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

Dorian Dumez Jocelin Cailloux

November 8, 2016

1 Heuristiques de constructions

1.1 N.N.H.

Cet algorithme est une heuristique de construction gloutonne qui, partant d'une ville donnée, construit le cycle hamiltonien de proche en proche. A partir d'une ville d'origine, l'algorithme sélectionne comme prochaine ville celle qui est la plus proche de la ville courante.
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 située 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

Nous avons choisi d'implémenter une heuristique de construction de notre invention, en effet, nous voulions essayer quelque chose d'original. Nous avons donc imaginé un algorithme qui se base sur celui de Gale et Shapley qui résout le problème du mariage stable. Nous l'avons donc appelé ainsi, Recursive Gale-Shapley Circuit.
Nous avons pensé qu'il est pertinent de construire des couples de villes de telle manière qu'aucune puisse s'unir avec une autre ville sans augmenter la longueur totale des distances entre les couples. L'idée est donc de former de tels couples de manière récursive, une fois que les villes sont unies deux à deux, nous réitérons l'algorithme afin d'unir les couples précédement formés deux à deux de la même manière.
Chaque ville (puis chaque sous-couple par la suite), possède une liste de préférences des autres couples triés par ordre décroissant des distances. Nous avons supposé que la

complexité de cet algorithme serait du même ordre que celui de Gale-Shapley multiplié par log(n), log(n) étant le nombre d'itérations effecutées. L'algorithme de Gale-Shapley s'effectue en $O(n^2)$, nous avons donc pour notre algorithme RGSC une complexité de $O(log(nbVille^2))$.

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 êtres 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 reviendrai 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 toute les paires d'arc et on applique seulement celle qui conduit à la meilleure solution, ici celle dont le parcours est le plus cour.
En descente on teste aussi toutes le paires d'arc mais cette fois ci on applique la modification dès qu'elle amélioré la solution. On ne s'arrête pas à la première solution trouve 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 $O(nbVille^2)$ changement mais appliquer l'opérateur 2-opt s'effectue en $O(nbVille)$ car on doit inverser le sens de parcours des villes situé 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 complexité sont permise 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 arc. Tout 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 don certaines ne peuvent pas

être réalisé par une succession de 2-opt.

Cet opérateur a aussi été implémenté en descente et en plus profonde descente.
Pour les même raison que pour le 2-opt, dans l'algorithme de descente on teste toutes les triplet possible à chaque fois au lieu de s'arrêter au premier mouvement améliorant trouvé.
Mais cette fois ci je n'ai pas réussis 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 applique puis la modification appliqué 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 $O(nbVille^3)$ ensemble de 3 arc possible et le test et la modification se font de en $O(nbVille)$, la modification restant en $O(nbVille)$ car des inversions du sens de parcours sont nécessaire sur la plupart d'entre elles.
Et avec le 3-opt la plus profonde descente à 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 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.
Tout ce qui concerne le vns dans tableau 4 et 5 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 6.

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

Nous avons rencontré plus de difficultés que prévu lors de l'implémentation de cet algorithme. En dehors de l'aspect programmation et déboguage qui étaient particulièrement compliqué, nous avons remarqué que celui-ci ne converge pas pour l'itération de 280 villes que nous avons.
Nous n'avons pas encore déterminé s'il s'agît d'une erreur de programmation ou bien de l'algorithme qui ne converge pas. En effet, l'algorithme de Gale-Shapley permet de créer un mariage stable entre deux ensembles différents, ici, nous marions les éléments d'un ensemble avec des élements de ce même ensemble. Cette différence fondamentale qui ne semble pas avoir d'impact intuitivement, pourrait en effet empêcher la convergence de l'algorithme.
Nous ne présentons donc pas de statistiques avec cet algorithme pour la dernière instance, mais nous présentons les autres instances, pour lesquelles l'algorithme converge.

nbVille	R.G.S.C.	2-opt	3-opt	2-opt PPD	3-opt PPD	vnd	vnd PPD	vns	vns PPD
48	357	220	239	225	222	220	225	219	226
52	24	1,55	5,61	2,67	2,57	1,55	2,57	4,98	2,06
130	38,53	5,14	6,51	2,84	NaN	4,67	2,81	NaN	NaN
150	38,82	6,68	7,62	3,95	NaN	6,68	3,95	NaN	NaN

Table 4: distance supplémentaire en pourcentage par rapport à la solution optimale

nbVille	R.G.S.C.	2-opt	3-opt	2-opt PPD	3-opt PPD	vnd	vnd PPD	vns	vns PPD
48	0,0049	0,00025	0,277	0,00096	2,64	0,144	0,145	0,13	0,33
52	0,0069	0,00028	0,411	0,00067	1,98	0,20	0,20	0,55	0,59
130	0,086	0,0025	15,63	0,019	NaN	15,84	7,96	NaN	NaN
150	0,13	0,0034	27,94	0,018	NaN	13,97	13,18	NaN	NaN

Table 5: temps d'exécution en secondes

nbVille	variation temps vns	variation résultat vns	variation temps vns PPD	variation résultat vns PP
48	314	224	301	225
52	106	2,39	115	6,06

Table 6: variation proportionnelle des algorithmes vns

5 Conclusion

Bien que pertinente, nous remarquons que notre heuristique de construction R.G.S.C. donne globalement une solution moins bonne que celle du N.N.H. En effet, bien le

R.G.S.C. effectue plus de comparaisons celui-ci fournit des chemins plus long d'au moins 10%. De plus, celui-ci donne une solution de départ plus difficile à améliorer avec les algorithmes que nous avons implémentés. Pour certaines instances, l'amélioration n'a pas pu se terminer en un temps raisonnable et a été stopée.
Cet algorithme est donc moins bon qu'il n'en avait l'air. Cela dit, nous allons réfléchir à d'éventuelles améliorations basées sur le même principe.
La plupart des difficultés rencontrées se situe dans l'algorithmie. En effet, estimer en permanence la taille de la solution après modification par le 3-opt était une tâche plus difficile qu'il n'en parraîssait. Ou encore, les conditions d'arrêt du VNS. Ces deux aspects sont donc à travailler et nous espérons pouvoir les améliorer avant un prochain rendu de notre travail. Une étude des instances et peut être pertinente afin d'adapter nos algorithmes d'amélioration aux caractéristiques d'un chemin fournie par une heuristique de construction pour une instance donnée.
Ces aspects sont importants pour réduire l'écart de temps d'éxecution entre les algorithmes VNS et VND. En effet, cet écart est élevé et peut encore être amélioré d'après nos tests de performance.