الجمهورية الجزائرية الديمقراطية الشعبية
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

وزارة التربية الوطنية
MINISTERE DE L'EDUCATION NATIONALE

ECOLE NATIONALE POLYTECHNIQUE

DEPARTEMENT GENIE INDUSTRIEL

PROJET DE FIN D'ETUDES

SUJET

TEST D'EFFICACITE D'ALGORITHMES POUR LE PROBLEME DU VOYAGEUR DE COMMERCE

Proposé par : Mr Vangélis Paschos

Etudié par : Melle Lamia Caidi

Dirigé par

PROMOTION
1995

الجمهوريـــة الجزائـــريـــة الديمقـــراطيـــة الشـعبيـــة
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

وزارة التربيـــة الوطنيـــة
MINISTERE DE L'EDUCATION NATIONALE

ECOLE NATIONALE POLYTECHNIQUE

DEPARTEMENT GENIE INDUSTRIEL

PROJET DE FIN D'ETUDES

SUJET

TEST D'EFFICACITE D'ALGORITHMES POUR LE PROBLEME DU VOYAGEUR DE COMMERCE

Proposé par : Mr Vangélis Paschos

Etudié par : Melle Lamia Caidi

Dirigé par

PROMOTION
1995

E.N.P. 10, Avenue Hacen Badi El-Harrach - ALGER

PLAN

Introduction

A l'essor des calculateurs électroniques modernes, la théorie de la complexité algorithmique prend une place de plus en plus remarquable et importante dans la science contemporaine.

Elle représente le principal outil théorique qui permet de juger la possibilité de résoudre efficacement, à l'aide d'ordinateurs, des problèmes dont les solutions sont en nombre fini mais considérablement élevé.

Parmi ces problèmes d'origine souvent concrète, on peut citer: des problèmes d'optimisation d'emplois du temps, des problèmes de découpe industrielle, des problèmes d'optimisation de réseaux de télécommunication ainsi que des problèmes d'optimisation de tournées de livraison ou d'ordonnancement.

Cette dernière catégorie de problèmes conduit à un problème majeur traité par la théorie de la complexité, celui du célébrissime « voyageur de commerce ».

Dans notre mémoire, nous aborderons un sous-problème de celui-ci à savoir le problème du voyageur de commerce symétrique avec des distances 1 et 2.

Notre travail consiste à tester l'efficacité de deux algorithmes pour ce cas du problème du voyageur de commerce, proposés par PAPADIMITRIOU et YANNAKAKIS dans leur article: (The traveling salesman problem with distances one and two, 1993).

Il faudra donc programmer ces algorithmes afin de comparer la valeur de la solution approchée qu'ils fournissent à celle de la solution optimale produite par un algorithme exact, puis effectuer des tests sur des instances de taille réduite.

Le cheminement adopté dans ce mémoire s'articule autour de cinq parties:

Une première partie consistera à présenter les concepts de la théorie des graphes qui seront utilisés ainsi que les différentes notions que renferme la théorie de la complexité algorithmique.

Une seconde partie sera consacrée à la présentation du problème du voyageur de commerce en commençant par le définir puis en énumérant certains résultats obtenus à son sujet.

La troisième partie sera axée sur l'explication en détail des deux algorithmes étudiés ainsi que la preuve théorique de leur efficacité. Elle sera suivie d'une partie qui consistera à décrire le travail de programmation effectué en explicitant les différentes procédures élaborées.

Enfin dans une dernière partie, seront fournis les résultats des tests d'efficacité des deux algorithmes.

1- Préliminaires

1-1- Quelques concepts de la théorie des graphes

Nous nous contenterons dans cette partie de rappeler les principales définitions qui serons utilisées dans le mémoire.

Le concept de graphe:

(1) Orienté:

Un graphe G(V,E) est déterminé par la donnée :

- d'un ensemble V dont les éléments sont des sommets (noeuds).

Si T est le nombre de sommets, on dira que le graphe est d'ordre T.

On suppose que les sommets sont numérotés i=1,2,...,T.

- d'un ensemble E dont les éléments e $\in$ E sont des couples ordonnés de sommets appelés arcs.

e(i, j) est un arc dont l'extrémité initiale est i et l'extrémité finale est j.

(2) Non orienté:

Dans ce cas, l'orientation des arcs ne joue aucun rôle. On s'intéresse simplement à l'existence ou la non-existence d'un arc entre deux sommets.

e = (i, j) = (j, i) est appelé arête.

Définitions:

♦Chemin de longueur q:

C'est une séquence (v1,v2,...,vq) d'arêtes de G telle que chaque arête ait une extrémité en commun avec l'arête précédente, et l'autre extrémité en commun avec l'arête suivante.

♦Cycle:

C'est un chemin (v1,v2,...,vq) tel que:

- le même arc ne figure pas deux fois dans la séquence.
- les deux sommets aux extrémités du chemin coïncident.

♦Graphe connexe:

C'est un graphe pour lequel deux sommets quelconques sont reliés par un chemin.

♦Degré d'un sommet v:

C'est le nombre d'arcs ayant une extrémité en v.

Degré intérieur de v = nombre d'arcs dont l'extrémité terminale est v.

Degré extérieur de v = nombre d'arcs dont l'extrémité initiale est v.

♦Arbre:

C'est un graphe connexe sans cycles.

♦Arbre intérieur:

C'est un arbre dont le degré extérieur de chaque sommet est égal à 1.

♦Graphe complet:

Un graphe G(V,E) est complet si pour toute paire de sommets (i, j), il existe au moins un arc de la forme (i, j) ou (j, i).

♦Graphe partiel:

Soient un graphe G(V, E) et un ensemble E' $\subset$ E. Le graphe partiel engendré par E' est le graphe ayant le même ensemble V de sommets que G, et dont les arcs sont les arcs de E'.

♦Graphe biparti:

Un graphe est biparti si l'ensemble de ses sommets peut être partitionné en deux classes V1 et V2 de sorte qu'il n'existe pas d'arcs entre deux sommets de la même classe.

♦Graphe fonctionnel:

Un graphe G est dit fonctionnel si le degré extérieur de chaque sommet est égal à 1.

♦Matrice associée à un graphe:

La matrice associée d'un graphe G(V, E) est une matrice X à coefficients 0 ou 1, dont chaque ligne et chaque colonne correspondent à un sommet de G, telle que:

$X(i, j) = 1$ si $(i, j) \in E$,

0 sinon.

Dans le cas non orienté $X(i, j) = X(j, i)$ $\forall i, j \in V$, la matrice associée est alors symétrique.

♦2-Couplage :

Un 2-couplage dans un graphe G(V, E) est un graphe partiel de G dont le degré de chaque sommet est exactement 2. Ce graphe partiel n'est pas nécessairement connexe, il est composé d'un ensemble de cycles.

Le 2-couplage optimum associé à un graphe pondéré est le 2-couplage dont le poids total est minimal.

♦Cycle Hamiltonien:

C'est un cycle qui passe par chaque noeud exactement une fois.

1-2- La théorie de la NP-Complétude

1-2-1- Problème d'optimisation:

Les problèmes d'optimisation se divisent en deux catégories: ceux à variables continues, et ceux à variables discrètes appelés combinatoires.

En optimisation continue, la solution est généralement cherchée dans un ensemble de nombres réels tandis que en optimisation combinatoire, la solution est cherchée dans un ensemble fini (un entier, une permutation, un graphe ...).

1-2-2- Problème de décision:

Un problème de décision comporte une entrée suivie d'une question. L'entrée précise une instance particulière du problème et la question n'admet que deux modalités de réponse, oui et non.

Tout problème d'optimisation combinatoire peut être transformé en un problème de décision.

1-2-3- Complétude:

Un algorithme est une représentation finie d'une méthode de calcul permettant de résoudre un problème.

La première tâche qui s'impose à celui qui tente de résoudre un problème à l'aide d'un algorithme est de savoir si ce problème est résoluble. Puis le cas échéant, si cette résolution est facile ou difficile à mettre en oeuvre.

Dans le cas de difficulté, il sera inutile de chercher un algorithme « exact », car, dans le pire des cas, on risque de n'être plus de ce monde avant que le programme ait pu être exécuté.

La théorie de la complexité permet de classer des problèmes selon la difficulté qu'il y a les résoudre.

1)- La classe P et NP :

La classe P est composée des problèmes de décisions pour lesquels il existe un algorithme dont le temps nécessaire à la résolution est borné par une fonction polynomiale de la taille du problème.

La classe NP est quant à elle constituée de tous les problèmes dont l'exactitude de la réponse (oui ou non) à leur question, peut être vérifiée en temps polynomial, même s'il n'existe pas d'algorithme polynomial pour leur résolution.

La relation entre la classe P et NP est fondamentale. La première observation faite est que $P \subseteq NP$, mais il existe plusieurs raisons pour croire que cette inclusion est propre, c'est à dire que $P \neq NP$.

Sous cette hypothèse la distinction entre P et (NP-P) devient très importante.

Pour prouver qu'un problème appartient à la classe P, il suffit de trouver un algorithme qui le résout en un temps polynomial.

Pour prouver qu'un problème appartient à la classe P, il suffit de trouver un algorithme qui le résout en un temps polynomial. Tandis que pour prouver qu'un problème fait partie de la classe (NP-P), il faut montrer qu'il n'existe pas d'algorithme polynomial pour le résoudre. Ce qui est plus difficile à établir. On utilise pour cela la réduction polynomial.

2)- Réduction polynomiale et NP-Complétude :

On dit que le problème (P1) est réduit polynomialement au problème (P2) (on note P1 $\propto$ P2) si à partir de n'importe quelle instance de (P1), on peut construire une instance de (P2), alors on peut résoudre (P1), ce qui entraîne que P2 est plus « difficile » que P1.

Deux problèmes P1 et P2 sont dits polynomialement équivalents si P1 $\propto$ P2 et P2 $\propto$ P1.

La réduction polynomiale forme donc des classes d'équivalence dans l'ensemble NP, reliées par une relation d'ordre.

La plus « faible » classe est la classe P qui peut être vue comme étant celle des problèmes « faciles ».

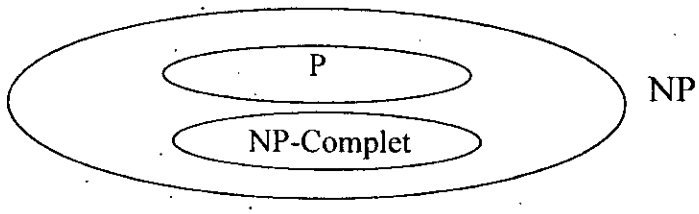
La classe des problèmes NP-complets va former une autre classe d'équivalence, celle des problèmes dits « difficiles ».

Ainsi on dit qu'un problème de décision A est NP-complet si :

(a) $A \in NP$.

(b) Tout problème de NP peut être réduit polynomialement en A.

Sous l'hypothèse que $P \neq NP$, il a été prouvé qu'il existe des problèmes dans NP qui ne sont ni P ni NP-complets d'où la représentation de la classe NP.



Le plus grand intérêt est porté à la classe des problèmes NP-complets.

Pour prouver qu'un problème P1 $\in$ NP, est NP-complet, il faut montrer qu'un certain problème P2 qu'on sait NP-complet se réduit polynomialement en P1, (une fois qu'au moins un problème est connu NP-complet).

En effet, si P2 $\propto$ P1 et P2 est NP-complet alors P1 est NP-complet, car si P1 est polynomial, avec P2 $\propto$ P1, ceci entrainerait que P2 est également polynomial, ce qui contredit le fait que P2 est NP-complet.

Le premier problème prouvé NP-complet a été le problème de satisfiabilité [Cook, 1971] à partir duquel d'autres problèmes ont été reconnus NP-complets.

En conclusion, pour les problèmes polynomiaux, il est aisé de trouver une solution optimale, alors que cela parait impossible pour les problèmes NP-complets du fait du temps que la résolution prendrait.

Une fois qu'un problème a été classé NP-complet, on est sûr qu'on ne peut pas trouver une solution exacte en un temps polynomial. Il ne nous reste donc qu'à trouver une solution approchée de la solution optimale en un temps polynomial.

Afin de vérifier que cette solution est suffisamment proche de l'optimum, on utilisera le rapport d'approximation.

3)- Rapport d'approximation:

Soit un problème d'optimisation combinatoire (P), on note OPT(I) la valeur optimale associée à une instance I du problème (P), ou une borne pour celle-ci.

Soit A un algorithme approché pour résoudre (P) et soit A(I) la solution obtenue par A pour l'instance I ou une borne pour cette solution.

On définit les rapports d'approximation suivants:

$\rightarrow |A(I) - OPT(I)|$: représente l'écart absolu de la solution approchée par rapport à la solution optimale.

→ $\left|\frac{A(I)-OPT(I)}{OPT(I)}\right|$: représente l'erreur relative de la solution approchée par rapport à la solution optimale.

→ $\frac{A(I)}{OPT(I)}$: représente l'écart relatif de la solution approché par rapport à la solution optimale. Il est inférieur à 1 pour un problème de maximisation et supérieur à un pour un problème de minimisation.

Dans notre étude, nous nous intéresserons uniquement au dernier rapport que l'on veut dans tous les cas le plus proche possible de 1.

2-Présentation du problème du voyageur de commerce.

2-1 Définition du problème

Le problème du voyageur de commerce se définit par la donnée de n villes et des distances qui les séparent.

Un voyageur de commerce démarre d'une ville pour effectuer une tournée, et donc visiter chacune des autres villes exactement une fois, puis revenir à son point de départ. Il doit sélectionner l'ordre dans lequel il effectuera sa tournée afin de minimiser la distance totale parcourue.

Ce voyageur doit choisir le tour qui optimisera sa traversée parmi les (n-1)! tours possibles, il est donc confronté à un problème d'optimisation combinatoire qui se formule ainsi:

Instance : . n villes.

. $d_{i,j}$ la distance qui sépare la ville i de la ville j, pour chaque i et j.

Question: .quelle est la permutation P : {1,2,....,n} → {1,2,...,n} qui minimise la distance $D = \sum_{i=1}^{n-1} d_{P(i),\,P(i+1)} + d_{P(n),\,P(1)}$.

Comme tout problème d'optimisation combinatoire, le problème du voyageur de commerce peut se formuler sous forme d'un problème de décision de la façon suivante:

nstance: .n villes,

.$d_{i,j}$ pour chaque couple de villes (i, j).

.un entier D.

Question: .existe-t-il une permutation P : {1,2,....,n} → {1,2,....,n}, telle que:

$$\sum_{i=1}^{n-1} d_{P(i),\,P(i+1)} + d_{P(n),\,P(1)} \leq D.$$

2-2 Quelques résultats sur le problème du voyageur de commerce

◆ Le problème du voyageur de commerce est classé comme étant NP-Complet. En effet, le problème du cycle Hamiltonien, qui est connu NP-Complet, se réduit polynomialement au problème du voyageur de commerce.[KARP].

Preuve:

L'instance du problème du cycle Hamiltonien est un graphe G(V,E), sa question est de savoir s'il existe un cycle qui passe par chaque noeud exactement une fois.

Pour obtenir une instance du problème du voyageur de commerce, il suffit de transformer le graphe G en un graphe complet G'(V,E') en affectant la valeur 1 aux arêtes qui existaient déjà dans G et la valeur 2 aux arêtes ajoutées (les arêtes de E-E').

La question qui correspond à cette nouvelle instance est de savoir s'il existe un tour dont la distance totale est inférieure ou égale à n où n est le nombre de sommets du graphe.

Une réponse positive signifie que le tour est constitué uniquement d'arêtes de valeur 1, ce qui veut dire que la réponse est positive également pour le problème du cycle Hamiltonien.

Une réponse négative signifie que pour construire un tour dans G', il faut utiliser au moins une arête de valeur 2. donc on ne peut pas trouver de cycle Hamiltonien dans le graphe G puisque celui-ci ne contient pas les arêtes dont la longueur est 2.

La construction du graphe G' à partir de G se fait en un temps polynomial, d'où la réduction polynomiale entre le problème du cycle Hamiltonien et celui du voyageur de commerce.

◆ Tant que P $\neq$ NP, il n'existe pas d'algorithme approché pour le problème du voyageur de commerce avec un rapport d'approximation constant.[SAHNI,GONZALES,1976].

Preuve:

Soient A un algorithme approché pour le problème du voyageur de commerce dont le rapport d'approximation r est constant, et G(V,E) une instance du problème du cycle Hamiltonien tel |V| = n.

Construire une instance I pour le problème du voyageur de commerce de la façon suivante:

$d_{i,j} = 1$ si l'arête $(i, j) \in E$ et $d_{i,j} = r \times n$ si l'arête $(i, j) \notin E$, puis appliquer l'algorithme A à cette instance:

- Si G contient un cycle Hamiltonien, alors il existe un tour de distance totale égale à n, d'où $OPT(I) = n$. Sachant que $A(I) / OPT(I) \leq r$, on a $A(I) \leq r \times n$.
- Si G ne contient pas de cycle Hamiltonien, alors le tour construit par l'algorithme A utilisera au moins une arête de valeur $r \times n$, donc $A(I) \geq (n-1) + r \times n$, et $A(I) \geq r \times n$.

Finalement, il suffit d'appliquer l'algorithme approché A à l'instance transformée I de n'importe quel problème de cycle Hamiltonien et de comparer $A(I)$ à $r \times n$, pour savoir si la réponse à la question de ce problème est positive ou négative (selon que $A(I) \leq r \times n$ ou $A(I) \geq r \times n$).

Ainsi, l'algorithme approché A peut être utilisé pour résoudre polynomialement le problème du cycle Hamiltonien qui est prouvé NP-Complet. Ce qui contredit l'hypothèse que $P \neq NP$.

♦ Dans le cas particulier du problème du voyageur de commerce où :

. la matrice des distances est symétrique : $dij = dji$, $\forall$ i, j

. l'inégalité triangulaire est satisfaite $dij + djk \geq dik$, $\forall$ i, j,k ,

des solutions proches de l'optimum peuvent être trouvées en un temps relativement réduit.

En effet, Christophides a établi un algorithme approché polynomial, pour ce cas, qui garantit un rapport d'approximation de 3/2.[CHRISTOFIDES,1976].

Il est à noter que les instances du problème du voyageur de commerce construites lors de la preuve précédente, ne satisfont pas l'inégalité triangulaire. Le résultat précédent n'est donc pas vrai dans ce cas spécial.

♦ Pour le problème du voyageur de commerce qui vérifie les conditions suivantes:

(a) la matrice des distances est symétrique,

(b) les distances entre sommets sont 1 et 2,

(c) l'inégalité triangulaire est vérifiée,

Papadimitriou et Yannakakis proposent deux algorithmes approchés dont les rapports d'approximation sont respectivement 4/3 et 11/9, puis ils améliorent le second pour obtenir un rapport de 7/9.[PAPADIMITRIOU et YANNAKAKIS,"The traveling salesman problem with distances one and two",1993].

Notre travail consiste à étudier ces deux algorithmes et de les programmer afin de tester leurs rapports d'approximation.

3 - Les algorithmes approchés associés au problème du voyageur de commerce avec des distances 1 et 2

Cette partie est consacrée à l'étude des deux algorithmes proposés par les auteurs et de donner la démonstration théorique de leurs rapports d'approximation.

3-1- Un algorithme approché avec un rapport de 4/3 (Algorithme 1)

3-1-1- Enoncé de l'algorithme:

Cet algorithme se base sur la technique d'unification des sous-tours. Il procède comme suit :

Etape 1 :

Chercher un 2-couplage optimum. Ce 2-couplage est constitué d'un certain nombre de cycles, mais il peut arriver qu'il ne contienne qu'un seul cycle, ce qui nous amène directement à la solution approchée.

Etape 2 :

Relier les différents cycles obtenus dans le 2-couplage, en procédant ainsi:

Choisir deux cycles au hasard et enlever une arête sur chacun d'eux.

Remplacer les deux arêtes ôtées par deux autres qui lient les deux cycles pour n'en faire qu'un seul.

Recommencer une nouvelle liaison mais cette fois entre le cycle qu'on vient d'obtenir et un nouveau .

Itérer jusqu'à l'obtention d'un unique tour qui représentera la solution approchée.

Durant cette étape, veiller à chaque fois à éliminer le plus possible d'arêtes de valeur 2 afin d'obtenir une bonne solution approchée.

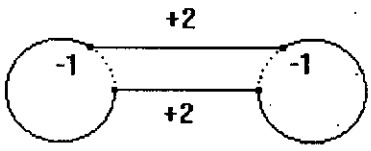
3-1-2- Preuve du rapport d'approximation:

La valeur de la solution approchée est celle du 2-couplage optimum augmentée du coût des liaisons. Il faut donc évaluer le coût des liaisons.

Coût unitaire maximal d'une liaison:

Afin de calculer le coût maximal, on considérera dans tout ce qui suit le pire des cas.

•Au départ, les deux premiers cycles sont pris au hasard. Au pire, deux arêtes de longueur 1 sont remplacées par deux arêtes de longueur 2, ce qui entraîne un coût maximal de 2.



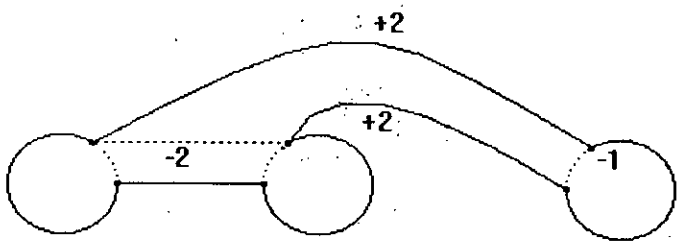
coût max = 2 +2 -1 -1 = 2.

•A partir du moment où une arête de valeur 2 est présente, le coût maximal d'une liaison devient 1:

Soient cycle1 le cycle qui vient d'être construit et cycle2 le nouveau cycle qu'on veut relier à cycle1. Deux cas de figure peuvent se présenter:

(a) le cycle1 contient au moins une arête de valeur 2.

Afin de relier cycle1 et cycle2, éliminer cette arête de cycle1 ainsi qu'une arête de cycle2 qui vaudra au moins 1, puis les remplacer par deux arêtes qui lient les deux cycles et dont la valeur est au pire égale à 2.



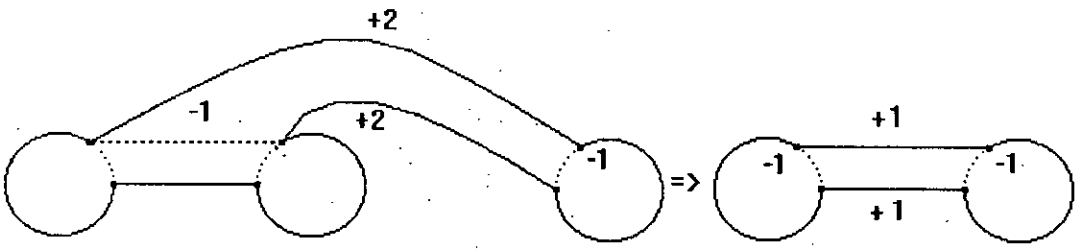
cycle 1 cycle 2

(b) le cycle1 ne contient pas une arête de valeur 2.

La liaison coûtera au pire 2 car on sera obligé d'enlever une arête de valeur 1 de cycle1. On éliminera également une arête de cycle2 qui vaudra au moins1, puis on relira les deux cycles avec deux arêtes de valeur 2, au pire.

Mais si un tel cas se présente, supposons à la k-ème itération, alors aucune arête de valeur 2 n'a été ajoutée lors de la (k-1)-ème itération (sinon on l'aurait choisie afin de l'éliminer à la k-ème itération).

Donc lors de la (k-1)-ème liaison, deux arêtes de valeur au moins 1 ont été remplacées par deux arêtes de valeur 1, ce qui entraîne un coût maximal nul.



k-ème itération.

coût max = + 2 + 2 -1 - 1 = 2.

(k-1)-ème itération.

coût max = + 1 + 1 - 1 - 1 = 0.

Finalement, si le coût d'une liaison est égal à 2, alors soit elle est la première, soit elle est précédée par une liaison dont le coût maximal est nul.

En conclusion, le coût unitaire maximal d'une liaison vaut 2 pour la première liaison et 1 pour toutes les autres.

Le rapport d'approximation:

Le 2-couplage initial contient au maximum n/3 cycles, puisque chaque cycle est composé d'au moins trois arêtes. Il faudra donc effectuer (n/3 - 1) liaisons au maximum.

Le coût unitaire des liaisons est au plus de 2 pour la première itération, puis 1 pour toutes les autres, qui sont au plus au nombre de (n / 3 - 2), d'où :un coût total maximal de n / 3 car 2 + 1 x (n /3 - 2) = n /3, donc (coût total des liaisons) $\leq$ n / 3.

Soit 2CP le coût du 2-couplage optimum et A(I) le coût de la solution approchée.

A(I) = 2CP + (coût total des liaisons), d'où:

$$A(I) \leq 2CP + (n / 3).....(1)$$

Le tour optimal est un 2-couplage où l'on n'obtient qu'un seul cycle, c'est donc une solution particulière du 2-couplage.

La valeur de ce 2-couplage est supérieure ou égale à celle du 2-couplage optimum, donc le coût de la tournée optimale est supérieur au coût du 2-couplage optimum, d'où:

$$OPT(I) \geq 2CP.....(2)$$

D'après (1) et (2), $A(I) \leq OPT(I) + (n / 3)$

Mais $OPT(I) \geq n$, (le nombre de villes étant n), d'où : $A(I) \leq OPT(I) + (OPT(I) / 3)$.

Finalement $\frac{A(I)}{OPT(I)} \leq \frac{4}{3}$.

3-2 Un algorithme approché avec un rapport de 11/9 (Algorithme 2)

L'idée de cet algorithme consiste à relier les cycles dans un ordre spécifié par la solution d'un problème de couplage.

3-2-1- Enoncé de l'algorithme:

Soit G(V,E) le graphe de l'instance d'un problème du voyageur de commerce.

L'algorithme procède comme suit :

Etape 1 :

• Trouver un 2-couplage optimum.

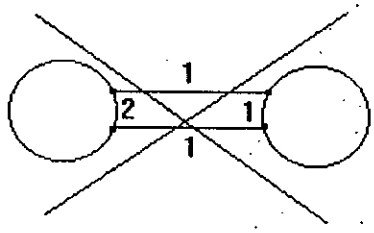
La solution de ce 2-couplage est sous forme d'un ensemble C de cycles qui ne sont pas nécessairement connectés. En général $|C| > 1$, sinon le problème est résolu.

Certains cycles ne contiennent que des arêtes de valeur 1, ils seront appelés cycles purs. Un cycle est impur s'il contient au moins une arête de valeur 2.

• Transformer le 2-couplage obtenu de façon à réaliser les deux conditions suivantes:

Condition 1 : Il n'y a qu'un seul cycle contenant des arêtes de valeur 2, tous les autres cycles sont purs.

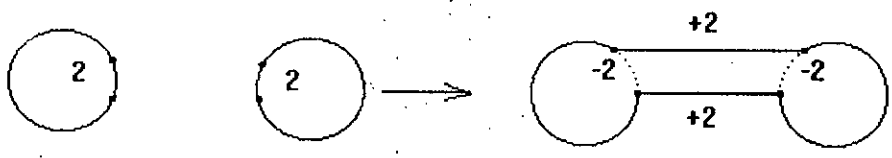
Condition 2 : Il n'y a pas d'arête de valeur 1 reliant une arête de valeur 2 du cycle impur et une arête des cycles purs.



cycle impur. cycle pur.

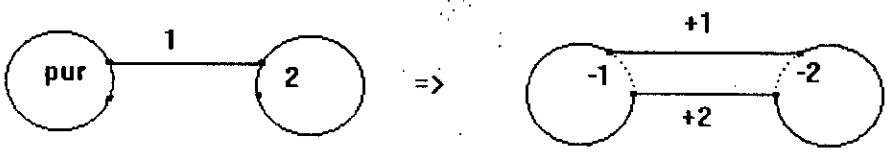
Si une des deux conditions est violée, les deux cycles en question sont fusionnés sans coût supplémentaire de la manière suivante:

- Si la condition 1 n'est pas vérifiée, c'est-à-dire qu'il existe deux cycles impurs, alors les fusionner en enlevant de chacun une arête de valeur 2 et en les reliant par deux autres arêtes qui auront au pire, la valeur 2.



Donc le coût maximal d'une telle fusion est : -2 -2 +2 +2 = 0

-Si la condition 2 n'est pas vérifiée, c'est-à-dire qu'il y a un cycle impur dont une arête de valeur 2 est reliée à une arête d'un cycle pur par une arête de valeur 1, alors fusionner ces deux cycles en enlevant une arête du cycle pur (de valeur 1) et l'arête de valeur 2 du cycle impur, puis les relier par deux arêtes dont l'une est celle de valeur 1 qui est déjà présente, la seconde sera de valeur 2 au pire.



D'où le coût maximal de cette fusion : -1 -2 +1 +2 = 0.

Etape 2 :

Soit C le nouvel ensemble de cycles (ceux vérifiant les deux conditions), et soit Cp l'ensemble des cycles purs uniquement.

Former un graphe biparti B avec d'un côté l'ensemble Cp, et de l'autre côté l'ensemble de tous les sommets V du graphe G en créant une arête entre une composante c de Cp et un sommet v de V si:

- v n'appartient pas à c.
- il existe une arête [u, v] de valeur 1 entre un sommet u du cycle c, et le sommet v.

Etape 3 :

Chercher un couplage maximum dans le graphe biparti B, c'est-à-dire coupler le plus possible de cycles de Cp à des sommets de V.

Etape 4 :

Construire un graphe orienté F(C,A) dont l'ensemble des sommets est celui des cycles obtenus par le 2-couplage et l'ensemble des arêtes A est tel que: (c, c') $\in$ A si c est couplé à un noeud de c' dans le graphe biparti B.

F est un graphe fonctionnel partiel: en effet, chaque sommet de F est un cycle de C, et chaque cycle est couplé à un sommet au maximum. D'où le degré extérieur de tous les sommets est au plus 1, (le degré extérieur du cycle impur est nul).

Etape 5 :

Pour pouvoir utiliser cette partie de l'algorithme, il faut d'abord démontrer ce lemme sur les graphes fonctionnels.

Lemme :

Tout graphe fonctionnel admet un sous-graphe couvrant composé:

- d'arbres entrants de profondeur 1 (c'est à dire un fils et plusieurs pères),
- de chemins de longueur 2.

Preuve :

Remarque: ne pas perdre de vue que les sommets de F sont des cycles.

F est constitué de composantes faiblement connexes qui sont des cycles avec certains arbres convergeant vers eux. En effet, puisque le degré extérieur de chaque sommet d'un graphe fonctionnel est exactement 1, les cas de figures suivants sont impossibles:

- deux arbres qui convergent vers un même sommet,
- deux cycles connectés,
- une composante faiblement connexe qui ne contient pas de cycle.

Pour décomposer F, considérer les composantes faiblement connexes une à une.

Commencer par choisir le noeud le plus éloigné du cycle qui constitue la racine de la composante traitée, soit l ce noeud. Si le successeur de l n'appartient pas au cycle, alors définir un arbre par s et tous ces prédécesseurs.

Enlever cet arbre et réitérer ce processus jusqu'à ce qu'il ne reste plus que le cycle et certains prédécesseurs immédiats de ses noeuds.

Pour les noeuds du cycle qui ont des prédécesseurs hors du cycle, on a le choix d'inclure ou non leur prédécesseur qui se trouve dans le cycle afin de former un arbre entrant.

Dans ce cas, choisir la solution qui lasse un nombre pair (même zéro) de noeuds entre ce noeud et le précédent noeud du cycle qui ce trouve dans son cas(ie: qui a un prédécesseur hors du cycle). Les chemins de longueur paire ainsi obtenus peuvent être décomposés en chemins de longueur 1 (qui sont également des arbres de profondeur 1).

Dans le cas où il n'y a pas de noeuds hors du cycle, décomposer celui-ci en chemins de longueur 1 et un seul chemin de longueur 2.

Décomposer le graphe F suivant ce lemme afin d'obtenir un graphe F' fait d'arbres de profondeur 1 et de chemins de longueur 2.

Etant donné que F est un graphe fonctionnel partiel du fait que certains sommets ont un degré nul, le graphe F' contiendra des composantes isolées qui peuvent être les cycles non couplés dans le graphe biparti ou le cycle impur (si leur degré intérieur est nul, c'est-à-dire qu'aucun cycle n'est relié à eux dans le graphe F).

Etape 6 :

- Afin de revenir au graphe initial, transformer le sous graphe couvrant F' en un graphe non orienté. Il faut donc associer à chaque arête orientée de F' une arête non orientée qui ira d'un sommet à un autre et non pas d'un cycle à un autre.

Une arête orientée entre un cycle c et un cycle c' sera remplacée par une arête non orientée entre un sommet v de c et un sommet v' de c'. On est sûr que l'arête (v,v') existe car c'est à partir d'elle que le cycle c a été couplé au sommet v' de c' dans le couplage biparti et que l'arête orientée (c,c') a été construite dans le graphe F.

Remarques:

1- Dans les arbres de profondeur 1, les cycles pères sont reliés au cycle fils à travers des sommets différents. En effet, supposons que deux cycles pères soient reliés au même sommet du cycle fils, cela veut dire que ces deux cycles ont été associés à un même sommet lors du couplage biparti, ce qui contredit le principe d'un couplage.

2- Par contre dans les chemins de longueur 2, le premier et dernier cycles peuvent être reliés au cycle du milieu à travers le même sommet. Ce sommet sera l'extrémité terminale de l'arc sortant du le premier cycle et l'extrémité initiale de l'arc entrant dans le dernier cycle.

3- On remarque ici l'intérêt du couplage biparti: il permet de relier les composantes de cycles obtenues à travers des arêtes de valeur 1.

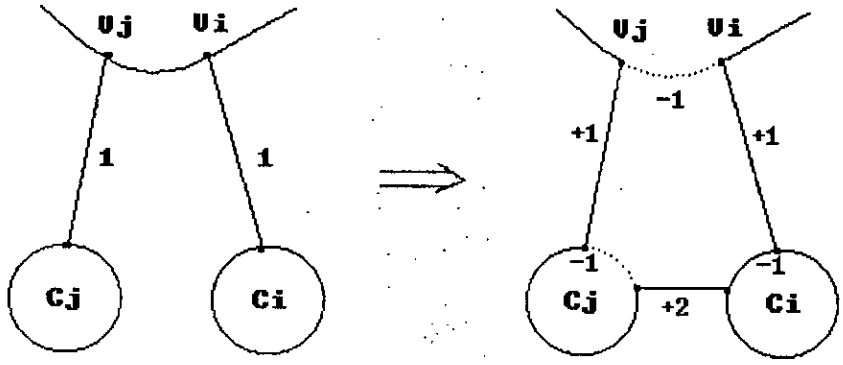
- Considérer maintenant les composantes du sous graphe couvrant F'.

1)-Les arbres de profondeur 1 :

Ils sont composés (après le passage aux arêtes non orientées) d'un cycle c (la racine) relié à m autres cycles (feuilles) $c_1, c_2, ..., c_m$ .Chaque cycle c_i est relié à c à travers arête de valeur 1 qui joint un noeud de c_i à un noeud v_i de c, pour $i = 1, 2, ..., m$, où les v_i sont distincts (d'après la remarque 1).

Lier le cycle racine aux cycles feuilles de la façon suivante:

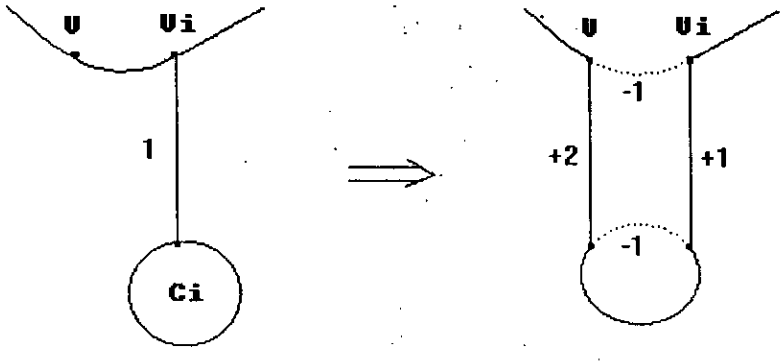
- Parcourir cycle racine c dans le sens des aiguilles d'une montre, en choisissant comme point de départ un noeud qui ne soit pas un vi (ie: non relié à une feuille). Si un tel noeud n'existe pas, prendre un noeud arbitraire.
- Si deux noeuds adjacents vi et vj reliés à des feuilles sont rencontrés, alors relier les cycles correspondants (les feuilles) ci et cj au cycle racine en procédant ainsi:



fusion (a)

coût max = (-1 -1 -1) + (1 +1 +2) = 1

- Si le noeud v qui succède un noeud vi n'est pas relié à une feuille ou si v_i est le dernier noeud de c pas encore examiné, alors fusionner le cycle feuille c_i et le cycle racine c de cette façon :



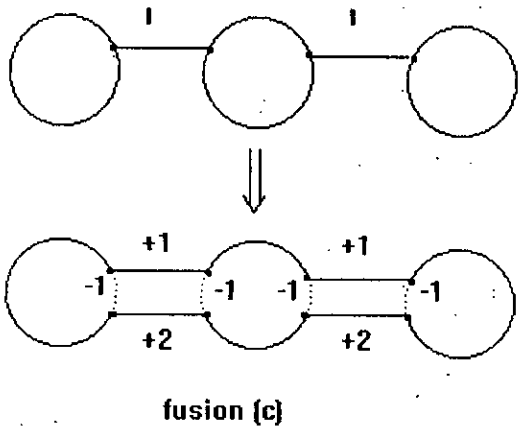
fusion (b)

coût max = (-1 -1) + (2 +1) = 1

2) - Les chemins de longueur 2 :

Ils correspondent à trois cycles purs tels qu'il existe entre le cycle du milieu et les deux autres des arêtes de valeur 1

Les fusionner de la manière suivante :



fusion (c)

coût max = (-1 -1 -1 -1) + (1 +1 +2 +2) = 2

A partir de maintenant, tous les cycles formés contiennent des arêtes de valeur 2. Si les arêtes ajoutées précédemment lors des fusions (a), (b) et (c) sont de valeur 1, on augmente leur valeur à 2, ce qui ne risque pas de diminuer la valeur du tour construit.

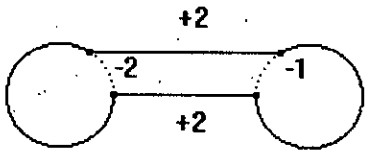
Les nouveaux cycles peuvent être donc reliés entre eux à un coût supplÈmentaire nul. Pour cela, Il suffit d'éliminer les arêtes de valeur 2 provenant des fusions précédentes qui seront remplacées par d'autres arêtes dont la longueur sera au pire égale à 2, d'où:

coût maximum = +2+2-2-2 = 0.

3) - Les noeuds isolés :

Commencer par fusionner les cycles purs isolés.

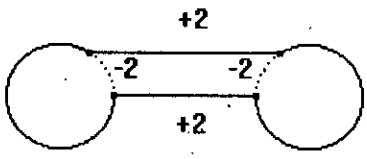
Etant donné que le cycle obtenu par la fusion de toutes les composantes non isolées contient des arêtes de valeur 2, la liaison se fera à un coût maximum de 1:



fusion des composantes non isolées.

coût max = (-2 -1) + (2 + 2) = 1.

Finalement, fusionner le cycle impur s'il est isolé à un coût supplémentaire nul car il contient des arêtes de valeur 2 ainsi que le cycle qui vient d'être obtenu, ce qui permet d'enlever deux arêtes de valeur 2 pour les remplacer par deux arêtes de valeur 2 au pire



fusion de toutes les composantes sauf le cycle impur.

coût max = + 2 + 2 - 2 - 2 = 0.

3-2-2- Preuve du rapport d'approximation

Voici maintenant la preuve que le rapport d'approximation de cet algorithme est au pire 11/9. La démonstration se fait en deux parties :

Cas où le graphe G(V, E) contient un cycle Hamiltonien :

Le coût optimal est dans ce cas égal à n. OPT(I) = n, donc le coût du 2-couplage optimal 2CP vaut n également.

Soit C l'ensemble des cycles obtenus.

Lemme : Il existe un couplage dans le graphe biparti formé B qui couvre tous les cycles de C.

Preuve du lemme :

Soit $(v_1, v_2 ..., v_n)$ le cycle Hamiltonien de G.

Dans le couplage du graphe biparti B, choisir d'associer chaque cycle c à l'arête de valeur 1 qui le relie au premier sommet v_i du cycle Hamiltonien, qui n'appartient pas à c, tel que le sommet qui le précède v_{i-1} appartient à c.

- L'arête (v_i , v_{i-1}) telle que $v_i \notin c$ et $v_{i-1} \in c$ existe dans E puisqu'elle appartient au cycle Hamiltonien.
- Un sommet v_i ne peut pas être choisi pour deux cycles c_1 et c_2 sinon cela entraînerait que $v_{i-1} \in c_1$ et $v_{i-1} \in c_2$, ce qui n'est pas possible puisqu'une arête ne peut appartenir qu'à un seul cycle.
- Tous les cycles sont couverts puisqu'une telle arête existe pour chaque cycle.

Le 2-couplage optimum ne contient que des cycles purs, donc le graphe biparti se fait entre l'ensemble des cycles C tout entier et l'ensemble V.

Le couplage du graphe B étant parfait, le graphe orienté F ne contiendra pas de composantes isolés.

Donc une fois le graphe partiel F' construit, on n'aura à effectuer que des liaisons du type : fusion(a), fusion(b) et fusion(c).

Proposition:

Chaque liaison coûte au maximum 2/9 par noeud impliqué.

Preuve:

Commencer par calculer le coût de chaque type de liaison:

- Fusion (c) : dans la liaison d'un chemin de longueur 2, neuf noeuds au minimum sont impliqués, à savoir trois par cycle. Le coût maximal par fusion étant 2, le coût de liaison par noeud est 2/9.

- Fusion (b): impliquer les noeuds du cycle feuille ainsi que le v_i correspondant, et son successeur v du cycle racine. Donc cinq noeuds au minimum sont impliqués d'où un coût maximal par noeud de 1/5.

- Fusion (a) : pour les arbres de profondeur 1, les sommets du cycle racine ne seront pas impliqués car certains d'entre eux ont déjà été impliqués dans la fusion (b). Le cycle racine étant relié à deux autres cycles au minimum, le nombre de noeuds impliqués sera au minimum 6 (trois noeuds par cycle), d'où un coût maximum par liaison de 1/6.

Il est clair qu'aucun noeud du graphe n'est chargé dans deux fusions différentes. Le coût maximum de liaison par noeud pour tout le graphe est 2/9, (2/9 > 1/5 > 1/6).

Le coût maximum des fusions = (coût maximum par noeud) x (nombre des noeuds).

= 2/9 x n.

Le coût du tour produit = (coût du 2-couplage optimum) + (coût total des liaisons).

≤ (coût du 2-couplage optimum) + (coût maximum des liaisons).

donc $A(I) \leq 2CP + 2/9 \times n$, soit $A(I) \leq OPT(I) + 2/9 \times OPT(I)$,

d'où $\frac{A(I)}{OPT(I)} \leq \frac{11}{9}$.

Cas où le graphe G(V, E) ne contient pas de cycle Hamiltonien :

Le coût optimal est dans ce cas supérieur à n: $OPT(I) \geq n$.

Cette fois-ci le couplage du graphe biparti B peut ne pas être parfait, il peut donc exister des cycles purs isolés. De ce fait, le graphe orienté contient certains noeuds dont le degré extérieur est nul (les cycles purs qui n'ont pas été couplés dans le graphe biparti et le cycle impur).

Supposons que le 2-couplage optimum 2CP contient k arêtes de valeur 2. Il a donc un coût de $2 \times k + 1 \times (n - k) = n + k$.

Etant donné que $OPT(I) \geq 2CP$, le tour optimal a un coût supérieur ou égal à n+k:

$OPT(I) \geq n + k$.

Soient un tour optimal $(v_1, v_2, ..., v_n, v_1)$ et U l'ensemble des sommets v_i tels que l'arête $[v_i, v_{i+1}]$ a la valeur 2 dans le tour optimal.

Soit c_2 le nombre de cycles purs contenant un noeud de U.

- Dans le tour optimal, il n'y a pas de sommet appartenant à deux cycles différents. S'il y a c_2 cycles purs contenant un noeud de U, alors il y a au moins c_2 arêtes de valeur 2 dans le tour car il y a aussi des arêtes de valeur 2 provenant du cycle impur. D'où le coût minimal du tour optimal est $n + c_2$. ie $OPT(I) \geq n + c_2$.

- Soit r_2 le nombre de cycles purs isolés dans F' le graphe partiel de F. Soit n_2 le nombre des sommets des r_2 cycles.

Un cycle pur isolé dans F', est un cycle qui n'a pas été couplé dans le graphe biparti B,d'où $r_2 \leq$ nombre de cycles non couplés.

A partir du tour optimal, on construit un couplage dans B de la même façon que dans la preuve du lemme précédent ie en reliant un cycle c à un sommet v_{i+1} du tour optimal tel que $v_{i+1} \notin c$ et son prédécesseur vi $\in$ c avec l'arête (v_i, v_{i+1}) de valeur 1. Etant donné qu'il y a c2 cycles purs contenant un sommet de l'ensemble U, alors il y aura, au plus, c_2 cycles qui ne seront pas couplés par cette méthode. Ainsi est obtenu, à partir du tour optimal, un couplage dans B qui contient au maximum c_2 cycles non couplés, d'où $r_2 \leq c_2$.

- Le coût de la fusion des cycles des composantes non isolées de F' est 2/9 par noeud impliqué comme pour le cas où il existe un cycle Hamiltonien dans le graphe car rien n'a changé quant à la manière de relier les cycles. Par contre, ce qui a changé est le nombre total de nœuds impliqués dans les liaisons.

- Le nombre maximal de noeuds impliqués est $(n- n_2 -k)$ car:

..le nombre des sommets des composantes non isolées de F' est $n - n_2$.

..si le cycle impur est dans une composante non isolée de F', alors il est la racine d'une telle composante puisqu'il n'a pas été couplé, étant donné qu'il n'est pas inséré dans le graphe B.

..chaque noeud impliqué du cycle impur est précédé par une arête de valeur 1 (d'après la condition 2).

..puisque k est le nombre de toutes les arêtes de valeur 2 dans le cycle impur, au moins k arêtes ne sont pas impliquées, d'où le nombre maximal de noeuds impliqués est égal à $n - n_2 - k$..

- Soit A(I) le coût du tour construit par l'algorithme.

A(I) $\leq$ 2CP + coût de fusion des cycles des composantes non isolées de F'.

+ coût de fusion des cycles des composantes isolées de F'.

$A(I) \leq (n+k) + 2/9\,(n - n_2 - k) + r_2$

$\leq 11/9\,n + 7/9\,k - 2/9\,n_2 + r_2$

mais $r_2 \leq n_2/3$, chaque cycle contient au minimum trois sommets.

alors $A(I) \leq 11/9\,n + 7/9\,k - 2/3\,r_2 + r_2$, étant donné que $r_2 \leq c_2$,

$A(I) \leq 11/9\,n + 7/9\,k + 1/3\,c_2$

- Si $c_2 > k$ ie $n + k < n + c_2$ alors

$A(I) \leq 11/9\,n + 7/9\,c_2 + 1/3\,c_2$

$\leq 11/9\,n + 10/9\,c_2$

$\leq 11/9\,(n + c_2)$

- Si $c_2 < k$ ie $n + k > n + c_2$ alors

$A(I) \leq 11/9\,n + 7/9\,k + 1/3\,k$

$\leq 11/9\,(n + k)$

d'où $A(I) \leq 11/9 \times \max\{ n + c_2 , n + k \}$

$OPT(I)$ = coût du tour optimal $\geq \max\{ n + c_2 , n + k \}$

car $OPT(I) \geq 2CP = n + k$

et $OPT(I) \geq n + c_2$.

Finalement $A(I) \leq 11/9.\ OPT(I)$. d'où: $\dfrac{A(I)}{OPT(I)} \leq \dfrac{11}{9}$

Un tour pour le problème du voyageur de commerce avec des distances 1 et 2, meilleur que 11/9 fois le tour optimal, peut être obtenu en un temps polynomial

Papadimitriou et Yannakakis proposent également d'améliorer cette borne à 7/6 en utilisant un résultat de Harvingen qui a développé un algorithme polynomial permettant de trouver un 2-couplage optimum qui ne contient pas de triangles (cycles formés de trois sommets).
En effet les coûts des liaisons par noeud seront modifiés de la façon suivante:

- fusion (a) : huit sommets sont impliqués au lieu de six, d'où un coût de liaison de 1/8 par noeud.

- fusion (b) : six noeuds sont impliqués au lieu de cinq, d'où un coût de liaison par noeud de 1/6.

- fusion (c) : douze noeuds impliqués au lieu de neuf, donc le coût de liaison par noeud devient 2/12 soit 1/6.

Finalement, le coût de liaison maximum par noeud est 1/6.

Dans le cas où le graphe G contient un cycle Hamiltonien, on obtient:

$A(I) \leq 2CP + 1/6\ OPT(I)$, soit $A(I) \leq OPT(I) + 1/6\ OPT(I)$ et enfin $\frac{A(I)}{OPT(I)} \leq \frac{7}{6}$.

Dans le cas où le graphe G ne contient pas de cycle Hamiltonien,

$A(I) \leq (n + k) + 1/6\ (n - n_2 - k) + r_2$, donc $A(I) \leq 7/6\ n + 5/6\ k - 1/6\ n_2 + r_2$.

mais $r_2 \leq n_2/4$ et $r_2 \leq c_2$, d'où $A(I) \leq 7/6\ n + 5/6\ k + 1/3\ c_2$ soit $A(I) \leq \max\{(n+k), (n+c)\}$.

et finalement $\frac{A(I)}{OPT(I)} \leq \frac{7}{6}$.

4- Description des programmes

Tous les programmes ont été élaborés en langage PASCAL (annexe).

Dans tout ce qui suit nous adopterons les notations suivantes:

- G(V,E) est le graphe initial.
- T = taille du problème, c'est à dire le nombre de villes.
- D = matrice des distances symétrique de dimension (T × T).

  D(i, j) = la distance entre les villes i et j.

  $\forall$ i, $\forall$ j : a) d (i, j) = d (j, i)

  b) d (i, j) = 1 ou 2 si $i \neq j$

  c) d (i, j) = 0 si i = j.
- X = matrice des solutions telle que X (i, j) = variable de décision pour tout i et tout j.

  X(i, j) = 1 si on décide d'aller de la ville i à la ville j,

  X(i, j) = 0 sinon.

4-1- Programme du 2-couplage optimum

C'est un sous-graphe de G(V,E) qui vérifie les conditions suivantes:

- le degré de chaque sommet est exactement égal à 2,
- le poids total de ce sous-graphe est minimal,
- le sous-graphe ne contient pas de boucle.

4-1-1- Mise en oeuvre de l'algorithme:

Le problème du 2-couplage optimum peut être formulé de la façon suivante:

$$\text{Min } Z = \sum_{i=1}^{T} \sum_{j=1}^{T} D(i,j) \times X(i,j) \quad \text{.....(1)}$$

$$\sum_{j=1}^{T} X(i,j) = 2 \quad \text{.....(2)}$$

$$\sum_{j=1}^{T} X(j,i) = 2 \quad \text{.....(2')}$$

$$X(i,i) = 0 \ , \forall i \quad \text{.....(3)}$$

$$X(i,j) \in \{0,1\} \ , \forall i, \forall j \quad \text{.....(4)}$$

(1) Z est la fonction objective à minimiser, à savoir la distance totale parcourue.

(2), (2') Ces contraintes représentent le fait que le degré de chaque sommet est exactement 2.

(3) Cette contrainte exprime le fait que le sous-graphe ne contient pas de boucles.

Nous avons T×T variables de décision, mais X(i, j) = 0 pour tout i, nous en éliminons donc T variables.

Le problème étant symétrique, (X(i, j) = X(j, i), $\forall$ i, j), nous ne considérons donc que la moitié des variables; c'est à dire la partie triangulaire supérieure: X(i, j) tels que $i < j$ ou inférieure: x(i, j) tels que $i > j$. Nous choisirons la première.

Soit N le nombre de variables, N = ((T-1) × T)) / 2.

La formulation du problème de 2-couplage optimum devient:

$$\text{Min } Z = \sum_{i=1}^{T} \sum_{j=i+1}^{T} D(i, j) \times X(i, j),$$

$$\sum_{j=i+1}^{T} X(i, j) + \sum_{j=1}^{i-1} X(j, i) = 2 , \ \forall i,$$

$$X(i, j) = 0,1 \text{ pour tout } i, j \in V, \ j>i.$$

Afin de simplifier le problème, nous allons l'exprimer sous la forme d'un problème linéaire:

Soit Y l'ensemble de toutes les variables de décision, nous transformons la matrice triangulaire supérieure X en un vecteur Y en mettant les éléments de X dans le vecteur Y, ligne par ligne. La taille de Y sera N.

Y = (X(1,2) , X(1,3) , ... , X(1,T), X(2,3) , ... ,X(2,T) , ... , X(T-1,T))

Soit C le vecteur des distances qu'on obtient à partir de la matrice D de la même façon que Y.

C = (D(1,2) , D(1,3) , ... , D(1,T) ; D(2,3) , ... ,D(2,T) , ... ,D(T-1,T)). La taille de C est N.

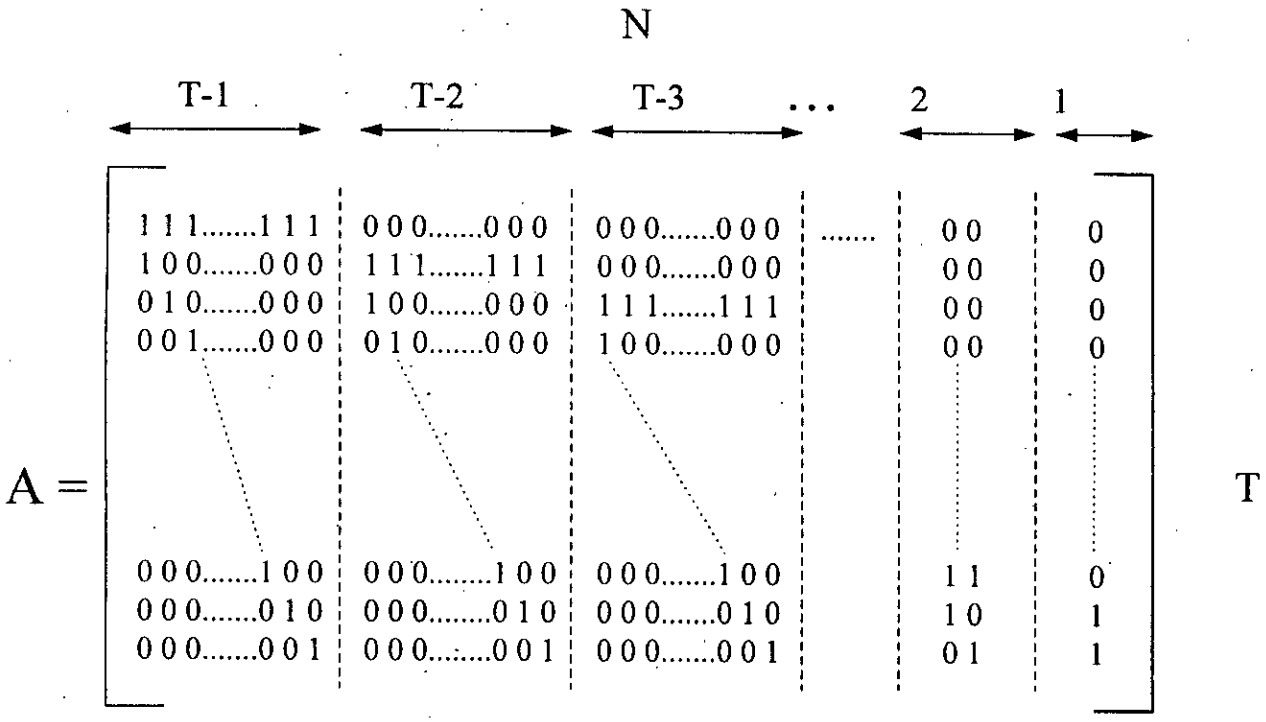
Le problème devient:

$$\text{Min } C.Y^T$$

$$A.Y^T = B, \quad y_i \in \{0,1\}$$

Où $B = (2, 2, ..., 2)^T$, de taille T et A est la matrice des contraintes obtenues par construction.

On obtient la matrice A suivante:



En effet, les contraintes initiales sont les suivantes:

$$\sum_{j=i+1}^{T} X(i,j) + \sum_{j=1}^{i-1} X(j,i) = 2 \text{ , pour i allant de 1 à T. ie:}$$

$$X(1,2) + X(1,3) + X(1,4) + \ldots\ldots + X(1,T-1) + X(1,T) = 2$$

$$X(1,2) + X(2,3) + X(2,4) + \ldots\ldots + X(2,T-1) + X(2,T) = 2$$

$$X(1,3) + X(2,3) + X(3,4) + \ldots\ldots + X(3,T-1) + X(3,T) = 2$$

$$X(1,T) + X(2,T) + X(3,T) + \ldots\ldots + X(T-2,T) + X(T-1,T) = 2$$

On remplace chaque X(i, j) par le Y(k) correspondant:

$$Y(1) + Y(2) + Y(3) + \ldots\ldots + Y(T-1) = 2$$

$$Y(1) + Y(T) + Y(T+1) + \ldots\ldots + Y((T-1)+(T-2)) = 2$$

$$Y(2) + Y(T) + Y((T-1)+(T-2)+1) + \ldots\ldots + Y((T-1)+(T-2)+(T-3)) = 2$$

Afin de former un solution, nous devons choisir T variables du vecteur Y. Pour que celle-ci soit réalisable, il faut que la somme des colonnes de A correspondant aux variables choisies soit le vecteur B.

Afin de minimiser l'espace mémoire, la matrice A ne sera pas stockée entièrement, mais on appellera à chaque fois la colonne de A dont on a besoin à l'aide de la procédure suivante:

Procédure Contrainte (calcule aa : la j-ème colonne de A),

(0) Initialisation:

$S_1 \leftarrow 0$;

$S_2 \leftarrow 0$;

aa $\leftarrow$ vecteur nul;

(1) Pour i = 1 à T-1 faire

$S_2 \leftarrow S_2 + 1$;

$S_2 \leftarrow S_2 + (T-i)$;

Si $S_2 \leq j \leq S_2$ alors aa (i) $\leftarrow 1$ et aa $(i+j-S_2+1) \leftarrow 1$

sinon aller en (1).

4-1-2- Enoncé de l'algorithme

Entrée: le vecteur des distances C.

Sortie: la matrice X.

Variables:

$B = (2,2,....,2)^T$.

$C = (c_1, c_2, ..., c_N)$.

J= ensemble de toutes les variables, c'est à dire: J = {1,2,...,N}.

Sol= ensemble des variables choisies, Sol = {i tel que y(i) = 1}.

R= vecteur de taille T représentant les ressources totales cumulatives des variables qui appartiennent à l'ensemble Sol: Une solution est réalisable si le vecteur R correspondant est inférieur ou égal au vecteur B

Pour i = 1 à N, aa (i) est le vecteur colonne de la matrice des contraintes A, qui correspond à la variable i, ie : aa (i) = $(a_{1i}, a_{2i}, ..., a_{Ti})$, d'où R= $\sum$ aa (i).

Enoncé:

(0) $S \leftarrow \varnothing$;

$R \leftarrow$ T-vecteur nul;

$Y \leftarrow$ N-vecteur nul;

$Z \leftarrow 0$.

(1) Soit Cand l'ensemble des variables candidates, c'est à dire:

Cand = $\{j$ tel que $j \in J\text{-}S$ et $R + aa(j) \leq B\}$.

Si Cand = $\varnothing$ alors STOP

(2) Parmi les éléments de Cand, choisir une variable j telle que c (j) est égal à 1.

Si une telle variable n'existe pas, en prendre une arbitrairement.

(3) $S \leftarrow S \cup \{j\}$;

$R \leftarrow R + aa(j)$;

$Z \leftarrow Z + c(j)$;

$Y(j) \leftarrow 1$;

Aller en (1).

(4) A partir du vecteur Y, reconstruire la matrice X de la façon suivante:

$k = 1$,

pour $i = 1$ à T faire

pour $j = 1+i$ à T faire $X(i, j) = Y(k)$, $k = k + 1$.

Remarque

Etant donné que les variables sont choisies au hasard parmi les variables candidates, on peut dérouler cet algorithme un certain nombre de fois (5 par exemple) afin de garder la meilleure solution obtenue.

4-1- Programme de Algorithme 1

Nous relions les cycles obtenus à partir du 2-couplage afin de faire un tour, en éliminant le plus possible d'arêtes de valeur 2.

Enoncé de l'algorithme:

Entrée: vecteur des solutions Sol.

Sortie: T-vecteur Tour.

Etape1: Initialisation.

Tour ← T-vecteur nul ,

ntour = 0 (où ntour est le nombre de variables du vecteur tour).

Etape2: Procédure Cherche-Cycle.

Reconstituer les cycles à partir de la matrice X :

(1) choisir un sommet i

(2) chercher un sommet j tel que x(i, j) = 1 ou x(j, i) = 1

(3) i ← j, aller en (2).

Réitérer jusqu'à ce qu'un tel sommet j n'existe pas.

On obtient ainsi un cycle représenté dans un vecteur qu'on appellera Cycle, soit ncycle le nombre de sommets qu'il contient.

Si le cycle trouvé est le premier **alors** Tour ← Cycle,

ntour ← ncycle,

aller à l'étape 2.

Sinon aller à l'étape 3.

Etape 3: Procédure Position.

Choisir une arête de valeur 2 du cycle obtenu dans l'étape 2, et une arête du vecteur Tour de valeur 2 également.

Si une telle arête n'existe pas, alors choisir une arête au hasard, soit la première dans le vecteur considéré.

Etape 4: Procédure Tournée.

Relier le cycle et le tour en enlevant les 2 arêtes trouvées en étape 3.

ntour ← ntour + ncycle.

Si ntour = T alors STOP

Sinon aller à l'étape 2.

Remarque:

Relier un vecteur A dont le nombre éléments est na à un vecteur B dont le nombre éléments est nb, signifie la procédure suivante:

(1) Chercher i tel que D(A(i) , A(i +1)) = 2.

(2) Chercher j tel que D(B(j) , B(j+1)) =2.

(3) Insérer B dans A de la façon suivante:

A(1),A(2),...,A(i), B(j+1), B(j+2),..., B(nb), B(1), B(2),...,B(j),A(i+1),A(i+2),...,A(na).

Ainsi les arêtes (A(i) , A(i+1)) et (B(j) , B(j+1)) sont éliminées.

Dans tout ce qui suit, à chaque fois qu'on dira relier A à B on sous-entendra " utiliser cette procédure".

4-3- Programme de Algorithme 2

4-3-1- Mise en oeuvre de l'algorithme

L'algorithme2 se résume aux étapes suivantes:

Etape 1: Chercher l'ensemble des cycles à partir de la solution du 2-couplage obtenue, et le transformer afin qu'il vérifie la condition (1) et la condition (2).

Etape 2: Former le graphe biparti B.

Etape 3: Chercher un couplage optimum dans B.

Etape 4: Former le graphe orienté F.

Etape 5: Chercher F' le sous-graphe couvrant de F, c'est à dire les chemins de longueur 2 et les arbres de profondeur 1.

Etape 6: Relier les éléments du sous-graphe couvrant ainsi que le cycle impur afin de former un tour.

4-3-2- Enoncé de l'algorithme

Entrée: X : a matrice des solutions du 2-couplage.

Sortie: Tour : un vecteur qui représente le tour obtenu.

Etape1: Procédure Matrice-Cycles.

Entrée: la matrice du 2-couplage X.

Sortie: X33: une matrice dans laquelle sont classées tous les cycles, où chaque ligne représente un cycle.

nligne: le nombre total de cycles obtenus qui vérifient les deux conditions, c'est à dire le nombre de lignes de X33.

colonne: c'est un vecteur donnant le nombre de sommets de chaque cycle, c'est à dire le nombre de colonnes occupées par chaque ligne de X33.

Impur : vecteur dans lequel sera mis le cycle impur.

n-impur : nombre de sommets du cycle impur.

- Cette procédure commence par former la matrice de tous les cycles à partir de la matrice des solutions X, en utilisant la procédure Cherche (comme pour Algorithme 1).

  (0) initialiser nligne 0.

  (1) chercher un cycle, soit ncycle le nombre de ses sommets (le cycle est représenté par un vecteur).

  (2) nligne ← nligne +1;

  X33 (nligne , j) ← cycle (j);

  colonne (nligne) ← ncycle.

  S'il existe des sommets pas encore rencontrés alors aller en (1).

- Puis elle transforme X33 pour que tous les cycles vérifient la condition 1.

  (0) initialiser n-impur ← 0.

  (1) pour i allant de 1 à nligne faire

  Si D(X33(i, j) ; X33(i, j+1)) = 2 **alors**

Début

Si n-impur = 0 alors Impur ← i-ème ligne de X33;

n-impur ← colonne (i).

Sinon relier impur et la i-ème ligne de X33;

colonne(i) = 0 (on élimine la i-ème ligne).

Fin.

- Afin de vérifier la condition (2), la procédure procède ainsi:

(1) pour i allant de 1 à nligne faire

Si colonne (i) ≠ 0 **alors**

Début

Pour j allant de 1 à n-impur faire

Si D (Impur (j), Impur (j+1)) =2 alors aller en (2).

Fin.

(2) pour k allant de 1 à colonne (i) faire

Si D (X33 (i, k), Impur (j)) =1 **alors**

Relier la i-ème ligne de X33 au vecteur Impur , colonne (i) =0.

Finalement, la procédure réorganise la matrice X33 en éliminant les lignes i telles que colonne (i) = 0 et calcule le nouveau nombre nligne.

Etape 2: Procédure Graphe-Biparti.

Entrée: X33 , nligne , colonne.

Sortie: Biparti = une matrice (nligne × T) représentant le graphe biparti.

(0) initialiser Biparti (i, j) ← 0 pour i = 1 à nligne,

pour j = 1 à T.

(1) Biparti (i ,j) = 1 si

- X33 (i, k) ≠ j pour k = 1 à colonne (i)
- ∃ k ≤ colonne (i) tel que D (j, k) = 1.

Etape 3: Procédure Couplage-Biparti.

Entrée: Biparti.

Sortie: match1 = vecteur de taille nligne tel que match (i) est le sommet auquel est couplé le cycle contenu dans la i-ème ligne de la matrice X33.

- Le graphe Biparti est construit de l'ensemble C vers l'ensemble V.

La procédure utilise 3 vecteurs: match1, match2 et exposé.

match2 : vecteur de taille T tel que match2 (j) = le cycle i couplé au sommet j.

exposé : vecteur de taille nligne tel que exposé(cycle i) = un sommet j tel que B(i, j) = 1 et qui est exposé, c'est à dire non couplé.

- L'idée de la procédure est d'associer à chaque cycle i, le sommet exposé (i).

Si exposé(i) = 0 et match1 (i) = 0, c'est à dire le cycle i n'a pas été couplé et aucun des sommets auxquels il est relié dans B n'est exposé, **alors** examiner ces sommets. Soit j un tel sommet:

- si exposé (match2 (j)) $\neq$ 0, on peut alors associer le cycle match2 (j) à un autre sommet que j, à savoir exposé (match2 (j)), ainsi on libère le sommet j afin de lui coupler le cycle i..
- si par contre, pour tous ces sommets j, exposé (match2 (j)) = 0 alors on ne peut pas coupler le cycle i donc le couplage ne sera pas parfait.

- On utilise un graphe (C, A) où C est l'ensemble des cycles tel que A est construit de la façon suivante: (i1, i2) $\in$ A si match1 (i1) = 0 ,

$\exists$ j tel que B(i1, j) = 1 avec match2 (j) = i2.

Il servira à trouver les cycles qu'on peut recoupler afin de libérer un sommet pour un cycle qui n'est pas couplé et dont aucun des sommets auxquels il est relié n'est exposé.

Enoncé de la procédure

Le vecteur étiquette et l'ensemble Q seront utilisés pour la recherche.

(0) initialiser : match1 ← vecteur nul.

match2 ← vecteur nul.

(1) exposé(i) ← 0 pour i allant de 1 à nligne.

pour tout (i, j) ∈ B faire

si match2 (j) = 0 **alors** exposé(i) = j **sinon**

si match2 (j) ≠ i **alors** A ← A ∪ (i, match2(j)).

(2) Q ← ∅,

Pour tout i ∈ C faire si match1 (i) = 0 alors Q ← Q ∪ {i}

étiquette(i) ← 0.

Tant que Q ≠ ∅ faire

Début

Soit i un cycle de Q;

enlever i de Q,

Si exposé(i) ≠ 0 **alors** augmenter (i) et aller en (1)

Sinon pour tout k non étiqueté (étiquette(k) = 0) tel que (i, k) ∈ A

faire étiquette(k) = i et Q = Q ∪ {k}.

Fin.

Procédure augmenter (i)

Si étiquette(i) = 0 **alors** match1 (i) ← exposé (i)

match2 (exposé (i)) ← i,

Sinon exposé (étiquette (i)) ← match1 (i);

match1 (i) ← exposé (i);

match2 (exposé (i)) ← i;

augmenter (étiquette (i)).

Etape 4: Procédure Graphe-F.

Entrée : match1.

Sortie : FF matrice (nligne × nligne) qui représente le graphe F.

(0) Initialiser FF ← 0,

(1) Pour tout couple de cycles (i1 ,i2) faire

FF(i1, i2) = 1 si match1(i1) = j avec j ∈ i2 (c'est à dire ∃ k tel que X33(i2, k) =).

Etape5, Etape 6 : Procédure Tournée2.

Entrée : la matrice FF.

Sortie : Tour.

Soient :

I = ensemble de tous les sommets de F.

deg = vecteur qui représente le degré intérieur de chaque sommet.

suc = vecteur de taille nligne tel que suc (i) = le sommet successeur de i dans le graphe F.

(0) Chercher le successeur de chaque sommet.

(1) Calculer le degré intérieur de chaque sommet du graphe.

(2) Pour tous les sommets i tel que deg (i) = 0 faire

Début

Soit j = suc (i);

Si deg (j) = 1 **alors** si deg (k) = 1 alors procédure Chemin2 (i, j, k)

J ← J- {i, j, k};

sinon procédure Chemin1 (i, j)

J ← J-{i, j};

Sinon - chercher dans J les sommets l tels que : suc (l) = 0 et deg (i) = 0

soient l1, l2, ..., lm ces sommets;

- Procédure Arbre (j, i, l1, l2, ..., lm) ;

(j est la racine de l'arbre et les lm sont les feuilles de l'arbre).

- J ← J-{j, l1, l2, ..., lm}.

(3) Relier le cycle obtenu (par la procédure chemin1, chemin2 ou arbre) au vecteur Tour.

(4) - Relier au vecteur Tour final les sommets isolés du graphe F' c'est à dire les sommets i tels

que deg (i) = 0 et suc (i) = 0.

- J ← J-{i}.

- **Si** J ≠ ∅ **alors** aller en (1)

Sinon Relier le cycle impur au vecteur Tour . STOP.

4-4- Algorithme exact

Afin d'obtenir le rapport d'approximation pour les deux algorithmes, il faut connaître pour chaque instance utilisée dans le test, la valeur de la solution optimale correspondante.
Un algorithme exact, basé sur la méthode de Branch and Bound, sera élaboré à cette fin.
La méthode de Branch and Bound est une méthode énumérative qui résout les problèmes d'optimisation combinatoire, elle se résume dans les étapes suivantes:
(1) Diviser l'ensemble des solutions réalisables en sous-ensembles successivement plus petits.
(2) Calculer des bornes pour la valeur de la fonction objective pour chaque sous-ensemble.
(3) Utiliser les bornes afin d'écarter certains sous-ensembles des prochaines considérations.
(4) La procédure est arrêtée lorsque chaque sous-ensemble produit une solution meilleure que la solution obtenue.
La meilleure solution trouvée durant cette procédure constitue un optimum global.
Les ingrédients essentiels pour toute méthode de Branch and Bound sont:
- une règle de séparation pour diviser l'ensemble courant en sous-ensembles,
- une borne,
- une procédure d'évaluation pour décider d'écarter ou non le sous-ensemble courant.
- une règle d'exploration (en profondeur d'abord ou en largeur d'abord).

4-4-1- Algorithme de Branch and Bound , profondeur d'abord.

Initialisation

OPT = 2 × T

Tour = (1 0 0 . . . 0 0)

Création d'un noeud père.

Début

Répéter tant que toute l'arborescence n'a pas été parcourue:

Si l'exploration du noeud père est autorisée à droite et à gauche

création d'un noeud fils gauche. Mise à jour de Tour

Si le noeud se stérilise à cette étape

exploration du noeud fils interdite à droite et à gauche.

Sinon

Si le noeud fournit une solution réalisable,

meilleure que la solution déjà enregistrée

alors mémorisation de cette solution.

Descendre d'un noeud dans l'arborescence:

ce noeud fils gauche, dernièrement créé, devient le noeud père.

Si l'exploration du noeud père est autorisée à droite et interdite à gauche

création d'un noeud fils gauche. Mise à jour de Tour

Si le noeud se stérilise à cette étape

exploration du noeud fils interdite à droite et à gauche.

Sinon

Si le noeud fournit une solution réalisable,

meilleure que la solution déjà enregistrée

alors mémorisation de cette solution.

Descendre d'un noeud dans l'arborescence:

ce noeud fils droit, dernièrement créé, devient le noeud père.

Si l'exploration du noeud père est autorisée à droite et interdite à gauche

Remonter d'un noeud:

Le noeud immédiatement au dessus dans l'arborescence devient le noeud père.

Si l'ancien noeud père était gauche

exploration à gauche de l'actuel noeud père interdite.

Sinon

(l'ancien père était à droite)

exploration à droite de l'actuel noeud père interdite.

La place mémoire occupée par l'ancien noeud père est désallouée.

Fin.

4-4-2- Mise en oeuvre de la Procédure Algorithme-Exact

Entrée : Zmin = borne.

D = matrice des distances.

Sortie : Z = longueur du tour optimal.

(0) Initialisation:

Etant donné qu'un tour est un cycle, certaines permutations peuvent être les mêmes. On

peut remédier à cela en fixant la première variable du tour, soit la ville1.

ntour ← 1 : c'est le nombre de variables sélectionnées dans le vecteur Tour.

Tour (1) ← 1,

Tour (i) ← 0 pour i = 2 à T.

Z ← 0 : c'est la longueur du tour obtenu.

Zmin = min {Z1, Z2}, où Z1 et Z2 sont respectivement les longueurs des tours obtenus par algorithme1 et algorithme2, qui constituent une bonne borne pour l'algorithme exact.

(1) Procédure Permutation.

```
Pour i = 2 à T faire
pour j = 1 à ntour faire
   Si tour(j) ≠ i alors
   Début
   ntour ← ntour + 1;
   Tour (ntour) ← i;
   Z ← Z + D (tour (ntour - 1) , tour (ntour));
   Si Z < Zmin alors
      si ntour = T alors Zmin ← Z
      sinon Permutation;
   Fin.
```

5- Résultats

5-1- Mode opératoire

Pour évaluer les performances des algorithmes proposés, nous avons crée aléatoirement des groupes de 10 instances arbitraires du problème du voyageur de commerce avec des distances 1 et 2.

Ainsi nous avons successivement 10 instances pour un problème de T = 30 villes, 10 instances pour T = 40, ..., jusqu'à T = 100.

Nous avons choisi des instances dont la taille est supérieure ou égale à 30 afin que Algorithme2 ne perde pas son intérêt. En effet, il faut que le 2-couplage contienne suffisamment de cycles pour que la technique de Algorithme 2 soit intéressante à appliquer.

Le programme élaboré produira pour chaque instance la valeur du tour produit par Algorithme1 (Z1), la valeur du tour de Algorithme2 (Z2) puis la valeur du tour optimal (Z3).

R1 = rapport d'approximation de Algorithme1.

= Z1/Z3.

R2 = rapport d'approximation de Algorithme2.

= Z2/Z3.

Nous devons vérifier que $R1 \leq 4/3$ et que $R2 \leq 11/9$.

- Durant les tests, pour les instaces de grande taille, nous n'avons pas toujours déroulé l'algorithme exact pour réduire l'espace temps. En effet, dans certains cas, il n'est pas néssaire de connaître la solution exacte:

nous savons que $OPT(I) \geq n$ d'où $\frac{A(I)}{OPT(I)} \leq \frac{A(I)}{n}$.

On commence donc par comparer $\frac{A(I)}{n}$ au rapport théorique, et on n'utilise l'algorithme exact que si le rapport $\frac{A(I)}{n}$ est superieur au rapport théorique.

Dans ces cas, R1 = A(I) / n et R2 = A(I) / n.

5-2- Etude comparative.

T =30

Instance	1	2	3	4	5	6	7	8	9	10
Z1	37	35	38	32	36	34	31	41	38	30
Z2	35	35	35	32	35	35	32	40	38	30
Z3	35	32	34	30	32	34	30	39	35	30
R1	1.057	1.000	1.118	1.067	1.125	1.000	1.033	1.051	1.085	1.000
R2	1.000	1.094	1.O29	1.067	1.094	1.029	1.066	1.025	1.085	1.000

T = 40

Instance	1	2	3	4	5	6	7	8	9	10
Z1	45	47	41	47	43	51	42	45	42	49
Z2	45	43	42	45	43	48	42	42	40	45
Z3	41	42	40	44	40	48	41	42	40	42
R1	1.097	1.119	1.025	1.068	1.075	1.062	1.024	1.071	1.050	1.089
R2	1.097	1.023	1.050	1.022	1.075	1.000	1.024	1.000	1.050	1.071

T =50

Instance	1	2	3	4	5	6	7	8	9	10
Z1	56	53	58	54	54	61	56	52	53	54
Z2	54	53	57	51	55	58	55	52	51	52
Z3	54	51	55	51	54	56	52	52	50	52
R1	1.037	1.039	1.017	1.059	1.000	1.089	1.077	1.000	1.060	1.038
R2	1.000	1.039	1.036	1.000	1.018	1.036	1.058	1.000	1.020	1.000

T = 60

Instance	1	2	3	4	5	6	7	8	9	10
Z1	65	62	69	71	63	65	67	72	63	68
Z2	65	60	64	67	62	64	67	69	63	65
Z3	62	60	63	64	60	62	66	68	63	64
R1	1.048	1.033	1.095	1.109	1.050	1.048	1.015	1.059	1.000	1.062
R2	1.048	1.000	1.016	1.047	1.033	1.032	1.015	1.014	1.000	1.015

T =70

Instance	1	2	3	4	5	6	7	8	9	10
Z1	73	75	78	72	83	78	76	70	79	74
Z2	71	75	74	72	80	77	76	70	76	75
Z3	70	71	74	70	79	72	74	70	75	72
R1	1.043	1.056	1.054	1.028	1.051	1.083	1.027	1.000	1.053	1.028
R2	1.014	1.056	1.000	1.028	1.013	1.069	1.027	1.000	1.013	1.042

T = 80

Instance	1	2	3	4	5	6	7	8	9	10
Z1	89	85	84	90	88	92	82	88	92	93
Z2	84	84	82	90	85	89	82	87	91	91
Z3	—	—	—	84	—	88	—	87	90	87
R1	1.112	1.062	1.050	1.071	1.100	1.045	1.025	1.011	1.022	1.069
R2	1.050	1.050	1.025	1.071	1.062	1.011	1.025	1.000	1.011	1.046

T = 90

Instance	1	2	3	4	5	6	7	8	9	10
Z1	94	98	104	96	110	94	112	109	93	101
Z2	92	96	102	92	110	92	110	104	93	101
Z3	—	—	97	—	106	—	101	99	—	96
R1	1.044	1.088	1.072	1.067	1.038	1.044	1,109	1.101	1.033	1.052
R2	1.022	1.066	1.051	1.022	1.038	1.022	1.089	1.050	1.033	1.052

T =100

Instance	1	2	3	4	5	6	7	8	9	10
Z1	110	105	113	114	104	116	108	112	119	108
Z2	107	105	112	110	102	115	105	112	113	105
Z3	—	—	106	108	—	112	—	107	113	—
R1	1.100	1.050	1.066	1.055	1.040	1.036	1.080	1.047	1.053	1.080
R2	1.070	1.050	1.057	1.018	1.020	1.027	1.050	1.047	1.000	1.050

Interpretation:

- Pour algorithme 1, les rapports d'approximation de toutes les instances sont inferieurs à 1.333 (4/3) qui est le rapport prouvé par les auteurs.

- Pour algorithme 2, les rapports d'approximation de toutes les instances sont inferieurs à 1.222 (11/9) qui est le rapport prouvé par les auteurs.

T = 60

Instance	1	2	3	4	5	6	7	8	9	10
Z1	65	62	69	71	63	65	67	72	63	68
Z2	65	60	64	67	62	64	67	69	63	65
Z3	62	60	63	64	60	62	66	68	63	64
R1	1.048	1.033	1.095	1.109	1.050	1.048	1.015	1.059	1.000	1.062
R2	1.048	1.000	1.016	1.047	1.033	1.032	1.015	1.014	1.000	1.015

T =70

Instance	1	2	3	4	5	6	7	8	9	10
Z1	73	75	78	72	83	78	76	70	79	74
Z2	71	75	74	72	80	77	76	70	76	75
Z3	70	71	74	70	79	72	74	70	75	72
R1	1.043	1.056	1.054	1.028	1.051	1.083	1.027	1.000	1.053	1.028
R2	1.014	1.056	1.000	1.028	1.013	1.069	1.027	1.000	1.013	1.042

T = 80

Instance	1	2	3	4	5	6	7	8	9	10
Z1	89	85	84	90	88	92	82	88	92	93
Z2	84	84	82	90	85	89	82	87	91	91
Z3	—	—	—	84	—	88	—	87	90	87
R1	1.112	1.062	1.050	1.071	1.100	1.045	1.025	1.011	1.022	1.069
R2	1.050	1.050	1.025	1.071	1.062	1.011	1.025	1.000	1.011	1.046

Conclusion

Dans le cadre de ce mémoire, nous avons étudié deux algorithmes approchés pour le problème du voyageur de commerce avec des distances 1 et 2, proposés par PAPADIMITRIOU et YANNAKAKIS dans leur article «The traveling salesman problem with distances 1 and 2; 1993 ».

Nous avons programmé les algorithmes et avons testé leurs rapports d'approximation. Nous avons pu ainsi vérifier par l'expérimentation que les valeurs théoriques de ces rapports sont exactes.

Néanmoins, il reste à tester la performance du troisième algorithme proposé par les auteurs, à savoir celui dont le rapport d'approximation est meilleur que 7/6. Pour cela, il faudrait utiliser l'algorithme polynomial développé par HARVIGSEN qui permet de trouver un 2-couplage optimum ne contenant pas de triangles, puis appliquer le reste de Algorithme2 intégralement.

ANNEXE

```
**************************************************************************
```

Des algorithmes pour le probleme du voyageur de commerce avec des distances 1 et 2:

- Algorithme 1 : rapport d'approximation = 4/3.

- Algorithme 2 : rapport d'approximation = 11/9.

- Algorithme 3 : algorithme exact.

```
**************************************************************************
```

DECLARATION DES VARIABLES

```
const  m1=100; m2=5000; m3=33;

type        vecteurT = array[1..m1] of integer;
            vecteurN = array[1..m2] of integer;
            matrice1 = array[1..m1,1..m1] of integer;
            matrice2 = array[1..m3,1..m1] of integer;
            matrice3 = array[1..m3,1..m3] of integer;

var         y,cand,tour1,tour2,tour3,tour,p : vecteurT;
            c:vecteurN;
            i,j,l,k,h,N,T,e,f,g,Z1,Z2,Z3,Z,ntour,ncand : integer;
            verif,fin,ed,stop,check : boolean;
            XD : matrice1;
            X33,Biparti : matrice2;
            FF : matrice3;
```

Afin d'économiser de l'espace mémoire, nous utiliserons la procedure XDMatrice pour représenter les matrices des solutions X et des distances D qui sont symetriques. Ainsi la partie triangulaire superieure de la matrice XD sera reservée aux solutions et la partie triangulaire inferieure aux distances.

```
Procedure XDMatrice;

var     ch : boolean;
        cc : vecteurN;

      Begin
       for i:=1 to N do cc[i]:=0;
       for i:=1 to T do  begin  e:=y[i];cc[e]:=1 end;
       l:=1;
       for i:=1 to T do
       Begin
        XD[i,i]:=0;
        for j:=1+i to T do
        Begin
          XD[i,j]:=cc[l];
          XD[j,i]:=c[l];
          l:=l+1;
        End;
       End;
      End;
```

Ces deux fonctions sont utilisées pour lire la matrice XD selon que l'on a besoin d'une distance ou d'une variable solution.

```
Function     Distance(a,b : integer): integer;

      Begin
       if a>b then distance:=xd[a,b]
            else distance:=xd[b,a]
      End;

Function Arc(a,b : integer):integer;

      Begin
        if a<b then arc:=xd[a,b]
          else arc:=xd[b,a]
      End;
```

La procedure position cherche une arête de valeur 2 dans un cycle.

```
Procedure Position (var a:vecteurT; na:integer; var p:integer);
    var trouve : boolean;
    Begin
    trouve:=false;
    a[na+1]:=a[1];
    for h:=1 to na do
     if not trouve then
     Begin
      g:=distance(a[h],a[h+1]);
      if g=2 then
      Begin
       trouve:=true;
       p:=h
      End;
     End;
    if not trouve then p:=1;
    End;
```

Cette procedure cherche les cycles formés dans le 2-couplage à partir de la matrice des solutions X.

```
Procedure Cherche(var a :vecteurT ; var na :integer);
 var  ch : boolean;
 Begin
  a[1]:=cand[1];
  cand[1]:=cand[ncand];
  ncand:=ncand-1;
  na:=na+1;
  ch:=false;
  while not ch do
  Begin
   ch:=true;
   for i:=1 to ncand do
   if ch then
   Begin
    g:=Arc(a[na],cand[i]);
    if g=1 then
    Begin
     ch:=false;
     a[na+1]:=cand[i];na:=na+1;
     cand[i]:=cand[ncand];
     ncand:=ncand-1;
    End;
   End;
  End;
 End;
```

Nous utiliserons la procedure Longueur à chaque fois que nous voudrons calculer la longueur d'un tour.

```
Procedure Longueur(aa:vecteurT;var zz:integer);

 Begin
  zz:=0;
  for i:=1 to T-1 do
  zz:=zz+distance(aa[i],aa[i+1]);
  zz:=zz+distance(aa[T],aa[1]);
 End;


    (*-------------------------------------------------------*)
              Procedure Deux_Couplage;
    (*-------------------------------------------------------*)

Var     numcan,numsol,ncan,rd,Jmin,essai,zmin : integer;
        a,sol,r : VecteurT;
        cand : vecteurN;
        egal : array[1..2500] of integer;
```

Etant donné que la matrice des contraintes A prend une place mémoire trés mportante, elle ne sera pas stockée , mais on calculera à chaque fois la colonne dont on aura besoin à l'aide de cette procedure.

```
Procedure  Contrainte(v:integer; var b:vecteurT);
  var  s1,s2 : integer;

Begin
 for i:=1 to T do b[i]:=0;
 s1:=0;s2:=0;cd:=false;
 for i:=1 to T-1 do
 if not cd then
 Begin
  s1:=s2+1;s2:=s2+T-i;
  if v<=s2 then if v>=s1 then
  Begin
   b[i]:=1;b[i+v-s1+1]:=1;
   cd:=true
  End;
 End;
End;
```

```
Begin (* Programme principal de la procedure 2_Couplage *)

Zmin:=2*T;
for l:=1 to T do y[l]:=0;

for essai:=1 to 5 do

Begin
 stop:=false;
 while not stop do
 Begin

 {Choix de la premiere variable}

 for i:=1 to T do r[i]:=0;
 k:=1;
 numcan:=N;
 numsol:=1;
 Z:=0;
 for j:=1 to numcan do
 if k<=2500 then
 Begin
  cand[j]:=j;
  if c[j]=1 then  Begin  egal[k]:=j;  k:=k+1;  End;
 End;
 randomize;
 rd:=random(k-1)+1 ;
 Jmin:=egal[rd];
 Z:=Z+c[Jmin];
 sol[numsol]:=Jmin;
 cand[Jmin]:=cand[numcan];
 numcan:=numcan-1;
 Contrainte(Jmin,a);
 for i:=1 to T do r[i]:=r[i]+a[i];

 {Choix des autres variables}

 fin:=false;
 while fin=false do
 Begin
   ncan:=numcan;
   numcan:=0;
   for j:=1 to  ncan do
   Begin
     h:=cand[j];
     check:=true;
     Contrainte(h,a);
```

```
      for i:=1 to T do
      if (r[i]+a[i]>2) then check:=false;
      if check then
      Begin
        numcan:=numcan+1;
        cand[numcan]:=h;
      End;
    End;

    l:=1;
    for i:=1 to numcan do
    if l<=2500 then
    Begin
      e:=cand[i];
      if c[e]=1 then begin egal[l]:=i;l:=l+1 end;
    End;
    randomize;
    rd:=random(l-1)+1;
    if l=1 then e:=1 else e:=egal[rd];
    Jmin:=cand[e];
    if numcan=0 then fin:= true;
    if fin=false then
    Begin
      Z:=Z+c[Jmin];
      numsol:=numsol+1;
      sol[numsol]:=Jmin;
      cand[e]:=cand[numcan];
      numcan:= numcan-1;
      Contrainte(jmin,a);
      for i:=1 to T do r[i]:=r[i]+a[i];
    End;

  End;

  verif:=true;
  if z>=T then
  if z<zmin then
  Begin
    if numsol<T then verif:=false;
    if verif then
    Begin
      zmin:=Z;
      for i:=1 to numsol do y[i]:=sol[i];
    End;
  End;if y[1]<>0 then stop:=true;
  End;
End;
End;
```

```
(*-------------------------------------------------------------*)
                Procedure Algorithme_1;
(*-------------------------------------------------------------*)

Var     cycle : vecteurT;
        ncycle : integer;
```

Cette procedure cherche les cycles puis les relie suivant la méthode décrite dans l 'algorithme 1et calcule la longueur du tour obtenu.

```
Procedure Tournee1;

 var c1 : vecteurT;
    n1,n2 : integer;

 Begin
  for l:=1 to T do
  Begin
   cand[l]:=l;tour[l]:=0;cycle[l]:=0
  End;
  ncand:=T;ntour:=0;ncycle:=0;
  Cherche(cycle,ncycle);
  for l:=1 to ncycle do tour[l]:=cycle[l];
  ntour:=ncycle;
  While ntour<T do
  Begin
   for l:=1 to T do cycle[l]:=0;
   ncycle:=0;
   cherche(cycle,ncycle);
   position(tour,ntour,n1);
   position(cycle,ncycle,n2);
   for i:=1 to (ncycle-n2) do c1[i]:=cycle[n2+i];
   for i:=1 to n2 do c1[(ncycle-n2)+i]:=cycle[i];
   for i:=1 to n1 do p[i]:=tour[i];
   for i:=1 to ncycle do p[i+n1]:=c1[i];
   for i:=1 to ntour-n1 do p[n1+ncycle+i]:=tour[n1+i];
   ntour:=ntour+ncycle;
   for i:=1 to ntour do tour[i]:=p[i];
  End;
  for i:=1 to T do tour1[i]:=tour[i];
 End;

Begin
 Tournee1;
 Longueur(tour1,Z1);
End;
```

```
(*-------------------------------------------------------------*)
                Procedure Algorithme_2;
(*-------------------------------------------------------------*)

Var             colonne,match1,npur,cycle : vecteurT;
                nligne,nnpur : integer;
```

Cette procedure cherche tous les cycles produits par le 2-couplage, puis les arrange de façon à ce qu'ils vérifient les deux conditions imposées. Elle finit par classer les cycles purs dans la matrice X33 et le cycle impur dans le vecteur npur.

```
Procedure Matrice_Cycles;

 var  cycle,c1,npur1:vecteurT;
      ncycle,n1,n2,nnpur1:integer;

Begin

{chercher tous les cycles et les mettre dans la matrice X33}
ncand:=T;h:=0;
for l:=1 to T do cand[l]:=l;
while ncand<>0 do
Begin
 for l:=1 to T do cycle[l]:=0;ncycle:=0;
 Cherche(cycle,ncycle);
 h:=h+1;
 colonne[h]:=ncycle;
 for l:=1 to ncycle do X33[h,l]:=cycle[l];
End;
nligne:=h;

{verifier la condition 1}

nnpur:=0;
for i:=1 to nligne do
Begin
 verif:=true;
 for l:=1 to colonne[i] do cycle[l]:=X33[i,l];
 ncycle:=colonne[i];
 cycle[ncycle+1]:=cycle[1];
 for j:=1 to ncycle do
```

```
Begin
 if verif then
 Begin
  e:=cycle[j]; f:=cycle[j+1];
  g:=distance(e,f);
  if g=2 then
  Begin
   verif:=false;
   if nnpur=0 then
   Begin
    for l:=1 to ncycle do npur[l]:=X33[i,l];
    nnpur:=nnpur+ncycle
   End
   else
   Begin
    n2:=j;
    position(npur,nnpur,n1);
    for l:=1 to ncycle-n2 do c1[l]:=cycle[n2+l];
    for l:=1 to n2 do c1[ncycle-n2+l]:=cycle[l];
    for l:=1 to n1 do p[l]:=npur[l];
    for l:=1 to ncycle do p[n1+l]:=c1[l];
    for l:=1 to nnpur-n1 do p[n1+ncycle+l]:=npur[n1+l];
    nnpur:=nnpur+ncycle;
    for l:=1 to nnpur do npur[l]:=p[l];
   End;
   colonne[i]:=0
  End;
 End;
End;

{verifier la condition 2}

if nnpur <>0 then
if nnpur <>T then
Begin
nnpur1:=nnpur;
for i:=1 to nnpur do npur1[i]:=npur[i];
 j:=1;
 while j<=nnpur do
 Begin
  verif:=false;
  npur[nnpur+1]:=npur[1];
  e:=npur[j];f:=npur[j+1];
  g:=distance(e,f);
  if g=2 then
```

```
Begin
 i:=1;
  while i<=nligne do
  Begin
   if colonne[i]<>0 then
   Begin
    ncycle:=colonne[i];
    for l:=1 to ncycle do cycle[l]:=X33[i,l];
    cycle[ncycle+1]:=cycle[1];
    k:=1;
    while k<=ncycle do
    Begin
     e:=npur[j];f:=cycle[k];
     g:=distance(e,f);
     if g=1 then begin verif:=true;n1:=j;n2:=k-1; end
     else
     begin
      e:=npur[j+1];g:=distance(e,f);
      if g=1 then begin verif:=true;n1:=j+1;n1:=k end
      else k:=k+1;
     end;
     if verif then
     Begin
      colonne[i]:=0;
      i:=i+nligne;
      j:=0;
      k:=k+ncycle;
      for l:=1 to ncycle-n2 do c1[l]:=cycle[n2+l];
      for l:=1 to n2 do c1[ncycle-n2+l]:=cycle[l];
      for l:=1 to n1 do npur1[l]:=npur[l];
      for l:=1 to nnpur-n1 do npur1[n1+ncycle+l]:=npur[n1+l];
      for l:=1 to ncycle do npur1[n1+l]:=c1[l];
      nnpur:=nnpur+ncycle;
      for l:=1 to nnpur do npur[l]:=npur1[l];
     End;
    End;
   End;
   i:=i+1;
  End;
 End;
 j:=j+1;
End;
End;
```

```
{arranger la matrice X33}

h:=0;
for i:=1 to nligne do
if colonne[i]<>0 then
Begin
 h:=h+1;
 for j:=1 to colonne[i] do biparti[h,j]:=X33[i,j];
 colonne[h]:=colonne[i];
End;
nligne:=h;
for i:=1 to nligne do
for j:=1 to colonne[i] do
X33[i,j]:=biparti[i,j];
ntour:=0;
if nligne=1 then
Begin
 for i:=1 to colonne[1] do tour[i]:=X33[1,i];
 ntour:=colonne[1];
 if nnpur<>0 then
 Begin
 position(tour,ntour,n1);
 position(npur,nnpur,n2);
 for i:=1 to (nnpur-n2) do c1[i]:=npur[n2+i];
 for i:=1 to n2 do c1[(nnpur-n2)+i]:=npur[i];
 for i:=1 to n1 do p[i]:=tour[i];
 for i:=1 to ncycle do p[i+n1]:=c1[i];
 for i:=1 to ntour-1 do p[n1+ncycle+i]:=tour[n1+i];
 ntour:=ntour+nnpur;
 for i:=1 to ntour do tour2[i]:=p[i];
 End;
End;
if nnpur=T then
Begin
 for i:=1 to T do tour2[i];=npur[i];
 ntour:=T
End;

End;
```

Cette procédure forme le graphe biparti entre l'ensemble des cycles (la matrice X33)et l'ensemble de tous les sommets.

```
Procedure Graphe_Biparti;

Begin
 for i:=1 to nligne do
 for j:=1 to T do Biparti[i,j]:=0;
 for l:=1 to T do
 for i:=1 to nligne do
 Begin
  verif:=true;
  for j:=1 to colonne[i] do
  if verif then if X33[i,j]=l then verif:=false;
  if verif then
  Begin
   check:=false;
   for j:=1 to colonne[i] do
   if not check then
   Begin
    e:=X33[i,j]; g:=distance(l,e);
    if g=1 then check:=true
   End;
   if check then Biparti[i,l]:=1
  End;
 End;
End;
```

Cette prucédure trouve le couplage optimal dans le graphe biparti.

```
Procedure Bipartite_Matching;

 var      etiq,expose,q,match2:array[1..33] of integer;
          aa:matrice3;
          stage,verif : boolean;

 Procedure Augment(e:integer);

 Begin
  if etiq[e]=0 then
  Begin
   match1[e]:=expose[e];
   f:=expose[e];
   match2[f]:=e;
  End
  else
```

```
  Begin
    f:=etiq[e];expose[f]:=match1[e];
    match1[e]:=expose[e];
    f:=expose[e];match2[f]:=e;
    f:=etiq[e];augment(f)
  End;
 End;

Begin
 for l:=1 to nligne do match1[l]:=0;
 for l:=1 to T do match2[l]:=0;
 stage:=true;
 while stage do
 Begin
  for l:=1 to nligne do expose[l]:=0;
  for i:=1 to nligne do for j:=1 to nligne do aa[i,j]:=0;
  for i:=1 to nligne do
  for j:=1 to T do
  if Biparti[i,j]=1 then
  Begin
   if match2[j]=0 then expose[i]:=j
   else
   if match2[j]<>i then begin e:=match2[j];aa[i,e]:=1 end
  End;
  for l:=1 to nligne do q[l]:=0;
  k:=0;
  for i:=1 to nligne do
  if match1[i]=0 then
  Begin
   k:=k+1;
   q[k]:=i;
   etiq[i]:=0
  End;
  if k=0 then stage:=false;
  while k<>0 do
  Begin
   i:=q[1]; q[1]:=q[k]; k:=k-1;
   if expose[i]<>0 then begin augment(i);k:=0 end
   else
   Begin
    stage:=false;
    for j:=1 to nligne do
    if etiq[j]=0 then
    if aa[i,j]=1 then begin etiq[j]:=i;k:=k+1;q[k]:=j end;
   End;
  End;
 End;
End;
```

Cette procedure forme le graphe F à partir du couplage du graphe biparti;

```
Procedure Graphe_F;

Begin
 e:=nligne;
 if nnpur<>0 then begin
 for i:=1 to nnpur do X33[nligne+1,i]:=npur[i];
 colonne[nligne+1]:=nnpur;
 nligne:=nligne+1; end;
 for i:=1 to nligne do for j:=1 to nligne do FF[i,j]:=0;
 for l:=1 to e do
 Begin
  verif:=true;
  for i:=1 to nligne do
  if verif then
  for j:=1 to colonne[i] do
  if verif then
  if X33[i,j]=match1[l] then begin FF[l,i]:=1;verif:=false end;
 End;
End;
```

Cette procédure englobe les étapes 1 et 2 de l'algorithme2, elle commence par chercher les arbres de profondeur 1, les chemins de longueur 2 ainsi que les chemins de longueur 2, puis relie chaque composante trouvée suivant la méthode citée par les auteurs en utilisant la procédure correspondante.

```
Procedure  Tournee2;

 var       a,deg,suc,zero,cycle : vecteurT;
           na,nzero,compt,compt1,ncycle,v,m,s : integer;

 Procedure Relier(b:vecteurT;nb:integer);

  var      n1,n2 : integer;
           c1,tour1 : vecteurT;

  Begin
   if ntour=0 then
   Begin
    for i:=1 to nb do tour[i]:=b[i];
    ntour:=nb;
   End
   else
```

```
  Begin
    position(tour,ntour,n1);
    position(b,nb,n2);
    for l:=1 to nb do b[nb+l]:=b[l];
    for l:=1 to n1 do p[l]:=tour[l];
    for l:=1 to nb do p[n1+l]:=b[n2+l];
    for l:=1 to ntour-n1 do p[n1+nb+l]:=tour[n1+l];
    ntour:=ntour+nb;
    for l:=1 to ntour do tour[l]:=p[l];
  End;
 End;

Procedure Chemin_1(e,f:integer);

 var     cycle1,cycle2 : vecteurT;
         ncycle1,ncycle2 : integer;

 Begin
  ncycle1:=colonne[e];
  ncycle2:=colonne[f];
  for l:=1 to ncycle1 do cycle1[l]:=X33[e,l];
  for l:=1 to ncycle2 do cycle2[l]:=X33[f,l];
  verif:=false;
  for l:=1 to ncycle1 do
  if not verif then
  for h:=1 to ncycle2 do
  if not verif then
  Begin
   g:=distance(cycle1[l],cycle2[h]);
   if g=1 then begin i:=l;j:=h;verif:=true end;
  End;

  for l:=1 to i do cycle[l]:=cycle1[l];
  for l:=1 to (ncycle2-j+1) do cycle[i+l]:=cycle2[j+l-1];
  for l:=1 to j-1 do cycle[i+ncycle2-j+l]:=cycle2[l];
  for l:=1 to ncycle1-i do cycle[i+ncycle2+l]:=cycle1[i+l];
  ncycle:=ncycle1+ncycle2;

  cand[ncand+1]:=e;
  cand[ncand+2]:=f;
  ncand:=ncand+2;
  Relier(cycle,ncycle);
  deg[e]:=-1; deg[f]:=-1;

 End;
```

```
Procedure Chemin_2(e,f,g:integer);

 var      cycle1,cycle2,cycle3 : vecteurT;
          ncycle1,ncycle2,ncycle3,i1,i2,j1,j2 : integer;

Begin

 ncycle1:=colonne[e];
 ncycle2:=colonne[f];
 ncycle3:=colonne[g];
 for l:=1 to ncycle1 do cycle1[l]:=x33[e,l];
 for l:=1 to ncycle2 do cycle2[l]:=X33[f,l];
 for l:=1 to ncycle3 do cycle3[l]:=X33[g,l];
 verif:=false;
 for l:=1 to ncycle2 do
 if not verif then
 for h:=1 to ncycle1 do
 if not verif then
 Begin
  v:=distance(cycle2[l],cycle1[h]);
  if v=1 then begin verif:=true;i1:=l;i2:=h end;
 End;
 verif:=false;
 for l:=1 to ncycle2 do
 if not verif then
 for h:=1 to ncycle3 do
 if not verif then
 Begin
  v:=distance(cycle2[l],cycle3[h]);
  if v=1 then begin verif:=true;j1:=l;j2:=h end;
 End;
 if i1<=j1 then
 Begin
  for l:=1 to i2 do cycle[l]:=cycle1[l];
  for l:=1 to (j1-i1+1) do
  cycle[i2+l]:=cycle2[i1-1+l];
  for l:=1 to (ncycle3-j2+1) do
  cycle[i2+j1-i1+1+l]:=cycle3[l];
  for l:=1 to (j2-1) do
  cycle[ncycle3+i2-i1+j1-j2+2+l]:=cycle3[l];
  for l:=1 to (ncycle2-j1) do
  cycle[ncycle3+i2-i1+j1+1+l]:=cycle2[j1+l];
  for l:=1 to (i1-1) do
  cycle[ncycle3+ncycle2+i2-i1+1+l]:=cycle2[l];
  for l:=1 to (ncycle1-i2) do
  cycle[ncycle2+ncycle3+i2+l]:=cycle1[i2+l];
 End
```

```
else
Begin
 for l:=1 to j2 do cycle[l]:=cycle3[l];
 for l:=1 to (i1-j1+1) do
 cycle[j2+l]:=cycle2[j1-1+l];
 for l:=1 to (ncycle1-i2+1) do
 cycle[j2+i1-j1+1+l]:=cycle1[i2+l-1];
 for l:=1 to (i2-1) do
 cycle[ncycle1+j2-j1+i1-i2+2+l]:=cycle1[l];
 for l:=1 to (ncycle2-i1) do
 cycle[ncycle1+j2-j1+i1+1+l]:=cycle1[l];
 for l:=1 to (j1-1) do
 cycle[ncycle1+ncycle2+j2-j1+1+l]:=cycle2[l];
 for l:=1 to (ncycle3-j2) do
 cycle[ncycle2+ncycle1+j2+l]:=cycle3[j2+l];
End;
ncycle:=ncycle1+ncycle2+ncycle3;
cand[ncand+1]:=e;
cand[ncand+2]:=f;
cand[ncand+3]:=g;
ncand:=ncand+3;
Relier(cycle,ncycle);
deg[e]:=-1; deg[f]:=-1; deg[g]:=-1;

End;

Procedure Arbre(e:integer;b:vecteurT;nb:integer);

 var     racine,cycle1,cycle2,feuille : vecteurT;
         nracine,ncycle1,ncycle2 : integer;

        Procedure Fusion_1(l:integer);

         Begin

          f:=feuille[l];ncycle1:=colonne[f];
          for h:=1 to ncycle1 do cycle1[h]:=X33[f,h];

          f:=feuille[l+1];ncycle2:=colonne[f];
          for h:=1 to ncycle2 do cycle2[h]:=x33[f,h];

          verif:=true;
          for h:=1 to ncycle1 do
          if verif then
          if distance(cycle1[h],racine[l])=1 then
          begin i:=h;verif:=false end;
```

```
verif:=true;
for h:=1 to ncycle2 do
if verif then
if distance(cycle2[h],racine[l+1])=1 then
begin j:=h;verif:=false end;
for h:=1 to ncycle1 do cycle1[h+ncycle1]:=cycle1[h];
for h:=1 to ncycle2 do cycle2[h+ncycle2]:=cycle2[h];
cycle[ncycle+1]:=racine[l];
ncycle:=ncycle+1;
for h:=1 to ncycle1 do cycle[ncycle+h]:=cycle1[i+h-1];
ncycle:=ncycle+ncycle1;
for h:=1 to ncycle2 do cycle[ncycle+h]:=cycle2[j+h];
ncycle:=ncycle+ncycle2+1;
cycle[ncycle]:=racine[l+1];
End;

Procedure Fusion_2(l:integer);

Begin

f:=feuille[l];ncycle1:=colonne[f];
for h:=1 to ncycle1 do cycle1[h]:=X33[f,h];
verif:=true;
for h:=1 to ncycle1 do
if verif then
if distance(cycle1[h],racine[l])=1 then
begin i:=h;verif:=false end;

for h:=1 to ncycle1 do cycle1[h+ncycle1]:=cycle1[h];
cycle[ncycle+1]:=racine[l];
ncycle:=ncycle+1;
for h:=1 to ncycle1 do cycle[h+ncycle]:=cycle1[h+i-1];
ncycle:=ncycle+ncycle1+1;
cycle[ncycle]:=racine[l+1]

End;

Begin

ncycle:=0;
nracine:=colonne[e];
for i:=1 to nracine do racine[i]:=X33[e,i];
v:=1;
for l:=1 to nracine do
```

```
Begin
 verif:=true;
 for i:=1 to nb do if verif then
 Begin
  f:=b[i];
  if match1[f]=racine[l] then verif:=false;
 End;
 if not verif then feuille[l]:=f
          else begin feuille[l]:=0;v:=l end;
End;

for i:=1 to nracine do racine[i+nracine]:=racine[i];
for i:=1 to nracine do cycle[i]:=racine[v-1+i];
for i:=1 to nracine do racine[i]:=cycle[i];

for i:=1 to nracine do feuille[i+nracine]:=feuille[i];
for i:=1 to nracine do cycle[i]:=feuille[v-1+i];
for i:=1 to nracine do feuille[i]:=cycle[i];

m:=1;
while m<=nracine do
Begin
 if feuille[m]<>0 then
 Begin
  if feuille[m+1]<>0 then Fusion_1(m)
  else Fusion_2(m);
  m:=m+2
 End
 else
 Begin
  ncycle:=ncycle+1;
  cycle[ncycle]:=racine[m];
  m:=m+1
 End;
End;
for h:=1 to nb do cand[ncand+h]:=b[h];
ncand:=ncand+nb+1;
cand[ncand]:=e;
for h:=1 to nb do begin f:=b[h];deg[f]:=-1 end;
Relier(cycle,ncycle)

End;
```

```
Begin (*Programme principal de la procedure Tournee2 *)

compt:=nligne;
fin:=false;
for i:=1 to nligne do deg[i]:=0;

while not fin do

Begin

 ncand:=0;

 for j:=1 to nligne do
 Begin
 suc[j]:=0;
 if deg[j]>=0 then
 Begin
  e:=0;
  for i:=1 to nligne do e:=e+FF[i,j];
  deg[j]:=e
 End;
 End;
 for i:=1 to nligne do
 for j:=1 to nligne do
 if FF[i,j]=1 then suc[i]:=j;

 nzero:=0;
 for i:=1 to nligne do
 if deg[i]=0 then begin
                  nzero:=nzero+1;
                  zero[nzero]:=i
                 end;
 for s:=1 to nligne do
 if deg[s]=0 then
 Begin
  i:=s;
  j:=suc[i];
  k:=suc[j];
  if deg[j]=1 then
  Begin
   if deg[k]=1 then Chemin_2(i,j,k)
   else Chemin_1(i,j)
  End
  else
```

```
 Begin
  verif:=true;
  na:=0;
  for k:=1 to nligne do
  Begin
   if suc[k]=j then
   if deg[k]<>0 then verif:=false
   else begin na:=na+1;a[na]:=k end;
  End;
  if verif then Arbre(j,a,na);
 End;
End;

for i:=1 to nligne do
if deg[i]=0 then
if suc[i]=0 then
Begin
 for l:=1 to colonne[i] do cycle[l]:=X33[i,l];
 ncycle:=colonne[i];
 ncand:=ncand+1;
 cand[ncand]:=i;
 Relier(cycle,ncycle);
End;
for i:=1 to ncand do
Begin
 e:=cand[i];
 for j:=1 to nligne do
 Begin
  FF[e,j]:=0;
  FF[j,e]:=0;
 End;
End;

compt:=compt-ncand;
if compt=0 then fin:=true
else
if ncand=0 then
Begin
 verif:=true;
 for i:=1 to nligne do
 if verif then
 for j:=1 to nligne do
 if verif then
 if FF[i,j]=1 then if deg[i]<>0 then
```

```
        Begin
          verif:=false;
          FF[i,j]:=0
        End;
       End;
      End;

      for i:=1 to T do tour2[i]:=tour[i];

     End;

    Begin

     Matrice_Cycles;
     if ntour<T then
     Begin
      Graphe_biparti;
      Bipartite_matching;
      Graphe_F;
      Tournee2;
     End;
     Longueur(tour2,Z2)

    End;


    (*-------------------------------------------------------------*)
          Procedure Algorithme_Exact;
    (*-------------------------------------------------------------*)

Var  Zmin,ntour,b,a : integer;

    Procedure Permutation(var s1,k,ntour1 :integer);

     Var s2,ntour2,i,j,e :integer;

    Begin
     if not stop then
     Begin
      fin:=false;
      if ntour1=T then
      Begin
       a:=tour[T];b:=tour[1];
       s1:=s1+Distance(a,b);
       if s1<zmin then
```

```
          Begin
           Zmin:=s1;
           for i:=1 to T do y[i]:=tour[i];
          End;
          fin:=true;
          if zmin=T then stop:=true;
         End;
        if s1>=Zmin then fin:=true;
        if fin=false then
         Begin
          for j:=2 to T do
          Begin
           cd:=true;
           for l:=1 to ntour1 do if cd then
           if tour[l]=j then cd:=false ;
           if cd then
           Begin
            ntour2:=ntour1+1;
            tour[ntour2]:=j;
            a:=tour[ntour1]; b:=tour[ntour2];
            s2:=s1+Distance(a,b);
            e:=j;
            permutation(s2,e,ntour2);
           End;
          End;
         End;
        End;
       End;

Begin (* Programme principal de la procedure Algorithme_Exact*)

if Z1<=Z2 then Zmin:=Z1 else Zmin:=Z2;
if Zmin<>T then
Begin
 for i:=1 to T do y[i]:=i;
 stop:=false;
 h:= 1;
 Z:=0;
 tour[1]:=h;
 ntour:=1;
 permutation(Z,h,ntour);
End;

Z3:=Zmin;
for i:=1 to T do tour3[i]:=y[i];

End;
```

```
BEGIN       (*PROGRAMME PRINCIPAL*)

 Writeln('Quelle est la taille du probleme?');
 read(T);
 N:=(T*T-T) div 2;

 For i:=1 to N do
 Begin
  randomize;
  repeat e:=random(1000)-997 until e>0;
  c[i]:=e;write(c[i])
 End;

 DEUX_COUPLAGE;
 XDMATRICE;
 ALGORITHME_1;
 ALGORITHME_2;
 ALGORITHME_EXACT;
 WRITELN('Longueur du tour optimal :  ',Z3);
 WRITELN('Longueur du tour de algorithme 1 : ',Z1);
 WRITELN('Longueur du tour de algorithme 2 : ',Z2);

END.
```

61

BIBLIOGRAPHIE

- The traveling salesman problem with distances one and two.
C.H.Papadimitriou et Yannakakis (1993).

- The traveling salesman problem.
E.L.Lawer, J.K.Lenstra, A.H.G.Rinnooykan, D.B.Shmoys (Jhon Wiley and Sons, 1987).

- Combinatorial optimisation, algorithms and complexity.
C.H.Papadimitriou, K.Steiglitz (Prentice-Hall, INC, 1982).

- Combinatorial heuristic algorithms with FORTRAN.
H.T.Lau , (Springer-Verlag, 1986)

- Computers and intractability, a guide to the theory of NP-Completeness.
M.R.Garey, D.S.Johson (W.H.Freeman and company, 1979).

- Complexité algorithmique et problème de communication.
J.P.Barthélemy, G.Cohen, A.Lobstein (Masson, 1992).