

D.E.A. M.I.A.S.H.

Année 2001-2002

Mathématiques, Informatique,

Applications aux Sciences de l'Homme.

Méthodes de partitionnement pour la classification de protéines

Mémoire de D.E.A. réalisé par Jean-François Culus
sous la direction d'Olivier Hudry (E.N.S.T.)

Contents

1	Introduction	3
1.1	Etude du cas facile de la bipartition. (k=2)	3
1.2	Etude du cas k fixé hors de l'instance.	3
1.3	Lien avec la programmation quadratique en variables 0-1.	5
2	Etude d'un premier critère	5
2.1	Présentation.	5
2.2	Une première notion de voisinage.	6
2.3	Un premier algorithme.	7
2.4	Une seconde notion de voisinage.	7
2.5	Un second algorithme.	8
2.6	Les premiers résultats	8
2.7	Un premier emploi de la méthode de bruitage.	8
2.8	Une étude critique de nos voisinages.	8
2.9	De la présence des singletons.	9
2.10	Etude de premières matrices test.	10
2.11	Un algorithme polynomial à garantie de performance pour le problème de la k partition.	11
3	A la recherche d'un autre critère.	14
3.1	La problématique.	14
3.2	Une notion de cohérence probabiliste	15
3.3	Des algorithmes.	16
4	Les méthodes de bruitages.	18
4.1	Présentation de ces méthodes.	18
4.2	Le bruitage des données	19
4.3	Le bruitage des variations.	19
5	De la simulation informatique.	20
5.1	Introduction.	20
5.2	Les premiers tests.	20
5.3	Des améliorations du programme second voisinage.	21
5.4	Les bruitages.	23
5.5	Pourquoi le bruitage des données donne t'il de si mauvais résultat ?	24
5.6	Concernant la notion de cohérence.	26
6	Les résultats des tests.	32
6.1	Relativement au critère de séparation.	32
6.2	Sur la notion de cohérence.	34
7	Que reste il à faire?	38
8	Bibliographie.	39

1 Introduction

Le problème ici étudié est le partitionnement d'un graphe valué par des poids positifs en k classes. Ce problème est issu en partie de travaux de biologie relatifs au séquençage de l'ADN. Grace à diverses méthodes, les biologistes établissent des degrés d'interaction entre deux protéines, représentant par exemple le pourcentage des séquences communes d'ADN de ces deux protéines.

Pour de plus amples informations sur ces sujets, on se référera aux articles d'Alain Guénoche [Gue]. Les résultats sont alors mis sous la forme d'un graphe, dont les sommets codent les protéines, et les arcs valués représentent l'interaction d'une protéine sur l'autre. Notons qu'à ce niveau, le graphe peut être non symétrique. On note $p(i, j)$ le poids (ou l'interaction) de l'arc (i, j) .

A partir donc de ce graphe ayant n sommets (x_1, x_2, \dots, x_n) , on cherche à réaliser une k -partition (X_1, X_2, \dots, X_k) de ces sommets, k étant un entier fixé. On notera que ce problème est NP-complet (sauf dans le cas facile que sont les instances avec $k=2$, ou plus généralement dans les cas où k est spécifié hors de l'instance.).

En effet, on peut transformer le problème de la recherche d'une clique maximale à ce dernier.

Le problème de décision associé à max clique est le suivant: *Etant donné un graphe G et un entier M , est ce que le plus grand sous graphe complet (ie la plus grande clique) de G contient exactement M sommets?* Nous allons réduire une instance quelconque de Max clique à une instance particulière de k -part: si les arrêtes sont valuées par $(0,1)$, alors une k partition de G revient à rechercher une k partition des sommets maximisant le nombre d'arrêtes ayant leurs deux extrémités dans la même classe de partition.

Si G dispose d'une clique de cardinal $M = |V| - (k - 1)$, alors toutes solutions du problème de k -partition exhibera celle ci comme l'une de ces classes de partitions.

Ainsi, réduit on le problème Max Clique au problème de la k partition, en choisissant k tel que $M = |V| - (k - 1)$... d'où ce dernier est NP-Complet.

1.1 Etude du cas facile de la bipartition. ($k=2$)

On souhaite donc effectuer une bipartition du graphe $G=(V,E)$.

Pour ce faire, considérons un sommet quelconque x dans cet ensemble; on utilise alors l'algorithme de flot, afin de déterminer une coupe minimale isolant x (sommets source) d'un sommet y (puits). En faisant varier y parmi les autres sommets du graphe (il y en a $n-1$ autre donc), on en déduit alors au final une coupe d'indice de séparation minimal parmi toutes les coupes envisagées. Ainsi, notre algorithme de bipartition, basé sur l'algorithme de flot, permet il de donner une séparation optimale¹.

1.2 Etude du cas k fixé hors de l'instance.

Bien que le problème soit NP-Complet lorsque k est un entier faisant partie de l'instance, le problème k -cut avec k fixé hors de l'instance est lui polynomial, depuis l'article d'O.Goldschmidt et D.S. Hochbaum. On ne détaillera pas ici le pourquoi, mais esquissons l'algorithme sous jacent.

¹Nous savons de plus que ce problème est polynomial

- Le cas de la tripartition.

Ces algorithmes sont basés sur une énumération des solutions possibles. Notons $w^* = \infty$.

Phase 1.

★ Pour tout sommet $v \in V$, on note w_1 sa somme des degrés, et w_2 le poids de la plus petite bipartition de $V - v$.

★ Si $w_1 + w_2 < w^*$, alors $w_1 + w_2 = w^*$ et on sauvegarde temporairement la solution associée.

Phase 2. (La phase 1 permet de trouver la tripartition minimale ayant un singleton; ici, on considère les autres tripartitions).

★ Considérons successivement tout les quadruplets de sommets (s_1, s_2, t_1, t_2) . On introduit deux sommets virtuels s et t que l'on relie, s à s_1 et s_2 , t à t_1 et t_2 avec des arrêtes de poids infinies. On détermine alors une coupe minimale séparant s et t , puis une autre séparant t_1 et t_2 ; si w le poids total de ces deux coupes est inférieur à w^* , alors on remplace w^* par w et on sauvegarde la solution associée.

Voilà l'algorithme dans le cas (simple) de la tripartition. Ensuite, cet algorithme fonctionne par récurrence. Pour une k partition:

Phase 1: On débute avec $w^* = \infty$

Considérons tout les ensembles S possibles avec $|S| = i$, pour i variant de 1 à $k-3$. Pour chacun d'eux, on note w_1 la somme des poids des arrêtes séparant S de $V-S$. Par récurrence, on trouve w_2 le poids de la plus petite $k-1$ partition de l'ensemble $V-S$.

Si $w_1 + w_2 < w^*$, alors on sauvegarde w dans w^* et on retiens la solution associée. Ainsi, on viens de balayer l'ensemble des partitions ayant leurs première classe de partition (S) de cardinal inférieur ou égale à $k-3$. Maintenant on va s'intéresser aux autres ...

Phase 2.

Considérons successivement tout les sous ensemble de sommets K de taille $(k-2)+(k-1)=2k-3$. Pour chacun des $C_{|K|}^{k-2}$ ensemble S , on note T le complémentaire de S dans l'ensemble K , et on réalise une coupe (dite maximal minimum (S,T)-cut), séparant les ensemble S et T , coupe réalisée dans le graphe G . Le pourquoi de cette coupe, l'idée initiale, est qu'il y a $k-1$ autres classes, non vide, donc quand on recherche ce premier ensemble, qui contient donc dans cette phase au moins $k-2$ sommets, on tente de le séparer d'avec au moins $k-1$ autres sommets ... d'où le cardinal de l'ensemble K , représentant donc le coeur de cette première classe (celle qui sera de cardinal au moins $k-2$), et un élément de chacune des $k-1$ autres classes, donc $2k-3$ éléments en tout!

La classe contenant le coeur (ie les $k-2$ éléments de S) est mise à part, et les autres sommets (donc ceux contenant le sous graphe engendré par les $k-1$ sommets de T) se voient réappliquer l'algorithme de $k-1$ cut optimale. On note donc encore w_1 le poids de cette max min (S,T)-cut, et w_2 le poids de la $k-1$ cut de l'ensemble contenant le sous graphe T .

On regarde ensuite si $w^* > w_1 + w_2$; si oui, on sauvegarde le tout en lieu et place de la solution anciennement optimale et de w^* ...

Complexité de cet algorithme: Si $T(n,m)$ désigne le temps mis pour réaliser une coupe minimale séparant deux sommets s et t d'un graphe quelconque ayant n sommets et m arrêtes, alors la complexité de cet algorithme est en $O(n^{k^2/2} - 3k/2 + 4) * T(n, m)$.

1.3 Lien avec la programmation quadratique en variables 0-1.

Le but ici est de donner une forme équivalente du problème de la k-partition. Associons à chaque sommet $x(i)$ du graphe un k-vecteur à composante 0-1:

$\begin{pmatrix} \alpha_1^{(i)} \\ \alpha_2^{(i)} \\ \vdots \\ \alpha_k^{(i)} \end{pmatrix}$. La convention sera que seul l'un des $(\alpha_j^{(i)})_{1 \leq j \leq k}$ sera égal à 1, les autres étant donc nuls; le sommet $x(i)$ appartiendra alors à la classe de partition X_{j_0} , si et seulement si $\alpha_{j_0}^{(i)} = 1$.

Remarquons que le produit scalaire du vecteur associé à $x \in X_i$ avec le vecteur associé à $y \in X_j$ vaut 1 si et seulement si les sommets x et y appartiennent à la même classe de partition (ie $i = j$).

Nous en déduisons donc que le problème revient à minimiser la fonction:

$$\sum_{1 \leq i, j \leq n} p(i, j) * \left(1 - \begin{pmatrix} \alpha_1^{(i)} \\ \alpha_2^{(i)} \\ \vdots \\ \alpha_k^{(i)} \end{pmatrix} \cdot \begin{pmatrix} \alpha_1^{(j)} \\ \alpha_2^{(j)} \\ \vdots \\ \alpha_k^{(j)} \end{pmatrix} \right)$$

Sous les $n+k$ contraintes linéaires suivantes:

$\forall 1 \leq i \leq n, \sum_{l=1}^k \alpha_l^{(i)} = 1$ (chaque sommet n'appartient qu'à une seule classe).

$\forall 1 \leq j \leq k, \sum_{i=1}^n \alpha_j^{(i)} \geq 1$ (aucune des k classes n'est vide).

Ce problème est donc bien un problème de programmation quadratique en les $n * k$ variables $(\alpha_j^{(i)})_{1 \leq i \leq n; 1 \leq j \leq k}$; comme bien souvent en programmation quadratique, c'est un problème difficile à résoudre pour de grandes tailles de manière exacte (ici $n=100$).

Remarquons enfin, que, sous cette forme, il semble bien évident que tenter de minimiser la somme des poids inter classe ou maximiser la somme des poids intra-classe revient au même.

Nous allons donc par la suite concentrer nos efforts sur des méthodes de descentes à voisinage, donnant au final une solution localement optimale, que l'on espère bonne. Ces méthodes partiront toutes initialement d'une solution tirée au hasard.

2 Etude d'un premier critère

2.1 Présentation.

Au vue du problème, il semble naturel de s'intéresser au critère de séparation inter classes. Formalisons ceci.

On appellera indice de séparation des classes X_i et X_j la somme des poids des arcs ayant leur sommet initial dans X_i , et leur sommet terminal dans la classe X_j . Ainsi, $Sep[X_i, X_j] = \sum_{x \in X_i, y \in X_j} p(x, y)$.

Définition 1 On définit la séparation (globale) du graphe par la somme des

séparations des différentes classes; ainsi, on a:

$$Sep = \sum_{1 \leq i, j \leq k} Sep(X_i, X_j).$$

On remarque alors que cette définition implique une certaine symétrie dans le problème: en effet, le critère final considère en même temps $Sep(X_i, X_j)$ et $Sep(X_j, X_i)$. Notons bien que cela n'enlève rien à l'intérêt ² dudit critère, puisqu'il semble bien normal de compter à la fois le poids de l'arc (x, y) et aussi de l'arc (y, x) si x et y sont deux sommets de classes de partitions différentes.

Ainsi, pour commodités et simplifications, on considérera dans la suite la matrice symétrique S , obtenue à partir de la matrice des poids en la symétrisant. Le critère de séparation se réécrit alors pour cette nouvelle matrice:

$Sep = \sum_{1 \leq i \leq j \leq k} Sep(X_i, X_j) = \frac{1}{2} \sum_{1 \leq i, j \leq k} Sep[X_i, X_j]$. Le but dans la suite sera alors bien entendu de minimiser ce critère de séparation.

2.2 Une première notion de voisinage.

Notre but est de réaliser différents programmes approchant la solution optimale basés sur des algorithmes à voisinage. Ainsi, notre premier soucis est celui de définir une première notion de voisinage. La plus simple consiste à poser:

Définition 2 Deux k -partitions (X_1, X_2, \dots, X_k) et $(X'_1, X'_2, \dots, X'_k)$ seront dites voisines si elles ne diffèrent que d'un unique élément, donc s'il existe deux indices i et j dans $\{1 \dots n\}$ et un sommet x tels que $\forall l \neq i, j, X_l = X'_l; X_i - x = X'_i; X_j = X'_j - x$.

Nous allons définir alors une matrice³ Gain, de taille (n, k) , telle que: $Gain_{x,i} = \sum_{y \in X_j} p(x, y)$. Le coefficient $Gain_{x,j}$ représenteront simplement l'interaction entre le sommet x et la classe X_j (que celle ci contienne ou non le sommet x , puisque $p(x,x)=0$). Considérons donc un sommet x quelconque, et (X_1, X_2, \dots, X_k) une solution courante. Supposons que $x \in X_i$.

A quelles conditions, afin que le critère relatif à la solution courante diminue, le sommet x a t'il intérêt à être mis dans la classe X_j ?

Si le sommet x passe dans la classe X_j , le gain au niveau du critère et provenant du fait que x appartiendra alors à la classe X_j sera de $Gain_{x,j}$, alors que la perte au niveau du critère sera elle égale à $Gain_{x,i}$, puisque le sommet x n'appartiendra alors plus à la classe X_i .

Proposition 1 La solution $(X'_1, X'_2, \dots, X'_k)$ sera meilleure pour le critère de séparation si et seulement si $Gain_{x,j} \leq Gain_{x,i}$.

Enfin, si x est déplacé de la classe X_i vers la classe X_j , alors, toutes les séparations entre l'une de ces deux classes et une autre classe tierse s'en trouveront perturbés; X_i perdra les interactions relatives à x avec les classes tierces, alors que X_j les gagnera [cf figure 1.].

²escompté

³remarquons que l'on suppose ici que les interactions d'un sommet avec lui même est toujours nul.

Proposition 2 (Mise à jour de Sep et Gain) *Si le sommet x passe de la classe X_i dans la classe X_j , alors les matrices Sep et Gain relatives à la nouvelle solution sont donnés par:*

$$\forall 1 \leq l \leq k, l \neq \{i, j\}, Sep(X_i, X_l) \leftarrow Sep(X_i, X_l) - Gain_{x,l};$$

$$Sep(X_j, X_l) \leftarrow Sep(X_j, X_l) + Gain_{x,l}. \text{ Il en va aussi de même avec les symétriques } Sep(X_l, X_i \text{ ou } j).$$

$$\forall y \neq x, Gain_{y,i} \leftarrow Gain_{y,i} - p(x, y); \quad Gain_{y,j} \leftarrow Gain_{y,j} + p(x, y).$$

2.3 Un premier algorithme.

Nous avons donc maintenant la notion de voisinage, ainsi qu'un critère simple afin de savoir s'il est intéressant d'effectuer le changement de classe pour le sommet x ; reste donc à choisir la stratégie pour mener cette descente.

définir la matrice Gain à partir de la matrice S.

★ pour x variant de 1 à n ; (x représente un sommet).

★ pour i variant de 0 à k ; (i représente une classe).

★ retenir le minimum des $Gain_{x,i}$.

★ Déplacer le sommet x dans sa meilleure classe d'accueil (ie celle minimisant le Gain).

Répéter cette boucle tant que l'on bouge au moins un sommet de classe.

Note: On effectue un balayage sur les sommets et non sur les classes, cela afin de tenter d'éviter l'hémorragie d'une classe. Cette considération cyclique des sommets peut s'apparenter, de loin, à une méthode tabou.

On remarquera aussi que l'on effectue le meilleur choix de classes (parmi celles constituées) pour le sommet x ; nous verrons plus tard certains des inconvénients de ces choix.

2.4 Une seconde notion de voisinage.

Les méthodes de descentes à voisinage dépendent de manière cruciale de ladite notion de voisinage. Aussi, une seconde notion peut-elle être intéressante...

Définition 3 *Deux k -partitions (X_1, X_2, \dots, X_k) et $(X'_1, X'_2, \dots, X'_k)$ seront dites voisines si elles ne diffèrent l'une de l'autre que par 2 classes; ie s'il existe i et j tel que $\forall l \neq \{i, j\}; X_l = X'_l$.*

La mise en oeuvre pratique dans un algorithme de ce voisinage est assez simple: considérons (X_1, X_2, \dots, X_k) une solution courante. On considère alors deux classes de partition au hasard: X_i et X_j ; on les fusionne entre elles, et on effectue une bipartition de $X_i \cup X_j$.

Comment faire cette bipartition?

En fait, assez simplement, grâce au préliminaire: considérons un sommet x dans cet ensemble, on utilise l'algorithme de flot, afin de déterminer une coupe minimale isolant x (sommet source) du sommet y (puits). En faisant varier y parmi les autres sommets de $X_i \cup X_j$, on en déduit alors au final la coupe d'indice de séparation minimal parmi toutes les coupes envisagées. Ainsi, notre algorithme de bipartition, basé sur l'algorithme de flot, permet il de donner la séparation optimale⁴.

⁴et en plus, en un nombre polynomial d'appel à l'algorithme de flot.

2.5 Un second algorithme.

Testfin=0; Tant que Testfin $\leq k^2$ répéter:

★pour i variant de 1 à k;

★ pour j variant de i+1 à k.

★ appliquer la bipartition à l'ensemble $X_i \cup X_j$.

★ Si la partition est identique à la partition initiale, alors Testfin+=1, sinon Testfin=0.

2.6 Les premiers résultats

A ce niveau ci, considérons donc déjà deux matrices tests naïves: La première sera une matrice (15,15) sous forme solution (3 blocs de 5), un peu perturbée (ie quelques coefficients hors des blocs diagonaux non nuls). La seconde matrice est une matrice (50,50) tirée aléatoirement, dont les coefficients sont entre 0 et 20; on cherche pour celle ci des 3-partitions, 5 partitions, et 7-partitions.

Les résultats sont alors décevants:

La première matrice est bien obtenue, mais cela est à postèriori dûe à la taille très restreinte de la matrice.

Déjà, on constate, en répétant l'algorithme pour des solutions initiales différentes, que la forme de la solution obtenue, en particulier pour la seconde matrice, est toujours identique: une grosse classe, et k-1 singletons.

Ensuite, la descente avec première notion de voisinage est très loin de l'indice du critère obtenu par l'autre algorithme, utilisant la bipartition.

Néanmoins, pour le moment, nous avons encore espoir d'obtenir de meilleurs résultats en utilisant les méta-heuristiques telles que les méthodes de bruitages...

2.7 Un premier emploi de la méthode de bruitage.

Cette méthode consiste à bruite les données (ie les poids des arrêtes de la matrice). Appliquons donc un bruit à la matrice S: $S_b=S+B$; on effectue sur la matrice S_b une descente grâce au second algorithme, puis avec la solution trouvée, on effectue, de façon systématique une descente (toujours avec second voisinage) à partir de la solution obtenue pour le problème non bruité.

Quelle n'est pas notre surprise en constatant que même pour de fort bruit B, la solution finalement obtenue après les descentes systématiques est identique! Le bruitage par oubli des données est aussi tenté, mais sans plus de succès... d'où la légitime question: qu'est ce qui ne va pas dans nos algorithmes et qui force certaines solutions, en l'occurrence, celle donnant de nombreux singletons?

2.8 Une étude critique de nos voisinages.

Pour la première notion de voisinage.

Ce voisinage présente un inconvénient majeur: un seul sommet peut changer de classe à chaque coup. Ainsi, peut il y avoir blocage, lorsqu'il serait nécessaire par exemple de faire passer plusieurs sommets en même temps dans une autre classe (cf figure 2).

Pour la seconde notion de voisinage.

Le défaut vient ici du fait que lors de la bipartition, on oublie tous les autres sommets. Ainsi, imaginons qu'une partie du graphe soit bien cohérente (graphe

complet suffisamment valué), et qui serait divisée en 3 classes de partition suite au tirage de la solution aléatoire; à chaque fois que l'on tente une fusion entre deux des classes de partition, les intersections entre ces classes et le sous graphe complet se voit être pénalisé par l'absence des sommets du graphe hors des 2 classes de partition. Un exemple de ce phénomène est donné sur la figure 3.

• *Quelques idées pour tenter de contrebalancer ces phénomènes négatifs:*

Pour la seconde notion de voisinage: déjà, le fait que l'on regarde de façon systématique les fusions X_i et X_j implique un biais: on risque ainsi artificiellement de forcer une classe à devenir de plus en plus grosse ... En effet, par exemple X_1 sort grandi de sa rencontre avec X_2 ; il aura alors tendance à remporter face à X_3 ... si par exemple il sort avec uniquement un singleton à l'issue de sa rencontre avec X_4 , on attendra que ce soit au tour d' X_4 d'affronter les autres classes pour retrouver ce phénomène.

Pour contrebalancer ce fait, il suffit de fusionner $X_{hasard(i)}$ et $X_{hasard(j)}$, mais il sera nécessaire à l'issue de cette descente d'effectuer une autre descente systématique avec l'algorithme premier afin de s'assurer d'obtenir au final une solution qui soit bien un minimum local.

Enfin, remarquons que si un sous graphe cohérent est divisé en 2 partitions, alors on a toutes les chances de le retrouver dans son intégrité au final, puisque lors de la fusion deux classes, la bipartition rétablira le sous graphe.

Hélas, il n'en va pas de même si une classe cohérente est divisée en trois; on ne peut alors plus garantir de retrouver à un moment donné l'intégralité du sous graphe dans une même classe. Une idée est alors d'alterner les fusions par bipartition avec des phases de fusions / éclatement: on fusionne deux petites classes, et la plus grosse classe est coupée en deux par la bipartition. L'algorithme de cette descente pourrait être du type:

Tirer au hasard deux sommets.

★ si les deux sommets sont dans deux classes distinctes, fusionner celle-ci, et appliquer l'algorithme de bipartition pour réobtenir deux nouvelles classes de séparation minimale.

★ si les deux sommets sont dans la même classe: fusionner deux autres petites classes, et appliquer une bipartition à la classe contenant les deux sommets tirés. Nous allons, dans un avenir proche, tenter d'étudier le plus précisément possible une structure de matrice test, afin d'obtenir une matrice (non triviale) dont on connaisse la solution optimale, afin de tester à posteriori nos améliorations.

2.9 De la présence des singletons.

Le but de ce petit paragraphe est de formaliser, pour des graphes particulièrement simples, le sentiment qu'il est préférable, si les arêtes sont de poids assez similaires, d'avoir des singletons et une grosse classe plutôt que des classes ayant des cardinaux proportionnels. A titre d'exemple, si on enlève la condition d'aucune classe vide, alors on obtient trivialement une partition minimisant la séparation en ne formant qu'une seule classe, toutes les autres étant vides: l'indice de séparation de cette solution est alors nulle, ce qui est optimal⁵.

De retour avec la condition de classes non vides, supposons pour simplifier, que l'on dispose d'un graphe complet ayant n sommets et dont toutes les arêtes sont valuées par 1. On considère alors la partition $P = (X_1, X_2, \dots, X_k)$

⁵rappelons que les poids sont positifs ou nuls.

, et on note n_i le cardinal de la classe X_i . On souhaite donc minimiser la fonction: $\sum_{i=1}^{k-1} \sum_{j=i+1}^k Sep(X_i, X_j) = \sum_{i=1}^{k-1} \sum_{j=i+1}^k n_i * n_j$ sous les contraintes: $\sum_{i=1}^k n_i = n$; $n_1 \geq n_2 \geq \dots \geq n_k \geq 1$.

Comme $Sep(X_i, X_j) = Sep(X_j, X_i) = n_i * n_j$, on remarque que:

$$2 * \sum_{i=1}^{k-1} \sum_{j=i+1}^k n_i * n_j = \sum_{i=1}^k \sum_{1 \leq j \leq k; j \neq i} n_i * n_j.$$

Or,

$$\sum_{i=1}^k \sum_{j=1}^k n_i * n_j = \sum_{i=1}^k n_i * \sum_{j=1}^k n_j = n^2 = \sum_{i=1}^k \sum_{1 \leq j \leq k; j \neq i} n_i * n_j + \sum_{i=1}^k n_i^2$$

Ainsi, minimiser $\sum_{i=1}^k \sum_{1 \leq j \leq k; j \neq i} n_i * n_j = n^2 - \sum_{i=1}^k n_i^2$ revient donc à maximiser la somme des carrés des n_i .

Comme aucune classe n'est vide, on note $n_i = a_i + 1$; avec $a_i \geq 0$. On souhaite montrer que la solution $a_1 = n - k - 1$; $a_2 = a_3 = \dots = a_k = 0$ est optimale; pour cela, formons la différence suivante:

$$X = (n-k-1)^2 + (k-1) - [\sum_{i=1}^k (1+a_i)^2] = (1 + \sum_{i=0}^k a_i)^2 + k - 1 - [\sum a_i^2 + 2 * \sum a_i + k].$$

Par la formule du multinôme, nous savons que: $[\sum a_i]^2 = \sum a_i^2 + 2 * \sum a_i * a_j$. Il en résulte alors que $X = \sum_{i \neq j} a_i * a_j \geq 0$.

On en déduit que non seulement la solution initialement proposée est optimale, mais aussi que la solution sous optimale est celle minimisant la somme $\sum_{i \neq j} a_i * a_j$. La solution sous optimale est alors obtenue pour $a_1 = (n - k - 2)$, $a_2 = 1$; $a_3 = \dots = a_k = 0$.

Les valeurs des solutions optimales et sous optimales sont alors respectivement: $Opt = -k^2 + k * (2n + 1) - 2n$; $SousOpt = -k^2 + 2nk - n - 1$.

Supposons maintenant que les poids de notre graphes soient tous compris entre α et $\epsilon * \alpha$, avec α un entier strictement positif. Nous souhaitons avoir un intervalle pour ϵ dans lequel la solution optimale soit bien de la forme d'une grosse classe de $n-k+1$ éléments, et $k-1$ classes réduites à des singletons. Si tous les coefficients sont dans l'intervalle considéré, alors la solution optimale est majorée par $\epsilon * \alpha * (2nk - k^2 - 2n + k)$, alors que la solution sous optimale elle est minorée par $\alpha * (2nk - k^2 - n - 1)$; ainsi, l'ordre entre les deux solutions est toujours respecté si $\epsilon \leq \frac{2nk - k^2 - n - 1}{2nk - k^2 - 2n + k}$.

2.10 Etude de premières matrices test.

Le but ici est d'élaborer une (ou plutôt une famille) de matrices tests dont nous connaissons la solution optimisant le critère de séparation. Nous partirons d'un motif de base, qui sera un graphe complet dont toutes les arrêtes seront valuées par 10. Par exemple, on considérera dans l'exemple développé des pentagones 10-valué. La question qui se pose alors est: comment peut on perturber les arrêtes entres les différents pentagones afin que la solution optimale reste quand même la découpe formée par les divers pentagones?

Si l'on considère 2 pentagônes, joint par une seule arrete de poids w , on se rends vite compte que la coupe inter-pentagones est justement cette arrete, donc de

poids w . Or d'après la courte étude précédente, comme tout coupe autre que la coupe inter-pentagones passe par l'intérieur d'au moins un pentagone, nous savons que la coupe la plus petite dans ce type de pentagone consiste à isoler un sommet du reste du pentagone: bref, une coupe de poids 40. Ainsi, on peut affirmer que tant que l'arrête unique joignant les 2 pentagones a un poids inférieur à 39, la solution optimale reste les deux pentagones.; il en va de même si il existe plusieurs arretes, dont le poids total est inférieur à 39. [cf figures 4.]

Néanmoins, ne peut pas dire plus? Par exemple, si tout les sommets des pentagones admettent une arrete inter-pentagones (ie une arrete vers l'exterieur de son pentagone). Afin d'homogénéiser le probleme, on considérera 6 pentagones P_1, P_2, \dots, P_6 ; chacun des sommets des pentagones sera alors joint à un autre sommet d'un autre pentagone, de facon tel que tout sommet ait exactement les 4 autre sommets de son pentagone pour voisins, et 1 voisin d'un autre pentagone, et tel que deux sommets d'un même pentagone n'aient pas de voisin dans le même pentagone. Les arrêtes inter pentagones seront alors valuées par w , et on se demande quel sera le plus grand w admissible afin que la solution optimale du problème de 6-partition reste les 6 pentagones. Cette coupe, que l'on voudrait optimale, est de poids $15w$ (car si l'on assimile les pentagones à des points, on a à faire à un hexagone regulier w valué, et il nous faut couper toutes ses arretes).

Quelle est la coupe sous optimale? Facilement, cette coupe passe par l'intérieur d'au moins un pentagone: si elle isole un sommet, elle coute $40+w$. Les coupes suivantes seront alors nécessairement dans ce même pentagones, car si on isole un second sommet, la coupe ne coute plus que $30+w$, puis $20+w$, puis $10+w$, et finalement pour le dernier point du pentagone w . Ainsi, la coupe sous optimale⁶ est elle celle éclatant entièrement un pentagone en 5 singletons, et laissant pour sixième classe le reste du système composé de 5 pentagones. Cette coupe à un poids de $100+5w$. Ainsi, la valeur optimale de w est la plus grande valeur vérifiant $15w \leq 100 + 5w$, soit $w \leq 10$: bref, le $w=9$ est optimal ici.

On pourrait alors généraliser ce petit résultat, mais compte-tenu de la trop grande symétrie du problème⁷, on se contentera de cet exemple.

2.11 Un algorithme polynomial à garantie de performance pour le problème de la k partition.

Dans cette partie, nous allons détailler l'algorithme de bipartition itéré, qui non seulement est polynomial, mais qui en plus permet de garantir de s'approcher de la valeur optimale du problème d'un rapport précisé ci après. Mais pour cela, il va falloir introduire un certain nombre de notations; On peut diminuer ce nombre d'arguments en se reportant par exemple à l'ouvrage de Vazirani.

Détaillons déjà l'algorithme de bipartition itéré. Soit G un graphe; on considère X l'ensemble de ses sommets; on veut effectuer une k partitions de l'ensemble des sommets qui minimise la somme des poids inter-classe. Déjà, nous allons appliquer l'algorithme de bipartition à l'ensemble X tout entier: il va nous déterminer deux ensembles X_1^1 et X_1^2 formant une bipartition optimale de X . On notera par la suite cut_1 la somme des poids des arretes entre X_1^1 et X_1^2 . Ensuite, on applique l'algorithme de bipartition aux ensembles X_1^1 et X_1^2 , et

⁶illustrant donc cette notion d'entraide perdue que l'on évoquait tantôt à propos de la descente par bipartition

⁷bien loin de possibles données réelles

on effectue réellement la plus petite de ces deux coupes; on obtient alors une tripartition X_2^1, X_2^2 et X_2^3 . On notera cut_2 la somme des poids des arêtes inter-classes que l'on vient de couper (ie pas les arêtes déjà prises en compte par cut_1).

Puis ... on recommence ! .. jusqu'à obtenir finalement une k partition (bref, on effectue en tout $k-1$ étapes). Combien avons nous fait de fois appel à l'algorithme de bipartition en tout au cours de ce processus?

Une fois pour obtenir la bipartition, puis 2 fois (sur les 2 sous ensembles obtenus) pour obtenir la tripartition, puis ensuite 3 fois (sur les 3 ensembles de la tripartition) pour obtenir la 4-partition, puis ..., et enfin $k-1$ fois avant que d'effectuer finalement la dernière bipartition, donnant la k partition $X_k^1, X_k^2, \dots, X_k^k$. Donc, en tout, nous avons effectué $\frac{k*(k-1)}{2}$ bipartitions, d'où notre algorithme est polynomial!

Bien, maintenant, intéressons nous à sa garantie. Nous allons noter $\{Y_1, Y_2, \dots, Y_k\}$ une k -partition optimale de X .

Quelques notations:

Définition 4 $\{Y_1, Y_2, \dots, Y_k\}$ est une k -partition ordonnée si c'est une k -partition et que l'on peut l'obtenir par les bipartitions successives (non optimales nécessairement):

On fait une première bipartition de X en Y_1 et $\overline{Y_1}$;

Puis on fait la bipartition de $\overline{Y_1}$ en Y_2 et $\overline{Y_2}$.

Puis celle de $\overline{Y_2}$ en Y_3 et $\overline{Y_3}$, et ainsi de suite jusqu'à Y_{k-1} . Enfin, $Y_k = \overline{Y_{k-1}}$.

Notons⁸ bien à ce niveau que $\overline{Y_2}$ ne désigne pas le complémentaire de Y_2 dans X mais son complémentaire dans $\overline{Y_1}$.

Nous noterons ensuite $Sep(Z, \overline{Z}) = \sum_{x \in Z, y \in \overline{Z}} w(x, y)$.

$cut'_i = Sep(Y_i, \overline{Y_i})$.

Afin de manier quelque peu ces notations, laissons place à quelques exercices⁹: (les ensembles X, Y, Z sont supposés disjoints).

★ Montrer par exemple que $Sep(Y, Z) = Sep(Z, Y)$ en utilisant bien sûr le fait que la matrice des poids est symétrique.

★ Du même niveau de difficulté: $Sep(Y \cup Z, W) = Sep(Y, W) + Sep(Z, W)$.

Pour tout Y_j élément de notre k -partition ordonné et pour tout Z sous ensemble de X , on note:

$Sep(Y_j \cap Z, \overline{Y_j} \cap Z) = \sum_{x \in Y_j \cap Z, y \in \overline{Y_j} \cap Z} w(x, y) = cut'_j|_Z$.

★ Si $\{X_1, X_2, \dots, X_k\}$ forment une k -partition de X , montrer que pour tout sous ensemble Z , $Sep(Z, \overline{Z}) \geq cut|_{X_1} + cut|_{X_2} + \dots + cut|_{X_k}$.

★ La trace sur Z de $\{Y_1, Y_2, \dots, Y_k\}$ une k -partition ordonnée de X est une α partition ordonnée de Z , avec $\alpha \leq k$.

Nous en avons ici fini avec les préliminaires; énonçons donc le ...

Théorème 1 Considérons $\{Y_1, Y_2, \dots, Y_k\}$ est une k -partition ordonnée optimale du problème de k -partition pour le graphe $G=(X, V)$; on note $cut' = \sum_{i=1, \dots, k-1} cut'_i$, et soit $\{X_k^1, \dots, X_k^k\}$ la k -partition de X obtenue par l'algorithme de bipartition itéré; on note $cut = \sum_{i=1, \dots, k-1} cut_i$.

Alors, nous avons: $cut' \leq cut \leq \frac{2*(k-1)}{k} * cut'$.

⁸Ceci afin de ne compter finalement qu'une fois toutes les arêtes inter-classes

⁹Ces relations seront utiles pour la suite, et le terme d'exercice dédouane l'auteur du devoir d'effectuer lui même les calculs!

Démonstration: Déjà, nous effectuons par bipartition itérée une coupe dans X; il devient alors $X_1^1 \cup X_1^2$, et le poids de la coupe est $cut_1 = Sep(X_1^1, X_1^2)$. Par définition de l'algorithme de bipartition (qui est optimal), cette coupe est de poids inférieur ou égal au poids de toutes autres bipartition de X.

Prenons le cas particulier pour le moment $k=4$ ($Y_4 = \overline{Y_3}$), et mettons en évidence les bipartitions suivantes: $\{Y_1 \cup Y_2 \cup Y_3, \overline{Y_3}\}$, $\{Y_1 \cup Y_2 \cup \overline{Y_3}, Y_3\}$.

En écrivant que cut_1 est inférieur à la coupe faite pour obtenir ces ensembles, on obtient:

$$cut_1 \leq Sep(Y_1 \cup Y_2 \cup Y_3, \overline{Y_3}) = Sep(Y_1, \overline{Y_3}) + Sep(Y_2, \overline{Y_3}) + Sep(Y_3, \overline{Y_3})$$

$$cut_1 \leq Sep(Y_1 \cup Y_2 \cup \overline{Y_3}, Y_3) = Sep(Y_1, Y_3) + Sep(Y_2, Y_3) + Sep(\overline{Y_3}, Y_3).$$

En faisant la somme et en utilisant le fait que $Y_3 \cup \overline{Y_3} = \overline{Y_2}$, on obtient:

$$2 * cut_1 \leq Sep(Y_1, \overline{Y_2}) + Sep(Y_2, \overline{Y_2}) + 2 * Sep(Y_3, \overline{Y_3}) = Sep(Y_1, \overline{Y_2}) + cut'_2 + 2 * cut'_3.$$

Utilisons maintenant la bipartition suivante: $\{Y_1 \cup \overline{Y_2}, Y_2\}$: on a:

$$cut_1 \leq Sep(Y_1 \cup \overline{Y_2}, Y_2) = Sep(Y_1, Y_2) + cut'_2. \text{ Par sommation avec la première inégalité, on a:}$$

$$3 * cut_1 \leq Sep(Y_1, \overline{Y_1}) + 2 * cut'_2 + 2 * cut'_3.$$

Enfin, comme $\{Y_1, \overline{Y_1}\}$ est une bipartition, en ajoutant aux précédentes, cela nous donne:

$$4 * cut_1 \leq 2 * cut'_1 + 2 * cut'_2 + 2 * cut'_3 = 2 * cut'$$

Revenons maintenant au cas général; nous allons démontrer que donc $k * cut_1 \leq 2 * cut'$ pour tout k. Considérons déjà les deux bipartitions suivantes: $\{Y_1 \cup Y_2 \cup \dots \cup Y_{k-1}, \overline{Y_{k-1}}\}$ et $\{Y_1 \cup Y_2 \cup \dots \cup \overline{Y_{k-1}}, Y_{k-1}\}$

$$cut_1 \leq Sep(Y_1 \cup Y_2 \cup \dots \cup Y_{k-1}, \overline{Y_{k-1}}) = Sep(Y_1, \overline{Y_{k-1}}) + Sep(Y_2, \overline{Y_{k-1}}) + \dots + Sep(Y_{k-1}, \overline{Y_{k-1}})$$

$$cut_1 \leq Sep(Y_1 \cup Y_2 \cup \dots \cup \overline{Y_{k-1}}, Y_{k-1}) = Sep(Y_1, Y_{k-1}) + Sep(Y_2, Y_{k-1}) + \dots + Sep(\overline{Y_{k-1}}, Y_{k-1}).$$

La somme des deux inégalités donne alors:

$$2 * cut_1 \leq Sep(Y_1, \overline{Y_{k-2}}) + Sep(Y_2, \overline{Y_{k-2}}) + \dots + Sep(Y_{k-2}, \overline{Y_{k-2}}) + 2 * cut'_{k-1}. \text{ Puis on considère}$$

les partitions $\{Y_1 \cup Y_2 \cup \dots \cup \overline{Y_{k-3}}, Y_{k-3}\} \dots \{Y_1 \cup Y_2 \cup \dots \cup \overline{Y_j}, Y_j\} \dots \{Y_1 \cup \overline{Y_2}, Y_2\}$ et $\{Y_1, \overline{Y_1}\}$.

En sommant le tout et en simplifiant, on trouve finalement la relation: $k * cut_1 \leq 2 * cut'$.

Maintenant, nous allons étendre ce résultat aux autres cut_i , ie nous allons prouver, grace au résultat pour cut_1 que $k * cut_i \leq 2 * cut'$.

Etudions déjà posément le cas cut_2 : c'est la plus petite bipartition entre celle de X_1^1 et X_1^2 , ainsi est elle plus petite que toute bipartition de chacun est de ces ensembles. D'après un exo, la trace de $\{Y_1, Y_2, \dots, Y_k\}$, k-partition ordonnée de X sur par exemple X_1^1 est une $\alpha(1)$ partition ordonnée de X_1^1 . Ainsi, en reprenant le résultat obtenu pour cut_1 dans X, et en l'appliquant à cut_2 dans X_1^1 , on obtient: $\alpha(1) * cut_2 \leq \sum_{i=1, \dots, k-1} 2 * cut'_i \lfloor_{X_1^1}$

On obtient une même relation avec la trace de la k-partition ordonnée optimale sur l'ensemble X_1^2 : $\alpha(2) * cut_2 \leq \sum_{i=1, \dots, k-1} 2 * cut'_i \lfloor_{X_1^2}$

On en fait la somme, ce qui donne: $(\alpha(1) + \alpha(2)) * cut_2 \leq \sum_{i=1, \dots, k-1} 2 * cut'_i \lfloor_{X_1^1} + 2 * cut'_i \lfloor_{X_1^2} \leq 2 * cut'$.

Reste alors à minorer $\alpha(1) + \alpha(2)$ par k: Si l'on considère les ensembles $X_1^1 \cap Y_i$ et $X_1^2 \cap Y_i$, leurs unions donne Y_i qui est non vide par hypothèse, donc pour chaque $i \in \{1, 2, \dots, k\}$, au moins l'une des deux traces est non vide; Ainsi, nous en déduisons bien que $k \leq \alpha(1) + \alpha(2)$, et par suite, l'inégalité recherchée.

Soit alors j un indice quelconque; on veut prouver (cas général) que $k * cut_{j+1} \leq 2 * cut'$.

Les ensembles $\{X_j^1, X_j^2, \dots, X_j^j\}$ sont déjà construits;

on désigne par $\alpha(l)$ le cardinal des classes non vides parmi $\{X_j^l \cup Y_1, X_j^l \cup Y_2, \dots, X_j^l \cup Y_k\}$.

Comme la trace de la k-partition optimale sur X_j^l est une $\alpha(l)$ partition optimale de X_j^l , d'après ce qui fut fait pour cut_1 appliqué à ce cas donne:

$$\alpha(l) * cut_{j+1} \leq 2 * cut' \lfloor_{X_j^l} \text{ Par l'argument des traces, nous savons que } k \leq \alpha(1) + \alpha(2) + \dots + \alpha(j),$$

ce qui permet, par sommation des inégalités précédentes pour $1 \leq l \leq j$ d'obtenir:

$$k * cut_{j+1} \leq (\alpha(1) + \alpha(2) + \dots + \alpha(j)) * cut_{j+1} \leq 2 * cut' \lfloor_{X_j^1} + 2 * cut' \lfloor_{X_j^2} + \dots + 2 * cut' \lfloor_{X_j^j} \leq 10^2 * cut'$$

Bien! Nous en sommes arrivé au dessert, car il ne reste alors plus qu'à sommer ces différentes inégalités pour j variant de 0 à k-2 pour obtenir:

$$k * cut = k * (cut_1 + cut_2 + \dots + cut_{k-1}) \leq (k-1) * 2 * cut'$$

ce qui permet donc d'affirmer que l'algorithme de bipartition itérée permet d'approximer le problème de k partition avec une garantie de performance de $2 * \frac{k-1}{k}$.

¹⁰Cf exo puisque les X_j^l forment une partition de l'ensemble X

Enfin, certaines âmes chagrines se demanderont s'il n'est pas possible d'améliorer ce rapport. Si l'on me permet d'assumer que le programme de bipartition itéré peut choisir, lorsqu'il existe des bipartitions (optimales) différentes de poids identique, toujours la plus mauvaise stratégie pour son avenir, alors la réponse est non¹¹.

Voici pourquoi: supposons que l'on veuille effectuer une k partition: considérons alors un polygône régulier à k cotés (d'intérieur vide), de cotés valué à 1. Rajoutons lui sur chaque sommet un arc vers un sommet extérieur (un différent à chaque sommets du polygône) que l'on value par un poids de 2. Nous obtenons donc un graphe (connexe) ayant $2k$ sommets, et $2k$ arretes.

Appliquons lui mentalement l'algorithme de k partition itéré malchanceux: Au premier coups, il peut soit couper une antenne (arrete extérieur au polygône) pour un cout de 2, soit couper 2 cotés du polygône, aussi pour un coups de 2! Il choisit alors de couper une antenne ... (c'est bien à ce niveau une bipartition optimale); puis au coups suivant .. une autre antenne, et ainsi de suite ...

Notre algorithme va ainsi couper $k-1$ antennes de poids 2, ¹² et obtenir finalement $k-1$ singletons (les sommets externes au polygone) et une grosse classe (le polygône et une antenne survivante).

Quelle serait la solution optimale? Et bien elle consiste à couper tous les cotés du polygône (y'en a k), donc pour un poids total de k , et on obtient k classes homogène: les antennes.

La solution optimale est donc de poids inter-classe k , alors que la solution donnée par l'algorithme malchanceux est elle de poids $2^*(k-1)$: la garantie de l'algorithme est alors dans ce cas optimale¹³ !

3 A la recherche d'un autre critère.

3.1 La problématique.

Nous avons donc vu précédemment comment, à la fois les voisinages utilisés mais aussi le critère de séparation en lui même favorisaient les solutions ayant de nombreux singletons. Ainsi, il semblerait souhaitable de rechercher un autre critère qui tempèrerait cette manie des singletons, qui sont particulièrement peu expressif pour notre problème biologique initial. Comme nous avons étudié un critère inter classe, l'étude d'un critère intra-classe semblerait aussi permettre de couvrir un plus vaste champ; aussi la notion de cohérence d'une classe semble elle être un bon candidat pour ce nouveau critère.

Mais comment la définir? Mon souhait à ce niveau était de ne pas demander quoi que ce soit au niveau du cardinal de la classe étudiée¹⁴. Or, le problème se pose bien souvent en terme de cardinal, et celui ci se révèle bien souvent être un facteur prédominant pour des notions de cohérence envisagées.

Etudions un peu l'une de celles ci:

- *le poids moyen des arretes intra classe.* Cela pourrait être, à première vue, une bonne idée (et on pourrait même ne pas y voire de lien avec le cardinal de

¹¹enfin, le rapport obtenu est optimal

¹²n'y voire ici aucune allusion à l'ancienne chaine publique de télévision!

¹³Remarquez alors que ce même exemple permet d'affirmer que bipartition itéré chanceux peut tomber sur la solution optimale pour $k \geq 3$

¹⁴afin de na pas créer trop d'arbitraire dans la démarche.

la classe); les singletons auraient la pire cohésion alors (zéro par choix), ce qui ne serait pas mal pour contrebalancer le critère de séparation ... Mais ...

Le cardinal de la classe intervient en fait de manière fondamentale¹⁵: supposons que le poids moyen des arêtes du graphe complet soit 10 (les arêtes ayant des poids entre 0 et 20).

Que dire d'une classe ayant 15 pour moyenne de ces poids? Si le cardinal de la classe est petit (par exemple 3 sommets, donc 3 arêtes): bof, on doit pouvoir faire bien mieux; il n'y a rien d'extraordinaire, et vu le grand nombre de sommets dans le graphe, il est assez probable qu'il existe un triangle ayant tous ces poids à 20!

Mais si la classe est une grosse classe (par exemple la moitié des sommets), alors là, cette valeur de 15 aura une toute autre signification, et la classe sera alors particulièrement bien découpée.

Ainsi, le cardinal intervient il toujours¹⁶, et nous nous retrouvons alors avec deux paramètres: le cardinal de ladite classe, et le poids moyen de la classe: comment conjuguer leurs effets? Comment les interpréter de manière systématique? ...

3.2 Une notion de cohérence probabiliste

Une solution à ce dilemme pourrait nous être donnée par l'usage de la loi des grands nombres ...

Nous connaissons la matrice des poids: relevons les fréquences d'apparition des différents poids et construisons alors la probabilité régissant cette distribution de poids dans le graphe. Calculons alors les valeurs caractéristiques: espérance m et variance σ^2 , puis, faisons quelques rappels:

Théorème 2 (Inégalité de Tchébichev) *Soit X une variable aléatoire d'espérance m et de variance σ^2 . Alors, pour tout $\alpha \geq 0$, on a:*

$$P(|X - m| \geq \alpha) \leq \frac{\sigma^2}{\alpha^2}.$$

Si l'on signale cette inégalité, c'est qu'elle va servir à démontrer (rapidement) le théorème de la loi (faible) des grands nombres, et que certains arguments de cette démonstration serviront à la construction du critère de cohésion probabiliste.

On va maintenant considérer X une variable aléatoire, d'espérance m et de variance σ^2 . On effectue n expériences indépendantes¹⁷; on désigne par X_1, \dots, X_n les valeurs prises par X lors de ces expériences.

On note $Y = \frac{\sum_{i=1}^n X_i}{n}$; rappelons simplement que l'espérance de la variable aléatoire Y est m (en tant que fonction linéaire de variable aléatoire indépendantes...), et la variance de Y est: $\sigma_Y^2 = \frac{\sigma_X^2}{n}$.

Théorème 3 (Loi faible des grands nombres) *La moyenne arithmétique Y des X_i converge en probabilité vers m quand n tend vers l'infini; ie pour tout $\delta \geq 0$ et $\epsilon \geq 0$, pour n suffisamment grand, on a:*

$$P(|Y - m| < \epsilon) > 1 - \delta$$

Démo: Par l'inégalité de Tchébichev: $P(|Y - m| \geq \epsilon) \leq \frac{\sigma_X^2}{n * \epsilon^2}$. Aussi petit que soit ϵ , on peut trouver n grand tel que $\frac{\sigma_X^2}{n * \epsilon^2} < \delta$; la preuve se conclut en prenant l'événement contraire.

¹⁵bien que discret!

¹⁶dans les notions évoqués lors de l'étude préliminaire en tout cas

¹⁷très important pour la suite!

Maintenant, trichons un peu, car nous avons toutes les cartes en main: si on désire savoir si une classe est cohérente, on calcul sa moyenne des poids (Y) et on calcul son écart à la moyenne ($|Y - m| = \epsilon$); connaissant son cardinal (n) on pose $\delta = \frac{\sigma_X^2}{n * \epsilon^2}$. Nous en déduisons que nous sommes alors sous les hypothèse du théorème. Ainsi, fixons nous un seuil pour δ : ce seuil représentera alors la vraisemblance de l'hypothèse de l'indépendance des X_i . Si δ est trop grand (au dessus du seuil), alors nous estimerons qu'il est peu vraisemblable que l'on ait ce résultat pour des variables X_i indépendantes: ainsi, estimerons nous alors qu'elles sont "dépendantes" ! et que la différence entre leurs poids moyen et m_X est significatif ... bref, la seule répartition aléatoire ne peut justifier la présence d'un tel écart à la moyenne; on interprétera cela alors comme une vraie information sur les interactions des protéines.

Deux possibilités alors: soit Y , la moyenne des X_i est inférieure à m_X (le poids moyen sur l'ensemble de la matrice), auquel cas on pourra dire que cette classe est particulièrement incohérente¹⁸.

Soit Y est supérieur à $m_Y = m_X$, et alors nous pourrions dire que nous avons là une classe probabilistiquement cohérente¹⁹.

L'intérêt de l'appel à la loi faible des grands nombres et de ne plus avoir au final qu' à fixer un seul et unique paramètre: la valeur du seuil. Tout les problèmes relatifs à l'influence du cardinal de la classe sont pris en compte par la machinerie de la loi, en particulier dans le calcul du δ que l'on compare à la valeur seuil.

On peut aussi se demander si l'on n'a pas de restrictions quant aux tailles des ensembles (pour les questions de vraisemblances). La taille du graphe entier, avec disons une centaine de sommets, permet sans aucun problème de confondre la loi de probabilité avec les valeurs que nous observons. Par contre, je ne suis pas certain de la possibilité de prendre tout cardinal en ce qui concerne la classe: à priori oui (bien que le résultat de la convergence en probabilité soit lorsque n tend vers l'infini, on peut obtenir des indications et relations pour de petites valeurs de n par notre manipulation).

3.3 Des algorithmes.

Dans un premier temps (en fait, définitivement), on écarte la notion de seuil au bénéfice d'une minimisation de la valeur de δ pour une classe donnée: plus δ , calculé donc en supposant les poids soient indépendamment répartis, aura pour une classe donnée une petite valeur, plus l'écart à la moyenne de cette classe sera plausible.

Ainsi, les premiers algorithmes se sont ils concentrés sur la minimisation, pour une classe donnée, de son δ associé.

La première idée fût de construire peu à peu les classes : en partant d'une solution classe vide, et en testant ensuite si la classe gagnerait à se voir ajouter tel élément extérieur à elle. Un problème au démarrage de cette méthode à alors vite été constaté ! les classes vides ou singletons semblent, pour l'algorithme

¹⁸pour le moment, pas d'idée sur l'utilisation de cette notion, mais elle me paraît néanmoins intéressante, même si on ne puisse rien dire sur la relation entre 2 points quelconques de cette classe.

¹⁹Ici, on pourra signaler un petit problème: si on réfute l'hypothèse de l'indépendance des X_i , alors on ne peut plus assurer que $m_X = m_Y$, mais on assumera ici cet abus.

programmé, hermétiques à tout changement.

Une méthode partant d'une classe pleine et tentant l'élimination fût ensuite tentée, sans plus de succès (la classe étant stable).

Finalement, on se rabat, pour amorcer le processus, sur une solution aléatoirement tirée: l'intérêt est la simplicité, mais le défaut vient ensuite de la combinaison avec la transformation élémentaire considérée: le 2 opt.

On considère donc deux classes distincts, et de façon systématique, on tente l'échange d'un élément de l'une avec un élément de l'autre; si la somme des cohésions des δ relatifs à chaque classe est améliorée par les changements, alors on les effectuent réellement. Ce passage consistant à optimiser le critère entre deux classes fût surtout trouvé pour palier au temps (absolument déraisonnable) d'un premier algorithme tentant une optimisation par trops systématique. Mais la lacune de partir d'une solution aléatoirement créée associée à une transformation 2-opt est que l'on fixe alors de manière arbitraire le cardinal des classes²⁰. En effet, on n'a pas, pour ce choix de voisinage, la possibilité d'augmenter ou de diminuer le cardinal d'une classe donnée, puisque le 2 opt ne fait qu'échanger un de ces éléments contre un autre qui lui est extérieur.

Ainsi, on introduit une seconde transformation élémentaire, que l'on surnommera recherche de classe par le haut: On désigne une classe, et on balaye les sommets extérieurs à cette classe (à condition que la classe auquel on l'enlève ne soit pas justement réduite à un singleton) les uns après les autres: si jamais son ajout à la classe considérée améliore la cohérence de la classe, alors il y est ajouté.

A ce niveau, il semble préférable d'utiliser déjà la recherche d'une classe optimale par le haut (ie on tente l'ajout de tout les sommets pris successivement, et si son ajout améliore le delta, on le garde) avant que de faire la recherche par les 2-opt plutôt que l'inverse. Ces diverses considérations seront prises en compte lors de l'étude par simulation informatique; d'autres méthodes paliant certains défauts seront alors proposé où envisagés, mais on peut dore et déjà indiquer une certaine adéquation avec les classes testées²¹.

Maintenant, un petit interlude sur la forme des données.

Initialement, les matrices considérés devaient être à valeurs entières, mais les matrices provenant des biologistes contiennent des décimaux avec généralement 2 chiffres après la virgule. Une adaptation alternative à la solution couramment adoptée et qui devait être de considérer des réel, consiste à multiplier toutes ces valeurs par 100 ou 1000 et de ne travailler qu'avec des entiers. L'avantage espéré de ce biais est que, lors du calcul de l'espérance et la variance, mieux vaut avoir des nombres "grand", afin de moins souffrir d'éventuels arrondis. Ensuite, il est bien plus préférable de travailler avec des entiers pour déterminer aisément la probabilité de la répartition des poids dans le graphe (pour éviter une notion arbitraire de seuil ici).

Dernier point résultant du choix précédent: les cohérences (ie les δ) peuvent donc des réels bien supérieur à 1 (d'autant plus que l'on a implémenté un delta différant d'un facteur multiplicatif du delta de la loi des grands nombres). Rappelons encore une fois que dans les programmes, plus le delta fournit au résultat

²⁰on y reviens toujours!

²¹Néanmoins, ne pas se réjouir trops vite ... une étude critique la plus rigoureuse possible sera par la suite menée.

est faible, meilleur la classe est, ce qui présente un réel intérêt: nous savons que nous ne sommes pas à l'optimum, mais néanmoins, ce chiffre peut donner une indication sur quelles sont les classes rendant bonne cette solution, et quelles sont les classes défavorables.

En particulier, on pourrait relancer le programme avec des solutions initiales différentes et ne retenir entre chaque passages que les meilleures classes, quitte à ce que certaines aient des sommets communs.

Les résultats, pour les matrices tests très régulières²², furent surprenants, étant donné qu'aucun calcul de séparation n'a été effectué²³.

Test sur un problème bruité. Juste une petite réflexion sur un attrait possible de cette méthode probabiliste. On pourrait considérer que les valeurs initialement données par les biologistes sont légèrement erronées²⁴.

Le critère de séparation se trouve être assez sensible à ce bruit extérieur à nos méthodes. En effet, considérons un sommet quelconque (et rappelons que ce critère tend à trouver des singletons). Imaginons que ce sommet subisse sur chacune de ses arrêtes un bruit plus faible que pour les autres, par exemple, au lieu d'être un (petit) bruit entre 0 et 1, il n'est qu'un bruit entre 0 et 0,7. Alors, sur la centaine de sommets voisins, l'écart au niveau du critère de séparation sera alors de $50 - 35 = 15$, ce qui le rendra donc sensiblement meilleur que les autres au vue du critère de séparation.

A l'inverse, le critère de cohérence devrait bien mieux résister à ce même bruit ... des tests dans ce sens seront effectués à la fin.

4 Les méthodes de bruitages.

4.1 Présentation de ces méthodes.

Ces méthodes, développés principalement dans les articles de I.Charon, se proposent de perturber le problème ou la descente de manière aléatoire afin d'espérer obtenir de meilleures solutions finales. On distingue à l'heure actuel trois types de bruitage:

- *Le bruitage des données*, consistant dans notre étude à perturber la matrice S par un bruit B, dépendant d'un paramètre ($bruit_{max}$). Déjà cette méthode laisse un grands nombres de choix quand à la façon de bruite les données²⁵, dans le choix de la décroissance de $bruit_{max}$, dans le choix de la stratégie alternant descente bruitée et non bruitée ... Ainsi, plusieurs choix et tentatives devront être réalisés.
- *Le bruitage des variations*: Consiste non plus à bruite la matrice S mais les différentes valeurs prisent par la fonction objectif. A titre d'exemple, comme il s'agit de méthode de descente à voisinage, si f désigne la fonction objectif, x et x' deux solutions voisines, on calcul généralement dans l'algorithme des valeurs telles que $Max_{x' \text{ voisin de } x} f(x) - f(x')$ afin de récupérer au final le voisin maximisant le gain (ici pour une descente, ie problème de minimisation).

²²celles étudiées plus haut, un avec 6 pentagones, l'autre avec 8 heptagones

²³Néanmoins, pour ces matrices, les cohérences et séparations coïncident clairement dû fait de leurs grande simplicité.

²⁴erreurs d'approximations et autres

²⁵En particulier, quelle probabilité utiliser? l'uniforme entre 0 et $bruit_{max}$? une centrée en $bruit_{max}/2$? ...

Pour d'obtenir d'autres solutions, potentiellement meilleurs, on tente de bruite ces variations de f par un facteur (exemple: facteur multiplicatif proche de 1). Ainsi, on considère le x' maximisant $\text{bruit} * (f(x) - f(x'))$, où d'autres formes dérivées...

- *Le bruitage par oubli.* C'est, pour le moment, le dernier bruitage connu; on oublie, de manière aléatoire, certaines valeurs de S (ou certains sommets), afin d'espérer éliminer des points ou des poids qui perturbaient fortement la descente initiale.

Ainsi, de nombreux choix s'offrent désormais à nous, tant sur la stratégie des descentes bruitées (répétition, bruit jusqu'au bout ou non, descente systématique ou non, mixage des descentes) que sur des paramètres propres à ces méthodes (choix de la probabilité pour la décroissance, choix de la façon de faire décroître le bruit,).

4.2 Le bruitage des données

La méthode classique consiste donc à bruite les diverses données de la matrice initiale S . Le premier paramètre à ce niveau est de déterminer quelle bruit maximum admettre. La probabilité de bruitage sera elle considérée comme uniforme. Ensuite, viens le choix de la stratégie de la descente: on effectue une descente (complète) avec la matrice bruitée S_b , puis, à partir de la solution obtenue, on refait une descente systématique mais cette fois ci pour la matrice S . Habituellement, il y a un choix à ce niveau pour savoir comment faire décroître le bruit à partir de la valeur bruit max. Ici, étant donné que l'on opte pour une descente systématique après chaque descente bruitée, on utilisera une autre stratégie; les tests finaux seront en partie effectués à temps constant pour chaque méthodes; c'est pourquoi, au lieu de faire décroître le bruit, nous allons partir d'un bruit nul, et l'augmenter peu à peu ... le programme s'arrêtant alors dès que le temps maximum imparti pour le test aura été dépassé.

Ainsi, la seule question restante est de savoir comment faire croître le bruit. Cette croissance sera (initialement) fonction du plus grand coefficient de la matrice initiale.

Ainsi, obtenons nous l'algorithme suivant:

(Initialement) Bruit $B=0$; S = matrice à traiter.

Bruiter la matrice S en S_b par le bruit B .

★ Effectuer une descente à partir d'une solution tirée au hasard pour S_b .

★ Effectuer une descente à partir de la solution obtenue pour S .

★ Augmenter le bruit B , et retour en haut de l'algorithme.

Bien sûr, on retient la meilleur solution rencontrée pour S au cours des différentes itérations.

4.3 Le bruitage des variations.

Ici, on ne modifie donc plus les données initiales mais bel et bien les variations de la fonction f . Une stratégie de bruit croissant similaire à celle évoqué ci dessus sera initialement tenté ici aussi.

On considérera donc une fonction bruit (fonction de bruit maximum B), donnant des valeurs contenue entre $[1-B;1+B]$. Ainsi, le bruit maximum devra t'il être toujours compris entre 0 et 1; la façon d'augmenter le bruit devra donc tenir

compte de cet impératif.

On pourrat débuter nos test à partir d'un algorithme semblable à:

(Initialement) bruit maximum $B=0$; matrice S , fonction f .

On tire aléatoirement une solution x .

On considère la solution x' voisine de x réalisant:

$Max_y \text{ voisin de } x \text{ bruit}(B) * (f(x) - f(y))$, jusqu'a obtenir un minimum local.

Partant de ce minimum local, on fait une descente non bruitée pour f .

On augmente le bruit maximum B , et retour en haut de l'algorithme.

5 De la simulation informatique.

Ce chapitre est donc consacré à diverses simulations; il est construit de manière investigative, afin de rendre compte des premiers résultats et de leurs effets, d'expliquer ensuite au mieux les choix, défauts, test ratés, foule de programmes peu propres et finalement biscornus, et autres complexifications des algorithmes aux²⁶ lecteurs.

Pour ceux dont les tatonnements n'intéresse en rien, le chapitre prochain donne en vrac divers résultats sous forme de synthétiques tableaux de chiffres concernant les méthodes survivantes où apparue lors de ces tribulation.

Pour ceux doutant de la signification des chiffres par eux mêmes, ce (bien trop long) chapitre donnera plusieurs indications pour les interpréter.

Néanmoins, les algorithmes seront constamment remaniés entre les différents paragraphes, et avant le chapitre prochain. Ainsi, les chiffres ne seront qu'une sommaire indication, et ne peuvent servir de test de référence pour la qualité de tel ou tel algorithme.

5.1 Introduction.

En introduction, nous allons donc programmer ces différentes méthodes présentés ci dessus, et plusieurs facteurs seront tour à tour testés, en fonction des résultats aux tests précédement effectuées. Parmi ces facteurs, les plus simples d'accès seront: le temps, la séparation des classes, l'adéquation aux classes initiales, la robustesse à un bruit externe ... Pour le moment, les premiers tests sont là pour surtout chronométrés les différentes méthodes initiales.

5.2 Les premiers tests.

Ils ont donc pour but d'évaluer le temps mis par chacunes des méthodes initiales. Certaines méthodes, qui n'apparaîtrons que plus tards, ne seront pas mis ici pour plus de clareté. Ces tests seront effectués sur un pc personnel, tres peu optimisé lui même pour ce genre d'acrobaties, avec certainement une foulitude de programmes extérieurs aux sujet tournant en parallèle du programme testé. C'est pourquoi, ces temps seront là surtout à titre indicatif, et aussipourquoi on testera biensouvent sur un tres petit nombre de matrices. De même, à peu prés tout les chiffres donnés sont obtenue par des méthodes à constante multiplicative prés ; par exemple, les poids des arcs ont tous été multiplié par 100 ou 1 000 (afin de prendre en comptes d'éventuels nombres décimaux), ou encore, la valeur de δ , relatif à la notion de cphérence à été multiplié par un facteur 100

²⁶éventuels

(pour la rendre plus lisible), et encore n'a t'on pas mis toutes les constantes. Dans le chapitre suivant, ces manquements à la rigueur seront corrigés, afin de donner des résultats étalons, pouvant être comparés à ceux des éventuels utilisateurs ou lecteurs.

- Le premier test consiste en trouver un minimum local pour 200 matrices tirées au hasard, ayant toutes une forme solution (ie diagonale par bloc), sans aucun bruit: bref, ce devrait être très très facile. On test ici le programme basé descente simple, et réalisant²⁷ une descente complète avec la première notion de voisinage.

Pour des matrices de taille $n=100$ et de poids maximum 100:

La méthode de descente simple premier voisinage: trouve toutes ces solutions en ... 2 secondes, mais la moyenne des séparations des solutions est de 4000 (les solutions optimales étant nulles).

L'algorithme basé sur le second voisinage lui, portant le nom de descente, utilise un temps processeur de 559 secondes, soit un peu plus de 9 minutes... long!!, pour un bien meilleur résultat néanmoins.

- Interressons nous pour le moment aux capacités, dans ce cas ultra simple de la première notion de voisinage: Si l'on augmente la taille de la matrice à $n=200$, alors on met 7 secondes à traiter les 200 matrices (aléatoires sur les blocs diagonaux) à coefficients entiers entre 0 et 100, pour trouver à chaque fois des 10 partitions, ce qui est extrêmement raisonnable. Par contre, malgré la simplicité des solutions à trouver, la moyenne des séparations est de 10 000, ce qui est bien médiocre!

- Concentrons nous maintenant sur le second voisinage: au vue de sa lenteur, nous ne testerons plus sur un si grand nombre de matrice.

Sur une matrice de taille 100, il met 5 secondes à trouver une solution d'indice de séparation 4532.

Dans un temps similaire, le programme de descente simple réitéré pour lui laisser un temps similaire permet de trouver une bien meilleure solution. Ainsi, pour le moment, la descente simple aurait elle nos faveurs ...

5.3 Des améliorations du programme second voisinage.

Nous avons testé ici le programme de bipartition classique, sans aucune des améliorations citées plus haut²⁸. Munit de ces améliorations, ressemblant d'assez loin à un recuit simulé, quels sont les gains de ce programme?

- Par exemple: Une matrice donc toujours (100,100) diagonale par bloc et nulle ailleurs, doit être, pour diverses raisons, partitionnée en 10 classes. Une descente de bipartition simple met 20 secondes²⁹, pour trouver une solution de coefficient de séparation égal à 4 588.

Le programme de bipartition avec fusion met lui 1 seconde pour trouver une solution dévaluation 41 719. Ainsi, nous allons devoir effectuer une descente simple de manière systématique après une descente bipartition avec fusion.

- *Le retour de la bipartition.*

Ainsi armé, revenons à la charge. Laissons 20 secondes aux deux duélistes et

²⁷je l'espère

²⁸Le programme amélioré se nomme descente fusion, le basique étant descente

²⁹c'est beaucoup quand même!

ne retenons que la meilleure solution fournie. Bipartition fusion effectuée 3 descentes pendant ce temps, dont la meilleure est d'évaluation 3 604. Descente simple fait lui 4820 descentes dans le même temps, pour une solution optimale de 3 248.

Recommençons, mais avec une matrice de taille 100, et dont les blocs hors diagonaux sont perturbés par un bruit entre 0 et 10. En une minute, bipartition-fusion a eut le temps d'effectuer 33 itérations, pour une solution optimale de 10 684, contre seulement une solution minimale de 20 944 dans les 914 itérations de descente simple. Ainsi, cette augmentation de taille semble lui être favorable.

De plus, il reste encore une arme au second voisinage: l'algorithme de bipartition itéré, qui garantit quand même une certaine qualité à la solution.

Evidemment, avec une matrice non perturbée, bipartition itéré trouve la solution optimale de séparation 0...

Et si non seulement nous donnions à bipartition toutes ses armes, mais qu'en plus, nous la faisons tourner sur une matrice (50,50) légèrement perturbée (ie comme ci dessus), alors bipartition et compagnie trouve en 34 secondes une solution d'indice 2 947, alors que dans le même temps, descente simple répète 40000 fois trouve seulement une solution de séparation 4 338. Quelques autres valeurs: bipartition full trouve 226 contre 293 pour descente simple. Pour toujours des matrices (50,50) perturbées fortement cette fois, on trouve une séparation de 5 853 pour bipartition full contre 7 438 pour descente simple.

Au vu de certains des résultats précédents, il semblerait que descente simple perde de ses qualités lorsque n est grand. Augmentons donc la taille de la matrice³⁰: celle ci sera désormais assez perturbée, voire fortement perturbée (ie là où on avait des zéros, on perturbe par des poids entre 0 et 10; les poids des blocs diagonaux sont toujours compris entre 0 et 100).

Pour $n=60$, toujours pour une 10-partition, bipartition donne une séparation de 6 602, alors que descente simple trouve, en 13269 itérations, au mieux 10 811.

Pour $n=70$, bipartition full met 61 seconde pour trouver une solution de séparation 8 270; dans le même temps, descente simple en profite pour faire 24166 itérations, donnant au mieux une séparation de 12 929.

Ainsi, la courbe qui semblait se dessiner en faveur d'une prise du pouvoir par bipartition full au fur et à mesure du temps semblerait se confirmer, en tout cas pour des matrices légèrement perturbées.

Il est bien sûr inutile de préciser que les "bonnes" solutions fournies par les deux méthodes sont sous formes de singletons³¹ ...

Le dernier test de cette série est celui sur une matrice de taille $n=100$... Bipartition full nous offre un solution d'indice de séparation 7 190 en 212 secondes³², soit 4 minutes, alors que dans le même temps, les quelques 47300 itérations de l'algorithme descente simple, basé sur le premier voisinage, n'aboutissent finalement qu'à une solution d'indice de séparation 12 962.

Ainsi, la descente grâce à bipartition full (ie fusion et démarrage par bipartition itérée) permet-il d'obtenir, pour des matrices de taille disons assez grande, de bien meilleurs résultats que de très nombreuses itérations de la descente simple.

Déjà, regardons si l'absence de l'amorce par bipartition itérée nuit beaucoup

³⁰L'auteur étant partial et préférant bipartition, il convie les batailles à se dérouler sur le terrain de prédilection de ce dernier!

³¹Pour ces matrices pas trop perturbées, descente simple semble encore garder quelques peu la structure des blocs.

³²warf! commence être longuet!

à bipartition full pour cette même matrice. Ce nouveau assemblage permet de trouver en 224 secondes une solution de séparation 7 190; dans le même temps ... les 48450 itérations de descente donnent 19 327. Ainsi, pas de gain de temps, ni de perte de qualité au niveau de la solution obtenue.

Tentons alors d'améliorer les temps mis par l'algorithme de bipartition, afin de le rendre plus rapide.

Une première chose pourrait être de renoncer à chercher un minimum local pour la notion de second voisinage; ainsi fait, si l'on élimine le programme de descente suivant celui de descente avec fusion (qui ne garanti absolument pas d'obtenir un minimum local); ne gardons que le programme de descente fusion pour la bipartition, et tentons le sur la même matrice. Laissons aux deux algorithmes 1 minute; bipartition avec fusion a eût le temps de se répéter 65 fois, et trouve une solution de séparation 66 600; les 12112 itérations de descente simple fournissent elles une solution de séparation 9 355.

Evidement, là ... mieux vaudrait être certain de ne pas s'arrêter après que bipartition-fusion ai justement fait pleins de fusion (et donc ne se soit largement éloignée des petites valeurs de séparation); comme la descente sous le second voisinage n'est pas assez rapide, couplons bipartition fusion avec descente simple, et testons le face à descente simple : sous les mêmes conditions, la bipartitions avec voisinge variable donne après 66 itérations 7 319; descente simple seule, en 12803 itérations trouve une solution de séparation 12 495.... mieux!

5.4 Les bruitages.

D'après les premiers tests réalisés en off, et à moins d'erreurs de programmation, les méthodes de bruitage semblent bien peu constructive, et sont bien souvent dépassées ne serait ce que par les méthodes de descentes simples itérées.

Une possible explication de ce fait serait peut être la structure très particulière des solutions: Les bruitages interviendraient directement sur le critère alors, et non plus sur la descente. Suit quelques exemples.

- *Bruitage des données:*

La descente (sous la première notion de voisinage) d'une matrice (50,50), avec un certain choix de paramètres, obtient une solution d'évaluation 6 447, (obtenue pour un bruit maximum assez conséquent de 306); durant les 5 secondes accordées aux protagonistes, 3259 itérations de descente simple on donné une solution d'évaluation 6 398 (ce fut la 341 ième itération qui la donna.); à peu près égalité, et même avantage à descente simple.

Comme précédemment, augmentons peu à peu la taille de la matrice test ;

pour n=60: la meilleure solution bruitée est obtenue pour le fort bruit de 880, et vaut 7 651; descente simple obtient elle 7 320.

Pour n=70: 9 007 pour le bruitage, contre 8 731 pour descente simple itérée.

n=80: 9 747 contre 9 609.

Enfin, sautons directement à n=100: 7 577, contre 7 500, toujours en faveur de descente simple itérée. Si l'on refait ce jeu de données mais en laissant 20 secondes aux concurrents, on obtient: 7 469 pour bruit des données (bruit de 2046 à la meilleure solution); contre 7 419 pour l'itérée.

- *Bruitage des variations:* On va ici utilisé non pas un bruit multiplicatif des

variations de f mais un bruit additif. Pour un certain choix de paramètres³³, on obtient:

Pour $n=50$: 6 695 pour bruit variations, et 6 421 pour descente simple.

Pour $n=70$: 9 243 versus 8 724.

Pour $n=100$; 7 820 versus 7 630.

Bref, rien de mieux, et les descentes simples itérées reçoivent, pour le moment, nos faveurs!

Peut être devrions revisiter un peu ces différentes notions de bruitages afin de les rendre plus performantes qu'elles ne le sont actuellement. A cette fin, on recommande la lecture des différents articles de I.Charon, A.Germa et O.Hudry sur les "noising methods".

5.5 Pourquoi le bruitage des données donne t'il de si mauvais résultat ?

Hélas³⁴, il n'est point illogique que, dans les conditions dans lesquelles nous l'avons programmé, le bruitage des données donne de médiocres résultats.

Voici pourquoi: Nous effectuons une descente complète bruitée avec l'algorithme de descente (premier voisinage). Ainsi, qu'obtenons nous au final? ... des singletons !

Et ceux ci furent directement choisis en fonction du bruitage, puisque le bruit, porté sur le poids des arrêtes du graphe, se retrouve dans l'indice de séparation des classes; rappelons ici que la contribution à la séparation globale d'une classe réduite à un singleton $\{x\}$ est exactement son degré (ie la somme des poids des arrêtes ayant ce sommet pour extrémité). Ainsi, en bruitant, on déforme la solution (en perturbant les indices de séparations des singletons)...

Pas grave alliez vous dire, c'est la méthode qui veut cela?

Et bien le problème est qu'à la sortie de cette descente complète bruitée en donnée, on a une solution qui se trouve être un (mauvais) minimum local pour cette notion de voisinage appliqué à la matrice initiale (non bruitée). En effet, d'après les arguments relatifs à la présence des singletons dans les solutions, nous savons qu'il est improbable qu'une solution voisine de celle obtenue et contenant au lieu d'une classe singleton une classe paire soit de séparation meilleure.

Comme un singleton implique $n-1$ arrêtes dans la coupe le séparant du reste des sommets, l'ajout d'un sommet dans cette classe-singletonne implique le gain du poids d'une arrête, pour une perte de $n-2$ autres poids d'arrêtes. Il est assez peu vraisemblable qu'un sommet présente une hétérogénéité dans sa répartition de poids telle que son ajout à une classe singleton soit profitable au critère de cohésion.

Ainsi, on ne peut ajouter aucun sommet de la grosse classe aux classes singletonnes! L'inverse étant tout à fait impossible, puisque cela donnerait une classe vide, donc une solution non réalisable, cela explique en partie les mauvais résultats de la méthode de bruitage des données. Elle ne permet pas, comme à son habitude, de se diriger vers un bon voisinage, mais donne directement une solution qui se retrouve coincée.

Quelles seraient les possibilités de s'en sortir? Changer la notion de voisinage

³³probablement mauvais!

³⁴Milles fois Zélas

(faire à l'issue de la descente bruitée des transformation échangeant 2 sommets de classes), ou sinon stopper la descente bruitée avant qu'elle n'atteigne un minimum local pour elle (qui sera alors aussi local pour l'autre solution). Pour le moment, ces changements ne seront pas réalisés, dû fait des nombreuses autres possibilités non encore explorées, et elles, déjà réalisées.

• *Sur le bruitage des variations.*

Tentons autre chose pour ce bruitage. Classiquement, nous l'avons vu, il consistait en l'ajout d'un bruit (soit multiplicatif mais aussi additif) aux variations de la fonction à optimiser prise entre deux solutions voisines, comme par exemple: si $(f(x) - f(x')) + \text{hasard}_{\text{bruit}} > 0$, alors on remplace x par x' .

De nombreux problèmes ce posent alors à ce niveaux, citons en quelques uns. Quel bruit choisir? ca, c'est le plus facile si l'on accepte le choix de creer des bruits croissant. On pourrait aussi avoir l'idée de faire un bruit fonction du coefficient maximum de la matrice³⁵, afin de réelement influencer celle ci, sans trops la destabiliser. Et là, nous allons, telle Ulysse, de Charybdes en Scylla, quand à la fonction $\text{hasard}(\text{bruit})$. En effet, nous faisons appel à cette fonction hasard lorsque l'on calcul la différence entre des valeurs de la matrice³⁶ $\text{Gain}_{x,i}$. Ainsi, c'est à ce niveau que nous ajoutions le bruit ... mais quantitativement parlant, quel bruit ajouter (ou quel $\text{hasard}(\text{bruit})$)?

Supposons que i désigne une classe petite, contenant par exemple 5 sommets (si $n=100$); alors, $\text{Gain}_{x,i} \approx 5 * p_{\text{moyen}}$, où p_{moyen} désigne donc le poids moyen d'une arrête du graphe initial. Ainsi, un bruit raisonnable serait il de l'ordre de $3 * p_{\text{moyen}}$. Or, si maintenant, j désigne une classe énorme (comme il est probable qu'il en existe une rapidement), alors $\text{Gain}(x, j) \approx 90 * p_{\text{moyen}}$, soit 18 fois environ $\text{Gain}_{x,i}$. Ainsi, un bruitage (additif) adapté serait il beaucoup plus important que $\text{Gain}_{x,i}$... de plus, on ne sait pas, à priori, si l'on fait la différence entre deux solutions de types i , ou deux de type j , ou une de type i et l'autre de type j . Quand à un bruitage multiplicatif, il faudrait donc pouvoir atteindre de tres grande valeur (de l'ordre de n peut etre) pour qu'il puisse etre véritablement efficace ... ce qui est bien loin de 1 !

Proposons alors une alternative: supposons que notre $\text{hasard}(\text{bruit})$ donne un nombre entre 0 et 1. Maintenant, comme $(f(x) - f(x')) > 0 \Leftrightarrow f(x) > f(x') \Leftrightarrow \frac{f(x)}{f(x')} > 1$, on pourrait considérer que l'on échange x en x' si $p < \frac{f(x)}{f(x')}$. Ainsi, tous voisinage x' améliorant f seraient ils systématiquement pris, et ceux ne l'améliorant pas strictement, ie tels que $f(x) - f(x') < 0$ vérifieraient alors bien $\frac{f(x)}{f(x')} < 1$, et auraient alors une chance d'être pris si on tire un tout petit $\text{hasard}(\text{bruit}) = p$. On voit aussi clairement que plus le bruit est petit, plus les espérance et variance de la probabilité donnant p devront être proche de 1 (et 0 pour la variance). Le cas limite aucune perturbation acceptée étant alors régie par la loi ponctuelle $\text{hasard}(0) = p_0 = 1$.

Tentons tous ces changements... A titre indicatif, nous allons nous baser, pour définir cette probabilité, sur la fonction $x \mapsto \frac{1}{1+x}$, que nous multiplierons par un nombre généré pseudo-aléatoirement, grace à la fonction $\text{rand}()$: si rand_1 et rand_2 désignent des entiers naturels aléatoirement tirées (dans un intervalle quelconque), alors on associera la probabilité: $\frac{1}{1+\text{bruit}} * \frac{1+\text{bruit}*\text{rand}_1}{1+\text{bruit}*(\text{rand}_1+\text{rand}_2)}$; ainsi, est on certain que ce soit bien une probabilité, et que sa valeur renvoyée

³⁵ce fût fait.

³⁶On se reportera à la proposition 1 pour mémoire.

pour $\text{bruit} = 0$ sera bien 1.³⁷

- *Bruit en variation*³⁸

Avons nous, après bien des tribulations, enfin accomplie une amélioration substantielle de la méthode initialement évoquée?

Changeons donc quelques peu les règles du test: dorénavant, le bruit décroît (et oui, car pour $\text{bruit}=0$, on n'a plus de changement); ainsi, faisons décroître notre bruit maximum de 1000 à 0, en le décrémentant à chaque coup. Nous continuons bien entendu d'effectuer une descente systématique après chaque descente accomplie avec bruitage des variations. Retenons le temps mis à faire ces 1000 boucles par l'algorithme bruité, et appliquons durant ce même temps descente simple itérée. Alors ...

Déjà pour la matrice test présentant 8 heptagones: la descente bruitée obtient en 27 secondes une solutions de 273, alors que dans ce même temps, l'itérée donne 324... bien.

Reprenons maintenant notre matrice aléatoirement bruitée.

Pour $n=50$: en 17 secondes, l'algorithme bruité en variations trouve une solution de séparation 6 574, contre 6 764 pour le programme itéré.

Pour $n=60$: après 23 secondes: 7 558 contre 7 844 (en 1024 itérations).

Pour $n=70$, on trouve après deux fois 25 longues secondes d'angoisse des séparations de 8 999 contre 8 996 pour itération.

Pour $n=80$: 9 570 pour la méthode bruitée, contre 9 494 pour l'itérée....

Pour $n=100$ enfin: 7 499 contre 7 486 pour la descente simple.

Ainsi, malgré une notable amélioration pour les petites valeurs de n , l'algorithme actuel de bruitage des données souffre t'il d'une lacune³⁹ qui lui fait perdre de son intérêt lorsque le cardinal de l'instance augmente.

Vérifions notre conjecture sur un exemple⁴⁰ de taille $n=200$; toujours pour une 10-partition. La solution bruitée donne en 125 secondes une séparation de 25 595, contre 25 714 pour l'itéré, ce qui met à mal la conjecture formulée plus haut⁴¹.

En tout cas, l'attente envers le bruitage des variations pour fournir de bien meilleurs solutions se trouve actuellement déçue.

5.6 Concernant la notion de cohérence.

Désormais, on va s'intéresser aux algorithmes relatifs à la notion de cohérence probabiliste, et à leurs efficacité.

Dans ce qui précède, nous avons donc évalué des algorithmes devant théoriquement donner de bonnes solutions, relativement au critère de séparation. Néanmoins, un fait reste à ce niveau indiscutable: les solutions fournies par ces algorithmes sont extrêmement lointaines des solutions que l'on aimerait obtenir, car ne résistant pas aux tentations des singletons. De plus, nous avons travaillé avec des matrices dont les blocs diagonaux sont bien plus lourds (de poids tiré au hasard entre 0 et 100) que les poids externes (tirés au hasard entre 0 et 10), mais cela est peu représentatif de la disposition de données réelles; et lorsque les poids hors

³⁷Pour sa variance, nous ferons confiance au hasard.

³⁸Ou, le retour de la vengeance qui va faire mal....

³⁹Inexpliquée pour l'instant

⁴⁰bien qu'un seul et unique exemple ne puisse être considéré comme représentatif, mais ...

⁴¹comme quoi une seule matrice peut être représentative!

blocs diagonaux augmentent, les solutions sont de plus en plus aspirées vers les singletons. Il s'agit surtout de la notion de second voisinage qui supprime cette structure en classe.

Ainsi, sommes nous en droit à ce niveau de nous demander si le critère est bien adapté au problème, et donc si c'est bien le critère adéquate?

Evidement, nous ne pouvons juger ce nouveau critère de cohérence sur les mêmes bases que les anciens algorithmes.

Pour le tester, nous allons introduire une notion de qualité proche de l'adéquation aux classes initiales:

Définition 5 *Si une classe considérée de la solution fournie présente, à une exception près une classe entière (avec donc au plus un seul élément en plus ou en moins), nous l'accepterons, et dirons que cette classe de partition est en adéquation, ou adéquate, à une classe initiale.*

Débutons donc nos tests; de nombreuses procédures ayant été programmées, nous ferons de nombreux choix en cours de route.

Les matrices aléatoirement tirées seront toujours diagonales par blocs, a coefficients diagonaux entre 0 et 100, mais les poids extérieurs seront dorénavant tirés entre 0 et 30.

• *Une transformation proche du 2-opt.* Pour le moment, nous testerons donc cette transformation seule; le défaut, comme dit plus avant, est que le cardinal des différentes classes est fixé par la solution initiale aléatoirement générée.

Pour la matrice de test constituée des 8 heptagones, 5 classes de la solutions sont admissibles sur les 7 possibles! Rappelons aussi, que l'on peut aussi regarder la cohésion des classes trouvées afin de s'orienter vers les classes intéressantes.

Pour la matrice bloc perturbée et bruitée: n=50: là aucune des classes trouvées n'est adéquate à sa classe initiale. Néanmoins, on dira (grosso modo pour les valeurs que l'on observe) que lorsque la cohérence dépasse 0.5, ce n'est pas bon signe pour la classe correspondante; ici, la plus petite cohérence étaient de 11.13... Ainsi, est on averti du mauvais résultat⁴².

Pour n=60: Idem, aucune classe ne semble bien portante n=70: De même, nous ne pouvons nous en satisfaire.

Bannissons donc les algorithmes félons de nos programmes: utilisons d'abord un algorithme recherchant les classes "par le haut"⁴³ avant la transformation par les 2-opt ... résultat tout aussi médiocre ... surtout dû au fait que quelques classes se retrouvent trop grosse tres vite, et que les autres classes ne peuvent alors plus leurs reprendrent ce qu'elles ont perdues. Ce phénomén est bien sûr dû à la recherche des classes par le haut, quelques peu brutal.

Imbriquons alors une transformation 2-opt (afin de concentrer les ressources d'une classe vers les bonnes parties du graphes) avant que d'effectuer une recherche de classes par le haut (pour compléter ces classes): Le résultat n'est toujours pas à la hauteur de nos attentes; mais poursuivons, nous avons encore de nombreux algorithmes en réserve.

Tentons de coupler le 2-opt avec un algorithme effectuant k passages: à chaque passage, il fait une recherche de classe par le haut, et met de coté la meilleure des classe, qui ne sera plus ensuite remise en question.

Pour n=70, on observe un intéressant et curieux phénomén: une classe se trouve

⁴²rappelons que le delta était multiplié alors par 100

⁴³ie essaie pour une classe donnée de lui ajouter des éléments pour l'améliorer

être optimale, et sa cohésion associée est de 0.672; une autre classe, non optimale elle car recouvrant deux blocs diagonaux, se trouvent avoir un meilleur taux de séparation: 0.59.

Aussi étonnant que soit cette observation, elle n'en est pas moins explicable! Si l'on calcul les indices de séparation des classes relatives à la solution optimale, on constate que toutes n'ont pas le même indice (ce qui semble normal, puisque les poids diagonaux et hors blocs diagonaux sont tirés au hasard). Ainsi, il est tout à fait possible que deux classes ayant de toutes petites séparations donnent, après fusion, une classe dont l'indice de séparation reste encore un peu inférieur à celui d'une des plus mauvaises classes optimales!

Nous pouvons donc imaginer obtenir, non pas des classes découpées comme à l'initial, mais plutôt des classes englobant des sous graphes initiaux, avec bien sur à cotés quelques singletons: cette méthode crée néanmoins beaucoup moins de singletons que les précédentes. Un redécoupage manuel serait toujours possible suite à plusieurs itérations de cet algorithme (qui est très rapide) si l'on inclut un facteur aléatoire que l'utilisateur entre en début de programme, afin de modifier la solution initialement trouvée.

- L'algorithme évoqué plus haut présentant quelques qualités; détaillons le un peu. Il consiste en une boucle ayant k itérations; les éléments d'une classe répertoriée dans la liste tabou ne pourront servir à emplir d'autres classes.

Faire k fois les instructions suivante:

- ★ Effectuer une recherche par le haut pour toutes les classes.

- ★ Au final, placer la meilleure des classes construites dans la liste tabou.

Les avantages, en plus de sa rapidité, vont être mis en valeur sur les exemples suivants: (il tourne seul).

Pour la matrice test composée de 8 heptagones: une classe entière obtenue (les autres en englobant 2).

Pour ensuite $n=60$: 5 classes adéquate sur 10. (les autres n'étant pas absolument mauvaises ni catastrophiques).

Pour $n=80$: 3 classes adéquates sur 10.

Pour $n=100$: 3 classes encore sur 10. Ces classes ont par exemples des cohérences comprises entre 0.5 et 0.7. Notons encore que la classe ayant la plus petite cohérence dispose de 3 éléments hors de son bloc diagonal principal.

Ainsi, cette algorithme présente il, dû moins à mes yeux, un charme certain. Tentons de le parer d'autres atours pour vous le rendre irrésistible.

Insérons entre les phases de recherche par le haut et d'évaluation pour liste tabou, le programme (allégé) de 2-opt, et retentons nos sempiternels essais.

Pour $n=60$: 1 seule classe est adéquate, selon nos critères.

Pour $n=80$: 8 classes sur 10 sont adéquates !!!

Pour $n=100$: 3 classes (et 2 méritants bien d'y figurer).

On s'approche peu à peu de ce que l'on souhaite ...

- *De la complication du problème.*

Pour le moment, nous avons donc, comme dit à de trop nombreuses reprises⁴⁴, des matrices sous formes diagonales, mais cela implique un certain biais dans les mesures: en effet, les algorithmes explorent très souvent des boucles où il leur faut considérer des sommets les uns après les autres, ce qui, dans le cas où justement les sommets de même bloc se suivent, peut produire un défaut dans

⁴⁴que le lecteur pardonne cette lourdeur

le test. Pour révéler cet effet, deux nouveaux types de matrices seront mis en places⁴⁵.

La première aura ses éléments non plus les uns à côté des autres, mais espacé régulièrement. Comme cette régularité d'espacement pourrait lui même induire soit une pénalité, soit un biais, on utilisera aussi une permutation (valable sur des matrices (100,100)), qui codera la matrice; la solution sera passée à la décodeuse pour nous la rendre intelligible.

On test ce même programme à présent sur ces matrices.

Pour le premier type de matrice, on ne trouve plus, pour les 3 choix de n classique, de classes adéquates; néanmoins, cette façon de coder les éléments est la pire pour les algorithmes!

Pour la matrice (100,100) permutée grâce à une permutation (non régulière)... pas mieux! Ainsi, un certain biais peut être supposé dans les matrices précédentes.

De plus, nous n'avons pas, pour le moment, utilisé les classes présentant la caractéristique d'être incohérentes. Pourtant, comme il fût dit plus avant, l'incohérence pourrait être un précieux indice de classement.

Considérons le cas simplifié où toutes les classes trouvées par les solutions seraient de même cardinal.

Formons alors des vecteurs lignes, chacun représentant une classe, et dont les composantes sont les indices des sommets y appartenant. Ces vecteurs doivent donc être cohérents (ie minimiser le δ associé).

Par contre, si l'on considère le vecteur colonne formé par exemple des premières composantes de chacun des vecteurs précédents, alors nous obtenons une classe ayant k éléments, un de chaque classe: il semble alors logique d'affirmer que c'est un sous ensemble incohérent! Ainsi, autant nous devons minimiser le *delta* associé aux vecteurs lignes, autant nous devons maximiser celui associé aux vecteurs colonnes ... d'où une impression de dualité⁴⁶!

Néanmoins, nous pourrions désormais user de deux outils pour tenter d'optimiser notre solution courante : soit tenter améliorer la cohérence d'une classe donnée, soit tenter d'améliorer l'incohérence d'une "anti-classe" (ie k éléments, pris chacun dans une classe différente).

- *Le duel entre la cohérence et l'incohérence.*

Nous allons ici introduire une table, et reformaliser à l'aide de celle ci un certain nombre de transformations auparavant disparates; hélas, faute de temps, ces améliorations ne pourront être implémenté.

On considère donc toujours notre graphe ayant n sommets, et on se pose toujours le problème de sa k-partition.

Construisons dans une table ayant k lignes, et n-k+1 colonnes. Chaque ligne représente bien sûr une certaine classe; comme aucune classe ne peut être vide, nous en déduisons qu'il ne peut y avoir plus de n-k+1 éléments dans une classe donnée. Ainsi, nous avons ce nombre de colonnes.

Nous allons remplir ce tableau (initialement rempli de zéros) à l'aide d'une solution initialement tirée au hasard: Si le sommet d'indice i appartient à la classe d'indice j, alors, dans l'une des cases "libre" (ie ayant un zéro dedans) de la ligne j, on met un i. Ainsi, le tableau représente t'il bien un codage de notre solution.

⁴⁵pas pour bien longtemps néanmoins, car un nouveau type de matric test va bientôt être mis en place.

⁴⁶Impression néanmoins non confirmée.

Le problème est donc de maximiser la cohérence des classes, donc de minimiser le *delta* associé aux lignes, mais aussi maximiser l'incohérence (ie le delta) de chaque colonne.

Quels sont les transformations élémentaires associées à ces notations ?

Nous allons simplement utiliser le 2-opt: considérons 2 cases distinctes de la table: (k,i) et (k',j') . Le 2-opt consistera à échanger le contenu de ces deux cases.

Remarquons tout de suite qu'il peut ainsi échanger deux sommets de leurs classes respectives, comme aussi faire changer un sommet de classe, puisque dans le cas où l'une des deux case est vide, le 2-opt reviendrait au passage d'un sommet d'une classe à une autre.

Ainsi, nous constatons que cette transformation ci, qui devait améliorer la notion de cohérence, généralise aussi la notion de premier voisinage. En fait, on s'aperçoit bien vite en y réfléchissant un peu, qu'il y a un réel lien entre la notion de premier voisinage, et cette notion de cohérence, puisque augmenter par exemple le poids des arcs intra-classe permet à priori d'améliorer la cohérence de cette classe (puisque l'on devrait s'éloigner de la moyenne); cela dit, le cardinal intervient dans la formule de la cohérence, diminuant grandement les petites augmentations pour les classes un peu trop volumineuses.

Mais, nous savons que l'un des points faibles de la transformation associée au premier voisinage, est de bloquer un peu trop vite à cause de cette restriction qui est de ne pouvoir faire passer qu'un seul sommet à chaque coup. Ici, a t'on mieux?

Déjà, on bénéficie de la possibilité de permuter deux éléments, mais, à moindre frais, nous pouvons faire mieux!

Si l'on veut réaliser une permutation de p sommets, comment s'y prendre? Par exemple, supposons les codés dans la table par: $\{(k_1; i_1)(k_2; i_2)(k_3; i_3)...(k_p; i_p)\}$. Comme il est bien connu que les transformations engendrent le groupe des permutations, il suffit de combiner à la suite un certain nombre de transpositions, donc des 2-opt, pour aboutir au même résultat. Dans l'exemple cité par exemple, on considérera les transpositions:

$$\{(k_1, i_1)(k_2; i_2)\}\{(k_1, i_1)(k_3; i_3)\}\{(k_1, i_1)(k_4; i_4)\}... \{(k_1, i_1)(k_p; i_p)\}$$

Ainsi, nous pourrions choisir un p , réaliser p transpositions sur une table dupliquée, puis regarder à posteriori si ces p transformations n'améliorent pas la cohérence de la solution: si oui, alors remplace la vraie table par la table dupliquée, si non, on efface la copie pour ne retenir que l'initiale.

Enfin, comment se servir concrètement de l'incohérence? On notera anti-classe une collection de k éléments, chacun pris dans une classe différente. Il serait absolument souhaitable qu'une permutation des sommets renforce l'incohérence des deux anti classes respectives.

- *Un retour sur nos pas.* Reprenons en effet les algorithmes relatifs à la première notion de voisinage, qui est donc extrêmement proche de la transformation envisagée pour la notion de cohérence. Rappelons déjà leur défaut principal avant que de poursuivre: cette notion de voisinage ne permet pas, entre deux solutions voisines, d'échanger plus d'un sommet en même temps. Cela, comme nous l'avons vu plus haut pourrait être contourné grâce à une modification de l'algorithme qui ne calculerait pas l'amélioration entre deux solutions voisines de manière systématique, mais de temps à autre accepterait plusieurs

transformations sans s'inquiéter de leurs améliorations du critère. On retrouvera ce genre de méthodes, dite avec mutation, dans un article d'I.Charon, et O.Hudry. Mais, un bon point de cette transformation, est que, si une partie du graphe se trouve être à la fois cohérente et dans l'une des classes d'une partition initiale, alors, cette transformation présente moins de risque d'oublier ce précieux acquis. En effet, si on tente de faire sortir l'un des sommets composant cette partie cohérente, les autres sommets feront alors bloc pour le garder avec eux ... une sorte de force de rappel, ce qui n'est pas le cas, par exemple, des descente basée sur la seconde notion de voisinage qui accepte de nombreux changements de sommets en même temps ... nous l'avons déjà vu. Cette force ne serait néanmoins pas de taille à lutter face à une éventuelle énorme classe: imaginons par exemple que la classe cohérente soit un hexagone complet valué à 100; la force de rappel de n'importe quel sommet de cet hexagone est de 500. Mais, si une classe présente 60 sommets, tous ayant une arrête modestement valuée à 10 avec le sommet baladeur, alors les jeux sont fait.

Ainsi, serait il utile de tenter déviter qu'une classe ne prenne l'ascendant au niveau du cardinal sur les autres. Pour ce faire, il nous suffit de modifier notre algorithme; au lieu de considérer les sommets de manière cyclique, nous allons considérer les classes.

L'algorithme modifié devient alors:

Tant que la boucle suivante à un effet, la répéter.

★ Considérer tour à tour chaque classe (donc pour i variant de 1 à k).

★ Considérer le meilleur élément extérieur à cette classe; si l'ajouter améliore le critère, alors, l'ajouter.

Cet algorithme présente à ce niveau un défaut: il ne garanti pas pour le moment qu'une classe ne disparaisse pas absorbée par l'appétit des autres classes ... Mais laissons le tel quel pour le moment, et tentons de le faire tourner sur de nouvelles matrices tests⁴⁷. Cette dernière génération de matrice test présente tout pleins d'avantages: simplicité de génération, souplesse d'utilisation ... Il suffit intialement de tirer une solution au hasard qui sera considérée comme la solution optimale. Puis, on génère une matrice autour de cette solution: si par exemple i et j sont deux indices de sommets étant dans la même classe dans la solution optimale, alors le coefficient de la matrice $S_{i,j} = S_{j,i}$ sera tiré aléatoirement dans $[0, \text{poids max}]$. On bruite ensuite tout les coefficients de la matrice par un nombre pris aléatoirement dans $[0, \text{bruit max}]$. Enfin, un autre avantage sera d'avoir une lecture tres simple des résultats, ce qui nous aidera par la suite pour traiter de manière répétée des test sur la notion de cohérence des classes. Bien: allons y donc ! ... $n=100$ et $k=10$ comme toujours.

Pour $\text{poids}_{max} = 100$ et $\text{bruit}_{max} = 10$, on trouve, pour cet algorithme une solution de séparation 45 904 contre 40 371 pour la séparation obtenue par la solution optimale. De plus et surtout, 5 classes obtenue se trouvent etre cohérentes (et aucune n'est absurde, sauf une réduite à un singleton).

Pour $\text{poids}_{max} = 100$ et $\text{bruit}_{max} = 20$, 88 755 contre 84 341 pour solopt, et encore 5 classes en adéquation avec les classes de solopt.

Pour $\text{poids}_{max} = 100$ et $\text{bruit}_{max} = 30$, 6 classes en adéquations avec celles de solopt, et une séparation de 13 431 contre 12 9881 pour solopt (la solution optimale générant la matrice S).

Pour $\text{poids}_{max} = 100$ et $\text{bruit}_{max} = 50$, 5 classes adéquates, et des séparations

⁴⁷encore, mais les dernière cette fis ci!

similaire (224 000 contre 221 000).

Cette fois ci, il semble y avoir une méthode répondant assez bien à l'impératif d'obtenir de bonnes classes au sens de l'adéquation. Reste peut être à comparer avec des méthodes de descente, pour voir le critère de séparation: Comme on peut l'imaginer, les singletons réapparaissent pour les autres méthodes et donnent de bien bons résultats relativement au critère de séparation. Sur la dernière matrice par exemple, que des singletons sauf une classe, pour une séparation évaluée à 50 000, soit 4 fois moins au niveau de la séparation trouvée par des algorithmes recherchant la cohérence des classes ce qui confirme à mon sens que ce critère de séparation n'est pas bien utile pour donner de bonnes solutions, relativement au problème et aux sens des solutions que l'on veut lui donner.

6 Les résultats des tests.

Dans cette section, nous allons présenter différents tests relatifs aux algorithmes; nous ne remettons plus en causes ces derniers, mais simplement nous nous contenterons de parfois les rappeler, donner leurs noms pour pouvoir éventuellement se reporter en annexe au code source, et donc refaire certains de ces tests.

6.1 Relativement au critère de séparation.

- *Vitesse relative à la première notion de voisinage.* L'algorithme le plus simple consiste en le programme "descente simple"; il considère les sommets dans l'ordre cyclique, et les met les uns à la suite des autres dans leurs meilleurs classe (éventuellement celle où ils sont déjà) jusqu'à ce que l'on fasse une boucle sans rien changer. Cet algorithme est extrêmement rapide, testons le donc à chaque fois, sera donné la moyenne sur 10 matrices tests, en fonction des choix de n nombre de sommets, et k nombre de classes demandés:

	k=10	k=15	k=20	k=30	k=50
n=100	11s	13s	16	22s	36s
n=120	16s	17s	22s	30s	44s
n=150	21s	29s	33s	41s	75s

- *Nombre d'itérations.* Le but ici est d'avoir une idée sur le nombre d'itérations (complète, ie en repartant d'une solution tirée au hasard) moyen afin d'obtenir un meilleur minimum local. Le test considérera 30 matrices, et sur chacune d'elles fera 1000 itérations: on affichera alors le l'indice p moyen de l'itération donnant le meilleur résultat. Le but de ce test sera, lorsqu'il faudra le comparer avec d'autres méthodes, de pouvoir éventuellement changer de stratégie apres un nombre jugé suffisant d'itérations.

	sur 1000 itérations	pour 2000 itérations
n=50, k=5	p= 50	p=50
n=70, k=7	p= 67.	p=69
n=100, k=10	p= 96	p=99
n=120, k=12	p= 115.	p=119

Ainsi, il semble raisonnable de ne pas utiliser cette méthode plus de 100 ou 200 fois, sous peine de la voir piétinée sur un même résultat. Il ne semble pas utile de tester pour 3000 itérations compte tenu du temps assez long mis à l'exécution du programme pour 2000 itérations et n=100 ou 120. Cette méthode sera considérée, au vue de sa simplicité, comme la mesure étalon pour les autres algorithmes.

- *Algorithme second voisinage fusion.* On test ici le programme de descente baptisé "descente fusion", et relatif à la seconde notion de voisinage, modifié⁴⁸. Nous allons tester différentes versions (ie avec ou sans programme de bipartition itéré comme amorce). On test déjà, sur 30 matrices de taille donnée, l'amélioration de la solution par l'augmentation du temps acrodé au processeur; on donne la moyenne des résultats.

	3s	5s	7s	10s	15s	20s
n=50,k=50	24 000	23 450	23 150	23 000	22 250	21 600
n=100,k=10	75 300	74 000	73 700	69 950	70 000	69 450

Ainsi, on peut déjà constater que cette méthode trouve de meilleurs solutions avec le temps; ainsi, on ne peut dire, comme de fût le cas de la méthode itérée qu'il suffit de l'arreter à un certain nombre d'itérations ou de temps afin qu'elle obtienne son bon score.

Testons maintenant le programme de bipartition itéré (celui garantissant un rapport d'approximation).

	temps moyen pour 30 matrices
n=50,k=5	3sec
n=70,k=7	16 sec
n=100,k=10	80 sec = 1min 20
n=120,k=12	185 sec = 3 min

On constate donc que cet algorithme se retrouve bien gourmand en temps; il peut néanmoins donner une bonne solution initiale, mais ne garantit pas que celle ci soit un minimum local. Ainsi, une descente à partir de la solution fournie serait elle utile par la suite.

- *Premiers tests entre descente simple et descente fusion.* Donc, on cherche ici obtenir la meilleur performance possible entre les deux algorithmes, selon différents cas. En plus des critères habituellement testé⁴⁹, on testera aussi en même temps sur des matrices dont le bruit b varie; auparavant, il était de 10 ou de 20; ici, on fera apparaitre le choix du bruit. En effet, il semblerait (sur des test préliminaire) que la seconde notion de voisinage soit plus sensible au bruitage que la descente base sur la première notion de voisinage. On testera sur quelques matrices (1,2, 5 ou 10 selon les tests). En ce qui concerne les tests amorcés par bipartition itérée, seul une descente sera faites, simplement afin de savoir si le résultat est meilleur ou non. A divers reprises, effectuer une boucle d'un certain algorithme prendra plus de temps que le temps accordé aux autres méthodes, on le signalera alors ... Enfin, l'idée est de tester sur un tout petit nombre de matrice (50,50) ou (100,100), pour un bruit donné, puis augmenter le bruit...

	Descente simple +descente mutante	bipartition itérée +descente simple	descente fusion	bipartition itérée +descente fusion
(n,k)=(50,5),b=0,t=1min	0 (sol. opt.)	0	0	0
(n,k)=(100,10),b=0,t=1min	0	0	1 000	0
(n,k)=(50,5), b=30, t=1min	5 906	6 628	6 628	6 628
(n,k)=(100,10), b=30,t=1min	26 500	26 940 (1min)	26 920 (3min)	26 920 (1min)
(n,k)=(50,5), b=50, t=1min	9 225	10 430	10 437	10 437
(n,k)=(100,10), b=50, t=3min	40 000	45 000 (1min)	44 880 (4min)	44 880 (3min)
(n,k)=(50,5), b=80, t=3min	13 500	16 230 (3 sec)	16 230	16 230 (30sec)
(n,k)=(100,10),b=80,t=3min	60 550	68 425	68 425	68 425

On constate donc une certaine similarité dans les résultats des dernières méthodes pour ces quelques tests.

- *Pour des matrices tres faiblement perturbées.* On va tester ici une classe de matrice qui devrait être assez rencontrée dans l'exemple biologique soujacent

⁴⁸La fusion étant de loin une sorte de recuit simulé

⁴⁹n=nombre de sommets, k=nombre de classe de partition, t temps accordé

notre étude. Il s'agit de matrices que l'on qualifiera de tres faiblement perturbées. Pour simuler ces matrices, on utilisera aussi la souplesse des matrices tests finalement mise en évidences. Le poids des arrêtes "intra classes" sera tres faible, de l'ordre de 5 (et non 100 comme auparavant), et le bruit b de la matrice sera d'environ 1. Ainsi, se retrouvera t'on avec une matrice⁵⁰ ayant relativement peu d'arrête extra classe, berf, des matrices tres peu denses. Le bruit b représentera ici le couple (poids,bruit). On considérera dans les premiers exemples 10 matrices, puis le temps augmentant surtout pour la méthode de descente fusion, juste 2 matrices.

	Desc. simple +desc. mutante	bipart. itérée + desc. simple	desc. fusion	bipart. itérée + desc. fusion
n=50,k=5,b=5,t=15sec	0	0	9	0
n=100,k=10,b=5,t=15sec	15	0	37	0
n=50, k=5, b=(10,2), t=15s	390	330	329	327
n=100,k=10, b=(10,2), t=15s	1 327*	1 080* (10min)	1086* (3min)	1080* (4min)

*: Pour cette ligne ci, on teste sur 2 matrices, et non sur 10.

Le programme de descente fusion a été modifié juste cette ligne afin de tenter de le rendre bien plus rapide qu'il n'est pour le moment...

On augmente aussi le nombre d'itérations basique de decente simple, afin de tenir compte de l'augmentation du temps accordé.

6.2 Sur la notion de cohérence.

Nous allons maintenant aborder la dernière partie des tests, mais ausi la plus importante. En effet, le lecteur psychologue n'aura pas eût de mal à discerner un léger scepticisme⁵¹ de l'auteur quand à l'expressivité du critère de séparation. Ainsi, et plus que tout autre, le critère de cohérence probabiliste proposé doit être testé.

Plusieurs tests seront nécessaire: déjà, déterminer la performance de l'algorithme principal implémenté relativement à ce critère. Ensuite, comme nous avons proclamé que le delta associé à une classe pouvait donné une bonne idée de la cohérence de celle ci, indépendamment du fait que la solution soit un minimum local de la descente ou non, il nous faut tester cette affirmation. Un premier phénomen bien curieux et allant contre a déjà été mis au grand jour dans le chapitre précédent. Néanmoins, la classe qui en englobe deux réelles du graphe ne semble pas non plus être une abération (contrairement au singleton perdu ailleurs, mais lui est facilement éliminable lors de recherche d'information sur le problème initial.). Bref, nous tenterons de réaliser aussi prochainement un dispositif recherchant des classes à la fois incohérentes (au niveau de la solution que l'on sait être optimale), et pourtant ayant de bons estimateurs delta. Mais pour le moment, recherchons la cohérence: l'algorithme testé se nomme ici "opt classe bis" : c'est celui effectuant k boucles, et chaque passage, il retient la classe la plus cohérente (sans réemprunter les éléments de celles déjà trouvées). C'est cet algorithme au final, qui semble donner les meilleurs résultats! Néanmoins, deux sous algorithmes s'affrontes à l'intérieur de lui: on construit à chaque itération ces fameuses classes du graphe par un procédé de recherche de classe par le haut, mais on peut ensuite tenter d'améliorer ce bru-

⁵⁰ que l'on espèrerais connexe .. mais qui ne le seront pas forcément

⁵¹ Car s'il est d'évidence que les mathématiques soient objective, il n'en est pas de même de ceux qui les produisent.

tal algorithme. C'est là qu'il y a choix: soit grace à un algorithme proche du 2-opt, soit par l'algorithme dit de descente (premier voisinage) équilibré, qui ne considère plus de manière cyclique les sommets, mais les classes (évitant ainsi probablement le fait qu'une classe ne devienne trops grande trops vite). Nous allons donc déjà organiser un combat rituel entre ces deux algorithmes et l'algorithme témoin, celui avec juste la recherche des classes par le haut, sans modification aucune. Enfin, nous testerons tout ceci sur pas moins de 20 matrices à chaque fois! On donne le nombre moyen de classe cohérente trouvées (l'optimum étant de trouver k classes en adéquation avec des classes initiales).

	Rech. par le haut solo	avec 2-opt	avec Desc. équilibrée
n=50,k=5,b= 0	4.1 en moyenne	3.3	3.8
n=100,k=10,b=0	6.3	4.3	6.6
n=50,k=5,b=30	3.6	2.9	2.9
n=100,k=10,b=30	6.1	2.7	4.4
n=50,k=5,b=50	4.1	2.3	3.1
n=100,k=10,b=50	5.4	1.95	4.7
n=50,k=50,b=70	3.1	2.1	2.8
n=100,k=10,b=70	4.65	1.8	5.8
n=50,k=5,b=100	2.9	1.2	2.4
n=100,k=10,b=100	2.2	0	4.6

A noter de plus que la méthode avec le 2-opt est bien plus gourmande en temps que les deux autres.... Ainsi, la méthode témoin et celle utilisant la descente équilibrée semblent t'elles donner de bon résultats. La méthode avec descente équilibrée donnant en moyenne une classe sur deux en adéquation avec l'une des classes initiales, et ceci semble peu perturbé par la taille de la matrice et le bruit, alors que le première méthode témoin semble etre plus sensible à la conjonction de ces deux facteurs...

Nous allons tester maintenant un autre aspect de ces deux méthodes survivantes: on se donne une matrice, et on regarde combien d'itérations leurs sont nécessaires afin d'obtenir une solution optimale (ie ayant k classe en adéquation avec les classes initiales); on stoppe chacune des méthodes qui auraient dépassées les 2 mins de recherche infructueuses.

	Recherche solo	+Desc.Equilibre	#classe de solo	d'Equilibre
(n,k)=(50,5),b=0	6 en 1s	6 en 1s	4 en moy.	4.5 de moy.
(n,k)=(100,10),b=0	4 en 5sec	19 en 6sec	6.8 en moy.	7.3 de moy.
n=50,k=5,b=30	53	54 en 4sec	3.7 de moy	1.9 en moy.
n=100,k=10,b=30	11 en 3sec	11 en 9sec	7.9 de moy	6 de moy.
n=50,k=5,b=50	129	130 en 8sec	3.2 de moy	1.9 de moy.
n=100,k=10,b=50	11 en 8s	11 en 8sec	6.4 de moy.	7.2 de moy.
n=50,k=5,b=70	141	141 en 8sec	2.8 en moy.	1.7 de moy.
n=100,k=10,b=70	0 dans les temps	aucune	5.5 moy.	5.5 de moy.
n=50,k=5,b=100	0 dans les temps	aucune	1.5 moy.	2.5 de moy.
n=100,k=10,b=100	0 faute de temps	aucune	2.5 moy.	4.6 de moy.

Compte tenu des résultats de fin de tests, on doit renoncer aux k classes adéquates dans un temps raisonnable. On remarque aussi que bien que les 2 algorithmes donnent la solution optimale à peu près en même temps, la tendance à résister aux augmentations de tailles et de bruits pour la descente avec algorithme équilibré se perçoit ici aussi.

Avant de nous intéresser en dernier à comment trouver des classes adéquates dans ces cas difficiles, intéressons nous au lien entre l'adéquation et aux delta associés à la solution. En effet, inutile de s'acharner à optimiser le delta si ce dernier ne permet pas dans certains cas de trouver des classes cohérentes!

Pour se faire nous allons introduire la valeur que le lecteur souhaite depuis déjà bien longtemps poser: celle de la cohérence totale d'une solution.

Définition 6 *La cohérence totale d'une solution sera la somme des cohérences (ie des delta) des classes de cette solution.*

Nous allons alors tenter de minimiser cette notion par un algorithme de descente ci dessus.

Le test? On considère une matrice, de taille $n=100$, de poids et de bruit égaux eux aussi à 100, et on cherche une 10 partitions. On s'arrete soit quand le temps est épuisé (100 sec CPU), soit quand la solution optimale est obtenue. Au cours de l'algorithme, on retient la solution de cohérence générale la plus petite rencontrée jusqu'alors et on la compare avec ... une solution spécialement créée pour l'occasion: solopt elle même! Et, pour ce premier test, il est plutôt concluant: la classe la meilleur obtenue présente 2 classes non optimales (donc 8 optimales), et présente aussi une séparation globale plus forte que la séparation de solopt. Rappelons un point d'importance ici injustement pass{e sous silence: les singletons sont fortement pénalisés dans le calcul de la cohérence ie du delta; en effet, un test préliminaire regarde le cardinal de la classe a tester (programme du nom de donne proba); si jamais c'est un singleton, alors la valeur 1000 est renvoyée, ce qui est extrêmement pénalisant. Notez bien ici que l'on aurait bien du mal à définir un delta par un moyen plus respectueux du parallèle probabiliste puisque un singleton ne présente bien sûr aucune arrête, ce qui fait un bien maigre échantillon ... Mais, si jamais solopt présentait⁵² un singleton, on ne pourrait pas, à cause de cette manipulation sur le delta des singletons, la trouver ... Ainsi, faudra t'il veiller aux sommets absurdes éventuels des matrices.... Comme un seul test parrait bien peu, multiplions les alors! On laisse 50sec temps CPU à chacun des algorithmes. A chaque fois, il n'est ici fait mention que d'une seule matrice! On note entre parenthèse le nombre de classes en adéquation avec la solution initialement choisie (solopt) dans les cas où la solution finalement trouvée ne soit pas solopt.

	Opt.	Rech.Solo	Rech.Equilibre
n=50,k=5, b=0	0.054 372	sol. opt.	sol. opt.
n=100, k=100,b=0	0.034 230	sol. opt.	0.034 889 (10)
n=50,k=5,b=30	0.059 683	sol. opt.	sol. opt.
n=100,k=10,b=30	0.040 340	sol. opt.	0.041 407 (10)
n=50,k=5,b=50	0.070 899	sol. opt.	0.070 899 (5)
n=100,k=10,b=50	0.048 742	sol. opt.	sol. opt.
n=50,k=5,b=70	0.065 672	! 0.065 620 (5) !	0.069 150 (5)
n=100,k=10,b=70	0.054 918	0.055 611 (10)	0.057 158 (10)
n=50,k=5,b=100	0.107 436	! 0.105 840 (3) !	0.108 519 (5)
n=100,k=10,b=100	0.087 070	0.090 440 (10)	0.091 115 (10)

Ainsi, à part une fois où la cohérence générale est passée sous la cohérence associée à solopt et où cela a permis de perdre deux classes adéquates, on retrouve à tout coups une solution en adéquation avec la solution initiale. D'où une bien naturelle interrogation relative au tableau qui précédait celui ci ... comment est ce possible? Une erreur de programmation ? ... de réalisation ? ...

Refaisons finalement le tableau qui précédait donc, mais cette fois ci en permettant aux méthode de tomber sur une solution dont au plus une classe est incohérente. On retiendra en fait seulement la solution minimisant la cohérence globale de la solution. On recommencera directement au cas du bruit $b=70$.

⁵²ce serait son droit après tout

	Par Rech. seule	Par Rech.+ Equilibre
n=50,k=5,b=70	9 ^{ieme} itération pour opt.	141 ^{ieme} itération pour opt
n=100,k=10,b=70	57 ^{ieme} itér. pour opt	9 classes par manque de temps
n=50,k=5,b=100	5 ^{ieme} itération pour opt	2 ^{ieme} itération.
n=100,k=10,b=100	9 classes adequates	7 ^{ieme} itération.

Notons ici que quand on dit sol opt, cela serait plus exactement une solution ayant k classes adéquates, avec cette petite subtilité que l'on puisse avoir un sommet en plus où en moins dans cette classe. Ainsi, sans savoir comment expliquer l'ancien tableau, nous voila rassuré : l'algorithme de descente solo par recherche de classe par le haut semble bien convenir, en un temps raisonnable, à fournir une solution que l'on pourrait penser comme ayant un grand nombre de classes cohérentes . Ainsi, il suffira aux éventuels utilisateurs de simplement laisser tourner leurs machines disons une minute, et de sauvegarder lors de cette descente la solution ayant la plus petite cohérence totale.

Un dernier test s'imposerait, non plus sur la recherche de classes incohérentes, puisque lors de nos derniers tests, nous avons généré et brassé un grand nombre de matrices, sans que de criantes aberrations n'apparaissent. Il faudrait plutôt s'intéresser à la robustesse de la méthode: permet elle de retrouver des classes cohérentes, non seulement quand il y a du bruit, mais aussi quand on ne donne pas le bon nombre k de classes à chercher , et puis que ce passe t'il quand il n'y a rien a trouver ? Tout cela, nous le saurons dans le prochain chapitre, intitulé:

• *Je fais rien que des bêtises!*

Voici donc le moment du joyeux n'importe quoi, afin de savoir si notre méthode compensera nos erreurs, nos fatigues, nos rêveries ...

Sur les erreurs du nombre de classes. On laisse ici seulement 20 secondes CPU, et seulement pour la méthode dite solo⁵³

	On demande k' classes	Nbr de classes adéquates
n=100,k=10,b=50	k'=9	8 (ie une est double)
n=100,k=10,b=50	k'=8	7
n=100,k=10,b=100	k'=9	8
n=100,k=10,b=100	k'=8	5
n=100,k=10,b=100	k'=11	8
n=100,k=10,b=100	k'=12	5

Maintenant, regardons jusqu'ou l'on peut bruite la matrice initiale en conservant une certaine validité de la meilleur solution, toujours obtenue par Rech.Solo, en 20sec CPU.

	Nbr de classes adéquates
n=100,k=10,b=150	4
n=100,k=10,b=160	0
n=100,k=10,b=170	0

Et s'il n'y a rien à trouver? Alors, la cohérence de la classe devrait etre assez grande, supérieure peut etre à ce quelle aurait dû etre... présence de singletons faisant exploser la cohérence de la solution finalement obtenue ... en tout cas, sur les quelques essais machines, on obtenais une cohérence bien supérieure à celle d'ordinaire ...

⁵³Car faudrait pas abuser des bêtises et y prendre gout!

7 Que reste il à faire?

En tout cas directement lié aux quelques points invoqués, mais non réalisés!

- *Relativement à l'idée de table* On pourrait mettre en place cette fameuse table, permettant d'effectuer une descente pouvant faire des échanges multiples de sommets lors d'une même itération. L'idée, plutôt que d'effectuer une descente pouvant éventuellement, selon une probabilité décroissante, effectuer des sauts plus grand, serait d'effectuer une descente classique jusqu'à un optimum local, et de celui ci, tenter de franchir l'éventuelle barrière locale grâce à un élargissement de la notion de voisinage. A titre comparatif, l'image donnée pour les méthodes acceptant des remontées est celle d'un alpiniste perdu dans la montagne et cherchant le refuge au pied de celle ci. Il prends les chemins descendants, jusqu'à ne plus en trouver: alors, il remonte par le chemin de plus petite pente.

Maintenant, imaginons que notre alpiniste malchanceux ait appris que ses méseventures servaient à illustrer les livres de recherche opérationnelle, et qu'il décidât que l'on ne le reprendrait plus à se perdre en montagne: puisque le refuge se trouve en pied de celle ci, il achète à fort grand prix, un matériel GPS indiquant l'altitude; ce matériel se compose de balises, et d'un propulseur. Ainsi équipé, il a l'espoir, s'il se perd, d'envoyer tout autour de lui des balises: dès que l'une d'elles indique une altitude inférieure à la sienne, il s'y rends, et redébutte sa descente à pieds ... jusqu'au prochain minimum local, ou jusqu'au refuge. Ainsi, ici, l'idée serait tout autre, inverse même de celle acceptant une remontée sur une solution voisine: On n'accepte aucune remontée, mais par contre, une fois à un minimum local, on accepte donc d'élargir la transformation élémentaire, afin de tenter d'obtenir mieux. Il serait aussi bon de tester, non pas une probabilité décroissante de saut, mais bien une probabilité croissante de saut: il tente déjà de trouver un tel palier directement à coté de lui, sans succès. Alors, il passe à une recherche à un double voisinage (exhaustif ou non), puis à un voisinage triple, et ainsi de suite, jusqu'à épuisement des forces, des balises ou du propulseur de notre alpiniste. Au niveau de l'algorithme relatif à la table, cela devrait se traduire par une probabilité croissante d'accumuler des 2 opts (afin d'obtenir des transformations bien plus complexe, en fait, on devrait pouvoir obtenir n'importe quelle transformations) avant que de regarder où l'on est.

- *Relativement à l'incohérence.*

On pourrait aussi s'intéresser à l'implémentation du critère d'incohérence, et peut être le faire intervenir afin parfois de séparer les classes (plus que de tenter de les augmenter). Comment? Peut être en usant de condition double pour le transfert d'un sommet (diminuer le delta des deux classes et, en même temps, augmenter le delta associé à une classe transverse), où encore regarder de plus près les possibilités offertes par les anticlasses (k sommets, chacun appartenant à l'une des classes de la partition)....

- *Relativement au programme de second voisinages.*

Si le programme de descente par bipartition intéresse, regarder si en l'allégeant (en ne faisant pas par exemple la totalité des coupes ou des appel à Ford et Fulkerson), on ne le rendrait on pas toujours aussi bon au niveau de la descente mais bien plus rapide qu'il n'est pour le moment? (c'est son point critique majeur!). De même, il faudrait alléger beaucoup le programme avec fusion: celui ci doit simplement promener un peu la solution dans de nouveaux paysages: il est absolument inutile d'être aussi exhaustif qu'on ne le fait ici; pour cet algorithme en tout cas, ne pas effectuer l'intégralité des calculs.

8 Bibliographie.

- *Sur le problème initial et son lien avec la biologie:*

A.Guénoche et F. Denizot: "Stratégie de séquençage par ordonnancement de contigs.

Y.Quentin, G. Fichant- ABCdb; an ABC transporter database. J.Mol.Microbiol. Biotechnol, 2000, 2pp 501-504.

- *Sur les problème de complexité de la k-partition:*

"A polynomial algorithm for the k-cut problem for fixed k" d'O.Goldschidt et D. S. Hochbaum, dans le numéro 1. Fevrier 1994 de Mathematics for operations research.

- *A propos des problèmes de partitionnement:*

A.Guénoch: Partitions optimisées selon différents critères: évaluation et comparaison.

- *Sur bipartition itéré:* Cet algorithme fut présenté initialement dans l'article de E.Dahlaus, DS Johnson, C.H. Papadimitriou, P.D Seymour, M. Yannakakis: " The complexity of multiterminal cuts." SIAM Journal of computing.23: 864-894. (1994).

Cet algorithme fût repris dans l'ouvrage de Vazirani: " Approximation Algorithms " chapitre 4. Multiway Cut ant k-cut pp39-46 Springer 2001.

- *Pour les différentes méthodes de bruitages:* On se reportera aux articles de: I.Charon, A Germa, O.Hudry: "The noising method: a new methode for combinatorial optimization" dans Operations research letters vol 14, num3 d'octobre 1993, et suivants ainsi que dans le futur: "Métaheuristiques et outils nouveaux en recherche opérationnelle", chapitre de I.Charon, A Germa, O.Hudry, à paraitre donc chez Hermes.

Ou encore dans l'article "A new metaheuristic called descents with mutations applied to the aggregation of relations" relatif des méthodes de descentes bruitées par des mutations.