Quelques mots sur quelques Métaheuristiques

Les métaheuristiques sont des méthodes "génériques" pour rechercher une solution approchée aux problèmes d'optimisation.

Les méthodes par voisinage

L'idée sous-jacente aux trois méthodes exposées ci-dessous est simple: on explore l'espace des solutions en partant d'un point et en se "déplaçant" de de proche en proche; elles sont donc basées sur la notion de **voisinage** d'une solution: les voisins d'une solution seront des solutions proches de celle-ci, i.e. qu'on peut obtenir par exemple en transformant à peine la solution initiale. Par exemple, dans le problème du voyageur de commerce, les voisins d'une tournée sont les tournées obtenues en supprimant deux arcs non contigus de la tournée et en reconnectant les deux sous-tournées obtenues: on a donc de l'ordre de n(n-1)/2 vosins d'une tournée, si n est le nombre de villes. Pour appliquer les méthodes ci-dessous, il faut donc avoir au préalable défini une notion de voisinage d'une solution. Bien sûr, pour le même problème, plusieurs notions de voisinage peuvent être définies.

Recherche d'un optimum local (Hill-Climbing):

On part d'une solution si possible "bonne" (par exemple donnée par une heuristique ci-dessus) et on balaie l'ensemble des voisins de cette solution; si il n'existe pas de voisin meilleur que notre solution, on a trouvé un optimum local et on arrête; sinon, on choisit le meilleur des voisins et on recommence. Une autre implémenation consiste non pas à passer au meilleur des voisins à chaque étape mais au premier meilleur voisin trouvé.

La convergence vers un optimum local pouvant être très lente, on peut éventuellement fixer un nombre de boucles maximum, si on veut limiter le temps d'exécution.

Cette méthode a l'inconvénient de "rester bloquée" dans un optimum local: une fois un optimum local trouvé, on s'arrête, même si ce n'est pas l'optimum global. Selon le "paysage" des solutions, l'optimum local peut être très bon ou très mauvais par rapport à l'optimum global. Si la solution de départ est donnée par une heuristique déterministe, l'algo sera déterministe. Si elle est tirée au hasard, on a un algo non déterministe et donc plusieyrs exécutions différenbtes sur la même instance pourront donner des solutions différentes et de qualités différentes..

La notion de voisinage est primordiale! Si les voisins sont très nombreux, on a de fortes chances de trouver l'optimum global mais visiter un voisinage peut être très (trop) long: on visitera une grande partie de l'espace des solutions! Si le voisinage est très restreint, on risque fort de rester bloqué dans un optimum local de "mauvaise qualité": le choix de la notion de voisinage est un compromis entre efficacité et qualité. Les deux méthodes qui suivent peuvent être vues comme des méthodes de recherche locale avancées.

La méthode "Taboue":

A chaque étape, on recherche le meilleur voisin, mais on limite la recherche aux voisins non tabous: un voisin est à priori tabou si on a exploré cette solution durant les N précédentes itérations. La maintenance d'une liste taboue de voisins étant souvent trop coûteuse, on se borne souvent à stocker les transformations effectuées.

A chaque itération, on choisit le meilleur voisin (ou un meilleur voisin selon le cas) correspondant à une transformation non taboue; on effectue cette transformation, on la place dans la liste (file) taboue et on élimine la plus ancienne transformation de cette liste. si la file est pleine. De plus, si la solution actuelle est la meilleure trouvée depuis le début, on la stocke. On s'arrête soit après un nombre fixé d'itérations, soit après un nombre fixé d'étapes n'ayant pas amélioré la solution. Une fois la notion de voisin fixée, cette solution pourra donc être paramétrée par la longueur de la liste taboue.

La méthode "taboue" peut donc être vue comme une généralisation de la recherche d'optimum local (si N=0, on retombe dans le cas du hill-climbing); l'idée est de permettre de s'"échapper" d'un optimum local grâce à la liste taboue.

Le recuit simulé:

La méthode (simulated annealing) est inspirée de la physique: le recuit est utilisé pour obtenir des états de faible énergie; il est utilisé par exemple dans la fabrication du verre.

On part d'une solution quelconque (à priori, il n'est pas forcément intéressant de partir d'une "bonne" solution). A chaque étape, on tire au hasard une solution voisine. Si elle est meilleure, on l'adopte; sinon on calcule l'augmentation de coût Δ; on va adopter cette solution avec une probabilité liée à cette augmentation: plus le coût augmente, plus la probabilité de la garder sera faible. Mais cette probabilité dépend aussi du temps écoulé depuis le lancement de l'algorithme: au début, on accepte facilement un changement qui produit une solution plus coûteuse à la fin on l'accepte très difficilement. Plus précisément on calcule $accept = E^{-\Delta/T}$, T étant une quantité que nous appelerons *la température* (nous y reviendrons). On tire au hasard un réel p entre 0 et 1: si p est inférieur à *accept*, on adopte la nouvelle solution, sinon on garde l'ancienne. Le schéma général est donc:

```
init(sol);
T:=T0;
loop
  sol':=voisin_au_hasard(sol)
  if meilleur (sol',sol) then sol:=sol';
  else
    Delta=cout(sol') - cout(sol); --surcoût de sol'
    tirerproba(p); --p réel dans [0,1]
    if p<= e ^ (-Delta/kT)
    then sol:=sol';
    --on accepte sol' avec proba accept
    --sinon on ne change pas
    end if;
  T:=refroidissement(T);
end loop;
```

Il faut donc choisir une température initiale, et pour assurer l'arrêt, une façon de diminuer la température au cours de l'exécution.

Le choix de la température initiale: La température initiale doit être assez élevée pour que *accept* soit assez grand au départ même si l'augmentation de coût est grande. C'est un paramètre fixé par l'utilisateur.

Le refroidissement: Pour diminuer T, on peut par exemple multiplier T par un réel inférieur à 1 à chaque fois. Plus ce réel est proche de 1, plus le refroidissement est lent. Ce "coefficient de refroidissement" est lui aussi un paramètre fixé lors de l'utilisation. On peut aussi raffiner: on effectue plusieurs cycles de recuit simulé, en repartant à chaque fois de la solution trouvée au cycle précédent. Une difficulté de cette heuristique réside dans le réglage du refroidissement; celui-ci pourra être choisi différemment selon l'instance du problème considérée.

Les algorithmes génétiques

Les algorithmes génétiques, créés par J. Holland (75) puis développés par David Goldberg, sont basés sur deux principes de génétique: la survie des individus les mieux adaptés et la recombinaison génétique.

Le principe de base est de simuler l'évolution d'une population de solutions avec les règles citées ci-dessus en vue d'obtenir une solution ou un ensemble de solutions les plus adaptées. "A chaque génération, un nouvel ensemble de créatures artificielles est créé en utilisant des parties des meilleures solutions précédentes avec éventuellement des parties innovatrices." (Goldberg)

Le schéma d'un algorithme génétique est donc:

```
créer la population initiale --par exemple au hasard
boucle
  sélectionner les individus à utiliser pour créer la prochaine population
  --sélection des plus adaptés
  hybridation des invidus sélectionnés à l'aide des opérateurs de croisement
  -- crossover
  --production de chromososmes à partir de ceux des parents
  éventuellement effectuer des mutations aléatoires
  -- modification d'un gène (innovation)
```

```
fin boucle
```

L'arrêt se fera par exemple quand la population ne s'"améliore" plus de manière significtaive ou après un nombre donné de générations.

Pour la mise en oeuvre il faut donc:

.choisir le codage (souvent binaire) d'une solution

.fixer la taille d'une population

.définir la fonction d'adaptation (fitness)

.définir les opérateurs de sélection, le principe étant: meilleure est notre adaptation, plus on a de chances d'être sélectionné

.définir les opérateurs de croisement

.définir éventuellement les opérateurs de mutation.

L'un des problèmes de la méthode est encore la convergence prématurée vers une population non optimale.

D'autres métaheuristiques inspirées par le monde du vivant ont été introduites, comme les colonies de fourmis,