Métaheuristiques

Le mot ***métaheuristique*** est dérivé de la composition de deux mots grecs:
- *heuristique* qui vient du verbe heuriskein (ευρισκειν) et qui signifie 'trouver'
- *meta* qui est un suffixe signifiant 'au-delà', 'dans un niveau supérieur'.

Avant l'adoption de ce mot, on parlait plutôt d'*heuristique moderne*. Plusieurs définitions ont été proposées pour expliquer clairement ce qu'est une métaheuristique. Aucune de ces définitions n'est universellement reconnue :

"A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions."
[I.H. Osman and G. Laporte, *Metaheuristics: a bibliography*. Annals of Operations Research 63, 513-623, 1996]

"A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a constructive method."
[S. Voß, S. Martello, I.H. Osman and C. Roucairol (Eds), *Meta-Heuristics - Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers, Dordrecht, The Netherlands, (1999)]

"Metaheuristics are typically high level strategies which guide an underlying more problem specific heuristic, to increase their performance. The main goal is to avoid the disadvantages of iterative improvement and, in particular, multiple descent by allowing the local search to escape from local optima. This is achieved by either allowing worsening moves or generating new starting solutions for the local search in a more "intelligent" way than just providing random initial solutions. Many of the methods can be interpreted as introducing a bias such that high quality solutions are produced quickly. This bias can be of various forms and can be cast as descent bias (based on the objective function), memory bias (based on previously made decisions) or experience bias (based on prior performance). Many of the metaheuristic approaches rely on probabilistic decisions made during the search. But the main difference to pure random search is that in metaheuristic algorithms randomness is not used blindly but in an intelligent, biased form."
[T. Stützle, *Local Search Algorithms for Combinatorial Problems – Analysis, Algorithms and New Applications*. DISKI – Dissertationen zur Künstliken Intelligenz. Infix, Sankt Augustin, Germany (1999)]

"A metaheurisic is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework which can be applied to different optimization problems with relatively few modifications to make them adapted to a specific problem."
[Metaheuristics Network Website http://www.metaheuristics.net visité en novembre 2004]

Pour résumer ces définitions, on peut dire que les propriétés fondamentales des métaheuristiques sont les suivantes.
- Les métaheuristiques sont des stratégies qui permettent de guider la recherche d'une solution optimale
- Le but visé par les métaheuristiques est d'explorer l'espace de recherche efficacement afin de déterminer des solutions (presque) optimales.
- Les techniques qui constituent des algorithmes de type métaheuristique vont de la simple procédure de recherche locale à des processus d'apprentissage complexes.
- Les métaheuristiques sont en général non-déterministes et ne donnent aucune garantie d'optimalité
- Les métaheuristiques peuvent contenir des mécanismes qui permettent d'éviter d'être bloqué dans des régions de l'espace d recherche.
- Les concepts de base des métaheuristiques peuvent être décrit de manière abstraite, sans faire appel à un problème spécifique.
- Les métaheuristiques peuvent faire appel à des heuristiques qui tiennent compte de la spécificité du problème traité, mais ces heuristiques sont contrôlées par une stratégie de niveau supérieur.
- Les métaheuristiques peuvent faire usage de l'expérience accumulée durant la recherche de l'optimum, pour mieux guider la suite du processus de recherche.

Problème à résoudre

Soit S un ensemble de solutions à un problème d'optimisation.

Soit f une fonction qui mesure la valeur f(s) de toute solution s dans S.

On veut déterminer une solution $s \in S$ de valeur f(s) minimale. Le problème à résoudre est donc le suivant :

$$\min f(s)$$
$$\text{s.c. } s \in S$$

Une **structure de voisinage** (ou tout simplement un voisinage) est une fonction N qui associe un sous-ensemble de S à toute solution $s \in S$. Une solution $s' \in N(s)$ est dite voisine de s.

Une solution $s \in S$ est un **minimum local** relativement à la structure de voisinage N si $f(s) \leq f(s')$ pour tout $s' \in N(s)$. Une solution $s \in S$ est un **minimum global** si $f(s) \leq f(s')$ pour tout $s' \in S$.

Les métaheuristiques sacrifient la garantie d'optimalité ou d'approximation avec en contrepartie l'espoir de trouver *très rapidement* de bonnes solutions dans S.

Classifications possibles des métaheuristiques

On peut faire la différence entre les métaheuristiques qui s'inspirent de **phénomènes naturels** et celles qui ne s'en inspirent pas. Par exemple, les algorithmes génétiques et les algorithmes des fourmis s'inspirent respectivement de la théorie de l'évolution et du comportement de fourmis à la recherche de nourriture. Par contre, la méthode Tabou n'a semble-t-il pas été inspirée par un phénomène naturel. Une telle classification ne semble cependant pas très utile et est parfois difficile à réaliser. En effet, il existe de nombreuses métaheuristiques récentes qu'il est difficile de classer dans l'une des 2 catégories. Certains se demanderont par exemple si l'utilisation d'une mémoire dans la méthode Tabou n'est pas directement inspirée de la nature.

Une autre façon de classifier les métaheuristiques est de distinguer celles qui travaillent avec une **population de solutions** de celles qui ne manipulent qu'une seule solution à la fois. Les méthodes qui tentent itérativement d'améliorer une solution sont appelées **méthodes de recherche locale** ou **méthodes de trajectoire**. La méthode Tabou, le Recuit Simulé et la Recherche à Voisinages Variables sont des exemples typiques de méthodes de trajectoire. Ces méthodes construisent une trajectoire dans l'espace des solutions en tentant de se diriger vers des solutions optimales. L'exemple le plus connu de méthode qui travaille avec une population de solutions est l'algorithme génétique.

Les métaheuristiques peuvent également être classées selon leur manière d'utiliser la fonction objectif. Étant donné un problème d'optimisation consistant à minimiser une fonction f sur un espace S de solutions, certaines métaheuristiques dites **statiques** travaillent directement sur f alors que d'autres, dites **dynamiques**, font usage d'une fonction g obtenue à partir de f en ajoutant quelques composantes qui permettent de modifier la topologie de l'espace des solutions, ces composantes additionnelles pouvant varier durant le processus de recherche.

Des chercheurs préfèrent classifier les métaheuristiques en fonction du **nombre de structures de voisinages** utilisées. Étant donné qu'un minimum local relativement à un type de voisinage ne l'est pas forcément pour un autre type de voisinage, il peut être intéressant d'utiliser des métaheuristiques basées sur plusieurs types de voisinages.

Certaines métaheuristiques font **usage de l'historique** de la recherche au cours de l'optimisation, alors que d'autres n'ont aucune mémoire du passé. Les algorithmes sans mémoire sont en fait des processus markoviens puisque l'action à réaliser est totalement déterminée par la situation courante. Les métaheuristiques qui font usage de l'historique de la recherche peuvent le faire de diverses manières. On différentie généralement les méthodes ayant une **mémoire à court terme** de celles qui ont une **mémoire à long terme**.

Finalement, mentionnons que certaines métaheuristiques utilisent les concepts additionnels que sont la **diversification** et l'**intensification**. Par diversification, on sous-entend généralement une exploration assez large de l'espace de recherche, alors que le terme intensification vient plutôt mettre l'accent sur l'exploitation de l'information accumulée durant la recherche. Il est important de bien doser l'usage de ces deux ingrédients afin que l'exploration puisse rapidement identifier des régions de l'espace de recherche qui contiennent des solutions de bonne qualité, sans perdre trop de temps à exploiter des régions moins prometteuses.

Dans notre description des principales métaheuristiques, nous allons nous appuyer sur la classification qui distingue les méthodes de trajectoire des méthodes basées sur des populations de solutions.

Recherche Locale - Méthodes de trajectoire

L'ensemble S définit l'ensemble des points pouvant être visités durant la recherche. La structure de voisinage N donne les règles de déplacement dans l'espace de recherche. La fonction objectif f induit une topologie sur l'espace de recherche.

La méthode de recherche locale la plus élémentaire est la **méthode de descente**. Elle peut être décrite comme suit.

Méthode de descente

1. choisir une solution $s \in S$
2. Déterminer une solution s' qui minimise f dans N(s).
3. Si f(s') < f(s) alors poser s := s' et retourner à 2. Sinon STOP

Une variante consiste à parcourir N(s) et à choisir la première solution s' rencontrée telle que f(s')< f(s) (pour autant qu'une telle solution existe). Le principal défaut des méthodes de descente est qu'elles s'arrêtent au premier minimum local rencontré. Tel que déjà mentionné, un minimum local pour une structure de voisinage ne l'est pas forcément pour une autre structure. Le choix de N peut donc avoir une grand influence sur l'efficacité d'une méthode de descente.

Pour éviter d'être bloqué au premier minimum local rencontré, on peut décider d'accepter, sous certaines conditions, de se déplacer d'une solution s vers une solution $s' \in N(s)$ telle que $f(s') \geq f(s)$. C'est ce que font les méthodes que nous décrivons ci-dessous. Commençons par le **Recuit Simulé** (Kirkpatrick, Gelatt, Vecchi, 1983) qui peut être décrit comme suit.

Méthode du Recuit Simulé

1. Choisir une solution $s \in S$ ainsi qu'une température initiale T.
2. Tant qu'aucun critère d'arrêt n'est satisfait faire
3. Choisir aléatoirement $s' \in N(s)$;
4. Générer un nombre réel aléatoire r dans [0,1];
5. Si r < p(T,s,s') alors poser s := s';
6. Mettre à jour T;
7. Fin du tant que

La fonction p(T,s,s') est généralement choisie comme étant égale à la distribution de Boltzmann $e^{\frac{f(s)-f(s')}{T}}$. Ainsi,

- Si f(s')<f(s) alors $e^{\frac{f(s)-f(s')}{T}} > 1$, ce qui signifie que r < p(T,s,s') et on accepte donc s'
- Si T a une très grande valeur alors $e^{\frac{f(s)-f(s')}{T}} \cong 1$, et on est donc presque sûr d'accepter s'
- Si T a une très petite valeur et si f(s')>f(s) alors $e^{\frac{f(s)-f(s')}{T}} \cong 0$ et on va donc probablement refuser s'

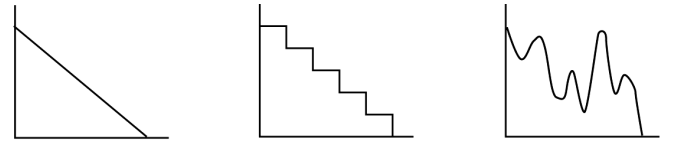
En général, on choisit une température initiale suffisamment élevée qui donne une plus grande liberté pour l'exploration de l'espace de recherche. Puis, petit à petit, la température décroît jusqu'à atteindre une valeur proche de 0, ce qui signifie que la méthode n'acceptera plus de détériorer une solution. Pour choisir la température initiale, on peut générer aléatoirement un ensemble de paires (s,s') où $s' \in N(s)$, et choisir ensuite la température initiale de telle sorte qu'une proportion ρ de ses solutions auraient été acceptées à l'étape 5 de l'algorithme.

Il a été démontré par Aarts, Korst et Laarhoven (1997) que sous certaines conditions de décroissance de la température, l'algorithme du Recuit Simulé converge en probabilité vers un optimum global lorsque le nombre d'itérations tend vers l'infini. Plus précisément, soit P(k) la probabilité que l'optimum global soit trouvé après k itérations, et soit T_k la température à la $k^{ème}$ itération.

$$\text{Il existe } L \in \mathbb{R} \text{ tel que } \lim_{k\to\infty} P(k)=1 \text{ si et seulement si } \sum_{k=1}^{\infty} e^{\frac{L}{T_k}} = \infty$$

On peut, par exemple, définir $T_{k+1} = \frac{L}{\log(k+c)}$ où c est une constante. Mais de telles stratégies de décroissance de la température sont trop lentes et nécessitent donc des temps de calcul astronomiques avant d'atteindre la convergence vers l'optimum global. En pratique, on préfère donc utiliser d'autres stratégies qui ne garantissent plus la convergence vers l'optimum global, mais qui convergent plus rapidement vers un état stable. La méthode la plus utilisée consiste à définir $T_{k+1} = \alpha T_k$ où α est un nombre réel choisi entre 0 et 1 (typiquement α=0.95).

La décroissance de la température peut également être réalisée par paliers. Certains préconisent l'utilisation de stratégies non monotones. On peut ainsi rehausser la température lorsque la recherche semble bloquée dans une région de l'espace de recherche. On peut alors considérer une grande augmentation de la température comme un processus de diversification alors que la décroissance de la température correspond à un processus d'intensification.



La méthode du Recuit Simulé est une méthode sans mémoire. On peut facilement l'améliorer en rajoutant une mémoire à long terme qui stocke la meilleure solution rencontrée. En effet, l'algorithme du Recuit Simulé décrit ci-dessus peut converger vers une solution s* en ayant visité auparavant une solution s de valeur $f(s)<f(s^*)$. Il est donc naturel de mémoriser la meilleure solution rencontrée durant le processus de recherche.

Comme critère d'arrêt, on peut choisir une limite sur le temps, ou une limite sur le nombre d'itérations sans modification de la solution courante. Lorsqu'on stocke la meilleure solution rencontrée, on peut décider de stopper la recherche dès qu'un certain nombre d'itérations a été effectué sans amélioration de cette solution.

L'un des principaux défauts de la méthode du Recuit Simulé est l'étape 3 où s' est choisi aléatoirement dans N(s). On peut donc être proche de l'optimum et passer juste à côté sans le voir. La **Recherche Tabou**, décrite ci-dessous, n'a pas ce défaut. Elle a été proposée par Glover en 1986.

Le principe de la Recherche Tabou est de choisir à chaque itération la meilleure solution $s' \in N(s)$, même si $f(s')>f(s)$. Lorsqu'on atteint un minimum local s par rapport au voisinage N, la Recherche Tabou va donc se déplacer vers une solution s' plus mauvaise que s. Le danger est alors de revenir à s immédiatement si $s \in N(s')$ puisque s est meilleure que s'. Pour éviter de tourner ainsi en rond, on crée une liste T qui mémorise les dernières solutions visitées et interdit tout déplacement vers une solution de cette liste. Cette liste T est appelée *liste taboue*, Les solutions ne demeurent dans T que pour un nombre limité d'itérations. La liste T est donc une mémoire à court terme. Si une solution s' est dans T on dit que s' est une solution taboue. De même, tout mouvement qui nous mène de la solution courante à une solution de T est appelé mouvement tabou.

En pratique, la mémorisation de plusieurs solutions dans T demande typiquement beaucoup de place mémoire. De plus, il peut être difficile de vérifier si une solution donnée est dans T. C'est pourquoi beaucoup préfèrent mémoriser des attributs de solutions, ou des modifications. Ainsi, on peut interdire de visiter une solution présentant un attribut présent dans T. Ou alors, on peut interdire une solution s' si le mouvement qui permet de se rendre de la solution courante s vers la solution s' est un mouvement appartenant à la liste T. La mémorisation de ces attributs ou modifications permet certes un gain en place mémoire, mais elle n'a pas que des avantages.

- Il se peut que l'on visite une même solution à plusieurs reprises, à intervalles plus petits que $|T|$. À titre d'illustration, supposons que l'on cherche à ordonner les 4 premières lettres de l'alphabet. Pour passer d'un ordre à un autre, on permute les positions de 2 lettres et on rend tabou une nouvelle permutation de ces deux lettres. On pourrait donc avoir la suite de solutions suivante :

  $$abcd \rightarrow bacd \rightarrow bcad \rightarrow dcab \rightarrow acdb \rightarrow abdc \rightarrow abcd$$

  Les permutations qui ont été effectuées sont les suivantes : ab, ac, bd, ad, bc et cd.
  On voit donc qu'aucune permutation n'a été réalisée deux fois et on retombe cependant sur la solution initiale.

- Il se peut que la liste T interdise la visite de solutions qui n'ont jamais été visitées (de telles solutions pouvant être des optima globaux). Par exemple, supposons à nouveau que l'on cherche à ordonner les premières lettres de l'alphabet. Comme ci-dessus, on permute les positions de 2 lettres pour se déplacer d'une solution à une autre et on rend tabou une nouvelle permutation de ces 2 lettres. On pourrait avoir la suite de solutions suivante :

  $$abcd \rightarrow bacd \rightarrow dacb \rightarrow dcab$$

  On a permuté les lettres ab, puis bd, et enfin ac. Si $|T| \geq 3$, on n'a donc plus le droit de permuter ab. Ainsi, on interdit la solution dcba qui pourrait être la solution optimale.

Pour éviter le 2ème défaut mentionné ci-dessus, il est d'usage de lever le statut tabou d'une solution si celle-ci satisfait certains *critères d'aspiration*. En règle générale, le statut tabou d'un solution est levé si celle-ci est meilleure que la meilleure solution s* rencontrée jusqu'ici.

Étant donnée une solution courante s, définissons l'ensemble $N^T(s)$ comme l'ensemble des solutions de N(s) vers lesquelles la Recherche Tabou accepte de se diriger. L'ensemble $N^T(s)$ contient donc toutes les solutions qui ne sont pas taboues ainsi que celles qui le sont mais dont le statut tabou est levé en raison du critère d'aspiration. Ainsi :

$$N^T(s) = \{ s' \in N(s) \text{ tel que } s' \notin T \text{ ou } f(s') < f(s^*) \}$$

La Recherche Tabou peut être décrite comme suit

Recherche Tabou

1. Choisir une solution $s \in S$, poser $T := \emptyset$ et $s^* := s$;
2. Tant qu'aucun critère d'arrêt n'est satisfait faire
3. Déterminer une solution s' qui minimise f(s') dans $N^T(s)$
4. Si $f(s') < f(s^*)$ alors poser $s^* := s'$
5. Poser $s := s'$ et mettre à jour T
6. Fin du tant que

Comme critère d'arrêt on peut par exemple fixer un nombre maximum d'itérations sans amélioration de s*, ou on peut fixer un temps limite après lequel la recherche doit s'arrêter.

Certains préconisent l'utilisation d'une liste taboue de longueur variable (Taillard, 1991). D'autres préfèrent les listes taboues réactives (Battiti et Tecchiolli, 1994) dont la taille varie selon les résultats de la recherche. Ainsi, si des solutions sont visitées de manière répétée (à intervalle > |T|) alors on peut augmenter la longueur de la liste, ce qui aura pour effet de diversifier la recherche. Par contre, si la solution courante n'est que rarement améliorée, cela peut signifier qu'il est temps d'intensifier la recherche en évitant d'interdire trop de solutions dans le voisinage de la solutions courante, et en diminuant donc la longueur de la liste taboue.

La liste taboue est une mémoire à court terme. On peut rajouter une mémoire à plus long terme. Quatre types de mémoire sont alors envisageables :

- Les mémoires basées sur la *récence* mémorisent pour chaque solution (attribut ou mouvement) l'itération la plus récente qui l'impliquait. On peut par exemple favoriser les mouvements vers des solutions contenant des attributs peu récents, afin d'éviter de rester bloquer dans une même région de l'espace de recherche.
- Les mémoires basées sur la *fréquence* mémorisent le nombre de fois que les solutions (attributs ou mouvements) ont été visitées. Cette information permet d'identifier les régions de l'espace de recherche qui ont été le plus visitées. On peut exploiter cette information pour diversifier la recherche.
- Les mémoires basées sur la *qualité* mémorisent les meilleures solutions rencontrées afin d'identifier les composantes communes de ces solutions. On peut ensuite faire usage de cette information pour créer de nouvelles solutions qui contiennent autant que possible ces apparemment bonnes composantes.
- Les mémoires basées sur l'*influence* mémorisent les choix qui se sont avérés les plus judicieux ou les plus critiques durant le processus de recherche. Ceci permet d'essayer de répéter les bons choix et d'éviter de répéter les mêmes erreurs.

La méthode de descente, le Recuit Simulé et la Recherche Tabou sont probablement les méthodes à trajectoires les plus connues et les plus utilisées. Il existe cependant d'autres méthodes ayant également leur cote de popularité. Nous en décrivons trois. La première, appelée **GRASP** pour Greedy Randomized Adaptive Search Procedure, a été proposée par Feo et Resende en 1995. Cette méthode est une procédure itérative composée de deux phases : une phase constructive et une phase d'amélioration.

- En supposant qu'une solution est constituée d'un ensemble de composantes, la phase constructive génère une solution pas à pas, en ajoutant à chaque étape une nouvelle composante. La composante rajoutée est choisie dans une liste de candidats. Chaque composante est en fait évaluée à l'aide d'un critère heuristique qui permet de mesurer le bénéfice qu'on peut espérer en rajoutant cette composante à la solution partielle courante. La liste de candidats, notée RCL (pour Restricted Candidate List) contient les R meilleures composantes selon ce critère.
- La phase d'amélioration consiste généralement en l'application d'une méthode de descente, de Recuit Simulé ou de Recherche Tabou.

En résumé, la méthode GRASP peut être décrite comme suit.

Méthode GRASP

1. f*=∞ (valeur de la meilleure solution rencontrée)
2. Tant qu'aucun critère d'arrêt n'est satisfait faire
3. Poser s:=∅
4. Tant que la solution courante s n'est pas complète faire
5. Évaluer l'ensemble des composantes pouvant être rajoutées à s
6. Déterminer la liste RCL des R meilleures composantes
7. Choisir aléatoirement une composante dans RCL et l'ajouter à s
8. Fin du tant que
9. Appliquer une méthode de descente, de Recuit Simulé ou de Recherche Tabou à s et soit s' le résultat.
10. Si f(s')<f* alors poser s*:=s' et f* :=f(s');
11. Fin du tant que

À titre d'exemple, pour le problème du voyageur de commerce, on peut rajouter une ville après l'autre pour créer une tournée. L'insertion d'une ville dans une tournée partielle est évaluée en mesurant la longueur du détour occasionné par le rajout de la ville.

La longueur R de la liste RCL joue un rôle particulièrement important. Si on pose R=1, la phase constructive est un algorithme glouton qui choisit à chaque étape la meilleure composante. Si on fixe R égal au nombre de composantes pouvant être rajoutées, on a alors un algorithme constructif purement aléatoire. Certains préconisent l'utilisation d'une même valeur R durant tout l'algorithme. D'autres préfèrent faire varier ce paramètre de manière adaptative.

La méthode GRASP est assez simple à mettre en place (surtout si on utilise une méthode de descente à l'étape 9). Pour qu'elle soit efficace, il est important d'utiliser une méthode constructive capable de générer des solutions dans des régions différentes de l'espace de recherche.

La **Recherche à Voisinages Variables** (RVV) a été proposé par Mladenovic et Hansen en 1997. Contrairement aux méthodes de recherche locale présentées jusqu'ici, une RVV utilise méthodiquement plusieurs types de voisinages. Soit $L = (N^{(1)},\ldots, N^{(T)})$ une liste finie de voisinages, où $N^{(t)}(s)$ est l'ensemble des solutions dans le t^e voisinage de s. Dans la plupart des méthodes de recherche locale, on a T=1. Ces voisinages interviennent de la manière suivante dans le processus de recherche d'une RVV. Étant donnée une solution initiale s, on génère une solution voisine s' dans $N^{(1)}(s)$ et on lui applique une procédure de recherche locale afin d'obtenir une solution s''. Si f(s'') < f(s), alors on pose s = s'' et on génère une nouvelle solution voisine dans $N^{(1)}(s)$. Sinon, la solution courante reste s et on change de voisinage en générant une solution s' dans $N^{(2)}(s)$. Plus généralement, on change de voisinage à chaque fois que l'un d'entre eux n'est pas parvenu, après application de la procédure de recherche locale, à améliorer la solution courante s. Par contre, dès qu'un voisinage permet d'améliorer s, alors on recommence le processus avec le premier voisinage de la liste L. Une version de base d'une RVV est décrite ci-dessous.

Recherche à Voisinages Variables

1. Choisir une solution s∈ S et poser t :=1
2. Tant qu'aucun critère d'arrêt n'est satisfait faire
3. Choisir aléatoirement une solution s' dans $N^{(t)}(s)$
4. Appliquer une procédure de Recherche Locale à s'; soit s'' la solution ainsi obtenue;
5. Si f(s'') < f(s) alors poser s:=s'' et t :=0;
6. Si t < T alors poser t := t+1, sinon poser t :=1;
7. Fin du tant que

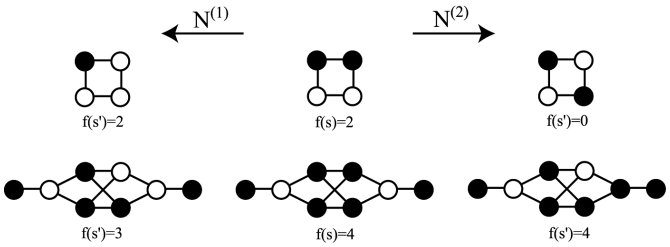
Le fait d'utiliser plusieurs voisinages permet de diversifier l'exploration de l'espace des solutions afin d'accéder à un plus grand nombre de régions intéressantes, ce qui conduit à une méthode plus robuste que le recuit simulé ou la recherche Tabou. Le voisinage utilisé dans la Recherche Locale à l'étape 4 n'est pas forcément le voisinage courant $N^{(t)}$. Il peut même s'agir d'une structure de voisinage totalement différente des $N^{(i)}$.

Considérons une fonction d qui mesure une distance d(s,s') entre 2 solutions. On peut par exemple définir d(s,s') comme étant égal au nombre de composantes qui diffèrent dans s et s'. Soit D_i la distance moyenne de s à une solution de $N^{(i)}$. Certains préfèrent choisir des structures de voisinages tel que les D_i sont croissants. Une telle stratégie permet de diversifier la recherche lorsqu'on est bloqué dans une région de l'espace de recherche. D'autres préfèrent choisir des voisinages tel que les D_i sont décroissants. Ceci permet d'imiter la stratégie du Recuit Simulé, où l'on tente d'abord de déterminer une bonne région, puis on intensifie petit à petit la recherche dans cette région.

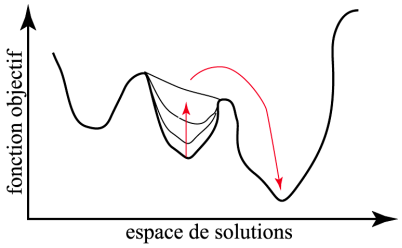
Pour qu'une Recherche à Voisinages Variables soit efficace, il est recommandé d'utiliser des structures de voisinage qui soient complémentaires en ce sens qu'un minimum local pour un voisinage ne l'est pas forcément pour un autre. Par exemple, considérons un problème de coloration des sommets d'un graphe G=(V,E). Supposons que toute fonction c : $V \rightarrow \mathbb{N}$ est considérée comme solution. Une arête est dite en conflit si ses deux extrémités sont de même couleur. De même, un sommet est dit en conflit s'il est l'extrémité d'une arête en conflit. Considérons la fonction f qui compte le nombre d'arêtes en conflit. Étant donnée une coloration c, considérons les 2 voisinages suivants :

- le voisinage $N^{(1)}$ consiste à changer la couleur d'un sommet en conflit (la nouvelle couleur doit être l'une des couleurs utilisées dans le graphe)
- Le voisinage $N^{(2)}$ consiste à choisir un sommet v en conflit, à donner à v une couleur i différente de c(v) mais présente dans son voisinage, à choisir ensuite un sommet w voisin de v et ayant la couleur i, et à donner la couleur c(v) à w. En d'autres termes, ce deuxième voisinage permute les couleurs de v et w, où v et w sont voisins, v est en conflit, et l'arête reliant v et w n'est pas en conflit.

Les deux exemples ci-dessous montrent qu'un minimum local pour un voisinage ne l'est pas forcément pour l'autre.



La **Recherche Locale Guidée**, proposée par Voudouris en 1997 consiste à utiliser une technique de recherche locale dans laquelle la fonction objectif varie durant le processus de recherche, le but étant de rendre les minima locaux déjà visités moins attractifs. Le principe est illustré ci-dessous.



Pour ce faire, notons $\{A_1,\ldots,A_m\}$ un ensemble de m attributs utilisés pour discriminer les solutions de S. Par exemple, pour la coloration des sommets d'un graphe, on peut associer un attribut à chaque arête du graphe et on dira qu'une coloration a l'attribut A_e si et seulement l'arête e est en conflit dans la solution. Pour le problème du voyageur de commerce, on peut par exemple associer un attribut à chaque arête du graphe et dire qu'une tournée possède l'attribut A_e si l'arête e fait partie de la tournée. Soit w_i le poids de l'attribut A_i et soit $\delta_i(s)$ une variable qui vaut 1 si s possède l'attribut A_i, et 0 sinon. La Recherche Locale Guidée utilise la fonction objectif suivante :

$$f'(s)=f(s)+\lambda\sum_{i=1}^{m} w_i \delta_i(s)$$

où λ est un paramètre qui permet de faire varier l'importance du deuxième terme de cette fonction. On peut modifier les poids w_i au cours de l'algorithme. En règle général, lorsqu'on se déplace d'une solution s vers une solution voisine s', on augmente le poids des attributs de s'. Différentes stratégies de modifications de poids sont possibles.

La Recherche Locale Guidée peut être résumée comme suit.

Recherche Locale Guidée

1. Choisir une solution $s \in S$; poser s*:=s;
2. Tant qu'aucun critère d'arrêt n'est satisfait faire
3. Appliquer une Recherche Locale à s avec f' comme fonction objectif; soit s' la solution ainsi obtenue;
4. mettre à jour les poids w_i
5. poser s:=s';
6. Si f(s) < f(s*) alors poser s*:=s
7. Fin du tant que

Méthodes basées sur les populations – Méthodes évolutives

Les méthodes basées sur des populations de solutions sont parfois appelées **méthodes évolutives** parce qu'elles font évoluer une population d'individus selon des règles bien précises. Ces méthodes alternent entre des périodes d'adaptation individuelle et des périodes de coopération durant lesquelles les individus peuvent échanger de l'information. En résumé un méthode évolutive peut être décrite comme suit.

Méthode Évolutive

1. Générer une population initiale d'individus
2. Tant qu'aucun critère d'arrêt n'est satisfait faire
3. Exécuter une procédure de coopération;
4. Exécuter une procédure d'adaptation individuelle
5. Fin du tant que

Nous décrivons ci-dessous les principales caractéristiques qui permettent de faire la différence entre diverses méthodes évolutives.

A. Types d'individus

Les individus qui évoluent dans une méthode basée sur les populations ne sont pas nécessairement des solutions. Il peut aussi s'agir de morceaux de solutions ou tout simplement d'objets que l'on peut facilement transformer en solutions. Considérons par exemple un problème de tournées de véhicules. Un individu peut être la tournée d'un véhicule qui visite un sous-ensemble de clients alors qu'une solution est un ensemble de tournées (et donc d'individus) qui visitent tous les clients. Pour la coloration de graphes, on peut considérer toute permutation des sommets comme un individu. En appliquant l'algorithme séquentiel de coloration, on peut transformer une permutation en une coloration et donc un individu en solution.

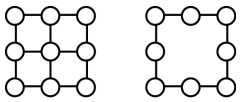
B. Type d'évolution

À chaque itération d'une méthode évolutive, de nouveaux individus sont créés et la population de l'itération suivante sera ainsi constituée d'anciens individus (qui auront survécu) et de nouveaux individus. La méthode évolutive doit indiquer comment décider de la survie des individus et comment choisir les nouveaux individus qui vont entrer dans la population. Lorsqu'on change totalement de population d'une itération à l'autre (c'est-à-dire que seuls les nouveaux individus sont conservés pour l'itération suivante), on parle de *remplacement générationnel*. Par contre, lorsque seulement une partie de la population peut varier d'une itération à la suivante, on parle de *remplacement stationnaire* (steady state).

La plupart des méthodes évolutives utilisent des populations de taille fixe, et on décide généralement de garder les p meilleurs individus (parmi l'union des anciens et des nouveaux). Des populations de taille variable (où on décide par exemple de manière aléatoire de la survie des individus) sont cependant également possibles.

C. Structure de voisinage

À chaque individu on associe un sous-ensemble d'autres individus avec lesquels il peut échanger de l'information. Si chaque individu peut communiquer avec tous les autres individus de la population, on parle de *population non structurée*. Par contre, si chaque individu ne peut communiquer qu'avec un sous-ensemble d'individus, on parle de *population structurée*. Les structures les plus communes sont la grille et le cercle.



structure en grille — structure en cercle

D. Sources d'information

Le nombre d'individus qui coopèrent pour créer un nouvel individu est souvent égal à 2. On parle alors de *parents* qui génèrent des *enfants*. On peut cependant également combiner plus de 2 solutions pour créer des enfants. Certaines méthodes évolutives utilisent par exemple l'information contenue dans toute la population pour créer un nouvel individu. D'autres méthodes utilisent même toutes les populations de toutes les itérations précédentes pour créer des enfants : on dit alors que la source d'information est *l'historique* de la recherche. Par historique on entend généralement toute information qui ne peut pas être obtenue en analysant les individus de la population courante; la connaissance des compositions des populations précédentes est nécessaire pour accéder à cette information. Comme nous le verrons sous peu, les méthodes de fourmis sont fortement basées sur l'historique de la recherche.

E. Irréalisabilité

Un individu est un objet défini avec des règles bien précises. En combinant des individus pour en créer des nouveaux, il peut arriver que le nouvel objet résultant de l'échange d'information n'est pas un individu admissible. Par exemple, définissons un individu comme une coloration sans conflit des sommets d'un graphe. Étant données deux colorations C_1 et C_2, on peut combiner celles-ci pour en créer une nouvelle en choisissant la couleur de chaque sommet aléatoirement dans C_1 ou C_2. Une telle coloration peut avoir des conflits et n'est donc pas nécessairement un individu. Dans une telle situation, on peut réagir d'au moins 3 façons :

- rejeter l'enfant;
- accepter l'enfant et rajouter une composante à la fonction objectif qui pénalise les violations de contraintes;
- développer des procédures de combinaisons qui garantissent l'obtention d'enfants admissibles (sans violation).

F. Intensification

Idéalement, on devrait pouvoir localiser l'information *pertinente* qui rend un individu meilleur qu'un autre. Lorsque ceci est possible, il reste alors à développer une procédure de coopération qui crée des enfants en combinant adéquatement les informations pertinentes de chacun des parents. La phase d'adaptation individuelle n'est alors pas vraiment indispensable au bon fonctionnement de la méthode évolutive. C'est le cas par exemple pour les problèmes de tournées de véhicules où la bonne qualité d'une solution peut être expliquée par l'utilisation d'arcs de faible coût. Il existe cependant de nombreux problèmes pour lesquels une telle information pertinente est difficile à localiser. Par exemple, si Π_1 et Π_2 sont deux permutations des sommets d'un graphe et si C_1 et C_2 sont les deux colorations résultant de l'algorithme séquentiel de coloration, avec C_1 utilisant moins de couleurs que C_2, il est difficile de localiser dans Π_1 les raisons qui font que Π_1 est meilleure que Π_2. Il est donc également difficile de transmettre une information pertinente lors de la phase de coopération. Il est alors important d'utiliser une bonne procédure d'adaptation individuelle. Il est courant de faire appel à une technique de Recherche Locale qui est appliquée à chacun des individus afin d'explorer les régions de l'espace de recherche qui leur sont proches.

G. Diversification

Une difficulté majeure rencontrée lors de l'utilisation des méthodes évolutives est leur convergence prématurée. Certains utilisent des procédures de *bruitage* qui modifient légèrement les individus de manière aléatoire. Ce bruitage est appliqué indépendamment sur chaque individu. Il diffère de l'utilisation d'une Recherche Locale par le fait que son effet sur la qualité de la solution n'est pas prévisible. L'opérateur de bruitage le plus connu est la *mutation* dans les algorithmes génétiques. Au lieu de modifier les individus aléatoirement, certains préfèrent créer de nouveaux individus différents des individus déjà rencontrés en faisant usage d'une mémoire à long terme basée par exemple sur la récence et la fréquence.

Quelques exemples

Les algorithmes génétiques sont inspirés de la théorie de l'évolution et des processus biologiques qui permettent à des organismes de s'adapter à leur environnement. Ils ont été inventés dans le milieu des années 60 (Holland, 1962; Rechenberg, 1965; Fogel et al, 1966). Dans l'algorithme ci-dessous qui n'est qu'une variante des algorithmes génétiques, la coopération a lieu à l'étape 5 alors que l'adaptation individuelle a lieu à l'étape 6. Une description des 7 caractéristiques est également donnée

Algorithme génétique avec remplacement générationnel

1. Générer une population initiale P_0 de $p \geq 2$ individus et poser i:=0;
2. Tant qu'aucun critère d'arrêt n'est satisfait faire
3. Poser i:=i+1 et $P_i := \emptyset$;
4. Répéter p fois les 2 lignes suivantes
5. Créer un enfant E en croisant 2 individus I_1 et I_2 de P_{i-1};
6. Appliquer l'opérateur de mutation à E et rajouter l'enfant modifié à P_i
7. Fin du tant que

Types d'Individus :	en général il s'agit de solutions
Type d'évolution :	remplacement générationnel avec une population de taille constante
Structure de voisinage :	population possiblement structurée
Sources d'information :	deux parents
Irréalisabilité :	l'opérateur de croisement évite la génération de solutions non admissibles
Intensification :	aucune
Diversification :	mutation qui est une procédure de bruitage

La **recherche dispersée** (scatter search) a été proposée par Glover en 1977. Elle consiste à générer un ensemble D_i de points dispersés à partir d'un ensemble R_i de points de références. Ces points dispersés sont obtenus en effectuant tout d'abord des combinaisons linéaires des points de référence. Ces combinaisons linéaires peuvent avoir des coefficients négatifs, ce qui veut dire que les points résultant peuvent être à l'extérieur de l'enveloppe convexe des R_i. L'ensemble C_i des points résultant de ces combinaisons linéaires n'est pas forcément un ensemble de solutions admissibles. C'est pourquoi, on applique une procédure de réparation sur chaque point de C_i pour obtenir un ensemble A_i de points admissibles. Les points de A_i sont finalement optimisés à l'aide d'une Recherche Locale pour obtenir l'ensemble D_i des points dispersés. Le nouvel ensemble R_{i+1} de points de référence est obtenu en sélectionnant des points dans $R_i \cup D_i$. Le pseudo-code et les caractéristiques de cette méthode sont donnés ci-dessous.

Algorithme de recherche dispersée

1. Générer une population initiale R_0 d'individus et poser i:=0;
2. Tant qu'aucun critère d'arrêt n'est satisfait faire
3. Créer un ensemble C_i de points candidats en effectuant des combinaisons linéaires des points de R_i;
4. Transformer C_i en un ensemble A_i de points admissibles (à l'aide d'une procédure de réparation);
5. Appliquer une Recherche Locale sur chaque point de A_i; soit D_i l'ensemble résultant.
6. Sélectionner des points dans $R_i \cup D_i$ pour créer le nouvel ensemble R_{i+1} de référence et poser i:=i+1;
7. Fin du tant que

Types d'Individus :	solutions admissibles
Type d'évolution :	remplacement stationnaire avec une population de taille généralement constante
Structure de voisinage :	population non structurée
Sources d'information :	au moins deux parents
Irréalisabilité :	les points non admissibles sont réparés
Intensification :	Recherche Locale
Diversification :	combinaison non convexe des points de référence.

L'**optimisation par colonies de fourmis** (ACO pour Ant Colony Optimisation) est une méthode évolutive inspirée du comportement des fourmis à la recherche de nourriture. Cet algorithme a été proposé par Dorigo en 1992. Il est connu que les fourmis sont capables de déterminer le chemin le plus court entre leur nid et une source de nourriture. Ceci est possible grâce à la phéromone qui est une substance que les fourmis déposent sur le sol lorsqu'elles de déplacent. Lorsqu'une fourmi doit choisir entre deux directions, elle choisit avec une plus grande probabilité celle comportant une plus forte concentration de phéromone. C'est ce processus coopératif qu'ACO tente d'imiter.

Chaque fourmi est un algorithme constructif capable de générer des solutions. Soit D l'ensemble des décisions possibles que peut prendre une fourmi pour compléter une solution partielle. La décision $d \in D$ qu'elle choisira dépendra de deux facteurs, à savoir la *force gloutonne* et la *trace*.

- la force gloutonne est une valeur η_d qui représente l'intérêt qu'a la fourmi à prendre la décision d. Plus cette valeur est grande, plus il semble intéressant de faire le choix d. En général, cette valeur est directement proportionnelle à la qualité de la solution partielle obtenue en prenant la décision d.
- la trace τ_d représente l'intérêt historique qu'a la fourmi de prendre la décision d. Plus cette quantité est grande, plus il a été intéressant dans le passé de prendre cette décision.

Étant donné deux paramètres α et ß qui donnent plus ou moins d'importance à ces deux facteurs, la fourmi va prendre la décision d avec une probabilité

$$\frac{(\eta_d)^\alpha (\tau_d)^\beta}{\sum_{r \in D} (\eta_r)^\alpha (\tau_r)^\beta}$$

Lorsqu'une fourmi termine la construction de sa solution, elle laisse une trace sur le chemin emprunté. Cette trace est proportionnelle à la qualité de la solution construite. De plus, il est important de mettre en place un processus d'évaporation de la trace afin d'oublier les choix réalisés dans un lointain passé et de donner plus d'importance aux choix réalisés récemment. Soit ρ un paramètre d'évaporation choisi dans l'intervalle]0,1[. Soit A l'ensemble des fourmis, et soit f(a) la valeur de la solution produite par la fourmi a. La qualité de la solution produite par la fourmi a est donc inversement proportionnelle à f(a). La mise à jour de la trace sur la décision d est réalisée comme suit

$$\tau_d = (1-\rho)\tau_d + c \sum_{a \in A} \Delta_d(a)$$

où c est une constante et $\Delta_d(a) = \begin{cases} 1/f(a) & \text{si la fourmi a a réalisé le choix d} \\ 0 & \text{sinon} \end{cases}$

Le pseudo-code d'un système de fourmi et ses caractéristiques sont donnés ci-dessous.

Système de fourmis

1. Initialiser les traces τ_d à 0 pour toute décision possible d
2. Tant qu'aucun critère d'arrêt n'est satisfait faire
3. Construire |A| solutions en tenant compte de la force gloutonne et de la trace
4. Mettre à jour les traces τ_d ainsi que la meilleure solution rencontrée;
5. Fin du tant que

Types d'Individus :	solutions admissibles obtenues à l'aide d'un algorithme constructif
Type d'évolution :	remplacement générationnel avec une population de taille constante
Structure de voisinage :	population non structurée
Sources d'information :	historique de la recherche (mémorisé dans la trace)
Irréalisabilité :	l'algorithme constructif ne produit pas de solution non admissible
Intensification :	aucune
Diversification :	aucune.

Cet algorithme peut être amélioré de diverses manières. On appelle **Optimisation par Colonies de Fourmis** (ACO pour Ant Colony Optimisation) toute variation ou extension du système de fourmis décrit ci-dessus. Le premier type d'extension consiste à réaliser une Recherche Locale sur chaque solution produite par l'algorithme constructif. L'étape 3 et la caractéristique *Intensification* deviennent donc :

3. Construire |A| solutions en tenant compte de la force gloutonne et de la trace;
   Appliquer une Recherche Locale sur chacune de ces solutions;

Intensification :	Recherche Locale

Une 2ème variation est de prendre un certain pourcentage de décisions en ne tenant compte que de la force gloutonne (les autres décisions étant prises en combinant la force gloutonne et la trace). Une 3ème variation consiste à faire une mise à jour supplémentaire des traces en tenant compte des choix ayant abouti à la meilleure solution s* rencontrée.

La **Méthode à Mémoire Adaptative** est une extension de la Recherche Tabou qui permet de réaliser automatiquement une diversification et une intensification de la recherche. Cette méthode a été proposée par Rochat et Taillard en 1995. Cette méthode fonctionne avec une mémoire centrale chargée de stocker les composantes des meilleures solutions rencontrées. Ces composantes sont combinées afin de créer de nouvelles solutions. Si la combinaison ne produit pas une solution admissible, un processus de réparation est mis en œuvre. Un algorithme de Recherche Locale est ensuite appliqué et les composantes de la solution ainsi obtenues sont considérées pour éventuellement faire partie de la mémoire centrale.
Au début de la recherche, la mémoire centrale contient des composantes provenant de solutions très diverses et le processus de combinaison aura donc tendance à créer une diversité de nouvelles solutions. Plus la recherche avance et plus la mémoire centrale aura tendance à ne mémoriser que les composantes d'un sous-ensemble très restreint de solutions. La recherche devient donc petit à petit un processus d'intensification.

Le pseudo-code et les caractéristiques de la méthode à mémoire adaptative sont donnés ci-dessous.

Méthode à Mémoire Adaptative

1. Générer un ensemble de solutions et introduire leurs composantes dans la mémoire centrale;
2. Tant qu'aucun critère d'arrêt n'est satisfait faire
3. Combiner les composantes de la mémoire centrale pour créer de nouvelles solutions;
4. Réparer si nécessaire les solutions non admissibles ainsi générées;
5. Appliquer une Recherche Locale sur chaque nouvelle solution admissible
6. Mettre à jour la mémoire centrale en ôtant certaines composantes et en y insérant de nouvelles provenant des nouvelles solutions générées à l'étape 5.
7. Fin du tant que

Types d'Individus :	composantes de solutions admissibles
Type d'évolution :	remplacement stationnaire avec une population de taille constante
Structure de voisinage :	population non structurée
Sources d'information :	au moins deux parents
Irréalisabilité :	les solutions non admissibles sont réparées
Intensification :	Recherche Locale + Intensification implicite durant les dernières itérations
Diversification :	Diversification implicite durant les premières itérations..

La tendance actuelle est d'avoir recours à des algorithmes dits **hybrides**. Il existe plusieurs types d'hybridations possibles

- on peut utiliser une Recherche Locale dans les méthodes évolutives (c'est ce que nous avons déjà fait souvent). L'avantage est que la Recherche Locale réduit le danger de passer à côté d'une solution optimale sans la voir. En règle générale, une méthode évolutive est excellente pour détecter de bonnes régions dans l'espace de recherche alors qu'une Recherche Locale explore efficacement les régions prometteuses.
- on peut exécuter en parallèle diverses métaheuristiques, voire même plusieurs fois la même métaheuristique mais avec divers paramètres. Ces processus parallèles communiquent entre eux régulièrement pour échanger de l'information sur leurs résultats partiels.
- une troisième forme d'hybridation consiste à combiner les métaheuristiques avec des méthodes exactes. Une métaheuristique peut par exemple fournir des bornes à une méthode de type branch-and-bound. Au contraire, une méthode exacte peut donner lieu à une technique efficace pour la détermination du meilleur voisin d'une solution (ce qui peut s'avérer plus judicieux que de choisir la meilleure solution parmi un échantillon de voisins).

Pour terminer ce chapitre, voici quelques principes qui permettent de guider toute personne intéressée à adapter une des méthodes décrites ci-dessus à un problème concret. Ces principes ont été rédigés sous forme de maxime dans la thèse de doctorat de Nicolas Zufferey.

Pour les méthodes de trajectoire

Maxime 1
Afin de faciliter la génération d'une solution voisine, il ne faut pas hésiter à travailler dans l'espace des solutions non admissibles ou partielles, en modifiant si besoin est la fonction objectif à optimiser

Maxime 2
À partir de n'importe quelle solution, il devrait être possible d'atteindre une solution optimale en générant une suite de solutions voisines.

Maxime 3
Afin de bien guider la recherche d'une solution optimale dans S, il faut que les solutions voisines d'une solution s ne soient pas trop différentes de s.

Maxime 4
Plutôt que d'optimiser directement la fonction objectif relative à un problème, mieux vaut optimiser une fonction objectif auxiliaire qui donne du relief à l'espace des solutions. De plus, cette fonction doit permettre d'évaluer rapidement une solution voisine, l'idéal étant d'évaluer une solution voisine en ne calculant que l'écart de valeur qui la distingue de la solution courante (*calcul incrémental*)

Maxime 5
Pour générer une solution voisine, il faut tenir compte des propriétés du problème : une solution voisine s' ne doit pas contenir les mêmes défauts que la solution courante s, même si s' est de bien moins bonne qualité que s.

Pour les méthodes évolutives

Maxime 6
Durant la phase de coopération, des informations pertinentes qui tiennent compte des propriétés du problème traité doivent être échangées

Maxime 7
Il faut disposer d'un outil d'intensification performant. Le plus souvent, cet outil est une procédure de Recherche Locale efficace.

Maxime 8
Afin de ne pas réduire à néant trop rapidement l'effet de l'opérateur de combinaison, il est important de préserver la diversité des informations contenues dans la population.

Maxime 9
L'opérateur d'échange d'information doit permettre, s'il combine de bonnes informations, de transmettre ces bonnes informations à la solution enfant.