Le problème du voyageur de commerce en informatique

Julien Baglio

TIPE 2003-2004 section MP
Thème : le développement durable

Le voyageur de commerce est, essentiellement, un problème d'optimisation de chemin : étant donné n villes, quel est le circuit passant par ces n villes une unique fois par ville qui soit le plus court ? Ce problème très célèbre intervient pratiquement par exemple pour l'optimisation des dépenses de cuivre lors de la création des cartes électroniques (et donc contribue aux économies de matières premières).

Ce problème a la particularité d'être impossible à résoudre de manière exacte en un temps raisonnable. Voici un tableau qui illustre bien le propos, en considérant qu'un trajet s'évalue en 1 microseconde (ce qui est déjà très rapide) :

Nombre de villes	**Nombre de possibilités**	**Temps de calcul**
5	12	12 microsecondes
10	181440	0,18 secondes
15	43 milliards	12 heures
20	60.10^{15}	1928 ans
25	310.10^{21}	9,8 milliards d'années

Si n est le nombre de villes, le nombre de possibilités est $(n-1)!/2$; la croissance est donc exponentielle. Dans la suite du TIPE, nous nous intéresserons à un problème à 250 villes publié sur Internet dans le cadre d'un défi, pour lequel des bonnes solutions ont déjà été trouvé (la meilleure

étant de 11.809). Il est alors nécessaire de s'intéresser à des méthodes d'approximations qui nous donnent un chemin relativement court en un temps raisonnable, le calcul exhaustif étant à ce stade impossible (il prendrait sans doute des milliards de milliards d'années).

1 Premières méthodes

Après avoir vu qu'un traitement exhaustif était impossible, l'étude va se porter sur des méthodes de résolution approchées, donnant un résultat raisonnable en un temps relativement court.

1.1 Définitions préliminaires

Pour la suite de l'étude, quelques définitions relatives à la théorie des graphes sont nécessaires.
- Un graphe non orienté G est un couple (S, A) où S est un ensemble fini (l'ensemble des sommets) et A l'ensemble des paires constituées de sommets (les arêtes). Pour orienter le graphe, on remplace A par une relation binaire sur S.

- Une chaîne d'un graphe non orienté G est une séquence de sommets de G

- Un graphe non orienté est connexe si chaque paire de sommet est reliée par une chaîne.

- Lorsque le graphe est orienté, il est dit fortement connexe si chaque sommet est accessible à partir d'un autre.

- Une chaîne $< v_i >_{1 \leq i \leq n}$ est un cycle si et seulement si ($i \geq 3$, $v_0 = v_n$ et $\forall i \neq j,\ v_i \neq v_j$).

- Un graphe non orienté acyclique est appelée forêt, un graphe non orienté connexe acyclique est appelé un arbre.

- Un cycle hamiltonien d'un graphe non orienté G est un cycle où chaque sommet de G n'est visité qu'une et une seule fois.

Il en ressort que le problème du voyageur de commerce est formellement la recherche du cycle hamiltonien minimisant la pondération du graphe, lorsque on associe à chaque sommet du graphe étudié une fonction de pondéation.

1.2 Le plus proche voisin

C'est une méthode très simple à mettre en oeuvre, et donnant de bons résultats lorsque le nombre de sommets est petit. Si ce nombre est trop grand, l'approximation devient moyenne, mais sert de base à l'élaboration de techniques plus élaborées. C'est pourquoi c'est une technique interessante en soi.
On construit la tournée (ie le cycle hamiltonien) de manière gloutonne, c'est-à-dire pas à pas. On part d'un sommet quelconque, puis on cherche le sommet le plus proche, et on y va ; à partir de ce sommet, on cherche encore le plus proche en excluant les sommets déjà parcourus, puis on y va et on réitère le procédé. Voici le code CAML qui gère cette procédure :

```
 exception Erreur;;

let get_nearest_neighbour m l = let mini = ref max_int
and indice = ref 0 and bidon = ref 0
in
for i = 0 to (Array.length l) -1 do
  if l.(i)=0 &&  m.(i) < !mini then begin
    bidon := 1;
    indice := i;
    mini := m.(i);
  end;
done;
if !bidon = 0 then raise Erreur else !indice;;

let rec greedy_approx1 g depart actuel deja_vu =
  deja_vu.(actuel) <- 1;
  try
    let k = get_nearest_neighbour g.(actuel) deja_vu in
    let tmp = greedy_approx1 g depart k deja_vu in
    (fst tmp +. g.(actuel).(k) , k::(snd tmp))
  with
      Erreur -> g.(actuel).(depart),[depart];;
```

La procédure "get_nearest_neighbour" sert à trouver le plus proche sommet d'un sommet courant, en excluant les sommets déjà vus ; ces derniers sont marqués d'un "1" dans un tableau qui contient tous les sommets

marqués , par 0 s'ils n'ont pas encore été visités et d'un 1 s'ils l'ont été. Enfin la deuxième procédure crée récursivement la tournée des plus proches voisins.

1.3 L'arbre couvrant minimal

Une autre technique complètement différente pour obtenir un premier résultat est la technique de l'arbre couvrant minimal, ou ACM. Dans le cadre du voyageur de commerce, les cycles que l'on construit sont pondérés par la fonction distance euclidienne (notée d). S'interesser à l'arbre couvrant minimal du graphe, c'est chercher l'arbre qui recouvre le graphe, et dont la distance totale est minimale.

Si le graphe est représenté par $G = (S, A)$, on recherche $T \subset A$ avec $\mathrm{d}(T) = \sum_{(u,v)\in T} \mathrm{d}(u, v)$ minimale ; T existe car l'ensemble des réels constitué des distances totales possibles pour tout sous-ensemble $X \subset A$ est fini (on peut donc en extraire un minimum).

Pour construire l'arbre, on utilise une stratégie dite gloutonne : à chaque étape de la contruction de l'arbre, on choisit la meilleure option possible et on construit ainsi l'arbre pas à pas.

On se donne donc E notre ensemble courant, qui est supposé à chaque instant inclus dans un arbre couvrant minimal T. A chaque instant, on détermine une arête sûre pour E à ajouter, c'est-à-dire telle que $E \cup \{(u, v)\}$ est encore inclus dans un ACM.

Pour cela on introduit la notion de coupure $(P, S - P)$, qui est une partition de S. (u, v) traverse la coupure si $u \in P$ et $v \in S - P$ (ou l'inverse). La coupure respecte $E \subset A$ si aucune arête de E ne traverse la coupure.

Théorème 1 *soit $G = (S, A)$ un graphe non orienté connexe ; soit $E \subset A$ inclus dans un ACM T et $(P, S - P)$ une coupure de G respectant E ; alors si (u, v) est une arête de poids minimal traversant la coupure, (u, v) est sûre pour E.*

Démonstration : soit $E \subset T$, et supposons que (u, v) ne soit pas dans T (sinon c'est fini).

L'arête (u, v), à laquelle on adjoint la chaîne p constituée des sommets de u à v, forme un cycle dans T. Or (u, v) traverse $(P, S - P)$ donc il existe (x, y) dans p qui traverse aussi $(P, S - P)$. La coupure respectant E, (x, y) ne peut être dans E. On supprime (x, y) de T qui se trouve séparé en deux composantes, qu'on réassemble en adjoignant (u, v) à la place ; on forme ainsi un arbre $T' = (T - \{(x, y)\}) \cup \{(u, v)\}$, qui couvre aussi G.

T' est aussi un ACM : on a $\mathrm{d}(T') = \mathrm{d}(T) - \mathrm{d}(x,y) + \mathrm{d}(u,v)$; or (u,v) est l'arrête de poids minimale traversant $(P, S-P)$ donc $\mathrm{d}(x,y) \geq \mathrm{d}(u,v)$ et $\mathrm{d}(T') \leq \mathrm{d}(T)$.

Mais on a T ACM donc $d(T) \leq d(T')$ donc nécessairement, $d(T) = d(T')$.

Enfin, (u,v) est alors bien une arête sûre pour E : on a $E \subset T'$ car $(x,y) \notin E$ et $E \subset T$; donc $E \cup \{(u,v)\} \subset T'$ qui est un ACM donc (u,v) sûre pour E.

Une fois l'ACM construit, on construit son parcours préfixe : on parcours l'ACM, en passant par tous les sommets deux fois, puis on supprime les doublons, et on obtient un cycle hamiltonien.

Théorème 2 *Le parcours préfixe de l'ACM est au pire deux fois la longueur du plus court chemin.*

Démonstration : soit M le plus court cycle hamiltonien d'un graphe G. Soit T un ACM de G.

On supprime une arête de M ; on obtient alors un arbre T' car M moins une arête est un graphe connexe acyclique.

T étant l'ACM de G, $\mathrm{d}(T) \leq \mathrm{d}(T') \leq \mathrm{d}(M)$ puisque T' est M moins une arête.

Alors en notant P le parcours préfixe de T, on a bien $\mathrm{d}(P) \leq 2\,\mathrm{d}(T) \leq 2\,\mathrm{d}(M)$.

2 Méthodes plus performantes

Il apparaît que le plus proche voisin (14,73) donne de meilleurs résultats que l'ACM (16,34) pour le défi ; nous allons le sélectionner comme base pour des méthodes plus performantes.

2.1 Optimisation locale 2-Opt

C'est une méthode d'optimisation d'un résultat déjà entre les mains de l'utilisateur, ici le résultat du plus proche voisin. Elle se base sur le fait que l'on raisonne ici dans un espace euclidien, où l'on a à disposition l'identité du parallélogramme. En vertu de celle-ci, décroiser les chemins qui se croisent revient à améliorer le parcours.

Cette méthode est locale, dans le sens suivant : si on note $Cycl$ l'ensemble des cycles hamiltoniens d'un graphe à n sommets, on peut définir une distance sur cet ensemble : si $(C1, C2) \in Cycl^2$ alors $\mathrm{d}(C1, C2) = k$ où $k \in \mathbb{N}$

désigne le nombre d'arêtes distinguant $C1$ de $C2$.

On vérifie que c'est bien une distance :
- $\mathrm{d}(C1, C1) = 0$ puisque rien ne distingue $C1$ de $C1$.
- $\mathrm{d}(C1, C2) = \mathrm{d}(C2, C1)$ de manière évidente, la définition étant symétrique en $C1$ et $C2$.
- $\mathrm{d}(C1, C3) \leq \mathrm{d}(C1, C2) + \mathrm{d}(C2, C3)$: si $\mathrm{d}(C1, C2) = k$ et $\mathrm{d}(C2, C3) = n$, quand on passe de $C1$ à $C3$ en passant par $C2$, lors du passage de $C2$ à $C3$ les n arêtes de différences sont au pire des arêtes non modifiées par le passage de $C1$ à $C2$; d'où $\mathrm{d}(C1, C3) \leq k + n$.

Lors du 2-Opt, on explore des boules de rayon 2 pour cette distance, puisque on décroise deux arêtes. Voici un extrait du code CAML :

```
let gain2 g p j n =
     g.(p.(n-1)).(p.(j)) +. g.(p.(0)).(p.(j+1))
-. ( g.(p.(n-1)).(p.(0)) +. g.(p.(j)).(p.(j+1)) );;

 let gain_1 = gain2 g tmp j n in
          if gain_1 < !meilleur_gain then
            (
              meilleur_gain := gain_1;
              meilleure_trans := tmp.(0), tmp.(j);
            )
```

la fonction "gain2" calcule la différence entre avant le décroisement et après le décroisement ; puis cette fonction est appelée sur tous les sommets parcourus pour déterminer si le décroisement éventuel est efficace, cette comparaison se réalisant avec la ligne "if then".

2-Opt est une méthode en $O(n^2)$ c'est-à-dire qu'elle va explorer n sommets à chaque itération, qui sont elles aussi au nombre de n. Ce temps de calcul, qui peut sembler lent, est pourtant très efficace : en 1 seconde, 2-Opt améliore grandement le résultat du plus proche voisin (12,09 obtenu à comparer à 14,73 et sachant que le meilleur à ce jour est 11,809).

Cela reste malgré tout local, et on obtient parfois un puit.

2.2 Le recuit simulé

Pour éviter le problème du puit, ou du moins l'atténuer, on utilise la méthode du recuit simulé, qui consiste en une succession de 2-Opt sur des tournés aléatoires, avec un pseudo-hasard qui tempère la modification. Cela permet de faire des sauts de puits, car le choix de la meilleure modification peut faire atteindre trop vite le puit local.

Cette méthode est assez efficace, et améliore les circuits ; elle nous permet de descendre en dessous des 12,09 obtenu par le 2-Opt initial. Il est toutefois bon de faire remarquer que le choix de la meilleure modification n'est pas toujours le meilleur des choix, et que cela est encore un problème ouvert dans notre étude ; quand décider de faire telle modification plutôt qu'une autre ? De la faire avant, ou après ?

Enfin voici quelques statistiques intéressantes qui relient les parcours avec recuits simulés et la proportion de parcours commun à l'ACM :

Longueur du parcours	**Pourcentage d'arêtes communes avec l'ACM**
12,102	72 %
11,989	73,2 %
11,952	75,2 %
11,933	75,6 %
11,909	75,2 %
11,891	74,4 %
11,882	74,8 %
11,809	73,2 %

La dernière statistique, en rouge, concerne le meilleur résultat connu à ce jour, que l'étude n'a pas fourni de manière expérimentale ; il sert de référence pour évaluer les résultats fournis par le programme ici développé.

3 Conclusion

La méthode du recuit simulé fournit des solutions variées, où il est remarquable de constater que bon nombre de parties des cycles sont communes. Certaines parties non communes sont les meilleures des tournées produites, ce qui amène à l'idée de sélection des "bonnes parties" afin de les garder et de fusionner avec les parties communes.

C'est un travail assez difficile mais qui conduit à notre meilleur résultat, de 11,82, à moins de 0,1 % du meilleur résultat connu, et cela en dix minutes ce qui est rapide. La fusion est une méthode intéressante qu'il pourrait être bon de creuser pour améliorer encore les résultats.

Bibliographie

[1] A. Aupetit. Défi à 250 villes du PVC. 2000.
URL: *http ://home.alex.tuxfamily.org/pvc/defi/*

[2] Th. Cormen, Ch. Leiserson, et R. Rivest. *Introduction à l'algorithmique*. Dunod, 1997, ch. 5 et 24, pp. 84–89 et 489–499.

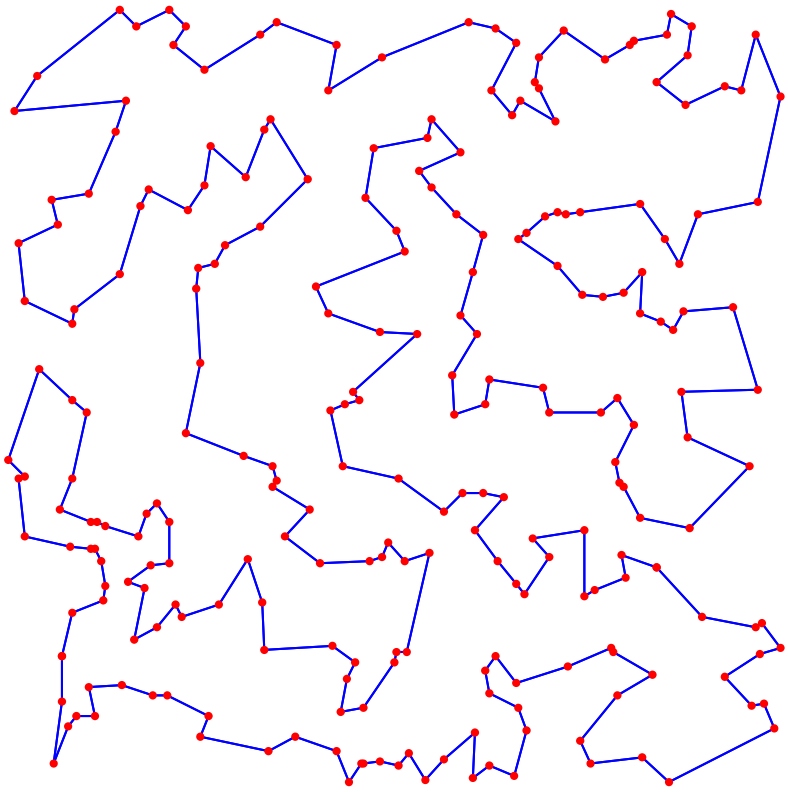

Fig. 1: Le meilleur résultat connu à ce jour ; distance de 11,809

Le meilleur résultat recensé sur le site du défi des 250 villes [1] ; l'étude présentée dans ce rapport s'est approchée à moins de 0,1 % de ce résultat mais n'a pas réussi à l'atteindre.

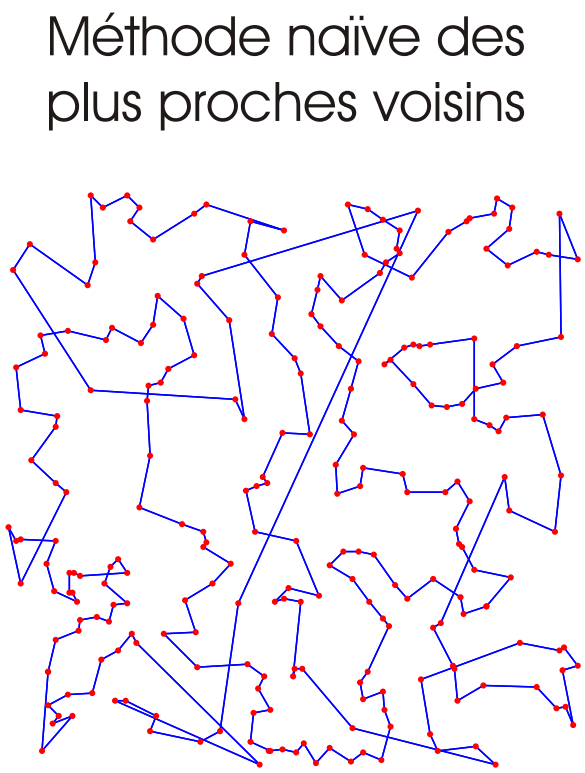

Fig. 2: Première méthode des plus proches voisins ; distance de 14,73

C'est la première des méthodes employées ; visuellement il est évident que le parcours proposé est loin d'être parfait.

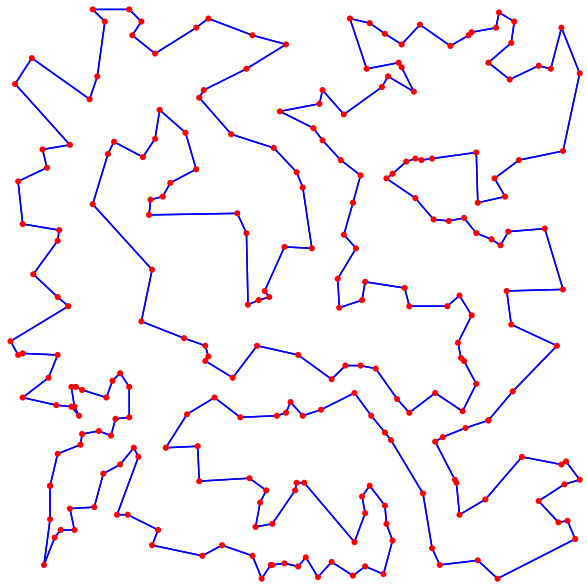

Fig. 3: L'optimisation 2-Opt du précédent ; distance de 12,08

L'optimisation locale du 2-Opt a supprimé tous les croisements générés par les plus proches voisins, et fournit un bon résultat.

Un arbre couvrant minimal

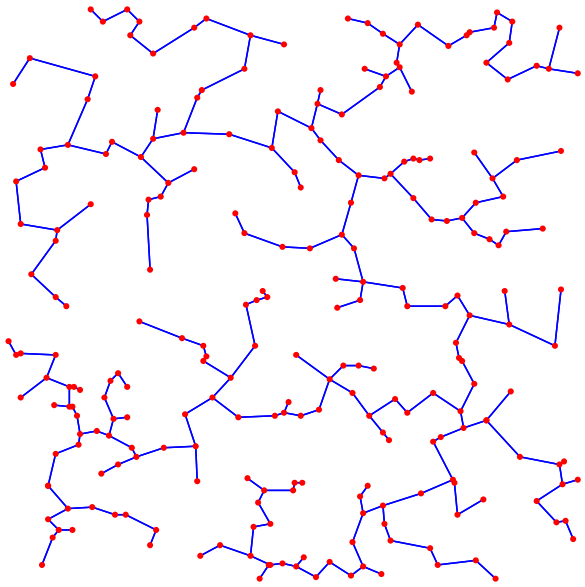

Fig. 4: Un arbre couvrant du défi ; longueur de 10,40

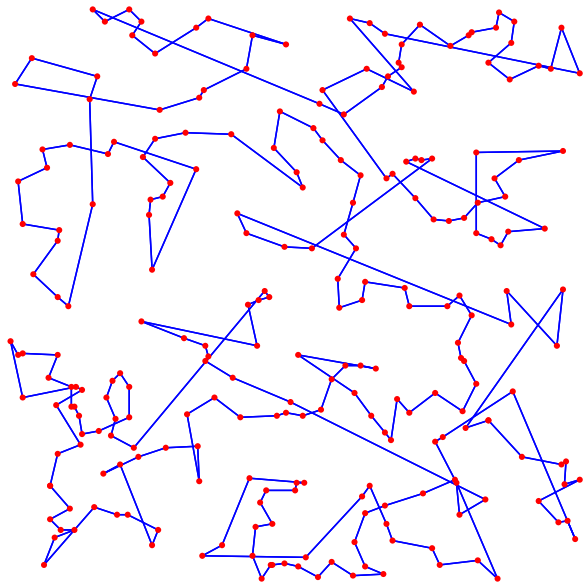

Fig. 5: Le parcours préfixe de l'arbre couvrant minimal ; distance de 16,34

Le parcours préfixe de l'arbre couvrant minimal, pour le transformer en cycle hamiltonien. Le résultat est plutôt médiocre par rapport à la méthode des plus proches voisins.

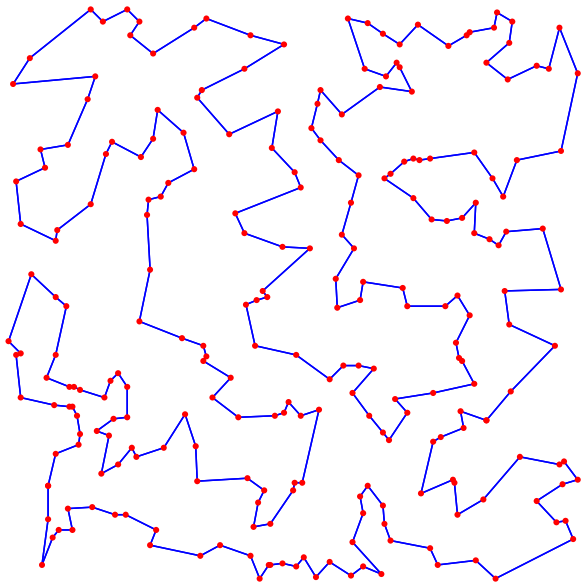

Fig. 6: Le meilleur résultat de l'étude ; distance de 11,82

Le meilleur résultat de l'étude, obtenu après recuit simulé et fusion de meilleures portions de trajets.