Méthodes de résolution approchées pour le T.S.P.

Dorian Dumez Jocelin Cailloux

December 29, 2016

Contents

1 Introduction

Ce projet pédagogique, réalisé dans le cadre de l'unité d'enseignement des métaheuristiques, a pour objectif nous faire implémenter différentes méthodes de résolution approchée, de les combiner et de comparer les résultats. Ainsi, le but final est d'implémenter une méthode GRASP utilisant de la manière la plus judicieuse possible nos algorithmes de recherche locales. Pour réussir ce projet, nous devons mettre en pratique différents éléments de cours, réaliser beaucoup d'expérimentations et essayer différentes astuces. Certaines astuces ont été récupérées ou inspirées d'anciennes recherches réalisées par des chercheurs.

2 Heuristiques de constructions

2.1 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.

Nous remarquons 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, nous choisissons donc le point de départ de manière arbitraire. En effet, nous sélectionnons 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, nous disposons toujours de la distance entre deux ville en temps constant. Cet algorithme énumère pour chaque ville (nous parcourons chaque ville une et une seule fois étant donné la construction d'un cycle hamiltonien), toutes les villes non déjà parcourues pour trouver la plus proche. Nous déduisons que cet algorithme a une complexité temporelle en $O(nbVille^2)$. De plus sa complexité spatiale est en $O(nbVille)$, en effet, il n'alloue que le parcours correspondant à la permutation identité avant de le modifier sur place.

2.2 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 $O(log(nbVille) \times nbVille^2)$.

Après avoir implémenté la méthode, nous avons pu constater que celle-ci ne converge pas. En effet, pour certaines instances, quelques groupes ne pouvaient être reliés, ils perdaient leur liaison au dépit d'autres groupes. Ce phénomène ne survient pas dans

le cadre d'utilisation de l'algorithme du mariage stable (deux groupes distincts avec des unions entre les éléments de chaque groupe). Nous avons donc choisi de modifier cet algorithme pour le faire converger. Ainsi, une fois relié, un groupe n'est pas candidat pour être relié à un autre groupe plus proche. Seuls les groupes non reliés sont reliés avec le premier groupe voisin le plus proche qui n'est pas relié non plus ou qui est relié avec un autre couple moins favorable pour lui. Avec cette méthode, nous avons pu faire converger l'algorithme, de plus, nous obtenons de meilleurs résultats. Cependant, l'heuristique de construction NNH reste plus efficace.

3 Opérateurs de modification

3.1 2-opt

Cet opérateur consiste à sélectionner deux arcs (i.e. deux paires de villes adjacentes dans le parcours) et à tester si leur croisement fournit une meilleure solution. Si nous posons iD, iF les extrémités du premier arc et jD, jF celle du deuxième arc, nous entendons par croiser les arcs allers 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 nous observons que c'est en réalité l'inverse qui améliore la solution, en effet si les deux arcs 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, autrement, nous ne ferions que modifier l'ordre de parcours de ces villes (on ne fait qu'inverser une permutation) mais le T.S.P. sur lequel nous travaillons est symétrique donc cela ne change rien. Il est aussi évident que nous considèrons des paires d'arcs 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 nous testons toute les paires d'arc et nous appliquons seulement celle qui conduit à la meilleure solution, ici celle dont le parcours est le plus cour.

En descente nous testons aussi toutes le paires d'arc mais cette fois ci nous appliquons la modification dès qu'elle améliore la solution. L'algorithme 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 nous testons $O(nbVille^2)$ changement mais appliquer l'opérateur 2-opt s'effectue en $O(nbVille)$ car nous devons 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 grâce l'obtention en avance et en temps constant la taille du cycle hamiltonien après la modification.

3.2 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 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 nous testons tous les triplet possibles à chaque fois au plutôt que de s'arrêter au premier mouvement améliorant trouvé.

Dans les deux cas nous testons tous les triplets d'arcs possibles, il y en a $O(nbVille^3)$. De même que pour le 2-opt appliqué, une modification s'effectue en $O(nbVille)$ car des inversions du sens de parcours sont parfois nécessaires. Donc selon le même principe que pour le 2-opt l'opérateur 3-opt est en $O(nbVille^4)$ pour la version classique et en $O(nbVille^3)$ pour la plus profonde descente. Mais cela ne prend pas en considération la vitesse de convergence vers un minimum local.

4 Algorithmes de recherche locale

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.

4.1 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. Nous utilisons toujours le même opérateur sur la solution sur laquelle nous travaillons. Nous le faisons jusqu'à tomber dans un minimum local, c'est à dire jusqu'à que 2-opt ne puisse plus améliorer la solution.

4.2 Descente avec 3-opt

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

4.3 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.

4.4 VNS

Cette fonction implémente un algorithme V.N.S utilisant les opérateurs 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 répétant ces 2 recherches plusieurs fois avant d'abandonner.

5 Expérimentation

5.1 En partant de N.N.H.

PPD désigne la variante de l'algorithme utilisant la plus profonde descente.
Tout ce qui concerne le VNS dans tableau **??** et **??** sont des moyennes effectuées sur 10 exécutions. Ces dernières sont données avec plus de détails dans le tableau **??**.

nbVille	N.N.H.	2-opt	3-opt	2-opt PPD	3-opt PPD	VND	VND PPD	VNS	VNS PPD
48	281	225	218	223	223	218	223	219	222
52	19.0	6.80	5.96	3.96	3.40	2.56	3.41	4.23	3.41
130	23.9	10.3	4.01	5.98	1.77	4.57	4.62	3.59	4.61
150	25.5	3.88	3.39	1.45	1.31	1.85	0.80	3.38	0.801
280	23.3	10.7	6.86	7.63	3.04	6.39	6.64	5.70	6.62

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	1e-05	0.0002	0.0151	0.0004	0.0380	0.0116	0.0042	0.0101	0.0821
52	1e-05	0.0002	0.0165	0.0010	0.0573	0.0102	0.0110	0.0143	0.0193
130	5e-05	0.0020	0.2013	0.0043	1.4373	0.1155	0.2935	0.3854	0.6382
150	0.0001	0.0032	0.2481	0.0055	1.4498	0.1817	0.1885	0.4651	0.5097
280	0.0004	0.0122	2.0138	0.0351	19.013	1.2042	3.0444	3.695	5.318

Table 2: temps d'exécutions en secondes

nbVille	variation temps VNS	variation résultat VNS	variation temps VNS PPD	variation résultat VNS PP
48	398	1.57	167	0.750
52	286	7.62	67.7	2e-14
130	247	4.17	83.1	0.0305
150	75.3	5.92	44.5	4e-14
280	334	5.91	25.2	0.183

Table 3: variation proportionnelle des algorithmes VNS

De ces première expérimentation nous pouvons déduire que :

- 2-opt est très clairement le pus rapide, malheureusement cela se ressent sur la qualité de ses solutions.
- la version plus profonde descente augmente bien la qualité des solutions (surtout pour 2 et 3-opt seul) mais cela se ressent fortement sur le temps d'exécution.

- 3-opt seul ne semble présenter aucun intérêt car VND le surpasse en tout point. Mais il sera tout de même gardé pour la suite car il pourrai réagir différemment face à la diversité de solution proposé par GRASP.

- 3-opt PPD seul propose les solutions de meilleures qualité, malheureusement son utilisation est très compromise par le temps de calcul qu'il requiert. Au contraire 2-opt PPD propose de bonnes solutions, sans être aussi bonne, mais sont temps d'exécution est des plus compétitif.

- Quand a VND et VNS il propose des solutions de qualités, légèrement meilleures que celles de 2-opt PPD, mais son plus lent que ce dernier. Et entre eux ils sont difficilement comparable car la comparaison dépend des instances.

- VND est relativement stable du point de vue de la qualité des solutions mais ne l'est pas du tout selon le temps d'exécution.

- VNS est déjà nettement plus stable du coté des temps d'exécutions et renforce encore plus cette stabilité au regard de la qualité des solutions.

5.2 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	221	225	222	220	218	221	216
52	24.1	1.55	3.91	2.56	2.56	1.55	2.56	2.37	1.80
130	38.5	5.14	2.98	2.83	0.845	1.60	1.43	2.92	1.49
150	38.8	6.68	4.43	3,95	1.26	4.89	3.48	2.62	3.48

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

Nous remarquons alors que les différents opérateurs conservent le même rapport entre eux en partant de solutions construites par RGSC. Mais même après amélioration cet algorithme de construction est totalement dominé par NNH.

nbVille	R.G.S.C.	2opt	3opt	2opt PPD	3opt PPD	VND	VND PPD	VNS	VNS PPD
48	0.0132	0.0003	0.0132	0.0010	0.0650	0.0041	0.0119	0.0239	0.0844
52	0.0155	0.0003	0.0135	0.0006	0.0382	0.0048	0.0052	0.0091	0.0887
130	0.1389	0.0015	0.1169	0.0071	1.620	0.1194	0.2361	0.2992	0.5526
150	0.2207	0.0019	0.1716	0.0087	2.593	0.1735	0.1914	0.5171	0.5229

Table 5: temps d'exécution en secondes

nbVille	variation temps VNS	variation résultat VNS	variation temps VNS PPD	variation résultat VNS PP
48	219	2.81	106	0.548
52	302	3.69	105	2.53
130	109	2.42	36.8	0.641
150	177	5.43	63.3	7e-14

Table 6: variation proportionnelle des algorithmes VNS

5.3 Modification du critère d'arrêt de VNS

Dans cette section je détaille les expérimentations faites sur VNS et VNS-PPD. En effet dans les tableaux précédent nous nous arrêtons dès que 2-opt puis 3-opt n'ont pas amélioré successivement. Mais leurs laisser plusieurs essais. Donc nous effectuons la boucle 2-opt - 3-opt k fois, nous relèvons la moyenne de la différence proportionnelle de valeur et la moyenne de temps d'exécution. Toutes ces exécutions sont effectués en partant de solution construite avec NNH.

nbVille	VNS -1	VNS PPD -1	VNS -2	VNS PPD -2	VNS -3	VNS PPD -3
48	219	223	219	223	217	223
52	4.48	3.41	2.98	3.66	2.48	3.40
130	3.26	4.62	2.03	4.62	1.85	4.53
150	3.28	0.80	2.43	0.80	1.86	0.99
280	4.64	6.44	3.53	6.53	3.10	6.59

Table 7: modification du critère d'arrêt de VNS : valeurs

nbVille	VNS -1	VNS PPD -1	VNS -2	VNS PPD -2	VNS -3	VNS PPD -3
48	0.0177	0.0114	0.0281	0.0123	0.0412	0.0162
52	0.0167	0.0194	0.0322	0.0271	0.0484	0.0368
130	0.4620	0.6899	0.7016	0.8776	1.200	1.029
150	0.4204	0.5822	0.9710	0.7370	1.445	1.007
280	4.587	5.923	10.63	8.204	12.34	9.188

Table 8: modification du critère d'arrêt de VNS : temps

Premièrement cette modification dessert complètement la version plus profonde descente. La qualité de ses solutions n'est même pas forcement augmenté (logiquement elle devrai toujours l'être mais l'aléatoire fait qu'elle ne l'est même pas forcement) tellement

l'amélioration est faible. Par contre l'impact sur le temps d'exécution est très clairement visible.

Sur la version classique le bilan est plus mitigé. En effet cette modification viens grandement améliorer la qualité des solutions, surtout le passage de 1 à 2 tours. Mais l'opérateur part alors en rapidité : nous avons en moyenne un facteur 2 entre le temps d'exécution entre 1 et 2 tours.

6 GRASP

6.1 Construction des solutions

GRASP construit ses solutions avec un algorithme de NNH modifié pour include de l'aléatoire. NNH à été choisis car il propose de meilleurs résultats que RGSC, et nos recherches locales se comportent aussi mieux dessus. De plus RGSC ne convergent pas dans tous les cas il aurait dangereux de l'utiliser de manière aussi intensive.

La version classique de NNH part d'une ville, parcours toutes les autres pour trouver la plus proche, y va et recommence. Etant donné que pour chaque ville nous devons parcourir toutes celles qui n'ont pas encore été visité cet algorithme est en $O(nbVille^2)$.

Dans cette version l'algorithme prend en paramètre "le taux d'aléatoire" α. Nous parcourons toutes les villes en allant ensuite vers une ville proche. Pour décider vers quelle ville nous nous dirigeons, nous parcourons toutes les villes qui n'ont pas encore été visitées pour trouver la plus proche et la plus loin. A partir de cela et de α, nous déterminons la distance maximale acceptable. Nous nous dirigeons alors vers une ville choisie aléatoirement dans l'ensemble de celles qui ne se situent pas trop loin, selon cette distance critique. Même si cette version est un peu plus lourde elle reste en $O(nbVille^2)$.

6.2 Premières expérimentations

Nous expérimentons l'algorithme GRASP avec plusieurs algorithme de recherche locale. Le critère d'arrêt choisis est un nogood réglé empiriquement, et de manière constante, sur le nombre de ville divisé par 5.
Les expérimentations sont, comme toujours, faites sur 10 expérimentations. Mais quand ce temps était trop grand une seule expérimentation a été faite. Ces cas sont noté d'une étoile. Les exécutions vraiment trop longue ont été arrêtés et noté par un tiret.
VNS à été testé avec 1 et 2 tours, tandis que sa version plus profonde descente n'a été utilisé qu'avec 1 tours.

nbVille	2opt	3opt	2optPPD	3optPPD	VND	VNDPPD	VNS-1	VNS-2	VNSPPD
48	217	217	218	216	216	216	215	215	216
52	2.56	1.44	2.01	0.77	1.83	1.64	0.67	0.10	0.94
130	4.10	1.63	2.99	0.66*	1.67	1.38	1.13	0.74	1.57
150	1.85	1.71	1.52	0.75*	1.84	1.03	0.91	0.86[1]	1.02
280	6.32	3.10*	6.01	-	3.06*	-	1.99*	1.67*	3.51*

Table 9: différence de valeur de GRASP en fonction de la recherche locale

[1] la huitième exécution sur les 10 à été arrêté au bout de plusieurs minutes.

Nous remarquons alors que des recherches locales comme le 3-opt PPD ou le VNS-2 permettent d'obtenir des solutions d'une grande qualité. Malheureusement ces versions

nbVille	2opt	3opt	2optPPD	3optPPD	VND	VNDPPD	VNS-1	VNS-2	VNSPPD
48	0.0146	0.0917	0.0169	0.9165	0.0886	0.1262	0.1924	0.3626	0.3424
52	0.0161	0.1231	0.0201	1.511	0.0779	0.1326	0.3465	0.5166	0.3408
130	0.1590	5.880	0.8385	249*	6.660	14.9	17.46	38.5	30.8
150	0.2312	8.868	1.364	309*	6.82	30.9	33.68	39.23[1]	66.1
280	1.653	124*	17.02	-	95*	-	397*	509*	1103*

Table 10: temps de GRASP en fonction de la recherche locale

passent mal à l'échelle à cause de leurs temps d'exécution. Donc si l'on souhaite traiter efficacement de grosse instance le 2-opt est obligatoire.

Mais dans tous les cas nous remarquons que l'utilisation en multistart de ces différentes heuristiques permet d'obtenir des solutions de bien meilleure qualité que lorsque elles ne sont appelé qu'une seule fois.

6.3 Arrêt probabiliste

C.C. Ribeiro, I. Rosseti et R.C. Souza ont montré dans leur papier "effective probabilistic stopping rules for randomized metaheuristics: GRASP implementations" que la qualité d'une solution généré et amélioré par GRASP suit une loi normale. Il est donc possible d'utiliser cela comme condition d'arrêt de GRASP. Au regard des solution déjà généré nous pouvons estimer les paramètre de cette loi normale, et l'on décide de s'arrêter si la probabilité d'avoir une meilleure solution devient trop faible.

Une loi normale est totalement décrite par sa moyenne μ et son écart-type σ. Pour estimer μ nous avons choisis l'estimateur des moments d'ordre 1 (simple et efficace) : $\widehat{\mu} = \frac{1}{n}\sum_{k=1}^{n} X_k$. Pour estimer σ nous avons utilisé la variance empirique corrigé (l'estimateur des moments d'ordre 2 avec un facteur correctif pour qu'il soit sans biais) : $\widehat{\sigma}^2 = \frac{1}{n-1}\sum_{k=1}^{n}\left(X_k - \bar{X}_k\right)^2$.
Mais pour éviter de refaire des calculs et d'avoir à mémoriser toutes les valeurs obtenues nous avons utilisé une forme développé de cette dernière : $\widehat{\sigma}^2 = \frac{1}{n-1}\sum_{k=0}^{n}(X_k^2) - \frac{\bar{X}_k}{n-1}\sum_{k=1}^{n}(X_k)$ qui se calcule aisément avec des accumulateurs. En effet si note $sumX_k = \sum_{k=1}^{n} X_k$ et $sumX2_k = \sum_{k=1}^{n} X_k^2$ alors $\widehat{\sigma}^2 = \frac{sumX2_k}{n-1} - \frac{sumX_k^2}{n(n-1)}$. Mais pour que ces formules soient valides il faut que $n > 1$ donc nous forçons toujours le premier tour de boucle.

Si l'on souhaite s'arrêter dès que la probabilité de trouver une meilleure solution est inférieur à α il faut calculer cette dernière. La probabilité d'avoir une solution d'une taille de moins de x une est égal à $F_{N(\mu,\sigma^2)}(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{(x-\mu)^2}{2\sigma^2}} dx$. Or nous ne pouvons pas exprimer cela sans intégrale. Mais nous obtenons certaines valeurs approchées de $F_{N(0,1)}$

:

- $F_{N(0,1)}(-2.06) \leqslant 0.02$
- $F_{N(0,1)}(-1.65) \leqslant 0.05$
- $F_{N(0,1)}(-1.48) \leqslant 0.07$
- $F_{N(0,1)}(-1.29) \leqslant 0.1$

De plus nous savons que si $X \sim N(0,1)$ et que $Y = \mu + \sigma X$ alors $Y \sim N(\mu, \sigma^2)$ donc $F_{N(\mu,\sigma^2)}(y) = F_{N(0,1)}(\frac{y-\mu}{\sigma})$. Donc nous pouvons dire que si $y \leqslant \mu + \sigma F^{-1}_{N(0,1)}(\alpha)$ alors la probabilité d'avoir une meilleur solution est inférieure à α.

En pratique nous ne connaissons bien évidement ni μ ni σ donc nous utilisons $\widehat{\mu}$ et $\widehat{\sigma}$ décrits précédemment. Et pour les valeurs de α, nous les choisissons à l'avance pour fixer la valeur de $F^{-1}_{N(0,1)}(\alpha)$ à utiliser, en effet α n'appairaît plus.

6.4 Nouvelles expérimentations

Au vu de cette nouvelle idée il faut désormais refaire des tests. Mais nous nous restreignons cette fois aux heuristiques d'amélioration les plus prometteuses :

- 2-opt pour sa rapidité
- VNS-2 pour sa qualité de solution, sans être aussi lent que VNS-ppd
- VND comme solution intermédiaire qui pourrait se révéler intéressante

Notre algorithme GRASP commence toujours par construire une solution avec un NNH déterministe avant de l'améliorer avec VND. Le but étant d'avoir une bonne solution dès le début. Mais lorsque GRASP utilise l'opérateur 2-opt nous remarquons qu'une part importante de son temps d'exécution est due à cette première recherche. En effet VND avait été choisis pour cette tache car il produit de bonnes solutions mais il n'est pas trop lent (surtout qu'il n'est utilisé qu'une seule fois). Mais étant donné la vitesse de GRASP avec 2-opt nous remplaçons, dans les exécutions avec 2-opt, cette amélioration par une descente avec 2-opt.

nbVille	2%	5%	7%	10%	nogood
48	216	217	218	219	217
52	1.34	3.03	3.47	4.61	2.98
130	2.87	3.51	4.49	4.57	3.67
150	3.38	3.68	3.71	3.73	3.75
280	7.33	7.81	8.15	8.27	6.64

Table 11: différence de valeur de GRASP en fonction de α avec 2-opt

Pour VND et VNS-2 nous gardons l'initialisation avec VND, car VNS-2 et VND s'exécutent dans des temps semblables.

nbVille	2%	5%	7%	10%	nogood
48	0.1599	0.0105	0.0070	0.0051	0.0033
52	0.0250	0.0137	0.0046	0.0042	0.0041
130	0.2038	0.0594	0.0344	0.0278	0.0743
150	0.1390	0.0501	0.0523	0.0293	0.0727
280	0.4688	0.2335	0.1735	0.1181	0.8435

Table 12: temps de GRASP en fonction de α avec 2-opt

nbVille	2%	5%	7%	10%	nogood
48	-	216	216	217	216
52	0.47	0.95	1.28	1.87	1.83
130	0.61*	0.95	2.11	1.98	1.67
150	1.21*	1.81	1.83	1.85	1.84
280	2.90*	3.06*	3.06*	3.06*	3.06*

Table 13: différence de valeur de GRASP en fonction de α avec VND

nbVille	2%	5%	7%	10%	nogood
48	-	0.4871	0.2548	0.0920	0.0886
52	0.3417	0.1380	0.1140	0.0929	0.0779
130	132.5*	0.1395	2.902	2.679	6.660
150	116*	10.54	4.295	2.443	6.82
280	220.8	118*	109.2*	111*	95*

Table 14: temps de GRASP en fonction de α avec VND

La combinaison de l'opérateur 2-opt avec l'arret probabiliste produit de bon résultats. Nous remarquons en effet qu'avec $\alpha = 0.02$ nous obtenons généralement de meilleures solutions, donc nous contrebalançons le défaut de 2-opt, sans que cela ne prenne trop de temps, donc sans perdre son avantage.

Avec VNS de nombreux tests ont été effectué, mais ils sont tellement mauvais que cette possibilité a été abandonné. Et avec VND ils ne sont pas probant non plus. Donc nous pouvons désormais exclure cette combinaison : VND et VNS-2 offre de très belle qualité de solution mais fonctionne beaucoup mieux avec le critère nogood.
Et nous pourrions associer nogood au critère probabiliste pour éviter la non terminaison (si la variance est trop importante le critère d'arrêt deviens impossible car requiert des solutions de meilleures qualité que l'optimum). Mais cela ne pressente aucun intérêt car le problème surviens quand le critère deviens impossible mais la force de ce critère est d'accepter d'attendre un peu plus longtemps en attendent une bonne solution, donc il serai coupé par le nogood avant qu'elle n'apparaisse.

6.5 Stabilité des meilleures possibilités

Maintenant que l'on a dégagé des paramétrages de GRASP qui semble le mieux fonctionner :

- 2-opt partant de 2-opt et arrêté par critère probabiliste à 2%
- VND partant de VND et arrêté par nogood
- VNS-1 partant de VND et arrêté par nogood. Nous ne prenons pas VNS-2 car le critère d'arrêt probabiliste ne permet pas de réduire son temps d'exécution. Donc VNS-1 donne des solutions de légèrement moins bonne qualité mais en un temps plus raisonnable.

Pour conclure nous étudions la stabilité de la qualité des solution et du temps d'exécution, de ces algorithmes.

nbVille	2-opt	VND	VNS-1
48	20779	145.3	263.1
52	2270	143.0	165.7
130	1226	62.86	149.6
150	2257	59.68	75.76
280	1912	136.1	-

Table 15: Variation proportionnelle du temps d'exécution

nbVille	2-opt	VND	VNS-1
48	0.529	1.03	0.404
52	3.32	2.53	2.53
130	3.12	0.691	1.02
150	1.03	0.064	1.14
280	3.30	0.951	-

Table 16: Variation proportionnelle de la qualité des solutions

Nous regrettons donc le manque de stabilité du point de vue temporel des algorithme, surtout celui de 2-opt. Mais pour ce dernier cette instabilité n'est pas trop dérangeante car il reste rapide même dans ses pires exécution. Par contre, étant donné leurs gros temps d'exécution, cette instabilité est plus préjudiciable aux deux autres algorithmes.

Tous les algorithme sont par contre stable du point de vue de la qualité de solution même si 2-opt varie un peu plus cela reste petit, et cela deviens négligeable pour les deux autres.

7 Path-relinking

Nous avons expérimenté deux manières différentes d'implémenter un path-relinking. Le premier, le plus naturel, consiste à inverser l'emplacement de deux villes. De cette manière nous conservons l'admissibilité de la solution. Le second, plus arbitraire, consiste à remplacer certaines villes par une autre. Cette dernière solution recquiert une réparation régulière.

7.1 Mouvement swap

Ce path-relinking a pour voisinage toutes les permutations possibles de deux villes. Nous avons donc un voisinage en $O(N^2)$. Dans sa première version, l'algorithme parcourt la liste des permutations et l'effectue si celle-ci n'éloigne pas la solution courante de la solution objectif. Un tel sous-ensemble de voisinage permet de ne jamais s'éloigner de la fonction objectif tout en offrant une grande liberté de mouvement afin d'explorer de nombreuses solutions.

Pour la deuxième version de cet algorithme, nous nous sommes inspirés d'un travail de Mauricio Resende et de Paola Festa. Dans cette version, nous parcourons tous les voisins possibles et nous effectuons celui qui améliore le mieux la solution tout en nous approchant d'au moins une ville de la solution objectif.

De plus, comme nous listons toutes les permutations possibles, nous nous permettons d'enregistrer une solution sans effectuer le mouvement si la solution améliore la fonction objectif.

Après expérimentations, nous avons constaté que le path-relinking ralentit de manière excessive le GRASP. Utiliser comme voisinage la liste des permutations consécutives ne garantit pas la convergence de l'algorithme, nous avons donc décider d'accélerer la convergence de l'algorithme. En effet, la dernière version de notre algorithme n'effectue un mouvement que si celui-ci permet d'approcher la fonction objectif avec les deux villes. Ce choix a permis de beaucoup accélérer le path-relinking en affectant peu son efficacité.

Cet algorithme est donc en $O(N^2)$ sans prendre en compte la vitesse de convergence, nécessitant au maximum $N/2$ swaps, nous avons donc une complexité en $O(N^3/2)$, soit $O(N^3)$.

7.2 Mouvement insertion

Ce path-relinking a pour voisinage toutes les insertions possibles d'une ville à la place d'une autre de telle manière que la nouvelle ville à l'emplacement i est celle de la ville objectif à l'emplacement i. Nous avons donc un voisinage en $O(N)$. Ce type de mouvement ne garantit pas l'admissibilité de la solution, ainsi, un algorithme glouton replace les villes manquantes (ou doublées) lorsqu'il y a un certain nombre nb de villes absentes

de la solution (nb étant un paramètre de la fonction).

Nous avons aussi tenté une version dans laquelle nous choisissons l'insertion améliorant le plus la solution. Cet algorithme cet avéré peu efficace, même s'il réussit parfois à trouver une meilleure solution. Nous avons donc préféré le path-relinking avec swaps.

Cet algorithme a donc une complexité de $O(N)$ sans considérer la vitesse de convergence ni la réparation. Il y a au plus N itérations, nous obtenons donc une complexité de $O(N^2)$ plus la réparation dont le coût dépend du paramètre donné (une valeur plus élevée permet des réparations plus longues mais moins régulières et inversement). Le coût de la réparation étant en $O(N * nb)$ avec $nb < N/2$, la réparation n'affecte pas la complexité algorithmique totale.

7.3 Inclusion dans GRASP

Nous commençons par tester le path-relinking avec la version de GRASP qui pourrait le plus en bénéficier : 2-opt à arrêt probabiliste à 2%. En effet cette version est rapide mais elle souffre d'un manque de qualité de ses solutions. Donc un ralentissement peut être acceptable et de meilleures solutions corrigeraient son défaut.

nbVille	en plus profonde descente	en descente	sans path-relinking
48	215	215	216
52	1.33	1.09	1.34
130	2.80	2.87	2.87
150	3.34	3.34	3.38
280	7.33	7.16	7.33

Table 17: différence de valeur de 2-opt probabiliste en fonction du path-relinking

nbVille	en plus profonde descente	en descente	sans path-relinking
48	0.2546	0.1317	0.1599
52	0.0390	0.0369	0.0250
130	0.2683	0.2410	0.2038
150	0.3009	0.1947	0.1390
280	0.5504	0.6139	0.4688

Table 18: temps d'exécution de 2-opt probabiliste en fonction du path-relinking

Donc le path-relinking n'améliore que très peu la qualité des solutions mais il ne ralentit pratiquement l'algorithme.

Dans cet optique de payer un peu plus de temps pour avoir de meilleures solutions le test a été fait avec 2-opt en plus profonde descente, toujours en arrêt probabiliste à 2%. Dans cette configuration la meilleure solution initiale généré par NNH est amélioré par 2-opt PPD.

nbVille	en plus profonde descente	en descente	sans path-relinking
48	215	215	215
52	0.4947	0.5664	0.5664
130	2.16	2.15	2.22
150	1.27	1.28	1.29
280	6.00	6.00	6.00

Table 19: différence de valeur de 2-opt PPD probabiliste en fonction du path-relinking

nbVille	en plus profonde descente	en descente	sans path-relinking
48	0.3901	0.392	0.3715
52	0.1152	0.109	0.1143
130	2.105	2.595	2.649
150	1.506	1.285	1.311
280	16.47	16.47	16.55

Table 20: temps d'exécution de 2-opt PPD probabiliste en fonction du path-relinking

Nous avons donc une nouvelle configuration intéressante de GRASP : 2-opt PPD avec l'arrêt probabiliste à 2% et un path-relinking en plus profonde descente. En effet dans cette configuration l'algorithme se termine en un temps respectable et trouve des solutions de bonne qualité. Et nous noterons que le path-relinking permet de faire ressortir l'arrêt probabiliste car les bonnes solutions trouvé par ce dernier permettent d'arrêter plus vite l'algorithme. De ce fait le coût supplémentaire du path-relinking se compense tellement bien de cette manière que l'algorithme est parfois plus rapide avec.

L'ajout du path-relinking ralentit peu GRASP mais il le fait tout de même donc la possibilité de l'ajouter aux autres configuration à été rejeté car ces dernières sont déjà très lentes.

8 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, alors que le R.G.S.C. effectue plus de comparaisons, celui-ci fournit des chemins plus longs. 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 stoppée.
Cet algorithme est donc moins bon que nous l'avons espéré. Nous l'avons modifié afin qu'il converge systématiquement. Nous l'avons modifié en nous rapprochant plus d'un algorithme glouton et moins de l'algorithme du mariage stable.
La plupart des difficultés rencontrées se situent 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 aspects sont importants pour réduire l'écart de temps d'exécution entre les algorithmes VNS et VND.
Une étude sur l'impact du choix de la ville de départ pour le NNH peut-être intéressante pour obtenir de meilleures solutions de départ. De plus, la sélection aléatoire de la ville de départ dans le GRASP peut en augmenter la diversité.
Nous avons choisi deux types de voisinages différents pour les appliquer au path-relinking. Parmi ces deux voisinages, nous avons créé une variante en plus profonde descente et une variante gloutonne. Celle qui améliore le plus souvent la solution est le path-relinking en plus profonde descente avec énumération exhaustive des swaps de la solution courante.
Plusieurs configurations intéressantes son retenues pour le GRASP. Selon notre exigence de temps, nous obtenons de plus ou moins bonnes solutions. Ainsi, par ordre croissant de temps disponible, nous retenons les configurations suivantes :

- 2-opt à 2% path-relinking en descente et une solution initiale amélioré par par 2-opt
- 2-opt PPD à 2% path-relinking en plus profonde descente et une solution initiale amélioré par 2opt PPD
- VND avec nogood de nbVille/5
- VNS-2 avec nogood de nbVille/5, dans l'absolu nous pouvons y adjoindre le path-relinking pour tenter d'avoir de meilleures solutions. Mais nous renforçons alors encore plus son côté extrême.

Cet exercice nous a permis d'expérimenter différentes méthodes de recherche locale appliquées au TSP, de comparer leur efficacité et de prendre conscience de l'utilité de la rapidité des méthodes approchées lors de leur usage dans la méthode GRASP.

P.S. : pour l'instant le code de GRASP est réglé sur le 2-opt à 2%, ceux de VNS et VNS PPD sur leurs versions à 1 tours. Le main lancera les exécutions sur les différentes méthodes en partant de solutions construite par NNH, et l'instance est la berlin52. Les tests sont faits sur des moyennes de 10 exécutions pour les méthodes comprenant de l'aléatoire. Toutes les expériences sont reproductibles car le générateur aléatoire est initialisé avec une graine constante au début du main.