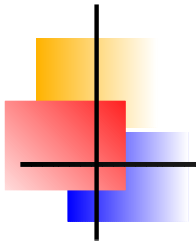
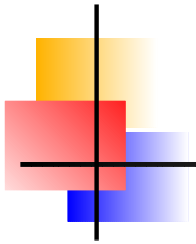


Rappel du plan du cours

- 1 - Introduction
  - Qu'est-ce qu'un problème « complexe » ?
  - Exemples de problèmes « complexes »
- 2 - Approches complètes
  - Structuration de l'espace de recherche en Arbre
    - Application à la plannification et la résolution de CSPs
  - Structuration de l'espace de recherche en Treillis $\rightsquigarrow$ J.-F. Boulicaut
    - Application à la recherche d'ensembles fréquents
- 3 - Approches incomplètes
  - Algorithmes gloutons
  - Notions de voisinage et de paysage de recherche
  - Recherche locale et Méta-heuristiques à base de recherche locale
  - Algorithmes génétiques $\rightsquigarrow$ G. Beslon
- 4 - Optimisation par colonies de fourmis
  - Principes généraux de la méta-heuristique ACO
  - Applications
  - Diversification vs intensification de la recherche par des fourmis



Approches incomplètes

- Définies pour des problèmes d'optimisation $P = (E, f)$
  - E = Espace de recherche = Ensemble des solutions candidates
  - $f : E \rightarrow \mathbb{R}$ = Fonction objectif à maximiser (ou minimiser)

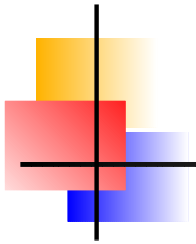
  $\rightsquigarrow$ But = chercher $e^* \in E$ tel que $f(e^*)$ soit maximal
- Peuvent aussi être utilisées pour des problèmes de satisfaction

  $\rightsquigarrow$ recherche de la solution « maximisant la satisfaction »
- Pas de garantie d'optimalité, et encore moins preuve d'optimalité

  ... mais complexité polynomiale

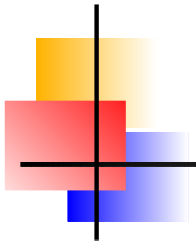
  $\rightsquigarrow$ trouvent rapidement de « bonnes » solutions
- Approches « anytime »

  $\rightsquigarrow$ la qualité de la solution est améliorée au fil du temps
- Exploration « opportuniste » de l'espace de recherche
  - Utilisation de (méta)heuristiques pour se diriger vers les zones prometteuses $\rightsquigarrow$ Intensification
  - Introduction de l'aléatoire pour découvrir/explorer de nouvelles zones $\rightsquigarrow$ Diversification



Panorama des approches incomplètes

- Construction incrémentale d'une solution $\rightsquigarrow$ Greedy (Glouton)
- Constructions itérées « indépendantes » $\rightsquigarrow$ Greedy randomized
- Constructions itérées « basées sur les états » (Instance-based)
  $\rightsquigarrow$ Nouvelles solutions construites à partir des solutions précédentes
  - Algorithmes génétiques
  - Recherche locale :
    - Un seul point de départ : Greedy local search, Random walk, Threshold accepting, Simulated annealing, Tabu search, Reactive local search, Variable neighborhood search, ...
    - Différents points de départ : Multi-start local search, Iterated local search, Go with the winner, Genetic local search (scatter search), ...
- Constructions itérées « basées sur un modèle » (Model-based)
  $\rightsquigarrow$ Nouvelles solutions construites à l'aide d'un modèle probabiliste
  - Greedy Randomized Adaptive Search Procedure (GRASP)
  - Ant Colony Optimization (ACO)
  - Estimation of Distribution Algorithms (EDA)



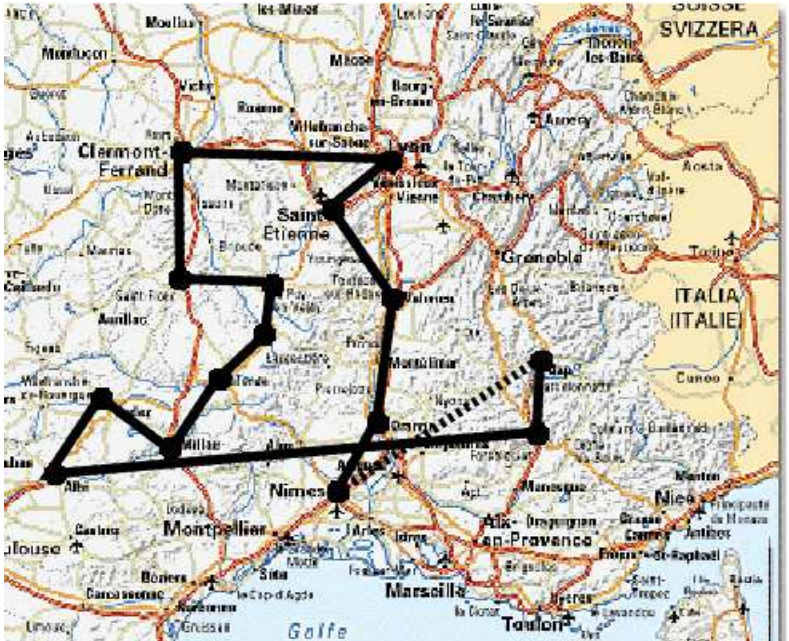
Approches incomplètes / Algorithmes Gloutons

- Construction incrémentale d'une solution $e \in E$ selon une heuristique gloutonne :
  - $e \leftarrow$ « début de solution »
  - $Cand \leftarrow$ ensemble des composants pouvant être ajoutés à e
  - tant que $Cand \neq \emptyset$ faire
    - Choisir le meilleur composant de $Cand$ et l'ajouter à e
      $\rightsquigarrow$ Heuristique de choix « gloutonne »
    - Mettre à jour l'ensemble $Cand$ des composants pouvant être ajoutés à e
  - Retourner e
- Pour résoudre un nouveau problème selon ce principe, il faut définir :
  - une fonction construisant un « début de solution »
  - une fonction donnant/mettant-à-jour l'ensemble des candidats
  - une fonction heuristique évaluant la qualité de chaque candidat
- Pour certains pb, les algorithmes gloutons trouvent la solution optimale
  $\rightsquigarrow$ Simplex/problèmes linéaires, Dijkstra/plus court chemin
- Pour de nombreux autres pb, ils trouvent une « assez bonne » solution...

Approches incomplètes / Algorithmes Gloutons

Exemple 1 : Le voyageur de commerce pour un graphe complet $G = (S, A)$
$\rightsquigarrow$ heuristique gloutonne du « sommet le plus proche »

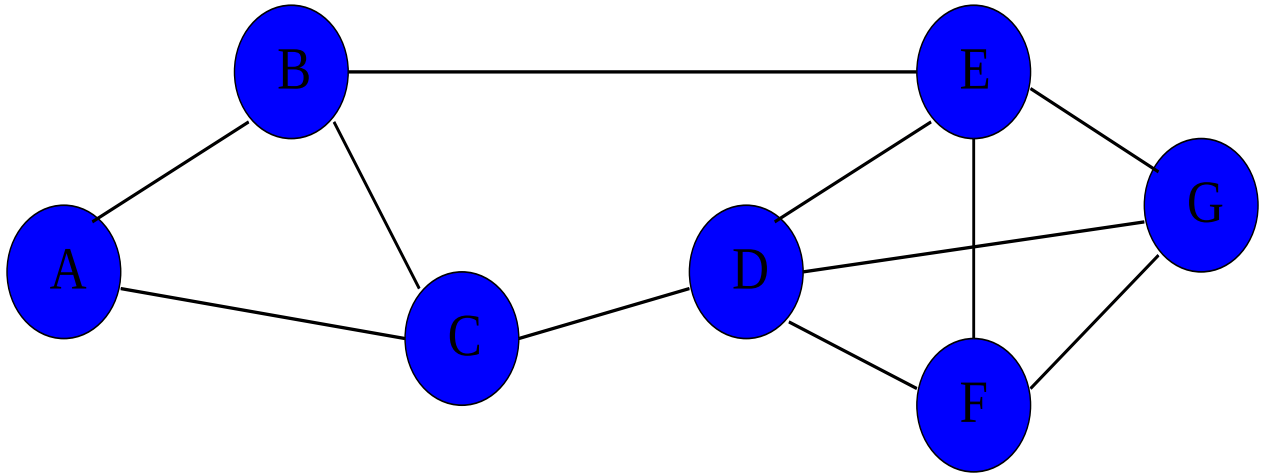
- Choisir aléatoirement un sommet $s_i \in S$, et initialiser le chemin π avec
- $Cand \leftarrow S - \{s_i\}$
- tant que $Cand \neq \emptyset$ faire
  - Soit s_j le sommet de $Cand$ le plus proche du dernier sommet ajouté dans π
  - ajouter s_j à la fin de π
  - $Cand \leftarrow Cand - \{s_j\}$



Approches incomplètes / Algorithmes Gloutons

Exemple 2 : Recherche d'une clique maximum dans $G = (S, A)$
$\rightsquigarrow$ heuristique gloutonne du « sommet de plus fort degré »

- $C \leftarrow \emptyset$
- $Cand \leftarrow S$
- tant que $Cand \neq \emptyset$ faire
  - Soit s_j le sommet de $Cand$ ayant le plus fort degré dans le sous-graphe induit par $Cand$
  - $C \leftarrow C \cup \{s_j\}$
  - $Cand \leftarrow Cand \cap \{s_i / (s_i, s_j) \in A\}$



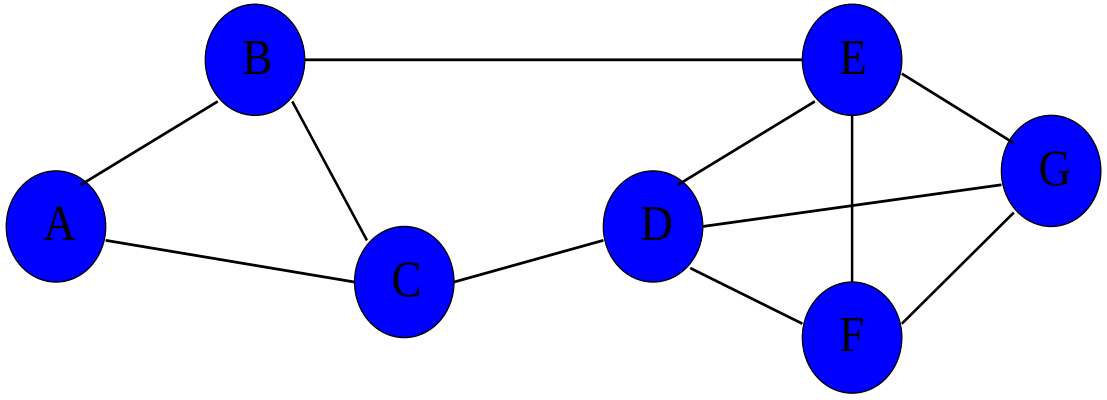
Approches incomplètes / Algorithmes Gloutons

Exemple 3 : Minimiser le nb de couleurs pour colorier $G = (S, A)$

$\rightsquigarrow$ heuristique gloutonne du « sommet le plus saturé » (Brélaz 79) :

$dsat(s_i)$ = nb de sommets adjacents à s_i coloriés avec des couleurs $\neq$

- $nbCoul \leftarrow 0$
- $Cand \leftarrow S$
- tant que $Cand \neq \emptyset$ faire
  - Soit s_j le sommet de $Cand$ tel que $dsat(s_j)$ soit maximal (en cas d'ex-aequo, choisir celui qui a le plus fort degré)
  - Si $\exists i \leq nbCoul$ tq $\forall s_k$ adjacent à $s_j, couleur(s_k) \neq i$
    Alors $couleur(s_j) \leftarrow i$
    Sinon $nbCoul \leftarrow nbCoul + 1;\ couleur(s_j) \leftarrow nbCoul$
  - $Cand \leftarrow Cand - \{s_j\}$



Approches incomplètes / Algorithmes Gloutons

Exemple 4 : Résoudre un CSP $(X, D, C) \rightsquigarrow$ MaxCSP
$\rightsquigarrow$ heuristique gloutonne de la « variable la plus contrainte/contraignante »

- $A \leftarrow \emptyset$
- $Cand \leftarrow X$
- pour toute variable $X_i \in X$, $D_{\text{filtré}}(X_i) \leftarrow D(X_i)$
- tant que $Cand \neq \emptyset$ faire
  - Soit X_j la variable de $Cand$ telle que $|D_{\text{filtré}}(X_j)|$ soit minimal (en cas d'ex-aequo, choisir celle qui intervient dans le plus grand nombre de contraintes)
  - Si $|D_{\text{filtré}}(X_j)| \neq \emptyset$
    Alors choisir une valeur $v \in D_{\text{filtré}}(X_j)$
    Sinon choisir une valeur $v \in D(X_j)$ qui minimise le nombre de contraintes violées
  - $A \leftarrow A \cup < X_j, v >$
  - pour toute variable $X_i \in X$ telle que X_i ne soit pas affectée dans A, $D_{\text{filtré}}(X_i) \leftarrow filtrage(D_{\text{filtré}}(X_i), < X_j, v >)$
  - $Cand \leftarrow Cand - \{X_j\}$

Approches incomplètes / Algorithmes Gloutons

Exemple 5 : ordonner n voitures sur une chaîne de montage

- Il faut installer des options (toit ouvrant, clim, ...) sur chaque voiture
  - k options possibles ; chaque option est installée par une station $\neq$
  - Capacité de la station $i = p_i/q_i \rightsquigarrow$ pour toute séquence de q_i voitures consécutives, pas plus de p_i voitures demandant l'option i
- Exemple : 10 voitures, 5 options de capacités : $1/3, 2/3, 1/3, 2/5$ et $1/5$

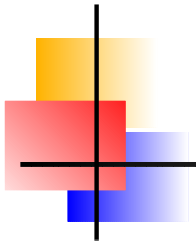
Options demandées par les voitures :

Voitures	Options requises				
1	1	0	1	1	0
2	0	0	0	1	0
3 et 4	0	1	0	0	1
5 et 6	0	1	0	1	0
7 et 8	1	0	1	0	0
9 et 10	1	1	0	0	0

- Plus d'infos sur http://www.csplib.org/ voir prob001

Une solution est :

1	1	0	1	1	0
2	0	0	0	1	0
9	1	1	0	0	0
3	0	1	0	0	1
7	1	0	1	0	0
5	0	1	0	1	0
6	0	1	0	1	0
8	1	0	1	0	0
4	0	1	0	0	1
10	1	1	0	0	0



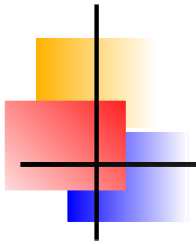
Approches incomplètes / Algorithmes Gloutons

Exemple 5 : Problème d'ordonnancement de voitures

$\rightsquigarrow$ heuristique gloutonne de la « voiture demandant les options les + contraintes »

Option i de capacité $p_i/q_i \rightsquigarrow \tau(i) = \frac{q_i.\text{nb voitures demandant } i}{p_i.\text{nb total voitures}}$

- $\pi \leftarrow <>$
- $Cand \leftarrow$ ensemble des voitures à séquencer
- tant que $Cand \neq \emptyset$ faire
  - Soit c_j la voiture de $Cand$ telle que
    - l'ajout de c_j à la fin de π minimise le nb de contraintes violées
    - et (en cas d'ex-aequo) $\sum_{i \in options(c_j)} \tau(i)$ soit maximal
  - Ajouter c_j à la fin de la séquence π
  - $Cand \leftarrow Cand - \{c_j\}$



Approches incomplètes / Gloutons aléatoires

- En général, un algorithme glouton retourne une assez bonne solution ... mais (à peu près) toujours la même
- Idée :
  - Introduire un peu d'aléatoire pour diversifier la recherche
  - Exécuter plusieurs fois l'algorithme... et retourner la meilleure solution
- Deux grandes approches pour « introduire un peu d'aléatoire »
  - Sélectionner les k meilleurs composants, et en choisir aléatoirement un parmi les k
    $\leadsto$ k = paramètre pour « régler » le degré d'aléatoire
  - Sélectionner le prochain composant selon une probabilité proportionnelle à sa qualité
    - $p(c_i) = \frac{qualite(c_i)^\alpha}{\sum_{c_k \in Cand} qualite(c_k)^\alpha}$
      $\leadsto$ $\alpha =$ paramètre pour « régler » le degré d'aléatoire
    - Choisir $c_i \in Cand$ avec la probabilité $p(c_i)$
      - Tirer un nombre flottant aléatoire x compris entre 0 et 1
      - Choisir c_i tq $\sum_{j<i} p(c_j) < x \leq \sum_{j \leq i} p(c_j)$

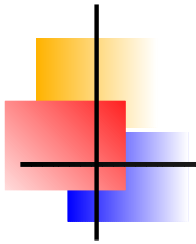
Comparaison de Glouton et Glouton aléatoire

Problèmes d'ordonnancement de voitures :

	Aléat.	Glouton	Glouton aléat.
60	5.4 (1.0)	1.6 (0.0)	0.0 (0.0)
65	3.0 (0.7)	2.3 (0.0)	0.0 (0.0)
70	1.4 (0.5)	0.9 (0.2)	0.0 (0.0)
75	0.6 (0.4)	0.6 (0.2)	0.0 (0.0)
80	2.0 (0.9)	0.5 (0.3)	0.0 (0.0)
85	8.3 (1.7)	0.5 (0.4)	0.0 (0.0)
90	28.1 (2.5)	1.2 (0.3)	0.0 (0.0)

	Aléat.	Glouton	Glouton aléat.
1	35.4 (1.7)	10.4 (1.6)	5.5 (0.6)
2	23.1 (1.1)	7.7 (0.9)	0.8 (0.4)
3	29.9 (1.7)	8.5 (1.1)	2.8 (0.4)
4	27.1 (1.5)	5.5 (0.7)	2.6 (0.5)
5	19.3 (1.3)	3.6 (0.9)	1.0 (0.2)
6	25.9 (1.7)	8.5 (0.5)	3.4 (0.6)
7	13.3 (1.3)	3.7 (0.7)	0.0 (0.0)
8	24.1 (1.6)	5.0 (0.0)	0.2 (0.4)
9	13.1 (0.9)	6.0 (0.0)	6.0 (0.0)

- Glouton aléatoire $\rightsquigarrow$ proba proportionnelle à la qualité / $\alpha = 6$
- Construction de 100 solutions successives par exécution
  $\rightsquigarrow$ retour de la meilleure solution sur les 100 calculées
- Moyenne (écart-type) sur 100 exécutions



Approches incomplètes / « instance-based »

- Glouton aléatoire :
  - Construction de plusieurs solutions (pour ne garder que la meilleure)
  - Chaque construction est indépendante des précédentes
- Idée : utiliser l'expérience passée pour construire de nouvelles solutions
  - Approches « instance-based » :
    - Faire évoluer une « population de solutions » $\Rightarrow$ algorithmes génétiques
    - Repartir de la dernière solution construite $\Rightarrow$ recherche locale
  - Approches « model-based » :
    1. Construire un modèle probabiliste
    2. Générer une ou plusieurs solutions avec ce modèle
    3. Améliorer le modèle en fonction des dernières solutions calculées
    4. Recommencer en 2

Approches incomplètes / Algos génétiques

- Basé sur la métaphore de l'évolution des espèces
  $\rightsquigarrow$ Faire évoluer une population de solutions
  - **Population initiale** : Générer un ensemble P de n solutions (génération aléatoire ou à l'aide d'un algorithme glouton)
  - **Tant que** « qualité de la meilleure solution de P insuffisante » **et** **Tant que** « temps limite non atteint » **faire** :
    - **Phase de sélection** : Sélectionner un ens. $P' \subseteq P$ de solutions
      $\rightsquigarrow$ Favoriser la sélection des meilleures solutions
      ... tout en préservant la diversité
    - **Phase de croisement** : Combiner les solutions de P' pour générer un ensemble P'' de nouvelles solutions
      $\rightsquigarrow$ Intensifier l'effort de recherche autour des « bonnes » solutions
    - **Phase de mutation** : Modifier aléatoirement quelques composants des nouvelles solutions de P''
      $\rightsquigarrow$ Diversifier la population
    - **Phase de remplacement** : Former une nouvelle population P à partir de P'' et P
- Plus d'infos dans le cours du 9 novembre fait par Guillaume Beslon

Approches incomplètes / Recherche Locale

- Structuration de l'espace de recherche E en termes de voisinage
  $\rightsquigarrow$ Exploration de E en passant de solutions en solutions voisines
- Définition d'une relation de voisinage entre solutions $V \subseteq E \times E$
  $\rightsquigarrow$ e_i et e_j sont « voisins » si $(e_i, e_j) \in V$
  $\rightsquigarrow$ Le graphe $G = (E, V)$ doit être connexe
  Notation : $V(e_i) = \{e_j \in E / (e_i, e_j) \in V\}$ = ens. des voisines de e_i
- En général, $V(e_i)$ = ensemble des solutions obtenues en appliquant k transformations élémentaires à e_i
  $\rightsquigarrow$ la taille du voisinage peut être plus ou moins grande :
  - Si k est petit, chaque solution a peu de voisines
    Si k augmente, chaque solution a plus de voisines
    (le nombre de voisines est souvent exponentiel par rapport à k)
  - Plus une solution a de voisines,
    ... plus cela prend de temps pour choisir la prochaine solution,
    ... mais plus la prochaine solution a de chances d'être bonne.
    $\rightsquigarrow$ compromis à trouver

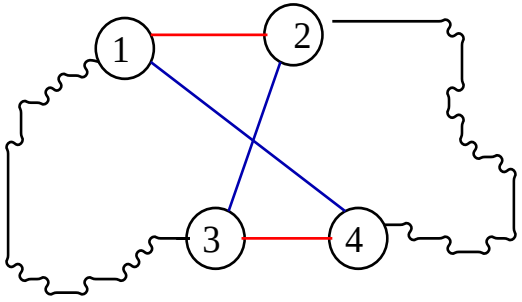
Recherche locale / Exemples de voisinages

Problème du voyageur de commerce pour un graphe $G = (S, A)$

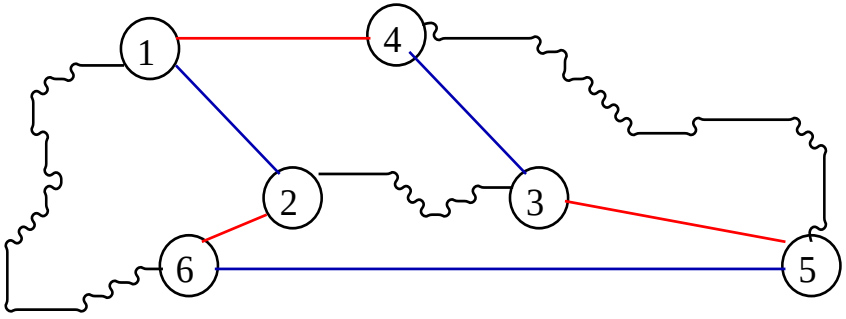
- Espace de recherche $E =$ ensemble des permutations de S
- Objectif : minimiser $f(e_i)$ = somme des coûts des arêtes de e_i
- Exemples de voisinages :
  - Voisinage « k-opt » ($k \geq 2$)

    $$V(e_i) = \{e_j \in E / e_j \text{ est obtenu en changeant } k \text{ arêtes de } e_i\}$$
  - Voisinage par « suppression/insertion » de k sommets consécutifs

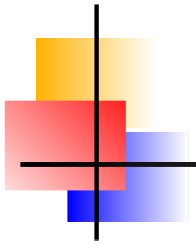
    $$V(e_i) = \{e_j \in E / e_i = u.x.u', e_j = v.x.v', |x| = k, u.u' = v.v'\}$$



Changer (1,4) et (3,2)
en (1,2) et (3,4)



Supprimer le chemin allant de 2 à 3,
et l'insérer entre 6 et 5

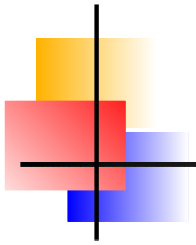


Recherche locale / Exemples de voisinages

Problème de la clique maximum pour un graphe $G = (S, A)$

- Espace de recherche $E \subseteq \wp(S) =$ ens. des cliques de S
- Objectif : maximiser $f(e_i) = |e_i|$
- Exemples de voisinages :
  - Voisinage par « ajout/suppression » :

    $V(e_i) = \{e_j$ obtenus en ajoutant ou supprimant k sommets de $e_i\}$
  - Voisinage par « échange $k/k+1$ » :
    $V(e_i) = \{e_j$ obtenus en remplacant k sommets de e_i par $k+1$ sommets de $S - \{e_i\}\}$



Recherche locale / Exemples de voisinages

Problème SAT, défini pour un ensemble de n variables booléennes

- Espace de recherche $E = \{\text{vrai, faux}\}^n$
- Objectif : maximiser $f(e_i) =$ nombre de clauses satisfaites par e_i
- Voisinage par « basculements/flips »

$$V(e_i) = \{e_j \text{ obtenus en « flippant » la valeur de } k \text{ var de } e_i\}$$

Problèmes de Satisfaction de Contraintes (X, D, C)

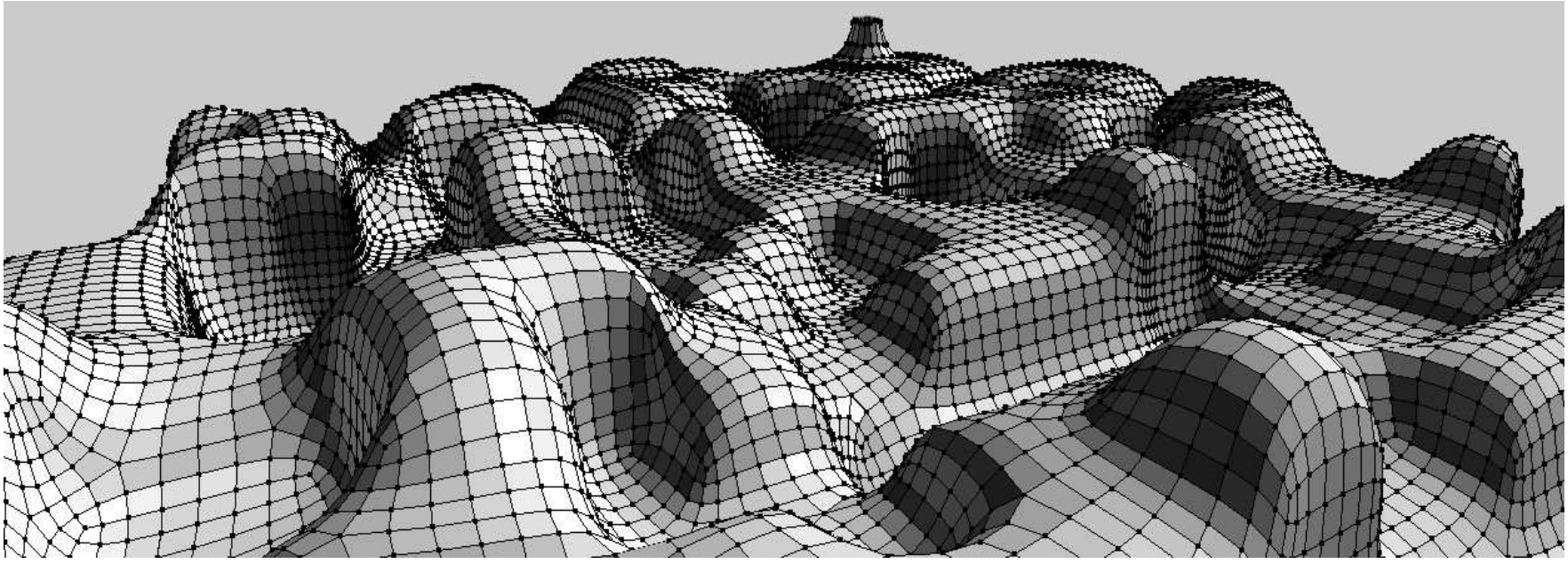
- Espace de recherche $E = D(X_1) \times \ldots \times D(X_n)$ si $X = \{X_1, \ldots, X_n\}$
- Objectif : maximiser $f(e_i) =$ nombre de contraintes satisfaites par e_i
- Voisinage par changement de valeurs

$$V(e_i) = \{e_j \text{ obtenus en changeant la valeur de } k \text{ variables ds } e_i\}$$

Recherche locale / Paysage de recherche

Un paysage de recherche = un problème $P = (E, f)$ + un voisinage $V \subseteq E \times E$

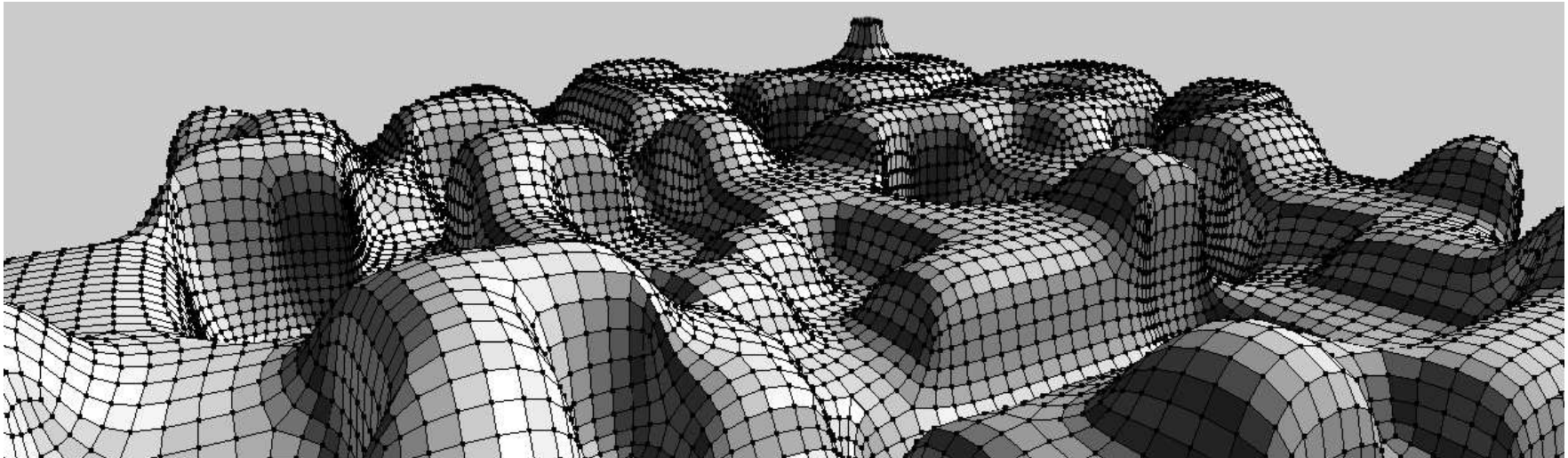
- Représentation « graphique » du pb (dans un hyper-espace)
  - Chaque configuration de E = 1 point
  - f donne l' « altitude » des points
  - V positionne les points dans les autres dimensions

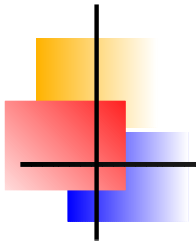


Recherche locale gloutonne (greedy local search)

(Aussi appelée « Hill climbing » et « montée de gradient »)

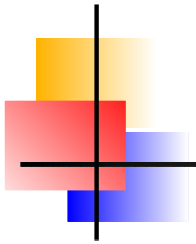
- Principe : toujours monter en suivant la plus forte pente
  - Partir d'une solution e (générée aléatoirement ou par algo glouton)
  - Tant que $MaxMove$ non atteint et qualité de e insuffisante faire
    - Remplacer e par une solution de $V(e)$ qui maximise f
  - Retourner e
- Variante (généralement utilisée quand le voisinage est important) : Remplacer e par la première solution trouvée $e' \in V(e)$ tq $f(e') > f(e)$
- En général : trouve une solution « localement optimale » (telle que $\forall e' \in V(e), f(e') \leq f(e)$) ... mais pas « globalement optimale » : e peut être sur un plateau, ou au sommet d'un pic qui n'est pas le plus haut





Recherche locale gloutonne / Applications

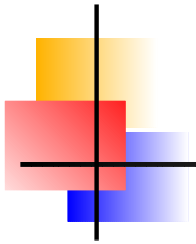
- Problème SAT $\rightsquigarrow$ GSAT (Selman, Levesque, Mitchell 92)
  - $A \leftarrow$ valuation générée aléatoirement
  - Tant que A n'est pas solution et $MaxFlip$ non atteint faire
    - $V(A) \leftarrow \{A'$ obtenus en flippant la valeur d'une variable de $A\}$
    - $Best \leftarrow \{A' \in V(A)/$ nb de clauses satisfaites est maximal$\}$
    - remplacer A par une valuation de $Best$ choisie aléatoirement
- Pb de satisfaction de contraintes $\rightsquigarrow$ Min Conflict Heuristic (Minton 92)
  - $A \leftarrow$ affectation générée aléatoirement
  - Tant que A n'est pas solution et $MaxMove$ non atteint faire
    - Choisir aléatoirement une variable X_i intervenant dans une contrainte violée
    - $V(A) \leftarrow \{A'$ obtenus en changeant la valeur de $X_i\}$
    - $Best \leftarrow \{A' \in V(A)/$ nb de contraintes satisfaites est maximal$\}$
    - remplacer A par une affectation de $Best$ choisie aléatoirement
- Ces procédures sont généralement répétées plusieurs fois, à partir de différentes valuations.



Recherche locale : Intensifier vs diversifier

Résoudre un problème avec une approche incomplète
$\rightsquigarrow$ trouver un compromis entre intensification et diversification

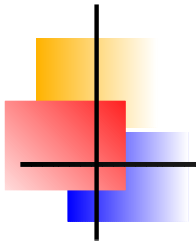
- Intensification :
  - Concentrer l'effort de recherche aux alentours des zones prometteuses
    - Algorithmes génétiques : croiser les bonnes solutions
    - Recherche locale : choisir le meilleur voisin
  - Postulat / paysage de recherche : corrélation entre la qualité d'un point et sa distance au point optimal
    $\rightsquigarrow$ OK si le paysage a peu de collines, et si elles sont regroupées
    $\rightsquigarrow$ KO si le paysage de recherche est « rugueux »
  - Risque : concentrer la recherche sur de trop petites zones
- Diversification :
  - Inciter la recherche à aller voir ailleurs s'il n'y a pas mieux...
    - Algorithmes génétiques : mutation + introduction d'aléatoire dans la sélection
    - Recherche locale : autoriser de s'éloigner des optima locaux
  - Risque : trop disperser la recherche



Recherche locale : « random walk »

- Diversifier la recherche en autorisant parfois un mouvement aléatoire
  $\rightsquigarrow$ Introduire la probabilité « p_{noise} » de choisir un mouvement aléatoire
  - Partir d'une solution e (générée aléatoirement ou par algo glouton)
  - Tant que $MaxMove$ non atteint et qualité de e insuffisante faire
    - Soit x un nombre tiré aléatoirement entre 0 et 1
    - Si $x < p_{\text{noise}}$ Alors // diversification
      - Remplacer e par 1 solution choisie aléatoirement dans $V(e)$

      Sinon // Intensification
      - Remplacer e par une solution de $V(e)$ qui maximise f
  - Retourner la meilleure solution calculée
- La valeur de p_{noise} détermine la diversification / intensification
  - $p_{\text{noise}} = 1 \Rightarrow$ aléatoire pur
  - $p_{\text{noise}} = O \Rightarrow$ glouton pur
  - En général, $0.01 \leq p_{\text{noise}} \leq 0.1$



Recherche locale : « threshold accepting »

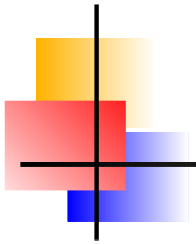
- Diversifier la recherche en autorisant des mouvements « moins bons »
  $\rightsquigarrow$ Introduction d'un seuil d'acceptation τ
  - Partir d'une solution e (générée aléatoirement ou par algo glouton)
  - Tant que $MaxMove$ non atteint et qualité de e insuffisante faire
    - Choisir aléatoirement $e' \in V(e)$ // diversification
    - Si $f(e') - f(e) > \tau$ Alors // intensification
      - Remplacer e par e'
  - Retourner la meilleure solution calculée
- La valeur de τ détermine la diversification / intensification
  - $\tau \rightarrow -\infty \Rightarrow$ aléatoire pur
  - $\tau \geq 0 \Rightarrow$ les mouvements dégradant la solution sont interdits
- Le seuil d'acceptation τ peut augmenter au fil du temps
  $\rightsquigarrow$ « gloutoniser » l'algorithme pour le faire converger

Recherche locale : « simulated annealing »

- Diversifier la recherche en autorisant des mouvements moins bons
  ... en fonction d'une probabilité d'acceptation qui décroit avec le temps
  (Kirkpatrick 82)
- Inspiré du « recuit simulé » en métallurgie
  (équilibre énergétique lors de la cristallisation des métaux)
  $\rightsquigarrow$ Introduction d'un paramètre « température » qui décroit avec le temps
- Algorithme :
  - Partir d'une solution e (générée aléatoirement ou par algo glouton)
  - Initialiser la température T
  - Tant que « système non gelé » et qualité de e insuffisante faire
    - Choisir aléatoirement $e' \in V(e)$
    - Si $f(e') \geq f(e)$ Alors // intensification
      - Remplacer e par e'

      Sinon
      - Soit x un nombre tiré aléatoirement entre 0 et 1
      - Si $x < e^{-(f(e)-f(e'))/T}$ Alors // diversification

        Remplacer e par e'
    - Diminuer la température T
    - Retourner la meilleure solution calculée



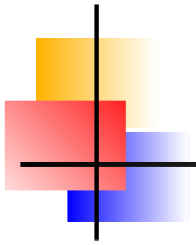
Recherche locale : « simulated annealing »

La température T détermine la diversification / intensification

- $T \to \infty \Rightarrow e^{-(f(e)-f(e'))/T} \to 1 \Rightarrow$ aléatoire pur
  $T \to 0 \Rightarrow e^{-(f(e)-f(e'))/T} \to 0 \Rightarrow$ glouton pur
- Au début, T a une valeur assez grande (de l'ordre de la centaine)
  $\leadsto$ Forte diversification
- Au fur et à mesure de la recherche, T diminue
  $\leadsto$ Intensification progressive autour des zones intéressantes
- Quand T se rapproche de 0, le système se « gèle » sur un optimum local
- La vitesse avec laquelle la température diminue détermine le temps de convergence : plus T diminue doucement, plus la convergence est lente... mais meilleure est la solution finale (en général)

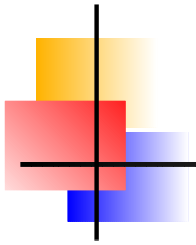
Le paramétrage peut s'avérer délicat :

- 3 paramètres à fixer : Valeur initiale de la température, Taux de diminution de la température, et Seuil de gel
- La qualité de la solution trouvée dépend fortement de ces 3 paramètres ... et la valeur optimale de ces paramètres dépend du pb à résoudre



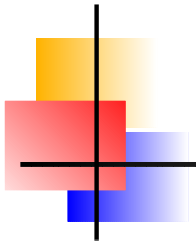
Recherche locale « Taboue »

- Toujours choisir le meilleur mouvement
  ... mais mémoriser les derniers mouvements faits (dans une liste « taboue »)
  ... et interdire les mouvements inverses (afin de ne pas tourner en rond)
  (Glover 86)
- Algorithme :
  - Partir d'une solution e (générée aléatoirement ou par algo glouton)
  - $ListeTaboue \leftarrow \emptyset$
  - Tant que $maxMove$ non atteint et qualité de e insuffisante faire
    - Choisir la solution $e' \in V(e)$ telle que
      - le mouvement $e \rightarrow e' \notin ListeTaboue$ // diversification
      - $f(e')$ soit maximal // intensification
    - Remplacer e par e'
    - Si $|ListeTaboue| = lg$ Alors
      - enlever le plus vieux mouvement de $ListeTaboue$
    - Ajouter le mouvement $e \rightarrow e'$ dans $ListeTaboue$
  - Retourner la meilleure solution calculée



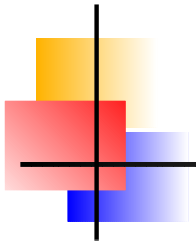
Recherche locale « Taboue »

- La liste taboue contient les lg derniers mouvements effectués
  (... et non les lg dernières solutions visitées)
  $\rightsquigarrow$ moins coûteux à vérifier et stocker
  Exemples :
  - Clique : mémoriser les lg derniers sommets ajoutés/supprimés
  - SAT : mémoriser les lg dernières variables « flippées »
- Possibilité d'ajouter un « critère d'aspiration » : si un mouvement tabou permet d'obtenir une solution meilleure que la meilleure solution trouvée jusque là, alors l'accepter.
- La longueur de la liste lg détermine la diversification / intensification
  - $lg = 0 \Rightarrow$ Glouton pur
  - Quand lg est petit, la recherche a tendance à tourner en rond autour d'optima locaux
  - Plus lg augmente, et plus la recherche est diversifiée
    ... mais plus on risque de s'interdire de monter sur le pic optimal
- Là encore, le paramétrage est un point délicat...



Recherche locale « réactive »

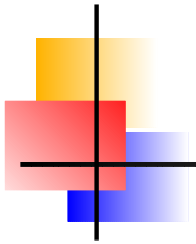
- La longueur de la liste Taboue est un point déterminant
  - Trop courte $\Rightarrow$ Intensification trop forte
    $\rightsquigarrow$ blocage de la recherche autour d'un optimum local
  - Trop longue $\Rightarrow$ Diversification trop forte
    $\rightsquigarrow$ la recherche risque de passer à coté des meilleures solutions
- La longueur optimale de la liste varie
  - d'un problème à l'autre
  - d'une instance à l'autre d'un même problème
  - au cours de la résolution d'une même instance
- Idée (Battiti, Protasi 2001) : adapter cette longueur dynamiquement
  - Besoin de diversification $\Rightarrow$ augmenter la longueur
  - Besoin d'intensification $\Rightarrow$ diminuer la longueur



Recherche locale « réactive »

- Mise-en-œuvre :
  - Mémoriser chaque solution visitée dans une table de Hachage
    $\rightsquigarrow$ Peut être fait en temps constant !!!
  - Collision dans la table $\Rightarrow$ besoin de diversification
    $\rightsquigarrow$ allonger la liste taboue
  - Longue période sans collision $\Rightarrow$ besoin d'intensification
    $\rightsquigarrow$ raccourcir la liste taboue
- Paramètres de l'algorithme :
  - Longueur initiale de la liste
  - Longueur minimale et maximale de la liste
  - Taille de l'allongement de la liste
  - Taille de la période sans collision
  - Taille du raccourcissement de la liste

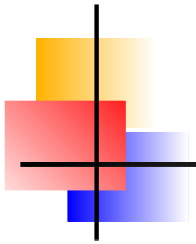
  $\rightsquigarrow$ Beaucoup plus de paramètres que la recherche taboue non réactive
  ... paramétrage beaucoup plus robuste en pratique
- Recherche réactive = meilleure approche connue (et de loin) pour la recherche de cliques maximum



Recherche à voisinage variable / Variable Neighborhood Search (VNS)

- Motivation : On peut généralement définir plusieurs voisinages différents
  $\rightsquigarrow$ le max local d'1 voisinage n'est pas toujours max ds d'autres voisinages
  $\rightsquigarrow$ changer de voisinage quand on arrive sur un maximum local
- Algorithme :
  - Soient V^t un ensemble fini de voisinages ($1 \leq t \leq t_{max}$)
    $V^t(e) =$ ensemble des solutions voisines de e pour le t^{ieme} voisinage
  - $t \leftarrow 1$
  - Partir d'une solution e (générée aléatoirement ou par algo glouton)
  - Tant que $maxMove$ non atteint et qualité de e insuffisante faire
    - Choisir aléatoirement une solution $e' \in V^t(e)$ // Diversification
    - $e'' \leftarrow$ rechercheLocale(e', V^t)
    - Si $f(e'') > f(e)$ Alors
      - $e \leftarrow e''$ // Intensification
      - $t \leftarrow 1$

      Sinon
      - $t \leftarrow t + 1$ // Changement de voisinage
      - Si $t > t_{max}$ Alors $t \leftarrow 1$
  - Retourner la meilleure solution calculée



Recherche locale / plusieurs points de départ

Un processus de recherche locale peut être itéré plusieurs fois,
à partir de points différents :

- Multi-start local search : Les points de départs sont indépendants
- Iterated local search : Le point de départ d'une recherche locale est une perturbation de la meilleure solution de la dernière recherche locale effectuée
- Go with the winner :
  - Plusieurs recherches locales en parallèle (particules)
  - Les moins bonnes sont « redistribuées » autour des meilleures
- Genetic local search (scatter search) :
  - Une population de solutions évolue par sélections/croisements/mutations
  - Chaque nouvelle solution générée est améliorée par recherche locale