UNIVERSITE MOHAMED KHIDER – BISKRA
FACULTE DES SCIENCES EXACTES, DES SCIENCES DE LA NATURE ET DE LA VIE
DEPARTEMENT D'INFORMATIQUE

Complexité et Optimisation

Support de cours et TD

Master d'informatique

Version amélioée

Réalisé par : Dr. S. SLATNIA

05/01/2016

Chapitre 5.
MÉTHODES APPROCHÉES, HEURISTIQUES SPECIALISÉES

1. Méthodes approchées

1.1. Algorithme Glouton

Pour les problèmes de grandes tailles, pas de temps de calculs raisonnables avec les méthodes exactes. Il faut donc, rechercher de bonnes solutions approchées.

Définition 3.
Un algorithme Glouton ou avide (en anglais, **Greedy**) est parfois intéressant pour les problèmes d'optimisation.
Un algorithme Glouton se déroule selon les phases à chaque phase : on espère que choisir un optimum local à chaque phase convergera vers une solution optimale globale «une solution locale → une solution globale».

Un algorithme avide emploie de simples stratégies, faciles à implanter et qui requièrent un minimum de ressources.
Construction d'une solution réalisable en se ramenant à une suite de décisions qu'on prend à chaque fois au mieux en fonction d'un critère local sans remettre en question les décisions déjà prises, la solution obtenue est approchée.

Avantage : algorithmes simples à implémenter.
Inconvénients : solutions approchées obtenues plus ou moins bonnes, critère local.

Exemple 1 : Placement optimal de pièces 2D (Bin Packing).

On dispose de plaques rectangulaires toutes identiques dans lesquelles on veut placer des pièces rectangulaires sans chevauchement. Les pièces à placer ont des dimensions différentes.

- On veut trouver le placement pour minimiser le nombre de plaques utilisées.

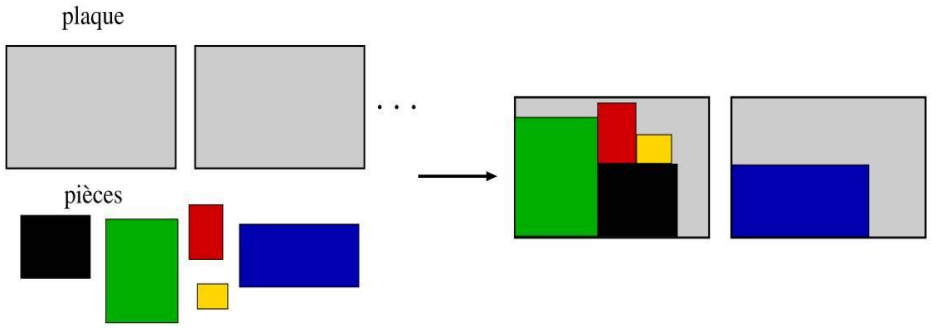

Fig 2. Exemple d'un placement optimal de pièces 2D (Bin Packing)

Algorithme glouton : trier les pièces en fonction de leur taille et placer d'abord les pièces les plus grandes.

2. Quelques Méthodes heuristiques

Définitions

Heuristiques

Une heuristique est une technique qui améliore l'efficacité d'un processus de recherche, en sacrifiant éventuellement l'exactitude ou l'optimalité de la solution.

Pour des problèmes d'optimisation (NP-complets) où la recherche d'une solution exacte (optimale) est difficile (coût exponentiel), on peut se contenter sur :

- une solution satisfaisante donnée par une heuristique avec un coût plus faible**.**

Règles empiriques simples qui ne sont pas basées sur l'analyse scientifique différents de l'algorithme, Elles sont basées sur :

- L'expérience et les résultats déjà obtenus et
- L'analogie pour optimiser les recherches suivantes.

⇨ La solution non optimale mais une solution approchée**.**

2.1. Algorithme Recherche Locale

Partir d'une solution sinon approchée du moins potentiellement bonne et d'essayer de l'améliorer itérativement [35].

⇨ Pour améliorer une solution on ne fait que de légers changements

On parle de changement local (solution voisine).
Relancer la méthode plusieurs fois en changeant le point de départ pour avoir plus de couverture.

```
A* ← creer une solution()
for all t=1 a max essais do
    A ← nouvelle solution()
    for all m=1 a max mouvements do
        A' ← choisir voisin(A)
        d ← (f(A')-f(A))
        if acceptable(d) then   A ← A'
    end for
    if f(A*)>f(A) then  A* ← A
end for
retourner A*,f(A*)
f : fonction à optimiser
```

L'insuffisance des méthodes de recherche locale

- incapables de progresser au-delà du premier optimum local rencontré

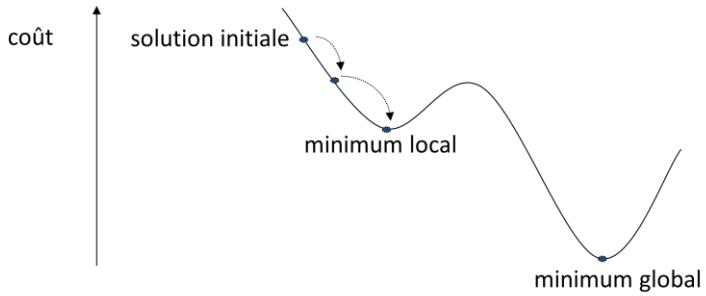

Fig 3. Problème de minimisation

2.2. Algorithme A*

L'algorithme A* a été créé pour que la première solution trouvée soit l'une des meilleures. Cet algorithme a été proposé pour la première fois par Peter E. Hart (en) [36], Nils John Nilsson et Bertram Raphael en 1968. Il s'agit d'une extension de l'algorithme de Dijkstra de 1959. Il s'agit d'un algorithme heuristique de programmation dynamique qui fournit généralement une solution approchée.

L'algorithme A* est un algorithme de recherche de chemin dans un graphe entre un nœud initial et un nœud final. Il utilise une **évaluation heuristique** sur chaque

nœud pour estimer le meilleur chemin y passant, et visite ensuite les nœuds par ordre de cette évaluation heuristique.

→ **Algorithme simple,**
→ **Ne nécessitant pas de prétraitement,** et
→ **Ne consommant que peu de mémoire.**

Propriété

1. Garantit de toujours trouver le chemin le plus court à un but s'appelle « algorithme admissible ».
2. Si A* utilise une heuristique qui ne surestime jamais la distance (ou plus généralement le coût) du but, A* peut être avéré admissible.
3. Une heuristique qui rend A* admissible est elle-même appelée « heuristique admissible ».
4. Si l'évaluation renvoie simplement toujours zéro, qui n'est jamais une surestimation, alors, A* exécutera une implémentation possible de l'algorithme de Dijkstra et trouvera toujours la solution optimale.
5. La meilleure heuristique, bien qu'habituellement impraticable pour calculer, est la distance minimale réelle (ou plus généralement le coût réel) au but.
   Un exemple d'une heuristique admissible pratique est la distance à vol d'oiseau du but sur la carte.
6. On peut démontrer que A* ne considère pas plus de nœuds que tous les autres algorithmes admissibles de recherche, à condition que l'algorithme alternatif n'ait pas une évaluation heuristique plus précise. Dans ce cas, A* est l'algorithme informatique le plus efficace garantissant de trouver le chemin le plus court.

Si la fonction choix fournit un sommet z vérifiant [37]:
$d(z)+h(z)=\min_{y \in OUVERTS}\{d(y)+h(y)\}$ où h(y) est une information "heuristique" sur la distance qui reste à parcourir.
et si l'instruction :"Ne rien faire" de l'algorithme de Dijkstra est remplacée par:
si d(z)+w(z,y)<d(y) **alors**
  Parent(y)←z
  d(y) ←d(z)+w(z,y)
  Ajout(y,OUVERTS)
finsi

On suppose que cette fonction h(y) est une information disponible en chaque sommet du graphe. L'usage de cet algorithme est adapté au cas où l'on cherche un plus court chemin d'un sommet x0 à un sommet t. La valeur de h(x) pour un sommet x donné, étant une estimation de la distance qu'il reste à parcourir pour aller de x à t.

Dans les deux instanciations précédentes, le goulot d'étranglement de complexité provient de la gestion de l'ensemble des OUVERTS. il faut utiliser une structure de données qui permette le calcul du minimum en O(logn) ou mieux.

Algorithme très utilisé pour sa rapidité (jeux vidéo, ...). C'est un algorithme de recherche d'un plus court chemin dans un graphe allant de x_0 à x_F. Il est basé sur une évaluation heuristique à chaque sommet pour estimer le meilleur chemin qui y passe. Les sommets sont parcourus en fonction de cette évaluation heuristique : on retient le sommet où l'évaluation est la plus petite.

On construit deux listes :
S liste ouverte : contient les sommets (successeurs) à examiner.
S liste fermée : contient tous les sommets déjà examinés, qui appartiennent au chemin solution.
On commence par le sommet $x=x_0$

1. On regarde tous les successeurs (voisins) x' de x
2. Si un sommet x' n'est pas dans la liste ouverte S ni dans la liste fermée $\bar{\bar{S}}$, alors on l'ajoute dans S.
3. Si un sommet x' est déjà dans S ou bien dans $\bar{\bar{S}}$ et si x' a un nouveau coût $\emptyset$ plus petit, alors on met à jour le coût.
4. De plus, si $x' \in \bar{\bar{S}}$ alors on l'enlève de $\bar{\bar{S}}$ et on l'ajoute à S.
5. On détermine le meilleur sommet x de toute la liste ouverte S (si $S=\emptyset$ alors arret, pas de solution).
6. On met x dans la liste fermée $\bar{\bar{S}}$ et on le supprime de la liste S.
7. On itère avec le sommet courant x (retour à 1).

2.3. Algorithme Hill Climbing

La **méthode *hill-climbing*** est une méthode d'optimisation permettant de trouver un optimum local parmi un ensemble de configurations.

Le *hill-climbing* est une méthode générale qui prend en entrée trois objets :

- Une configuration,
- Une fonction qui pour chaque configuration donne un ensemble de configurations voisines, et
- Une fonction-objectif qui permet d'évaluer chaque configuration.

La méthode consiste simplement à partir de la configuration initiale, à évaluer les solutions voisines, et à choisir la meilleure de celles-ci, et à recommencer l'opération

jusqu'à arriver à une position meilleure que les positions voisines (un optimum local) [36].

Le *hill-climbing* est une **méthode,**

1. **Itérative**
2. **Locale, et**
3. **Gloutonne.**

Elle est similaire à la méthode de la **descente de gradient** mais est plutôt utilisée dans un contexte discret où le gradient n'est pas défini. Certains auteurs donnent une définition plus restreinte du *hill-climbing*, où chaque configuration correspond à un vecteur, et chaque voisinage considéré ne fait varier qu'une seule coordonnée.

3. Méta-heuristiques

Algorithmes d'optimisation généralement de type stochastique combinant plusieurs approches heuristiques.

L'intérêt et domaines d'applications des méthodes heuristiques Algorithme A∗, Recuit simulé et Algorithmes génétiques est dans les problèmes d'optimisation de la forme :

$$\min_{x \in X} \mathbf{f(x)}$$
$$\begin{cases} \text{sous des contraintes} \\ \mathbf{g(x)} \leq \mathbf{b} \end{cases}$$

Fig 4. La fonction d'optimisation

La fonction f peut être optimisation multi-objectifs et la fonction objectif f et les contraintes g sont non linéaires.

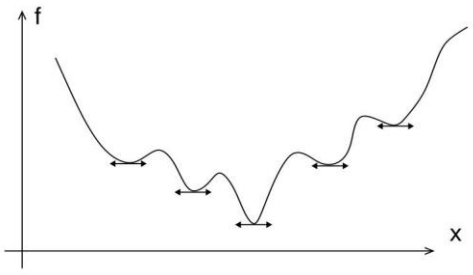

Fig 5. Exemple de fonction d'optimisation

Problème : Plusieurs minima locaux possibles ⇒ les méthodes classiques d'optimisation non-linéaire couteuse et incapable de capturer la solution globale.

Solution : Méthodes méta-heuristiques, capacité à s'extraire d'un minimum local pour aller vers un minimum global.

3.1 **Algorithme génétique**

Les AGs forment une famille très intéressante d'algorithmes d'optimisation. Ils ont été développés en première par John Holland à l'université de Michigan ("Adaptation in Natural and Artificial Systems", 1975)**.**

Définitions.

- **Individu :** une solution potentielle du problème (un élément de l'espace de recherche).
- **Génotype ou chromosome** : c'est une autre façon de dire "individu".
- **Gène** : un chromosome est composé de gènes. Dans le codage binaire, un gène vaut soit 0 soit 1.
- **Phénotype** : chaque génotype représente une solution potentielle à un problème d'optimisation. La valeur de cette solution potentielle est appelée le phénotype.

Le principe des AGs est de coder chaque solution potentielle d'un problème par un "*chromosome*". L'ensemble des chromosomes ou "individus" forme alors la "population" qui va être amenée à évoluer au cours du temps. Une "génération" est l'état de la population à un instant t. La population évolue au cours des générations en suivant des lois de sélection, de croisement et de mutation. En informatique, nous parlons d'opérateur génétique.

Dans AG de base, il existe trois opérateurs qui sont la sélection, le croisement et la mutation.

3.1.1. **Les opérateurs génétiques**

A. **La sélection :** l'opérateur de sélection donne plus de chance aux "bons" individus de participer à la génération suivante en fonction d'un certain critère, la fitness.
B. **Le croisement** : l'objectif de croisement est de combiner des chromosomes à partir de parents présentant des qualités pour obtenir de meilleurs individus. Le croisement consiste à mélanger, au niveau du génome, des gènes de deux individus parents pour générer deux enfants. La taille de la population reste constante.

- **Le croisement par coupure :** le génome des deux parents est sectionné en un même endroit, choisi le plus souvent aléatoirement. Le nombre de points de

coupure est un paramètre de l'opérateur. Les fragments sont recombinés pour constituer les enfants comme l'illustre la figure 16.

- **Le croisement uniforme :** dans le cas du croisement uniforme, un masque binaire aléatoire de même longueur que le génome d'un chromosome utilisé. Si le bit i du masque est à 1, alors l'enfant 1 prend le bit i du parent 1, et l'enfant 2 prend le bit i du parent 2. Si par contre c'est un 0, c'est l'inverse qui est effectué.

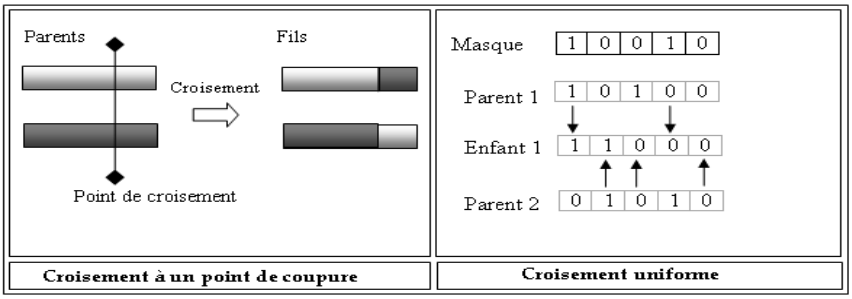

Fig 6. L'opérateur de croisement.

C. La mutation : elle effectue une modification des gènes des chromosomes des enfants. En fonction du problème à résoudre, à chaque individu est associé un 'degré d'adaptation à son environnement', appelé fitness. Après croisement et mutation, une nouvelle génération est construite en conservant les individus de la population précédente ayant une propriété de fitness particulière, jusqu'à. la convergence vers une solution optimale.

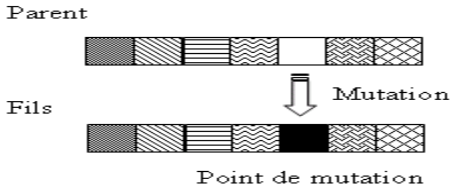

Fig 7. L'opérateur de mutation.

3.1.2. **Le processus d'optimisation**

1. Initialisation : générer un ensemble de solutions initiales.
2. Phase de mise à jour.
   A. Evolution :
   - Sélection : choisir avec une probabilité proportionnelle à leur qualité une liste d'individus.
   - Reproduction : générer à partir de cette liste de nouveaux individus à l'aide des opérateurs génétiques.

- Remplacement : éliminer avec une probabilité inversement proportionnelle à leur qualité certains individus.

B. réactualisation de la meilleure solution.

C. allez en A tant que Nombre de Générations ≤ Valeur pré-déterminée.

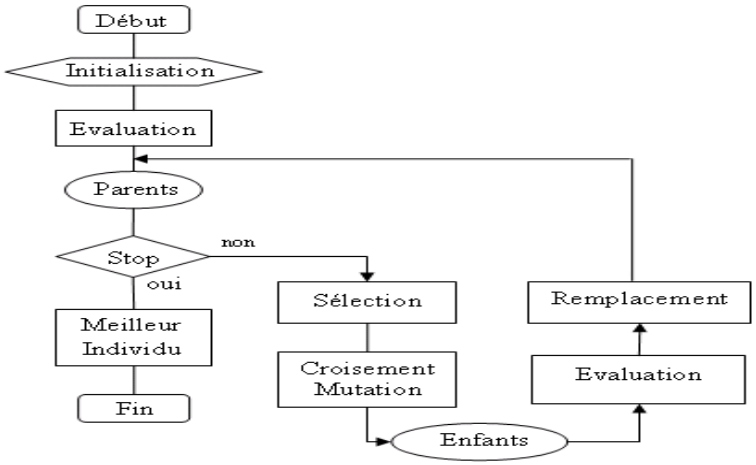

Fig 8. Principe de base d'un AG.

- La fonction objective f peut être non-linéaire.
- Temps de calculs souvent importants.
- Difficultés de mise en oeuvre.
- Fournit généralement une solution approchée.
- Problèmes d'extréma locaux.