

Heuristiques

Graphes et Réseaux

16 avril 2008



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

MATTHIEU PLUMETTAZ



Colorations

Une *coloration* est une attribution d'une couleur à chaque sommet.

Une coloration est dite *légitime* si aucune arête n'a la même couleur à ses deux extrémités.

Une coloration est dite *partielle* si une partie des sommets n'a aucune couleur.



GCP et k-GCP

- *Le GCP* (**G**raph **C**oloring **P**roblem) consiste à chercher une coloration légale minimisant le nombre de couleurs utilisées.
- *Le k-GCP* consiste à trouver une coloration admissible utilisant k couleurs.



Que faire lorsque l'on veut résoudre un problème difficile?

- Coloration de graphe
- Voyageur de commerce
- Atelier ouvert avec >2 machines
- ...



Énumération

Générer toutes les solutions possibles pour trouver la meilleure?

Pour le voyageur de commerce, il y a $\frac{1}{2} * (n-1)!$ solutions à évaluer ...

→ Impossible en pratique



Heuristique

Une *heuristique* est une méthode de recherche pour déterminer une bonne solution à un problème en un temps raisonnable.

Bonne solution?

Temps raisonnable?



Espace des solutions

L'espace S des solutions représente l'ensemble de toutes les solutions à un problème.



Heuristiques constructives

Une *heuristique constructive* est un algorithme qui construit pas à pas une solution.

- A chaque itération, on complète la solution partielle trouvée jusque là.
- A chaque étape, on fait des choix selon des *règles spécifiques*.



L'algorithme Glouton

L'algorithme *glouton* prend à chaque itération la décision qui maximise la valeur actuelle de la fonction objectif.

C'est un algorithme générique mais dont les performances dépendent beaucoup du problème.



DSATUR

DSATUR colore à chaque itération le sommet avec la plus grande saturation de couleurs.



Heuristiques constructives

- Facile à mettre en place
- En général, rapide d'exécution
- Mauvaises performances
→ améliorer la solution obtenue en la modifiant légèrement.



Heuristiques de Recherche Locale

Une *heuristique de recherche locale* (**Local Search**) change à chaque itération la solution courante en la modifiant légèrement.



Voisinage et mouvement

- Soit une solution s , on dit que s' est une *solution voisine* de s , si on peut obtenir s' en modifiant légèrement s .
- On dit aussi qu'on peut passer de s à s' en effectuant un *mouvement*.
- Le *voisinage* $V(s)$ de s est l'ensemble des solutions voisines de s .



Recherche Locale

- Construire une solution initiale s
- Poser $s^* = s$
- Tant qu'aucun critère d'arrêt est satisfait faire:
 - Choisir une solution s' dans $V(s)$ (choix d'un voisin)
 - Si $f(s') < f(s^*)$, poser $s^* = s'$ (mise à jour du record)
 - Poser $s = s'$ (exécution du mouvement)
- Retourner s^*



Principales heuristiques de recherche locale

- Descente
- Tabou
- Recuit simulé
- Recherche à voisinages variables



Heuristique de descente

- On choisit à chaque itération la meilleure solution voisine.
- On arrête l'algorithme lorsque aucun mouvement ne peut améliorer la solution actuelle.



Heuristique de descente

- Construire une solution initiale s
- Tant qu'il existe \hat{s} dans $V(s)$ t.q. $f(\hat{s}) < f(s)$:
 - Déterminer la meilleure solution s' de $V(s)$
 - $s = s'$
- Retourner s



Heuristique de descente

- Que se passe-t-il dans un minimum locale?
- Comment contourner ce problème?



Algorithme Tabou

Objectif : Eviter de rester bloquer dans un minimum local.

On veut donc:

- permettre de « remonter » une pente
- empêcher de revenir en arrière



Liste tabou (mouvements)

- On interdit d'effectuer l'inverse des mouvements que l'on vient de faire.
- On tient à jour une liste T contenant l'inverse des t derniers mouvements effectués. L'ensemble T est appelé *liste tabou* et les solutions de T sont les *mouvements tabous*.



Liste tabou (solutions)

- On interdit de retourner dans une solution qu'on vient de visiter.
- On tient à jour une liste T des t dernières solutions visitées. L'ensemble T est appelé *liste tabou* et les solutions de T sont les *solutions tabous*.



Liste tabou (solutions)

- On supprime certaines solutions du voisinage.
- On choisit le meilleur mouvement de $V(s)-T$



Critère d'arrêt d'un algorithme tabou

- On effectue un nombre fixe d'itérations
- On s'arrête après avoir fait p itérations sans améliorer la meilleure solution
- On s'arrête après un temps fixe d'exécution.



Algorithme tabou

- Construire une solution initiale s dans S
- Poser $s^* = s$
- Tant qu'aucun critère d'arrêt est satisfait faire:
 - Déterminer la meilleure solution s' de $V(s)-T$
 - Si $f(s') < f(s^*)$, poser $s^* = s'$
 - Poser $s = s'$
 - Mettre à jour T
- Retourner s^*



Ingrédients d'une recherche local

- Espace de solutions S
- Fonction objectif f
- Structure de voisinage / mouvement

- Critère d'arrêt
- Heuristique constructive
- Structure de la liste tabou
- Valeur pour la taille de la liste tabou



Algorithme tabou

- Construire une solution initiale s dans S
- Poser $s^* = s$
- Tant qu'aucun critère d'arrêt est satisfait faire:
 - Déterminer la meilleure solution s' de $V(s)$ - T
 - Si $f(s') < f(s^*)$, poser $s^* = s'$
 - Poser $s = s'$
 - Mettre à jour T
- Retourner s^*



Heuristiques évolutives

Idée : Utiliser un ensemble de solutions qui évoluent plutôt qu'une unique solution pour rendre l'algorithme plus robuste.



Heuristiques évolutives

- Une *population* est un ensemble de solutions.
- Chaque élément d'une population est appelé un *individu*.
- On parle de *génération* à la place d'itération dans les heuristiques évolutives.



Heuristiques évolutives

- Génétiques
- Fourmis
- Mémoire adaptative



Heuristique génétique

- S'inspire de la théorie de l'évolution
- Des individus parents vont engendrer des individus enfants



Heuristique génétique

- Générer une population initiale P
- Tant qu'aucun critère d'arrêt est satisfait faire:
 - Sélectionner des individus parents dans P
 - Produire des individus enfants par croisement des individus parents
 - Faire muter certains individus
 - Sélectionner les survivants qui composeront la population de la prochaine génération

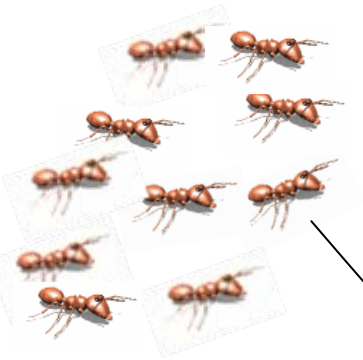


- Sélection des parents + croisement
→ robustesse
- Mutation + sélection des survivants
→ diversité de la population

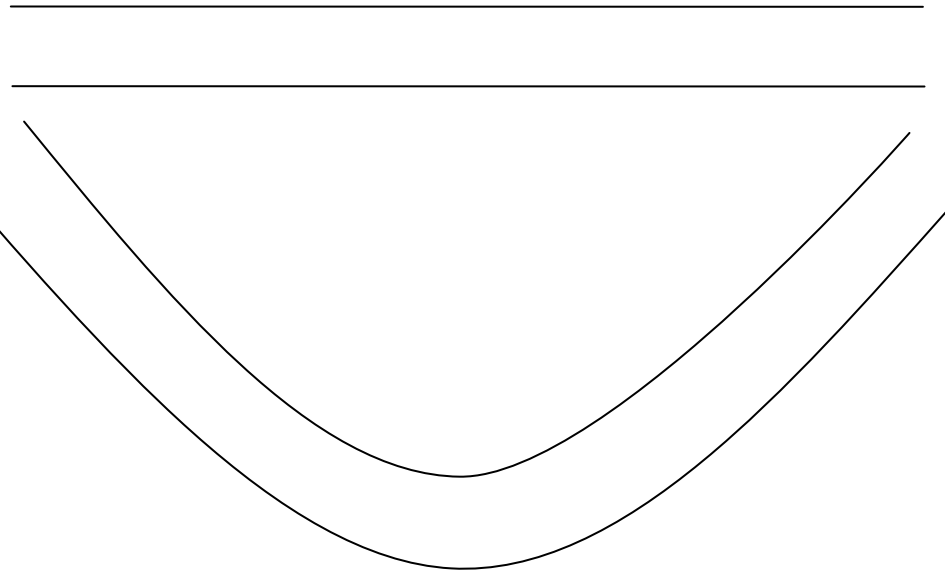


Algorithme de fourmis

Fourmilière



Nourriture





Algorithme de fourmis

- Idée: en plus d'une fonction objectif, ajouter une mémoire pour mieux diriger la création de solutions.
- Chaque fourmi va utiliser deux forces pour construire une solution:
 - *Force gloutonne Fg* (profit à court terme)
 - *Traces Tr* (informations des autres fourmis)



Algorithme de fourmis

- La probabilité que la fourmi k fasse un choix x est calculée grâce à la formule:

$$p_k(x) = \frac{Fg(x)^\alpha \cdot Tr(x)^\beta}{\sum_{x' \in M_k(adm)} Fg(x')^\alpha \cdot Tr(x')^\beta}$$

- α et β sont des paramètres
- $M_k(adm)$ est l'ensemble des choix admissibles pour la fourmi k



Algorithme de fourmis

- Initialisation (# fourmi, α , β et ρ)
- Tant qu'aucun critère d'arrêt est satisfait faire:
 - Pour chaque fourmi faire:
 - Construire pas à pas une solution
 - Mettre à jour localement les traces (optionnel)
 - Mettre à jour globalement les traces



Mise à jour des traces

- Après chaque génération, les traces sont mises à jour:

$$Tr(m) = \rho \cdot Tr(m) + \Delta Tr(m)$$

- Généralement:

$$\Delta Tr(m) = \sum_k \Delta Tr_k(m)$$



Heuristiques hybrides

- Recherche locale → manque de robustesse
- Heuristique évolutive → manque d'efficacité
→ On combine les deux types d'heuristique



Heuristique évolutive hybride

- Générer une population initiale P
- Tant qu'aucun critère d'arrêt est satisfait faire:
 - Sélectionner des individus parents dans P
 - Produire des individus enfants par croisement des individus parents
 - Faire évoluer chaque individus enfants avec une recherche locale
 - Mettre à jour la population



Heuristique évolutive hybride

- Coopération entre individus d'une population
- Intensification de la recherche dans les régions où se trouvent les individus enfants