

Rapport intermédiaire du projet de comparaison de méthodes de résolution approché du T.S.P.

Dorian Dumez

Jocelin Cailloux

November 7, 2016

1 Heuristiques de constructions

1.1 N.N.H.

C'est une heuristique de construction gloutonne qui, partant d'une ville donné, va construire le cycle hamiltonien en choisissant comme prochaine ville celle qui se situe le plus proche de la vile actuelle.

On vois tout de suite l'impact du choix de la ville de départ sur la solution. Mais estimer la qualité de la solution en fonction de chaque ville est compliqué donc elle est choisie de manière arbitraire. En effet on prend toujours comme point de départ la première ville situé dans le tableau de notre distancier.

Notre distancier représentant le graphe sous forme d'une matrice d'adjacence on dispose toujours de la distance entre deux ville en temps constant. Cet algorithme va, pour chaque ville (on les parcours toutes une unique fois étant donne que l'on construit un cycle hamiltonien), parcourir toutes les villes non déjà parcourues pour trouver la plus proche. On en déduit que cet algorithme à une complexité temporelle en $O(nbVille^2)$. De plus sa complexité spatiale est en $O(nbVille)$, en effet on ne fait qu'allouer le parcours correspondant à la permutation identité avant de le modifier sur place.

1.2 R.G.S.C

2 Opérateurs de modification

2.1 2-opt

Cet opérateur consiste à sélectionner deux arc (i.e. deux paire de ville adjacente dans le parcours) et a tester si leur croisement fournit une meilleure solution. Si on pose iD, iF les extrémités du premier arc et jD, jF celle du deuxième arc, on entend par croiser les arc aller de iD à jD puis de iF à jF au lieu de iD à iF puis de jD à jF comme c'était le cas dans la solution. Graphiquement on observe que c'est en réalité l'inverse qui améliore la solution, en effet si les deux arc s'intersectent dans le plan (d'une carte où les villes sont positionnées dans un repère par exemple) alors les inverser fait disparaître cette intersection et réduit la taille du chemin parcouru.

Il est alors évident que les 4 villes (iD, iF, jD, jF) doivent être différentes car sinon on ne fait que modifier l'ordre de parcours de ces villes (on ne fait qu'inverser une permutation) mais le T.S.P. sur lequel on travaille est symétrique donc cela ne change rien. Il est aussi évident que l'on considère des paires d'arc et non des couples car cela reviendrait à tester deux fois le même changement.

Nous avons implémenté cet opérateur sous deux formes : en descente et en plus profonde descente.

En plus profonde descente on teste toutes les paires d'arc et on applique seulement celle qui conduit à la meilleure solution, ici celle dont le parcours est le plus court.

En descente on teste aussi toutes les paires d'arc mais cette fois-ci on applique la modification dès qu'elle améliore la solution. On ne s'arrête pas à la première solution trouvée pour éviter de s'appesantir sur le début du cycle et pour éviter de le re-tester un trop grand nombre de fois à partir du moment où il se serait stabilisé.

Dans les deux cas on va tester $O(nbVille^2)$ changements mais appliquer l'opérateur 2-opt s'effectue en $O(nbVille)$ car on doit inverser le sens de parcours des villes situées entre iF et jD. Donc l'algorithme de descente est en $O(nbVille^3)$ tandis que celui de plus profonde descente est en $O(nbVille^2)$. Mais cette complexité n'est que celle de `amelioresol2opt` et `amelioresol2optPPD`, donc ne prend pas en compte la vitesse de convergence de la solution vers un minimum local.

Ces complexités sont permises par le fait que l'on peut connaître à l'avance et en temps constant la taille du cycle hamiltonien après la modification.

2.2 3-opt

Cet opérateur est une généralisation du 2-opt avec 3 arcs. Tous les mouvements permis par le 2-opt le sont donc aussi par le 3-opt. Donc contrairement au 2-opt qui n'offre qu'une seule possibilité de modification, le 3-opt en offre 7 dont certaines ne peuvent pas être réalisées par une succession de 2-opt.

Cet opérateur a aussi été implémenté en descente et en plus profonde descente.

Pour les mêmes raisons que pour le 2-opt, dans l'algorithme de descente on teste toutes les triplets possibles à chaque fois au lieu de s'arrêter au premier mouvement améliorant trouvé.

Mais cette fois-ci je n'ai pas réussi à trouver une méthode me permettant de connaître à l'avance la longueur du cycle hamiltonien après la modification. Donc pour tester une modification la solution est appliquée puis la modification appliquée avant de calculer la longueur du cycle obtenu. Donc chaque test se fait en $O(nbVille)$.

La complexité du 3-opt est donc de $O(nbVille^4)$ car il y a $O(nbVille^3)$ ensembles de 3 arcs possibles et le test et la modification se font en $O(nbVille)$, la modification restant en $O(nbVille)$ car des inversions du sens de parcours sont nécessaires sur la plupart d'entre elles.

Et avec le 3-opt la plus profonde descente a la même complexité puisque on est obligé d'appliquer toutes les modifications pour les tester.

3 Algorithmes de recherche

Les deux opérateurs ayant une version normale et une en plus profonde descente tous les algorithmes suivants ont aussi deux versions. Ce sont toujours les mêmes, à la différence que l'une utilisera les opérateurs classiques et l'autre ceux en plus profonde descente.

3.1 Descente avec 2-opt

Cet algorithme part d'une solution que lui fournit une heuristique de construction puis l'améliore uniquement à l'aide de l'opérateur 2-opt. On utilise toujours le même opérateur sur la solution sur laquelle on travaille. On le fait jusqu'à tomber dans un minimum local, c'est à dire jusqu'à ce que 2-opt ne puisse plus améliorer la solution.

3.2 Descente avec 3-opt

Cet algorithme est identique au précédent mais avec l'opérateur 3-opt.

3.3 vnd

Cette fonction implémente un algorithme V.N.D utilisant les opérateurs 2 et 3 opt dans cet ordre. Elle part d'une solution fournie par une heuristique de construction. Elle utilise le critère d'arrêt standard du vnd qui est d'avoir trouvé un minimum local, c'est à dire que ni 2-opt ni 3-opt ne peuvent améliorer la solution.

3.4 vns

Cette fonction implémente un algorithme V.N.S utilisant les opérateur 2 et 3 opt dans cet ordre. Elle part d'une solution fournie par une heuristique de construction. Elle va s'arrêter quand elle aura utilisé, successivement, une fois la recherche de voisin puis l'amélioration, avec 2-opt puis avec 3-opt sans amélioration. Il faudrait donc mener de plus ample expérimentation concernant ce critère d'arrêt. Notamment en leur permettant chacun d'échouer plusieurs fois avant de passer à l'opérateur suivant ou en répétant ces 2 recherches plusieurs fois avant d'abandonner.

4 Expérimentation

4.1 En partant de N.N.H.

PPD designe la variante de l'algorithme utilisant la plus profonde descente.

Tous ce qui concerne le vns dans tableau 1 et 2 sont des moyennes effectuées sur 10 exécutions (sauf pour l'instance à 280 variables). Ces dernières sont données avec plus de détails dans le tableau 3.

nbVille	N.N.H.	2-opt	3-opt	2-opt PPD	3-opt PPD	vnd	vnd PPD	vns	vns PPD
48	281	225	230	223	225	225	223	220	223
52	19	6,8	5,9	4	4	6,3	4	5.8	3.9
130	24	10	12	6	5,7	10	5,7	6.1	5,7
150	25	3,8	5.4	1,4	1,4	3,8	1,4	4.7	1.4
280	23,3	10,7	7,5	7,5	7,5	10	7,5	7,5	7,5

Table 1: distance supplémentaire en pourcentage par rapport à la solution optimale

nbVille	N.N.H.	2-opt	3-opt	2-opt PPD	3-opt PPD	vnd	vnd PPD	vns	vns PPD
48	1.2e-05	0.00026	0.322	0.0007	1.576	0.161	0.161	0.41	0.603
52	1.4e-05	0.00023	0.454	0.00091	2.618	0.442	0.223	0.65	0.594
130	6.7e-05	0.0024	17.9	0.0117	202.2	18.7	18.3	32.5	112
150	8.9e-05	0.0026	37.9	0.014	433.9	27.6	27.8	86	57.6
280	0.00058	0.0172	584	0.084	7785	671	213	440	843

Table 2: temps d'exécutions en secondes

nbVille	variation temps vns	variation résultat vns	variation temps vns PPD	variation résultat vns PP
48	304	2.18	305	2.84e-14
52	201	4.5	357	3.7
130	302	4.9	202	3.1
150	590	6.3	596	7.4

Table 3: variation proportionnelle des algorithmes vns

4.2 En partant de R.G.S.C

bla