Variantes (5) : Recherche Tabou

Tabu Search

- **Constat :**
  - Quand on est sur un optimum local : les solutions voisines sont toutes de moins bonne qualité ➔ bassin d'attraction
  - Glover 1986 / Hansen 1986
- **Idée :**
  - Sortir du bassin d'attraction en acceptant des solutions de moins bonne qualité
    - Choisir le meilleur voisin même si non améliorant
  - Mais interdire de revisiter des solutions déjà explorées
- Structure pour mémoriser des informations sur les solutions visitées, appelée **Liste Tabou** pendant un certain nombre d'itérations

Variantes (5) : Recherche Tabou

- **Stratégie d'exploration :**
  - Introduire une liste L (initialement vide)
  - A chaque itération : ajouter le dernier mouvement effectué dans L
  - Choisir une solution voisine s′ telle que :
    - Le mouvement $s \rightarrow s' \notin L$ -- **diversification**
    - Le cout $f(s')$ soit minimal -- **intensification**
  - **Algorithme**

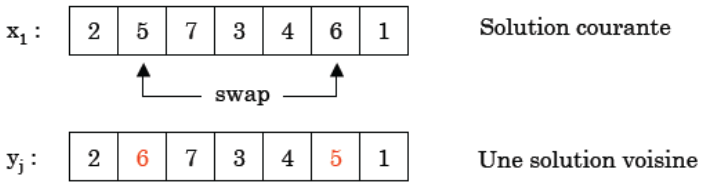
```
1.  s ← solution initiale
2.  best ← s
3.  L ← ∅      // Tabu
4.  while (Conditions) loop
5.     s' ← Meilleur-Voisin(N(s), L)  // Voisin non tabou
6.     if f(s') < f(best) then
7.         best ← s'
8.     end if
9.     Actu_Tabu(s', L)
10. end while
11. Return best
```

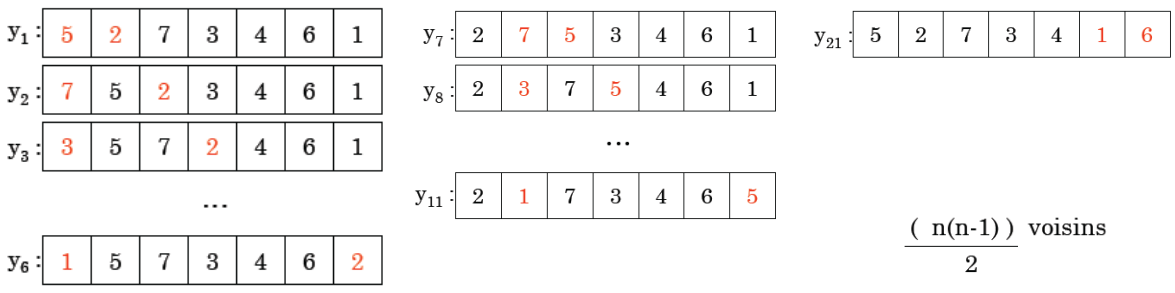
Variantes (5) : Recherche Tabou

- **« Liste » Tabou**
  - **Conserver les mouvements effectués** et non pas les solutions visités
    - Exemple : variables échangées (swap)
    - Plus rapide à vérifier et moins couteux à mémoriser …
  - **Est parcourue fréquemment** dans la recherche de solutions
    - Accès efficace pour vérifier si une solution est tabou
      - Table de hachage (sur les mouvements, sur la fonction objectif)
        - Si collision : Taille de la liste trop petite
  - **Ne pas déconnecter** la solution optimale de la solution courante
    - Les informations restent dans la liste pendant une durée limitée (ie un nombre d'itérations)

Exemple

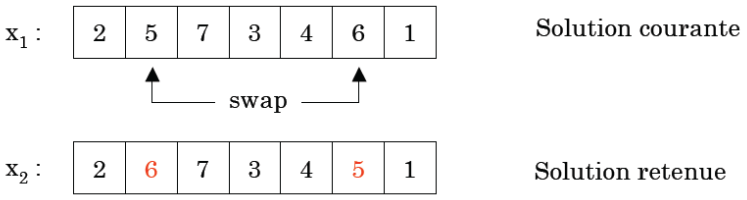
- **Exploration d'un voisinage**





Exemple

- **Sélection d'un voisin**

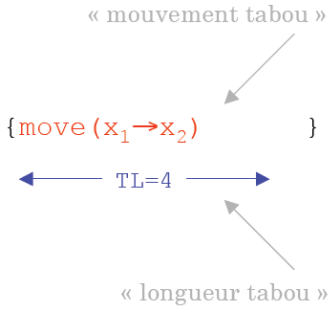


- **Mouvement «Tabou »**

Déclarer le mouvement

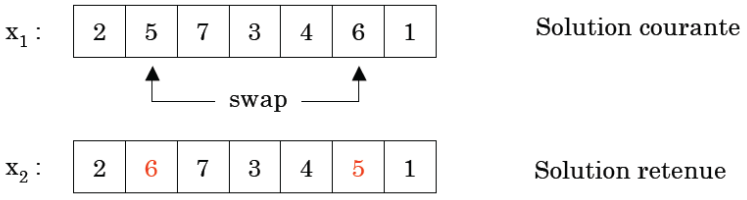
5 ⇄ 6

tabou durant #n itérations



Exemple

- **Sélection d'un voisin**



- **Structure pour les mouvements «Tabou »**

	2	3	4	5	6	7
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0		
4	0	0	0			
5	0	0				
6	0					

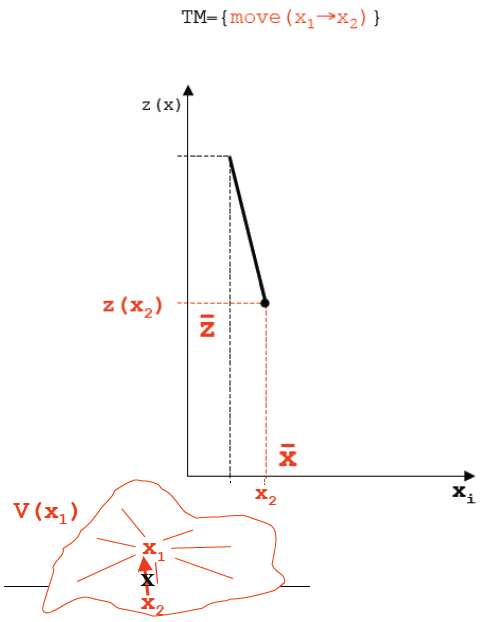
Déclarer le mouvement

5 ⇄ 6

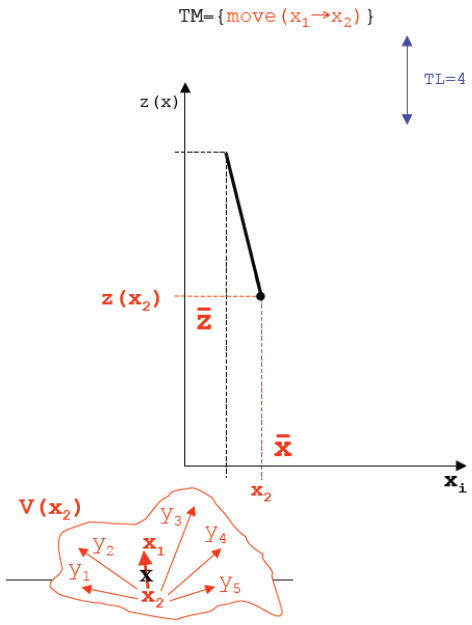
tabou durant #n itérations

	2	3	4	5	6	7
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0		
4	0	0	0			
5	4	0				
6	0					

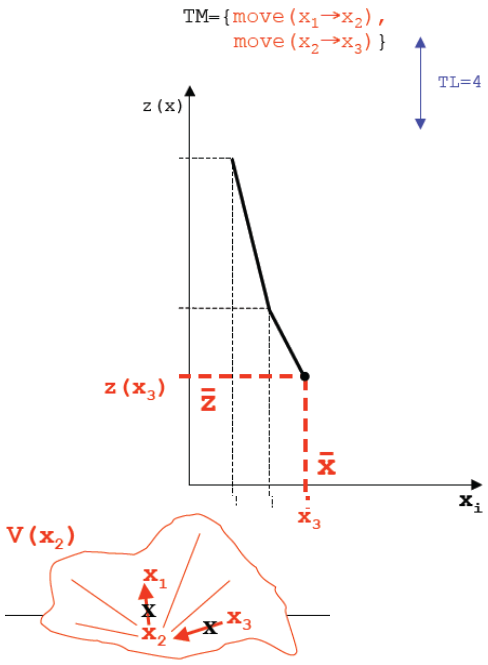
Choix d'un voisin et
ajout d'un mouvement tabou



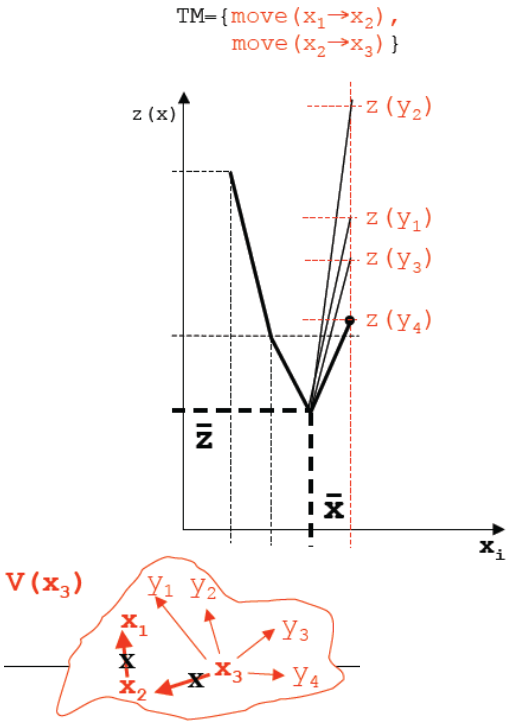
Exploration voisinage
Sauf le mouvement tabou



Choix d'un voisin améliorant et
ajout d'un mouvement tabou

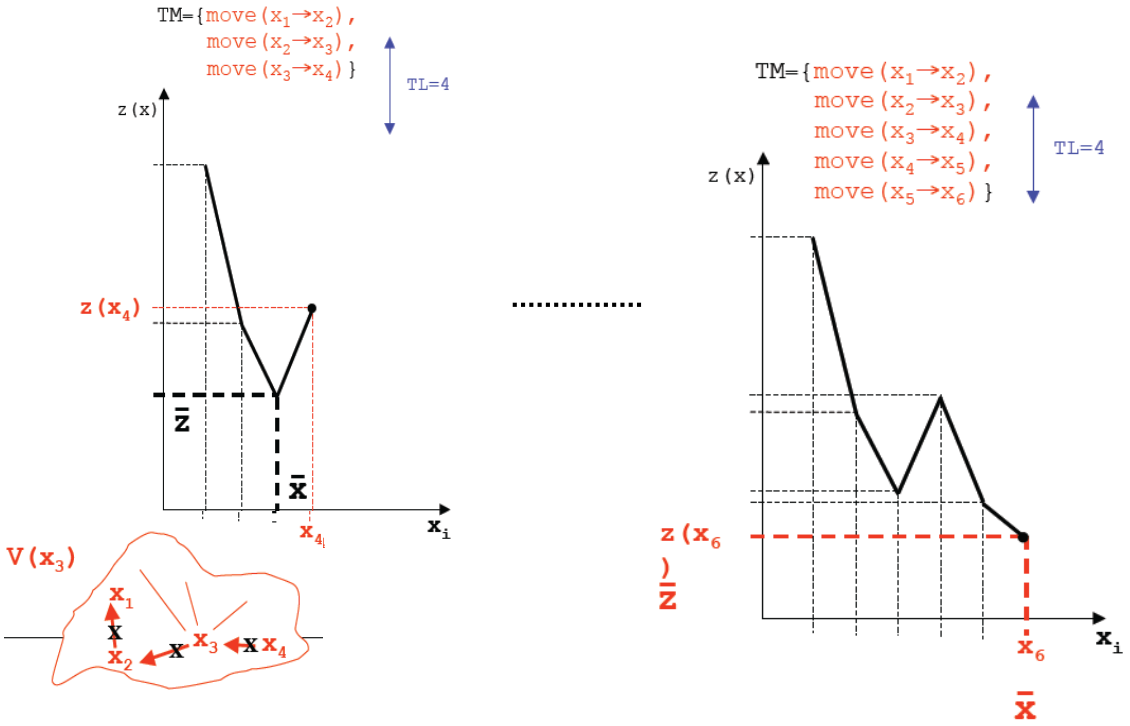


Exploration voisinage
Sauf le mouvement tabou
Aucun voisin améliorant



Choix d'un voisin et
ajout d'un mouvement tabou

Prise en compte de la durée des interdictions



Variantes (5) : Recherche Tabou

- **Exemple d'une liste Tabou**
  - **Mouvement effectué sur les solutions :**
    - Interdire le mouvement inverse pendant k itérations
      - Itération p : solution obtenue après swap(i, j)
      - Interdire swap(j, i) jusqu'à itération $p + k$
      - Matrice pour mémoriser toutes les paires de swap possibles
    - Mouvement inverse peut être complexe

- **Le contenu de la liste Tabou**
  - Peut interdire plus de solutions que celles réellement explorées
  - Ne prévient pas totalement des risques de cycle

Variantes (5) : Recherche Tabou

- **Durée des interdiction**
  - **Ne conserver que les k derniers mouvements effectués**
    - Valeur de k : longueur de la liste ➔ compromis diversification / intensification
      - k faible :
        - peu de voisins interdits risque de rester bloqué sur un optimum local
      - k élevé :
        - beaucoup de voisins interdits / parcours potentiellement plus long
        - diversification importante mais on risque de louper l'optimum global
      - Réglage adaptatif en fonction du problème / d'une instance
- **Annuler une interdiction**
  - Autoriser mouvement tabou si amélioration de la fonction objectif
  - Critère d'aspiration

Variantes (5) : Recherche Tabou

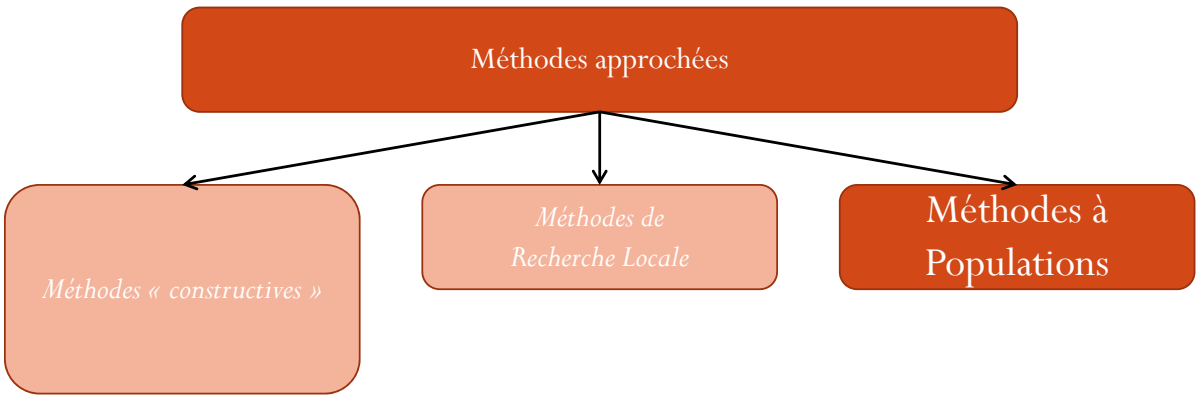
- **Conception d'une méthode Tabou**
  - Définir un voisinage et une fonction d'évaluation
  - Définir une structure pour la liste Tabou
    - Quoi mémoriser
    - Pendant combien de temps
  - Définir conditions d'arrêt
  - Mémoriser meilleure solution visitée

Variantes (5) : Recherche Tabou

- **Attention à l'exploration du voisinage**
  - Taille : se limiter si besoin à une liste de voisins candidats
    - Aléatoire
    - Les plus pertinents a priori
  - Evaluation :
    - doit être efficace (incrémentale, approchée)
- **Variante**
  - Mémoire dite à long terme pour guider la recherche
    - Mémoriser les mouvements effectués et leur qualités respectives
    - Diversification : Guider vers des parties non explorées
    - Intensification : Repartir de caractéristiques de bonnes solutions

Section 3. Méthodes approchées

- **Méthodes à population**



Méthodes à population

- **Idées :**
  - Considérer un ensemble de solutions
  - Faire évoluer cet ensemble de solutions
- **Méthodes :**
  - Algorithmes évolutionnaires (génétiques)
  - Colonie de Fourmis (Ant Colony Optimization
  - Essaim (Particle Swarm Optimization)
  - .....

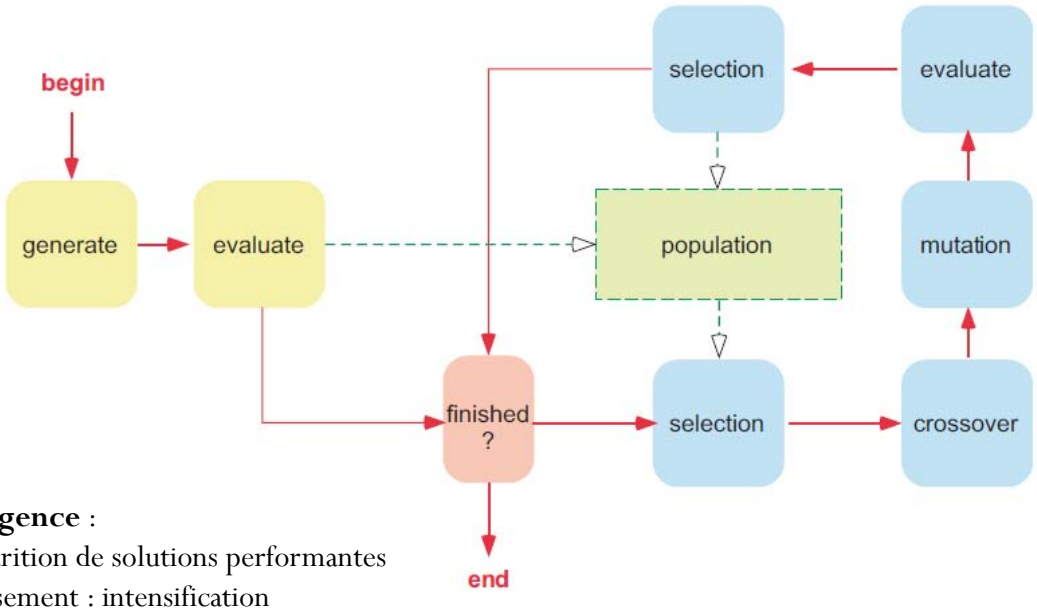
Algorithmes évolutionnaires (1)

- **Familles de méthodes inspirées des systèmes vivants**
  - Algorithmes génétiques : année 1970 / 1980 -- J. Holland 1975
  - Pour la résolution de problèmes d'optimisation (continue / discrète)
- Principe :
  - Faire évoluer progressivement un ensemble de solutions via des opérateurs « stochastiques » inspirés des processus de sélection naturelle et d'évolution génétique (Darwin)
    - Un enfant « hérite » du patrimoine génétique de ses deux parents
    - Des mutations génétiques (positives ou négatives) peuvent modifier certaines gènes
    - Parmi les descendants : seuls les plus « adaptés » à l'environnement survivent
    - Le hasard joue un rôle important pour produire des enfants différents de leurs parents
    - La sélection naturelle effectue le tri entre les évolutions favorables et celles qui ne le sont pas

Algorithmes évolutionnaires (2)

- **Vocabulaire** :
  - Solution = un individu ayant une évaluation (fitness)
  - Ensemble de solutions = une population
  - Evolution de la population de solutions
    - Croisement : combiner solutions (parents) pou obtenir une nouvelle solution (enfant)
    - Mutation : modifier solutions
  - Génération = une itération d'évolution
- **Mécanismes :**
  - Evaluation des solutions
  - Sélection des meilleures solutions
  - Croisements
  - Mutations

Algorithmes évolutionnaires (3)



Convergence :

- Apparition de solutions performantes
- Croisement : intensification
  - Les meilleures solutions « parents » donnent les meilleures solutions « enfants »
- Mutation : diversification

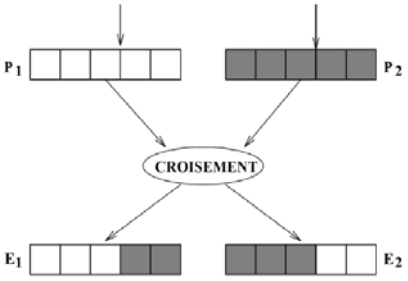
Algorithmes évolutionnaires (4)

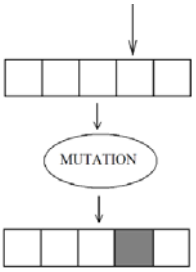
- **Algorithme général**

```
1.  Initialiser une population de solutions
2.  Evaluer les individus
3.  while (conditions) loop
4.     Sélectionner parents
5.     Combiner parents pour produire enfants
6.     Modifier enfants
7.     Evaluer les nouveaux individus
8.     Sélectionner la nouvelle population
9.  end while
10. return Meilleur individu
```


Algorithmes évolutionnaires (4)

- **Points clés de la méthode**
  - Codage des solutions
  - Génération d'une population initiale
  - Processus de sélection :
    - parents
    - nouvelle population
  - Opérateurs de mutation et de croisement
  - Paramètres :
    - Taille de la population
    - Critères d'arrêt
    - Probabilité de mutation





Représentation des solutions / Evaluation

- **Représentation des solutions**
  - Variables discrètes :
    - codage binaire

0	0	1	1	0	1	0	1

    - permutation

D	A	F	G	H	C	E	B

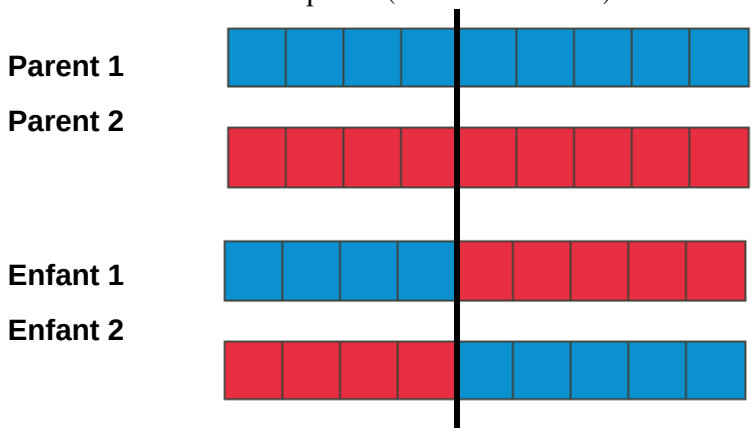
  - Fonction d'évaluation :
    - Mesure du score de chaque solution
      - Fonction objectif ou autres mesures
    - est utilisée pour le processus de sélection (solutions parents; nouvelle génération)
    - Attention au cout de calcul
      - Impact du codage sur la fonction d'évaluation

Population de solutions

- **Population de solutions :**
  - Ensemble de solutions réparties dans l'espace de recherche
    - Génération aléatoire
    - Heuristique gloutonne
    - Solution existante
  - Introduire de bonnes solutions
- **Taille population**
  - Si trop petite : perte de diversité
  - Si trop grande : temps de calcul important
  - Paramètre à régler expérimentalement

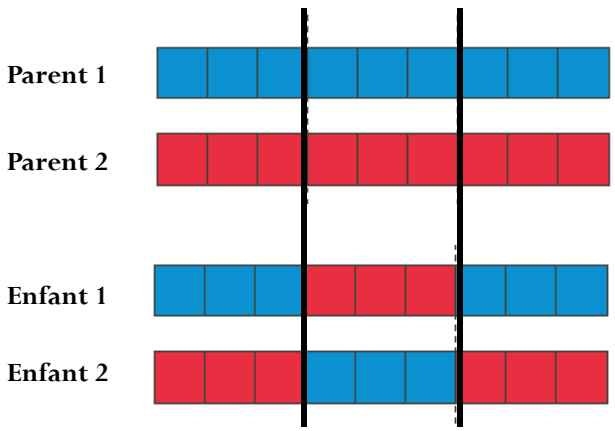
Evolution de solutions : croisement (1)

- **Evolution de solutions : combiner 2 solutions**
  - Croisement :
    - Découper le vecteur associé à chaque solution en k morceaux
    - Recombiner les morceaux pour obtenir de nouvelles solutions
    - Croisement à 1 point (choisi au hasard)

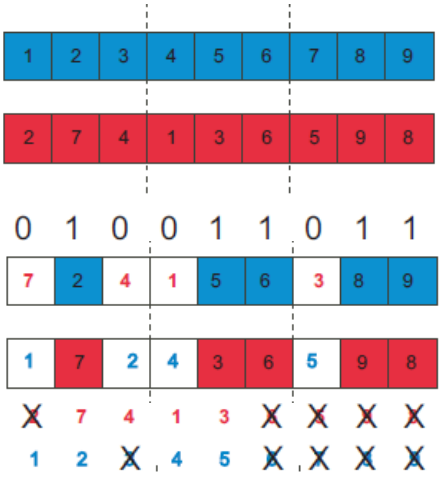


Evolution de solutions : croisement (2)

Croisement multi-point



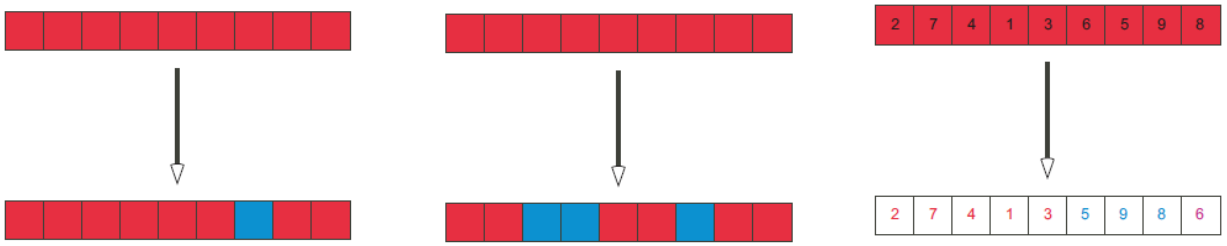
Masque



Evolution de solutions : mutation

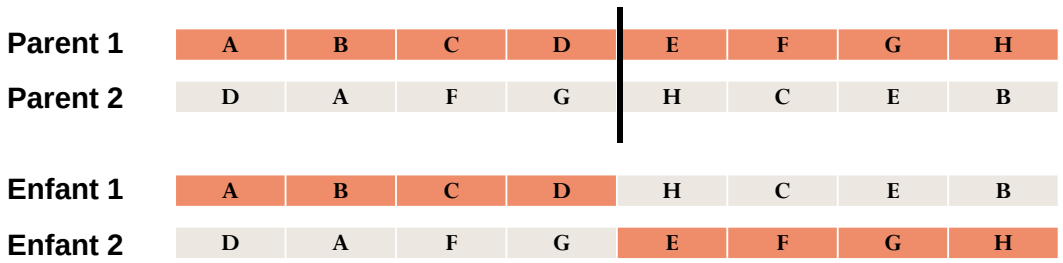
- **Mutation**
  - Perturber une solution de manière aléatoire
    - Probabilité assez faible

Mutation 1 élément **Mutation plusieurs éléments** **Décalage**

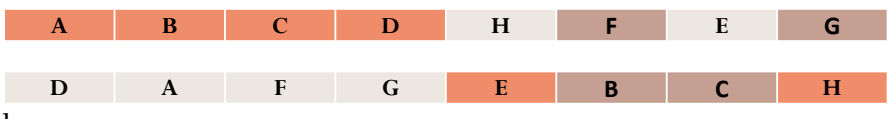


Evolution de solutions

- **Croisement et Mutation peuvent produire des solutions non réalisable**
  - Exemple : vecteur = une permutation (TSP)



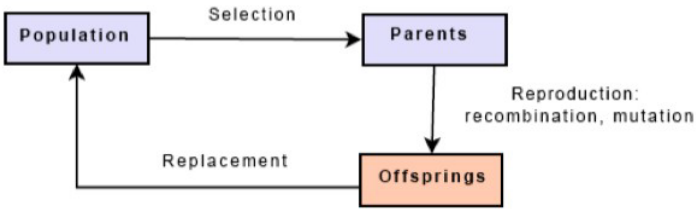
  - **Attention** : les enfants ne sont pas des solutions réalisables



    - Les réparer
    - Recherche locale

Sélection (1)

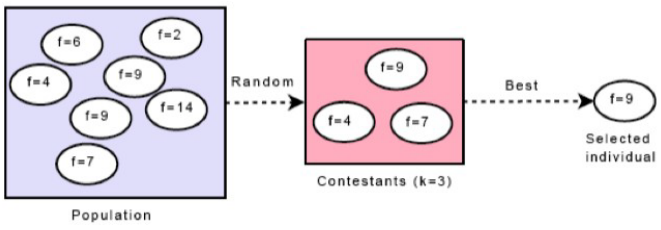
- **Deux étapes de sélection**
  - Sélection de solutions parents
  - Sélection pour nouvelle population
- **Sélection de parents :**
  - Fixer un nombre de solutions-enfants à générer



- Elitisme :
  - Sélectionner uniquement les solutions les plus performantes / fitness
  - Risque de convergence prématurée de la méthode

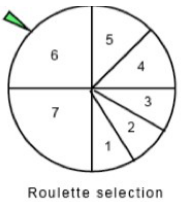
Sélection (2)

- **Sélection de parents :**
  - Par tournoi (2 à 2) :
    - choisir une paire de solutions au hasard et conserver la meilleure. Itérer jusqu'à avoir suffisamment de solutions sélectionnées



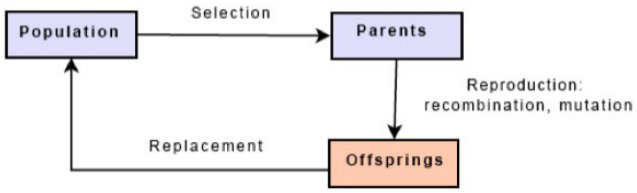
  - Par roulette/par rang :
    - associer une valeur / un rang à chaque solution. Tirer au hasard de telle sorte qu'un individu important ait une probabilité de sélection plus forte

Individuals:	1	2	3	4	5	6	7
Fitness:	1	1	1	1,5	1,5	3	3



Sélection (3)

- **Sélection (remplacement) d'une nouvelle population :**



  - A la fin d'une itération :
    - Solutions de la populations initiales
    - Solutions obtenues par combinaison / mutation
  - Nouvelle population :
    - Choisir les μ meilleures solutions parmi les l enfants générés
    - Choisir les $\mu + l$ meilleures solutions parmi les parents + les enfants
    - Taille constante

Variantes

- **Variantes :**
  - Très nombreuses
    - Liées aux différents paramétrages / options
      - Plein de noms de méthodes ….
    - Algorithmes génétiques : méthode « de base »
      - Codage binaire principalement
    - Programmation génétique :
      - Espace de recherche de très grande taille
      - Parallélisation calculs
    - Processus « d'éducation » des solutions enfants
      - Ajouter un mécanisme de recherche locale pour la mutation
    - Fonction d'évaluation : fonction objectif + diversité des solutions

Variantes

- **Algorithme général**

```
1.  Générer une population P de solutions
2.  Réparer P (recherche locale)
3.  répéter
4.      // Croisement
5.      Générer un ensemble P_c par croisement
6.      Réparer P_c
7.      // Mutation
8.      Générer un ensemble P_m par mutation
9.      Réparer P_m
10.     // Nouvelle population
11.     Sélectionner la nouvelle population à partir de P, P_c, P_m
12. Jusqu'à (conditions d'arrêt)
13. return Meilleur individu
```

Méthode (noms différents …) selon les instanciations de ces paramètres

Exemple de variante

```
1.  Générer une population P de solutions de taille N
2.  G ←Nb max générations
3.  Pour g de 1 à G
4.      P_n ← ∅
5.      pour i = 1 à c_cross.N faire
6.      (p_1, p_2) ← sélectionner 2 parents
7.      e ← Croisement(p_1, p_2)
8.      si Random(x ∈ [0,1])<ρ_mut alors
9.          e ← Mutation(e)
10.     e ← RechercheLocale(e)
11.     P_n ← P_n ∪ {e}
1.      P ← Remplacement(P, P_n)
2.  Fin pour
3.  return Meilleure solution
```

Paramètres

Taux de croisement

Croisement 1 point

Probabilité mutation

Swap

Descente (avec swap)

Elitisme

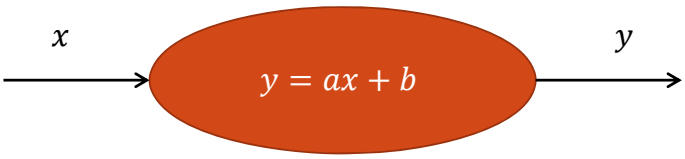
Autres variantes

- Processus « d'éducation » des solutions enfants
  - Ajouter un mécanisme de recherche locale pour la mutation
  - Algorithmes mémétiques
- Fonction d'évaluation :
  - fonction objectif + **diversité des solutions**
- Plateformes :
  - Exemple : DEAP (Distributed Evolutionary Algorithms in Python)

- Phase de validation expérimentale ….
  - Se référer à des exemples similaires de la littérature
  - Utiliser des « instanciations » types de méthodes

Application en apprentissage

- **Processus d'apprentissage supervisé**



Découvrir a et b

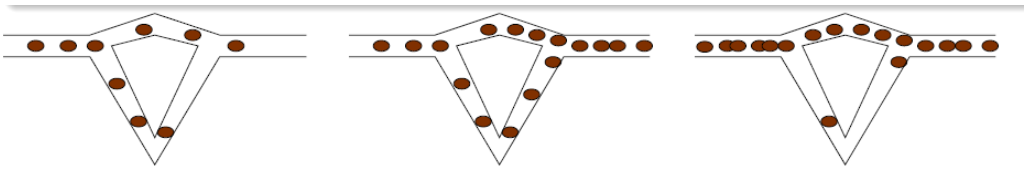
- Données d'apprentissage
  - $x = 3 \rightarrow y = 5$; hypothèse $a = 0,\ b = 5; a = 2,\ b = -1; \ldots.$
  - ….
- Solutions approchées (ne pas apprendre « par cœur » la base d'apprentissage)
  - Évaluation des meilleures solutions
    - Distance entre résultat fourni et résultat attendu pour un ensemble de données
    - Minimiser cette distance ➔ processus d'apprentissage

Colonies de Fourmis (1)

- **Idée :**
  - Auto-organisation des insectes sociaux
  - **Emergence** d'un comportement global à partir **d'interactions** locales
    - Emergence : comportement global non programmé
    - Comportement : structure
    - Interactions : communications directes ou indirectes
  - Systèmes dynamiques
  - Robustesse et flexibilité
  - Méthode ACO Dorigo 1992

Colonies de Fourmis (2)

- **Recherche de nourriture par une colonie de fourmis**
  - Emergence de « plus courts chemins »



    - Initialement : tous les chemins sont équiprobables
    - Les fourmis prenant le plus court chemin reviennent le plus vite
      - Chemin le plus court a plus grande fréquence de passage
        - Accroissement de la concentration en phéromones sur ce chemin
        - Evaporation sur les autres chemins
  - Méthode ACO Dorigo 1992

ACO (1)

- **Principe d'auto-organisation**
  - Ensemble d'agents (fourmis) communicant indirectement par des dépôts de phéromones
    - Remarque : ici les communications directes entre agents ne sont pas considérées
  - Les phéromones sont déposées par les agents au cours de leurs déplacements
  - La quantité déposée est contrôlée par chaque agent
  - Les phéromones déposées attirent les autres agents
  - Evaporation au cours du temps

ACO (1)

- **Algorithme général**

```
Initialisation des traces de phéromones
Répéter
      Chaque fourmi calcule un chemin : aléatoire / phéromones, …
      Mise à jour des traces de phéromones
Jusqu'à : Condition d'arrêt
```

  - Ajout de recherche locale pour améliorer les solutions
  - Définir un graphe pour représenter la construction de solutions
    - Sommets : composants de solutions
    - Arcs : succession de composants
    - Solution : meilleur chemin dans le graphe

Application ACO pour le TSP

- Un ensemble d'agents situés initialement sur le même sommet
- **Choix d'un nouvel arc**
  - Pour chaque agent (placé au sommet i) la robabilité de choisir le prochain arc (i, j) dépend de :
    - Concentration en phéromones sur l'arc (i, j) par rapport aux autres arcs
    - Mesure de la qualité de l'arc (inverse de sa longueur par exemple)
    - Pondérations entre ces 2 éléments
- **Mise à jour des traces**
  - Renforcement : ajouter des phéromones sur tous les arcs de la meilleure solution
  - Evaporation : diminuer la quantité de phéromones sur chaque arc (de manière proportionnelle à sa qualité)
    - Borner la quantité minimale et maximale de phéromones sur chaque arc

Composants ACO

- **Recherche Gloutonne**
  - Construction de solutions par une méthode gloutonne
- **Recherche Aléatoire**
  -
- **Recherche adaptative**
  - Adapter l'exploration en fonction d'un historique
    - Capitalisation de l'expérience par les dépôts de phéromones
    - Exploitation en biaisant la recherche gloutonne par rapport aux phéromones
- **Intensification / Diversification**
  - Intensifier : dépôt de phéromones sur les zones prometteuses – Sélection en fonction des quantités de phéromones
  - Diversifier : évaporation des phéromones

Essaims particulaires (1)

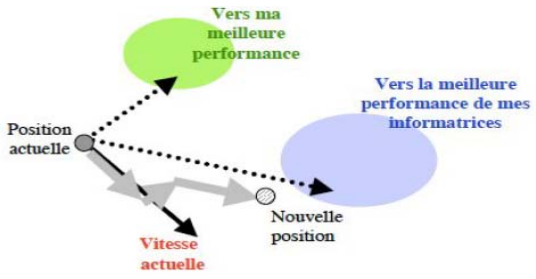
Particle Swarm Optimization

- **Idée :**
  - Population de solutions : essaim
  - Compromis entre trajectoire individuel et trajectoire du groupe

  - Un individu :
    - Suit sa propre trajectoire
    - Subit l'influence des autres
    - Mémorise sa meilleure performance
  - Exploration de l'espace des solutions
  - Origine : optimisation continue

Essaims particulaires (2)

- **Chaque individu est un élément de l'espace de recherche**
  - Position dans l'espace de recherche
  - Vitesse : dépend des voisins, des bonnes solutions visitées
  - Voisinage : ensemble d'individus auxquels il est relié
- **Déplacer chaque individu en fonction** :
  - Comportement « individuel » : suivre sa trajectoire
  - Comportement « conservateur » : revenir vers la meilleure position déjà visitée
  - Comportement « collectif » : suivre le meilleur voisin



Section 4. Pour aller plus loin ....

- Méthodes hybrides
- Méthodes multi-objectif

Contraintes et Objectifs

- **Contraintes vs Objectif**
  - Introduire des « contraintes » dans la fonction Objectif
  - Objectif :
    - nb de contraintes non satisfaites (exemple capacité à respecter)
    - on cherche à minimiser cet objectif, lorsqu'il vaut 0 :
      - obtention d'une solution admissible
  - **Intérêt :**
    - Explorer des affectations à la limite entre cohérentes et incohérentes
    - Traiter des pbs de satisfaction sous forme de pbs d'optimisation
  - **Difficulté :**
    - Combiner cet objectif avec celui (ceux) déjà existant(s)
    - Combinaison linéaire ➔ 1 seul objectif

Hybridation de méthodes

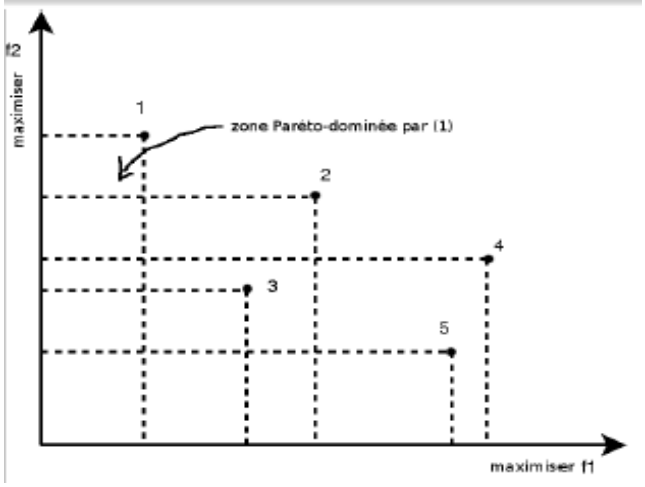
- **Hybridation de méthodes**
  - Algo. Mémétique
    - Combiner algorithmes évolutionnaires et recherche locale
      - Population initiale
      - Mutation ➔ recherche locale
  - Explorer plusieurs solutions :
    - Recherche locale + Mutation
  - PLNE / PPC + méthodes approchées :
    - Beam Search : Branch and Bound & heuristique
    - Exploration de grands voisinages avec PPC / PLNE

Optimisation multi-objectif (1)

- **Pourquoi ?**
  - Prendre en compte simultanément plusieurs objectifs contradictoires
    - Exemple : temps / argent
  - Comment comparer des solutions entre elles ?
  - Il existe un ensemble de bonnes solutions : les **solutions de compromis**
    - Rechercher un équilibre tel que :
      - on ne peut pas améliorer un objectif sans détériorer au moins un des autres objectifs
    - **Front de Pareto**

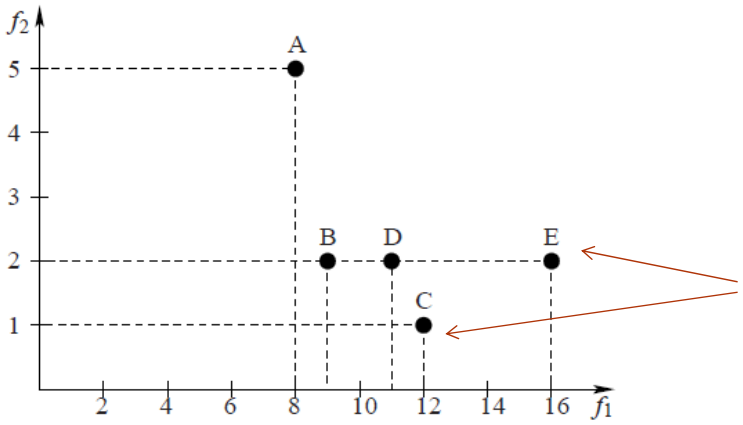
Optimisation multi-objectif (1)

- **Solution de Pareto :**
  - Une solution x est Pareto-dominante / une solution y si :
    - x est supérieure ou égale à y sur tous les objectifs
    - x est strictement meilleure que y sur au moins un des objectifs
    - Si x Pareto-domine y : on peut éliminer la solution y



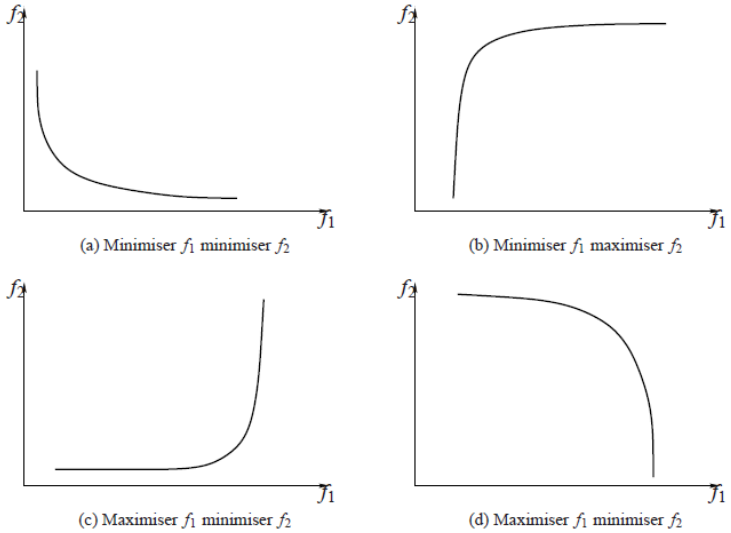
Illustration

- **Exemple : recherche des solutions non dominées**
  - Max f_1 et Min f_2



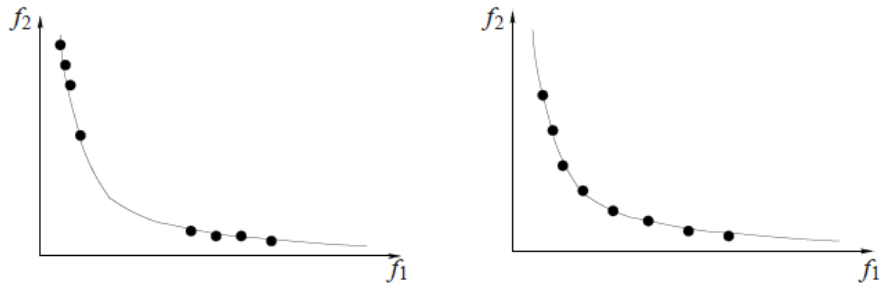
Formes de Front de Pareto

- **Forme des Fronts de Pareto à 2 objectifs**



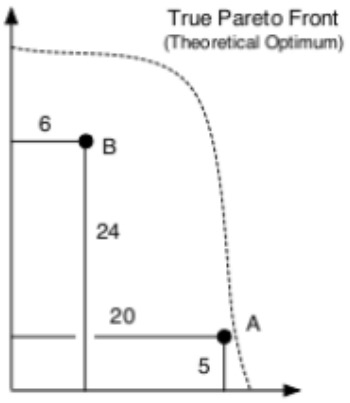
Calcul d'un Front de Pareto (1)

- **Comment obtenir un Front de Pareto**
  - Points extrêmes :
    - Optimiser 1 seul objectif
  - Bonne représentation du Front : solutions diversifiées



Calcul d'un Front de Pareto (2)

- **Réduction à un seul objectif**
  - Combinaison linéaire des objectifs
  - Exemple :
    - $f(x) = a \times f_1(x) - b \times f_2(x) + c \times f_3(x)$
  - Toute méthode mono-objectif
  - Difficulté : déterminer les coefficients
  - Approximation du front

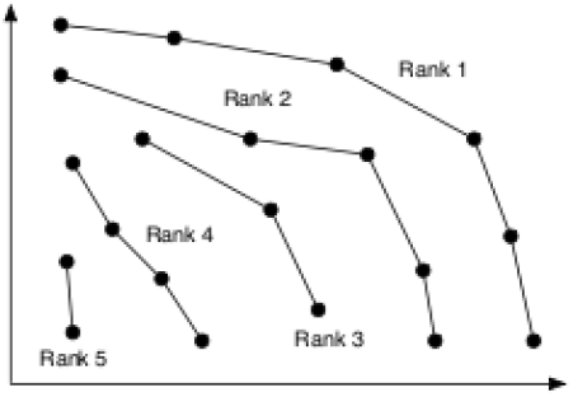


Calcul d'un Front de Pareto (3)

- **Réduction à un seul objectif**
  - Méthode évolutionnaire
    - Modifier le processus de sélection pour prendre en compte les différents objectifs
    - Utiliser la sélection par tournoi
      - avec ordre lexicographique sur les objectif
        - Exemple : $f_1 > f_2 > f_3$
      - Avec objectif choisi aléatoirement
      - Avec comparaison majoritaire sur les fonctions objectifs
      - Utiliser un rang de Pareto
        - Rang 1 : solutions non dominées
        - Rang 2 : solutions dominées uniquement par solutions de rang 1
        - Etc

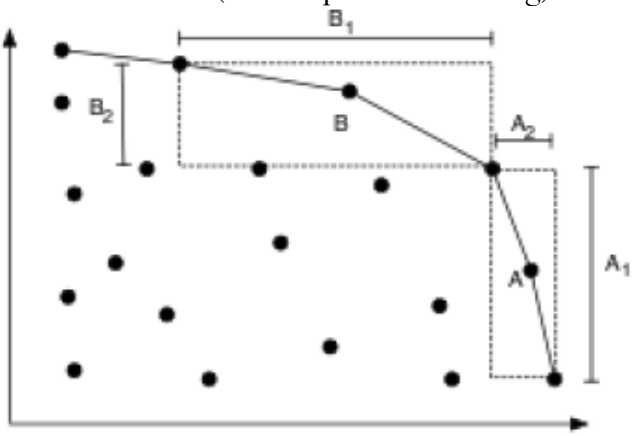
Calcul d'un Front de Pareto (4)

- **Considérer explicitement les objectifs**
  - Utiliser un rang de Pareto
    - Rang 1 : solutions non dominées
    - Rang 2 : solutions dominées uniquement par solutions de rang 1
    - Etc
  - Ignorer les solutions dominées pour la génération suivante



Calcul d'un Front de Pareto (5)

- **Diversité des solutions**
  - Bon échantillonnage du Front de Pareto
  - Introduire une mesure d'espacement entre solution (en complément du rang)



- **Algorithme NSGAII**
  - Algorithme évolutionnaire
  - Rang + espacement
    - Pour 2 solutions de même rang, privilégier celle ayant le plus grand espacement
  - Stratégie de remplacement + archivage meilleures solutions

Section 5. Conclusion

Conclusion (1)

- **Analyse du problème**
  - Quelles sont les décisions à prendre ?
  - Quelles sont les contraintes ?
  - Quelles sont les objectifs ?
    - Unique ?
    - Multiple ?
- **Modélisation**
  - Caractéristiques du modèle obtenu (taille, ….)
  - Lien avec des problèmes classiques
  - Complexité

Conclusion (2)

- **Résolution**
  - Concevoir une méthode spécifique ?
  - Utiliser des outils de résolution ?
- **Décomposer**
  - Parties essentielles du problème
  - Intégrer au fur et à mesure
- **Evaluation**
  - Instances « jouet » / Instances taille réelle
  - Analyse critique des résultats

Conclusion (3)

- **Plateformes pour méthodes approchées**
  - Metaheuristic optimization frameworks: a survey and benchmarking (2011 !!!)

Name	Ver.	Web
EasyLocal (Di Gaspero and Schaerf 2003)	2.0	http://satt.diegm.uniud.it/EasyLocal++/
ECJ (Luke et al. 2009)	20	http://cs.gmu.edu/~eclab/projects/ecj/
EO/ ParadisEO/ MOEO/ PEO (Cahon et al. 2004)	1.2	http://paradiseo.gforge.inria.fr http://eodev.sourceforge.net/
EvA2 (Kronfeld et al. 2010)	2	http://www.ra.cs.uni-tuebingen.de/software/EvA2/
FOM (Parejo et al. 2003)	0.8	http://www.isa.us.es/fom
HeuristicLab (Wagner 2009)	3.3	http://dev.heuristiclab.com
JCLEC (and KEEL) (Ventura et al. 2008)	4.0	http://JCLEC.sourceforge.net http://sci2s.ugr.es/keel/
MALLBA (Alba et al. 2007)	2.0	http://neo.lcc.uma.es/mallba/easy-mallba/index.html
Optimization Algorithm Toolkit (Brownlee 2007)	1.4	http://optalgtoolkit.sourceforge.net
Opt4j (Martin Lukasiewycz and Helwig 2009)	2.1	http://opt4j.sourceforge.net

  - dédiées algo évolutionnaires : JGAP, (Java), GAlib (C++), ….
  - dédiées recherche locale : JAMES (Java), LocalSolver (payant), …

Diversité / Similarités des méthodes (1)

- **Deux grandes familles d'approche :**
  - Méthode de Recherche locale (à solution unique) :
    - Evolutions successives d'une solution
  - Méthode à population de solutions :
    - Evolution de la population : combinaison/mutation
    - Processus de sélection
- **Des concepts similaires :**
  - Résultat : solution approchée
  - Sortir des optima locaux
    - Compromis entre Diversification et Intensification

Diversité / Similarités des méthodes (2)

- **Intensification**
  - Pousser la recherche autour de solutions de bonne qualité
- **Diversification**
  - Déplacer la recherche vers de nouvelles parties non explorées de l'espace des solutions
- **Compromis entre les deux :**
  - Spécifique de chaque méthode approchée
- **De très (trop ?) nombreuses méthodes**

Comment choisir ?

- **Quelle méthode choisir pour résoudre un problème**
  - Produire une bonne solution
  - Temps de calcul raisonnable
  - Mais pas de recette miracle
    - Tester plusieurs méthodes, bibliographie, expertise, connaissance métier, …
  - No Free Lunch Theorem (for Search / Optimization / Learning)
    - Il n'y a pas de méthodes meilleure que les autres sur l'ensemble des problèmes
    - " any two optimization algorithms are equivalent when their performance is averaged across all possible problems"