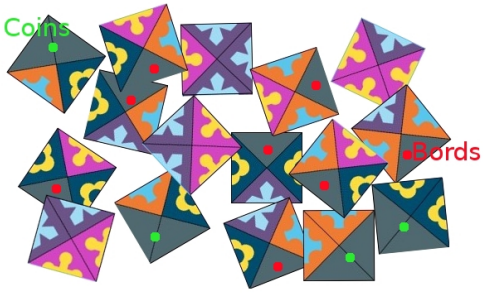
Cours d'Optimisation Combinatoire

Table des matières

I Exercice d'Introduction

1. Questions

Question 1



Faites le puzzle 4*4 (bord en gris) et notez combien de minutes cela vous a pris.

Question 2

Estimez combien de temps il vous faudrait pour faire un puzzle 16*16.

Question 3

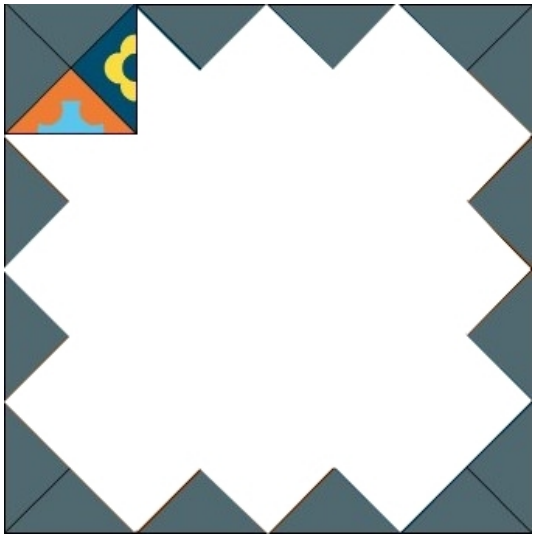
Imaginons que vous devez investir 40 euros et qu'on vous propose d'en gagner 1000 si vous réussissiez à faire le puzzle 16*16 en moins d'une journée. Choisissez-vous d'investir dans le puzzle ou dans 20 grilles de loto ?

Question 4

Trouvez une méthode (que l'on pourrait programmer) pour résoudre le puzzle 16*16 et estimez le temps qu'un tel programme (sur un pc standard) mettrait avant de trouver une solution.

Pensez-vous que ça soit rentable de programmer un tel programme dans le but de résoudre le puzzle le plus vite possible ?

Question 5



Déterminez le nombre de possibilités de placer les pièces du puzzle 4*4 (puis pour le 16*16) avec uniquement les contraintes suivantes :

- les bords et les coins doivent former le contour du puzzle
- un coin est fixé pour ne pas comptabiliser les symétries

« Dans ces conditions, on ne veut pas forcément que le puzzle soit correct. »

Question 6

A quoi et a qui peut servir l'optimisation combinatoire ?

II Introduction

1. Objectifs du cours

- Savoir évaluer la complexité d'un problème
- Savoir modéliser un problème
- Connaître certaines méthodes de résolution d'un problème difficile

2. Notion de complexité

Définition : la complexité d'un problème

La complexité d'un problème est la complexité minimale dans le pire des cas d'un algorithme qui le résout.

Exemple

Par exemple, si on dit que la complexité d'un problème est quadratique cela veut dire :

- qu'il existe un algorithme quadratique qui le résout
- **ET** que tout algorithme qui le résout sera au moins quadratique

Dans la pratique, soit deux algorithmes A et B, et respectivement leur complexité dans le pire des cas O_A, O_B tel que $O_A > O_B$.

Quel algorithme choisiriez-vous ?

Remarque

C'est souvent la complexité en temps que l'on considère mais on peut s'intéresser à d'autres mesures comme par exemple la complexité en espace.

Un algorithme répond à un problème de taille n (ex : n entiers à trier).

Il est composé d'un certain nombre d'étapes en fonction de n.

Plusieurs algorithmes peuvent répondre à un même problème (ex : tri bulle, tri fusion, etc.).

Pour savoir lequel est le plus efficace, il faut les comparer en fonction du nombre d'étapes nécessaires à la résolution du problème.

Notation	Type de complexité	Temps					Exemple
		n=5	n=10	n=20	n=50	n=1000000	
$O(1)$	constante	10 ns	10 ns	10 ns	10 ns	10 ns	accès tableau
$O(\log(n))$	logarithmique	10 ns	10 ns	10 ns	20 ns	60 ns	dichotomie
$O(n)$	linéaire	50 ns	100 ns	200 ns	500 ns	10 ms	parcours de liste
$O(n^2)$	quadratique	250 ns	1 μs	4 μs	25 μs	2,8 h	parcours tableau 2D
$O(n^3)$	cubique	1,25 μs	10 μs	80 μs	1,25 ms	**316 ans**	multiplication matrices
$O(e^n)$	exponentielle	320 ns	10 μs	10 ms	**130 jours**	**bcp trop**	sac à dos (force brute)
$O(n!)$	factorielle	1.2 μs	36 ms	**770 ans**	**10^{48} ans**	**encore pire**	T S P (approche naïve)

3. Problèmes

Modélisation

Résolution d'un problème :

- Problème --> modélisation --> solution(s)

Modélisation :

- simplification de la réalité (nombre de paramètres, bruit, informations inutiles, ...)

Conception d'un (bon) modèle :

- Connaissance experte du domaine
- Connaissance des méthodes de résolution (informatique)

Sac à dos

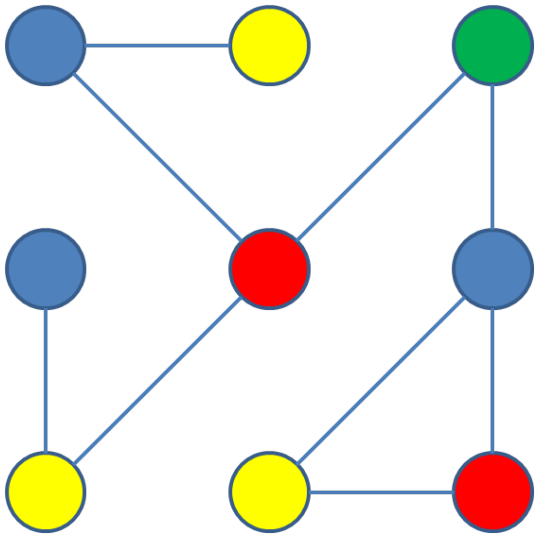
Choisir quels objets prendre pour maximiser l'intérêt :

- Chaque objet a un poids (ou volume) et une valeur d'intérêt
- Le sac a une capacité de poids (ou volume) limité.

Applications (sous-problème):

- gestion de portefeuille
- découpe de matériaux

Coloration de graphe



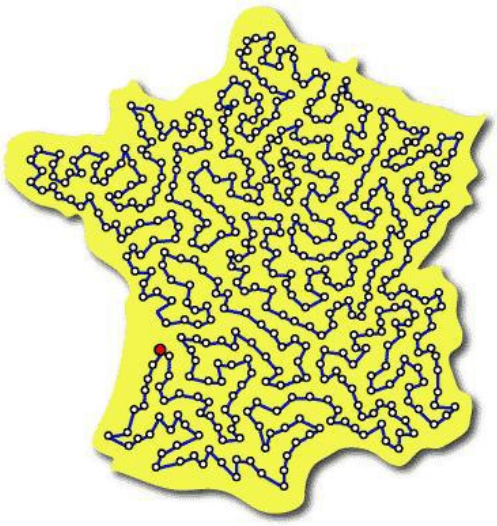
Colorier le graphe de sorte que 2 nœuds voisins n'aient pas la même couleur (éventuellement pour) :

- minimiser le nombre de couleurs nécessaires

Applications :

- emploi du temps
- coloration de cartes

Voyageur de Commerce (TSP)



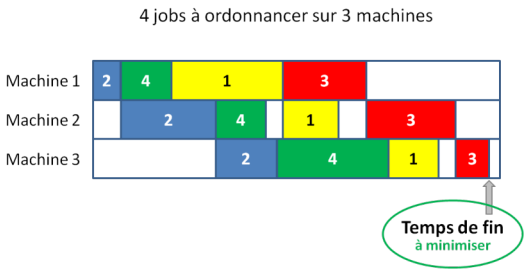
Déterminer un ordre de parcours des villes pour :

- minimiser la distance parcourue

Applications :

- Transports
- Tournées de véhicules

FlowShop (Ordonnancement)



Ordonnancer N jobs sur M machines pour :

- Minimiser le temps (fin du dernier job)

Applications :

- Chaînes de construction
- Informatique

Questions

Comment modéliser chacun des problèmes précédents ?

(Trouver la représentation informatique d'une solution au problème)

Questions

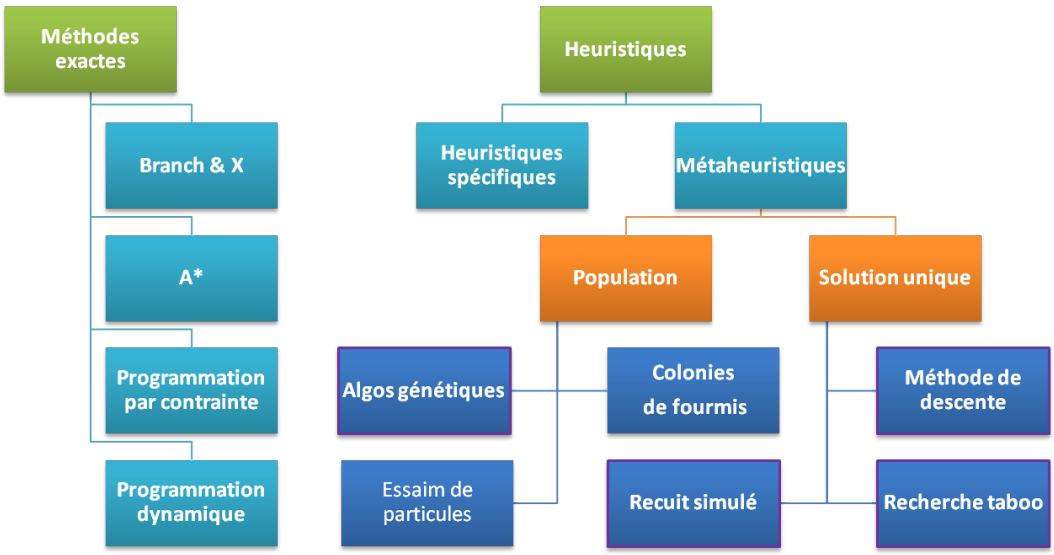
A quel problème pourrait-on rapporter le sudoku ? Préciser.

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

Comment résoudre ce genre de problème ?

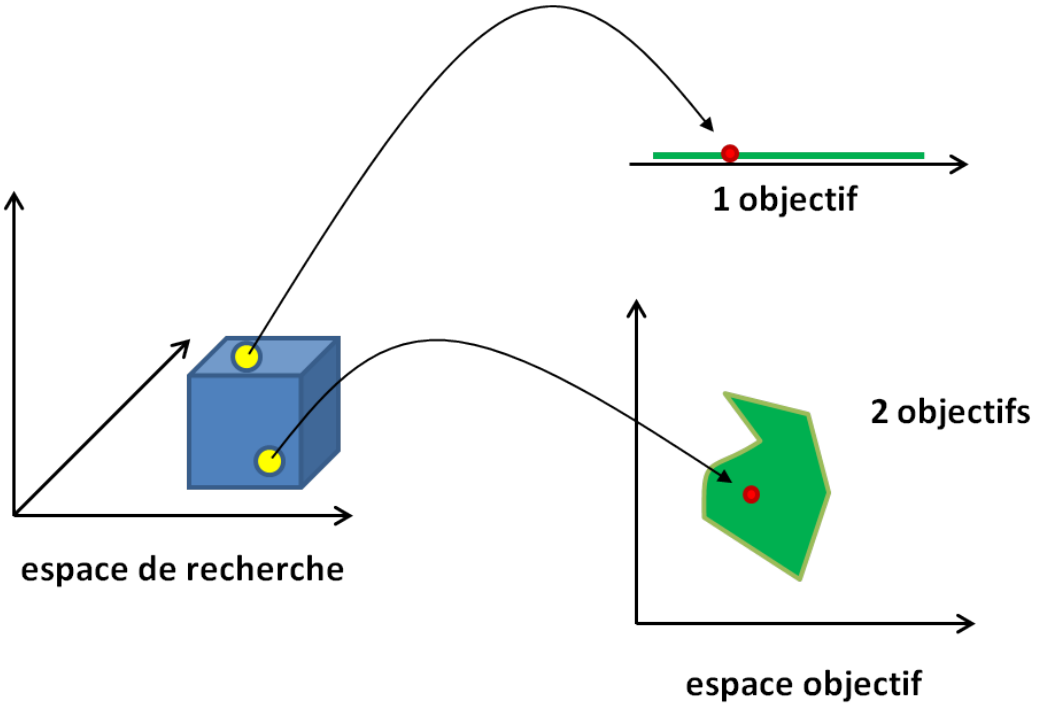
- Exhaustivement
- Aléatoirement
- Construction de solutions
- Méthodes exactes
- Méthodes d'approximations

Méthodes d'optimisation

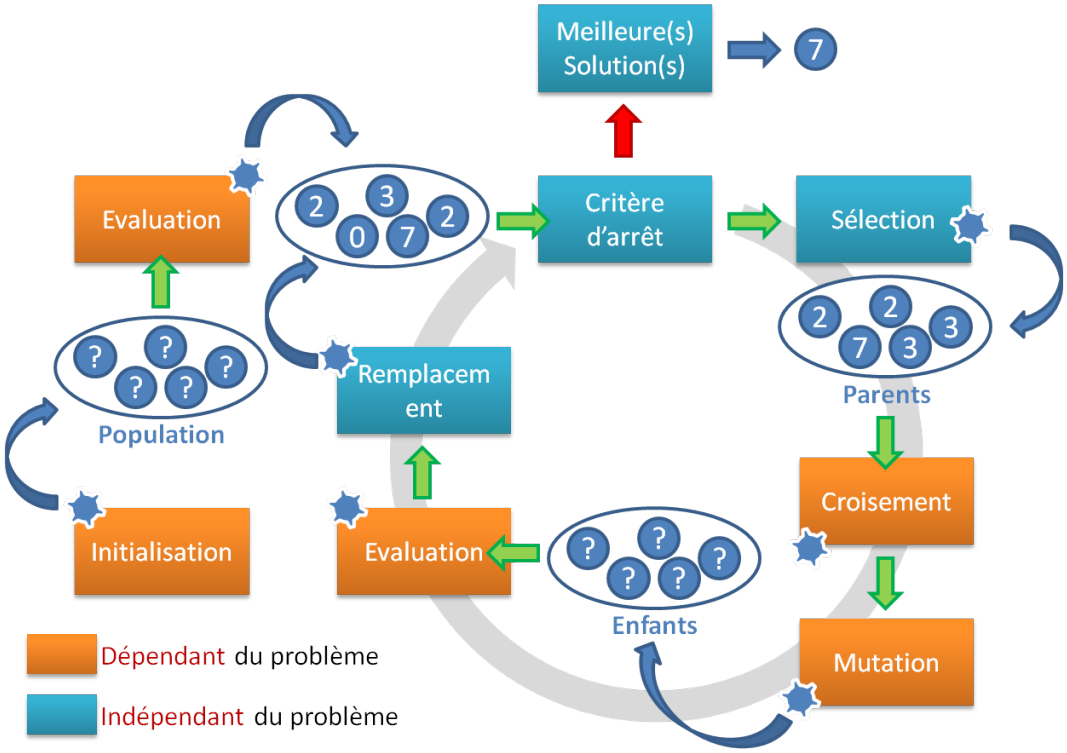


III Algorithmes génétiques

1. Espace de recherche et espace objectif



2. Algorithme génétique (AG)



3. Opérateurs dépendants du problème

Initialisation

Initialiser une solution en fonction de la représentation choisie

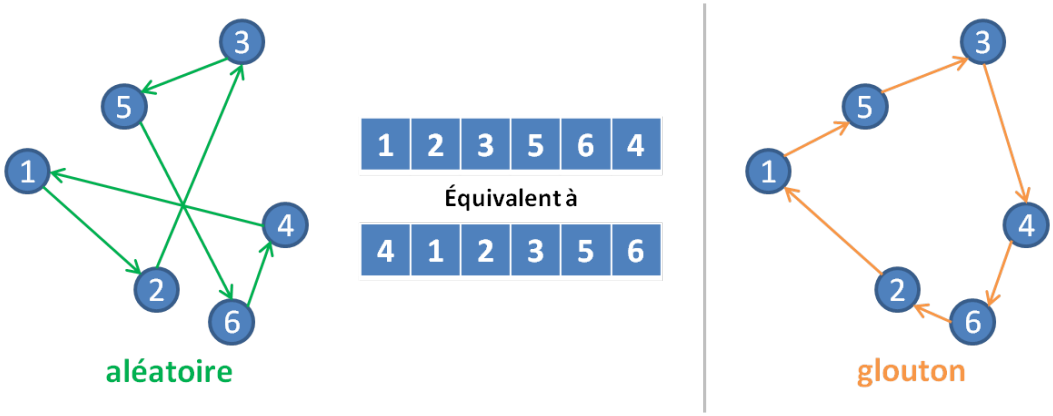
Plusieurs manières :

- aléatoire
- par construction

Attention

- Respecter les contraintes
- Solutions équivalentes

Exemple : Pour le TSP



Question : Est-ce que les deux solutions de l'exemple de l'initialisation aléatoire sont équivalentes pour un problème d'ordonnancement ?

Evaluation

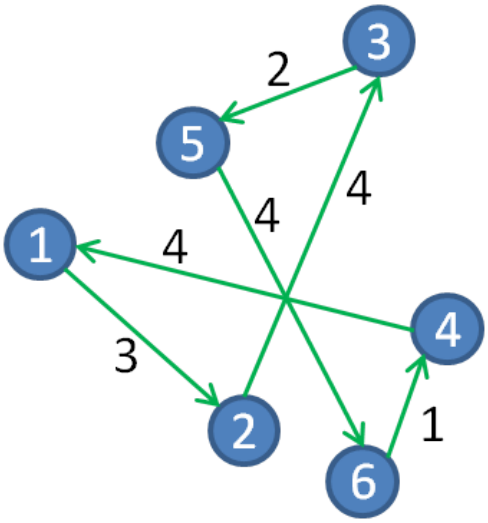
Définir une fonction d'évaluation d'une solution.

Cette fonction affecte un score (fitness) à la solution.

Un algorithme génétique ayant pour but principal de maximiser (minimiser) cette fitness.

Cela consiste à rechercher dans l'espace de recherche la solution qui a la plus grande (plus petite) fitness dans l'espace objectif.

Exemple : Pour le TSP



La fonction d'évaluation retournera la somme des distances parcourues, soit 18.

Une matrice des distances entre chaque ville est nécessaire.

Cette fitness est à minimiser.

Croisement (crossover)

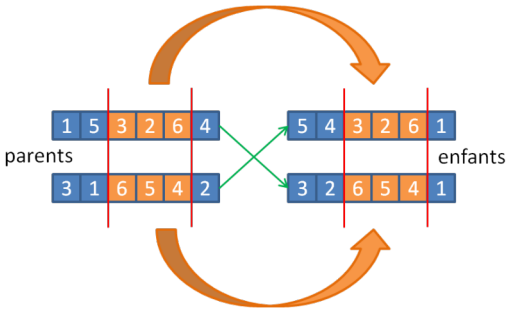
Générer une ou plusieurs solutions à partir de deux solutions "parents".

Un crossover a pour but d'essayer de combiner des parties intéressantes des solutions "parents".

Attention

- Crossover trop destructeur
- Respecter les contraintes :
  - géré par le crossover
  - technique de reconstruction

Exemple : Pour le TSP



Un crossover 2 points :

- recopie la partie entre les 2 points de coupure (tiré aléatoirement) du 1er parent
- réinsère les éléments manquants à partir du 2e point de coupure, dans l'ordre ou ils apparaissent dans le 2e parent

Question: Proposer un crossover qui a du sens dans un problème de coloration de graphe.

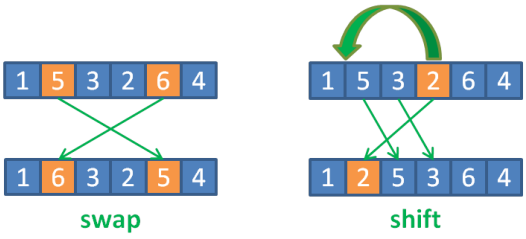
Mutation

Appliquer une légère variation à une solution

Attention

- Toute solution doit être atteignable par l'application consécutive d'une mutation
- Respecter les contraintes
  - géré par la mutation
  - technique de reconstruction

Exemple : Pour le TSP



2 exemples de mutations :

- swap
- shift

Question : Quelle mutation pourrait-on appliquer sur un problème de coloration de graphe ?

4. Opérateurs indépendants du problème

Critère d'arrêt

Indépendant de la convergence :

- un temps
- un nombre d'itérations (cycle de l'AG)
- un nombre d'évaluations

Dépendant de la convergence :

- Un nombre d'itérations sans amélioration

Remarque : Comparaison d'algorithmes

La comparaison d'algorithme n'est pas facile de part la nature non-déterministe des AG. Les résultats peuvent varier en fonction du critère d'arrêt choisi.

Exemple

Un algorithme A peut être statistiquement meilleur qu'un algorithme B pour un temps t.

Et il peut être moins bon pour un temps t+1.

Sélection

L'opérateur de sélection a pour but de sélectionner la population "parents".

On voudra souvent respecter le principe suivant :

« Plus un individu est bon (bonne fitness) plus il aura de chance d'être sélectionné (notion d'élitisme). »

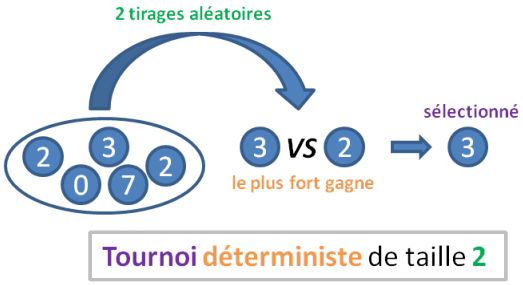
Différentes stratégies possibles :

- Roulette
- Tournoi

Exemple

La roulette : chaque solution de fitness f a une chance d'être sélectionnée avec une probabilité de $f / \sum fitness.$

Exemple



Tournoi déterministe : N solutions sont tirées aléatoirement (équiprobabilité).

La meilleure solution (meilleur fitness) est sélectionnée.

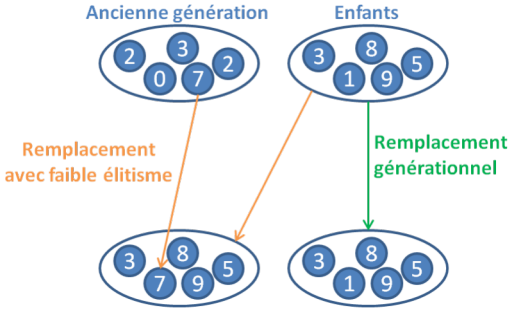
Question : Étant donné un pool de 3 solutions de fitness 1, 2 et 3 (maximisation) quelle est la méthode (parmi les deux exemples précédents) où la solution de fitness 3 a la plus forte probabilité d'être sélectionnée.

Remplacement

Consiste à choisir parmi l'ancienne et la nouvelle génération, quels individus vont participer à la prochaine génération...

Différentes stratégies existent avec plus ou moins d'élitisme...

Exemple



Un remplacement générationnel consiste à ne garder que la population "enfants".

Un remplacement avec élitisme consiste à garder le (ou les) meilleur(s) "individu (s)" de l'ancienne génération à la place du (ou des) moins bon(s) "enfants".

5. Problématique

Intensification vs diversification

Ces algorithmes ont pour but de traiter des problèmes ayant un vaste espace de recherche.

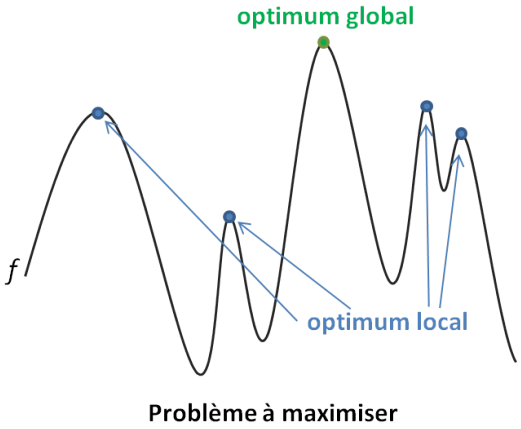
Il faut donc assurer qu'ils explorent cet espace de toutes parts -> diversification

Il faut aussi s'assurer qu'on améliore au mieux les solutions -> intensification

Ces 2 notions contradictoires dépendantes de nombreux paramètres vont affecter, pour un problème donné, l'efficacité de l'algorithme génétique.

Attention : Trop d'intensification

Un algorithme qui intensifie trop ses solutions risque de converger prématurément vers un optimum local (solution optimale dans un espace restreint mais pas optimale dans l'ensemble de l'espace de recherche).



Problème à maximiser

Attention : Trop de diversification

Un algorithme qui est trop diversifié risque de ne jamais converger vers de bonnes solutions

Exemple : Pousser l'élitisme à l'extrême

Utiliser une sélection par tournoi déterministe de taille N (N de l'ordre de la taille de la population).

Quel comportement peut-on observer ?

Exemple : Faire trop varier les solutions

Utiliser des taux de croisement et de mutation élevés (proche de 100%)

Quel comportement peut-on observer ?

6. Conclusion

- Les algorithmes génétiques sont des méthodes de résolutions contenant beaucoup d'opérateurs reposant sur des stratégies diverses et variées et sur de nombreux paramètres.
- Chaque problème a des spécificités différentes.

Remarque

Un algorithme A (opérateurs et paramétrage fixés) peut donc être bon pour résoudre un problème X et très mauvais pour résoudre un problème Y.

Attention

THE algorithme n'existe pas ! ! !

Les AGs comme l'ensemble des métaheuristiques sont donc des schémas de résolution dont il faut bien comprendre les mécanismes afin de pouvoir répondre à un vaste ensemble de problèmes...

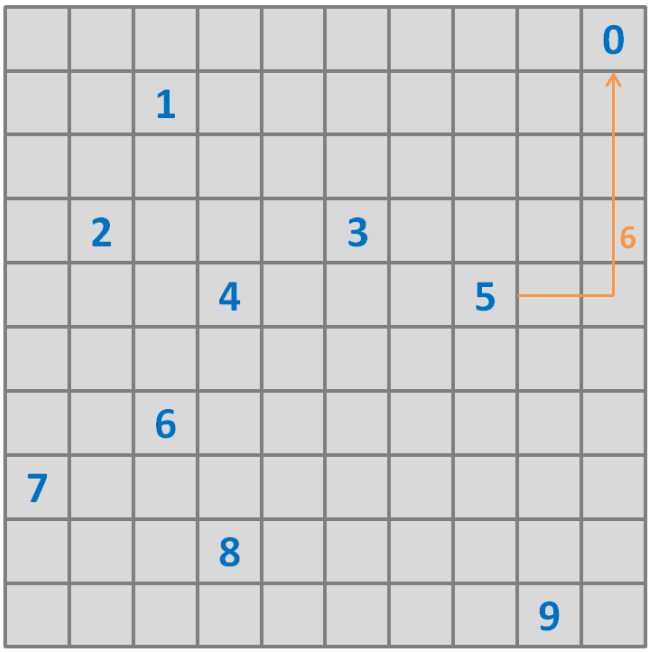
C'est à vous de les construire pour qu'elles soient efficaces pour vos problèmes...

Comment faire un bon algorithme ?

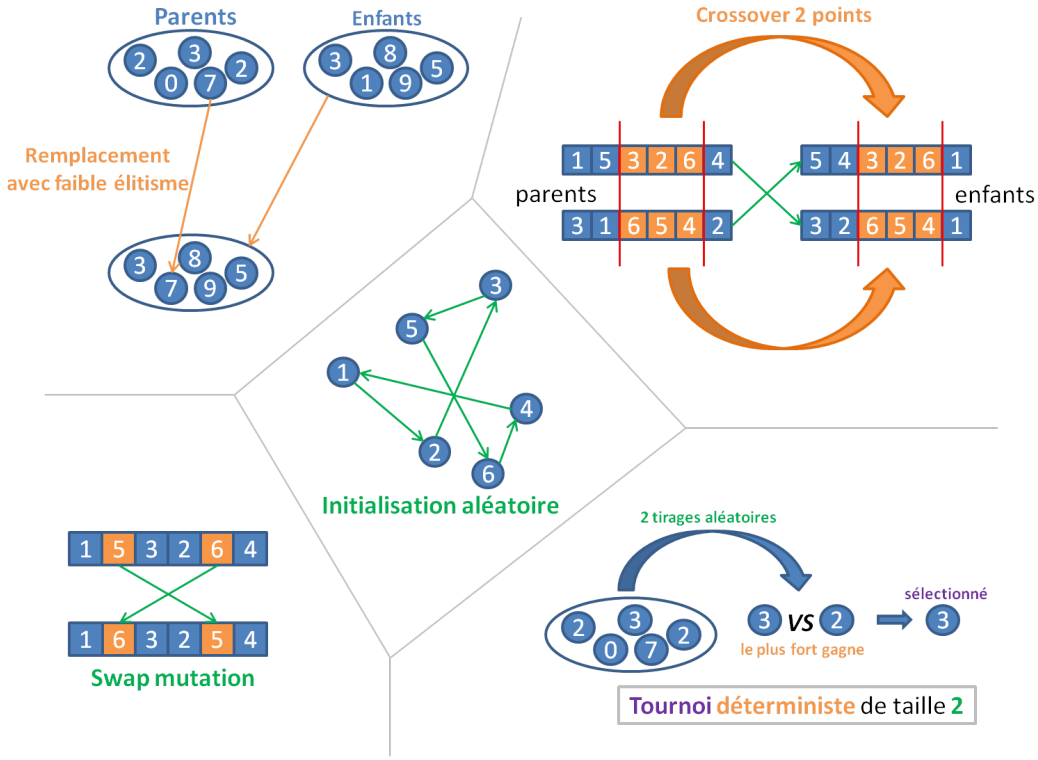
- Savoir ce que fait vraiment notre algorithme (comment chaque opérateur va influencer la résolution)
- Connaître les difficultés du problème
    - avoir un expert sur le sujet
    - faire une analyse de paysage
- Expérimenter

7. Exercice coopératif

Nous allons dérouler quelques générations d'un AG sur le problème du TSP. Chaque groupe (10 en tout) gérera une solution de la population...



TSP de 10 villes avec une distance de manhattan



Question : Trouvez un moyen efficace de générer une permutation (équiprobabilité) avec un dé 10.

IV Recherches Locales

1. Recherches locales

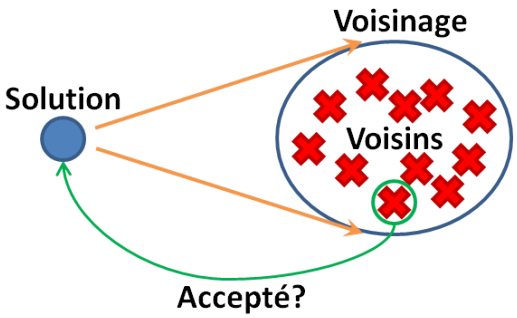
Idée générale

Dans le but d'optimiser une solution **S**, une recherche locale va faire évoluer **S** pas à pas en lui appliquant une légère variation.

Repose sur 3 notions

- **Voisin** : mouvement(variation) + sauvegarde d'informations : fitness (voir plus)
- **Voisinage** : définit comment construire tous les voisins
- **Évaluation** : affecte la fitness (voir plus) d'un voisin, peut être complète ou incrémentale

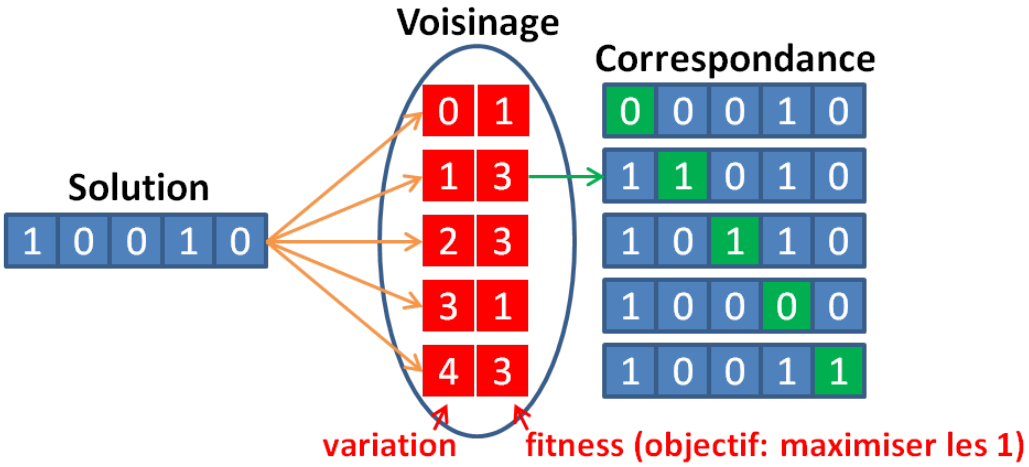
Schéma général d'une recherche locale



Explorateur :

- générer un ou plusieurs voisins du voisinage
- sélectionner un voisin
- décider d'appliquer à la solution, la variation correspondant au voisin sélectionné

Exemple : Voisinage 1 bit



Question

Trouvez tous les voisins (juste la variation) d'un voisinage 2 bits pour une solution de taille 4.

Algorithme simplifié d'une recherche locale

```
#Constructeurs
Voisinage(Voisin);
Explorateur(Voisinage, Evaluation);
#Algo
do {
    explorateur(sol);
    if(explorateur.accept(sol))
       explorateur.move(sol);
}
while(explorateur.continue(sol))
```

Recherches locales classiques

Toutes les recherches locales classiques de la littérature entrent dans ce schéma.

L'algorithme complet prend en compte la gestion de paramètres et des critères d'arrêt externes à l'explorateur.

Voici une liste non exhaustive de recherches locales que nous allons voir par la suite :

- Méthodes de descente
- Marches aléatoires
- Recuit simulé
- Recherche Tabou
- Recherche locale itérée
- Recherche locale à voisinages variables

2. Méthodes de descente

Méthode de descente simple

On génère tous les voisins du voisinage et on sélectionne le meilleur (meilleure fitness) :

- Si le meilleur voisin améliore la fitness de la solution, on applique la variation et on continue
- Sinon la recherche s'arrête

Méthode de descente neutre

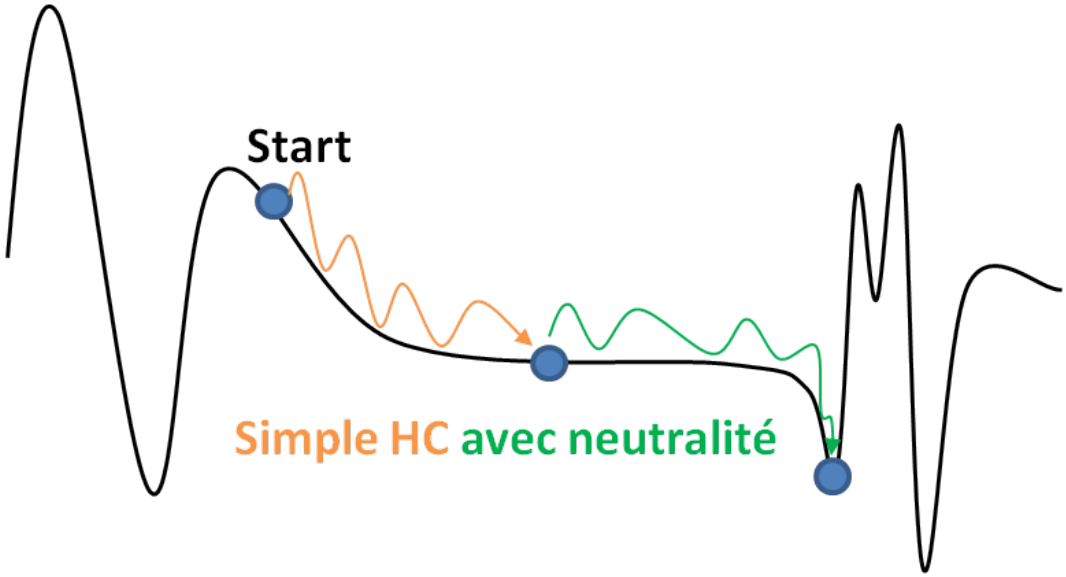
Idem que la méthode précédente sauf qu'on accepte aussi un voisin de même fitness que la solution courante.

Méthode de descente avec première amélioration

On génère les voisins un par un jusqu'à ce qu'on trouve un voisin améliorant

- Si un voisin améliorant est trouvé, on applique la variation et on continue
- Sinon la recherche s'arrête

Exemple



Remarque

Les méthodes de descente précédentes sont les plus classiques et les plus utilisées mais il existe d'autres variantes

Question

On optimise un oneMax (plus grand nombre de bit à 1) avec une méthode de descente avec 1ere amélioration utilisant un voisinage 1 bit.

On génère la solution suivante : 10010

Décrivez toutes les étapes de la recherche (voisins générés, variations appliquées, etc.) jusqu'à ce que le critère d'arrêt soit atteint.

3. Marches aléatoires

Marche aléatoire

Pendant un nombre d'itérations fixé, on choisit un voisin aléatoirement dans le voisinage et on applique la variation correspondante.

Marche neutre aléatoire

On choisit un voisin aléatoirement parmi les voisins ayant la même fitness que la solution courante et on applique la variation correspondante.

Remarque : Utilisation

Les marches aléatoires sont en général utilisées pour explorer l'espace de recherche afin d'obtenir de l'information sur le problème.

4. Recuit simulé

Principe

Un recuit simulé va sélectionner un voisin dans le voisinage. Deux cas se présentent :

- si le voisin améliore la solution courante alors on l'accepte
- sinon on l'accepte avec une certaine probabilité qui va décroître tout au long de la recherche

La température

La température T symbolise l'avancement de la recherche.

Cette température va décroître toutes les X itérations de la recherche jusqu'à un certain seuil S:

$T = T * \alpha, 0 < \alpha < 1$

Une fois S atteint, la recherche s'arrête.

algorithme

On initialise T, S, α et une solution de fitness f (aléatoirement par exemple).

On génère le 1er voisin du voisinage de fitness v.

Si il améliore la solution courante, on applique la variation.

Sinon on applique la variation avec une probabilité p.

$p = \exp((v - f)/T)$ (en maximisation)

$p = \exp((f - v)/T)$ (en minimisation)

On met ensuite à jour la température.

La recherche continue tant que le seuil S n'est pas atteint.

Remarque

Contrairement aux méthodes de descente, cet algorithme peut accepter un voisin de moins bonne qualité.

Cela permet potentiellement à la recherche de sortir d'un optimum local.

5. Recherche Tabou

Principe

Une recherche tabou va garder en mémoire dans une **liste tabou**, les éléments qu'elle a déjà explorés dernièrement pendant la recherche.

La liste tabou va être mise à jour à chaque itération de la recherche.

Les éléments de la liste ne pourront pas être sélectionnés sauf si ils vérifient un **critère d'aspiration**.

La recherche va, à chaque itération, chercher le meilleur voisin non tabou ou qui vérifie le critère d'aspiration afin de faire évoluer la solution courante.

Construction

Une recherche tabou nécessite deux éléments essentiels :

- la liste tabou
- le critère d'aspiration

et peut aussi comprendre deux éléments facultatifs :

- une fonction d'intensification
- une fonction de diversification

La liste tabou

Contient jusqu'à N éléments en mémoire.

Les éléments peuvent être de différents types (le plus souvent des voisins ou des solutions).

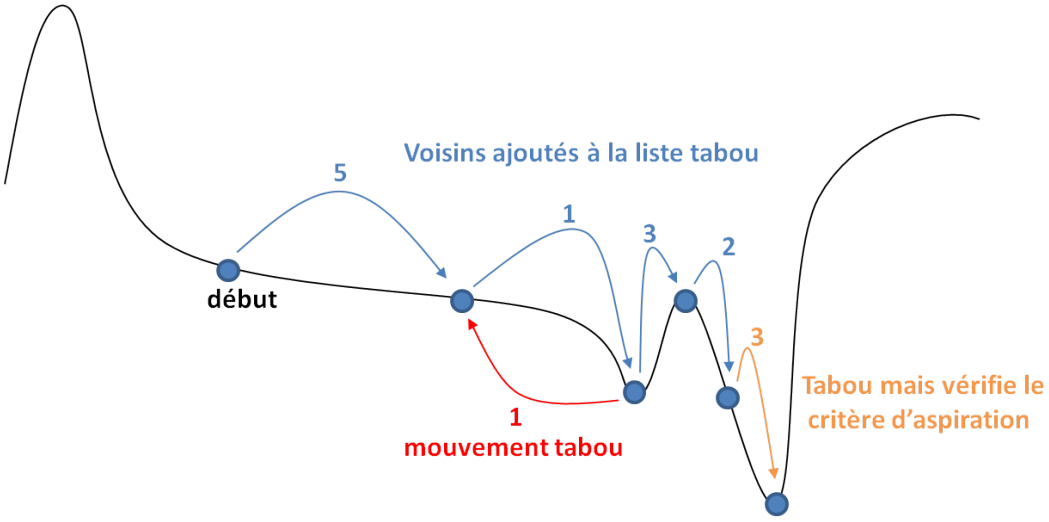
A chaque itération le nouvel élément exploré va être ajouté à la liste tabou.

Si la liste est trop grande (>N), l'élément le plus ancien sera enlevé de la liste.

Le critère d'aspiration

Si ce critère est vérifié, la recherche va autoriser un élément tabou.

Un critère souvent utilisé étant d'autoriser d'explorer une solution qui est meilleure que toutes celles explorées auparavant.



Fonctions d'intensification et de diversification

Des opérations d'intensification et de diversification peuvent être appliquées à la solution courante sous certaines conditions.

Ici tout peut être envisagé.

Exemple

On pourrait par exemple vouloir redémarrer d'une solution aléatoire afin d'explorer d'autres endroits de l'espace de recherche après une longue période sans amélioration.

6. Recherche locale itérée

Principe

Appliquer une recherche locale.

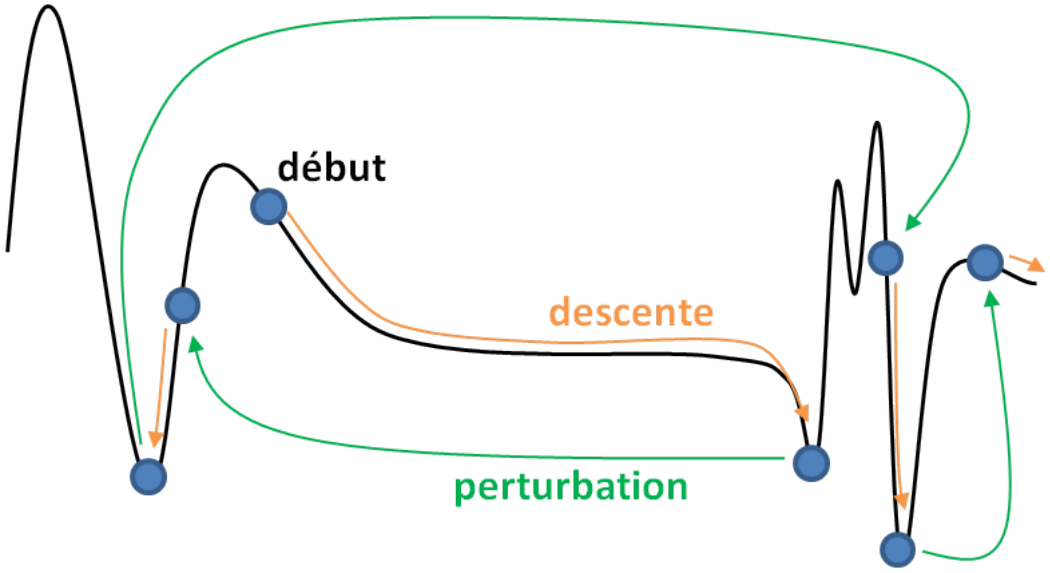
Perturber la solution trouvée et relancer la recherche locale.

Répéter ce schéma un certain nombre de fois tout en gardant en mémoire la meilleure solution trouvée.

Perturbation

Cela consiste à appliquer un changement à la solution trouvée par la recherche.

On pourra par exemple appliquer un certain nombre de mutations ou bien réinitialiser la solution aléatoirement.



7. Recherche à voisinages variables

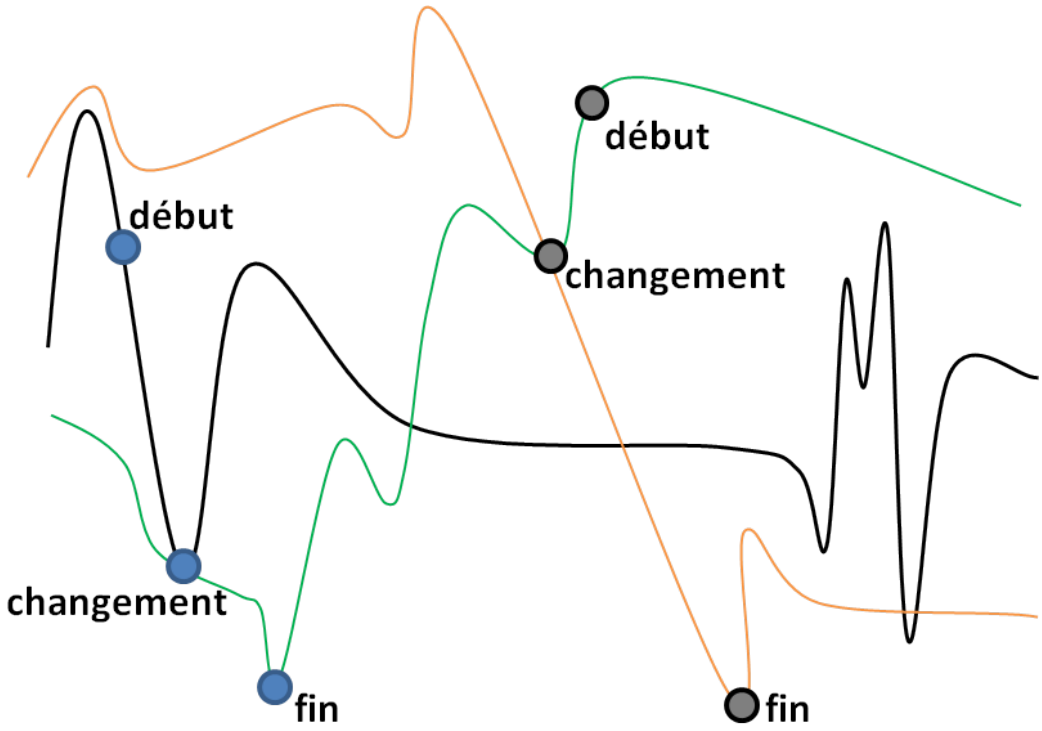
Principe

Repose sur le principe qu'un optimum local pour un voisinage ne l'est pas forcément pour un autre voisinage.

La recherche va donc changer de voisinage pour essayer de se "débloquer".

Il faudra donc définir plusieurs voisinages pour le problème.

Par exemple 3 voisinages : 1 bit, 3 bits, 5 bits.



8. Conclusion

De nombreuses façons de faire

Il existe beaucoup de recherches locales qui ont des particularités et des complexités différentes.

Pour certains problèmes, une méthode simple pourra être tout aussi efficace qu'une recherche plus complexe.

Comme les algorithmes génétiques, certaines recherches locales nécessitent d'être "tunées" afin d'être plus efficaces.

Quel algorithme choisir ?

Encore une fois l'algorithme parfait n'existe pas.

En plus de la connaissance et la compréhension des méthodes de résolution, il faudra un gros travail d'analyse du problème afin de le résoudre le plus efficacement possible.

V Optimisation multi-objectifs

1. Motivations

Les problèmes réels sont multi-objectifs...

Transport

- Coût
- Distance
- Temps

Ordonnancement

- Temps de fin
- Retard cumulé

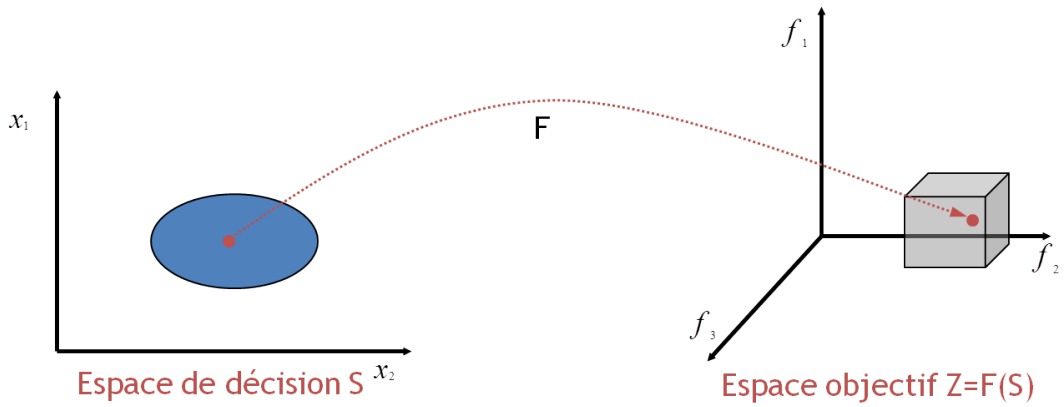
Portefeuille

- Profit
- Risque

2. Problème d'optimisation multi-objectifs

Un problème d'optimisation multi-objectifs (MOP) peut être défini comme :

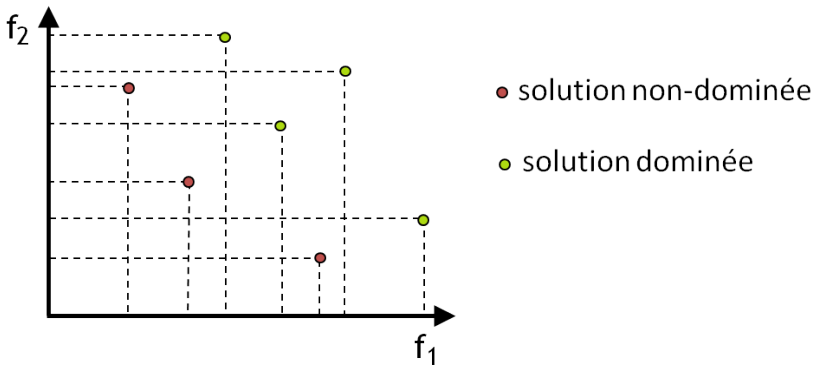
$$(MOP) = \begin{cases} min\ F(x) = (f_1(x), f_2(x), ..., f_n(x)) \\ x \in S \end{cases}$$



3. Dominance Pareto

Définition : Dominance Pareto

Dans un cas de minimisation, un vecteur objectif $u = (u_1, ..., u_n)$ domine un vecteur objectif $v = (v_1, ..., v_n)$ (noté $u \prec v$) si et seulement si aucun composant de v n'est plus petit que le composant correspondant de u et que au moins un composant de u est strictement meilleur, i.e. $\forall i \in 1, \ ..., n \ : u_i \leq v_i \wedge \exists i \in 1, ..., n \ : u_i < v_i$.



Définition : Optimalité Pareto

Une solution x^* est Pareto optimal si pour tout $x \in S, F(x)$ ne domine pas $F(x^*)$

Définition : Ensemble Pareto Optimal

Pour un MOP (F, S) donné, l'ensemble Pareto optimal est défini comme $P^* = x \in S | \nexists \ x' \in S, F(x') \prec F(x)$.

Question

Soit un MOP à 3 objectifs à minimiser et les vecteurs objectifs suivants:

(1,3,4) ; (4,2,1) ; (1,3,1) ; (2,3,1) ; (2,2,2) ; (4,4,0)

Quelles sont les solutions pareto?

4. Aide à la décision

- A priori
  - Préférences => Recherche
  - Utilisation de connaissances a priori du problème
- A posteriri
  - Recherche => Préférences
  - Choix à partir d'un ensemble Pareto (réduit)
- Interactive
  - Recherche <=> Préférences

5. Méthodes de résolutions

- Méthodes exactes
  - Cas simples
  - Problèmes à 2 objectifs de petite taille
- Métaheuristiques
  - Trouver une bonne approximation de l'ensemble Pareto
  - Comment gérer un très grand nombre de solutions ?

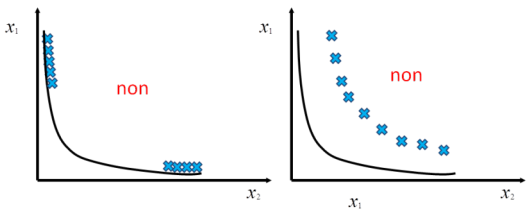
Plusieurs approches

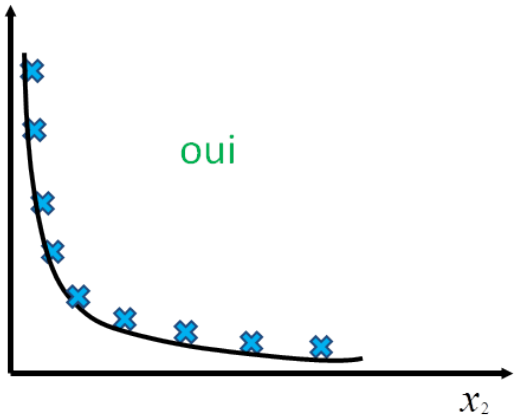
- Approches scalaires
  - méthodes d'agrégation des objectifs
- Approches basées sur les indicateurs de qualité
- Approches basées sur la dominance

6. But des métaheuristiques

Approximer la front Pareto est, en soit, un problème bi-ojectifs :

1. Minimiser la distance avec le front Pareto optimal
2. Maximiser la diversité dans l'espace de décision





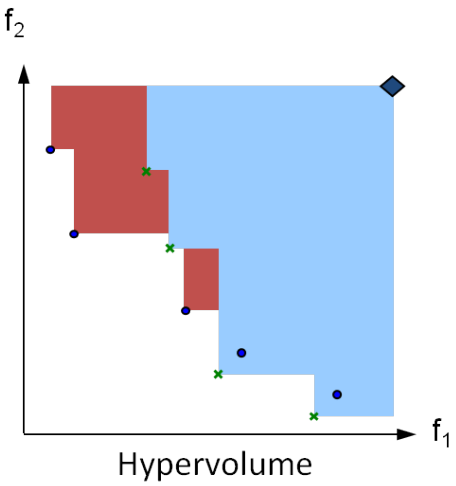
1. Bonne convergence de l'approximation
2. Bonne diversité de l'approximation

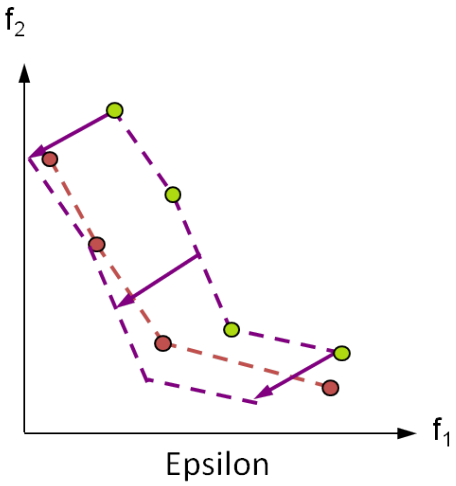
7. Indicateurs de qualité

Remarque

Il existe de nombreuses façons de mesurer la qualité d'un ensemble ayant chacune ses caractéristiques

Deux indicateurs souvent utilisés



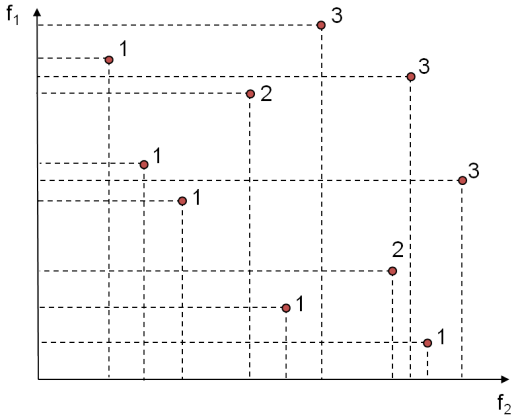


8. Algorithmes génétiques

Design

- Assignation de la fitness
  Guider la recherche jusqu'aux solutions "pareto optimale" afin d'avoir une bonne convergence
- Préservation de la diversité
  Générer un ensemble Pareto bien diversifié dans l'espace objectif
- Élitisme
  Préserver et utiliser les meilleurs solutions

Assignation de la fitness

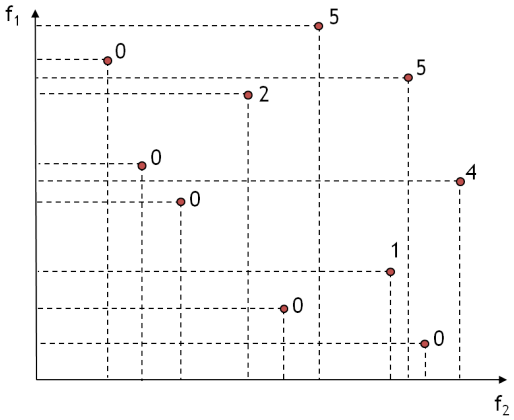


Dominance Depth :

Solutions classifiées en différents fronts (utilisée dans NSGA-II)

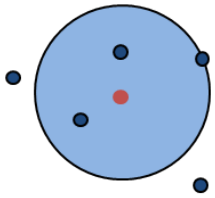
Dominance Rank :

Fitness(x) = nombre de solutions qui dominent x (utilisée dans MOGA)

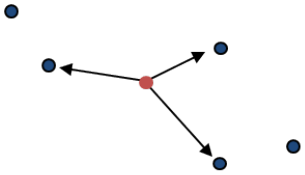


Quelques exemples d'affectation de la diversité :

En fonction des solutions voisines par rapport à une distance donnée



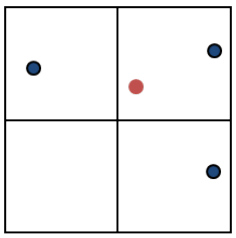
noyau



3 plus proches voisins

En fonction de la distance à son k^e plus proche voisin

En divisant l'espace à l'aide d'une grille



histogramme

Élitisme

Empêcher la perte des solutions non-dominées.

Deux approches :

- Dans la population principale : les solutions non-dominées survivent à l'étape de remplacemen
- Dans une seconde population : l'archive

Archive

L'archive sauvegarde les solutions non-dominées.

Elle est mise à jour à chaque itération de l'AG en fonction d'une relation de dominance.

Remarque

L'archive peut être utilisée par le processus de recherche.

ex : Utilisée en complément de la population pour l'étape de sélection

Management de l'archivage

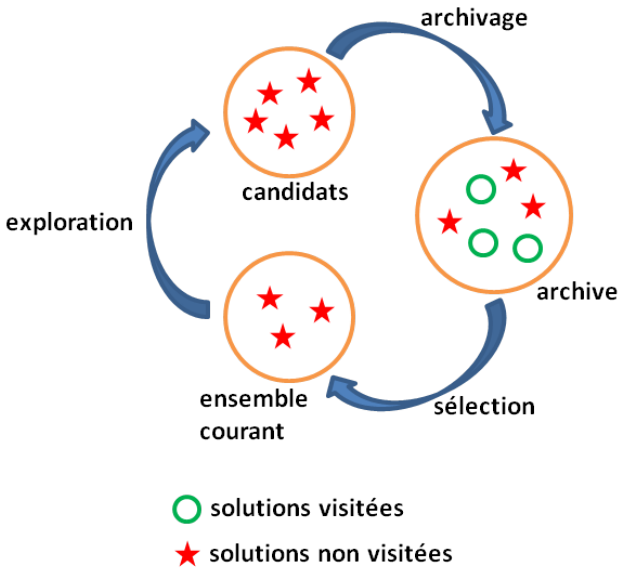
- Pas d'archive
- Archive non-bornée
- Archive bornée
- Archive de taille fixe

9. Recherches locales

Recherches locales multi-objectifs basées sur une relation de dominance (DMLS)

Design :

- Sélection
- Exploration (voisinage)
- Elitisme



- Sélection
  Sélectionner de une à toutes les solutions de l'ensemble des solutions non-dominées.
- Exploration
  Étant donné un voisinage V, explorer un à tous les voisins de V.

- Élitisme
  Sauvegarder l'ensemble des solutions non-dominées à l'aide d'une archive.

Exemple

$DMLS^{1,*}$ sélectionnera une solution de l'ensemble des solutions non-dominées, explorera la totalité de son voisinage et archivera le nouveau front à chaque itération.

$DMLS^{*,1}$ sélectionnera toutes les solutions de l'ensemble des solutions non-dominées, explorera un seul voisin et archivera le nouveau front à chaque itération.

Remarque

Différentes stratégies peuvent être envisagées pour sélectionner un certain nombre de solutions ou explorer une certaine partie du voisinage.

Question

Que pensez vous de $DMLS^{*,*}$?

10. Conclusion

Pour aller plus loin

- Hybridation
- Parallélisation
- Méthodes distribuées
- Analyse de paysage
- Tuning
- Analyse des résultats
- etc.