Les Cahiers du GERAD ISSN: 0711–2440

La recherche à voisinages variables

G. Caporossi
P. Hansen

G–2013–71

October 2013

La recherche à voisinages variables

Gilles Caporossi
Pierre Hansen

GERAD & HEC Montréal
Montréal (Québec) Canada, H3T 2A7

gilles.caporossi@gerad.ca
pierre.hansen@gerad.ca

October 2013

Les Cahiers du GERAD
G–2013–71

Résumé : La recherche à voisinages variables (RVV), ou *Variable Neighborhood Search (VNS)* en anglais est une métaheuristique dont l'invention est due à Nenad Mladenović et Pierre Hansen et développée au GERAD (Groupe d'Études et de Recherche en Analyse des Décisions, Montréal) à partir de 1997. Depuis cette période, la recherche à voisinages variables a connu divers développements et améliorations ainsi que de très nombreuses applications. Selon le Journal of Citation Reports, les travaux initiaux [N. Mladenović and P. Hansen. Variable neighborhood search. *Comput. Oper. Res.*, 24:1097–1100, 1997.] [P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European J. Oper. Res.*, 130:449–467, 2001.] sur la RVV ont été cités plus de 600 et 500 fois respectivement à ce jour (plus de 1700 et 1200 fois selon *Google Scholar*), ce qui indique l'intérêt pour la méthode tant au niveau des développements qu'elle a connu que de ses applications.

Le but de cet article n'est pas de faire un exposé exhaustif des variantes et des applications de la RVV, mais plutôt de poser, le plus clairement possible, ses bases afin d'en faciliter la mise en œuvre.

1 Introduction

La recherche à voisinages variables (RVV), ou *Variable Neighborhood Search (VNS)* en anglais est une métaheuristique dont l'invention est due à Nenad Mladenović et Pierre Hansen et développée au GERAD (Groupe d'Études et de Recherche en Analyse des Décisions, Montréal) à partir de 1997. Depuis cette période, la recherche à voisinages variables a connu divers développements et améliorations ainsi que de très nombreuses applications. Selon le Journal of Citation Reports, les travaux initiaux [15][9] sur la RVV ont été cités plus de 600 et 500 fois respectivement à ce jour (plus de 1700 et 1200 fois selon *Google Scholar*), ce qui indique l'intérêt pour la méthode tant au niveau des développements qu'elle a connu que de ses applications.

On trouve des applications de la RVV dans des domaines aussi variés que le *data mining*, la localisation, les communications, l'ordonnancement, les tournées de véhicules ou la théorie des graphes, par exemple. Le lecteur est invité à se référer à [12] pour une revue plus complète.

La RVV comporte plusieurs avantages : d'une part, elle permet généralement d'obtenir d'excellentes solutions en un temps raisonnable, ce qui est aussi le cas de la plupart des métaheuristiques modernes, mais elle est aussi très facile à mettre en œuvre. En effet, la RVV est basée sur une combinaison de méthodes très classiques en optimisation combinatoire ou en optimisation continue. En outre, très peu de paramètres (et parfois aucun) doivent être ajustés pour obtenir de bons résultats.

Le but de cet article n'est pas de faire un exposé exhaustif des variantes et des applications de la RVV, mais plutôt de poser, le plus clairement possible, ses bases afin d'en faciliter la mise en œuvre.

Les concepts clé seront exposés et illustrés par un exemple basé sur la recherche de graphes extrêmes. Ces illustrations sont fortement inspirées de l'optimisation telle que mise en œuvre dans la première version du logiciel AutoGraphiX [3] dédié à la recherche de conjectures en théorie des graphes. Le choix de cet exemple tient au fait que toutes les composantes de la RVV de base y sont utilisées, et qu'elles y sont intuitives.

2 Fonctionnement de l'algorithme

Comme d'autres métaheuristiques, la recherche à voisinages variables repose sur deux méthodes complémentaires : d'une part, la recherche locale et ses extensions qui visent à améliorer la solution courante, et d'autre part, les perturbations qui permettent d'élargir l'espace des solutions explorées. Ces deux principes sont généralement connus sous les noms d'intensification et de diversification. Dans le cas de la recherche à voisinages variables, ces principes sont combinées d'une manière intuitive et facile à mettre en œuvre.

2.1 Recherche locale

La **recherche locale** (RL), utilisée par un grand nombre de métaheuristiques consiste en des améliorations successives de la solution courante par l'entremise d'une transformation élémentaire, jusqu'à ce qu'aucune amélioration ne soit possible. La solution ainsi trouvée est appelée *optimum local* par rapport à la transformation utilisée.

Techniquement, la recherche locale consiste en une succession de transformations de la solution afin de l'améliorer à chaque fois. La solution courante S est remplacée par une meilleure solution $S' \in \mathcal{N}(S)$ dans son voisinage. Le processus s'arrête quand il n'est plus possible de trouver de solution améliorante dans le voisinage de S, tel que le décrit l'algorithme 1.

Si le voisinage $\mathcal{N}(S)$ est complètement exploré, la recherche locale garantit l'obtention d'un optimum local en fonction de la transformation utilisée. La solution S obtenue est donc telle qu'il n'existe aucune solution $S' \in \mathcal{N}(S)$ qui soit meilleure que S. Toutefois, cette notion d'optimum local reste relative à la transformation utilisée. Il est bien sûr possible qu'un optimum local pour une transformation ne soit pas un optimal local pour une autre transformation. Le choix des transformations et de leur mise en œuvre est donc une partie importante de la recherche à voisinages variables.

Algorithme 1: Algorithme RechercheLocale

Input : S
Input : $\mathcal{N}$
Posons $ameliore \leftarrow vrai$
tant que $ameliore = vrai$ **faire**
  $ameliore \leftarrow faux$
  pour tout $S' \in \mathcal{N}(S)$ **faire**
    si S' *meilleure que* S **alors**
      $S \leftarrow S'$
      $ameliore = vrai.$
retourner S

2.1.1 Modifier les voisinages

Une possibilité pour améliorer la qualité de la solution est d'envisager diverses transformations (donc divers voisinages) ou diverses manières de les utiliser.

La performance de la recherche locale, que ce soit par l'effort de calculs qu'elle requiert ou la qualité de la solution qu'elle permet d'obtenir, de sa mise en œuvre en terme de structure de données, dépend de la transformation utilisée et de la manière dont on choisit d'accepter une meilleure solution comme solution courante.

Globalement, la recherche locale peut se résumer à changer la solution courante pour une solution meilleure jusqu'à ce qu'on ne trouve plus de solution meilleure. Il est possible que le voisinage de la solution courante comporte plusieurs solutions améliorantes. Le choix de la meilleure d'entre elles semble intuitif, mais il implique une exploration totale du voisinage. L'effort de calculs demandé par la recherche de la meilleure solution du voisinage est peut-être trop important par rapport au gain qu'il permet. Pour cette raison, il est parfois meilleur de changer la solution courante pour la première solution améliorante rencontrée. Le lecteur peut se référer à [10] pour une analyse poussée dans le cas du problème du voyageur de commerce.

Outre la mise en œuvre d'une transformation au sein de la recherche locale, il est possible de travailler sur les transformations elles-mêmes. Notons t une transformation de la solution S, qui permet de construire un ensemble de nouvelles solutions $\mathcal{N}^t(S)$ à partir de S.

Pour un problème avec contraintes implicites ou explicites, étant donnée une transformation t, les solutions $S' \in \mathcal{N}^t(S)$ peuvent être toutes réalisables, toutes non réalisables ou parfois réalisables. Selon la nature de la transformation, il est parfois possible de prédire la réalisabilité des solutions dans le voisinage $\mathcal{N}(S)$ de S, et donc décider a priori de l'utiliser ou non.

Outre la préservation de la réalisabilité, les transformations ont d'autres caractéristiques qu'il convient d'analyser afin d'en évaluer la pertinence. En effet, chaque voisinage correspond à un ensemble de solutions et plus cet ensemble est important, plus on peut s'attendre que la recherche locale soit performante, dans le sens qu'elle permettra d'identifier des bonnes solutions. Par contre, l'exploration de ce voisinage sera plus long. Le voisinage idéal comporterait de bonnes solutions, afin d'améliorer la solution courante, mais il ne comporterait que peu de solutions mauvaises afin d'être rapide à explorer. Trouver un tel voisinage n'est malheureusement pas toujours facile, mais il est parfois possible d'utiliser l'apprentissage machine pour sélectionner les transformations les plus pertinentes durant l'optimisation [4].

2.1.2 La descente à voisinages variables

Pour tirer avantage des diverses transformations qui peuvent exister pour un problème donné, et de leurs particularités, il est possible d'adapter la recherche locale afin de ne pas utiliser seulement une transformation, mais une séquence de transformations différentes. C'est ainsi qu'est construite la descente à voisinages variables (DVV).

De même que la recherche locale explore les diverses manières d'appliquer la transformation choisie pour améliorer la solution courante, la descente à voisinages variables (DVV) explore une série de voisinages successivement. Considérons alors une liste $\mathcal{N}^t(S) \quad t = 1 \dots T$, où T est le nombre de transformations considérées.

En utilisant successivement tous les voisinages de la liste pour effectuer des recherches locales, la descente à voisinages variables ne s'arrête que lorsqu'aucun d'eux ne permet d'améliorer la solution. Un optimum local pour chacun des voisinages considérés est alors identifié.

La performance de la descente à voisinages variables dépend, bien sûr, des voisinages utilisés, mais aussi de l'ordre dans lequel ils sont considérés. Supposons deux transformations t_1 et t_2 à partir desquelles on peut construire deux voisinages. Si $\mathcal{N}^{t_1}(S) \in \mathcal{N}^{t_2}(S)$, utiliser le voisinage basé sur t_1 après le voisinage basé sur t_2 n'améliorera pas la solution. On pourrait en conclure que la recherche locale utilisant le voisinage $\mathcal{N}^{t_1}(S)$ est inutile, mais il faut aussi considérer l'effort que requiert l'exploration de ces deux voisinages.

Au début de la recherche locale, on a généralement des solutions de qualité médiocre, si l'exploration de $\mathcal{N}^{t_1}(S)$ est plus rapide que celle de $\mathcal{N}^{t_2}(S)$, mais qu'elle permet toutefois des améliorations importantes à la solution, l'utilisation de la séquence t_1 puis t_2 plutôt qu'une seule de ces transformations est justifiée. Si on en juge par la qualité de la solution obtenue, $\mathcal{N}^{t_1}(S)$ est peut-être moins performant que $\mathcal{N}^{t_2}(S)$, mais il n'est pas nécessairement moins efficace pour autant.

S'il existe des solutions de $\mathcal{N}^{t_1}(S)$ qui ne sont dans $\mathcal{N}^{t_2}(S)$, et réciproquement, alors l'utilisation de ces deux transformations est pleinement justifiée.

Pour améliorer la performance globale de l'algorithme, il est souvent judicieux d'utiliser les voisinages plus simples au début de la recherche, puis de ne recourir aux voisinages plus longs à explorer que si les premiers échouent.

Exploitant ce principe, la DVV consiste à effectuer séquentiellement une recherche locale avec chacune des transformations, jusqu'à ce qu'aucune d'elles ne permette d'améliorer la solution courante. La descente à voisinages variables est décrite par l'algorithme 2 et peut être assimilée à une *méta*-recherche locale.

Algorithme 2: Algorithme DVV

```
Input : S
Input : N^t,   t = 1...T
Posons ameliore ← vrai
tant que ameliore = vrai faire
    ameliore ← faux
    pour tout t = 1... T faire
        S' ← RechercheLocale(S, N^t)
        si S' meilleure que S alors
            S ← S'
            ameliore ← vrai.
retourner S
```

2.2 Diversification de la recherche

L'autre approche pour améliorer la qualité de la solution obtenue par recherche locale est de changer son point de départ.

2.2.1 Les recherches multiples

Une première technique consiste à multiplier les tentatives à partir de diverses solutions initiales aléatoires, et ne garder la meilleure solution obtenue, tel que décrit par l'algorithme 3. C'est ce que nous appelons *multistart* en anglais.

Algorithme 3: Algorithme Multistart

Input : S
Posons $S^* \leftarrow S$, la meilleure solution connue.
répéter
  Soit S une solution aléatoire.
  $S' \leftarrow RechercheLocale(S)$
  si S' *meilleur que* S^* **alors**
    $S^* \leftarrow S'$
jusqu'à *critère d'arrêt*;
retourner S^*

Si le problème admet un faible nombre d'optima locaux et que ceux-ci sont assez éloignés, le *multistart* fonctionnera assez bien. Malheureusement, dans la plupart des cas, le nombre d'optima locaux et leurs caractérisations font qu'il est peu probable que cette technique donne de bons résultats.

Pour illustrer cette difficulté, supposons deux problèmes d'optimisation à une seule variable. Les figures 1 et 2 représentent la valeur de la fonction objectif à minimiser en fonction de la variable x.

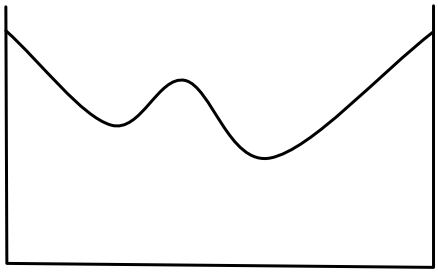


Figure 1: Illustration du premier problème à une variable.

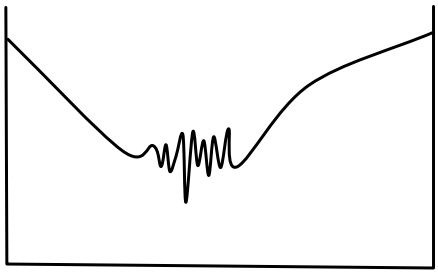


Figure 2: Illustration du second problème à une variable.

En prenant des valeurs initiales de x au hasard dans l'intervalle, les chances sont assez bonnes de trouver la solution optimale du problème 1, mais c'est beaucoup moins évident dans le cas du problème 2, d'une part parce que les optimum locaux sont plus nombreux, mais surtout parce que la plupart des solutions initiales mènent vers les mêmes optimums locaux (celui de droite et celui de gauche). Il est facile d'imaginer que l'augmentation de la dimension de l'espace dans lequel on recherche ne fera qu'amplifier ce problème, de sorte que le *multistart* est un très mauvais choix en général, surtout si les optimums locaux sont tous proches les uns des autres.

On peut commencer par se demander si notre problème est plutôt de la nature du problème 1 ou du problème 2. Bien qu'il n'y ait pas de règle générale à ce sujet, nous devons reconnaître que pour beaucoup de problèmes, les optimums locaux partagent un grand nombre de caractéristiques communes, ce qui laisse penser que le problème 2 est plus représentatif (bien qu'il faille toujours être prudent).

Prenons quelques exemples.

- Le problème du voyageur de commerce, problème dans lequel un voyageur de commerce doit rencontrer un certain nombre de clients et cherche l'ordre dans lequel il devra leur rendre visite s'il veut minimiser la distance totale parcourue. Il est peu probable, dans une bonne solution, que les deux clients les plus éloignés soit successifs. De même, deux clients très proches se succèderont sans doute. Globalement, les optimums locaux de ce problèmes comporteront un certain nombre de caractéristiques communes.
- La classification non supervisée vise à regrouper des objets de sorte que les objets qui se ressemblent soient dans la même classe (critère d'homogénéité) alors que des objets différents seront dans des classes différentes (critère de séparation). Il existe une multitude de critères pour évaluer la qualité d'une solution, mais dans tous les cas, nous observerons des faits similaires. Quel que soit l'optimum local considéré, il est plausible que deux objets très différents soient dans des classes différentes. À l'opposé, deux objets très similaires seront probablement toujours dans la même classe. Il semble donc que les optima locaux partagent des caractéristiques communes, et que la différence entre ces optima ne concerne qu'un nombre modéré d'objets. Là encore, les optimums locaux de ce problème comporteront un certain nombre de caractéristiques communes.

Il est bien sûr impossible de faire ici un inventaire complet des problèmes, mais l'hypothèse du problème 2, si elle ne peut être vérifiée, semble réaliste.

L'utilisation du *multistart* mènera alors aux optimums locaux les plus faciles à trouver, et la qualité globale de l'optimisation sera mauvaise.

2.2.2 Perturbations

Afin de réduire cet effort de calculs, plutôt que d'utiliser des solutions aléatoires comme point de départ des recherches locales, une autre approche consiste à modifier modérément la meilleure solution connue, ce que nous appelons *perturbation*.

Une méthode utilisant des recherches multiples à partir de perturbations se concentrera sur des solutions proches de la meilleure solution connue et profitera des caractéristiques de cette dernière (informations précieuses sur la caractérisation des bonnes solutions qui sont complètement ignorées par le *multistart*).

C'est pour cette raison que la recherche à voisinages variables ne procède pas par des recherches locales à partir de solutions aléatoires, mais à partir de solutions proches de la meilleure solution connue.

Le choix de l'amplitude de la perturbation à apporter à la meilleure solution avant de procéder à une nouvelle recherche locale est important. Si elle est trop faible, seul un étroit voisinage de la solution sera exploré (à l'extrême, on ne trouvera que l'optimum local courant). Si, au contraire, elle est trop importante, les caractéristiques de la meilleure solution courante seront ignorées et la perturbation ne sera pas mieux qu'une solution aléatoire. Pour palier ce problème, un paramètre k est utilisé qui caractérise l'amplitude de la perturbation à appliquer. Plus k sera élevé, plus la solution perturbée sera différente de la solution d'origine. Les voisinages utilisés pour les perturbations doivent donc avoir une magnitude reliée à la valeur k. Des voisinages imbriqués ou construits par une succession de transformations aléatoires sont en général appropriés tel que le montre l'algorithme 4. Une méthode simple mais efficace consiste à appliquer des transformations utilisées dans une recherche locale.

Algorithme 4: Algorithme PERTURBE

Input : S
Input : k
Input : $\mathcal{N}$
répéter k **fois**
  Choisir aléatoirement $S' \in \mathcal{N}(S)$,
  poser $S \leftarrow S'$.
retourner S

Si le problème comporte des contraintes (implicites ou explicites), il est conseillé que les transformations utilisées n'aient pas d'impact sur la réalisabilité de la solution. Par exemple, pour le problème du voyageur de commerce, une transformation qui provoque des sous-tours (solution consistant en plusieurs tournées disjointes) ne sera pas conseillée. Il est à noter aussi que ce schéma est donné à titre indicatif, il est possible pour certaines applications que la définition de voisinages imbriqués soit différente ou qu'une autre définition soit plus adaptée.

2.3 La recherche à voisinages variables

La recherche à voisinages variables fonctionne par une succession de recherches locales et de perturbations. Après chaque recherche locale infructueuse, l'amplitude de la perturbation k est augmentée pour permettre une exploration plus large. Au delà d'une valeur maximale k_{max} fixée comme paramètre, la valeur de k sera à nouveau réduite à son minimum pour éviter les inefficaces perturbations trop grandes.

Selon les applications, il conviendra de développer ou d'atrophier la recherche locale, ce qui donne lieu à diverses formulations de la recherche à voisinages variables. En effet, si la recherche locale permet d'améliorer la solution courante, elle est néanmoins exigeante en terme de calculs, et il y a un choix à faire entre la qualité de la solution et le temps nécessaire à l'obtenir. Le choix des transformations à utiliser lors de la recherche locale n'est donc pas anodin.

L'algorithme 5 décrit le fonctionnement de la recherche à voisinages variables de base.

Algorithme 5: Algorithme RVVB

Input : S
Noter $S^* = S$ la meilleure solution connue.
Poser $k = 1$
Définir k_{max}
répéter
  $S \leftarrow PERTURBE(S^*, k)$,
  $S' \leftarrow RechercheLocale(S')$.
  si S' *meilleure que* S^* **alors**
    $S^* \leftarrow S'$,
    $k \leftarrow 1$.
  sinon
    $k \leftarrow k + 1$.
    si $k > k_{max}$ **alors**
      poser $k \leftarrow 1$.
jusqu'à *critère d'arrêt*;
retourner S^*.

À partir de la structure que propose la recherche à voisinages variables, deux extensions sont envisageables. D'une part la recherche à voisinages variables générale favorise la qualité de la solution au détriment de l'effort de calculs, et au contraire, la recherche à voisinages variables réduite vise à réduire l'effort de calculs au détriment de la qualité de la solution.

2.3.1 La recherche à voisinages variables générale

Dans la recherche à voisinages variables générale, la recherche locale est remplacée par la descente à voisinages variables, cette dernière pouvant être considérée comme une méta-recherche locale. L'algorithme 6 décrit la recherche à voisinages variables générale.

Un des voisinages utilisés pour la descente à voisinages variables est généralement utilisé pour la perturbation.

Ce type de recherche à voisinages variables convient quand l'effort de calculs n'est pas crucial et que l'emphase doit être mis sur la qualité des solutions.

Algorithme 6: Algorithme RVVG

```
Input : S
Poser S* ← S la meilleure solution connue.
Poser k ← 1
Définir k_max
répéter
    S ← PERTURBE(S*, k),
    S' ← DVV(S').
    si S' meilleure que S* alors
        S* ← S',
        k = 1.
    sinon
        k ← k + 1.
        si k > k_max alors
            poser k ← 1.
jusqu'à critère d'arrêt;
retourner S*.
```

2.3.2 La recherche à voisinages variables réduite

À l'opposé de la descente à voisinages variables qui donne de meilleures solutions que la recherche locale de base, au coût de calculs plus intensifs, certaines applications vont nécessiter de réduire au maximum l'effort de calculs de la recherche locale. La recherche à voisinages variables peut fonctionner sans recherche locale, la succession de perturbations jouant à la fois le rôle de diversification et de recherche stochastique. Nous parlerons alors de recherche à voisinages variables réduite (la recherche locale étant éliminée).

L'algorithme 7 décrit la recherche à voisinages variables réduite.

Algorithme 7: Algorithme RVVR(S)

```
Input : S
Poser S* ← S la meilleure solution connue.
Poser k ← 1
Définir k_max
répéter
    S ← PERTURBE(S*, k),
    si S' meilleure que S* alors
        S* ← S',
        k ← 1.
    sinon
        k ← k + 1.
        si k > k_max alors
            poser k ← 1.
jusqu'à critère d'arrêt;
retourner S*.
```

3 Illustration et extensions

Afin d'illustrer le fonctionnement de la recherche à voisinages variables, nous allons en détailler le fonctionnement à partir d'exemples. Le premier exemple consiste à identifier des graphes extrêmes. Cet exemple est directement inspiré de l'algorithme de recherche à voisinages variables de la première version du logiciel AutoGraphiX [3].

Le second exemple est basé sur une extension possible d'un algorithme de classification non supervisée, k-means. Cet algorithme est considéré comme une référence pour la classification automatique et est très utilisé. Il se trouve que cet algorithme ne produit qu'un optimum local qui dépend fortement de la solution initiale utilisée. Nous proposerons ici une manière d'utiliser k-means au sein d'un algorithme de recherche à voisinages variables. Cette adaptation est simple, mais permet d'importantes améliorations de la performance de k-means.

Nous montrerons ensuite comment adapter la recherche à voisinages variables pour les cas d'optimisation de problèmes avec des variables continues.

3.1 Trouver des graphes extrêmes avec la RVV

Soit $G = (V, E)$ un graphe composé de $n = |V|$ sommets (*Vertices*) et $m = |E|$ arêtes (*Edges*). Un exemple de graphe avec $n = 6$ et $m = 7$ est représenté sur la figure 3. Bien que nous ne travaillons pas ici sur des graphes avec étiquettes (pour lesquels chaque sommet est caractérisé), les sommets sont numérotés de 1 à 6 afin de faciliter les descriptions.

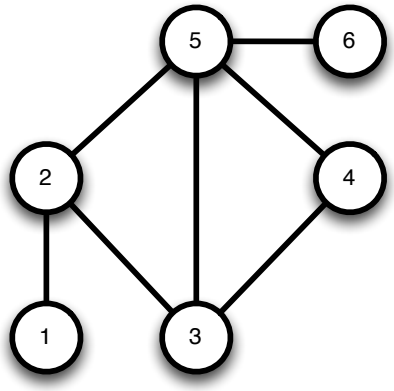


Figure 3: Un graphe G à $n = 6$ sommets et $m = 7$ arêtes.

Étant donné que la représentation du graphe n'a pas d'influence sur les calculs que nous effectuerons (la position des sommets n'importe pas, ce qui signifie que la distance mesurée entre des positions de deux sommets n'est pas importante), le même graphe peut tout aussi bien être représenté par sa matrice d'adjacence (voir figure 4) $A = \{a_{ij}\}$ avec $a_{ij} = 1$ si les sommets i et j sont adjacents, et $a_{ij} = 0$ si non, ou par une liste indiquant pour chaque sommet la liste des sommets qui lui sont adjacents. Le choix de la méthode de représentation (matrice d'adjacence, liste d'adjacence, ou autre) ou le choix de la numérotation des sommets sont purement arbitraires et n'ont aucun impact sur les calculs, l'objet étudié n'étant pas relié à sa représentation.

	1	2	3	4	5	6
1	0	1	0	0	0	0
2	1	0	1	0	1	0
3	0	1	0	1	1	0
4	0	0	1	0	1	0
5	0	1	1	1	0	1
6	0	0	0	0	1	0

Figure 4: Matrice d'adjacence du graphe G.

Notons $I(G)$ une fonction qui associe au graphe G une valeur indépendamment de la manière dont les sommets sont numérotés ou représentés, une telle fonction est appelée *invariant*. Le nombre de sommets n, ou d'arêtes m sont des invariants.

Pour donner d'autres exemples, citons le nombre chromatique $\chi(G)$, nombre minimum de couleurs requises pour associer à chaque sommet une couleur en s'assurant que deux sommets adjacents n'aient pas la même couleur. Ici, $\chi(G) = 3$, et une coloration pourrait consister à affecter le bleu aux sommets 1, 3 et 6, alors que le rouge serait affecté aux sommets 2 et 4. Le sommet 5 devra avoir une autre couleur, par exemple le vert. Toujours à titre d'exemple, nous pouvons ajouter l'énergie d'un graphe $E = \sum_{i=1}^{n} |\lambda_i|$, somme des valeurs absolues des valeurs propres de la matrice d'adjacence de G. Le nombre d'invariants graphiques est trop grand pour permettre de les énumérer ici, mais le lecteur peut se référer à [5] et [17] pour un recensement explicatif et relativement exhaustif des invariants qui existent.

La recherche de graphes extrêmes peut consister à trouver un graphe qui maximise ou minimise un invariant (ou une fonction d'invariants, qui est aussi un invariant), éventuellement avec des contraintes.

Les solutions de ces problèmes sont des graphes, chaque graphe différent étant une solution possible du problème. Les graphes ayant la meilleure valeur de la fonction objectif (plus grande ou plus petite selon que l'on maximise ou minimise) forment l'ensemble des solutions optimales. C'est un graphe parmi cet ensemble que nous chercherons à l'aide de la RVV.

3.1.1 Quelles transformations à utiliser ?

La recherche locale pourrait être définie à partir de l'ajout et le retrait d'une arête, mais on pourrait considérer beaucoup d'autres transformations telles que le déplacement d'une arête, ou d'autres transformations plus complexes. Puisque le voisinage d'un graphe change selon la transformation considérée, il est possible qu'une transformation ne permette pas d'amélioration de la solution courante, alors qu'une autre le permettrait. Les transformations utilisées dans la descente à voisinages variables de la première version d'AGX sont décrites dans [3] et sont présentées sur la figure 5. On constate par exemple que certains voisinages préservent le nombre de sommets, alors que d'autres le modifient. Étant donné que dans la plupart des cas, le nombre de sommets du graphe était fixé, certains de ces voisinages n'avaient pas d'intérêt.

On remarque aussi que la transformation $2 - Opt$ est la seule de la liste à préserver les degrés des sommets, elle sera donc la seule pertinente si on cherche des graphes extrêmes réguliers et que le graphe courant a déjà cette propriété. Les autres transformations peuvent alors permettre de trouver une première solution réalisable, mais leur utilité s'arrêtera là.

Toujours dans le cas précis où on se concentre sur les graphes réguliers, il sera peut-être pertinent d'inventer d'autres transformations spécifiques, telle que celle décrite sur la figure 6. Cette transformation, basée sur un sous-graphe à 5 sommets est exigeante en calculs mais peut être justifiée par sa spécificité.

3.1.2 Dans quel ordre utiliser les transformations ?

Considérons maintenant une transformation t_1 qui consiste en l'ajout ou le retrait d'une arête, et notons $\mathcal{N}^{t_1}(G)$ le voisinage de G correspondant. On constate que les graphes de $\mathcal{N}^{t_1}(G)$ ont une arête de plus ou de moins que G. Une autre transformation t_2 qui consiste en le déplacement d'une arête définira un voisinage (notons le $\mathcal{N}^{t_2}(G)$) constitué de graphes comportant le même nombre d'arêtes que G. Il est clair que ces deux voisinages sont exclusifs, un graphe ne pouvant appartenir aux deux.

On peut maintenant envisager une troisième transformation t_3 qui consiste à appliquer deux fois la transformation t_1. À partit de t_3, on peut construire un voisinage ($\mathcal{N}^{t_3}(G)$) composé des graphes obtenus à partir de G par deux ajouts, deux suppressions, ou encore un ajout et une suppression. Comme un déplacement peut être défini comme un ajout suivi d'une suppression, il s'avère que $\mathcal{N}^{t_2}(G) \in \mathcal{N}^{t_3}(G)$. Le voisinage $\mathcal{N}^{t_2}(G)$ sera inutile après le voisinage $\mathcal{N}^{t_3}(G)$ et on pourrait penser qu'il vaut mieux utiliser $\mathcal{N}^{t_3}(G)$ que $\mathcal{N}^{t_2}(G)$. Ce n'est pourtant pas certain. Il est vrai que $\mathcal{N}^{t_3}(G)$ permet d'accéder à des graphes inaccessibles à l'aide de $\mathcal{N}^{t_2}(G)$, bien que la réciproque ne soit pas vraie, mais l'exploration de $\mathcal{N}^{t_3}(G)$ est plus

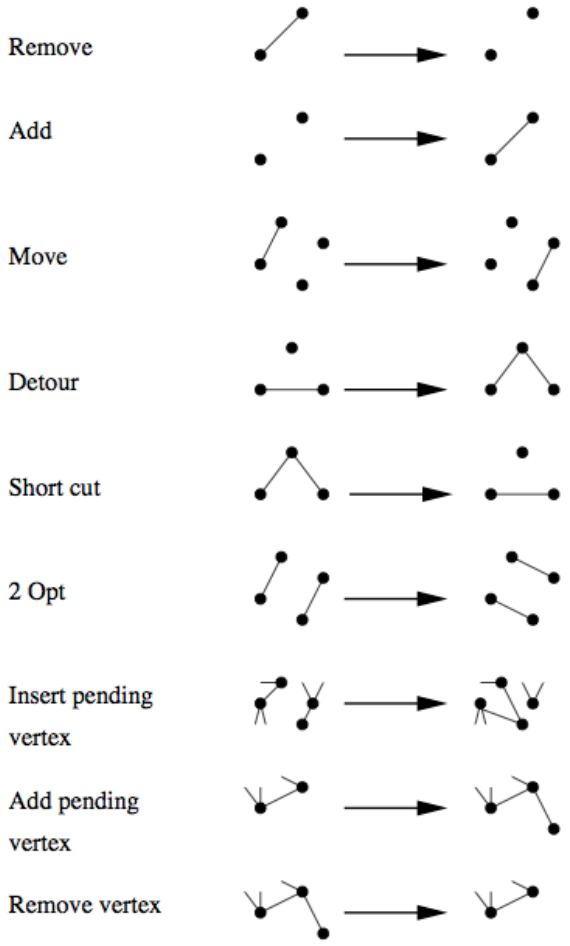


Figure 5: Voisinages utilisés dans la version initiale d'AGX.

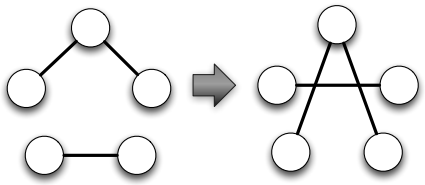


Figure 6: Un exemple de transformation spécifique aux graphes réguliers.

longue. Sans tenir compte des isomorphismes (dont l'identification est extrêmement difficile), si G comporte n sommets et m arêtes, nous avons :

- $|\mathcal{N}^{t_1}(G)| = \frac{n(n-1)}{2}$,
- $|\mathcal{N}^{t_2}(G)| = m(\frac{n(n-1)}{2} - m)$, et
- $|\mathcal{N}^{t_3}(G)| = (\frac{n(n-1)}{2})^2$.

Le temps passé à explorer un voisinage aussi vaste que $\mathcal{N}^{t_3}(G)$ n'est peut-être pas justifié. Au début de l'optimisation, alors que la solution courante n'est pas bonne, les voisinages simples et rapides à explorer semblent appropriés car ils permettent une amélioration rapide de la solution. À la fin, alors que l'exploration

se fait parmi de bonnes solutions, il est important de fouiller plus en détail l'espace des solutions. C'est dans ce contexte que les voisinages complexes sont justifiés.

3.1.3 Utilisation de la RVV de base

Pour illustrer le fonctionnement de la RVV de base, prenons le graphe G décrit sur la figure 3 comme solution initiale. Supposons que le problème consiste à identifier un graphe connexe à 6 sommets qui minimise l'énergie, et que la recherche locale soit basée sur la transformation t qui consiste à ajouter ou supprimer une arête.

La valeur de la fonction objectif $E(G) = 7,6655$ pour cette solution initiale.

Nous remarquons d'abord que, comme le problème étudié est restreint aux graphes connexes, certaines arêtes ne peuvent pas être retirées de G (par exemple l'arête entre les sommets 1 et 2, ou celle entre les sommets 5 et 6).

L'ensemble des graphes du voisinage $\mathcal{N}(G)$ de G (représenté sur la figure 3) par l'utilisation de la transformation t est représenté sur la figure 7.

En raison des isomorphismes, il est en général possible qu'un de ces graphes puisse être obtenu de diverses manières à partir du même graphe en appliquant une transformation donnée t, mais ce n'est le cas pour aucun des graphes de $\mathcal{N}(G)$ représentés sur la figure 7, puisque l'énergie, indiquée en dessous du graphe est toujours différente.

À ce stade, la meilleure solution connue est G, nous posons donc $G^* = G$ et $k = 1$.

En comparant les valeurs de l'énergie des graphes de $\mathcal{N}(G)$ à la valeur $E(G) = 7,6655$ du graphe initial G, on constate que G^1, G^2, G^3, G^4 et G^5 sont des solutions meilleures que G. La recherche locale pourrait donc continuer avec n'importe laquelle de ces solutions. Si on utilise le critère du meilleur d'abord, on choisira G^3. À l'itération suivante, nous aurons $G_1 = G^3$ comme solution courante à l'itération 1, et la valeur de la fonction objectif sera $E(G_1) = E(G^3) = 6,4852$.

En répétant le processus, lors de l'itération suivante (itération 2), nous obtenons un graphe G_2 dont l'énergie est $E = 5,81863$. Un examen du voisinage de G_2, $\mathcal{N}(G_2)$, nous remarquons qu'aucun de ces graphe n'est meilleur que G_2 qui est donc un optimum local pour l'ajout ou le retrait d'arêtes. Comme c'est alors la meilleure solution connue, nous notons $G^* = G_2$, $k = 1$ et la valeur de la fonction objectif $Z^* = 5,81862$.

Dans la suite de l'algorithme, nous allons alors procéder à une perturbation de la meilleure solution courante. Étant donné que $k = 1$, cette perturbation peut consister à ajouter ou enlever une arête au hasard. Après cette perturbation, nous effectuerons à nouveau une recherche locale. Si cette recherche locale échoue, nous augmenterons la valeur de k de 1 avant de faire une nouvelle perturbation. Cette fois, nous ajouterons ou enlèverons une arête au hasard et répèterons cette perturbation k fois avant d'effectuer la recherche locale suivante.

3.1.4 La descente à voisinages variables

Si on avait opté pour la recherche à voisinages variables généralisée, au lieu d'entrer dans la phase de perturbations juste après la recherche locale basée sur les ajouts/suppressions d'arêtes, nous aurions essayé d'autres transformations avant.

Dans le cas qui nous intéresse, nous aurions pu essayer de déplacer des arêtes en prenant le graphe G^3 comme point de départ. On remarque alors que déplacer l'arête entre les sommets 1 et 2 pour la placer entre les sommets 1 et 5 permettait d'améliorer la solution à nouveau. Dans ce cas-ci, la solution obtenue alors est une étoile à 6 sommets et son énergie est $E = 4,47214$. C'est la solution optimale puisque nous savons que $E \geq 2\sqrt{m}$, et que $m \geq n - 1$ pour les graphes connexes [2].

3.1.5 Perturbations

Comme nous le faisons souvent avec la recherche à voisinages variables, les perturbations peuvent être construites à partir des transformations utilisées pour la descente à voisinages variables. Par exemple, une

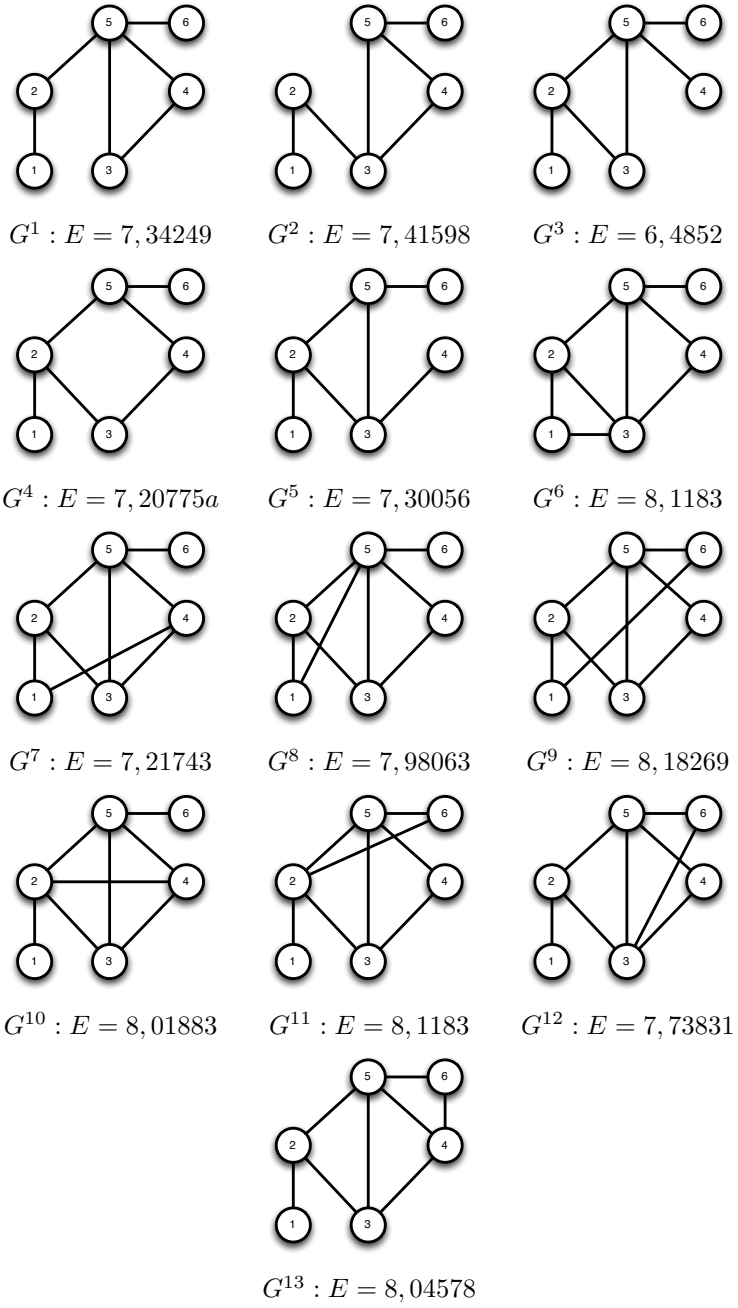


Figure 7: Graphes du voisinage $N^t(G)$

perturbation construite à partir de la transformation qui consiste à ajouter ou enlever une arête pourrait être construite comme indiqué sur la figure 8.

Si le nombre d'arêtes avait été fixé, il est probable que la perturbation basée sur des ajouts et suppressions ne donne pas de bons résultats. On utiliserait plutôt une transformation qui préserve le nombre d'arêtes comme $MOVE$ qui déplace une arête, tel que décrit l'algorithme 9.

Si les degrés avaient été fixés, nous aurions par exemple choisi *2-opt* comme base à la perturbation, etc.

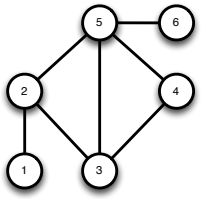
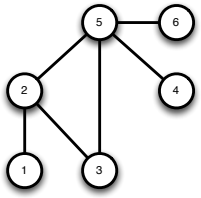
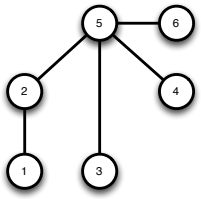
Itération	Graphe	Énergie
0		7,6655
1		6,4852
2		5,81863

Figure 8: Graphe courant à chaque itération de la recherche locale, ainsi que son énergie.

Algorithme 8: Algorithme PERTADDREM

Input : G
Input : k
répéter k **fois**
  Choisir une paire (i, j) de sommets de G.
  si *il existe une arête entre* i *et* j **alors**
    Soit G' le graphe obtenu en supprimant l'arête entre i et j.
  sinon
    Soit G' le graphe obtenu en ajoutant une arête entre i et j.
retourner G'

3.2 Améliorer k-means

Parmi les applications de la RVV, la classification non supervisée est un domaine important. L'algorithme de regroupement le plus connu est sans conteste k-means, ou *k-moyenne* [13]. Cet algorithme est disponible dans la plupart des logiciels d'analyse de données et est considéré comme une référence.

Le critère utilisé par k-means est la somme des carrés des erreurs, ce qui signifie la somme des carrés des écarts des observations au barycentre de leur classe est minimisée, tel que le décrit l'algorithme 10.

Pour ce problème, une solution est une partition des observations en P classes. Cette partition peut être décrite par les valeurs de variables binaires $z_{ip} = 1$ si et seulement si l'observation i appartient à la classe p. Une solution S du problème sera alors décrite par les valeurs z_{ip}. Chaque observation étant décrite par un vecteur de dimension m, x_{ij} étant la valeur de la variable j pour l'observation i. Nous notons μ_{pj} la moyenne de la variable j dans la classe p, qui définissent les coordonnées du barycentre de la classe p tel que défini par l'équation 1.

$$\mu_{pj} = \frac{\sum_i z_{ip} x_{ij}}{\sum_i z_{ip}}. \tag{1}$$

Algorithme 9: Algorithme PERTMOVE

Input : G
Input : k
répéter k **fois**
  Choisir une paire (i, j) de sommets de non adjacents de G.
  Choisir une paire (i', j') de sommets de adjacents de G.
  Soit G' le graphe obtenu en
  ajoutant une arête entre i et j, et en
  enlevant l'arête entre i' et j'.
retourner G'

Algorithme 10: Algorithme k-means

Input : S
Poser $ameliore \leftarrow vrai$
tant que $ameliore = vrai$ **faire**
  $ameliore \leftarrow faux$
  Étape 1:
  Calculer les valeurs μ_{pj} selon l'équation 1.
  Étape 2:
  pour tout i **faire**
    Noter $ptqz_{ip} = 1$
    Soit d_{ip} la distance euclidienne entre l'observation i et le vecteur μ_p.
    si $\exists p' tq d_{ip'} < d_{ip}$ **alors**
      $ameliore \leftarrow vrai$
      Modifier S : $z_{ip} \leftarrow 0$ et $z_{ip'} \leftarrow 1$.
retourner S

L'algorithme 10 (K-means) n'est autre qu'une recherche locale. Il se trouve que cette recherche locale donne une solution dont la qualité dépend fortement de la solution initiale utilisée. Malheureusement, pour ce type de problème, le nombre d'optimums locaux est important et la solution que produit k-means peut se montrer très mauvaise, ce qui peut induire certains chercheurs en erreur, convaincus que l'algorithme produit la meilleure solution possible [16].

Au lieu d'essayer de trouver une manière de construire la solution initiale à utiliser, nous allons proposer ici une manière d'adapter l'algorithme en l'intégrant à la recherche à voisinages variables. L'algorithme que nous proposerons ici est une extension naturelle de k-means facile à mettre en œuvre.

Pour de meilleurs résultats, le lecteur peut se référer à j-means [8], mais nous proposerons ici une manière simple d'intégrer k-means dans une recherche à voisinages variables.

Puisque nous disposons d'une recherche locale assez rapide, il nous reste à construire des perturbations. Une première perturbation $PERTBASE$ est décrite à l'algorithme 11.

Algorithme 11: Algorithme PERTBASE

Input : k
Input : S
répéter k **fois**
  Choisir aléatoirement une observation i.
  Choisir aléatoirement une classe p'.
    Poser $z_{ip} \leftarrow 0 \quad \forall p = 1 \dots P$.
    Poser $z_{ip'} \leftarrow 1$.
retourner S

La perturbation $PERTBASE$ consiste donc à déplacer k fois une observation de sa classe, vers une classe aléatoire. Cette perturbation se montrera peu efficace en pratique car elle ne modifie que peu la structure de la solution. On lui préfèrera des perturbations plus sophistiquées telles que $PERTSPLIT$ ou $PERTMERGE$ définies par les algorithmes 12 et 13 respectivement. Ces perturbations modifient la solution, mais prennent en considération la nature du problème et produisent une solution de nature différente mais consistante.

Algorithme 12: Algorithme PERTSPLIT

Input : k
Input : S
répéter k **fois**
  Choisir aléatoirement une classe p_1.
  Déplacer toutes les observations de p_1 vers la classe la plus proche.
  Choisir aléatoirement une classe p_2.
  Affecter aléatoirement les observations de p_2 à p_1 ou p_2.
  Appliquer l'algorithme k-means restreint aux observations des classes p_1 et p_2.
retourner S

Algorithme 13: Algorithme PERTMERGE

Input : k
Input : S
répéter k **fois**
  Choisir aléatoirement une classe p_1.
  Choisir aléatoirement une obersvation i de p_1.
  Noter p_2 la classe la plus proche de i ormis p_1.
  Affecter aléatoirement les observations de p_1 ou p_2 à p_1 ou p_2 (mélanger les classes p_1 et p_2).
  Appliquer l'algorithme k-means restreint aux observations des classes p_1 et p_2.
retourner S

L'algorithme $PERTSPLIT$ répète k fois le fait de vider une classe de ses éléments avant de choisir une autre classe qui sera séparée en deux par k-means. Si le choix des deux classes est approprié, et que les classes p_1 et p_2 sont proches, il sera beaucoup plus pertinent que $PERTBASE$.

L'algorithme $PERTMERGE$ prend deux classes, et redistribue leurs observations selon l'algorithme k-means. Comme dans le cas de $PERTSPLIT$, si le choix des deux classes est approprié, et que les classes p_1 et p_2 sont proches, il sera beaucoup plus pertinent que $PERTBASE$.

Une des raisons qui font que ces deux perturbations ont un bon potentiel de performance tient au fait qu'en général, l'algorithme k-means fonctionne bien quand le nombre de classes est modéré. k-means est ici utilisé pour optimiser localement la partition. L'utilisation de la version réduite de la recherche à voisinages variables est logiquement appropriée à ce type de problème, les perturbations jouant elles-mêmes le rôle de recherche locale.

Si la perturbation PERTMERGE semble se comporter légèrement mieux que PERTSPLIT, la combinaison des deux semble toutefois une stratégie à privilégier.

3.3 Adapter la RVV à des problèmes continus

Bien que nous n'ayons décrit l'application de la recherche à voisinages variables que dans le contexte de l'optimisation combinatoire, cette dernière peut aisément être utilisée pour résoudre des problèmes d'optimisation continue [14].

3.3.1 Recherche locale

Pour des problèmes d'optimisation continue, la recherche locale ne peut pas être décrite de la même manière que nous venons de la voir. Il y a alors deux possibilités :

- soit la fonction à optimiser est dérivable, et une recherche selon la méthode du gradient, ou si c'est possible, la méthode de Newton ou encore une méthode quasi-Newton peuvent être utilisées. La recherche à voisinages variables peut aisément être adaptée à n'importe quel type de recherche locale, et l'utilisation de la descente à voisinages variables sera plus ou moins pertinente, même si elle peut en principe être utilisée. Nous référons le lecteur à n'importe quelle technique d'optimisation continue, qui permet de trouver un optimum local.
- Soit la fonction à optimiser n'est pas dérivable et le recours à des méthodes de recherche directe sont nécessaires. Un exemple d'utilisation de la recherche à voisinages variables combinée à la recherche directe est décrite dans [1].

3.3.2 Perturbations

Dans le cas continu, les perturbations peuvent être aussi simples que modifier d'une valeur aléatoire proportionnelle à k la valeur de chaque variable, mais selon le problème, il est possible que des stratégies plus complexes soient nécessaires (ou plus efficaces).

4 Conclusion

Nous n'avons vu ici que les principales versions de la recherche à voisinages variables. Le lecteur désireux d'approfondir ses connaissances est invité à se référer aux articles suivants :

1. L'article de référence par excellence est [9]. Il donne une description de la recherche à voisinages variables ainsi que des applications telles que le problème du voyageur de commerce, le problème de la p-médiane, le problème de Weber multisources, la classification par la minimisation de la somme des carrés (k-means) et la programmation bilinéaire avec contraintes bilinéaires. Quelques extensions sont ensuites décrites parmi lesquelles la recherche à voisinages variables avec décomposition et les recherches à voisinages variables imbriquées.
2. Un complément d'information peut être trouvé dans [7] qui est consacré à la description de développements associés à la recherche à voisinages variables. Une première partie traite de la manière de mettre la méthode en pratique, particulièrement pour l'application à des problèmes de grande taille alors qu'un certain nombre d'applications sont ensuite décrites allant de la planification de tâches sur des architectures multiprocesseurs avec délais de communication au problème de la clique maximum en passant par des problèmes de localisation ou le problème de trouver l'arbre de recouvrement maximum avec contraintes de degrés.
3. L'application de la recherche à voisinages variables dans un contexte où les variables sont continues est décrite en détail dans [14]. Les cas contraints et non contraints y sont traités.
4. Plus récemment, deux publications [12] et [11] ont été dédiées à une revue de la littérature couvrant aussi bien les techniques qu'une grande variété d'applications. En plus du texte assez exhaustif, ces deux publications proposent une vaste liste de références couvrant aussi bien les techniques que les applications.

Pour finir, rappelons que la recherche à voisinages variables, comme la plupart des métaheuristiques, propose avant tout un schéma général, une trame sur laquelle le chercheur peut s'appuyer. Ce schéma n'est nullement limitatif et gagne souvent à être adapté ou modifié selon les spécificités de chaque problème.

References

[1] C. Audet, V. Béchard, and S. LeDigabel. Nonsmooth optimization through Mesh Adaptive Direct Search and Variable Neighborhood Search. *Journal of Global Optimization*, 41(2):299–318, 2008.

[2] G. Caporossi, D. Cvetković, I. Gutman, and P. Hansen. Variable neighborhood search for extremal graphs 2. Finding graphs with extremal energy. *Journal of Chemical Information and Computer Sciences*, 39:984–996, 1999.

[3] G. Caporossi and P. Hansen. Variable neighborhood search for extremal graphs. 1. The AutoGraphiX system. *Discrete Math.*, 212(1-2):29–44, 2000.

[4] G. Caporossi and P. Hansen. A learning optimization algorithm in graph theory - versatile search for extremal graphs using a learning algorithm. In *LION'12*, pages 16–30, 2012.

[5] J.L. Gross, J. Yellen, and P. Zhang. *Handbook of Graph Theory, Second Edition*. Chapman and Hall/CRC, 2013.

[6] P. Hansen, B. Jaumard, N. Mladenovic, and A.D. Parreira. Variable neighbourhood search for maximum weight satisfiability problem. Technical Report. Les Cahiers du GERAD G–2000–62, HEC Montréal, 2000.

[7] P. Hansen and N. Mladenović. Developments of variable neighborhood search. In F. Glover and G. Kochenagen, editors, *State-of-the-art Handbook of Metaheuristics*. Kluwer, Dordrecht, 2001.

[8] P. Hansen and N. Mladenović. J-MEANS, a new local search heuristic for minimum sum-of-squares clustering. *Pattern Recognition*, 34:405–413, 2001.

[9] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European J. Oper. Res.*, 130:449–467, 2001.

[10] P. Hansen and N. Mladenović. First vs. best improvement: An empirical study. *Discrete Applied Mathematics*, 154(5):802–817, 2006.

[11] P. Hansen, N. Mladenović, J. Brimberg, and J.-A. Moreno Perez. Variable neighborhood search. In Gendreau and Potvin, editors, *Handbook of Metaheuristics*, pages 61–86. Kluwer, 2010.

[12] P. Hansen, N. Mladenović, and J.-A. Moreno Perez. Variable neighborhood search: Methods and applications. *4OR*, 6(4):319–360, 2008.

[13] J.B. MacQueen. Some methods for classification and analysis of multivariate observations. In *5th Berkeley Symp. on Mathem. Statist. and Probab.*, pages 281–297, 1967.

[14] N. Mladenović, M. Dražić, V. Kovačević-Vujčić, and M. Čangalović. General variable neighborhood search for the continuous optimization. *European Journal of Operations Research*, 191:753–770, 2008.

[15] N. Mladenović and P. Hansen. Variable neighborhood search. *Comput. Oper. Res.*, 24:1097–1100, 1997.

[16] D. Steinley. Local optima in K-means clustering: What you don't know may hurt you. *Psychol. Methods*, 8(3):294–304, 2003.

[17] R. Todeschini and V. Consonni. *Handbook of Molecular Descriptors*. Wiley-VCH, 2000.