, Coloring and Satisfiability : Second DIMACS Implementation Challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 26. D.S. Johnson and M. Trick, 1996.

[31] MANNINO, C. et A. SASSANO, «An exact algorithm for the maximum stable set problem», *Computational Optimization and Application*, vol. 3, 1994, pp. 243–258.

[32] MANNINO, C. et A. SASSANO, «Edge projection and the maximum cardinality stable set problem», *in D.S. Johnson and M. Trick (editors). Cliques, Coloring and Satisfiability : Second DIMACS Implementation Challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 26, 1996, pp. 205–219.

[33] MANNINO, C. et E. STEFANUTTI, «An augmentation algorithm for the maximum weighted stable set problem», *Computational Optimization and Application*, vol. 14, no. 3, 1999, pp. 367–381.

[34] MARCHIORI, E., «A simple heuristic based genetic algorithm for the maximum clique problem», *presented at SAC'98 : ACM Symposium on Applied Computing*, 1998, pp. 366–373.

[35] MOSCA, R., «Stable sets in certain p-6-free graphs», *Discrete Applied Mathematics*, vol. 92, no. 2-3, 1999, pp. 177–191.

[36] MOTZKIN, T.S. et E.G. STRAUS, «Maxima for graphs and a new proof of a theorem of turan», *Canadian Journal of Mathematics*, vol. 17, no. 4, 1965, pp. 533–540.

[37] PARDALOS, P.M., J. RAPPE et M.G.C. RESENDE, «An exact parallel algorithm for the maximum clique problem», *in High Performance Algorithms and Software in Nonlinear Optimization, Kluwer : Applied Optimization*, vol. 24, 1998, pp. 279–300.

[38] PARDALOS, P.M. et G.P. RODGERS, «A branch and bound algorithm for the maximum clique problem», *Computers & Operations Research*, vol. 19, 1992, pp. 363–375.

[39] PELILLO, M., «Relaxation labeling networks for the maximum clique problem», *Journal of Artificial Neural Networks*, vol. 2, 1995, pp. 313–328.

[40] PELILLO, M., «Replicator equations, maximal cliques, and graph isomorphism», *Neural Computation*, vol. 11, no. 8, 1999, pp. 1933–1955.

[41] ROSSI, F. et S. SMRIGLIO, «A branch-and-cut algorithm for the maximum cardinality stable set problem», *Operations Research Letters*, vol. 28, no. 2, 2001, pp. 63–74.

[42] SEWELL, E.C., «A branch and bound algorithm for the stability number of a sparse graph», *INFORMS Journal on Computing*, vol. 10, 1998, pp. 438–447.

[43] SORIANO, P. et M. GENDREAU, «Diversification strategies in tabu search algorithms for the maximum clique problem», *Annals of Operations Research*, vol. 63, 1996, pp. 189–207.

[44] SORIANO, P. et M. GENDREAU, «Tabu search algorithms for the maximum clique problem», *in D.S. Johnson and M. Trick (editors). Cliques, Coloring and Satisfiability : Second DIMACS Implementation Challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 26, 1996, pp. 221–242.

[45] XUE, J., «Edge-maximal triangulated subgraphs and heuristics for the maximum clique problem», *Networks*, vol. 24, 1994, pp. 109–120.