EA de Recherche Opérationnelle : Optimisation des transports dans les compétitions sportives

Maximilien Burq Sebastien Boyer

Septembre-Décembre 2013

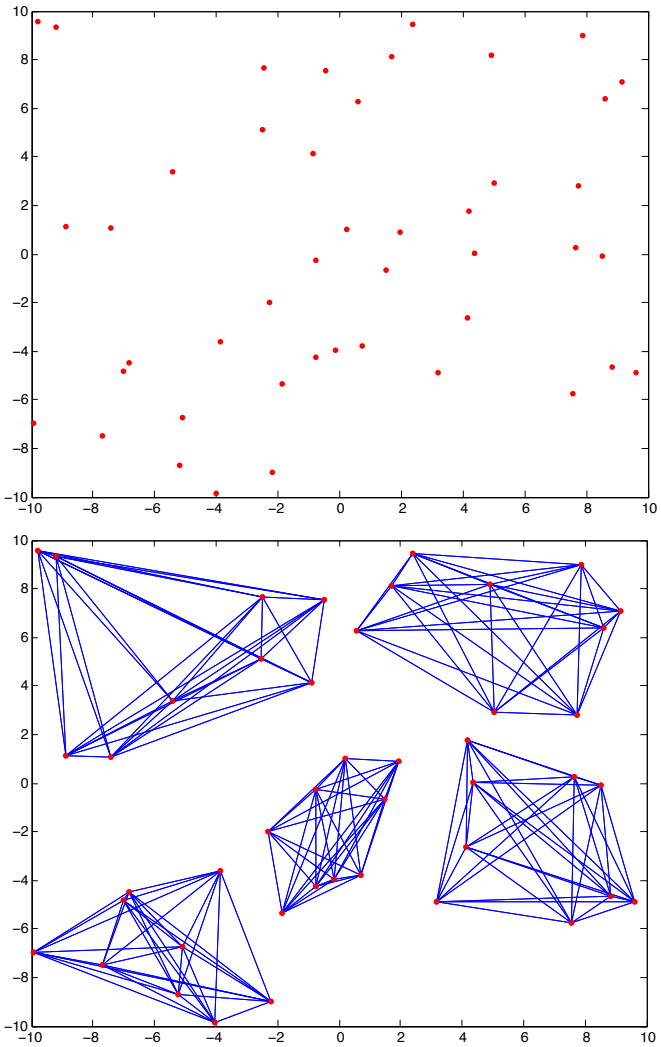


Table des matières

I Introduction

1.1 Contexte

Lors d'un championnat sportif, tous les clubs d'un même catégorie de niveau sont divisés en poules au sein desquelles ils effectuent la plupart de leur match lors d'une saison sportive. A la fin de la période de championnat tous les clubs d'une même poule se seront rencontrés les uns les autres (tous le même nombre de fois fixé par la fédération). En plus des émissions de gaz à effet de serre qu'ils provoquent, les transports induis par ces championnats à l'échelle nationale représente un des premiers postes de dépense pour la plupart des fédérations.

Le sujet de ce rapport est de discuter de la qualité du choix des poules vis-à-vis des coûts qu'elles induisent dans le championnat. A travers différents outils de recherche opérationnelle nous proposerons des méthodes pour choisir rationnellement une répartition des clubs en poules.

1.2 Etat de l'art

La littérature est à la fois riche et très variée sur le sujet. Le problème du partitionnement de N points en k groupe de taille égale de sorte à maximiser les coupes (équivalent à notre problème) est canoniquement appelé "Max k-section". D'autre problème similaire comme "Max k-cut" (on n'impose plus au partie d'être de cardinal égale) ont également été étudié.

"Improved Approximation Algorithms for MAX k-CUT and MAX bissection " de Mark Jerrum et Alan Frieze propose un algorithme de d'approximation polynomiale pour Max k-cut. Cet algorithme basé sur la relaxation SDP, renvoi une solution approchée avec une précision théorique (en espérance sur des instances de points aléatoires) de 60% pour $k = 2$ et qui descend jusqu'à 2% pour $k = 100$. Dans leur papier "Online semidefinite programming relaxations of

max k-section", E. De Klerk, D. Pasechnik, R. Sotirov, And C. Dobre donne plusieurs relaxations SDP pour le problème de max k-section, comparant les bornes obtenues pour chacune d'entre elles. Nous réutiliserons l'une d'entre elle dans la partie SDP pour ... .

1.3 Notation

La formalisation naturelle de ce problème consiste à considérer les clubs comme les nœuds d'un graphe complet, le coût de chaque arête représentant la distance entre ces deux clubs. On se donne donc :

- Un ensemble de sommet noté $\mathcal{V}$ (qui représenté l'ensemble des clubs)
- L'ensemble $\mathcal{E}$ des arêtes reliant ces sommets entre eux tel que le graphe $(\mathcal{V},\mathcal{E})$ soit complet
- Le cardinal de $\mathcal{V}$ noté N.
- La matrice C des distances entre les clubs.
- Le cardinal de l'ensemble des poules noté K.

1.4 Approximations

Pour pouvoir traiter ce problème et lui appliquer certain résultats théoriques nous justifions les approximations suivantes :

- $C \in S_n$. Cette approximation n'est pas restrictive puisque dans le cas où la relation distance ne serait pas symétrique, il suffira de symétriser C en prenant la moyenne (éventuellement pondérée par la proportion de match joué chez l'un) des deux distances.
- Le cardinal de chacune poules est le même.
- K divise N (assure que tous les points appartiennent bien à une poule grâce à l'approximation précédente).

II bissection par l'algorithme de Branch and bound

2.1 Démarche

Nous avons commencé par développer une heuristique pour traiter le problème de la bissection

:

$$arg \min_{S \subset \mathcal{V}, |S| = \frac{|\mathcal{V}|}{2}} Val(S, \overline{S})$$

où on a défini $Val(S, \overline{S})$ par : $Val(S, \overline{S}) = \sum_{i,j \in S} c_{i,j} + sum_{i,j \in \overline{S}} c_{i,j}$

On ne traitera pas dans cette partie de notion d'approximation mais bien d'un calcul déterministe d'une solution optimale.

L'algorithme glouton parvient à diviser en deux, en moins d'une minute (sur une machine commune), jusqu'à $N = 14$ points. Cette heuristique permet de traiter jusqu'à $N = 34$ points . Elle repose principalement sur la technique du Branch and bound et sur une estimation originale des Borne en chaque nœuds de l'arbre.

Donnons nous N points ainsi que la matrice des distances C entre ces points. Nous choisissons arbitrairement un des points, noté x_1 par la suite, comme racine de notre arbre et nous appellerons S la poule contenant ce point. Chaque nœud de l'arbre est un point et chaque sous arbre de ce nœud correspond a l'ensemble des configurations possibles telles que la poule S contienne x_1 ainsi que tous ses parents dans la généalogie de l'arbre.

L'algorithme construit ainsi progressivement la poule contenant x_1. La profondeur de l'arbre à explorer est donc de $N/2$ et chaque feuille représente une configuration possible de la bissection (on a alors : $\mid S \mid = N/2$). L'enjeux de cet algorithme est de "couper" un maximum de sous-arbre pour n'avoir à explorer finalement que les possibilités les plus plausibles. L'algorithme renvoi principalement la valeur optimale, $ValOpt$ ainsi que la configuration de S, noté $SOpt$, qui correspond.

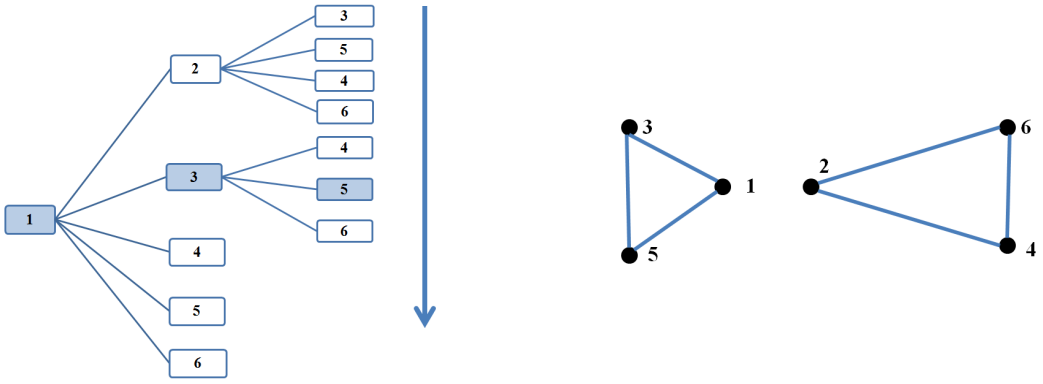


FIGURE 1 – Arbre de parcours des bissections possibles sur une instance particulière

- 1. Choix de x_1 (Nous discuterons ce choix par la suite) on appelle S_1 la poule à laquelle il appartient.
- 2. Pour un nœud k, on rajoute ce point à S : $S := S \bigcup k$
  Si $\mid S \mid = N/2$ alors on calcul $Val(S)$ pour cette configuration puis si $Val(S) < ValOpt$ on met à jour $ValOpt$ ainsi que $SOpt$. Après cette opération, on sort k de S et on passe au point suivant.
  Sinon, on calcul les Borne pour chacun des points $x_i \in \mathcal{V} \backslash S$. La borne d'un point correspond à un minorant de la valeur minimale du problème atteignable à partir du sous arbre formé par ce nœud :

$$\forall i, Borne(x_i) \leqslant \min_{S \supset k, Parents(k)} Val(S, \overline{S})$$

- 3. On ordonne les points restant x_i indéterminés à cette étape (ie les fils de k) par ordre croissant de Borne puis on explore les sous-arbres dans cet ordre en revenant à 2 pour chacun (appel récursif).

Note : On passe ici sur plusieurs raffinement que le lecteur trouvera directement dans les programmes. Par exemple, une fois exploré un sous-arbre correspondant à un point, ce point est retiré de la liste des "points restant indéterminés" pour tous ses "frères" et est placé temporairement (pendant l'exploration des sous-arbres correspondant à ses frères) dans la poule $\overline{S}$.

2.2 Evaluation de la Borne

Comme dans la plupart des problèmes de séparation-évaluation, la rapidité du problème est extrêmement sensible à la qualité de la borne. Une borne parfaite à chaque nœud (impossible en pratique car équivalent au problème initiale) permettrait de résoudre le problème en $N/2$ itérations de cet algorithme. Au contraire, une exploration totale de l'arbre (correspondant au cas

d'un Borne nulle) devrait faire environ $\binom{N}{N/2}$ itérations. Le calcul de Borne proposé ci-dessous permet de parcourir moins de $0,1\%$ de l'arbre total et donc de résoudre le problème en moins d'une minute pour des instances allant jusqu'à $N \approx 30$.

Exemple : Un ensemble de 16 points générés aléatoirement dans un carré du plan est traité en moyenne en 60 itérations ($\binom{16}{8} = 300000$ donc $0,02\%$ du total). Pour chaque nœud i on nomme $S(i)$ les points déjà attribué à la poule S avant d'y placer x_i (les Parents de i dans l'arbre) et $resteAChoisir = N/2 - |S(i)Ui|$, le nombre de points manquant pour compléter la poule.

Definition : On dira que x_k est le point le plus proche de l'ensemble W si

$$k \in arg\min_{j \in W} \sum_{i \in W} c_{i,k}$$

La borne est calculé en ajoutant les minorants correspondant au coût du sous-graphe complet de S et au coût du sous-graphe complet de $\overline{S}$.

Le minorant du coût du sous-graphe complet de S associé au point i est calculé en ajoutant les valeurs suivantes :

La somme des distances entre les points déjà attribué à S :

$$\sum_{(k,l) \in S(i) \bigcup (i)} C_{k,l}$$

La somme des distances entre les parents de i et les $resteAchoisir - 1$ points qui suivent i dans l'ordre des points les plus proche de l'ensemble Parents de i.

$$\min_{(j_1, j_2, \ldots, j_{resteAChoisir}) \subset (1, \ldots, N) \setminus Parents(i)} \sum_{k=1}^{resteAChoisir-1} \sum_{p \in Parents(i)} C_{j_k,i}$$

La somme des a plus petites arrêtes du graphe où a est le nombre d'arrêtes restant à ajouter pour que la borne contienne exactement $N * (N/2 - 1)/2$ arrêtes.

$$\min_{(i_1, j_1, \ldots, i_a, j_a) \subset (1, \ldots, N)^2} \sum_{k=1}^{a} c_{j_k, i_k} \quad \text{avec} \quad a = (resteAchoisir * (resteAChoisir - 1)/2)$$

A ce minorant on ajoute ensuite la valeur similaire correspondant au point d'ores-déjà attribué à $\overline{S}$ (ie, un minorant du coût du sous-graphe complet de $\overline{S}$).

2.3 Quelques statistiques

On observe d'une part que malgré la borne, le nombre moyen d'itération est une fonction exponentielle de N. D'autre part, que la durée moyenne d'une itération n'est pas sensible à N (durée = fonction linéaire du nombre d'itération). L'explosion du temps de calcul réside donc bien

N	16	18	20	22	24	26	28
Temps de calcul moyen	0.06	0.12	0.27	0.56	1.31	2.30	8.02
Nombre moyen d'appel récursif	51	90	159	287	507	849	2057
Ecart type	15	30	76	144	325	562	1191
Iteration moyenne du succès	13	13	35	46	57	91	196
Ecart type	12	16	53	83	128	172	491
Succès dès la première feuille	73%	83%	77%	77%	80%	80%	87%
Erreur max à la première feuille	17%	16%	13%	15%	12%	ND	ND

FIGURE 2 – Statistiques obtenues avec 30 simulations de N points dans le carré $[-10; 10]^2$ du plan pour N de 16 à 28 points.

dans le nombre exponentiel d'appel récursif et non dans la complexité des calculs à chaque itération.

On observe premièrement que le nombre d'appel récursif du programme croît exponentiellement avec N. L'algorithme du BandB n'est donc pas parvenu à se débarrasser du caractère combinatoire du problème. Nous notons également que le temps de calcul par appel récursif évolue quasiment linéairement avec N. L'explosion du temps de calcul provient bien de celui du nombre d'appel récursif.

Enfin la remarque cruciale de cette étude vient des deux dernières lignes du tableau. On remarque que dans environ 80% des cas (et quelque soit la valeur de N) l'algorithme trouve la solution optimale dès la première feuille explorée. De plus dans le cas où la solution optimale n'est pas trouvée à la première feuille, on remarque que la solution trouvée est expérimentalement inférieur à $1.20 * SolutionOptimale$.

Une première question intéressante serait de savoir si l'erreur commise au pire des cas par cette première configuration est théoriquement borné.

La seconde question que l'on doit se poser est si la précision de la valeur obtenu lors de l'exploration de la première feuille est sensible au choix de x_1 (aléatoire ici). En effet tous les points de départs ne menant pas à la même première feuille. On se demandera donc aussi si il est possible de procéder de manière déterministe à un "bon choix" de ce x_1.

III Relaxation SDP

3.1 Modélisation

Le problème de la bissection

peut s'écrire sous forme d'un MAXCUT en remarquant que minimiser les arrêtes dans chacune des composantes connexes de la bissection

revient à maximiser les arrêtes entre ces composantes. En notant L la matrice vérifiant

$\forall i \in [\![1;n]\!] L_{i,i} = \sum_{j=1}^{n} C_{i,j}$ et $\forall i \neq j L_{i,j} = -C_{i,j}$, On obtient le problème de maximisation [Gaubert] :

$$\max_{x\in(-1,1)^n} \frac{1}{4} x^T L x = \max_{X\in S_n^+, \forall i X_{i,i}=1, rg(X)=1} \frac{1}{4}\langle L, X\rangle$$

Nous avons donc étudié le problème SDP correspondant, après avoir relâché la contrainte de rang. Plusieurs constats :

- La résolution avec le solveur cvx, pour des instances de l'ordre de $n = 30$ est de l'ordre de la seconde. Cela rend une utilisation de la borne du problème SDP dans un algorithme de Branch and Bound impossible
- Puisque ce problème s'apparente à un MAXCUT, cette méthode ne donne pas, dans le cas général, une équipartition.

3.2 Pénalisation

Pour résoudre cela, nous avons procédé à une pénalisation. Un simple calcul montre que si l'on note p le nombre de clubs dans la première des deux poules, nous avons p^2 arrêtes dans cette poule. On obtient donc $n^2 - p^2 - (n-p)^2 = 2p(n-p)$ arrêtes coupées. On remarque donc que ce nombre est maximal pour $p = \frac{n}{2}$

Une solution pour imposer la contrainte d'équipartition consiste donc à pénaliser le problème SDP de façon à maximiser le nombre d'arrêtes. Nous avons donc augmenté artificiellement la valeur de toutes les arrêtes par $C'_{i,j} = K + C_{i,j} \forall i \neq j$ Pour I^+, I^- une bissection

des noeuds de $\mathcal{V}$, on note $c(I^+, I^-)$ la valeur de la coupe associée aux arrêtes C et $c'(I^+, I^-)$ la valeur associée à C'. On obtient donc

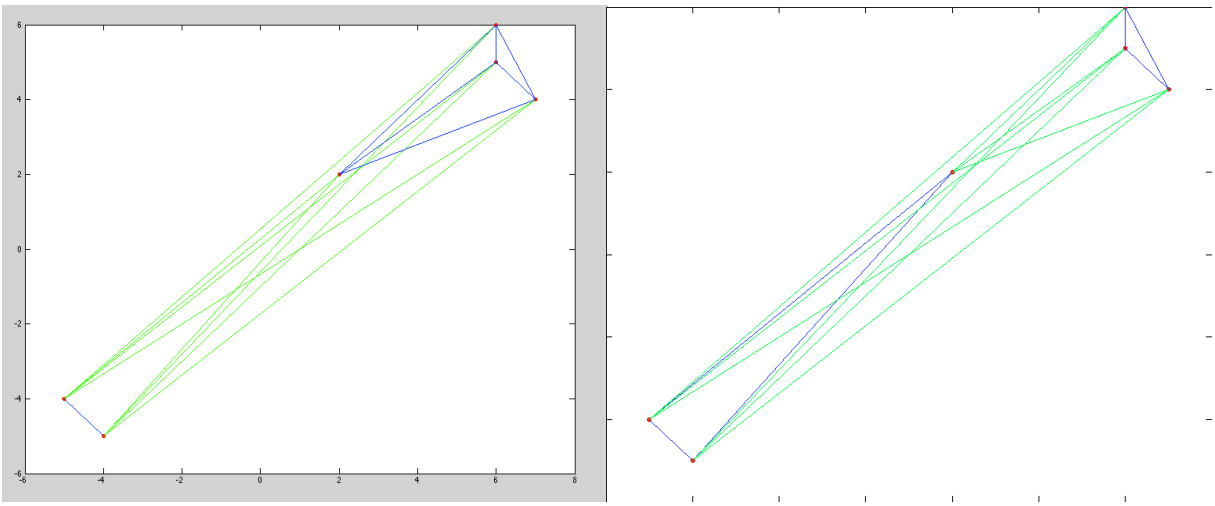
$$c'(I^+, I^-) = c(I^+, I^-) + KN$$

où $N \leqslant \frac{n^2}{2}$ est le nombre d'arrêtes entre I^+ et I^-.

3.3 Résultats

Nous avons remarqué plusieurs choses lorsque la pénalisation tend vers $+\infty$

- En théorie, les valeurs prises par la solution du programme SDP sont stationnaires (lorsque le nombre de parties est maximal). Ce n'est pas observé en pratique.
- Pour K grand, les matrices réelles solution du SDP ont leurs coefficients dans $\{-1, 1\}$. On remarque également que dans de "nombreux" cas, le rang de la matrice est 1. Dans ces cas là, on a donc une solution exacte du problème de répartition en deux poules.
- En pratique, les K nécessaires à l'obtention d'une matrice X de rang 1 croissent exponentiellement en la taille du problème. Pour des $n \approx 100$ on ne parvient plus à obtenir de solution exacte.
- Nous avons enfin remarqué que dans le cas général, le rang de X est une mesure du "degré de connexité" du graphe. Pour des solutions de rang faible, on peut donc envisager d'effectuer un Branch and Bound sur un petit nombre de points indécis



(a) Sans pénalisation (b) Avec pénalisation

FIGURE 3 – Penalisation des partitions deséquilibrées (en vert les arêtes coupées)

IV Ploblème de la k-équipartition

Nous nous intéressions désormais au cas général de N clubs à séparer en k poules de même taille. Nous avons adopté une modélisation générale sous forme de programme linéaire de notre problème.

4.1 Modélisation

Fonction à minimiser

On utilise pour cela une matrice $M \in \{0,1\}^{N^2}$. On utilise la sémantique $M_{i,j} = 1$ si et seulement si les clubs i et j sont dans la même poule.

Quelques remarques a priori :

– La matrice M est symétrique.

– Les éléments diagonaux sont égaux à 1

De même que dans le cas de la bissection

, on se donne la matrice C des coûts, ie : $C_{i,j}$ est égal à la distance entre les clubs i et j. La matrice C n'est pas nécessairement symétrique.

Contraintes

Les contraintes de la k-équipartition sont relativement difficiles à expliciter :

– Tout d'abord, il est nécessaire que chaque club ait exactement N/k voisins :

$$\forall i \in [\![1, N]\!], \sum_{j=1}^{N} M_{i,j} = \frac{N}{k}$$

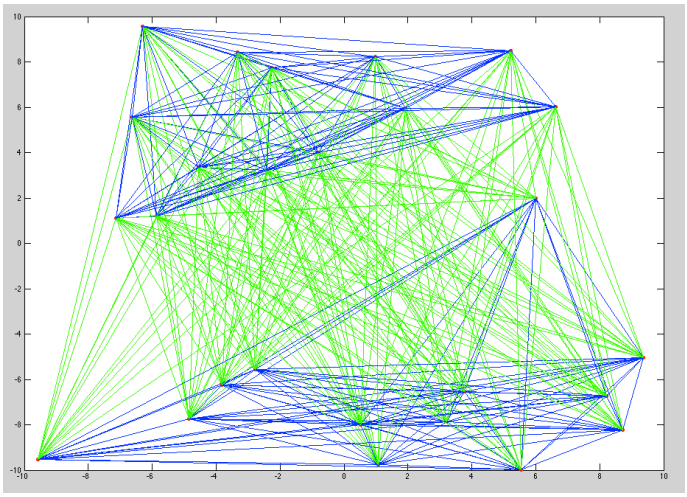


FIGURE 4 – Pour 50 points, cas où le rang de la matrice M est égal à 1

– Ensuite, il est nécessaire d'avoir la transitivité de la relation "être dans la même poule" :

$$\forall (i,j,k) \in [\![1,N]\!]^3, (M_{i,j} = 1) \Rightarrow (M_{j,k} = M_{k,i})$$

– Ces N^2 contraintes peuvent se traduire sous forme linéaire par les $2 * N^2$ nouvelles contraintes :

$$\forall (i,j,k) \in [\![1,N]\!]^3, (M_{j,k} - M_{k,i}) \leqslant 1 - M_{i,j}$$
$$\forall (i,j,k) \in [\![1,N]\!]^3, (M_{k,i} - M_{j,k}) \leqslant 1 - M_{i,j}$$

– Il est clair que dans un soucis d'optimisation il est possible de réduire grandement le nombre de contraintes, notamment en exploitant la symétrie de la matrice M.

4.2 Résolution exacte

On obtient donc un programme linéaire en nombres entiers. Plusieurs possibilités s'offrent à nous pour tenter de le résoudre.

Méthode du simplexe

Nous avons tenté d'utiliser directement les algorithmes pré-implémentés dans Matlab pour la résolution des PLNE, qui sont basés sur la méthode du simplexe. Résultat : le temps de calcul explose pour des instances dépassant 10 clubs.

Cf la table des temps de calcul obtenus.

Branch And Bound

L'implémentation de la méthode de Branch and bound repose sur l'obtention d'une borne pour des problèmes intermédiaires. Pour obtenir une telle borne, nous avons tenté le solveur linéaire relâché continu, pré-implémenté Matlab, mais les résultats étaient décevants. Nous verrons plus loin comment l'utilisation d'une borne obtenue par le solveur SDP nous a permis de surmonter cette difficulté.

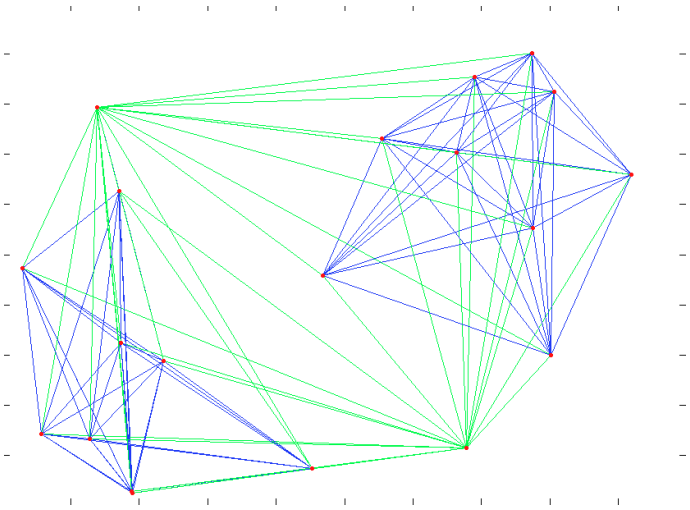


FIGURE 5 – Cas où le rang de la matrice M est égal à 3

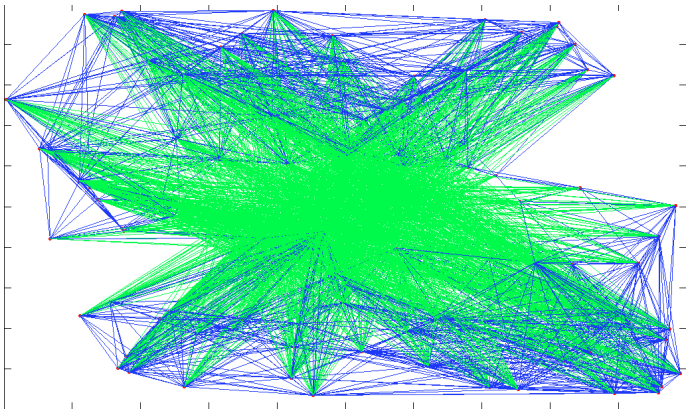


FIGURE 6 – Cas d'une matrice M de rang 7, faible devant la dimension globale (100 points)

4.3 Méthode SDP

Nous avons vu qu'on pouvait écrire l'algorithme de la bissection

comme un problème de MAXCUT, qu'on pouvait approcher par un problème de minimisation SDP. Cette démarche est tout-à-fait généralisable au cas général de la K-équipartition. Une possibilité est de relâcher les contraintes de transitivité pour écrire le problème SDP :

$$\max \frac{1}{4}\langle L, X\rangle \ \ s.c. \ \ X \in S_n^+, \forall i \ X_{i,i} = 1, \forall i \sum_{j=1}^{N} X_{i,j} = \frac{N}{K}$$

Il est en fait inutile d'ajouter la contrainte de nombre de voisins $\forall i \sum_{j=1}^{N} X_{i,j} = \frac{N}{K}$, puisque l'on peut généraliser le résultat de pénalisation de la section précédente :

Soit $(S_1, ..., S_k)$ une partition de $\mathcal{V}$. On note $(p_1, ..., p_K)$ les nombres de sommets associés à chaque partie. On différentie $p_K = n - \sum_{i=1}^{K-1} p_k$. On s'intéresse au nombre total d'arrêtes coupées :

$$N = n^2 - \sum_{i=1}^{K} p_k^2 = -\sum_{i=1}^{K-1} p_k^2 + 2n \sum_{i=1}^{K-1} p_k - (\sum_{i=1}^{K-1} p_k)^2$$

$$\frac{\partial N}{\partial p_i} = -2p_i + 2n - 2 \sum_{i=1, i\neq k}^{K-1} p_i$$

$$\frac{\partial N}{\partial p_i} = -2p_i + 2n - 2(n - p_K)$$

Donc $(\nabla N = 0) \Rightarrow (\forall j \in [\![1; K-1]\!], p_j = p_k)$ On a donc un extremum (il est facile de vérifier que c'est un max) du nombre d'arrêtes quand les parties sont toutes de la même taille.

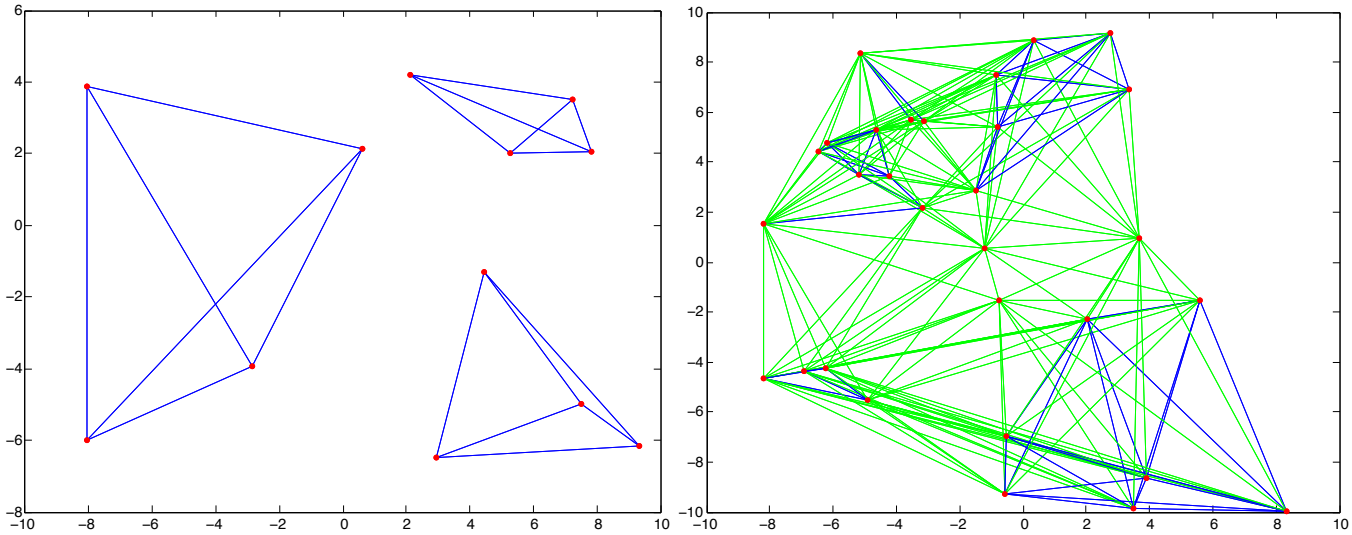
Comme précédemment il s'agit donc seulement de résoudre le cas de la K-coupe maximale.

$$\max \frac{1}{4}\langle L, X\rangle \ \ s.c. \ \ X \in S_n^+, \forall i \ X_{i,i} = 1, \forall i, j \ \ 0 \leqslant X_{i,j} \leqslant 1$$

Problème rencontré : même pour des instances "simples", la matrice M résultat de ce problème SDP n'a pas les propriétés de partition. Tout d'abord, la matrice M a ses coefficients dans $[0, 1]$. La solution pour se ramener à une matrice M' dont les coefficients sont dans $\{0, 1\}$ a été de déterminer une précision $p \in]0, 1[$ et d'écrire $\forall i, j \ M'_{i,j} = 0$ si $M_{i,j} < 1 - p$, $M'_{i,j} = 1$ si $M_{i,j} > 1 - p$. Dans toutes les figures qui suivent, on a pris $p = 0.1$ choisi de tracer en bleu les arêtes correspondant à $M_{i,j} > 1 - p$ et en vert celles qui correspondent à $M_{i,j} < 1 - p$ On observe deux types de résultats

- La répartition donnée par M' correspond bien à une répartition. C'est un cas fréquent pour des petites instances, mais qui devient rare lorsque N et K augmentent
- La répartition est mauvaise car les coefficients de la matrice sont tous fractionnaires.

Une des solutions trouvées pour palier au problème des arêtes fractionnaires a été de pénaliser le choix de valeurs fractionnaires pour les arêtes de la matrice solution. Pour cela une possibilité consiste à travailler sur une matrice C' sur laquelle on a "éclaté" les valeurs des distances entre les points. Un moyen de le faire consiste à prendre $C'_{i,j} = (C_{i,j})^l$ pour l assez grand. Discussion :

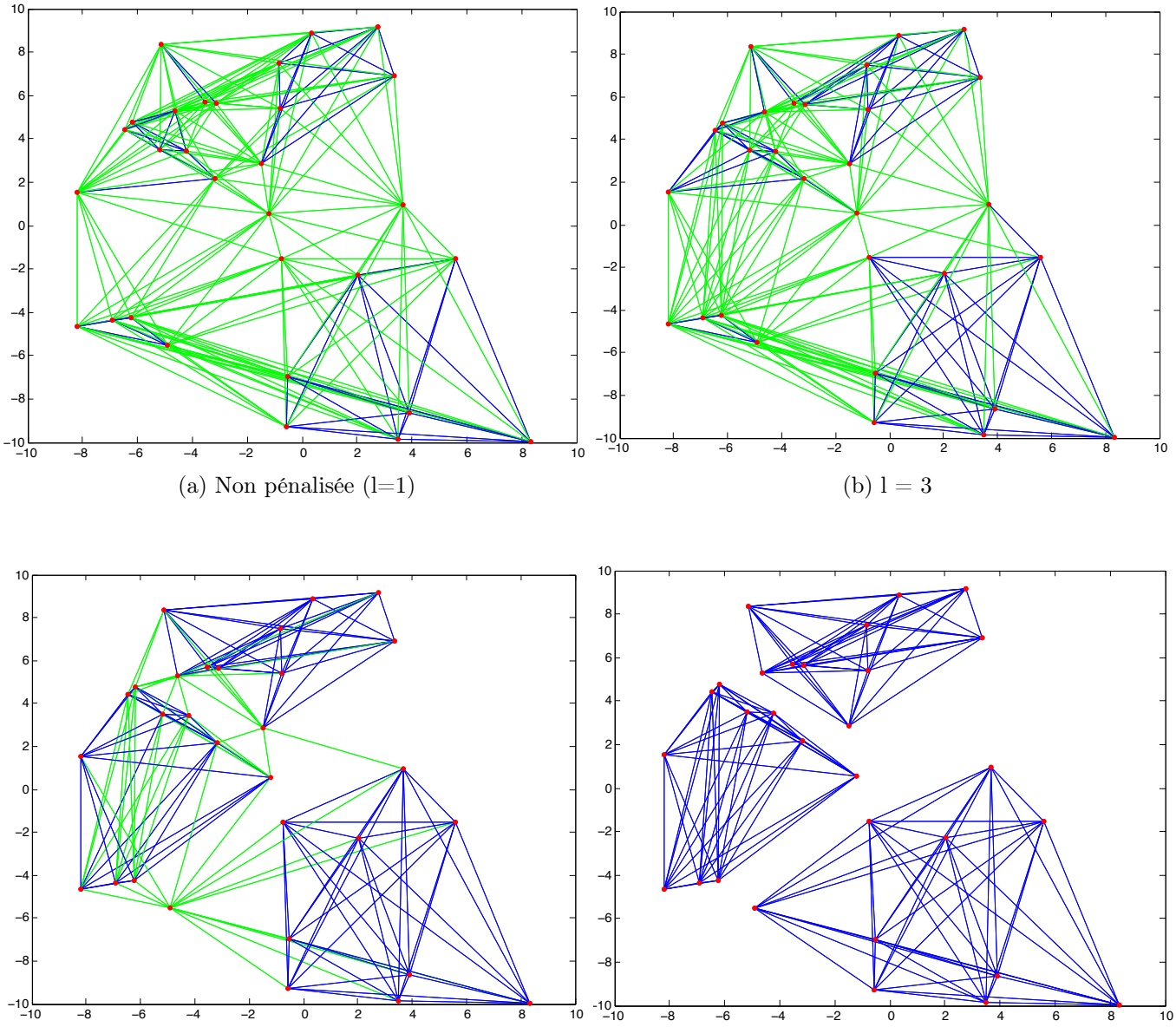


(a) Solution convenable(12 points) (b) Solution incompatible (30 points)

FIGURE 7 – Penalisation des partitions deséquilibrées (en vert les entre 0.1 et 0.9 strictement)

- La configuration ainsi obtenue n'est pas un optimum du problème initial, mais elle peut fournir une solution (par exemple pour ensuite construire un Branch and Bound
- Cette pénalisation des plus grosses arêtes correspond dans notre cas à un facteur d'"équité" : on s'interdit d'avoir de trop grandes arêtes même si cela minimise la distance globale à parcourir.
- L'étude théorique des raisons asymptotiques de la convergence ainsi que des vitesses de convergence de cette méthode nous semble très intéressante, malheureusement par manque de temps nous n'avons pu l'aborder.

4.4 Résultats



(a) Non pénalisée (l=1)

(b) l = 3

(c) l=5

(d) l=7

FIGURE 8 – Effet de la pénalisation sur un même ensemble de 30 points

V Conclusion

Les résultats obtenus sont prometteurs, puisqu'ils nous permettent de traiter le cas de la bissection de façon exacte en un temps raisonnable.D'un point de vu pratique, l'algorithme renvoi la solution optimale en temps polynomial dans 80% des cas. Même si on ne peut savoir a priori si la solution renvoyée dès la première feuille est la solution optimale, un piste est de chercher une borne théorique qui majore l'erreur alors commise. De plus la variation de la précision en fonction du choix du premier point sera discuté en soutenance orale. Nous ne sommes malheureusement pas parvenus à établir une borne théorique pour encadrer la valeur de ce résultat.

Pour le cas de la K-section, l'algorithme SDP donne des résultats satisfaisants dans certains cas, mais nous manquons là encore de théorie pour encadrer les résultats et affirmer avoir résolu le problème.

VI Bibliographie

[1] Alexander A. Ageev and Maxim I. Sviridenko « An Approximation Algorithm for Hypergraph Max k-Cut with Given Sizes of Parts" in BRICS, 1999.

[2] Etienne De Klerk, Dmitrii Pasechnik, Renata Sotirov,And Cristian Dobre " On Semidefinite Programming Relaxations Of Maximum K-Section"

[3] Alan Frieze Carnegie Mellon University, Mark Jerrum University of Edinburgh. "Improved Approximation Algorithms for MAX k-CUT and MAX bissection ". June 1994

[4] Gunnar Anderson , Royal Institute of technology Sweden. "An approximation algorithm for max p-Section"

VII Annexe - Code MATLAB

Dans tous les programmes qui suivent, on utilise la fonction de génération aléatoire de points dans $[-10; 10]^2$ qui suit :

```
function Coordonnees=GenerationPoints(N)
    Coordonnees = 20*rand(2,N)-10;
end
```

Nous utiliserons également la fonction Trace pour visualiser les résultats :

```
function [] = Trace(Coordonnees,M,N)

clf;
%Representation des sommets du graphe
plot(Coordonnees(1,:),Coordonnees(2,:),'.r');
hold on;

%Representation des sommets relies par la matrice M
%(bleu si Mij = 1 et vert si Mij = -1)
for i=1:N
    for j=1:N
       if M(i,j) == 1
           plot([Coordonnees(1,i) Coordonnees(1,j)],
              [Coordonnees(2,i) Coordonnees(2,j)],'-b');
           hold on;
       end
       if M(i,j) == -1
           plot([Coordonnees(1,i) Coordonnees(1,j)],
              [Coordonnees(2,i) Coordonnees(2,j)],'-g');
           hold on;
       end
    end
end
plot(Coordonnees(1,:),Coordonnees(2,:),'.r');
hold on;
end
```

7.1 Programme de bissection par Branch and Bound

```
function []=bissection

BAndB()

global Coordonnees
global Distances
global Borne
```

```
global DistTotMin
global PlusProchesVoisins
global N
global IndDistOrd
global DistOrd
global coupe
N=16;
coupe = 0;
    %Coordonnees = [-5 -4 4 6 6 7;-4 -5 -4 6 5 4];
    Coordonnees=GenerationPoints()
    Distances = CalculDistances(Coordonnees);

    DistancesVect = zeros(1,N^2);
    for i=1:N
        for j=1:N
            DistancesVect(N*(i-1)+j) = Distances(i,j);
        end
    end
    [DistOrd,IndDistOrd]=sort(DistancesVect);

    PlusProchesVoisins = CreePlusProchesVoisins(Distances);
    DistTotMin = EstimDistTotMin(PlusProchesVoisins,N/2);

    Poule1=ones(1,N/2);

    CurrentValue=100000;
    Borne1=Distances(1,:)*(1+(N/2-2)/(2))+ DistTotMin;
    [Borne,k]=sort(Borne1);
    Parents = [1];
    AntiParents = [];
    [k,Borne]=EnleverLesParents(k,Borne,Parents);

    distTotAntiPar = 0;
    [CurrentValue,Poule1]=MAJMilieuArbre(CurrentValue,Poule1,Borne,k,
                                Parents,AntiParents,0,distTotAntiPar);
    toc();

    M = zeros(N,N);
    for i=1:N
        for j=1:N
            if ismember(i,Poule1) && ismember(j,Poule1)
                M(i,j) = 1;
            end
            if ismember(i,Poule1)==0 && ismember(j,Poule1)==0
                 M(i,j) = 1;
            end
        end
    end

    Trace(Coordonnees, M,N)
    coupe
    CurrentValue
```

```
end
```

```
%Mise a jour Milieu arbre
function [CurrentValue,Poule1]=MAJMilieuArbre(CurrentValue,Poule1,Borne,k,
                        Parents,AntiParents,DistTotPar,distTotAntiPar)
    global P
    global DistTotMin
    global N
    global Distances
    global IndDistOrd
    global DistOrd
    global coupe

    [t,taillek] = size(k);
    [r,s]=size(Parents);

     while taillek>=1 && s<=(N/2-1) && Borne(1)<CurrentValue
            % On continue a tester seulement si la Borne est inferieure a la CV

            NewParents=[Parents,k(1)];
            DPar3=DistTotRec(Parents,DistTotPar,k(1));
            elemEnleve = k(1);
            [k,taillek]=Reduirek(k,taillek);
            if s + taillek >= N/2
                 [CVACT,kkkk]=CalculBorne(taillek,NewParents,
                        AntiParents,DPar3,distTotAntiPar,k);
                [CurrentValue,Poule1]=MAJMilieuArbre(CurrentValue,Poule1,CVACT,
                            kkkk,NewParents,AntiParents,DPar3,distTotAntiPar);
            end
            distTotAntiPar = DistTotRec(AntiParents,distTotAntiPar,elemEnleve);
            AntiParents = [AntiParents,elemEnleve];

     end% On itere sur les indices de la liste k
        % On passe a l'iteration suivante
     if s==N/2    % On est presque arrive a une feuille,
                %on calcule les Value de chacunes des poules possibles
            DistTotComplPar = 0;
            Value = 0;
            for i=1:N
                for j=(i+1):N
                    if ismember(i,Parents) || ismember(j,Parents)
                    else
                        DistTotComplPar = DistTotComplPar + Distances(i,j);

                    end
                end
            end
            Value = DistTotPar + DistTotComplPar;
            if Value<CurrentValue
                CurrentValue = Value;
                Poule1 = Parents;
```

```
            end
     end % On est encore "au milieu" de l'arbre on appelle
     % recursivement sur chaque nouveau noeud dans l'ordre adequat

end
```

```
% Creation de la matrice Distances
function Distances=CalculDistances(Coordonnees)
global N

    Distances= zeros(N,N);
    for i=1:N
        for j=1:N
            Distances(i,j)=sqrt((Coordonnees(1,i)-Coordonnees(1,j))^2+
                                (Coordonnees(2,i)-Coordonnees(2,j))^2);
        end
    end

end
```

```
%Definition du tableau (N,N%2) des sommes des distances de
% i a ses j plus proche voisins.
function PlusProchesVoisins= CreePlusProchesVoisins(Distances)
global N
    PlusProchesVoisins=zeros(N,N/2);
    for p=1:N/2
        for o=1:N
            [ValPPV,indicesPPV]=sort(Distances(o,:));
            S=0;
            for q=1:p
                S=S+ValPPV(q);
            end
            PlusProchesVoisins(o,p)=S;
        end
    end

end
```

```
%Estimation de la distance mini a l'interieur d'une poule DistTotMin
function DistTotMin=EstimDistTotMin(PlusProcheVoisins,k)
global N
    [SumDist,IndSumDist]=sort(PlusProcheVoisins(:,N/2));
    S=0;
    for t=1:k
        S=S+SumDist(t);
    end
    DistTotMin=S/2;
```

```
end
```

```
%Enleve les elements des listes k et Borne correspondant aux indices de Parents
function [kk,CV]=EnleverLesParents(k,Borne,Parents)
    [un,taillek]=size(k);
    kk=[];
    CV=[];
    for p=1:taillek
        if ismember(k(p),Parents) == 0
            kk=[kk,k(p)];
            CV=[CV,Borne(p)];
        end
    end

end
```

```
% Fonction reduire la liste k (liste des points restant "indecis")
%en enlevant le dernier point
function [k,taillek]=Reduirek(kdonnee,taillekdonnee)
taillek=taillekdonnee-1;
k=zeros(1,taillek);
for j=1:taillek
    k(j)=kdonnee(j+1);
end
taillek=taillekdonnee-1;
end
```

```
%Fonction calcul la distance totale entre [Parents,j]
% sachant la distance totale entre Parents
function S=DistTotRec(Parents,DistTotPar,j)
global Distances
    S=DistTotPar;
    for i=Parents
        S=S + Distances(i,j);
    end
end
```

```
%Calcul de la Borne
function [Borne,newk]=CalculBorne(taillek,Parents,AntiParents,
                        DistTotPar,distTotAntiPar,k)
global Distances
global P

global DistTotMin
global IndDistOrd
global DistOrd
```

```
global N

    newk=zeros(1,taillek);
    Borne=zeros(1,taillek);
    BorneInt=zeros(1,taillek);

    %--- Estimation de la poule Parents + element g
    distancesEntreParents=zeros(1,taillek);
    for g=1:taillek
        distancesEntreParents(g)=DistTotRec(Parents,DistTotPar,k(g));
        distancesEntreAntiParents(g)= DistTotRec(AntiParents,distTotAntiPar,k(g));
    end

    [DEPtriee,Ind]=sort(distancesEntreParents);
    [DEAPtriee,Ind2]=sort(distancesEntreAntiParents);
    [r,s2]=size(Parents);
    [r,s3]=size(AntiParents);
    points = N/2 - s2;    %Points restant a mettre
    pointsAnti= N/2 - s3;
    SParents = 0;
    SAntiParents = 0;
    for i=1:points

        SParents= SParents + DEPtriee(i) - DistTotPar;
    end
    for i=1:pointsAnti

        SAntiParents= SAntiParents + DEAPtriee(i) - distTotAntiPar;
    end

    Sppa = SommePlusPetitesAretes(Parents,(points-1)*(points)/2);
    SppaAntiPar = SommePlusPetitesAretes(AntiParents,(pointsAnti-1)*(pointsAnti)/2);

    % --- Estimation de la poule Antiparents
    BorneAntiPar = max(SAntiParents + SppaAntiPar + distTotAntiPar,DistTotMin);

    for i=1:taillek
        BorneInt(i) = distancesEntreParents(i)+ SParents + Sppa + BorneAntiPar;
    end


    [Borne, kRand] = sort(BorneInt);

    for i=1:taillek
        newk(i) = k(kRand(i));
    end

end
```

```
% Calcul du tableau somme des i plus petites aretes.
function distSumMin=SommePlusPetitesAretes(Parents,i)
```

```
IntDistOrd
global DistOrd
global N

    distSumMin=0;
    Nb=0;
    for j=1:(2*i+N) % La matrice Distance est symetrique!! =>  gain facteur 2

        distSumMin = distSumMin + DistOrd(j);
    end
    distSumMin=distSumMin/2;
end
```

7.2 Programmes SDP

bissection

```
function [BorneSDP] = TestPenalisation()
global N
N=30;
K=2;
Penalisation =10^4;
precision = 0.2;


Coordonnees = GenerationPoints();

% Creation de la matrice des distances C
C= zeros(N,N);

for i=1:N
    for j=1:N
        if j==i
        else
        C(i,j)=Penalisation + sqrt((Coordonnees(1,i)-Coordonnees(1,j))^2+
                                   (Coordonnees(2,i)-Coordonnees(2,j))^2);
        end
    end
end

% Calcul de la somme totale des aretes
Tot = 0;
for i=1:N
    for j=1:N
        Tot = Tot + C(i,j);
    end
end
Tot = Tot/2;

%Creation du Laplacien la matrice de couts du graphe
```

```
L = zeros(N,N);
for i=1:N
    for j=1:N
        L(i,j) = - C(i,j);
    end
end
for i=1:N
    L(i,i) = 0;
    for j=1:N
        L(i,i) = L(i,i) + C(i,j);
    end
end


%Solveur SDP CVX
cvx_begin sdp
    variable X(N,N) symmetric
    X == semidefinite(N);
    diag(X) == 1;
    maximize( sum(sum(L.*X)) );
cvx_end

M=zeros(N,N);
for i=1:N
    for j=1:N
        if X(i,j) > 1-precision
            M(i,j)=1;
        end
        if X(i,j) < precision-1
            M(i,j)=-1;
        end
    end
end


Trace(Coordonnees, M);
end
```

K-section

```
function [BorneSDP] = KsectionSDP()
N=30;
K=3;

% Choix des parametres.
precision = 0.2;
penalisation = 10;
puissance = 8;

Coordonnees = GenerationPoints(N);

% Creation de la matrice des distances C
C= zeros(N,N);

for i=1:N
    for j=1:N
        if j==i
        else
        C(i,j)= sqrt((Coordonnees(1,i)-Coordonnees(1,j))^2+
                    (Coordonnees(2,i) - Coordonnees(2,j))^2)/10;
        C(i,j) = C(i,j)^puissance;
        end
    end
end

Total = 0; % Valeur de la somme totale des aretes du graphe
for i=1:N
    for j=1:N
        Total = Total + C(i,j);
    end
end
Total = Total/2;

L = zeros(N,N); % Matrice du programme SDP
for i=1:N
    for j=1:N
        L(i,j) = - C(i,j);
    end
end
for i=1:N
    L(i,i) = 0;
    for j=1:N
        L(i,i) = L(i,i) + C(i,j);
    end
end


cvx_begin sdp % On utilise le solveur CVX
variable X(N,N) symmetric
```

```
X == semidefinite(N);
diag(X) == 1;
for i=1:N
    sum(X(i,:)) == N/K
    % On impose le nombre de voisins a N/K pour chaque sommet i.
end

for i=1:N
    for j=1:N
        X(i,j) >= 0 % On impose la positivite des Xij
    %(le fait qu'ils soient plus petits que 1 est
    % une consequence de la maximisation de L.X
    end
end


maximize( sum(sum(L.*X)) );
cvx_end


M=zeros(N,N);
for i=1:N
    for j=1:N
        if X(i,j) > 1-precision
            M(i,j)=1;
        end
        if X(i,j) > precision && X(i,j) < 1-precision
            M(i,j)=-1;
        end
    end
end


figure(1)
clf
Trace(Coordonnees, M,N);

end
```