Rapport BE Ocaml : Recherche de clique maximale dans un graphe

Ranem Nadir
Diop Abdoulaye
Le Roho Adrien
Hammouamri Ilyass

Sommaire

Introduction

Le problème de la clique maximale est un problème d'optimisation combinatoire important avec des applications dans de nombreux domaines, y compris la recherche d'informations, le traitement du signal, la classification, l'économie, la planification et l'ingénierie biomédicale. Outre ses applications pratiques dans des zones du monde réel, le problème de la clique maximale est étroitement lié à un certain nombre de problèmes combinatoires importants tels que la coloration des graphes, emballage de jeu et détection de communauté dans des réseaux complexes.

Cependant, on sait que les problèmes d'optimisation combinatoire sont difficiles à résoudre en général. La plupart d'entre eux, en particulier ceux qui présentent un intérêt pratique, appartiennent à la classe des problèmes NP-difficiles, et donc ne peuvent pas être efficacement résolus de façon optimale. Pour ces problèmes difficiles à résoudre à grande échelle, une variété d'approches d'optimisation, appelées heuristiques, a été proposée pour trouver des solutions sous-optimales de haute qualité en un temps de calcul raisonnable, constituant ainsi un cadre naturel et utile pour approximer des problèmes d'optimisation combinatoire difficiles.

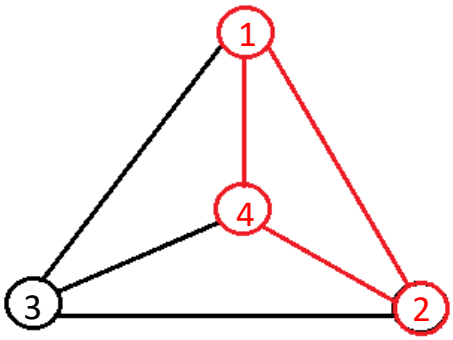
Un problème plus général que celui de la clique maximale est le problème de la clique maximale pondérée (*MWCP* pour *Maximum Weighted Clique Problem*) qui consiste toujours à trouver la clique maximale mais en termes de poids. En effet, le problème de la clique maximale (*MCP* pour *Maximum Clique Problem*) est un cas particulier des MWCP où le poids attribué à chaque sommet est unitaire. Cette étude s'intéresse particulièrement à une des différentes approches développées pour la résolution du MWCP : *Breakout Local Search (BLS).* Il s'agit d'un algorithme de type ILS (*Iterative Local Search*) basé sur deux phases : une recherche locale (intensification) pour trouver la meilleure solution dans un certain voisinage, suivi d'une perturbation de l'optimum local obtenu (diversification-pour explorer d'autres optimums locaux) par l'utilisation d'une liste Tabou dynamique et de différentes stratégies de perturbations.

Définition et Notations

Notion de Clique et de pondération

Soit $G = (V, E)$ un graphe non orienté avec *V* l'ensemble des sommets pondérés et $E \subset V \times V$ l'ensemble des arrêtes. On note w_v le poids du sommet v.

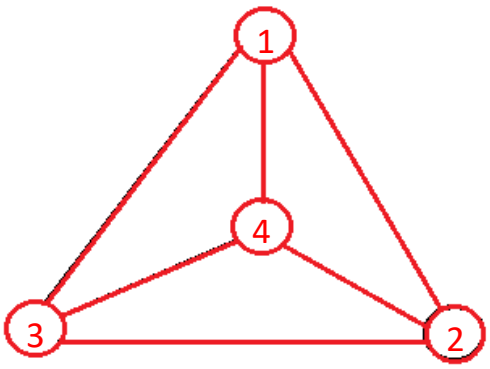
Une clique, $C \subset V$, est un ensemble de sommets deux-à-deux adjacents i.e. $\forall\, u, v \in C$ tels que $u \neq v$, $(u, v) \in E$. L'exemple suivant illustre la notion de clique pour un graphe dont tous les sommets ont le même poids, 1.



C = {1, 2, 4} est une clique

On appelle poids de la clique, noté $W(C)$, la somme des poids de ses nœuds, soit $W(C) = \sum_{v \in C} w_v$. La clique de l'exemple précédent a pour poids 3.

La clique est dite maximale lorsque son poids est maximal comparé aux poids de toutes les autres cliques du graphe.



C = {1, 2, 3, 4} est la clique maximale

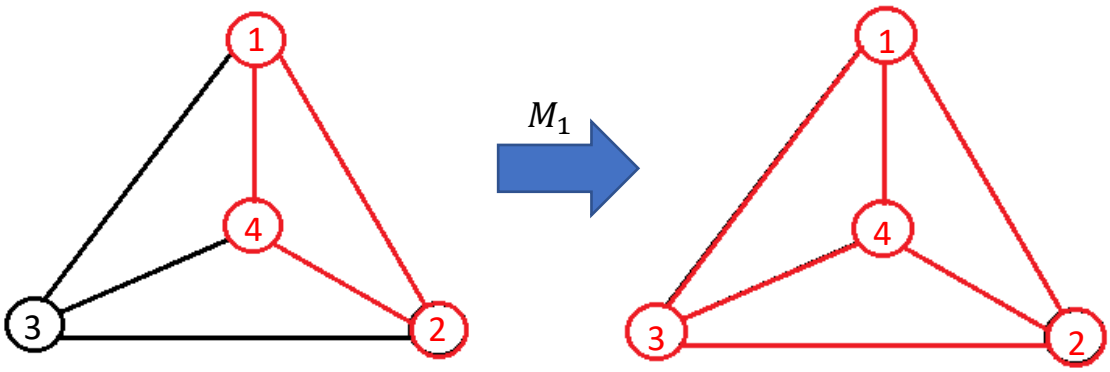
Mouvement

La recherche de la clique maximale dans cet algorithme repose sur la transformation de la solution courante. Pour cela, BLS utilise quatre opérateurs de mouvement distincts dont l'idée de base est de générer une nouvelle clique à partir de la clique actuelle C. Soit en ajoutant des sommets $v \in V\backslash C$ à C, soit en permutant les sommets u et v tel que $u \in C$ et $v \in V\backslash C$, ou soit en supprimant des sommets $v \in C$ de C.

Trois ensembles PA, OM et OC sont impliqués dans la définition de ces mouvements. L'ensemble de sommets PA est composé des sommets exclus de la clique C qui sont connectés à tous les sommets de C, c'est-à-dire $PA = \{v : v \notin C \; \forall \, u \in C, \{v, u\} \in E\}$. L'ensemble OM se compose des paires de sommets (v, u) telles que v soit exclu de C et soit connecté à tous les sommets de C sauf à un sommet $u \in C$ de C, c'est-à-dire $OM = \{(v, u) : v \notin C \; et \; u \in , |N(v) \cap C| = |C| - 1, \{v, u\} \notin E\}$ où $N(v) = \{i : i \in V, \{i, v\} \in E\}$. Enfin, l'ensemble OC se compose de tous les sommets exclus de la clique C, c'est-à-dire $OC = \{v : V\backslash C\}$.

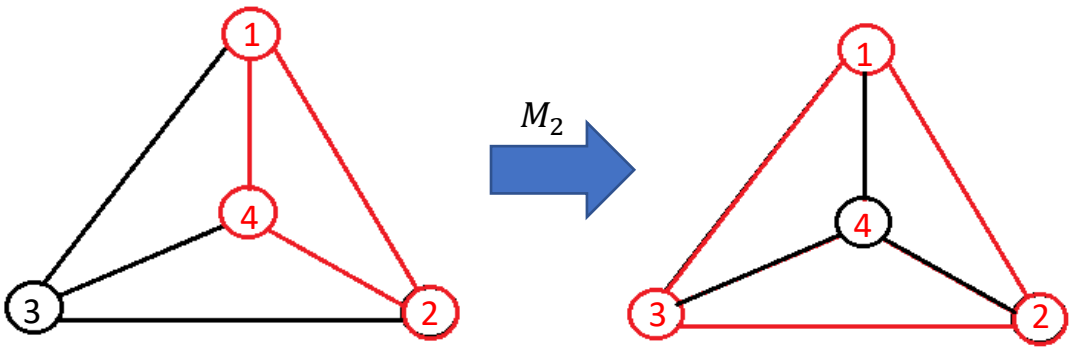
À l'aide de ces ensembles, on peut définir quatre mouvements comme suit :

- M_1 : Sélectionner un sommet $v \in PA$ et l'insérer dans C.



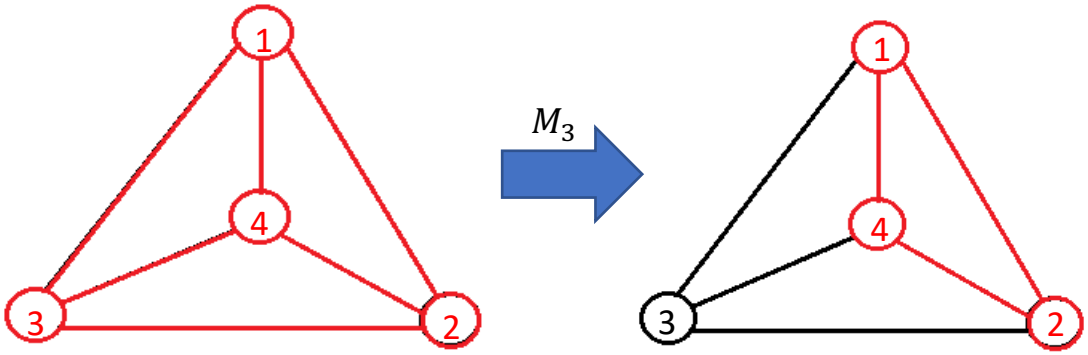
Mouvement d'ajout : ajout du nœud 3 à la clique

- M_2 : Sélectionner une paire de sommets $(v,u) \in OM$, insérer v dans C et retirer u de C.



Mouvement de Swap : ajout du nœud 3 et retrait du nœud 4

- M_3 : Sélectionner un sommet $v \in C$ et le retirer de C.



Mouvement de retrait : retrait du nœud 3 de la clique

- M_4 : Sélectionner un sommet $v \in OC$ tel que $\left(w_v + \sum_{\{v,u\} \in E, u \in C} w_u\right) \geq \alpha * f(C)$, où $f(C)$ est le coût actuel de la solution et $0 < \alpha < 1$. Ensuite, l'ajouter à C et réparer la clique résultante en enlevant tous les sommets x tels que $\{v,x\} \notin E$.

On peut observer ici que les différents mouvements peuvent améliorer ou détériorer la solution courante :

- M_1 : Ajout -> Amélioration de la solution courante du poids du sommet ajouté.
- M_2 : Échange -> Amélioration ou détérioration de la solution courante suivant que le poids du sommet à échanger est supérieur ou non au poids du sommet échangé.

- M_3 : Retrait -> Détérioration de la solution courante du poids du sommet supprimé.
- M_4 : Ajout + Correction -> Amélioration ou détérioration de la solution courante suivant que $\left(w_v + \sum_{\{v,u\} \in E, u \in C} w_u\right)$ est supérieur ou au poids de la solution courante $f(C)$.

État de l'art

Bien qu'ils ne soient pas adaptés aux problèmes de grandes tailles, on retrouve toujours des algorithmes exacts pour la résolution du problème de la clique maximale. Ils sont de manière générale basés sur la méthode branch-and-bound. Cependant nous nous intéresserons uniquement aux algorithmes utilisant des heuristiques. Dans ce qui suit, on retrouve les méthodes les plus connues.

Algorithmes *Gloutons*

La majorité des algorithmes gloutons dans la littérature pour le problème de la clique maximale sont appelés « heuristique séquentielle gloutonne ». Ces heuristiques génèrent une clique maximale à travers l'ajout répété de sommets dans une clique partielle ou la suppression répétée de sommets à partir d'un ensemble qui n'est pas une clique.

Kopf et Ruhe [Kopf et Ruhe 1987] ont nommé ces deux classes d'heuristique *Best in* et *Worst out*. Les Décisions concernant le sommet à ajouter ou à enlever sont basées sur des indicateurs particuliers associés aux sommets candidats. Par exemple, une heuristique de type *Best in* pourrait construire une clique maximale en ajoutant successivement le sommet de poids maximal parmi les sommets candidats. Dans ce cas, l'indicateur est le poids d'un sommet. D'autre part, une heuristique de type *Worst out* pourrait commencer avec l'ensemble de sommets entier *V* et supprimer successivement des sommets de *V* jusqu'à obtenir une clique.

On peut voir dans la littérature que la majorité des heuristiques gloutonnes pour le *MCP* appartiennent à ces deux classes. On retrouve par exemple l'algorithme glouton de Johnson [Johnson 1974], et celui de Tomita et al. [Tomita et al. 1988]. Les différences entre ces heuristiques sont leur choix d'indicateurs et la façon dont ces derniers sont mis à jour. Ces heuristiques

séquentielles gloutonnes sont très rapides par nature mais leur qualité est généralement peu satisfaisante.

Recherche Tabou

La recherche Tabou (TS) est une méta-heuristique de recherche locale dont la caractéristique essentielle est l'utilisation de Mémoire. Cette heuristique a été proposée indépendamment par Glover [Glover 1989] et par Hansen et Jaumard [Hansen et Jaumard 1990]. En général, la recherche Tabou vise à maximiser une fonction f en utilisant une recherche locale itérative (dans le cas d'un problème de maximisation). A chaque étape du processus itératif, le mouvement réalisé est celui qui maximise f dans le voisinage. Ce mouvement est exécuté même si f diminue par rapport à sa valeur au point actuel, pour sortir des optima locaux. Dès qu'un mouvement est appliqué, le mouvement inverse est interdit pour les *T* prochaines itérations (où *T est appelé « durée taboue » ou « tabu tenure »*). La valeur de la durée taboue *T* contrôle le degré de diversification de la recherche. Plus *T* est élevé plus la recherche est diversifiée.

Dans [Battiti et Protasi, 2001], Battiti et Protasi ont proposé un algorithme de recherche locale dynamique, *Reactive Local Search (RLS),* pour le MCP. RLS a été dérivé de la recherche Tabou dynamique [Battiti et Tecchiolli, 1994], qui est une méthode de recherche Tabou avancée et générale qui adapte automatiquement le paramètre de durée taboue *T* pour déterminer le degré de diversification adapté. RLS affiche des performances compétitives sur l'ensemble des instances de cliques maximales DIMACS, à la fois pour ce qui est de la qualité de la solution et du temps de calcul.

Recuit simulé

Le recuit simulé [Kirkpatrick et al. 1983] se déroule de la même manière qu'une recherche locale ordinaire, mais intègre une certaine randomisation dans la sélection des déplacements afin d'éviter d'être piégé dans un optimum local au moyen de déplacements non améliorants. Ces mouvements sont acceptés en fonction de probabilités tirées de l'analogie avec le processus de recuit.

Geng [Geng et al. 2007] ont présenté un algorithme de recuit simulé (SAA) pour le MCP. A chaque itération, la méthode l'algorithme génère un déplacement aléatoire (échange entre un sommet de la clique actuelle et un sommet situé en dehors de la clique actuelle). S'il s'agit d'un mouvement en

amélioration, il est automatiquement exécuté. Sinon, il peut toujours être fait avec une certaine probabilité (selon la distribution de Boltzmann) en utilisant un paramètre appelé température. L'algorithme commence par une température initiale TI = 100.0 qui est réduite progressivement en fonction du schéma de refroidissement. La méthode SAA se termine lorsque la température actuelle $t < TS$, où TS est la température finale réglée sur 0,001.

Autres méthodes

Parmi les méthodes les plus utilisées, on retrouve :

- Des méthodes à voisinage variable VNS (*Variable Neigbhorhood Search)*
- Des Méthodes à population comme ACO (*Ant Colony Optimization*)
- Utilisation de réseaux de neurones
- Et des algorithmes hybrides combinant différentes méthodes

BLS

Introduction

L'approche de BLS consiste à se déplacer d'un bassin d'attraction formé par un optimum local (ou attracteur) à un autre bassin en appliquant des sauts dirigés ou non, grands ou petits selon l'état de la recherche.

BLS part d'une solution initiale C_0 (voir partie *Génération d'une solution initiale*) et lui applique la recherche locale pour atteindre un optimum local C (voir partie *Recherche Locale*). Chaque itération de l'algorithme de recherche locale balaye tout le voisinage et sélectionne la solution voisine la plus performante pour améliorer la solution actuelle.

Si aucun voisin améliorant la solution courante existe, l'optimalité locale est atteinte. À ce stade, BLS tente de sortir du bassin d'attraction de l'optimum local actuel et de se déplacer dans un bassin d'attraction voisin. À cette fin, BLS applique un certain nombre de mouvements à l'optimum actuel C (on dit que C est perturbé). Chaque fois qu'un attracteur est perturbé, la solution perturbée est utilisée comme nouveau point de départ pour le prochain cycle de la procédure de recherche locale.

Si la recherche revient à l'attracteur C, BLS perturbe C plus fortement en augmentant le nombre de mouvements à appliquer pour la perturbation. Après avoir visité un certain nombre d'optimums locaux sans améliorer la meilleure solution, BLS applique une perturbation bien plus forte afin de conduire définitivement à la recherche d'une nouvelle région plus éloignée dans l'espace de recherche (voir partie *Stratégies de perturbations*).

Génération d'une solution initiale

La solution initiale C_0 utilisée par BLS est générée de la façon suivante. On commence par sélectionner un sommet $v \in V$ aléatoirement et on le place dans la clique C_0. Ensuite, tant qu'il existe un sommet $u \in V \backslash C$ tel que $\forall c \in C, \{u, c\} \in E$, ajouter u à C.

La génération de la solution initiale s'arrêtera donc quand aucun autre sommet ne pourra être ajouté à celle-ci sans qu'elle devienne invalide.

Recherche Locale

Le principe de cette étape est d'améliorer la clique jusqu'à obtenir un optimum local. La clique est dite améliorable lorsqu'on peut trouver un mouvement qui augmente strictement la fonction objective. De ce fait seuls les mouvements de type M_1 ou M_2 sont utilisés.

La recherche consiste à choisir le meilleur mouvement (*best_move)* parmi les mouvements de type M_1 et M_2. Il s'agit d'un mouvement dont le poids est strictement positif et maximal. Puis on l'applique en mettant à jour la clique, la liste Tabou et les ensembles *PA*, *OM* et *OC*. On répète l'opération tant qu'on peut améliorer la clique. Sinon on arrête la recherche locale.

Pour pouvoir explorer d'autres minimums locaux, on perturbe la recherche avec une amplitude qui dépend de la qualité du minimum local actuel et des précédents minimums locaux.

Stratégies de perturbations

Perturbation dirigée (recherche Tabou)

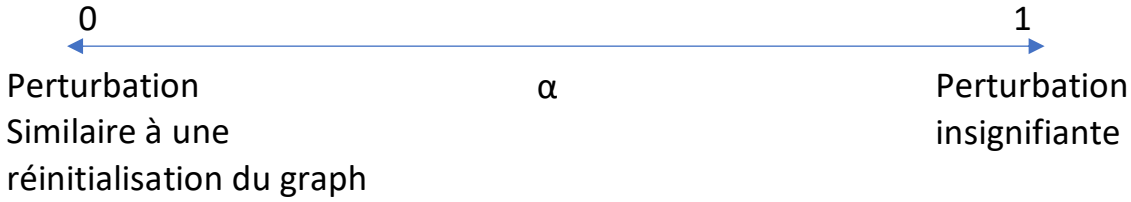
La perturbation ou recherche dirigée consiste à déployer la recherche avec les mouvements M_1, M_2 et M_3. Pour cela on choisit un mouvement dans l'ensemble A, ensemble qui contient tous les mouvements n'étant pas interdits par la liste Tabou et qui améliore au maximum la fonction objective.

$$A = \{m \mid m \in \{M_1 \cup M_2 \cup M_3\}, max\{\Delta_m\}, prohibited(m) = false\ or\ (\Delta_m + f(C)) > f_{best}\}$$

Notons que si le mouvement améliore la meilleure valeur de la fonction objective obtenue lors des itérations précédentes, on ne tient pas compte de la liste Tabou.

Grande perturbation

Il s'agit d'appliquer un mouvement de type M_4 avec un degré de α compris entre *0* et *1*. Plus *α* est proche de *0*, plus la perturbation est importante.



Cette perturbation est utilisée lorsqu'une stagnation est détectée, c'est-à-dire lorsque la solution (globale) n'est pas améliorée après un certain nombre (qui sera noté T) de recherches locales successives. Pour α assez petit, cette perturbation permet de sortir d'un minimum local et d'explorer d'autres régions de l'espace pour trouver une solution meilleure.

Cependant si α est trop petit, on détériore trop la solution et la recherche devient trop aléatoire. Il est donc très important de bien choisir α pour avoir un bon compromis entre diversification et intensification de la recherche.

Algorithme de perturbation

Si une stagnation est détectée, alors on applique une perturbation de type *M4* répétée *L* fois (voir avant) et $\alpha = \alpha_s$ (α_s faible). Sinon on calcule la probabilité *P* :

$$P = \begin{cases} e^{-\omega/T} & \text{if } e^{-\omega/T} > P_0 \\ P_0 & \text{otherwise} \end{cases}$$

Puis avec une probabilité *P*, on applique une perturbation dirigée avec l'ensemble *A* ou une perturbation de type M_4avec $\alpha = \alpha_r$ (α_r élevé) avec une probabilité *1 – P*. Cette perturbation est répétée *L* fois.

De plus à chaque étape, on met à jour la liste Tabou. La mise à jour de la liste Tabou consiste dans l'utilisation de deux tableaux :

- Le premier tableau stocke, pour chaque nœud (indice), l'itération où ce nœud a été retiré de la clique pour la dernière fois.
- Le second tableau stocke $\gamma = \emptyset + random|OM|$ pour chaque nœud. $\emptyset$ est un coefficient qui sera un paramètre global de la recherche, et $random|OM|$ une valeur aléatoire entre *1* et $|OM|$ (nombre d'éléments dans *OM*). Ainsi, la liste Tabou interdit de remettre un nœud déjà retiré dans la clique pendant γ itérations. Plus OM est grand plus la durée de l'interdiction est longue.

Implémentation

Structures de données utilisées

Graphe

On a choisi d'abord de représenter les nœuds du graphe par un type ***node*** qui est un couple ***id*** (*int*) et ***weight*** (*int*), ainsi le reste du programme utilisera le type ***node*** au lieu d'utiliser directement l'id ou le weight, comme ça on restera modulaire et un changement ultérieur dans le type des nœuds n'impactera pas le reste du programme.

On a ensuite représenté le graphe comme un tableau de nœuds ***nodes*** (*node array*) où la i-ème case contient le (i+1)-ème nœud, ce tableau est utilisé pour accéder aux attributs d'un nœud ayant son id ou bien de parcourir tous les nœuds du graphe. Et un tableau d'ensembles de nœuds ***edges*** (*node set array)* où la i-ème case contient un ensemble (*set*) de ses nœuds voisins, on a choisi la structure d'ensemble parce qu'elle facilite la recherche, on va faire des

recherches en temps logarithmique au lieu de linéaire si on utilisait des listes ou tableaux.

Clique

Les principales opérations qui seront effectuées sur la clique sont l'ajout et la suppression c'est pour ça qu'on a choisi d'implémenter la clique avec un Set de nœuds (*node set*), pour avoir des opérations d'ajouts et de suppression rapide en O(log(n)).

Ensembles PA, OM.

Les principales opérations qui seront effectuées sur ces deux ensembles sont l'insertion et l'extraction. Sachant que l'on cherche à extraire l'élément qui améliorera au maximum la clique, nous avons choisi comme structure une *pqueue*. Il s'agit d'une file de priorité ordonnée selon une priorité décroissante. La priorité, *prio*, d'un élément est définie comme la variation de la fonction objective lorsqu'on effectue le mouvement associé :

- Pour un mouvement de type *M1* qui ajoute un nœud ***v*** à la clique, on a prio(***v***) = ***v****.weight*
- Pour un mouvement de type *M1* qui swappe les nœuds ***u*** et ***v*** (où ***u*** est le nœud retiré de la clique), on a prio (***u***, ***v***) = ***v****.weight* – ***u****.weight*

Cette structure est très efficace pour notre problème. En effet l'extraction s'effectue en temps constant et l'insertion en O(log(n)). Cependant, elle est moins adaptée pour la mise à jour de l'ensemble *OM* qu'on est obligé de recalculer entièrement après chaque mouvement.

Liste Tabou.

On a implémenté la liste tabou en tant que tableau d'entiers (*int Array)* comme il a été suggéré dans le document [1], où la i-ème case correspond au nombre de la dernière itération ou le i-ème nœud a été supprimé de la clique, comme ca il sera interdit de le rajouter dans la clique avant un certain nombre d'itération.

Pseudocode

Algorithm 1. Breakout Local Search for the Maximum Clique Problem.

Require: Graph $G = (V,E)$, initial and maximal jump magnitude L_0 and L_{Max}, max. number T of non-improving attractors visited before strong perturb., coefficients α_r and α_s for random and strong random perturbations.
Ensure: The largest clique (for MCP) or the largest weight clique (for MWCP).

$C \leftarrow generate\_initial\_solution(G)$
Create initial PA, OM and OC sets /* Section 2.1.2 */
$f_c \leftarrow f(C)$ /* f_c records the objective value of the solu- tion */
$C_{best} \leftarrow C$ /* C_{best} records the best solution found so far */
$f_{best} \leftarrow f_c$ /* f_{best} records the best objective value reached so far */
$C_p \leftarrow C$ /* C_p records the last local optimum */
$\omega \leftarrow 0$ /* Set counter for consecutive non-improving local optima */
while stopping condition not reached **do**
  Select the best move m from the set of moves formed by the union $M_1 \cup M_2$ /* Section 2.1.2 */
  while $f(C \oplus m) > f_c$ **do**
    $C \leftarrow C \oplus m$ /* Perform the best-improving move */
    $f_c \leftarrow f(C \oplus m)$
    Update PA, OM and OC vertex sets
    $TL \leftarrow update\_tabu\_list(m,Iter)$ /* Section 2.2.2 */
    $Iter \leftarrow Iter + 1$
  end while
  if $f_c > f_{best}$ **then**
    $C_{best} \leftarrow C$; $f_{best} \leftarrow f_c$ /* Update the best solution found so far */
    $\omega \leftarrow 0$ /* Reset the counter of consecutive non-improving local optima */
  else
    $\omega \leftarrow \omega + 1$
  end if
/* Determine the perturbation strength L to be applied to C */
  if $\omega > T$ **then**
  /* Search seems to be stagnating, strong perturbation required */
    $L \leftarrow L_{Max}$
    $\omega \leftarrow 0$
  else if $C = C_p$ **then**
  /* Search returned to the previous local optimum, increment perturbation strength */
    $L \leftarrow L + 1$
  else
  /* Search escaped from the previous local optimum, reinitialize perturbation strength */
    $L \leftarrow L_0$
  end if
  /* Perturb the current local optimum C with perturbation strength L */
  $C_p \leftarrow C$
  $C \leftarrow Perturbation(C,L,TL,Iter,\omega,\alpha_r,\alpha_s)$ /* Section 2.2.2 */
end while

Tests et performances

Nous avons testé notre code sur quelques instances de cliques DIMACS. Notre implémentation de l'algorithme BLS utilise comme critère d'arrêt, un nombre maximum de 4000 itérations sur nos différents exemples. Pour chaque instance, l'algorithme a été exécuté 100 fois. Le tableau présenté ci-dessous présente les résultats obtenus pour chaque instance avec notamment la moyenne Avg, l'écart-type σ, le maximum et le minimum.

nom	|V|	|E|	Max connu	Max	Min	Avg	σ
c125.9	125	6 963	34	32	28	29.3	1.23
c250.9	250	27 984	44	38	32	35.9	0.65
brock200_4	200	13 089	17	15	15	15	0
Keller4	171	9 435	11	11	8	10.4	0.82
brock200_2	200	9 876	12	12	9	10.1	0.22
gen200_p0.9_44	200	17 910	44	34	31	32.7	0.61
gen200_p0.9_55	200	17 910	55	38	34	36.1	0.92
hamming8-4	256	20 864	16	16	16	16	0
p_hat300-1	300	10 933	8	8	7	7.84	0.37
P_hat300-2	300	21 928	25	25	23	24.6	0.55
P_hat300-3	300	33 390	36	32	27	30.2	1.08

On observe que l'on arrive à s'approcher des valeurs optimales, pour les différentes instances et que dans certains cas cette valeur est atteinte. Cependant nous avons remarqué que notre algorithme reste assez lent. En effet comparé aux temps d'exécution que l'on peut observer sur les différentes instances le nôtre est dix à cent fois plus grand. Cela s'explique en partie par le fait que la majorité de ces algorithmes sont implémentés en langage C ou en C++, d'où leur rapidité. Notre structure de donnée est aussi très certainement améliorable notamment pour les ensembles PA et OM.

Nous avions également envisagé d'effectuer des tests pour illustrer l'influences des paramètres de perturbation tels que α_s, α_r et P0 sur la qualité de la recherche. En effet ils ont un rôle important et leur choix doit être fait de manière à obtenir une recherche ni trop aléatoire ni trop locale. Malheureusement nous n'avons pas eu le temps de réaliser ces tests.

Optimisation (plateau)

Nous allons considérer dans cette partie un phénomène dit de plateau au cours de la recherche locale. Cela consiste à considérer les mouvements de swap qui laisse la fonction objective constante à condition qu'à l'itération suivante, on obtienne une amélioration de la fonction objective. En effet cette considération peut nous permettre d'intensifier la recherche et donc de converger plus vite vers la valeur optimale. Le tableau suivant traduit les résultats de l'algorithme amélioré exécuté 10 fois avec 4000 itérations à chaque exécution.

nom	|V|	|E|	Max connu	Max	Min	Avg	σ
c125.9	125	6 963	34	34	32	33.6	0.80
c250.9	250	27 984	44	41	39	39.8	0.87
brock200_4	200	13 089	17	16	15	15.7	0.46
Keller4	171	9 435	11	11	11	11	0
brock200_2	200	9 876	12	11	10	10.1	0.30
gen200_p0.9_44	200	17 910	44	40	35	38.1	1.44
gen200_p0.9_55	200	17 910	55	41	38	39.3	1.10
hamming8-4	256	20 864	16	16	16	16	0
p_hat300-1	300	10 933	8	8	8	8	0
P_hat300-2	300	21 928	25	25	25	25	0
P_hat300-3	300	33 390	36	35	33	34.2	0.6

On n'observe effectivement que la considération du phénomène de plateau nous permet d'améliorer grandement la qualité des solutions grâce à une recherche locale plus intensifié. La seule contrepartie est que le code résultant est un peu plus lent, ce qui est logique puisqu'on évalue le coût des mouvements de type plateau avant de les appliquer. Mais le résultat est quand même très satisfaisant.

Organisation du groupe

Durant ce projet, nous avons essayé de nous organiser du mieux possible pour pouvoir respecter les délais du projet. Pour cela nous avons essayé de diviser le travail en différents modules. Ces modules ont été organisés de façon à pouvoir travailler indépendamment (sans s'interdire de travailler sur différents modules pour résoudre les problématiques globales). Ainsi nous avons réalisé 6 modules :

- Un module pour l'implémentation des fonctions perturbations

- Un module consacré aux mouvements et à leurs applications
- Un module pour l'implémentation des différentes structures de données
- Un module main qui implémente l'algorithme BLS
- Un module de visualisation des graphes et des cliques
- Et enfin un module de test pour récupérer les performances de notre algorithme.

Conclusion

Dans cette étude, nous avons présenté l'algorithme BLS pour la résolution du problème de la clique maximale pondérée (MWCP) et le cas particulier du problème de la clique maximale (MCP).

BLS alterne entre une phase de recherche locale permettant de trouver un optimum local et une phase de perturbation permettant de diversifier la recherche dans le but d'explorer d'autres optimums locaux. La stratégie de perturbation est très importante. Dans BLS elle est déterminée en fonction de l'état de la recherche. C'est-à-dire que l'amplitude et le type des sauts effectués pour sortir d'un optimum local sont déterminés en fonction de la qualité de cet optimum et de l'historique de la recherche.

Nous avons également vu l'importance qu'il fallait accorder aux différents paramètres permettant la diversification de la recherche. En effet, il faut les choisir de sorte à obtenir un bon compromis entre diversification et intensification de la recherche. Enfin, il existe différents points auxquelles on pourrait s'intéressé par la suite pour améliorer les performances de notre algorithme, tels que la prise en compte des phénomènes de plateaux au cours de la recherche locale et l'utilisation de structures de données plus efficaces et plus adaptées pour ce problème. À terme, notre algorithme pourrait permettre de résoudre des problèmes réels tels que la résolution de conflits dans le domaine du trafic aérien.

Bibliographie

[1] Una Benlic, Jin-Kao Hao. Breakout Local Search for maximum clique problems. Computers & Operations Research 40 (2013) 192–206.

[2] Wayne Pullan. Approximating the maximum vertex/edge weighted clique using local search. J Heuristics (2008) 14: 117–134.

[3] Clique (graph theory). Wikipedia. https://en.wikipedia.org/wiki/Clique_(graph_theory)

[4] Clique problem. Wikipedia. https://en.wikipedia.org/wiki/Clique_problem