INF4705 — Analyse et conception d'algorithmes

CHAPITRE 8 : ALGORITHMES MÉTAHEURISTIQUES

Ce chapitre et le suivant présentent avant tout une *caractérisation* des algorithmes selon la qualité des solutions qu'ils apportent.

Pour certains problèmes d'optimisation, il n'existe pas (encore) d'algorithme donnant la solution optimale (la meilleure parmi l'ensemble des solutions possibles) qui soit vraiment efficace (p.ex. le problème du sac à dos : ni vorace, ni programmation dynamique, ni séparation et évaluation progressive arrivent à résoudre de façon optimale des exemplaires de grande taille). Puisqu'on veut les résoudre, il faut alors se contenter d'un algorithme efficace qui donnera une solution de bonne qualité sans être nécessairement la meilleure.

Nous distinguerons trois types d'algorithme :

exact : donne une solution de la meilleure qualité possible (optimale).

d'approximation : donne une solution qui est garantie d'être à au plus une "distance" donnée de la meilleure solution (p.ex. à 25% de l'optimum).

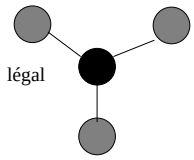
heuristique : donne une solution sans aucune garantie de sa qualité (p.ex. algorithme vorace pour sac à dos sans fragmentation) ou possiblement ne donne pas de solution du tout (p.ex. algorithme vorace pour faire de la monnaie au Portugal).

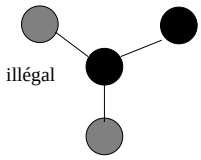
Dans ce chapitre, nous discuterons aussi de certaines techniques de conception d'algorithmes : les heuristiques d'amélioration locale et les algorithmes métaheuristiques.

8.1 Heuristiques constructives

Comme leur nom l'indique, elles bâtissent une solution à partir de zéro.

Exemple 1 *Coloriage de graphe*
Utiliser le moins de couleurs possibles pour colorier tous les sommets d'un graphe non-orienté t.q. deux sommets adjacents sont de couleur différente.
Applications : confection d'horaires, allocation de fréquences en téléphonie cellulaire, etc.

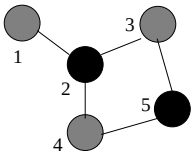


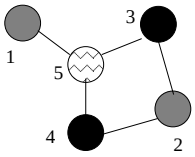


Algorithme vorace naïf :

$F \leftarrow$ *les sommets, dans un ordre arbitraire ;*
tant que *F n'est pas vide*
    retirer le prochain sommet v de F ;
    colorier v avec la première couleur disponible ;

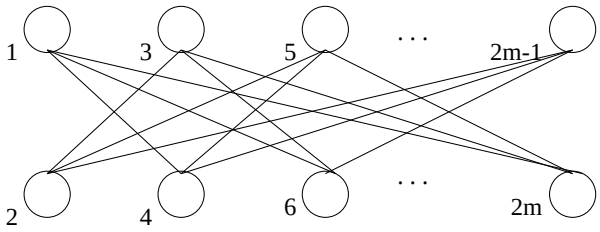
p.ex.





complexité : *on peut l'implanter pour prendre un temps dans* $\mathcal{O}(n^2)$.

Il s'agit bien d'une heuristique car la solution obtenue peut être arbitrairement mauvaise. p.ex. considérons le graphe biparti quasi-complet :



On peut toujours colorier un graphe biparti avec 2 couleurs.
Or ici, en utilisant l'ordre indiqué, l'algo. vorace naïf prend m *couleurs, pour un* m *arbitrairement grand.*

Algorithme DSATUR (Brélaz) :
idée : s'occuper d'abord des sommets les plus contraints.

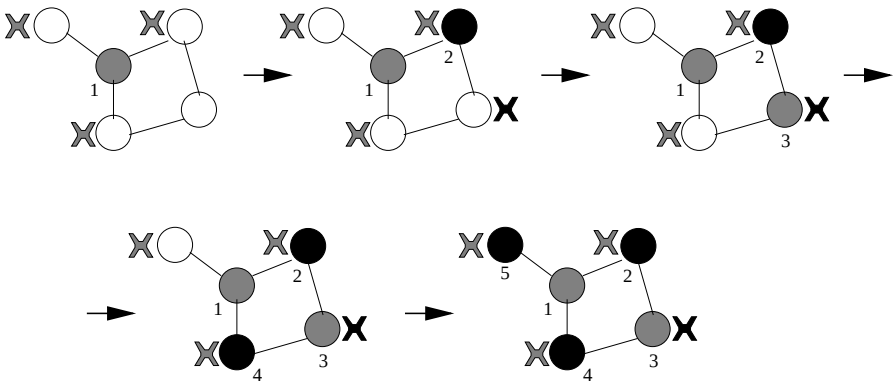
$E \leftarrow \{$ *sommets* $\}$;

tant que *E n'est pas vide*

retirer de E le sommet v qui est adjacent au plus grand nombre de couleurs (différentes), en utilisant au besoin le degré maximum dans le sous-graphe non colorié comme bris d'égalité ;

colorier v avec la première couleur disponible ;

p.ex.



complexité : *temps* $\in \mathcal{O}(n^2)$.

L'heuristique DSATUR donne la solution optimale sur un graphe biparti ; par contre, elle ne donne pas de garantie en général sur la qualité de la solution obtenue.

8.2 Heuristiques d'amélioration locale

À partir d'une solution initiale s_0, appliquer une succession de modifications locales afin d'arriver à une meilleure solution s_t après t itérations. Soit V_s l'ensemble des solutions voisines de s (selon une certaine structure de voisinage qu'il faut définir). Soit f la fonction qui, appliquée à une solution, en retourne le coût et supposons finalement que nous recherchons la solution qui maximise f.

Schéma général :

Démarrer avec une certaine solution s_0 ;

$i \leftarrow 0$;

tant que $\exists\, s \in V_{s_i}$ telle que $f(s) > f(s_i)$

$i \leftarrow i + 1$;

$s_i \leftarrow s$;

retourner s_i ;

Rien ne nous garantit qu'on obtiendra ainsi la meilleure solution (au plus un optimum local).

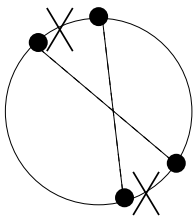
Exemple 2 *Problème d'affectation*
On doit affecter chacun des n magasins d'une chaîne à un entrepôt de ravitaillement parmi m, ou encore affecter chacune des n cellules d'un réseau sans fil à un commutateur parmi m, de manière à minimiser la somme des coûts de liaison individuels.

Un premier voisinage d'intérêt, très simple, consiste à changer une affectation : pour une solution s, l'ensemble de ses voisins V_s s'obtient en modifiant une seule affectation dans s. La taille de ce voisinage est dans $\Theta(nm)$, ce qui reflète la complexité en pire case d'une itération du schéma général.

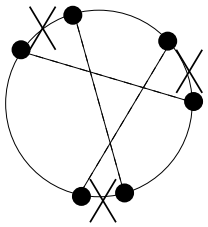
Si les entrepôts (commutateurs) ont une certaine capacité et qu'ils sont particulièrement chargés dans la solution courante, il pourrait être difficile d'ajouter un magasin (cellule) supplémentaire par un changement d'affectation. Un second voisinage, échangeant deux magasins (cellules) entre deux entrepôts (commutateurs), sera plus approprié. La taille de ce voisinage est dans $\Theta(n^2)$.

Exemple 3 *Voyageur de commerce*
Faire la tournée d'un ensemble de villes en minimisant la distance totale parcourue.

Une modification locale qui a du succès pour ce problème retire un certain petit nombre d'arêtes de la tournée et en ajoute autant de nouvelles afin de reconnecter la tournée.
Par exemple,



2-échange



3-échange

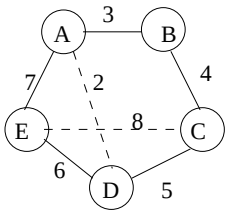
Algorithme k-opt :

1. *Démarrer avec une certaine tournée τ ;*
2. **tant que** *on peut remplacer k arêtes $a_1, \ldots, a_k$ de τ par k autres $a'_1, \ldots, a'_k$ reformant une tournée moins longue*

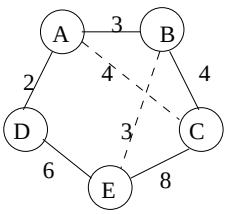
   $\tau \leftarrow \tau - \{a_1, \ldots, a_k\} + \{a'_1, \ldots, a'_k\}$;
3. **retourner** τ ;

p.ex., un 2-opt avec la matrice de distances suivante :

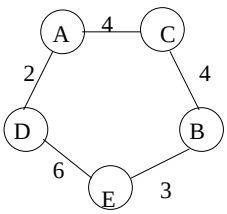
	B	*C*	*D*	*E*
A	*3*	*4*	*2*	*7*
B		*4*	*5*	*3*
C			*5*	*8*
D				*6*



distance totale = 25



distance totale = 23



distance totale = 19

complexité : temps $\in \mathcal{O}(n^k)$ pour chaque tour de boucle.

8.3 Métaheuristiques

Il s'agit d'une foule de méthodes génériques dont l'intuition de départ emprunte souvent au monde physique et même animal.

8.3.1 métaheuristiques basées sur les trajectoires

Il existe des méthodes d'amélioration locale plus sophistiquées qui permettent de quitter un optimum local, sans bien sûr garantir qu'on trouvera un optimum global. Ce sont également des méthodes itératives. En voici brièvement quelques-unes.

recuit simulé

Cette approche s'inspire d'une technique visant à produire une forte structure cristalline chez une substance en la chauffant puis en la refroidissant lentement. Par analogie avec l'origine thermodynamique de la méthode, on définit une *température* θ_i à l'itération i, qui décroit par paliers selon un horaire de refroidissement.

```
...
tant que  critère d'arrêt
    choisir s dans V_{s_i} de façon aléatoire ;
    si f(s) ≥ f(s_i) alors
        s_{i+1} ← s ;
    sinon
        s_{i+1} ← s avec probabilité p_i ;
        s_{i+1} ← s_i avec probabilité 1 − p_i ;
    i ← i + 1 ;
...
```

où p_i croît inversement avec i et $f(s_i) - f(s)$. On définit souvent la probabilité requise par

$$p_i = e^{-(f(s_i)-f(s))/\theta_i}.$$

Parmi les critères d'arrêt les plus courants :
- un nombre limite d'itérations,
- un nombre limite d'itérations sans avoir changé de solution,
- un nombre limite d'itérations sans avoir amélioré la valeur de la solution d'un certain pourcentage.

On accepte donc toujours une nouvelle solution si elle améliore la solution courante et dans le cas contraire on l'accepte parfois si elle n'est pas trop mauvaise mais de moins en moins souvent. Ceci permet d'échapper à un optimum local.

recherche tabou

Lorsqu'une solution est choisie, elle devient taboue pendant un certain nombre d'itérations. Pour ce faire, on définit un ensemble T de solutions taboues.

```
...
T ← ∅ ;
tant que critère d'arrêt
    choisir s ∈ V_{s_i} \ T qui maximise f ;
    i ← i + 1 ;
    s_i ← s ;
    retirer de T certaines solutions qui ont "fait leur temps" ;
    T ← T ∪ {s} ;
...
```

Les critères d'arrêt sont semblables à ceux du recuit simulé. Ici on échappe à un optimum local en acceptant des solutions voisines moins bonnes que la solution courante car on a potentiellement $f(s) < f(s_i)$. L'ensemble tabou sert à éviter qu'on retourne simplement à la solution précédente dès la prochaine itération. Il faut aussi définir la durée du statut tabou (qui peut être aléatoire).

recherche à voisinage variable

On définit pas un mais plusieurs voisinages $V_s^1, \ldots V_s^k$ complémentaires. Il y a plusieurs variations sur le thème ; en voici une :

```
...
j ← 1 ;
tant que critère d'arrêt
    tant que ∃ s ∈ V_{s_i}^j telle que f(s) > f(s_i)
        i ← i + 1 ;
        s_i ← s ;
    j ← 1 + (j mod k) ;
...
```

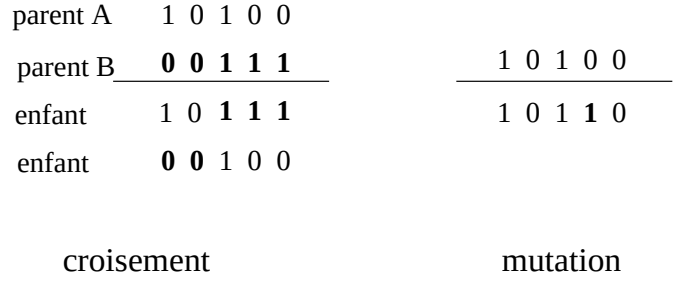
Lorsqu'une certaine structure de voisinage nous a mené à un optimum local, on passe tout simplement à une autre structure de voisinage afin de s'en échapper.

8.3.2 métaheuristiques basées sur les populations

On accumule ici un bassin de solutions que l'on combine, modifie, dissèque.

algorithmes génétiques

L'inspiration provient ici de la génétique. Plutôt que de maintenir une seule solution courante, on maintient toute une population de solutions appelées *chromosomes*. À chaque itération, plusieurs chromosomes sont ajoutés et autant sont retirés de la population. Deux opérateurs sont appliqués aux chromosomes afin d'en créer de nouveaux :



Les meilleurs chromosomes (selon la fonction de coût) servent de parents alors que les pires seront éliminés. Il s'opère donc un semblant de sélection naturelle.

colonies de fourmis

Basée sur l'analogie d'une colonie de fourmis cherchant de la nourriture et laissant une trace de phéromone au retour d'une exploration fructueuse.