# Méthodes d'optimisation avancées

E-G. Talbi

Professeur - EUDIL - GIS

Chercheur – LIFL – CNRS

talbi@lifl.fr

# Plan du cours

- Introduction
- Recherche locale
- Recherche locale avancée (recuit simulé, recherche tabou, …)
- Algorithmes évolutionnaires (Algorithmes génétiques, …)
- Algorithmes coopératifs : colonies de fourmis, …
- Optimisation multi-critères
- Théorie des jeux
- …

# Difficulté des problèmes réels

- Nombre important de solutions possibles dans l'espace de recherche → Recherche exhaustive impossible
- Problème complexe → Utilisation de modéles simplifiés
- Fonction d'évaluation qui décrit la qualité des solutions "noisy" et dynamique (fonction du temps) Plusieurs solutions à trouver →
- Problèmes avec des contraintes fortes Construire une solution réalisable est difficile. →
- Multi-critère, ...

# Taille de l'espace de recherche

- Problème SAT : Boolean Satisfiability Problem (First NP-hard problem).
- Plusieurs applications : emploi du temps, routage, …
- Trouver un ensemble de variables booléennes

$$X = (X_1, ..., X_n)$$

pour qu'une expression booléenne donnée soit = TRUE

- L'expression booléenne doit être une conjonction de clauses ; où les clauses sont représentées par des disjonctions de k variables (k-SAT) :

$$F(X) = (\overline{X}_1 \vee X_3) \wedge (X_1 \vee \overline{X}_2) \wedge (X_4 \vee X_2) \wedge ...$$

- Pour n=100, taille espace = $2^{100} \approx 10^{30}$,
- Evaluation de 1000 chaînes / sec ; 15 milliards d'années (Big Bang) pour évaluer moins de 1% de l'espace.
- k>2, problème NP-difficile ; k=2, Algorithme polynomial

# Taille de l'espace de recherche

- Problème TSP : n villes, trouver un circuit qui minimise la distance totale parcourue ; toutes les villes doivent être visitées une seule fois.
- TSP symétrique  dist(x,y)=dist(y,x)
- |S| = n!/2n = (n-1)! / 2 ;  n>6  TSP > SAT
- 10 villes = 181 000 solutions
- 20 villes = 10 000 000 000 000 000 solutions
- 50 villes = 100 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 solutions
- Il y a 1 000 000 000 000  000 000 000 litres d'eau dans la planète http://www.ing.unlp.edu.ar/cetad/mos/TSPBIB_home.html

# Taille de l'espace de recherche

- Problème NLP : Non-Linear Programming Problem
- Un exemple difficile étudié dans la littérature scientifique
- Plusieurs applications : mécanique, aéronautique, …
- Méthodes classiques ne donnent pas de résultats satisfaisants

$$G2(x) = \left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|$$

Fonction non linéaire
Optimum global inconnu

$$\prod_{i=1}^{n} x_i \geq 0.75 \qquad \sum_{i=1}^{n} x_i \leq 7.5n \qquad 0 \leq x_i \leq 10$$

Contrainte non-linéaire + contrainte linéaire

# Taille de l'espace de recherche

- Traité comme un problème mathématique = infinité de solutions / dimension.
- Sur machine : supposons une précision de six décimaux, chaque variable peut prendre 10 000 000 valeurs différentes.
- $|S| = 10\,000\,000^n = 10^{7n}$
- Espace plus grand que celui du TSP
- Fonction d'évaluation et contraintes ?
- Maximiser G2(x) ; Solutions non réalisables = 0

# Modélisation du problème

- Dans la réalité, nous trouvons une solution à un modèle du problème.
- Tous les modèles sont des simplifications de la réalité.
- Problème ==> Modèle ==> Solution
- SAT, TSP, et NLP sont 3 formes canoniques de modèles qui peuvent être appliqués à différents problèmes.
- Exemple : Société de production de voitures de *n* couleurs différentes. Trouver un ordonnancement optimal qui minimise le coût total de la peinture des voitures.

# Modélisation du problème

- Toute machine utilisée doit être « switchée » d ’une couleur à une autre ; le coût d’un tel changement dépend des deux couleurs.
- Jaune à Noir = 30, Noir à Blanc = 80
- Trouver une séquence de tâche de peinture des

rouver une séquence de tâche de peinture des

e tâche de peinture des différentes voitures

asymétrique ; nœud = tâche peinture couleur

ique ; nœud = tâche peinture couleur donnée.

- **Modélisation simplifiée :**
  **Complexité, Changement dans le temps**

- Importance du modèle

# Contraintes

- La plupart des problèmes réels possèdent des contraintes.
- Ex : Problème d'emploi du temps :
  - Liste des cours, classes, étudiants / classe, professeur / classe, salles disponibles, taille des salles, logistique des salles (vidéo, …).
  - Contraintes fortes (hard) :
    - Toute classe doit être affectée à une salle disponible avec un nombre de places suffisant et des facilités requises,
    - Les étudiants affectés à plusieurs classes ne doivent pas être affectés au même temps.
    - Les professeurs ne doivent pas être affecté à des cours en parallèle.

# Exemple : Réseaux de téléphonie mobile

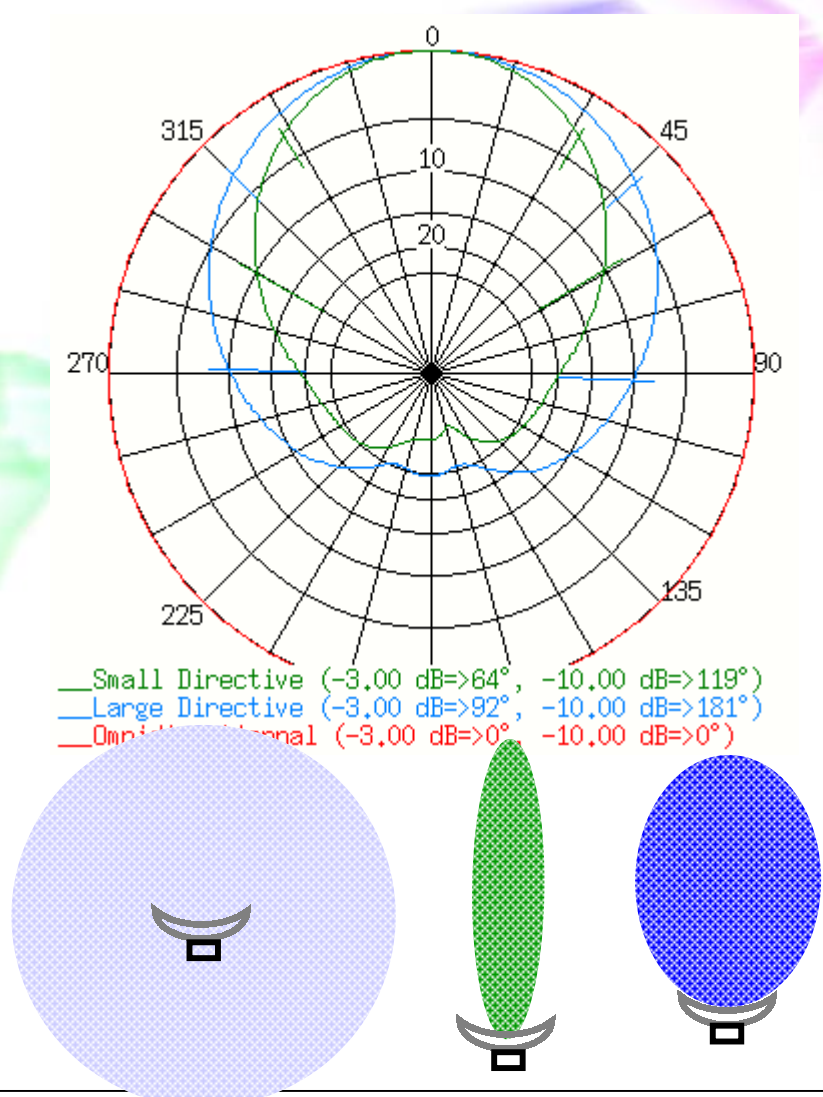
- Enjeux économiques
  - Coût du réseau : nombre de sites, …
  - Qualité de service : trafic, interférences, …
- Design de réseau
  - Positionnement de sites
  - Paramétrage de BTS (dispositifs de transmission, antennes, …)

Un site :
De 1 à 3 BTS par site

BTS : Base Transceiver Station

# Paramètres des antennes

| Données | bornes |
|---|---|
| Puissance | [26, 55] dBm |
| Diagramme | 3 types |
| Hauteur | [30, 50] m |
| Azimut | [0, 359] ° |
| Inclinaison | [-15, 0] ° |
| Transmetteurs | [1, 7] |



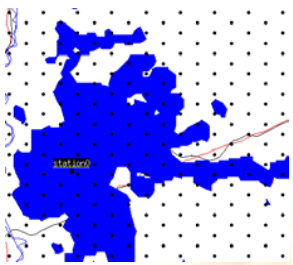

La **combinatoire** est **importante** :

~ $600.10^9$ choix par antenne

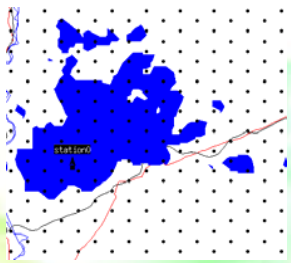
1 omni ou 1-3 sectorielles

# Effets du paramétrage

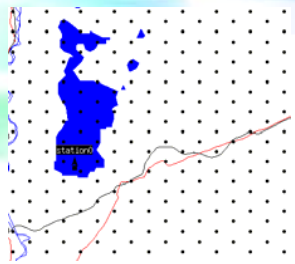
- Cellule = zone couverte par une BTS
- Trafic limité pour un site (contrainte technologique)
- Plusieurs sites sont nécessaires



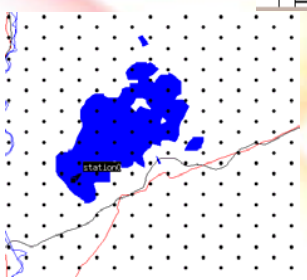
Omni-directionnelle
60 dBm



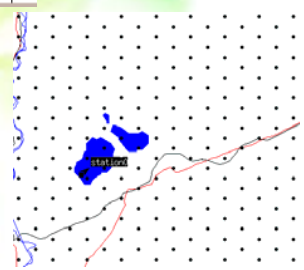
Sectorielle type 1



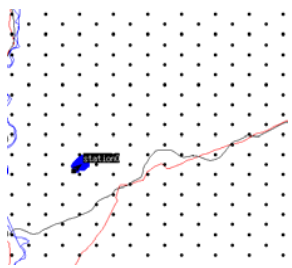
Sectorielle type 2

Azimut 50°



PIRE -10 dBm

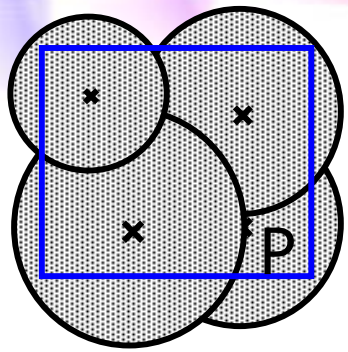


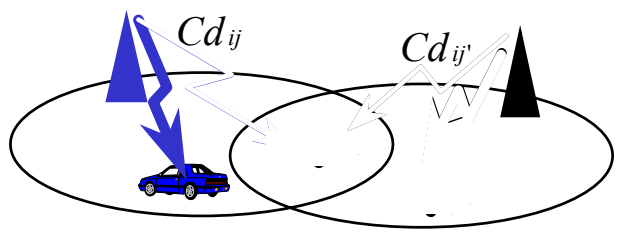
Inclinaison -15°

**Modèle de propagation utilisé : Espace libre**

# Modèle : Contraintes

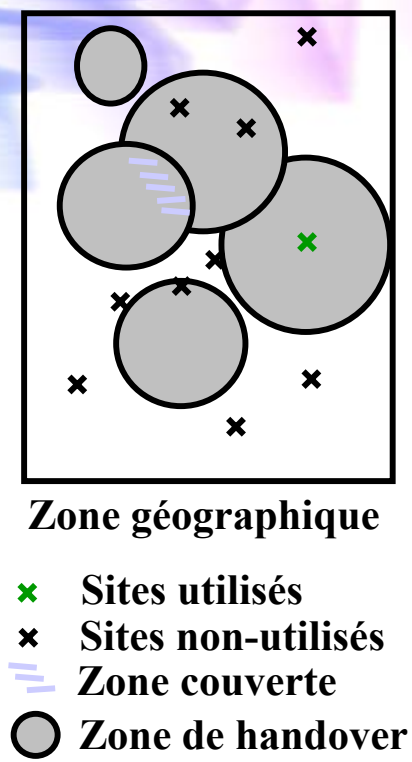
- Un ensemble de BTS qui satisfait les contraintes de :
  - Couverture
  - Handover (mobilité)



# Modèle : Objectifs

- Min Nombre de sites (coût)
- Min Interférences
- Max Trafic écoulé



# Ex : Le problème du sac à dos (Knapsack Problem)

- Un randonneur doit décider de l'équipement qu'il veut emporter pour sa prochaine expédition. Pour des raisons évidentes, il veut limiter le poids total des objets emportés et s'est fixé une borne supérieure de *b* kg à ne pas dépasser.

- Chacun des n objets qu'il peut emporter possède un poids $a_i$, $i=1,\ldots,n$ ainsi qu'une utilité $c_i$, $i=1,\ldots,n$, cette dernière étant d'autant plus grande que la randonneur juge l'objet intéressant.

# Ex : Le problème du sac à dos (Knapsack Problem)

- Définissant pour chaque objet une variable de décision binaire
- $x_i$ = 1 si l'objet i est retenu
- $x_i$ = 0 sinon
- Déterminer une sélection optimale revient à résoudre le problème d'optimisation suivant (programme linéaire en nombre entiers) :

$$MaxZ = \sum_{i=1}^{n} c_i x_i$$

$$\sum_{i=1}^{n} a_i x_i \leq b$$

# Exercice

- Mr Dupont et sa femme ont invité 4 couples pour une soirée. Certains invités présentes ont salué d'autres invités. Personne n'a salué sa femme, et personne n'a salué une même personne deux fois.

- Mr Dupont a questionné toutes les personnes présentes : combien de personnes avez vous salué ?

- Toutes les réponses reçues sont différentes.

- Combien de personnes Mme Dupont  a salué ?

# Concepts de base

Indépendamment des algorithmes et techniques utilisés, on a besoin de spécifier :

- Représentation : code les solutions candidates
- Objectif : but à atteindre
- Fonction d'évaluation : retourne une valeur spécifique qui indique la qualité d'une solution

- Définition de l'espace de recherche
- Voisinage et optima locaux

# Représentation

Representation ==> Espace de recherche + Taille

- Problème SAT avec n variables : chaîne binaire de longueur n, elément = variable
- Taille de l'espace de recherche = $2^n$.
- Toute solution de l'espace de recherche est réalisable
- Problème TSP avec n villes : permutation des nombres entiers 1, …, n. elément = ville.
- Taille de l 'espace de recherche = n!
- TSP symétrique : n! / 2
- Ville de départ : (n-1)! / 2

# Représentation

- Problème NLP : tous les nombres réels dans les n dimensions.
- Représentation flottante pour approximer les nombres réels : 6 digits de précision, $10^{7n}$ différentes solutions possibles.
- Choix de la représentation est important.

# Objectif et fonction d'évaluation

- Objectif : formulation mathématique du but à atteindre.
- TSP : minimiser la distance totale $\min \sum dist(x, y)$
- SAT : formule booléenne satisfaite (TRUE).
- NLP : minimiser la fonction G2(x).
- Fonction d 'évaluation : fonction de l'espace de recherche vers l'ensemble des réels R (ordinal, numérique, …), qui représente la qualité de toutes les solutions.
- TSP : distance totale, évaluation coûteuse.
- Approximer la qualité de la fonction, comparer deux solutions (fonction opérant sur deux solutions).
- NLP : fonction G2(x).

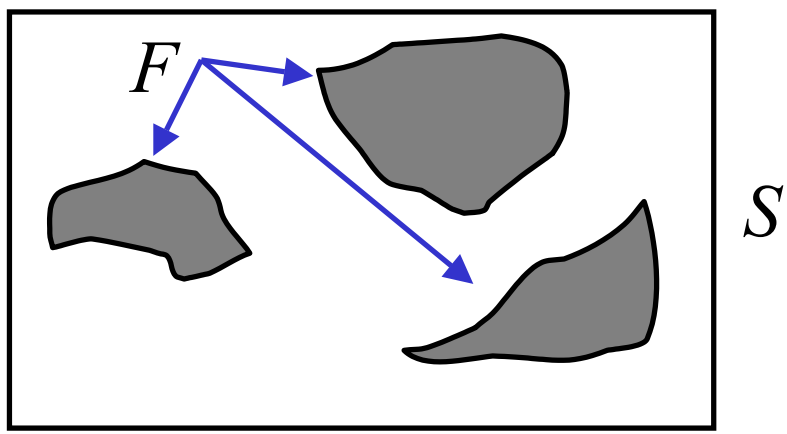
# Objectif et fonction d'évaluation

- SAT : toute solution non optimale = FALSE ==> aucune information sur la qualité des solutions, aucune indication sur l'amélioration des solutions, recherche de bonnes solutions, ...

- SAT et TSP : $F = S$
- NLP : $F \subseteq S$

$S$ : Espace de recherche
$F$ : Solutions réalisables (contraintes satisfaites)

$$F \subseteq S$$

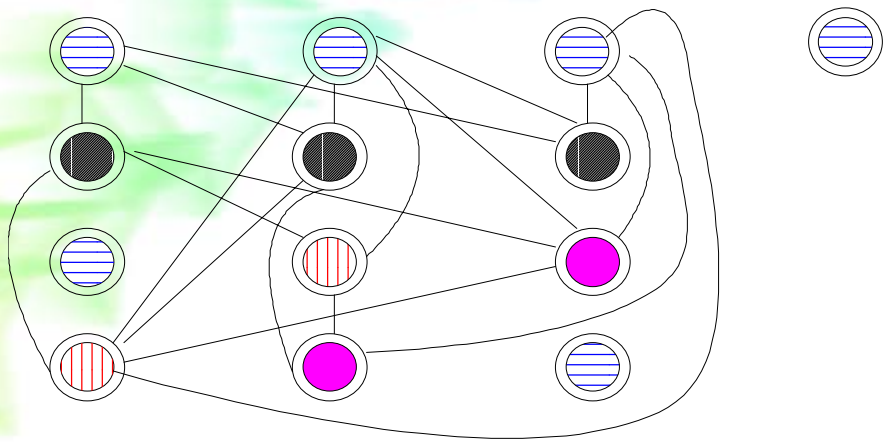


# Coloration de graphes

$G = (S,A)$
coloration $f : S \rightarrow C$ telle que $(s,t) \in A \Rightarrow f(s) \neq f(t)$

$\mathrm{Chr}(G) = \min_{f} \mathrm{card}\, f(S)$, nombre chromatique de $G$

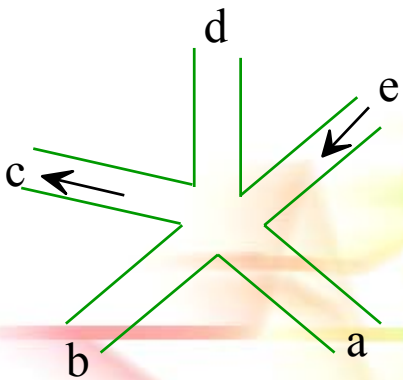
Chr($G$) = 4



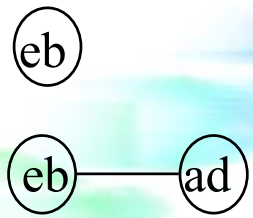
Plusieurs applications : affectation de fréquences (télécommunication), ordonnacement (productique), emploi du temps, allocation de registres, …

# Problème des feux

Coloration de graphe pour la modélisation
d ’un problème

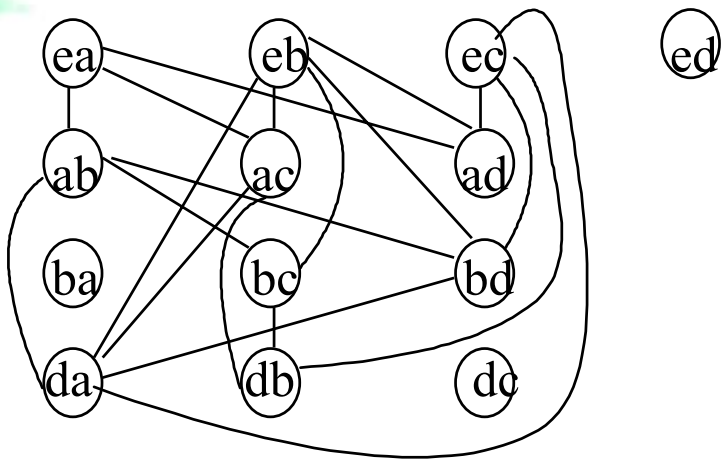



sens unique




traversée possible

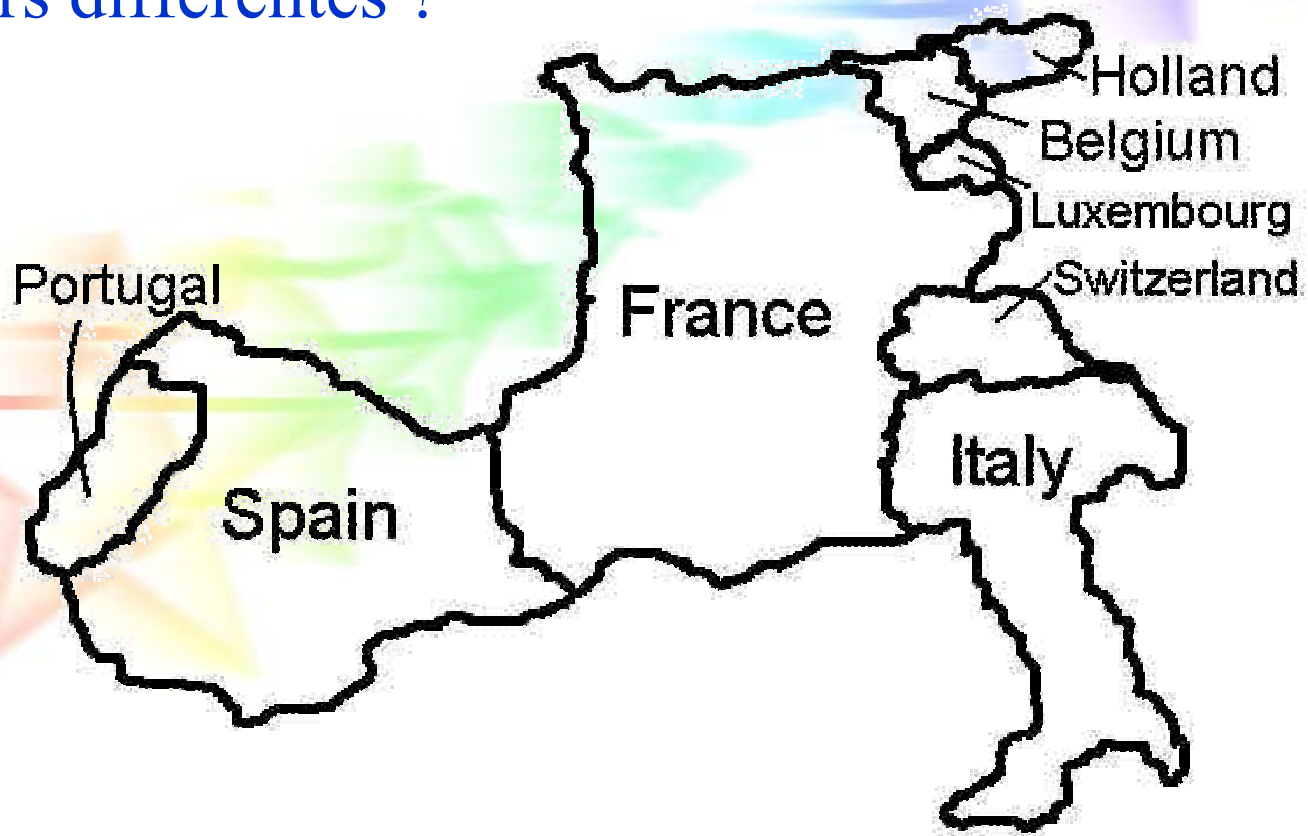
traversées incompatibles



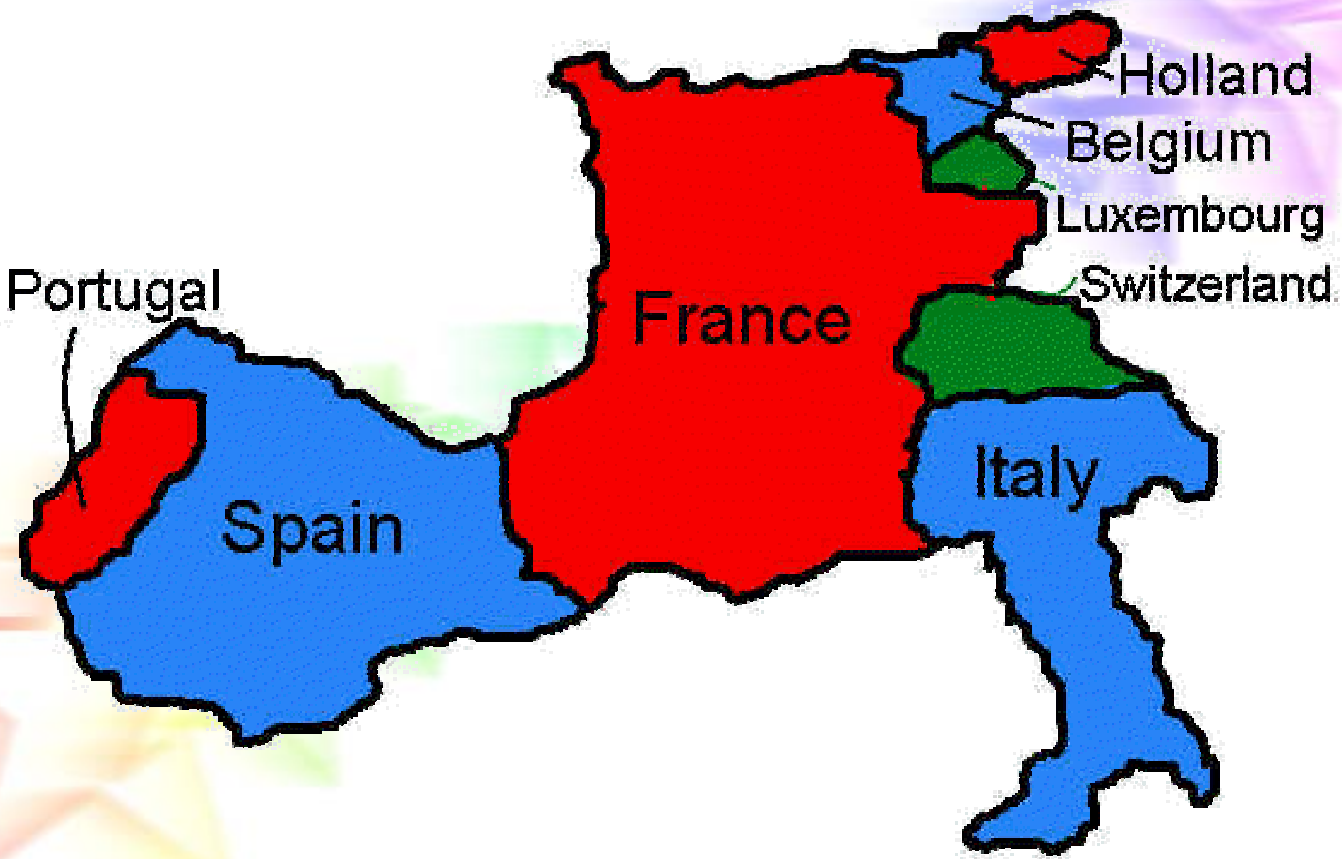

couleur = ensemble de traversées compatibles

# Coloration de carte (map)

nombre minimum de couleurs nécessaires pour colorier une carte / régions frontalières ont des couleurs différentes ?



# Coloration de carte (map)



# Problème d'optimisation

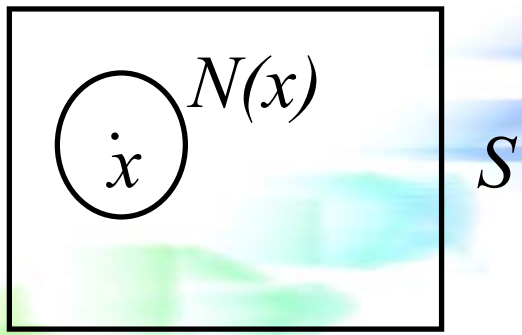
- Définition (problème de minimisation) :
  - Etant donné un espace de recherche *S* avec sa partie réalisable *F*, $F \subseteq S$
  - Trouver $x \in F$ tel que :

$$eval(x) \leq eval(y) \qquad \forall y \in F$$

  - x : optimum global
  - Trouver x pour TSP, SAT et NLP est difficile
  - Problèmes d'optimisation de grande dimension et complexes dans de nombreux secteurs de l'industrie (Télécommunications, Biologie, Génomique, Transport, Environnement, Mécanique, ...).
  - Problèmes de taille sans cesse croissante (explosion combinatoire) et/ou Délais de plus en plus courts.

# Voisinage et optima locaux

- Voisinage d'une solution :




- On peut définir une fonction de distance dans l'espace S :

$$dist : S \times S \rightarrow R$$

- et définir le voisinage N(x) comme :

$$N(x) = \{y \in S : dist(x, y) \leq \varepsilon\}$$

- y est dans le $\varepsilon$-voisinage de x

# Voisinage et optima locaux

- NLP (variables continues) : choix naturel pour définir la distance
- Distance euclidienne

$$dist(x,y) = \sqrt{\sum_{i=1}^{n} \left(x_i - y_i\right)^2}$$

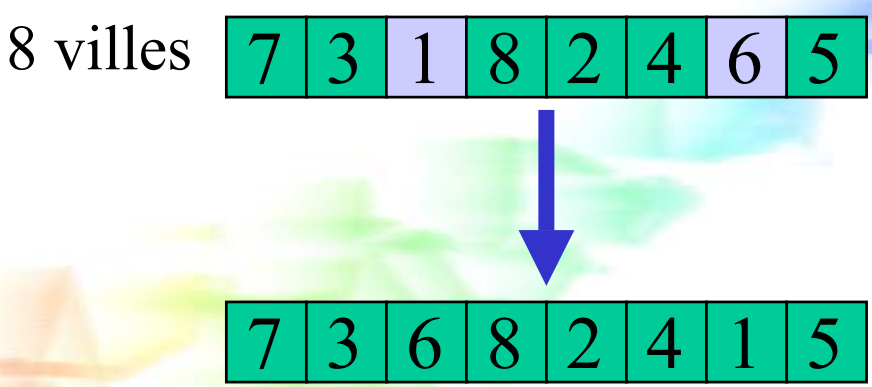
- SAT : distance de Hamming (nombre de bits avec des affectations différentes).
- On peut définir une fonction m de l'espace de recherche S :

$$m : S \rightarrow 2^S$$

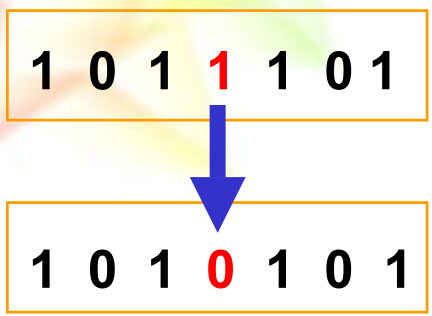
qui définit un voisinage pour chaque solution $x \in S$

# Voisinage et optima locaux

- TSP : opérateur 2-swap - Echange de 2 villes du tour.
- Une solution donné possède $\frac{n(n-1)}{2}$ solutions voisines.




- SAT : opérateur 1-flip - changer l'état d'un bit de la chaîne




Une solution donnée possède n voisins

x et y voisins <==> $dist(x, y) = 1$

# Voisinage et optima locaux

- Optimum local : Une solution $x \in F$ est un optimum local ssi :

  $eval(x) \leq eval(y) \qquad \forall y \in N(x)$

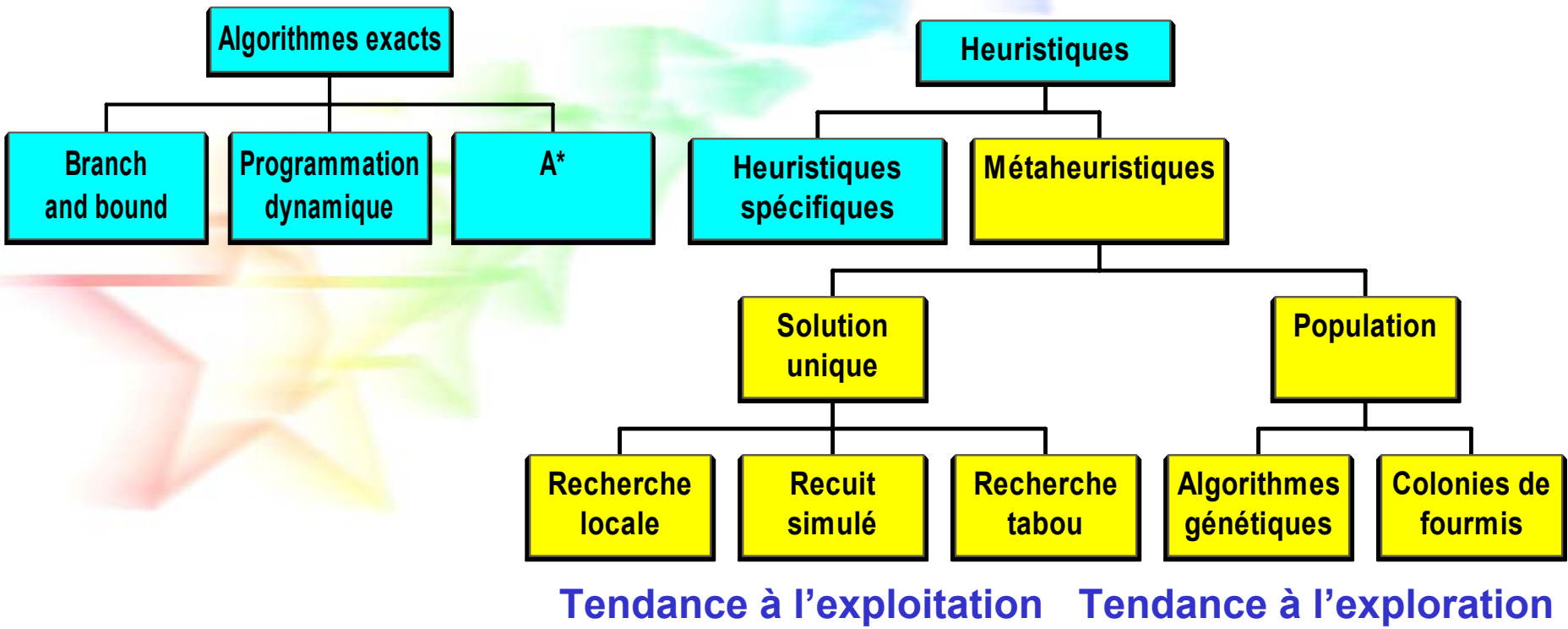
- L'optimalité locale est facile à tester si taille voisinage n'est pas importante, ou bien on connaît la dérivée de la fonction d'évaluation.
- Plusieurs méthodes d'optimisation sont basées sur des statistiques sur le voisinage de solutions (Méthodes de recherche locale).

- Exemple : max f(x) = $-x^2$, solution courante x=2,

$\varepsilon$=0.1 ; N(x)=[1.9,2.1] ; meilleure solution voisine ...

# Méthodes d'optimisation

- Changement de problème ==> Changement d'algorithme
- Algorithmes évaluant des solutions complètes - Algorithmes évaluant des solutions partielles (incomplète, problème réduit)
- Algorithmes exacts - Algorithmes approximatifs, heuristiques



# Méthodes d'optimisation

- Les méthodes exactes sont vite inutilisables (problème, instance).
- Les métaheuristiques sont efficaces (bornes inférieures, meilleurs résultats).
- Manque de résultats théoriques (applicables dans un contexte réaliste).

# Recherche exhaustive (énumérative)

- Evalue toute solution de *S* jusqu'à solution optimale globle trouvée.
- Plusieurs approches sont basées sur la recherche exhaustive (Branch and bound, A*, …).
- Question fondamentale : Comment générer une séquence de toutes les solutions de S ?

# Recherche exhaustive (SAT)

- Générer toutes les chaînes binaires de taille *n* ($2^n$).
- Générer les entiers de 0 à $2^n$-1 et convertir chaque entier en format binaire.

# Heuristiques

- Une heuristique est un algorithme de résolution ne fournissant pas nécessairement une solution optimale pour un problème d'optimisation donné.

- Une bonne heuristique possède cependant plusieurs caractéristiques :
  - Elle est de complexité raisonnable (idéalement polynomiale mais en tout cas efficace en pratique) ;
  - Elle fournit le plus souvent une solution proche de l'optimum ;
  - La probabilité d'obtenir une solution de mauvaise qualité est faible ;
  - Elle est simple à mettre en œuvre.

# Recherche locale (Hill-climbing)

Procédure Hill-climbing

local = FALSE ; Générer une solution initiale $v_c$ ;

Evaluer $v_c$ ;

Répéter

Générer le voisinage de $v_c$ ;

Soit $v_n$ la meilleure solution voisine ;

Si $v_n$ est meilleur que $v_c$ Alors

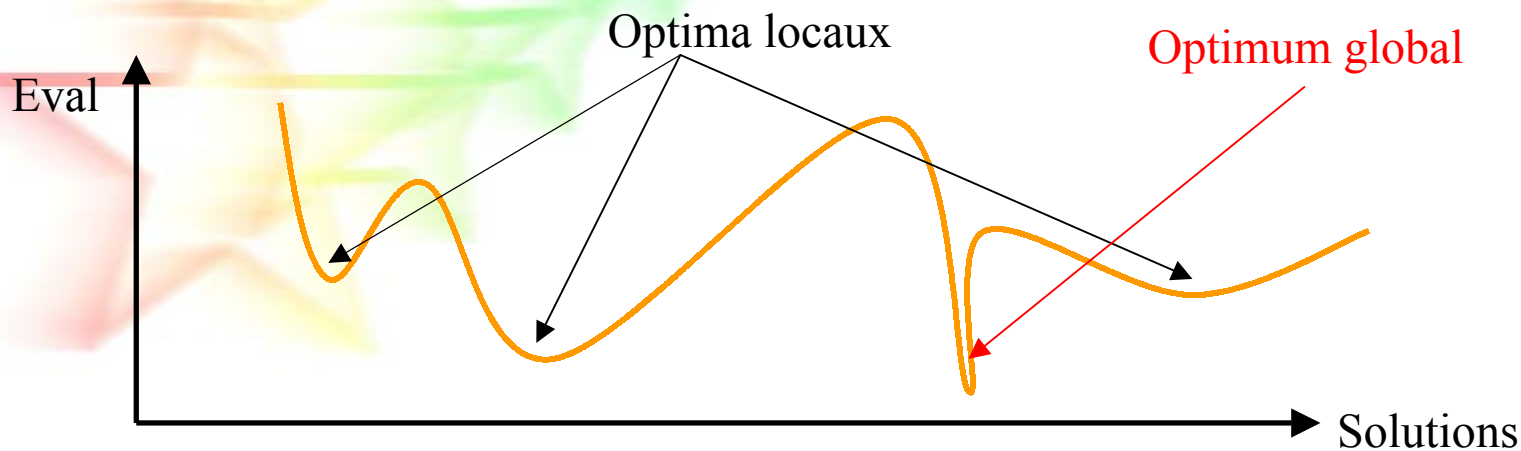
$v_c = v_n$ ;

Sinon local = TRUE ;

Jusqu'à local ;

# Recherche locale

- Facile à appliquer (représentation, fonction d'évaluation, voisinage).
- Converge uniquement vers des optima locaux
- L'optimum local trouvé dépend de la solution initiale (HC itéré)
- Pas de démarche générale pour borner l'erreur relative / optimum global
- Impossible d'avoir une borne supérieure sur le temps d'exécution



# Recherche locale

- Compromis Exploitation / Exploration
- Exploiter les meilleures solutions trouvées
- Explorer l'espace de recherche non visité
- Extrêmes :
  - Méthode Hill-climbing exploite la meilleure solution trouvée, mais néglige l'exploration d'une large portion de l'espace *S*.
  - Méthode de Recherche Aléatoire explore l'espace de façon efficace, mais mauvaise exploitation des régions prometteuses.
- Pas de meilleure méthode pour tous les problèmes
- Evaluation rapide des voisins (technique *delta)*

# Recherche locale (SAT)

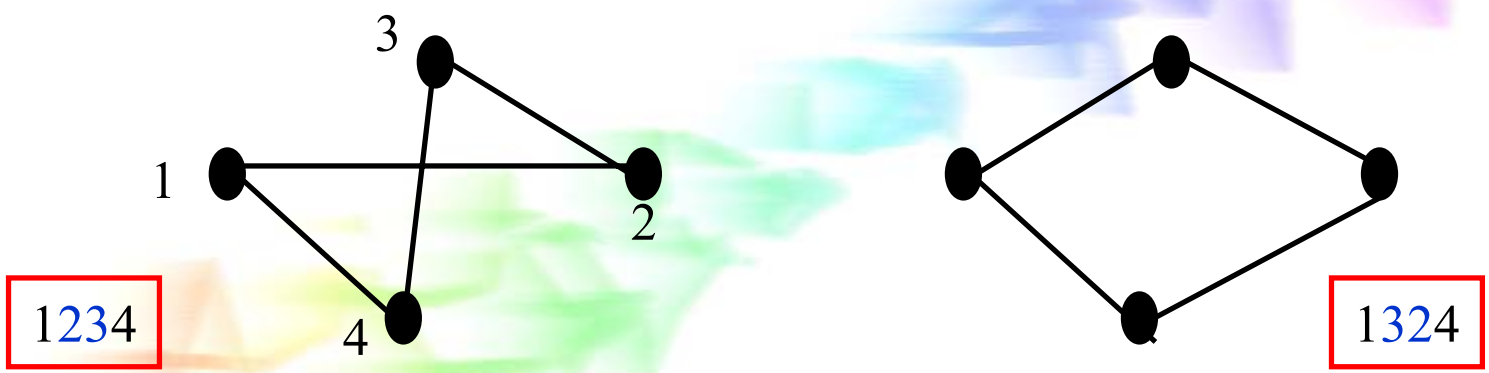
```
Procédure GSAT
    For i=1 To MAX-TRIES Faire
        T = affectation générée aléatoirement ;
        For j=1 To MAX-FLIPS Faire
            If T satisfait la formule Alors Return(T) ;
            Else Flip ;
    Return ('Pas d'affectation satisfaisante trouvée') ;
```

- Opérateur Flip : changer l 'état d'un bit.
- 2 paramètres : MAX-TRIES (nombre de recherches réalisées) et MAX-FLIPS (nombre d'itérations réalisées).
- Flip : Choisir la solution voisine / Minimise le nombre de clauses insatisfaites.

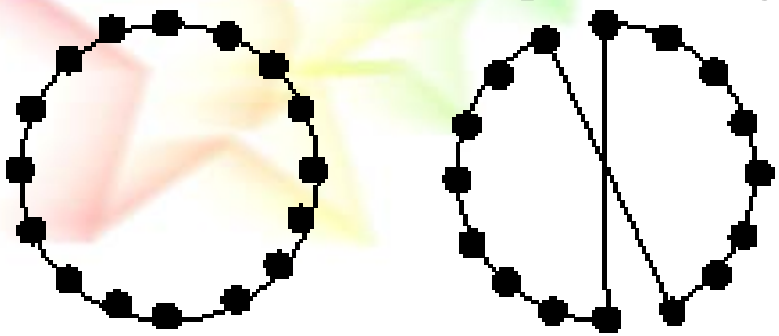
# Recherche locale (TSP)

- Opérateurs de recherche locale : city-swap ; 2-opt.

city-swap = Echange de villes




2-opt = Echange d'arcs non adjacents



Choisir 2 villes, les retirer, et transposer toutes les villes entre la ville de départ et la ville finale.

# Recherche locale (NLP)

- Optimizer f(X), $X = (x_1,...,x_n) \in R^n$ $\quad X \in F \subseteq S$

  $l(i) \le x_i \le u(i) \qquad 1 \le i \le n$

- *F* est défini par un ensemble de *m* contraintes :

$g_j(X) \le 0, j = 1,...q \qquad h_j(X) = 0, j = q+1,...m$

- Méthodes d'intervalles (Bracketing methods)
- Méthodes des points fixes
- Méthode du Gradient

# Exercice : Trouver les nombres

- Peter Ross (University of Edinburg)

- Trouver les nombres naturels a, b, c et d tel que :

ab=2(c+d), et

cd=2(a+b)

# Méthodes constructives (Greedy)

- Manipulent des solutions incomplètes, partielles.

- Affectation des variables de décision une par une
- A chaque itération, prendre la décision optimale pour la variable de décision courante
- Suppose une heuristique qui fournit le meilleur “profit” à chaque itération.
- Prendre la décision optimale à chaque étape n’implique pas une optimalité globale.

- Popularité (Simplicité)
- Complexité reduite

# Méthodes constructives (SAT)

- Heuristique : Pour chaque variable de 1 à n, dans un ordre quelconque, affecte la valeur qui permet de satisfaire le plus grand nombre de clauses non satisfaites (courantes).
- Exemple :

$$\overline{x_1} \wedge (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_1 \vee x_4)$$

- Considérons la variable $x_1$, $x_1$=TRUE (3 clauses)
- Reste de l'effort perdu … Clause 1 non satisfaite.
- Performances mauvaises.

- Heuristique pour le choix de l'ordre : Classer toutes les variables en se basant sur leurs fréquences, de la plus petite à la plus grande.

# Méthodes constructives (SAT)

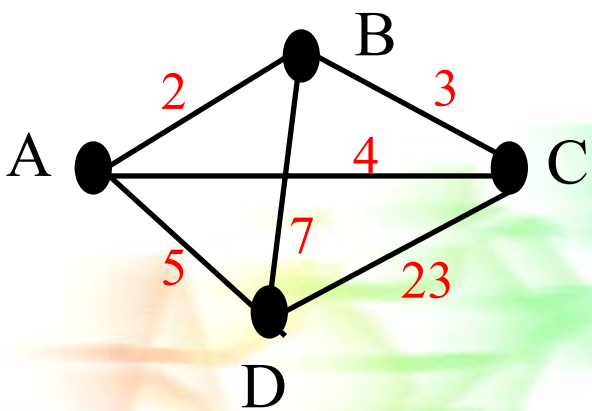
- Exemple :

$$(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (\overline{x_1} \vee x_4) \wedge (\overline{x_2} \vee \overline{x_4}) \wedge (x_2 \vee x_5) \wedge (x_2 \vee x_6) \wedge F$$

- F ne contient pas les variables $x_1$ et $x_2$, mais plusieurs occurrences des autres variables.
- Autres variables = fréquences plus élevées
- $x_1$=TRUE (2 clauses), $X_2$=TRUE (2 clauses)
- Impossible de satisfaire les clauses 3 et 4
- Autres améliorations : Interdire une affectation qui met une clause à FALSE, fréquences dans les clauses restantes, taille des clauses dans la prise en compte de l'ordre, ...

# Méthodes constructives (TSP)

- Heuristique : plus proche voisin - choix aléatoire de la ville de départ




A-B-C-D-A Dist=33

A-C-B-D-A Dist=19

- Autre heuristique : Choisir l'arête la plus courte restante, évitant la situation où une ville est adjacente à plus de 2 arêtes.

A-B B-C ~~C-A~~ A-B-C-D-A

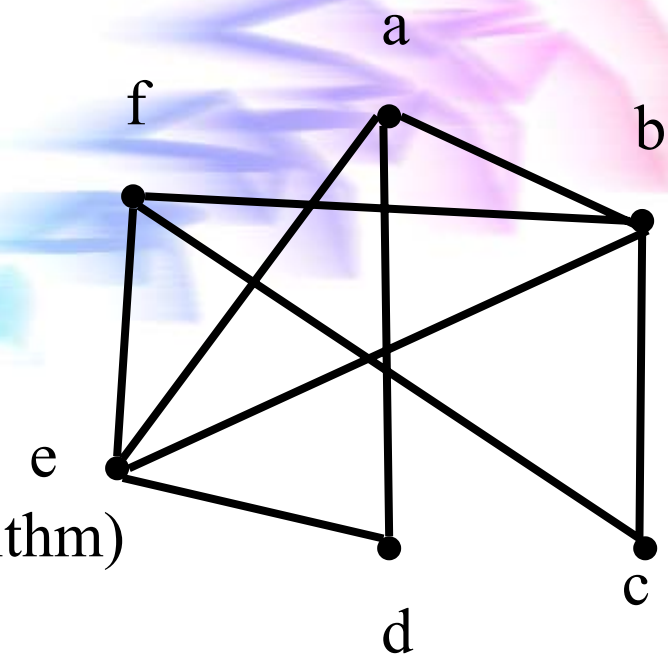
# Méthodes constructives (NLP)

- Pas d'algorithmes gloutons efficaces pour les NLP

- Exemple : fonction à deux variables $x_1$ et $x_2$
- Initialiser une variable (ex. $x_1$) à une constante, et varier $x_2$ jusqu'à trouver l'optimum.
- Une fois l'optimum trouvé ($x_2$=constante), varier $x_1$ jusqu'à trouver l'optimum.

- Efficace lorsque peu d'interactions entre les variables.

# Ex : Méthodes constructives (Coloration de graphes)




Color Largest Degree First (LF Algorithm)

Welsh and Powell [1967]

Tri par degré

Sommet : b, e, a, f, c, d

Couleur : 1 2 3 3 2 1

# Ex : Méthodes constructives (Coloration de graphes)

Merging Non-adjacent vertices

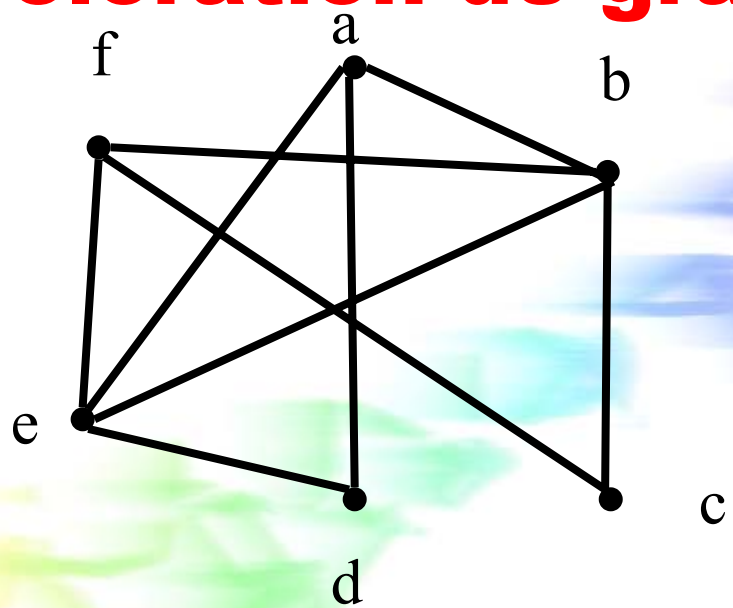
Dutton and Brigham [1981]

DB Algorithm

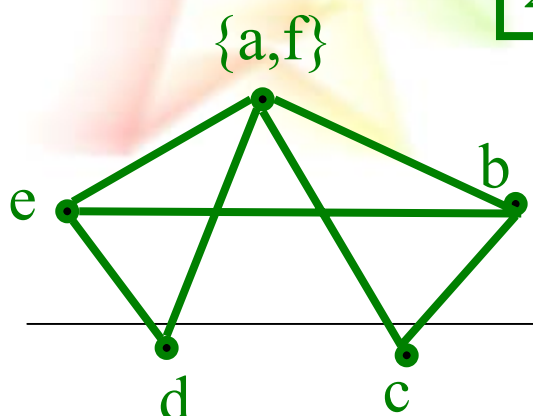
Grouper successivement les noeuds non adjacents,
Tantqu'il y a des noeuds non adjacents.
*i.e. le graphe résultant* est un graphe complet.

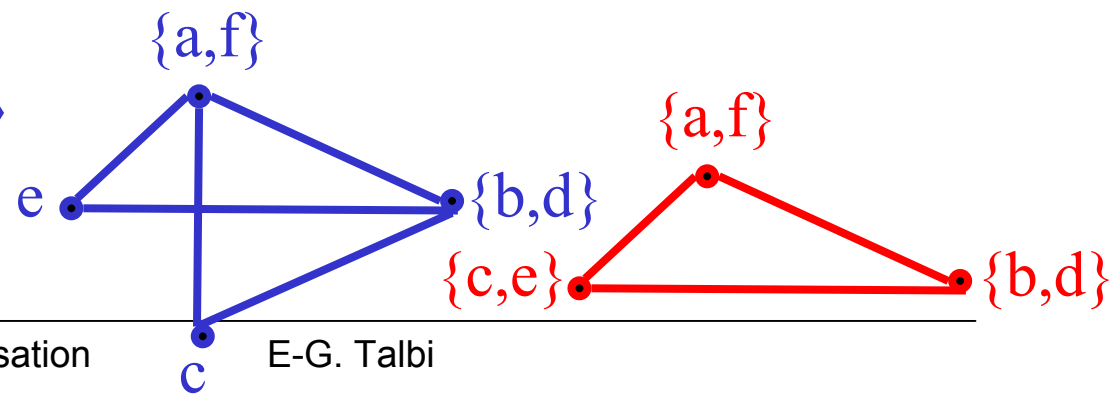
Heuristique : grouper en premier les noeuds non adjacents qui possèdent le maximum de noeuds adjacents communs

# Ex : Méthodes constructives (Coloration de graphes)




| (a, c) | (a, f) | (b, d) | (c, d) | (c, e) | (d, f) |
|---|---|---|---|---|---|
| 1 | 2 | 2 | 0 | 2 | 1 |
| | | 2 | 1 | 2 | |



# Diviser pour régner

Procédure D&C (P)

Partitionner le problème P en sous-problèmes $P_1$, …, $P_k$

For i=1 to k do

If Taille($P_i$) < p Then Résoudre ($P_i$, résultat $S_i$)

Else $S_i$ = D&C($P_i$)

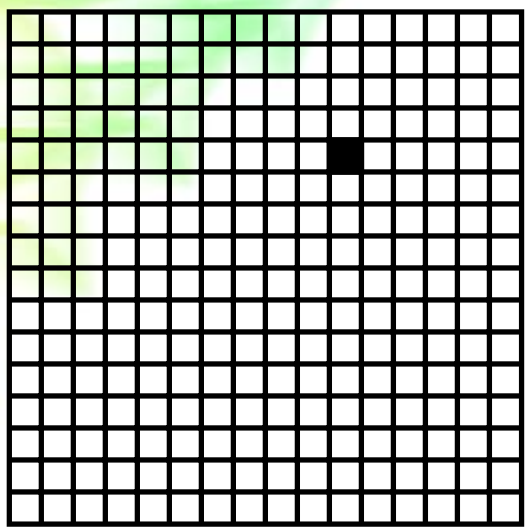
Combiner $S_i$S dans la solution finale

- Approche utilisée :
  - Plusieurs algorithmes de tri (quicksort, mergesort)
  - Multiplication de matrices et de polynômes

# Diviser pour régner

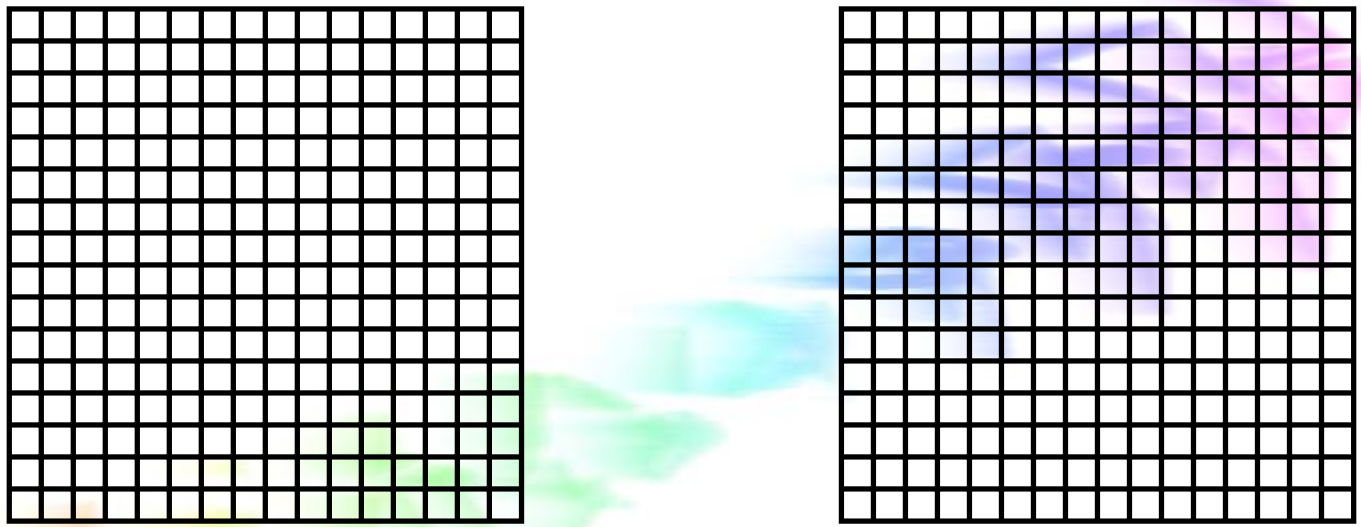
- Exemple : Un jeu d'échec de dimension $2^m$ ($2^{2^m}$ carrés)
- Une case vide arbitraire
- L tuiles
- Objectif : couvrir le jeu par les tuiles, orientation des tuiles quelconque

m=4



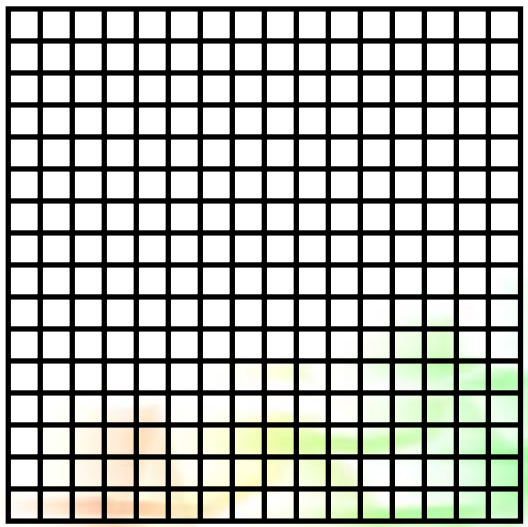
tuile

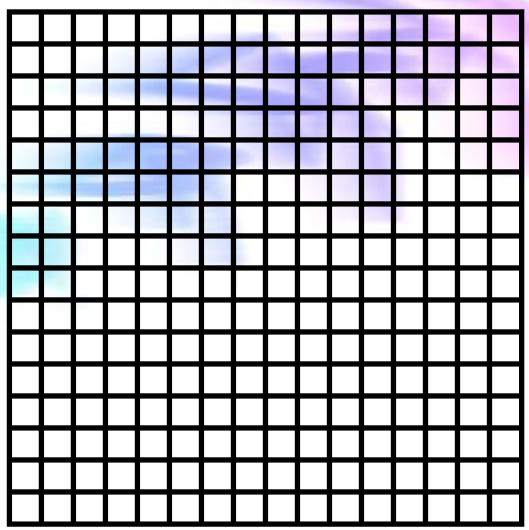
# Diviser pour régner



- Diviser le jeu en 4 zones de même taille
- Placer une tuile dans la frontière des 3 zones ne contenant pas la case vide ; chaque zone ayant une case vide.
- Partitionner les zones en 4 zones. Placer les tuiles / Chaque sous-zone possède une case vide.

# Diviser pour régner





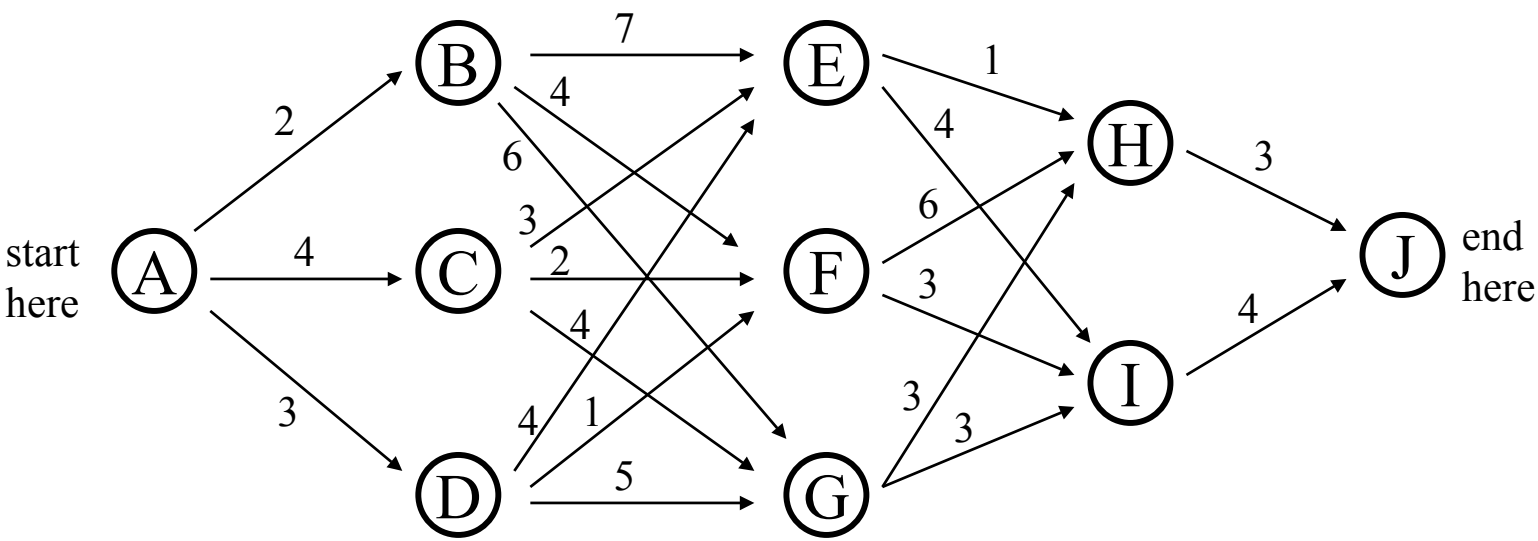
- Tâche primiare : placer une tuile dans un carré 2*2 dont une case vide (facile).

# Dynamic Programming

- provides a systematic method to determine an optimal combination of decisions; typically involves glueing together smaller optimal solutions, e.g., as in shortest path problem
- it's a general approach to solving many different types of problems
  - e.g., The Stagecoach Problem: Imagine you're in the 19th century and you are a fortune seeker Missouri who wishes to join the gold rush in California. In order to get there you must cross unsettled country where you are in danger of attack. You have a choice of states through which you can pass, so there are several possible routes to choose from. How should you choose the best route?
  - Your primary concern is safety. Suppose you have the following piece of information available to you. You may buy a life insurance policy as a stagecoach passenger. Standard policies are available for each leg of your journey.
  - Could you use this information to help you make an "optimal" choice of route?

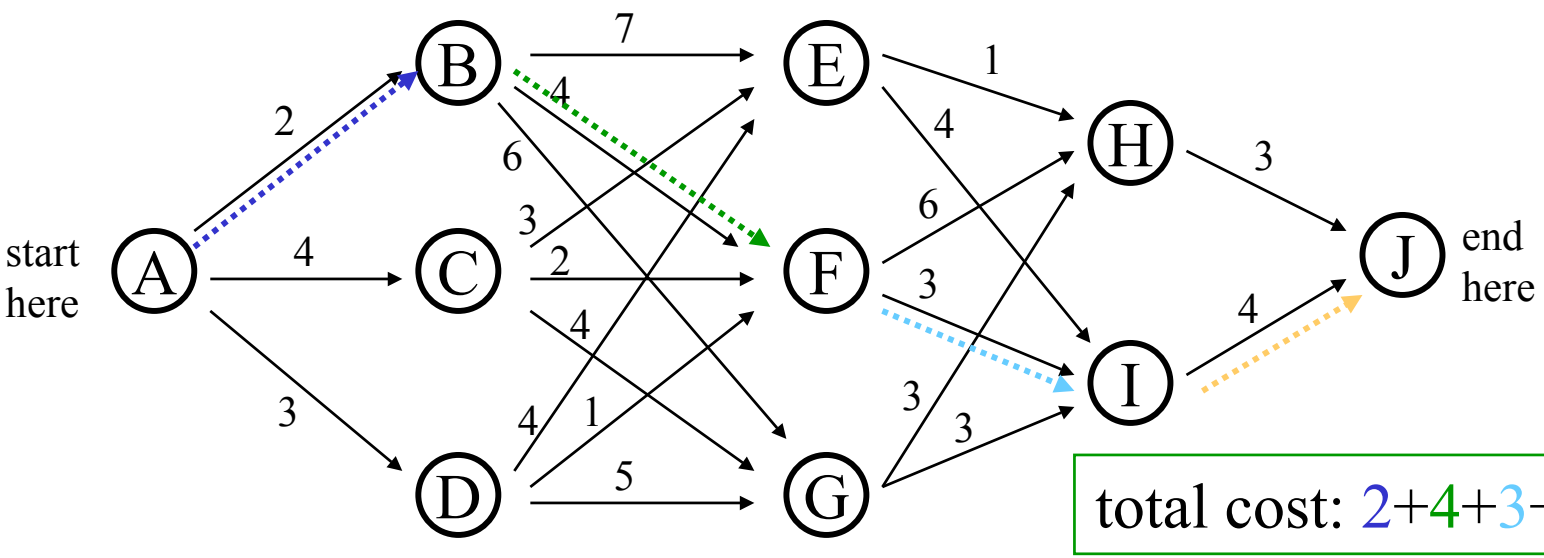
# The Stagecoach Problem

- The life insurance policy costs are a good indicator for the safety of each leg of your journey.
- The cheaper the policy, the safer the route (i.,e, the lower the risk).
- Here is some information of the costs of each stage of your journey:
  - You start in state A (Missouri) and your destination is state J (California)
  - The road system and costs are shown below.



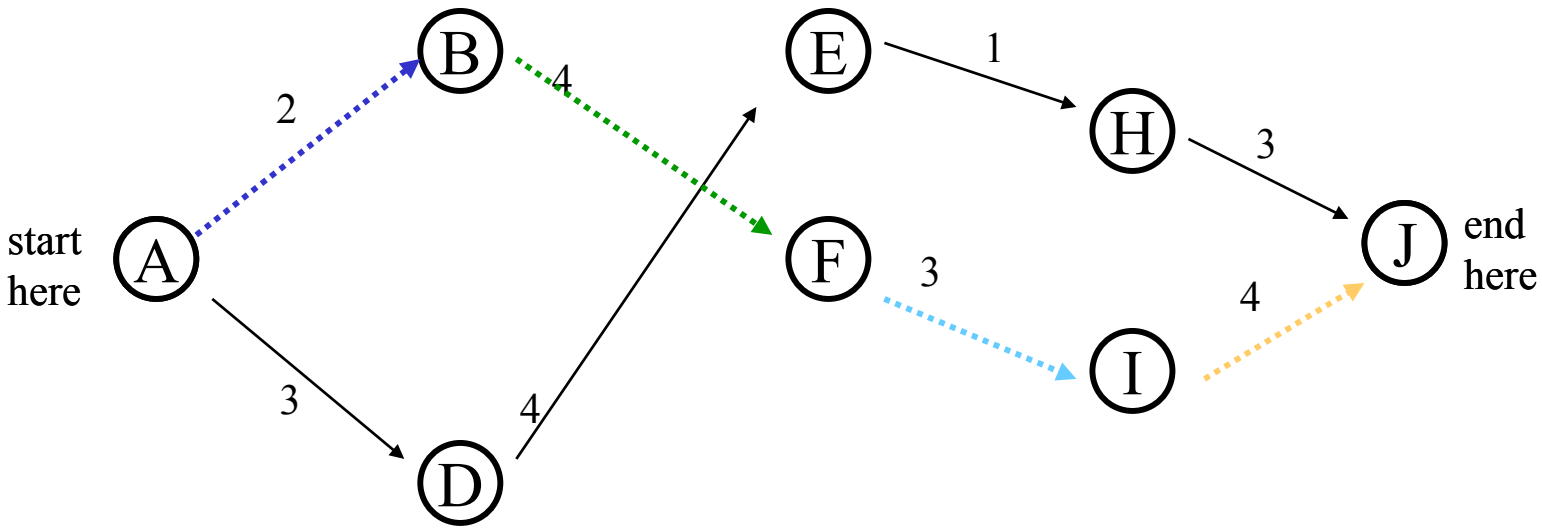
# A strategy …?

- What if you started at A and took the cheapest policy for each successive stage of your journey…. is this the best strategy?



# A better route ....

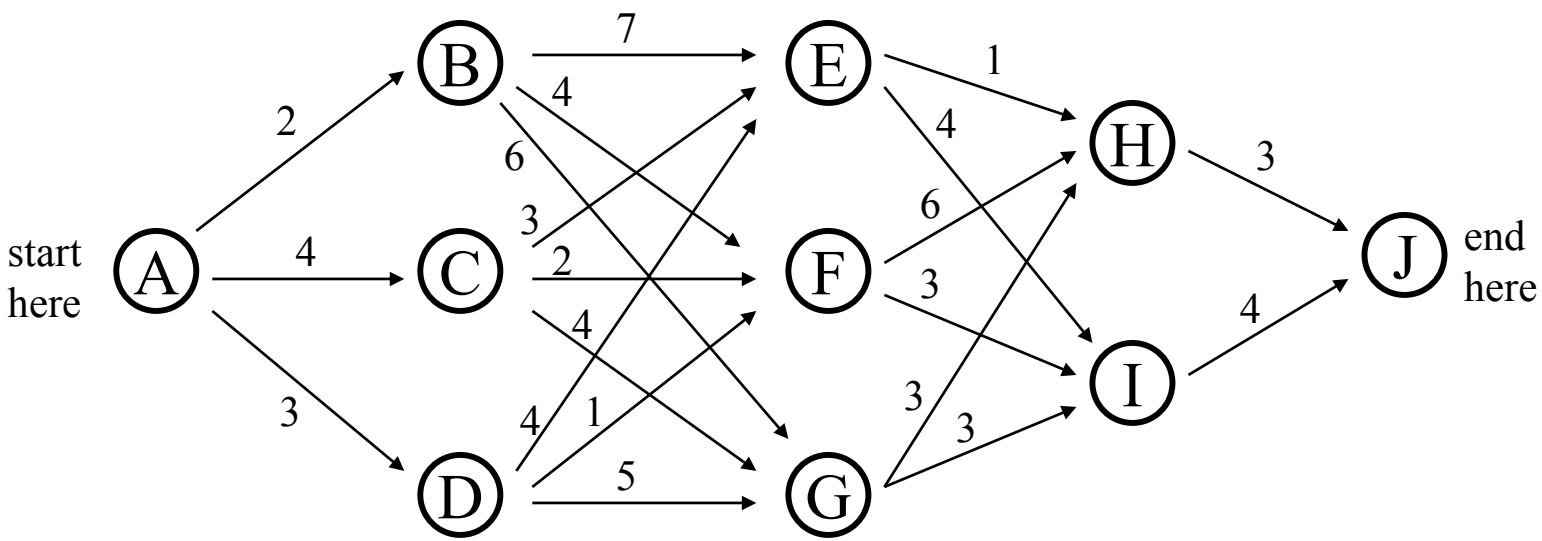
- It turns out there are “better” routes …
- e.g., route A-D-E-H-J has total cost 3+4+1+3=11




the route we chose previously was
A-B-F-I-J at a total cost of 13

# The Stagecoach Problem

- dynamic programming methods can be used to solve this problem in such a way that we do not have to exhaustively test every possible route
- the algorithm builds up the optimal solution by finding optimal solutions of smaller problems and then using these to build the final solution
- essentially we move backwards stage by stage and use recursion



# Formulation & Procedure:

- A decision at stage *n* is made as follows:

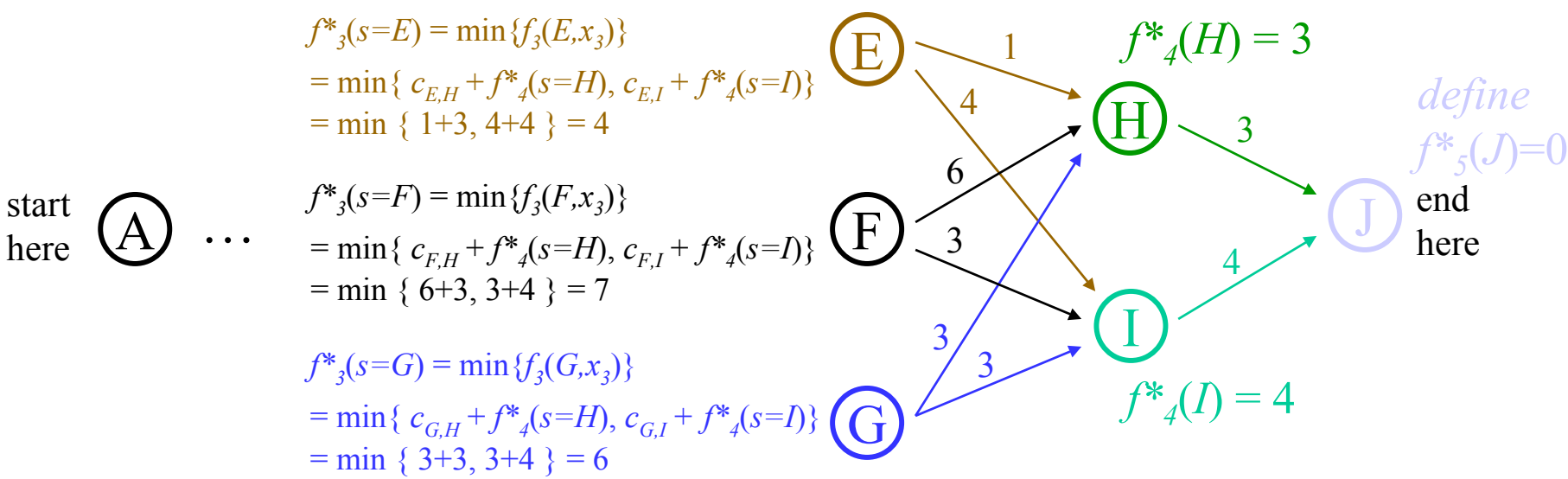
$f^*_n(s) = \min_{x_n} \{ f_n(s,x_n) \}$ = minimum future cost from state *s (stage n onwards)*

$f_n(s,x_n) = c_{s,x} + f^*_{n+1}(x_n)$

= immediate cost at stage *n* + min. future cost from stage *n+1* onwards

= total cost of best route to state $x_n$ given that you are in state *s* at stage *n-1*

start here




A ⟶ ... ⟶ $x_2$ ⟶ $x_3$ ⟶ $x_4$=J

stage 3 stage 4

# Example Procedure ctd ..

- A decision at stage *n* is made as follows:

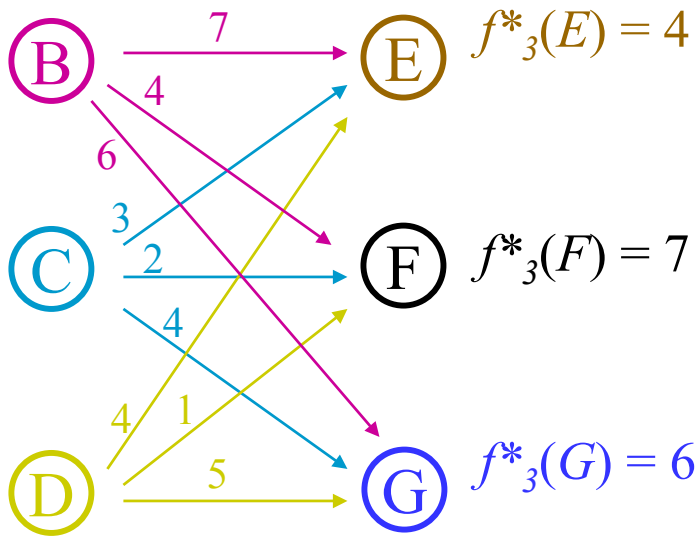
$f^*_n(s) = \min_{x_n} \{ f_n(s,x_n) \}$ = minimum future cost from state *s (ready to enter stage n)*

$f_n(s,x_n) = c_{s_n x} + f^*_{n+1}(x_n)$

$f^*_2(s=B) = \min\{f_2(B,x_2)\}$
$= \min\{ c_{B,E} + f^*_3(E),\ c_{B,F} + f^*_3(F),\ c_{B,G} + f^*_3(G)\}$
$= \min\ \{ 7+4,\ 4+7,\ 6+6 \} = 11$

$f^*_2(s=C) = \min\{f_2(C,x_2)\}$
$= \min\{ c_{C,E} + f^*_3(E),\ c_{C,F} + f^*_3(F),\ c_{C,G} + f^*_3(G)\}$
$= \min\ \{ 3+4,\ 2+7,\ 4+6 \} = 7$

$f^*_2(s=D) = \min\{f_2(D,x_2)\}$
$= \min\{ c_{D,E} + f^*_3(E),\ c_{D,F} + f^*_3(F),\ c_{D,G} + f^*_3(G)\}$
$= \min\ \{ 4+4,\ 1+7,\ 5+6 \} = 8$




A ⟶ … $x_1$ ⟶ $x_2$ ⟶ $x_3$

stage 3

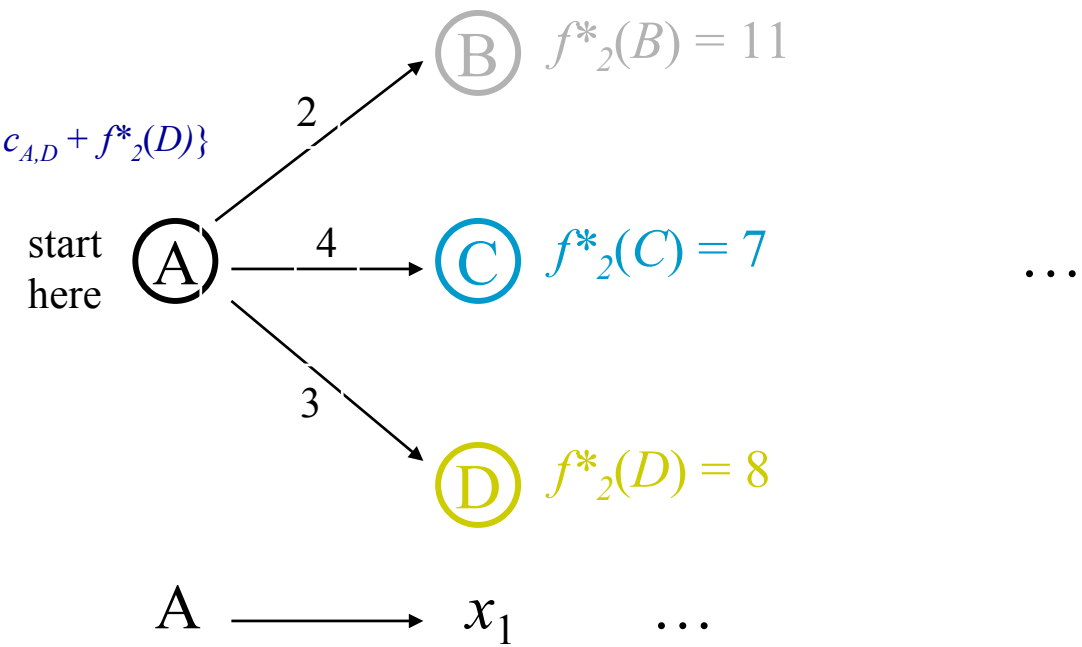
# Example Procedure (ctd)

- Once we reach A we can traceback the optimal path.
- At each step, we keep track of the state that minimized $f_n(s, x_n)$, and call it $x_n^*$
- $f_n^*(s) = f_n(s, x_n^*)$
- NOTE: there may be more than one state which leads to the optimal path

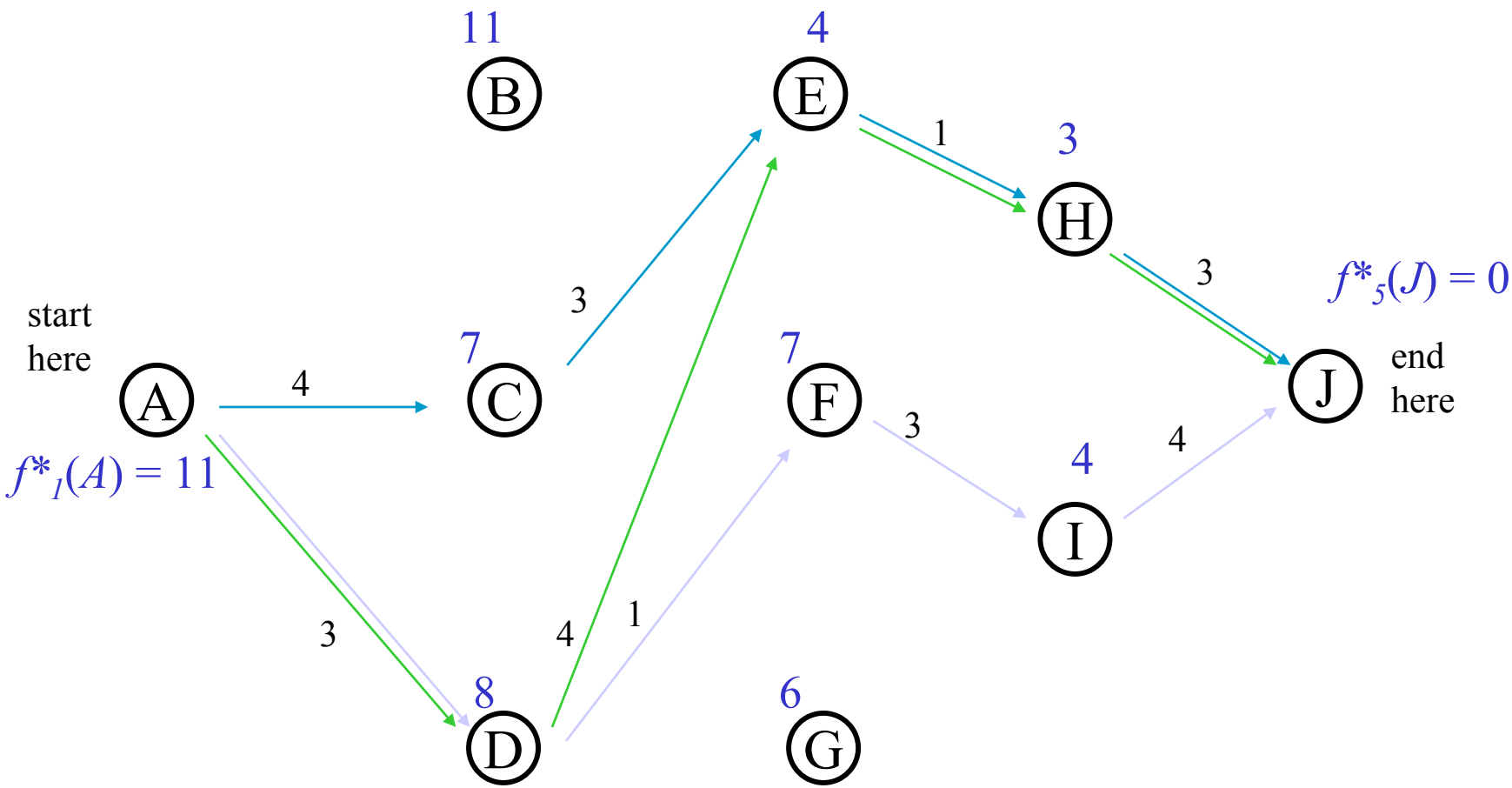
$f^*_1(s=A) = \min\{ f_1(A,x_1) \}$

$= \min\{ c_{A,B} + f^*_2(B), c_{A,C} + f^*_2(C), c_{A,D} + f^*_2(D)\}$

$= \min\{ 2+11, 4+7, 3+8 \} = 11$



# Tracing the optimal paths



# Characteristics of Dynamic Programming Problems

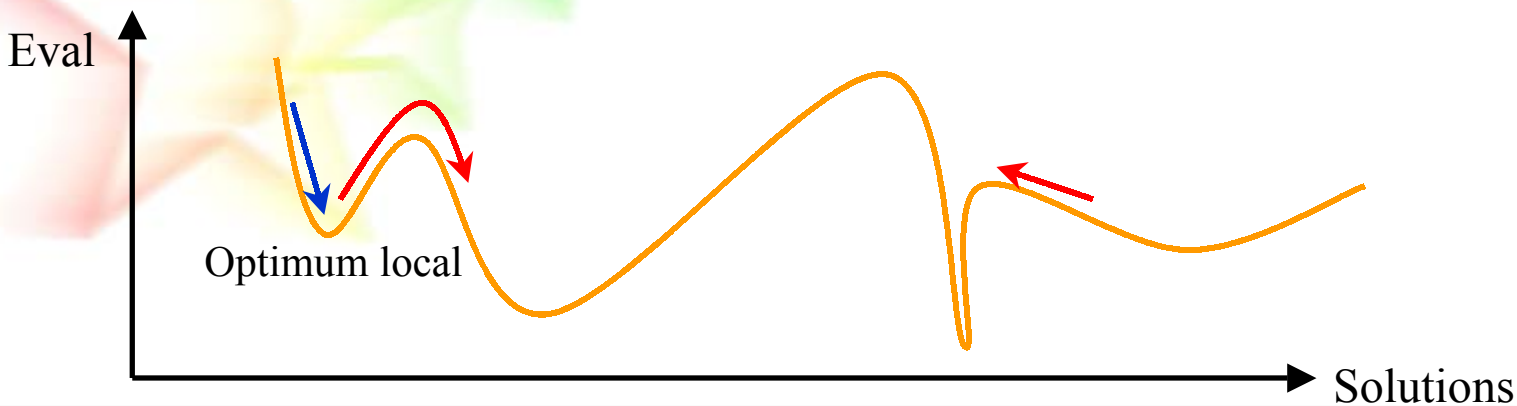
- problems can be divided into stages
- a decision is required at each stage
- each stage has a number of states associated with it (the states are the various possible conditions in which the system may be at that point)
- the effect of the decision is to transform the current state into a new state associated with the next stage (possibly according to some probability distribution)
- GOAL: to find an optimal policy for the overall problem by computing the optimal policy decision at each stage for *each* of the possible states
- given the current state, the best policy for the remaining stages is independent of the policy decisions in the previous stages. This is the *principle of optimality** for dynamic programming methods
- solution procedure finds the optimal policy for the final stage and a recursive relationship is used to find the optimal policy at the previous stages

# Recherche locale avancée

S'échapper des optima locaux

# Recuit simulé

- Utilisation depuis le début des années 80
- Metropolis [1953] : Algorithme de simulation du refroidissement d'un matériau (Thermodynamique)
- Kirkpatrick [1983] : Utilisation pour résoudre des problèmes d'optimisation
- Sélection itérative aléatoire de solutions voisines
- Accepter la solution générée avec une certaine probabilité (autoriser les dépalcements vers le haut)



# Recuit simulé

| Système physique | Problème d'optimisation |
|---|---|
| Energie | **Fonction d'évaluation** |
| Etat d'équilibre | **Solution optimale** |
| Rapid quenching | **Recherche locale** |
| Température | **Paramètre de contrôle T** |
| **Careful annealing** | **Simulated annealing** |

Analogie entre le système physique et le problème d 'optimisation

# Recuit simulé

Procédure Recuit simulé (minimisation)

t = 0 ; Initialiser $T=T_{max}$ /* Température initiale */ ;

Génération aléatoire solution initiale $v_c$ ; Evaluer $v_c$ ;

Répéter

Répéter

Générer un voisin aléatoire $v_n$ de $v_c$ ;

Si $eval(v_c) > eval(v_n)$ Alors $v_c=v_n$ ;

Sinon Si random[0,1] < $e^{\frac{-(eval(v_n)-eval(v_c))}{T}}$

Alors $v_c=v_n$ ;

Jusqu'à (condition-terminaison)

T=g(T,t) ; t=t+1 ;

Jusqu'à (critère d'arrêt)

# Recuit simulé (Probabilité)

$$P(\Delta E) = e^{\frac{eval(v_n) - eval(v_c)}{T}} = e^{\frac{-\Delta E}{T}}$$

- Rôle de la température :
  - T petit : recherche locale (fin de la recherche)
  - T grand : recherche aléatoire (début de la recherche)
- Rôle de la qualité de l'évaluation :
  - Plus petite est la différence de qualité, plus grande est la probabilité

# Recuit simulé

- Questions posées (+RL) :
  - Comment déterminer la température initiale ? : $T_{max}$
  - Comment déterminer la procédure de refroidissement g(T,t)? : T=rT  (0 < r <= 1)
  - Comment déterminer la condition de terminaison ? : un certain nombre d'itérations ($k_T$) à chaque pallier de température (dépend de la taille du voisinage).
  - Comment déterminer le critère d'arrêt ? : T >= $T_{min}$

# Recuit simulé (SAT)

Procédure Recuit simulé (SAT)

Génération aléatoire solution initiale v ; j=0 ;

Répéter

Si v satisfait les clauses Alors Return(v) ;

$T=T_{max}.e^{-j.r}$

Pour k=1 à nombre-de-variables Faire

Calculer $\Delta E$ (nombre de clauses) si $v_k$ est modifiée ;

Modifier la variable $v_k$ de v avec probabilité $(1+e^{-\frac{\Delta E}{T}})^{-1}$

j=j+1;

Jusqu'à ($T<T_{min}$)

# Recherche tabou

- Méthode proposée par F. Glover en 1986
  - *Future paths for integer programming and links to artificial intelligence*
- Introduire une notion de mémoire dans la stratégie de recherche (exploration)
- Recherche tabou : Interdiction de reprendre des solutions récemment visitées.
- Déterministe (recuit simulé - stochastique)

# Recherche tabou (SAT)

- SAT avec 8 variables (n=8)
- X=(0,1,1,1,0,0,0,1)
- Eval = maximiser les clauses satisfaites ; Eval(X)=27
- Voisinage(X) = 8 solutions (flip / bit)
- Meilleur voisin = flip de la troisième variable ; Eval= 31
- Mémoire M = indice de la variable modifiée + itération
- M(i)=j (quand j<>0) - j la plus récente itération où le ième bit est modifié.
- Information détruite après 5 itérations
- M(i)=j (quand j<>0) - le ième bit modifié après 5-j itérations

# Recherche tabou (SAT)

- Contenu de la mémoire après une itération

  M=0,0,5,0,0,0,0,0

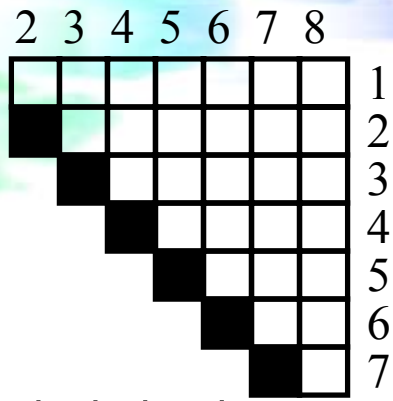
- Pour les 5 itérations suivantes, le 3ième bit ne peut être modifié (tabou)
- Après 5 itérations  M=3,0,1,5,0,4,2,0
- Seuls bits non tabou = 2, 5, et 8.
- Solution courante = 1,1,0,0,0,1,1,1 ; Eval=33
- Meilleur voisin = 1,1,0,0,1,1,1,1  ; Eval=32 (décroit)
- Après 6 itérations  M=2,0,0,4,5,3,1,0

# Recherche tabou (SAT)

- Restriction : un des voisins tabous représente la meilleure solution (jamais rencontré).
- Critère d'aspiration : Eliminer le caractère tabou des solutions.
- Mémoire de fréquence (long terme) H :
  - H(i)=j /* le ième bit a été modifié j fois pendant les h dernières itérations */ avec h >> 0
  - Montre la distribution des transformations réalisées
  - Comment utiliser cette information ? Diversification
- Diversification : explorer les possibilités les moins fréquentes

# Recherche tabou (TSP)

- Voisinage = Echange de 2 villes
- X=(2,4,7,5,1,8,3,6) possède 28 voisins
- Liste tabou = Matrice




- M(i,j) = Echange de ville (i,j) (i<j)
- H(i,j) même structure matricielle

# Recherche tabou (TSP)

```
Procédure Recherche tabou (TSP)
    Génération aléatoire tour ; count=0 ;
    Répéter
        Identifier un ensemble V de voisins ;
        Sélectionner le meilleur voisin ;
        Maj liste taboue et d'autres structures ;
        Si nouveau tour "best so far" Alors best=tour ;
        count=count+1 ;
    Jusqu'à count=ITER ;
```

# Recherche tabou

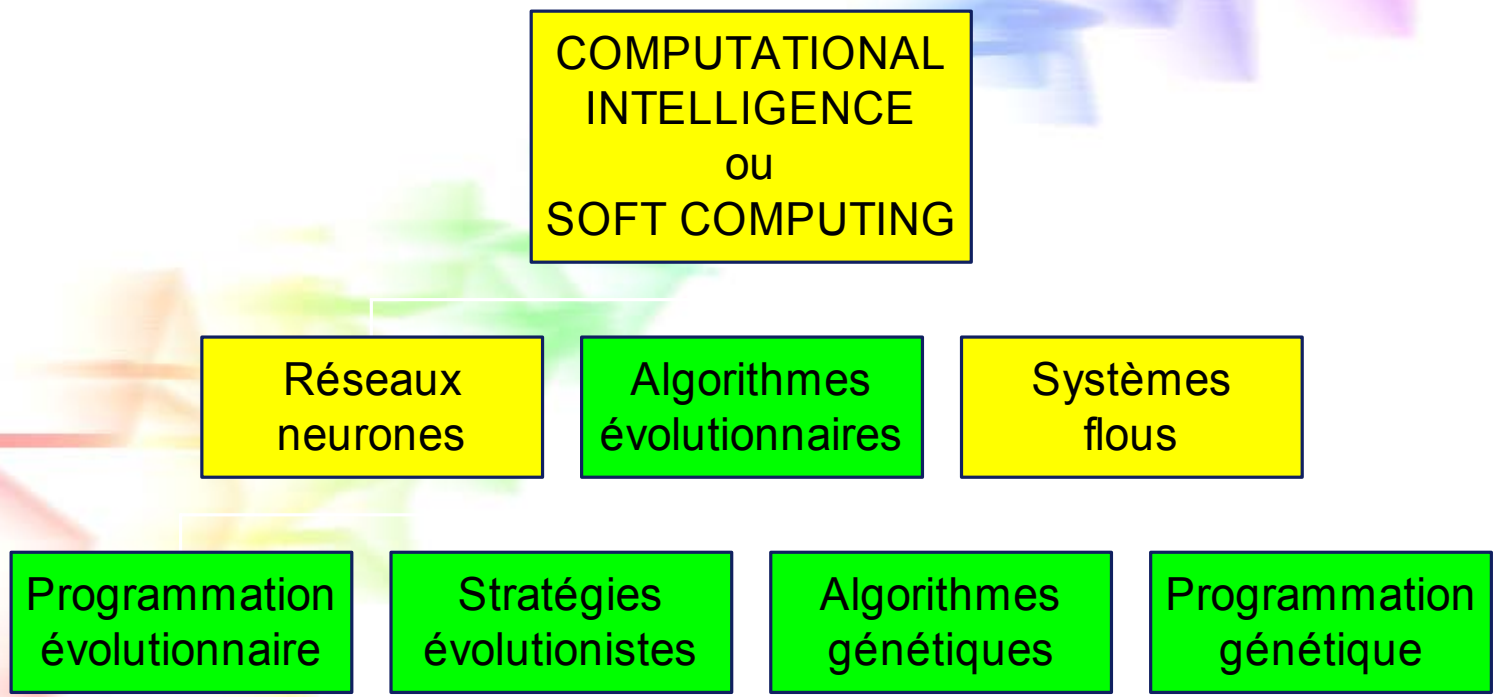
- Questions posées (+RL) :
  - Information à stocker dans la liste tabou ?
  - Longueur de la liste tabou (statique ou dynamique) ?
  - Mémoire long-terme (diversification)
  - Critère d'aspiration

# Algorithmes évolutionnaires

**Construire des solveurs de problèmes en s'inspirant de la nature**

- cerveau $\rightarrow$ Réseaux de neurones
- evolution $\rightarrow$ Algorithmes évolutionnaires

# Taxonomie


COMPUTATIONAL INTELLIGENCE ou SOFT COMPUTING
Réseaux neurones
Algorithmes évolutionnaires
Systèmes flous
Programmation évolutionnaire
Stratégies évolutionistes
Algorithmes génétiques
Programmation génétique

# Historique

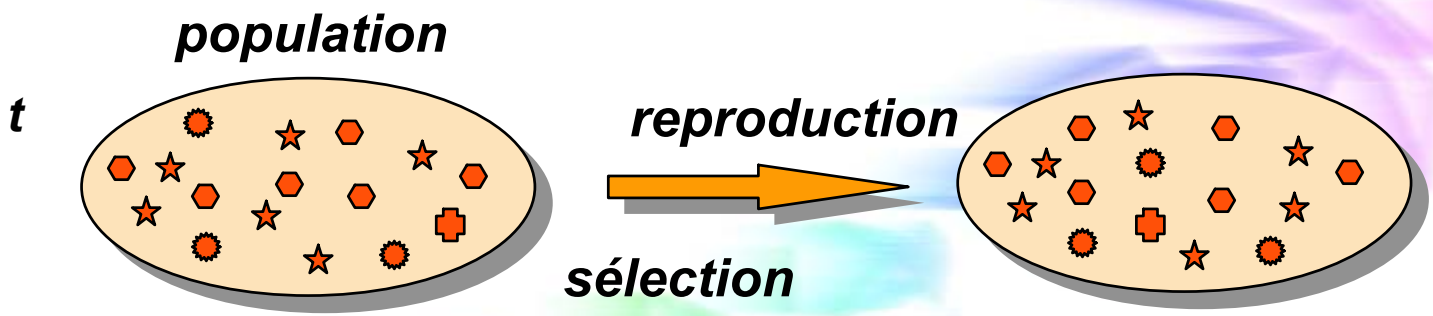
- L. Fogel 1962 (San Diego, CA): *Programmation évolutionnaire*
- J. Holland 1962 (Ann Arbor, MI): *Algorithmes génétiques*
- I. Rechenberg & H.-P. Schwefel 1965 (Berlin, Germany): *Stratégies évolutionistes*
- J. Koza 1989 (Palo Alto, CA): *Programmation génétique*

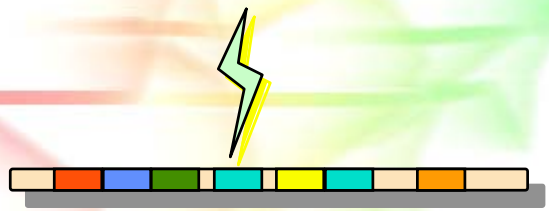
# Métaphore

| EVOLUTION | | PROBLEME D'OPTIMISATION |
|---|---|---|
| Individu | ⟷ | Solution Candidate |
| Adaptation | ⟷ | Qualité |
| Environnement | ⟷ | Problème |

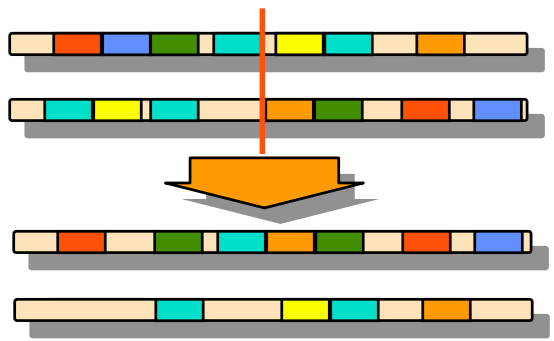
- Basés sur l'évolution d'une population de solutions
- Méthodes de recherche locale et constructives basées sur une solution

# Ingrédients



mutation

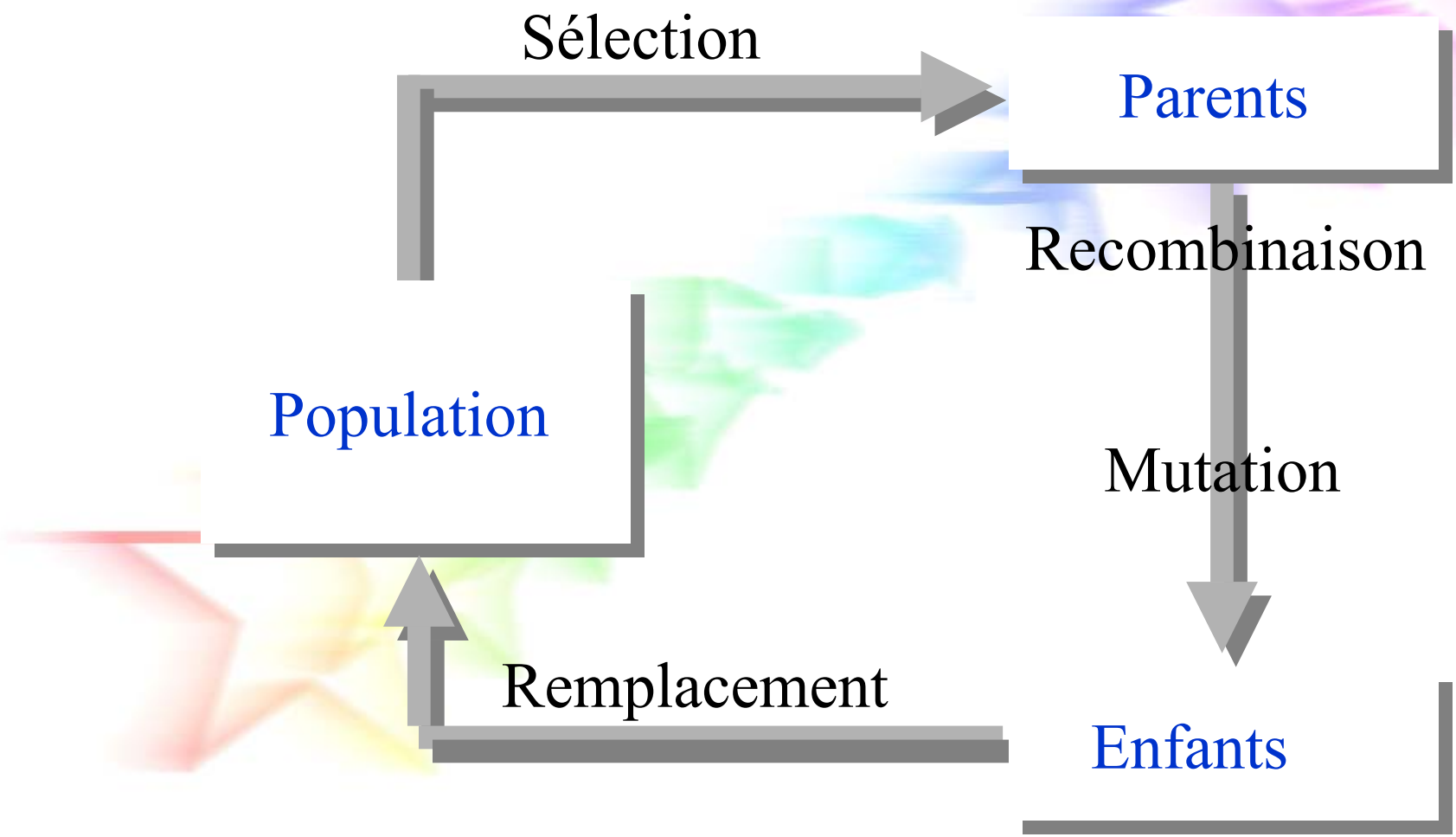


recombinaison

# Mécanismes d'évolution

- Diversifier par les opérateurs génétiques
  - mutation
  - recombinaison
- Intensifier par la sélection
  - des parents
  - des survivants

# Cycle de l'évolution



# Algorithmes évolutionnaires

```
Procédure Algorithme évolutionnaire
    t=0 ;
    Initialiser la population P(t) ;
    Evaluer la population P(t) ;
    Répéter
        Sélection P(t+1) à partir de P(t) ;
        Reproduire P(t+1) ;
        Evaluer P(t+1) ;
        Remplacer P(t+1) ;
        t=t+1 ;
     Jusqu'à critère d'arrêt ;
```

# Domaines d'application

- Numérique, Optimisation combinatoire
- Modélisation de systèmes et identification
- Planification et contrôle
- Design ingénieurs
- "Data Mining"
- Apprentissage
- Vie artificielle
- ...

# Performances

- Performances acceptables à des coûts réduits sur une grande variété de problèmes
- Parallélisme intrinsèque (robustesse, tolérance aux fautes)
- Supérieur à d'autres techniques sur des problèmes complexes avec :
  - taille importante de données, plusieurs paramètres libres
  - relation complexe entre les paramètres
  - plusieurs optima locaux

# Avantages

- Pas d'hypothèse sur l'espace de recherche
- Largement applicable
- Développment peu important & coût réduit
- Facile à incorporer d'autres méthodes (méthodes hybrides)
- Solutions sont interprétables (ce qui n'est pas le cas pour les réseaux de neurones)
- Peuvent être exécutés de façon interactive (s'accommoder à des solutions utilisateur
- Fournissent plusieurs alternatives (solutions)
- S'adaptent rapidement au changement de l'environnement (données, objectif, …).
- Co-évolution (compétition), Parallélisme et distribution, ...

# Inconvénients

- Ne garantissent pas de trouver la solution optimale dans un temps fini
- Base théorique faible
- Besoin d'initialiser plusieurs paramètres (parameter tuning)
- Coût d'exécution important / recherche locale et constructives

# Ouvrages


- Th. Bäck, "Evolutionary Algorithms in Theory and Practice", *Oxford University Press, 1996*
- L. Davis, "The Handbook of Genetic Algorithms", *Van Nostrand & Reinhold, 1991*
- D.B. Fogel, "Evolutionary Computation", *IEEE Press, 1995*
- D.E. Goldberg, "Genetic Algorithms in Search, Optimisation and Machine Learning", *Addison-Wesley, 1989*
- J. Koza, "Genetic Programming", *MIT Press, 1992*
- Z. Michalewicz, "Genetic Algorithms + Data Structures = Evolution Programs", *Springer, 3rd ed., 1996*
- H.-P. Schwefel, "Evolution and Optimum Seeking", *Wiley & Sons, 1995*

# Résumé

## ALGORITHMES EVOLUTIONNAIRES :

- Sont basés sur un métaphore biologiste
- possède un nombre important d'applications potentielles
- de plus en plus populaires (dans plusieurs domaines)
- donnent de bonnes performances à des coûts réduits
- AND IT'S FUN !

# Comment concevoir un Algorithme Evolutionnaire

# Etapes à suivre

Dans la conception d'un algorithme évolutionnaire, il y a plusieurs étapes à réaliser :

- Construire une représentation
- Comment initialiser la population
- Comment transformer un génotype en un phénotype
- Comment évaluer un individu

# Etapes à suivre

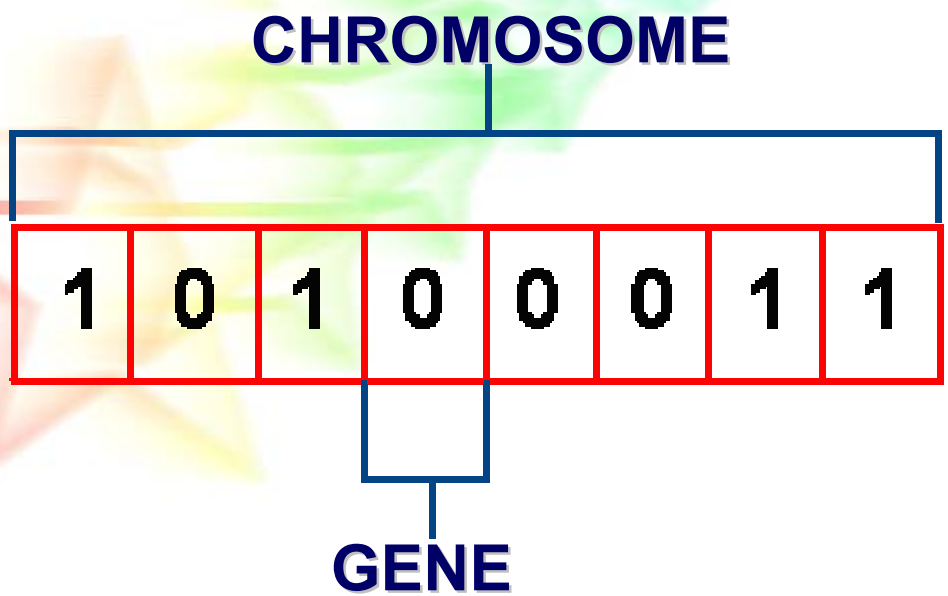
- Opérateur(s) de mutation adapté
- Opérateur(s) de recombinaison adapté
- Comment gérer la population
- Comment sélectionner les parents
- Comment remplacer les individus
- Critère d'arrêt

# Représentation

- Méthode de représentation d'un individu comme un génotype.
- Plusieurs solutions possibles : dépend entre autres du problème traité.
- Dans le choix de la représentation, on doit prendre en compte la procédure d'évaluation des génotypes, et des opérateurs génétiques utilisés.

# Exemple : Représentation discrète (Alphabet binaire)

- Représentation d'un individu utilisant des valeurs discrètes (binaire, entiers, ou tout système avec un ensemble discret de valeurs).
- Exemple de représentation binaire **: Problème SAT**



# Exemple : Représentation discrète (Alphabet binaire)

**8 bits Génotype**


**Phénotype:**

- Entier
- Réel
- Ordonnancement
- ...
- ?

# Exemple : Représentation discrète (Alphabet binaire)

Phénotype peut être un entier

**Génotype:**

| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**Phénotype:**

**= 163**

$1*2^7 + 0*2^6 + 1*2^5 + 0*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 1*2^0 =$

$128 + 32 + 2 + 1 = 163$

# Exemple : Représentation discrète (Alphabet binaire)

Phénotype peut être un réel

ex. un nombre entre 2.5 et 20.5 utilisant 8 digits binaires (problème NLP)

**Génotype:**

| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**Phénotype:**

**= 13.9609**

$$x = 2.5 + \frac{163}{256}(20.5 - 2.5) = 13.9609$$

# Exemple : Représentation discrète (Alphabet binaire)

Phénotype peut être un ordonnancement

e.g. 8 jobs, 2 machines

**Génotype:**


=

| Job | Machine |
|---|---|
| 1 | 2 |
| 2 | 1 |
| 3 | 2 |
| 4 | 1 |
| 5 | 1 |
| 6 | 1 |
| 7 | 2 |
| 8 | 2 |

**Phénotype**

# Exemple : Représentation basée sur des réels

- Les individus sont représentés comme un tuple de n valeurs réelles (exemple : NLP) :

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \bullet \\ x_n \end{bmatrix}, x_i \in R$$

- La fonction objectif qui associe à chaque tuple un nombre réel :

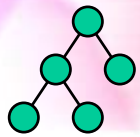
$$f : R^n \rightarrow R$$

# Exemple : Représentation basée sur un ordre

- Les individus sont représentés comme des permutations
- Utilisée pour des problèmes d'ordonnancement - séquencage
- **Exemple** : Voyageur de commerce (TSP) où chaque ville lui est affectée un nombre unique de 1 à *n.* Une solution peut être (5, 4, 2, 1, 3).
- Besoin d'opérateurs spécifiques pour générer des individus valides.

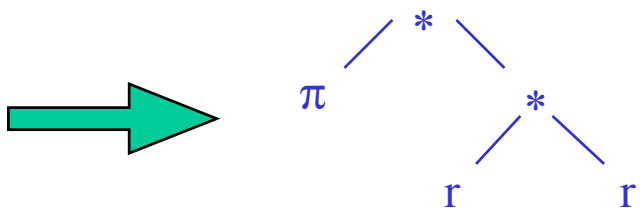
# Exemple : Représentation basée sur les arbres

- Les individus dans la population sont des arbres.



- Toute S-expression peut être représentée comme un arbre de fonctions et de terminaux.
- Les fonctions et les terminaux :
  - Fonctions : sin, cos, add, sub, and, If-Then-Else,...
  - Terminaux : X, Y, 0.456, true, false, π, …
- Exemple: calcul de la surface d'un cercle :

$$\pi * r^2$$



# Autres représentations

- Matrices
- Graphes
- Règles (longueur variable)
- Machines d'états finis
- Programme
- ...

# Initialisation de la population

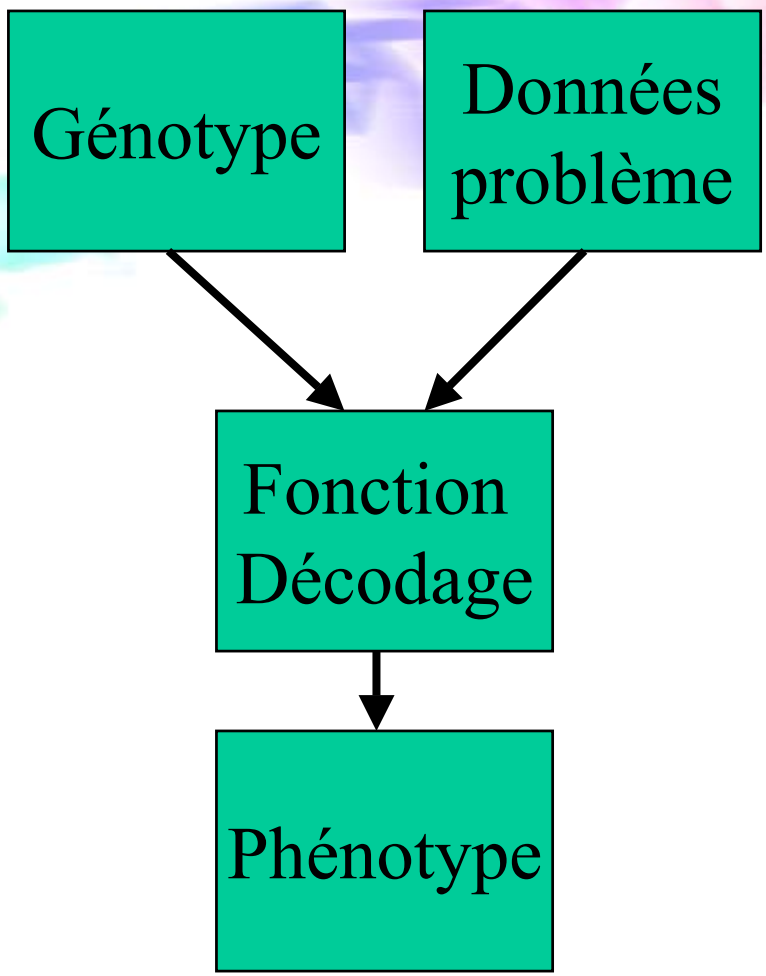
- Uniforme dans l'espace de recherche … Si possible
  - Chaînes binaires : 0 ou 1 avec une probabilité de 0.5
  - Nombres réels : Uniforme sur un intervalle donné (OK pour des valeurs bornées seulement)
- Initialiser la population avec des résultats antérieurs ou obtenus par d'autres méthodes Attention ! :
  - Perte possible de diversité
  - Biais impossible à corriger

# Exemple : Représentation basée sur les arbres

- Choisir une fonction *f* de façon aléatoire à partir de l'ensemble des fonctions *F*. f représente la racine de l'arbre.
- Chaque fonction possède un nombre fixe d'arguments (unaire, binaire, …, n-aire), *z(f)*. Pour chaque argument, créer un nœud à partir de F ou de l'ensemble des arguments T.
- Si un terminal est sélectionné alors c'est une feuille.
- Si une fonction est sélectionnée alors le processus est itéré.
- Une profondeur maximal est utilisé pour arrêter le processus.

# Obtenir le phénotype à partir du génotype

- La plupart du temps, produire le phénotype à partir du génotype est un processus simple.
- Le génotype peut être un ensemble de paramètres d'un algorithme, qui à partir des données du problème produit le phénotype.



# Evaluation d'un individu

- L'étape la plus coûteuse pour des applications réelles.
- Peut être une procédure, un simlateur "boite noire", ou tout processus externe
- On peut approximer la fonction objectif.

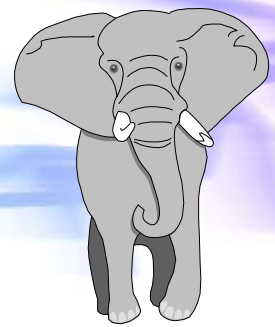
# Evaluation d'un individu

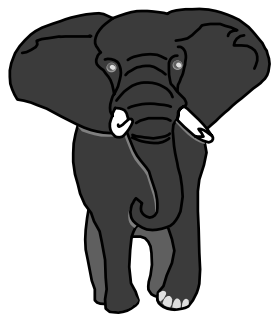
- Prise en compte des contraintes - si le phénotype ne respecte pas certaines contraintes :
  - pénaliser la fonction coût
  - utiliser des méthodes évolutionnaires spécifiques
- Optimisation évolutionnaire multi-critère : fournit un ensemble de solutions compromis

# Opérateurs de mutation

- On peut avoir un ou plusieurs opérateurs de mutation pour une représentation donnée.
- Quelques points importants :
  - Au moins un opérateur de mutation permet d'atteindre tout point de l'espace de recherche.
  - La probabilité de mutation est importante et doit être contrôlée.
  - La mutation doit produire des chromosomes valides.

# Exemple : Mutation pour représentation discrète





avant 1 1 1 1 1 1 1

après 1 1 1 0 1 1 1

Gène muté

Mutation souvent appliquée avec une probabilité $p_m$ pour chaque gène

# Exemple : Mutation pour représentation basée sur des réels

Perturber les valeurs en rajoutant un bruit aléatoire :

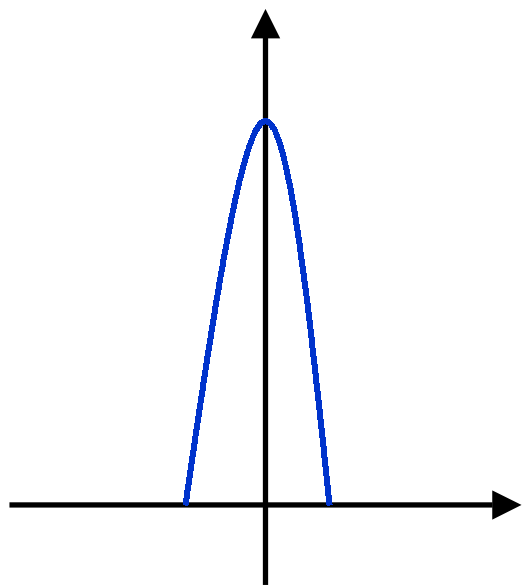
Souvent, une distribution normale de Gauss $N(0,\sigma)$ est utilisée, où

- 0 est la valeur moyenne
- $\sigma$ est la déviation standard

et

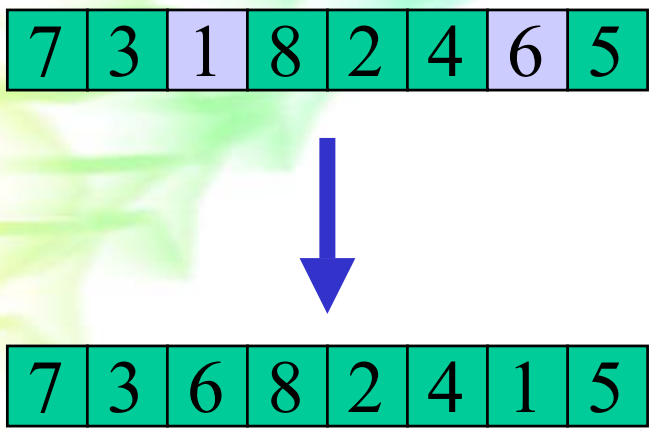
$$x'_i = x_i + N(0,\sigma_i)$$

pour chaque paramètre



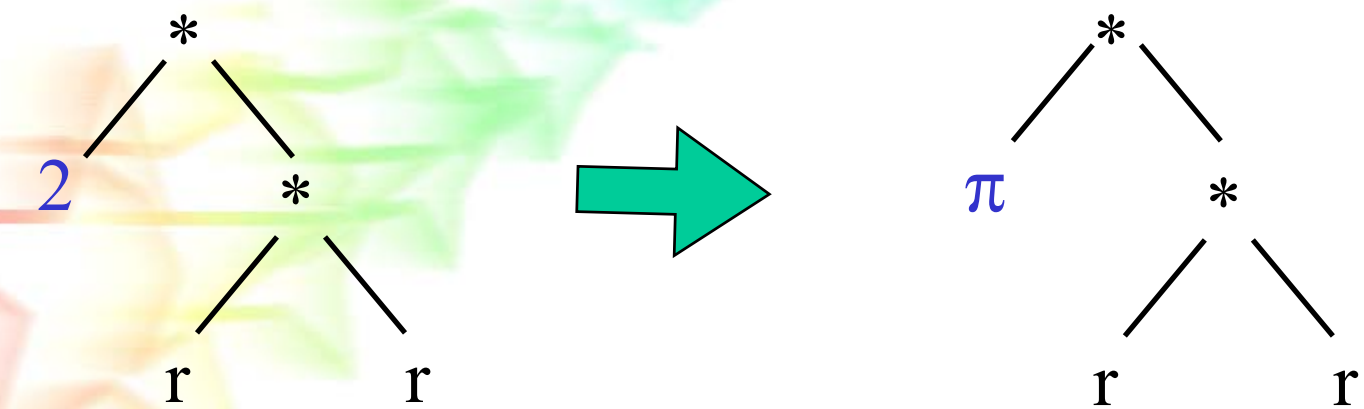
# Exemple : Mutation pour représentation basée sur l'ordre (Swap)

Sélectionner de façon aléatoire deux gènes et les transposer



# Exemple : Mutation pour représentation basée sur des arbres

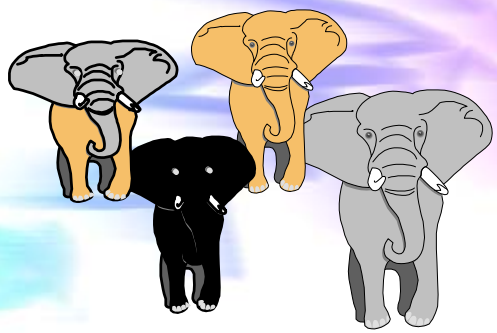
Mutation “un-point” sélectionne un nœud et le remplace par un autre



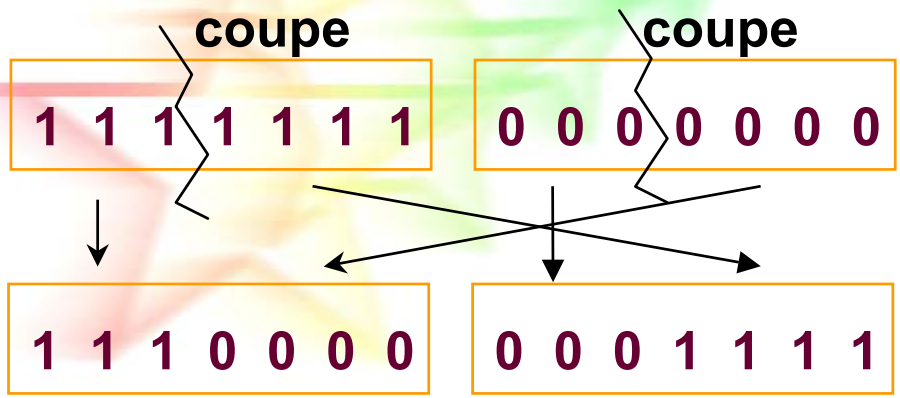
# Opérateurs de recombinaison

- On peut avoir un ou plusieurs opérateurs de recombinaison pour une représentation donnée.
- Quelques points importants :
  - L'enfant doit hériter une partie de son génotype de **chaque** parent.
  - L'opérateur de recombinaison doit être conçu en fonction de la représentation.
  - Recombination doit produire des chromosomes valides.

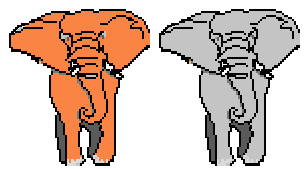
# Exemple : Recombinaison pour représentation discrète

Population complète :  . . .

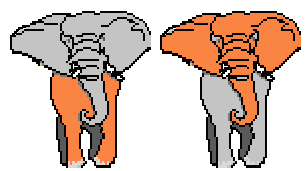
Chaque chromosome est partitionné en n pièces qui sont recombinées (Exemple pour *n=1*)




**parents**

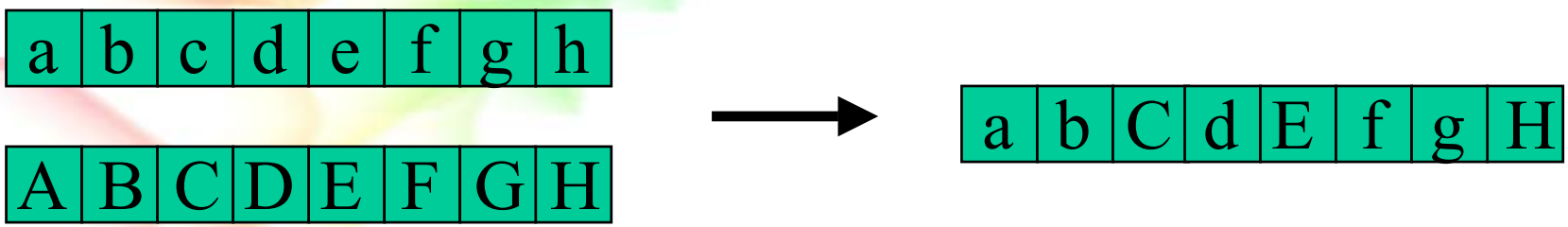


**enfants**



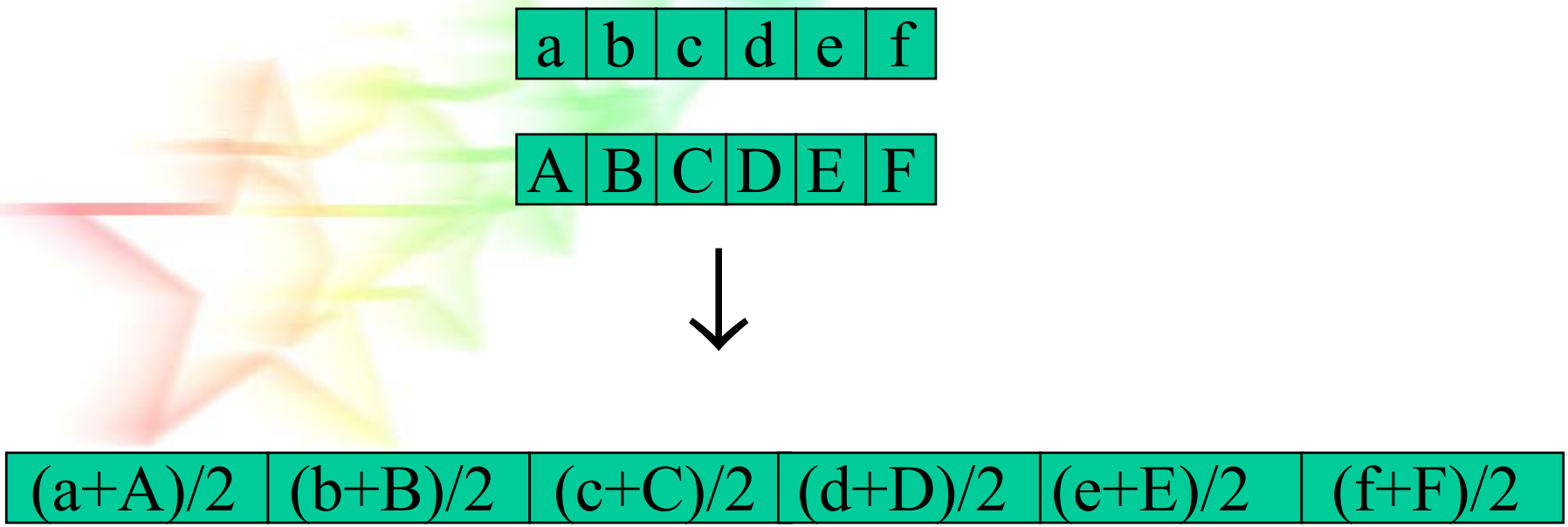
# Exemple : Recombinaison pour représentation basée sur des réels

Recombinaison discrète (crossover uniforme) :
Etant donné 2 parents, un enfant est généré de la façon suivante :



# Exemple : Recombinaison pour représentation basée sur des réels

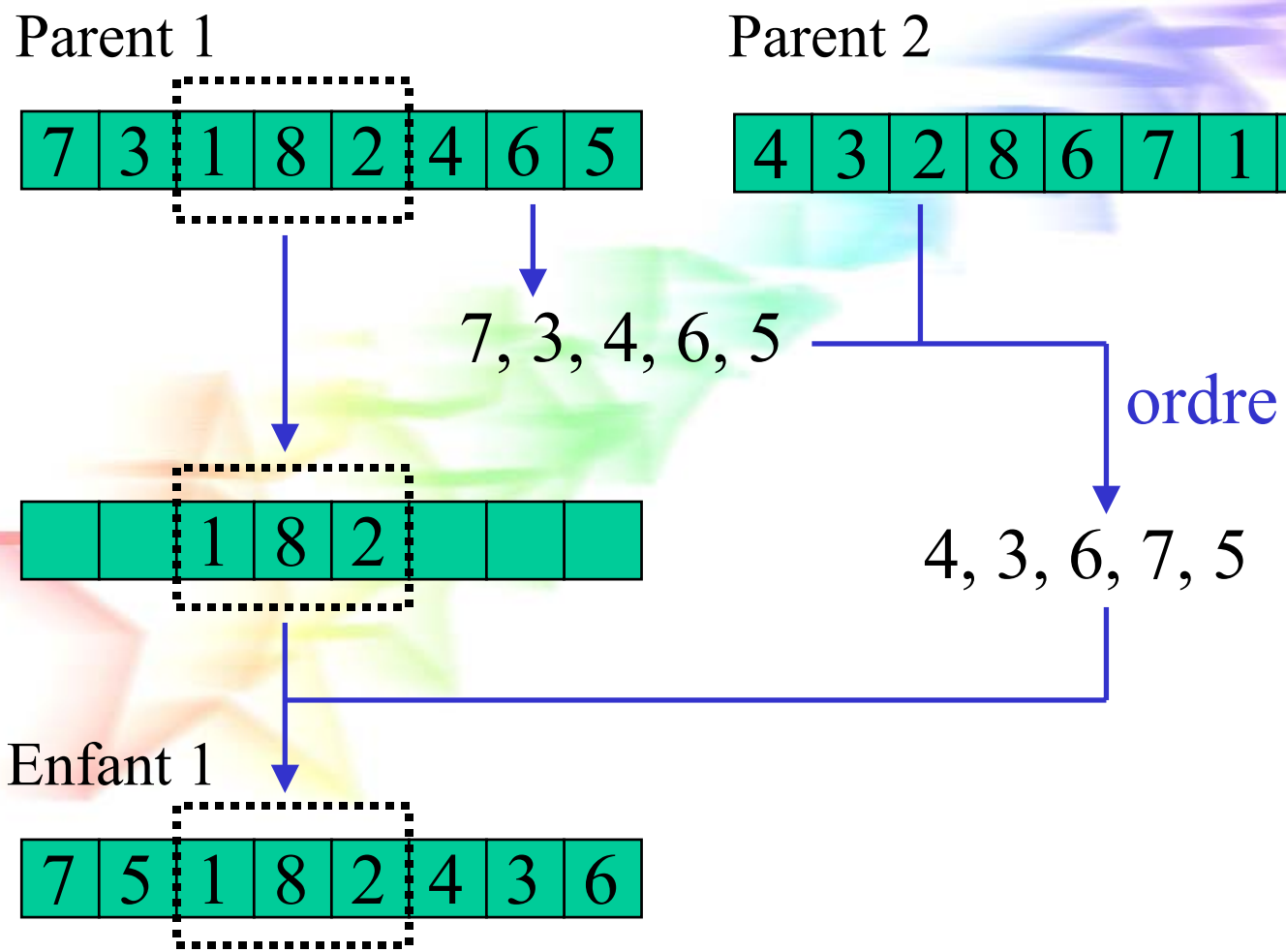
Recombinaison intermédiaire (crossover arithmétique) : Etant donné 2 parents, un enfant est généré de la façon suivante



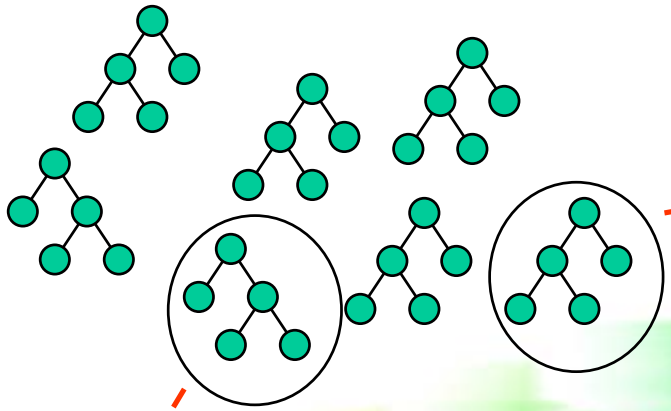
# Exemple : Recombinaison représentation basée sur un ordre (Order1)

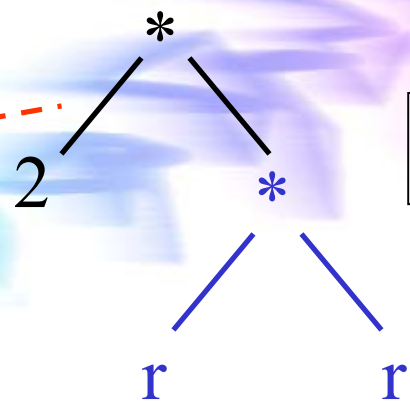
- Choisir une partie arbitraire à partir du premier parent et la copier dans le premier enfant
- Copier les gènes restants qui n'ont pas été copiés du premier parent au premier enfant :
  - commencer à droite du point de coupure de la partie copiée
  - utiliser l'ordre des gènes du second parent
- Répéter ce processus avec le rôle des parents inversé

# Exemple : Recombinaison représentation basée sur un ordre (Order1)

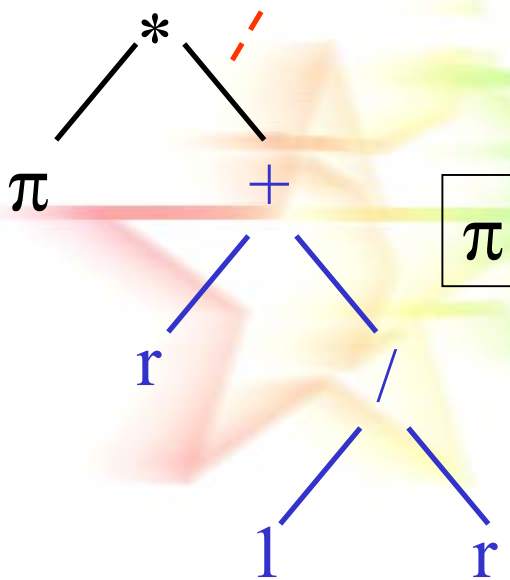


# Exemple : Recombinaison pour représentation basée sur un arbre





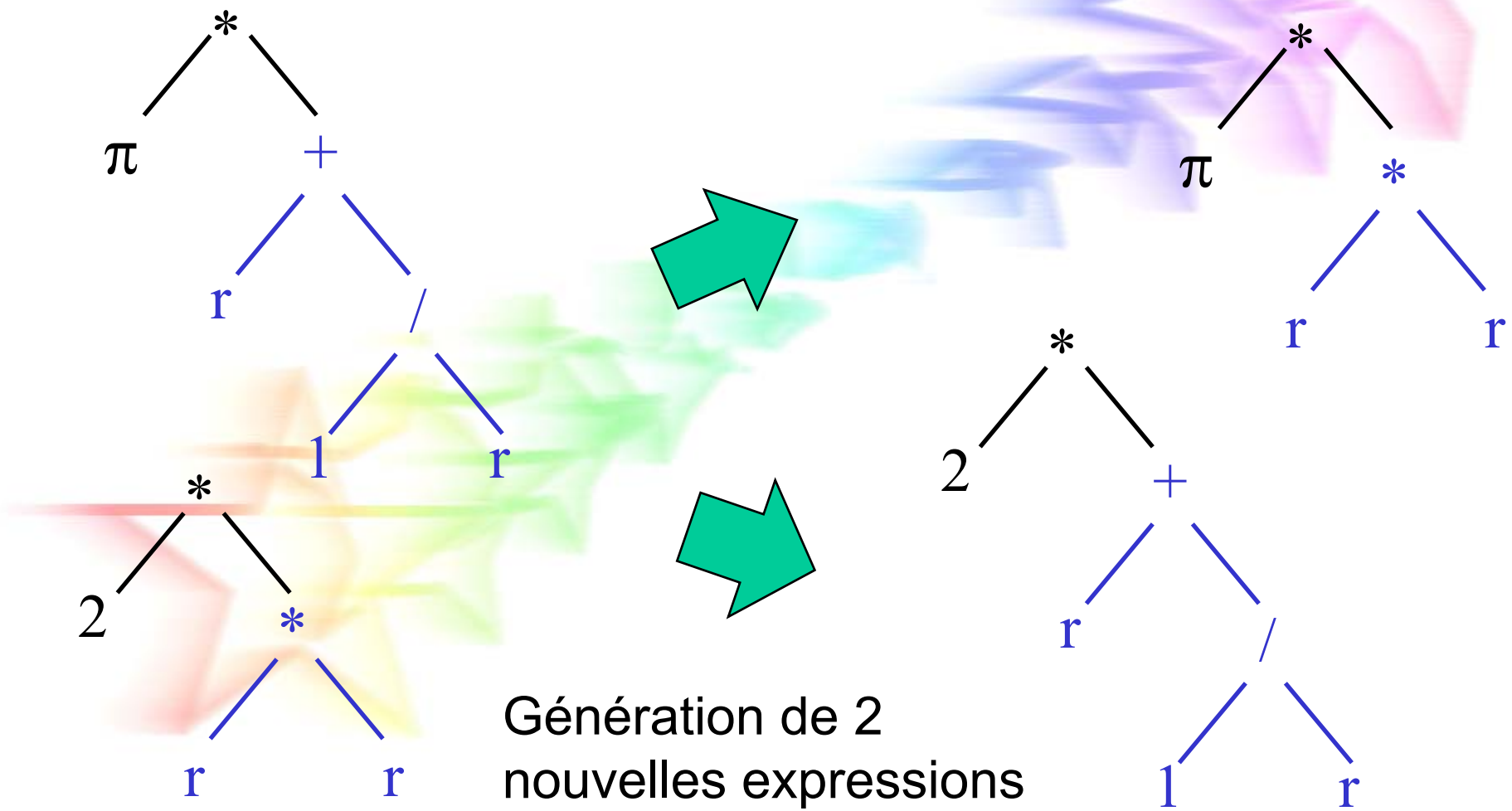

$2 * (r * r)$




$\pi * (r + (l / r))$

Deux sous-arbres sont sélectionnés pour être échangés

# Exemple : Recombinaison pour représentation basée sur un arbre




Génération de 2 nouvelles expressions

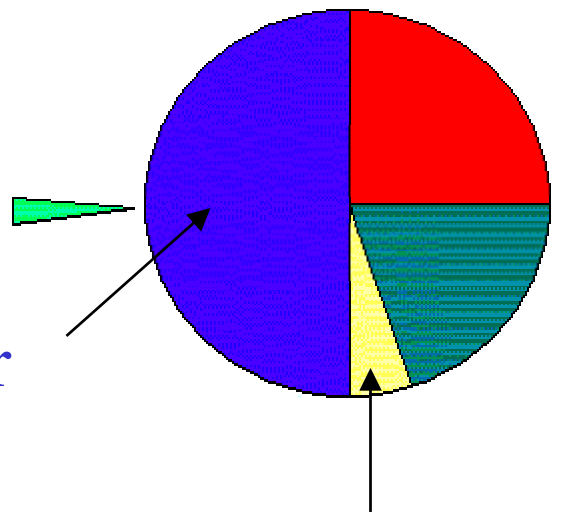
# Stratégies de sélection

- Principe de sélection : “Meilleur est un individu, plus grande est sa probabilité d’être parent”.
- Ceci donne une pression de sélection qui tend à faire converger la population vers de meilleures solutions.
- Faire attention à donner aux individus moyens une chance d’être parents - Ceci peut introduire un matériel génétique utile.

# Exemple : Sélection Roulette

- Nombre de fois $f_i$ est sélectionné pour une reproduction est : $f_i / \bar{f}$

- Meilleurs individus ont :
  - plus d'espace
  - plus de chances d'être sélectionnés



# Exemple : Sélection Roulette

## Inconvénients :

- Danger de convergence prématurée - les individus exceptionnels dominent la population de façon rapide.
- Pression de sélection faible lorsque les valeurs de coût sont proches les uns aux autres.
- Comportement différent sur des versions transposées de la même fonction objectif.

# Exemple : Sélection Roulette

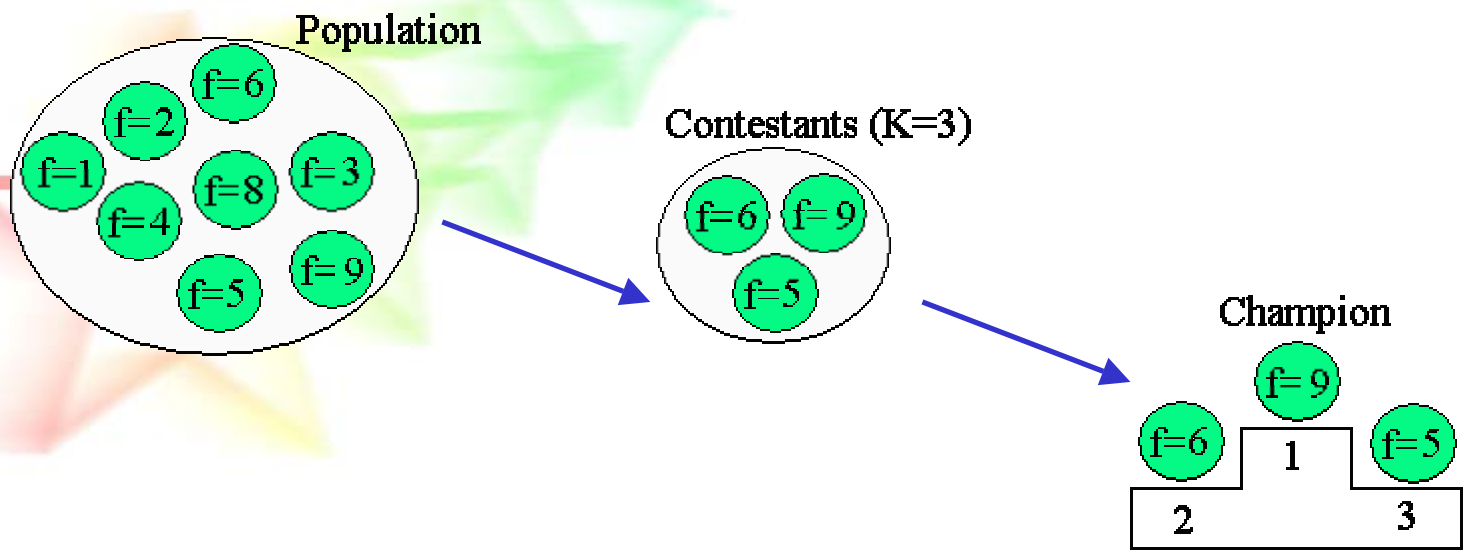
## Fitness scaling :

Commencer avec la fonction originale $f$:

- Normaliser pour vérifier :
  - Coût inférieur représente un meilleur individu.
  - Coût optimal égal à 0.
  - Coût entre 0 et 1.
  - La somme des valeurs coût est égale à 1.

# Exemple : Sélection par tournoi

- Sélectionner $k$ individus de façon aléatoire, sans remplacement
- Choisir le meilleur
  - $k$ est la taille du tournoi



# Exemple : Sélection basée sur le rang

- Les individus sont triés par rapport à leur coût, du meilleur au plus mauvais. Leur place dans cette liste triée est appelée rang.

- Au lieu d'utiliser le coût d'un individu, le rang est utilisé par une fonction de sélection des individus. La fonction est biaisée envers les individus de meilleurs rangs (= bonnes solutions).

# Exemple : Sélection basée sur le rang

- Fitness: *f(A) = 5, f(B) = 2, f(C) = 19*
- Rang : *r(A) = 2, r(B) = 3, r(C) = 1*

$$h(x) = \min + (\max - \min) * \frac{(r(x) - 1)}{n - 1}$$

- Fonction: *h(A) = 3, h(B) = 5, h(C) = 1*
- Proportion sur la roulette :

  *p(A) = 11.1%, p(B) = 33.3%, p(C) = 55.6%*

# Stratégies de remplacement

- La pression de sélection est aussi utilisée dans l'étape de remplacement (survivants des deux populations).
- On peut utiliser des méthodes de remplacement stochastiques ou déterministes.
- On peut décider de ne pas remplacer le meilleur individu de la population : Elitisme.

# Critères d'arrêt

- La solution optimale est trouvée !
- Limite sur les ressources CPU
- Nombre maximal d'évaluations de la fonction objectif
- Un certain nombre de générations sans améliorations

# Performances des algorithmes

- Ne Jamais tirer des conclusions sur une seule exécution
  - utiliser de mesures statistiques (moyenne, variance, ...) à partir d'un nombre suffisant d'exécutions indépendantes
- Point de vue applicatif
  - Perspective design :
    - trouver une très bonne solution au moins une fois
  - Perspective de production :
    - trouver une bonne solution à chaque exécution

# Performances des algorithmes (2)

Se rappeler du principe WYTIWYG :

“What you test is what you get”

Ne pas évaluer les performances de l’algorithme sur des données “jouet” de petites tailles et s’attendre aux mêmes résultats sur des données réelles de grandes tailles

# A Taxonomy of Hybrid Metaheuristics

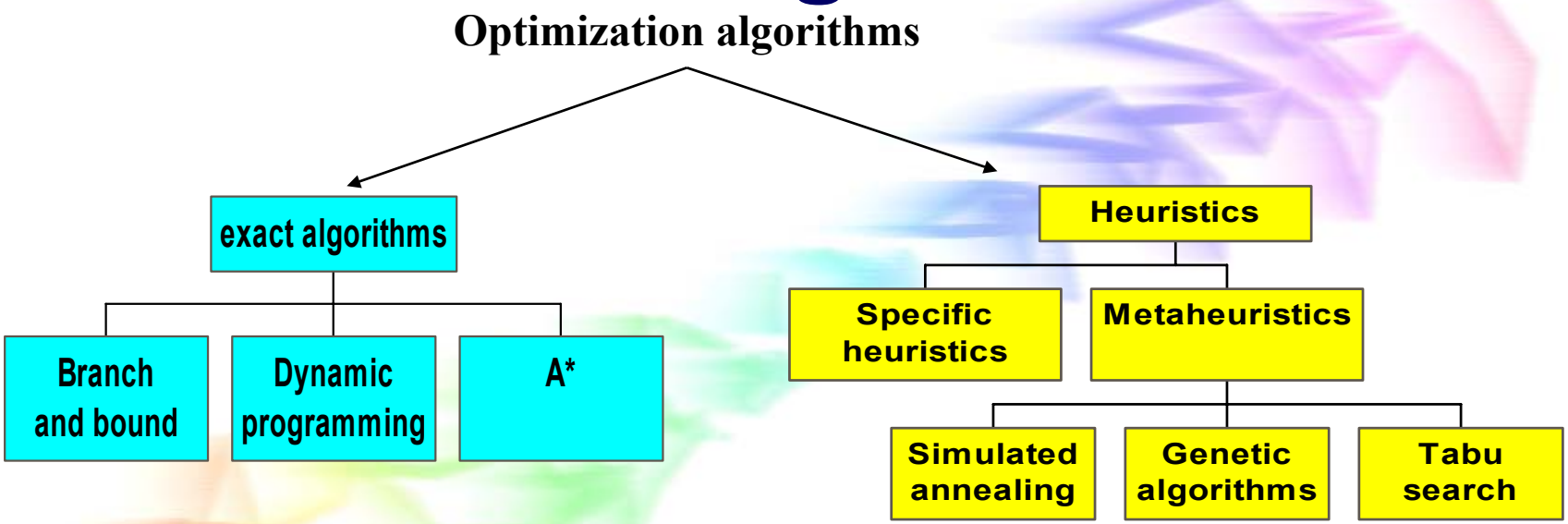
# Plan de la présentation

- Motivations
- Design issues
- Implementation issues
- A grammar for extended hybrid schemes
- Examples
- Conclusions and Perspectives

# Motivations

- Considerable interest in recent years (combinatorial optimization, ...).
- Best found solutions for academic and real applications are obtained by hybrid algorithms (QAP, JSP, TSP, …).
- A wide variety of hybrid approches in the literature.
- Common terminology and classification mechanisms (Unifying view).
- Comparison of hybrid algorithms in a qualitative way.
- Areas in need of future work.

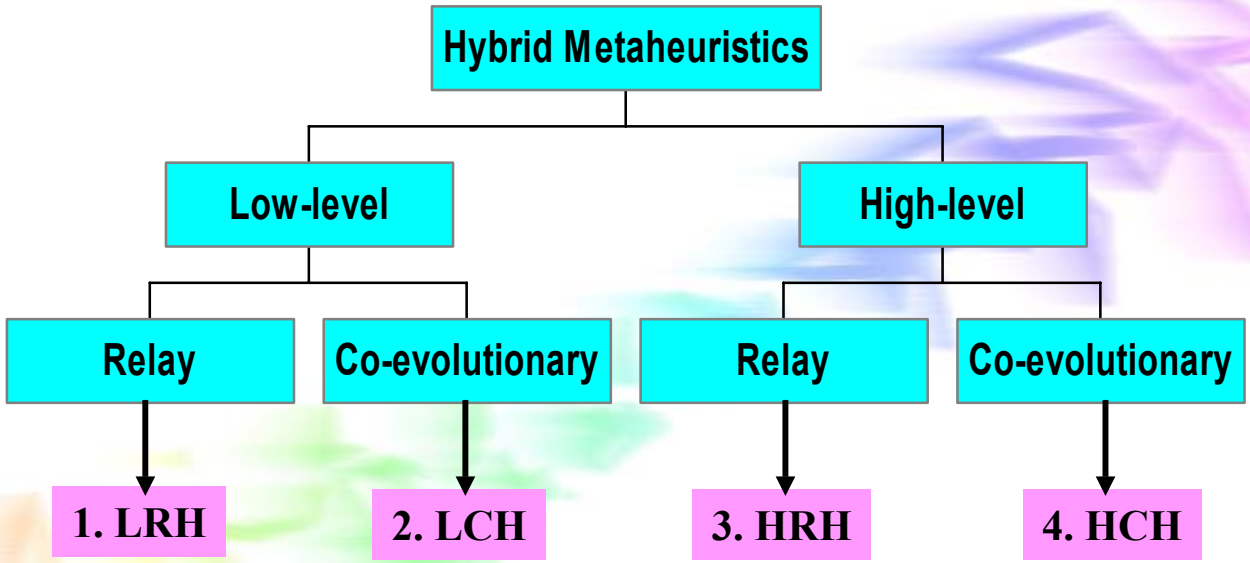
# Search algorithms




Metaheuristics :

- unique solution : simulated anneling, tabu search, ...
- population : genetic algorithms, scatter search, ant colonies, ...

# Design issues (Hierarchical)

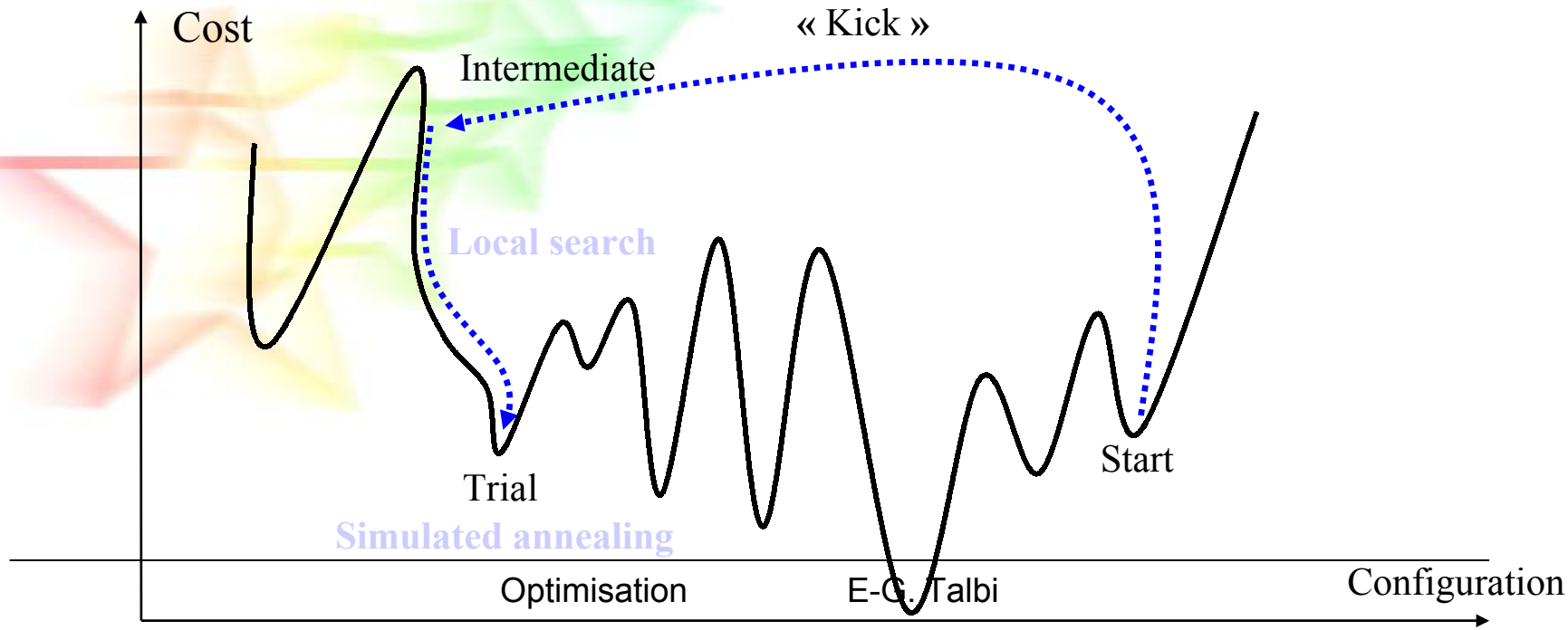



- Low-level / High-level
  - Low-level : Functional composition of a single method.
  - High-level : Different methods are self-contained.
- Relay / Co-evolutionary
  - Relay : Pipeline fashion.
  - Co-evolutionary : Parallel cooperating agents.

# Low-Level Relay Hybrid

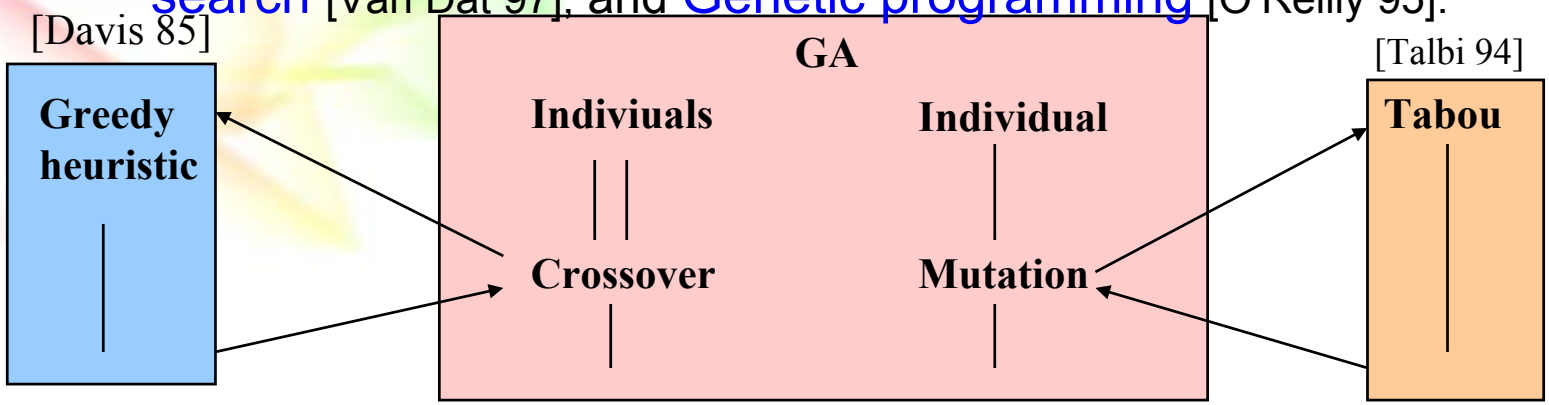
- LRH : Hybrids in which a given metaheuristic is embedded into a single-solution metaheuristic

- Example : Local search embedded in Simulated annealing (Markov chain explores only local optima) [Martin et Otto 92]



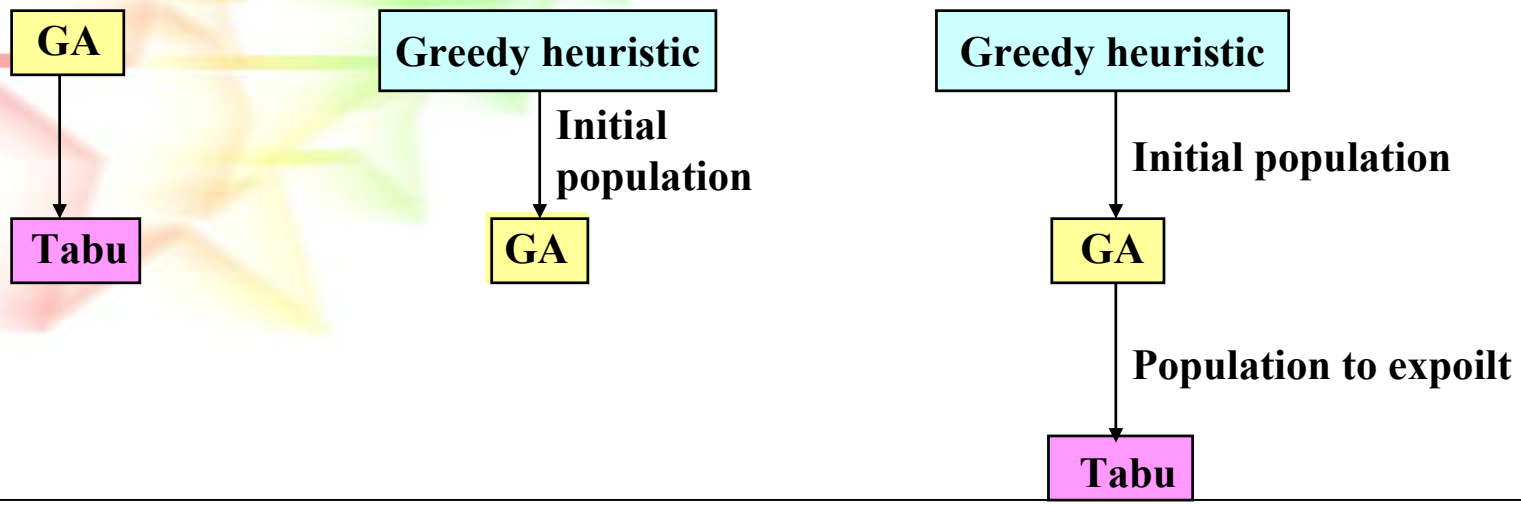
# Low-Level Co-evolutionary Hybrid

- LCH : A metaheuristic is embedded into a population-based metaheuristic.
- Examples :
  - Greedy crossover [Grefenstette 85], local search for mutation [Fleurant 94, Chu 97] (lamarckian, baldwin, …) in GAs.
  - Local search in Ant colonies [Taillard 97, Talbi et al. 99], Scatter search [Van Dat 97], and Genetic programming [O'Reilly 95].



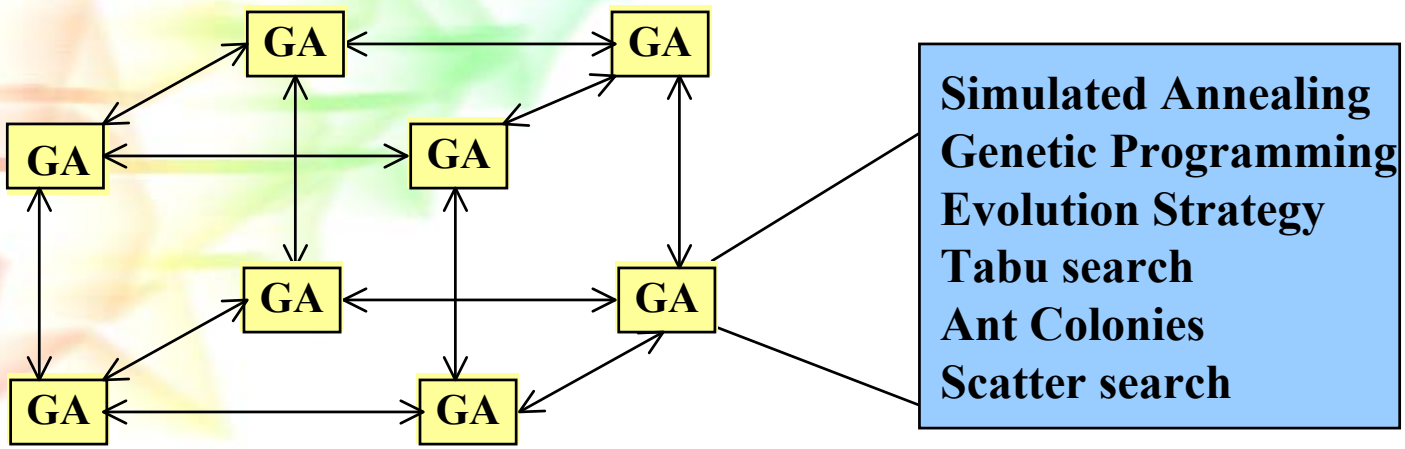
# High-Level Relay Hybrid

- HRH : Self-contained metaheuristics are executed in a sequence.
- Example :
  - GA + Simulated annealing [Mahfoud 95]. GA + Tabu search [Talbi 94].
  - ES + Local search [Nissen 94]. Simulated annealing + GA [Lin 91]



# High-Level Co-evolutionary Hybrid

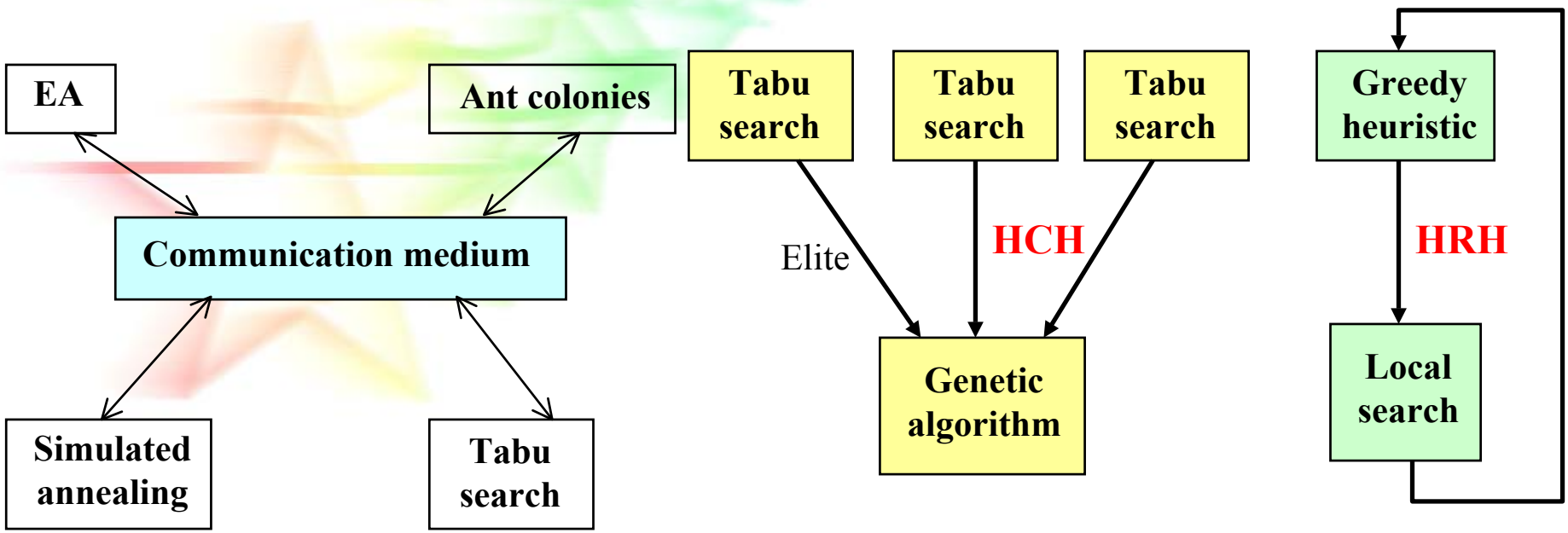
- HCH : Parallel cooperating self-contained algorithms.
- Example :
  - Island model for GAs [Tanese 87], Genetic programmig [Koza 95], Evolution strategies [Voigt 90].
  - Tabu search [Rego 96], Simulated annealing [De Falco 95], Ant colonies [Mariano 98], Scatter search [Van Dat et al. 99].




**Topology, Frequency of migration, which individuals to migrate ?, Which individuals to replace ?, ...**

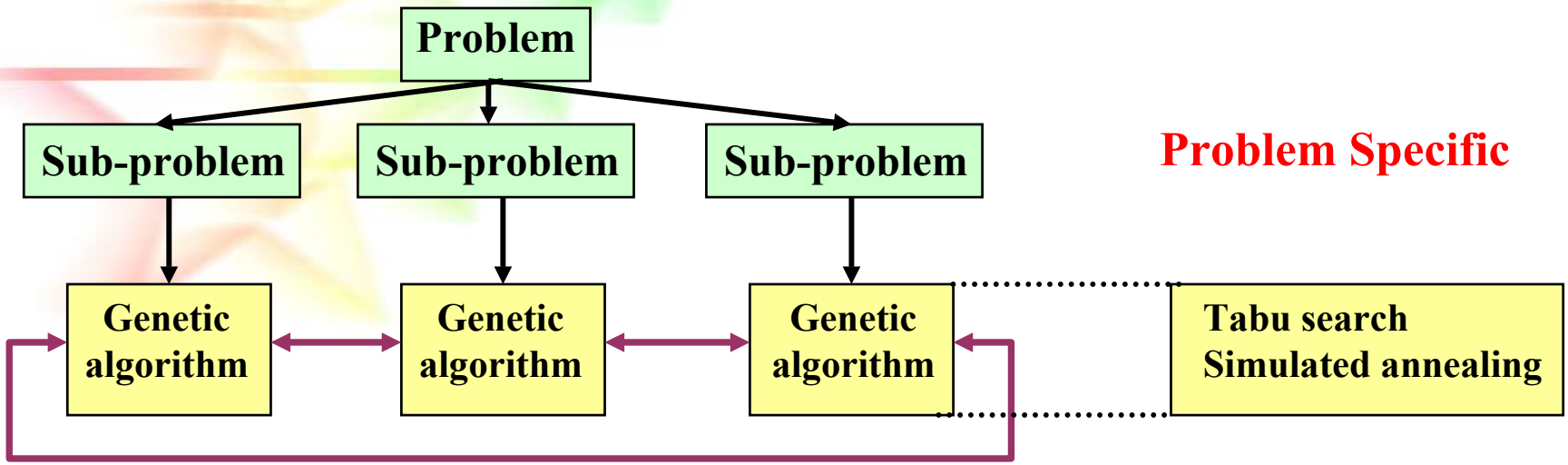
# Homogeneous/Heterogeneous (Flat)

- Heterogeneous : Different metaheuristics.
- Examples :
  - HCH : GAs and Tabu search [Crainic et al. 97].
  - HRH : GRASP (Greedy + Local search) [Feo and Resende 94].



# Global versus Partial

- Partial : Problem is decomposed in sub-problems. Each algorithm is dedicated to one sub-problem.
- Examples :
  - HCH : Tabu search (Vehicle routing) [Taillard 93], Simulated annealing (placement of macro-cells) [Casoto et al. 86].
  - HCH : GA (Job-shop scheduling) [Husbands et al. 90].

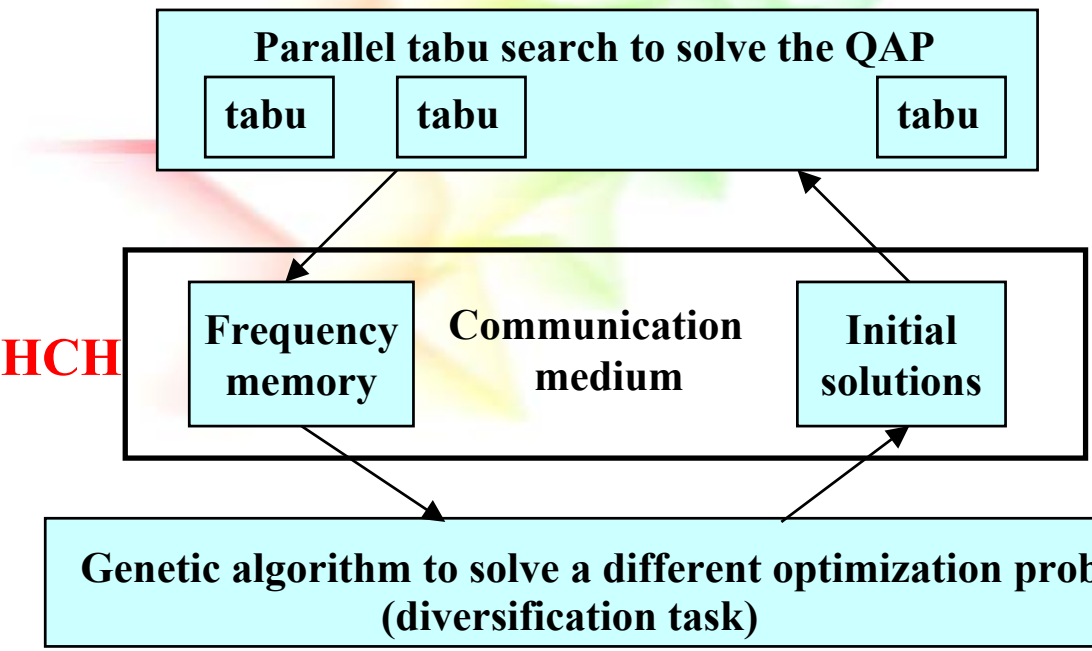


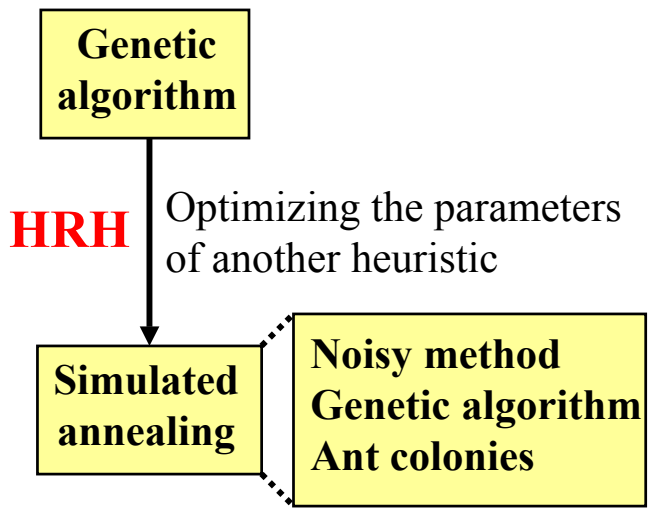



**Synchronisation : Build a global viable solution**

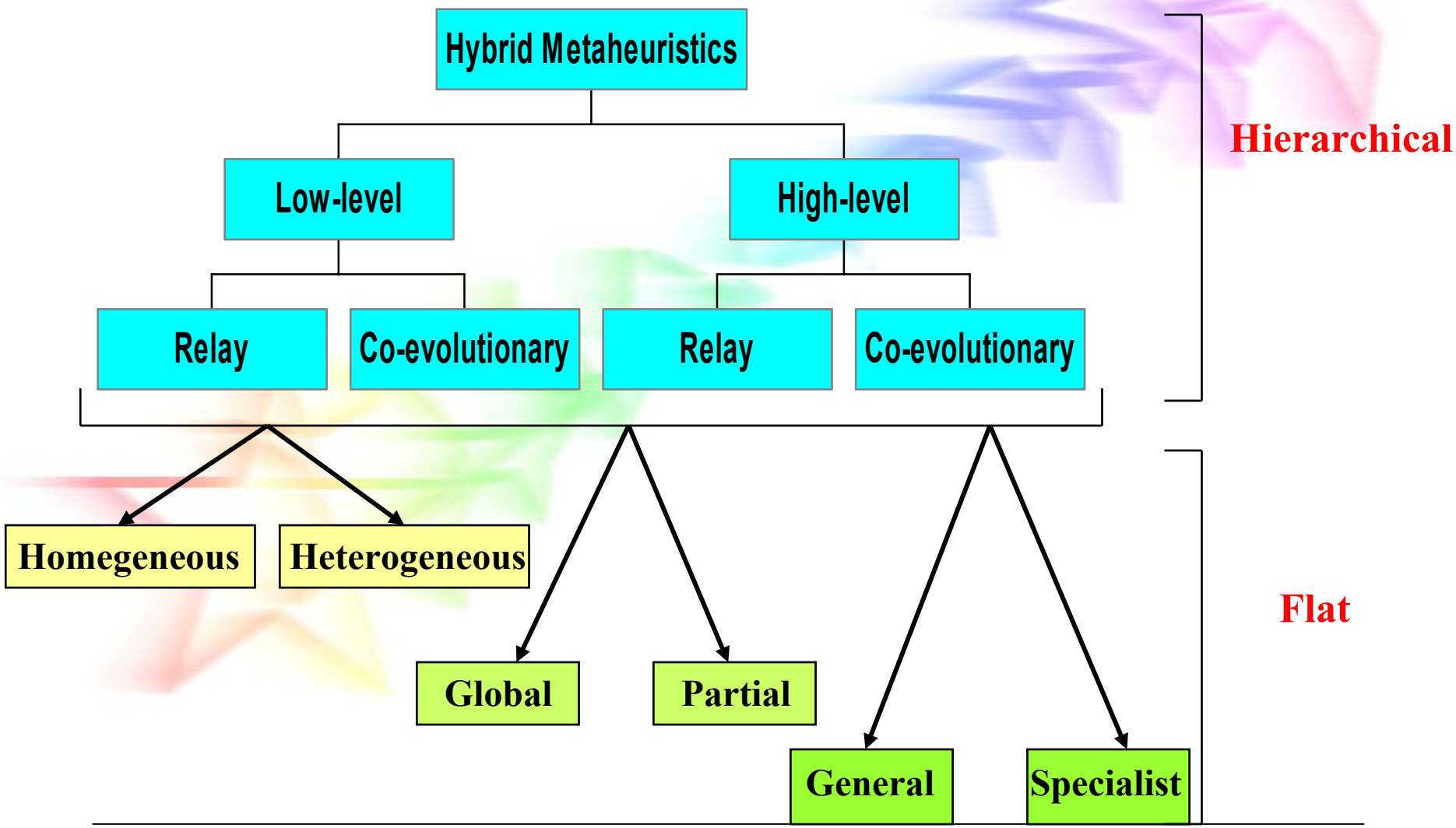
# Specialist versus General

- Specialist : Algorithms which solve different problems.
- Examples :
  - **HCH** : GA + Tabu (QAP) [Talbi et al. 98]
  - **HRH** : GA + (SA | NM | GA | AC) [Krueger 93][Shahookar et al. 90][Abbattista et al. 95]



# Design issues



# Specific / General computers

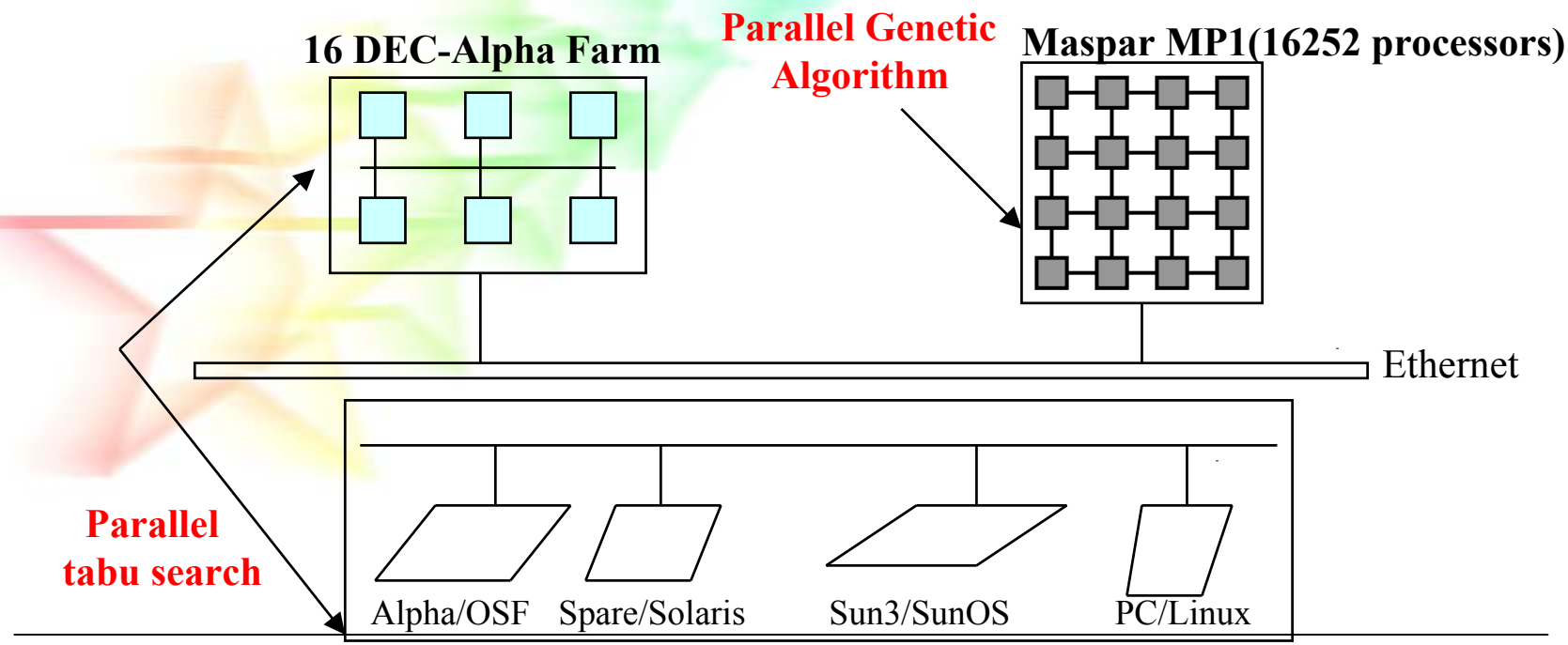
- Specialist : Application specific computers.
  - Tailored for a particular problem.
  - Much higher efficiency and lower cost.
- Examples :
  - Programmable logic devices : Xilinx FPGA, APTIX interconnexion chips, ...
  - SA [Abramson 92] : Speedup 37 times over an IBM RS6000 (TSP)
  - GA [Salami and Cain 96].
  - Template for local search (TS, SA, …) [Abramson et al. 97].

# Sequential versus Parallel

- SIMD / MIMD :
  - SIMD : Single Instruction stream, Multiple Data stream.
    - Processors execute the same program.
    - Efficient for regular computations and regular data transfers.
    - Example : HCH (Tabu Search) on MasPar MPP-1 [De Falco et al. 96]
  - MIMD : Multiple Instruction stream, Multiple Data stream.
    - HCH (Tabu search) [Fiechter 94]
    - HCH (Simulated annealing) [Sloot et al. 95]
    - HCH (Genetic algorithms) [Muhlenbein 88, Talbi 89]
- Shared memory / Distributed memory :
  - Shared memory : Simplicity
  - Distributed memory : More flexible and fault-tolerant

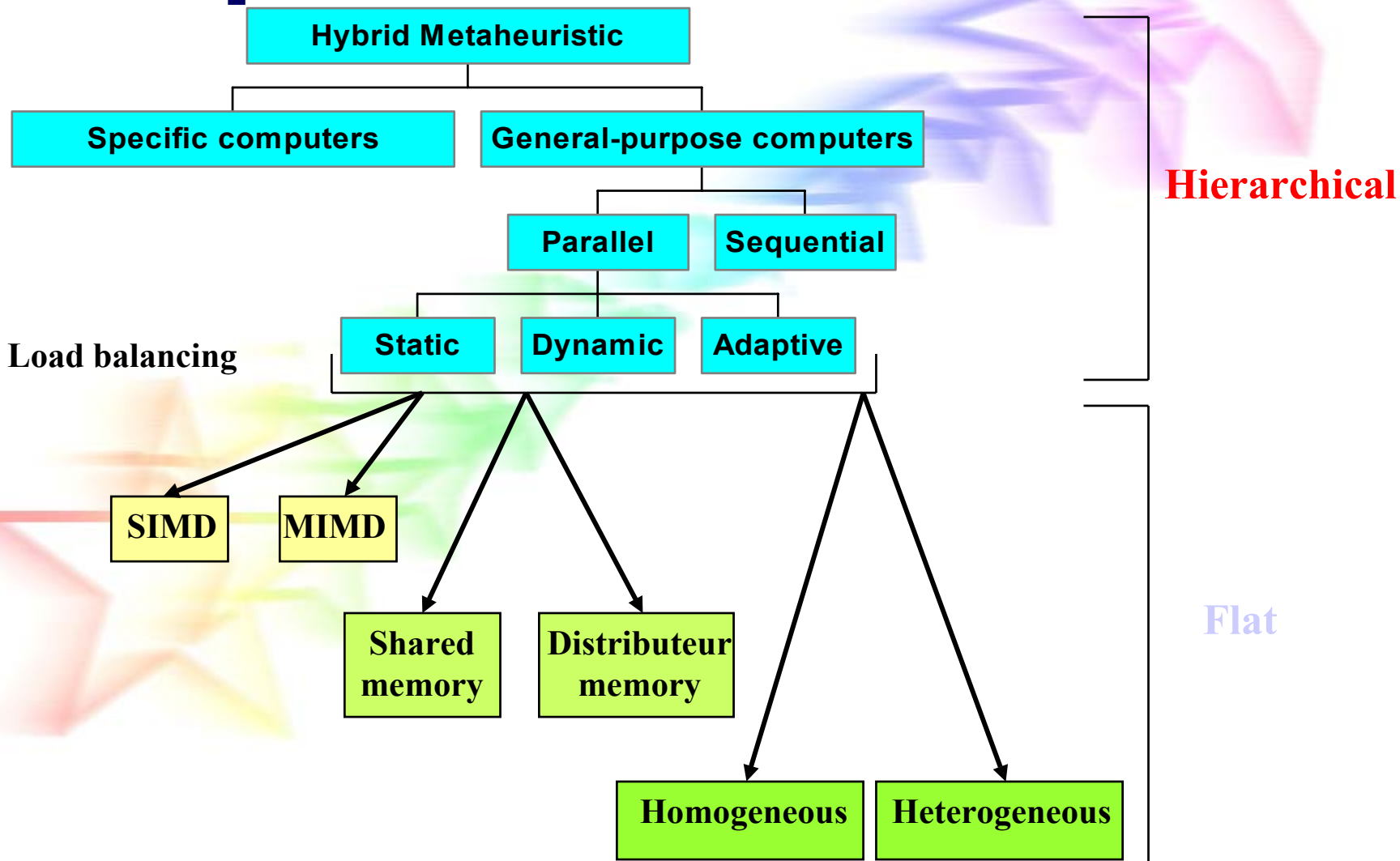
# Sequential versus Parallel

- Homogeneous / Heterogeneous :
  - Homogeneous : Massively parallel machines (Cray T3E, IBM SP/2, …) and clusters of processors (Myrinet PC network, …).
  - Heterogeneous : Network of workstations.
- Example : [Talbi et al. 96]




Large network of 128 heterogeneous workstations

# Implementation Issues



# Grammar for extended schemes

< *hybrid metaheuristic* > ⟶ < *design-issues* > < *implementation-issue* >

< *design-issues* > ⟶ < *hierarchical* > < *flat* >

< *hierarchical* > ⟶ < LRH >|< LCH >|< HRH >|< HCH >

< *LRH* > ⟶ LRH (< *metaheuristic* >( < *metaheuristic* >))

< *LCH* > ⟶ LCH (< *metaheuristic* >( < *metaheuristic* >))

< *HRH* > ⟶ (< *metaheuristic* >+ < *metaheuristic* >)

< *HCH* > ⟶ HCH (< *metaheuristic*>)

< *HCH* > ⟶ HCH (< *metaheuristic* >,< *metaheurisic* >)

< *flat* > ⟶ (< *nature* >,< *optimization* >,< *funcion* >)

< *nature* > ⟶ homogeneous | heterogeneous

< *optimization* > ⟶ global | partial

< *function* > ⟶ general | specialist

< *implementation-issue* > ⟶ sequential | parallel < *scheduling* >

< *scheduling* > ⟶ static | dynamic | adaptive

< *metaheuristic* > ⟶ LS | TS | SA | GA | ES | GP | NN

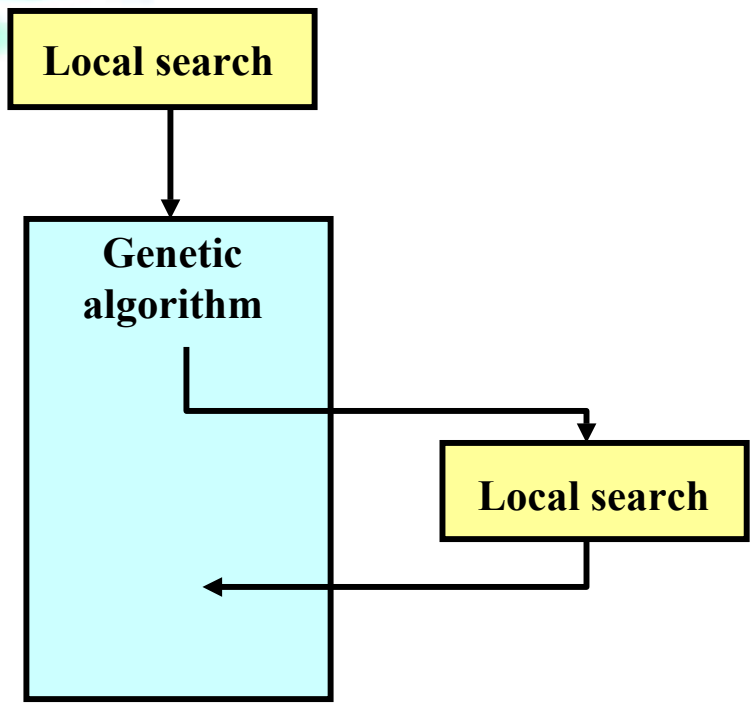
< *metaheuristic* > ⟶ GH | AC | SS | NM | CLP| < *hybrid-metaheuristic* >

# Example 1

**HRH(LS + LCH(GA (LS) ) (het,glo,gen) (sequential)) (het,glo,gen) (sequential)**

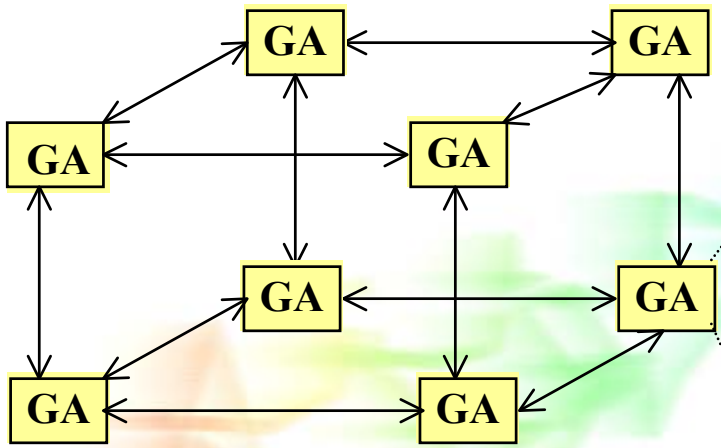
**[Boese et al. 94]**

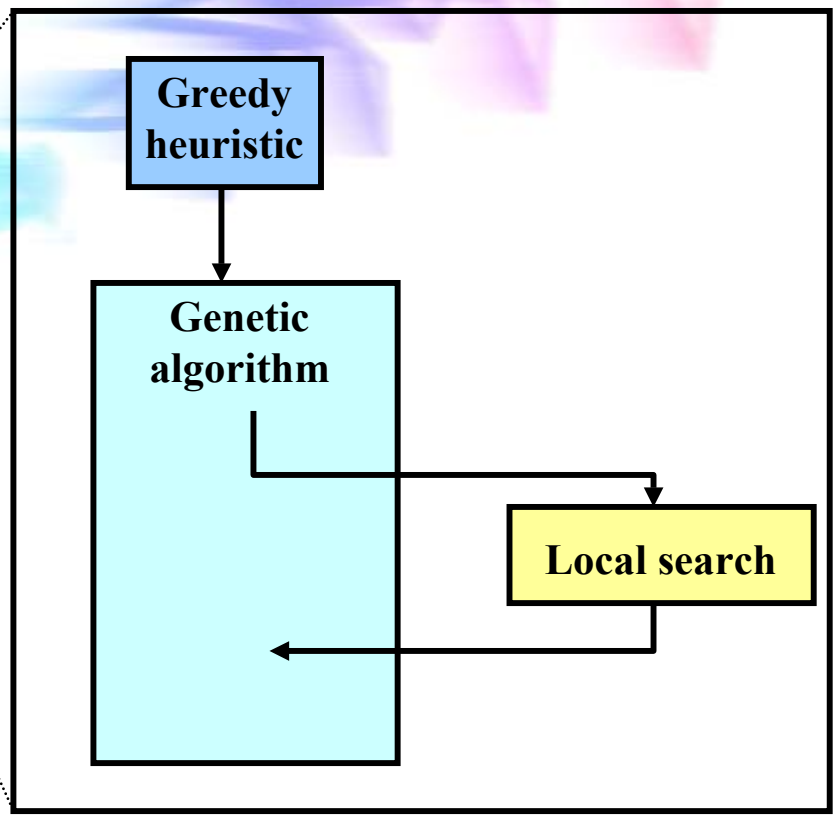
**Traveling Salesman Probem**
**Graph Partitioning Problem**



# Example 2

## HCH (HRH (GH + LCH(GA(LS))))




**[Levine 94]**

**Set Partitioning Problem**
**Airline crew scheduling**

**Parallel static implementation**

**[Braun 90]**

**Traveling salesman problem**

**Sequential implementation**

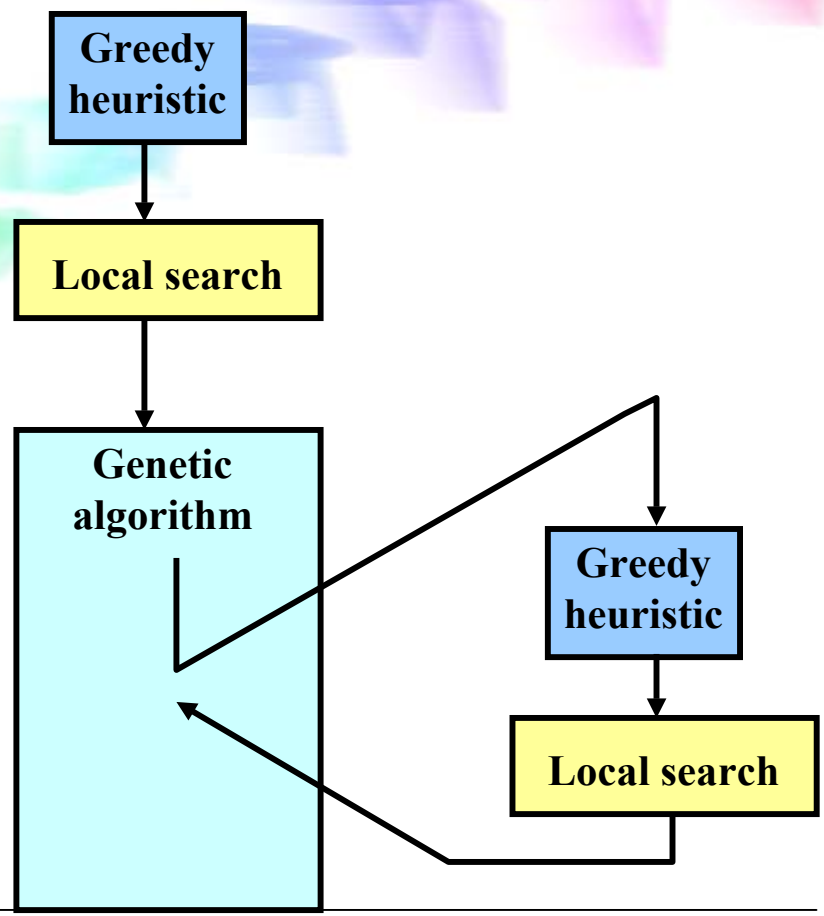
# Example 3

## HRH (GH + LS + LCH(GA (HRH(GH+LS))))

[Freisleben and Merz 96]

Traveling Salesman Problem

Sequential implementation



# Conclusions and perspectives

- Usefulness of the taxonomy : More than 129 annoted references.
- Applicable to exact algorithms and specific heuristics.
- Design of efficient hybrids (landscapes, …)
- Hybrid metaheuristics for multi-objective optimization problems

# Optimisation Combinatoire Multi-objectifs : Etat de l'art

# Optimisation multi-objectifs

- Nombreux secteurs de l'industrie concernés (**Télécommunications**, Transport, Environnement, Mécanique, Aéronautique, ...).

- Racines dans le 19ième siècle dans les travaux en **économie** de Edgeworth et Pareto (Sciences de l 'ingénieur, Management).

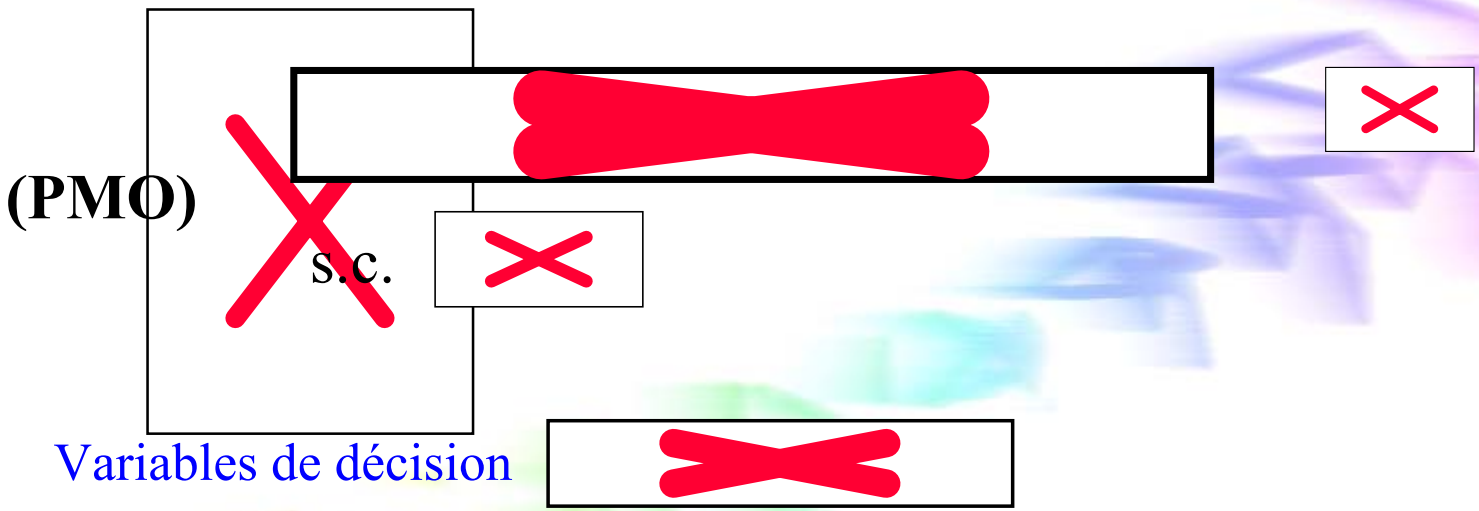
- Optimisation multi-critères linéaire ou non-linéaire en variables continues [Steuer 86, White 90].

  **Optimisation combinatoire** multi-critères.

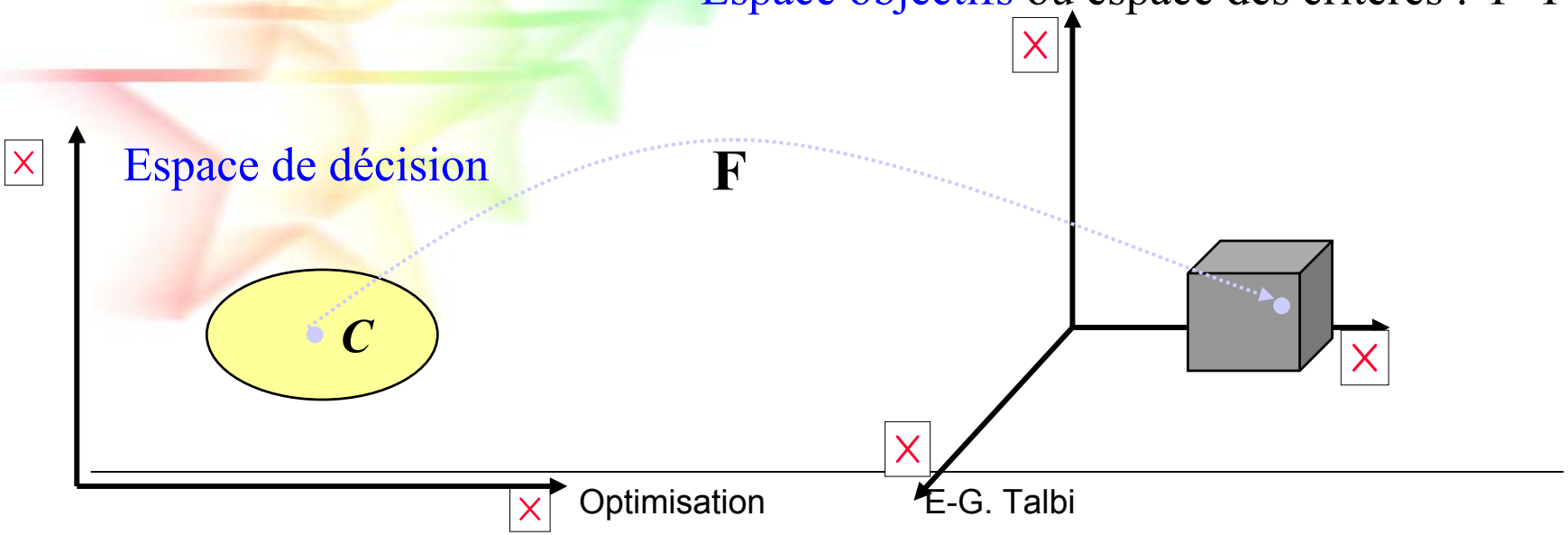
# Plan de la présentation

- Optimisation multi-objectifs : Définitions, Problèmes
- Classification des méthodes de résolution (métaheuristiques associées)
- Evaluation des performances et paysages
- Conclusions
- Axes de recherche future : Algorithmes parallèles et hybrides, ...

# Optimisation multi-objectifs




Espace objectifs ou espace des critères : Y=F(C)



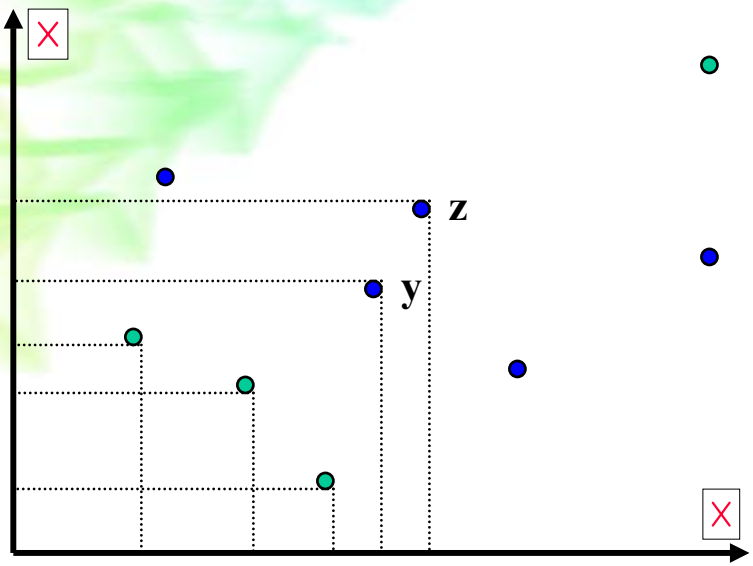
# Définitions

- **Dominance**

  y domine z ssi [redacted]

- **Pareto optimalité**

  Une solution [redacted] est **Pareto optimale** ssi il n 'existe pas une solution [redacted] tel que [redacted] domine [redacted]

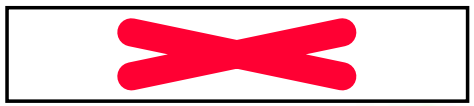



- **Solution Pareto optimale** (admissible, efficace, non inférieure, non dominée)
- **Solution réalisable dominée**

**PO : ensemble exact des solutions Pareto optimales**

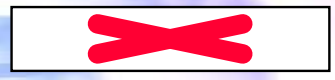
# Définitions

- Solutions supportées et non supportées
- Vecteur idéal y* et vecteur de référence z*



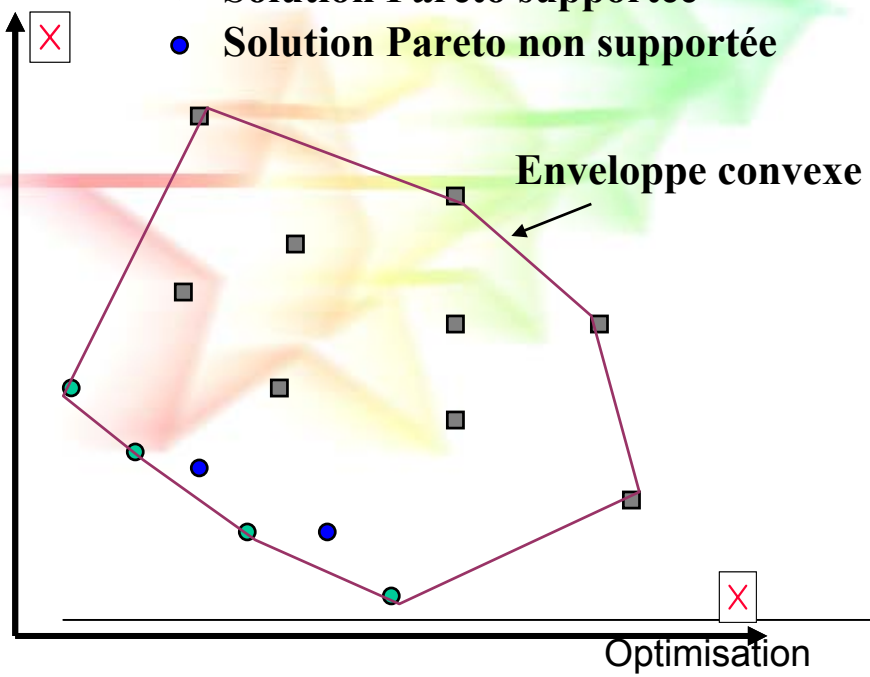
**Voisinage N :** opérateur de recherche locale

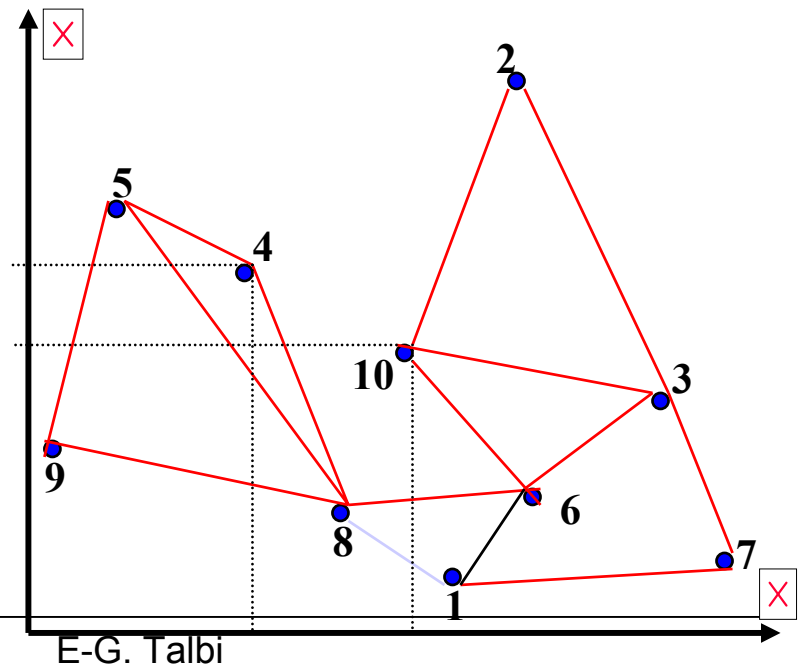
- Pareto localement optimale



**1, 8, 9 : Pareto optimale**
**4, 10 : Pareto localement optimale**

**voisinage**



# Difficultés des PMO

- Définition de l ’optimalité : relation d ’ordre partiel, choix final dépend du décideur, environnement dynamique, …
- Nombre de solutions Pareto croît en fonction de la taille des problèmes et le nombre de critères utilisés.
- Pour les PMO non convexes, les solutions Pareto sont localisées dans les frontières et à l ’intérieur de l ’enveloppe convexe de *C*.

# Coopération solveur-décideur

- **A priori :** Préférences ⟶ Recherche (fonction d 'utilité)

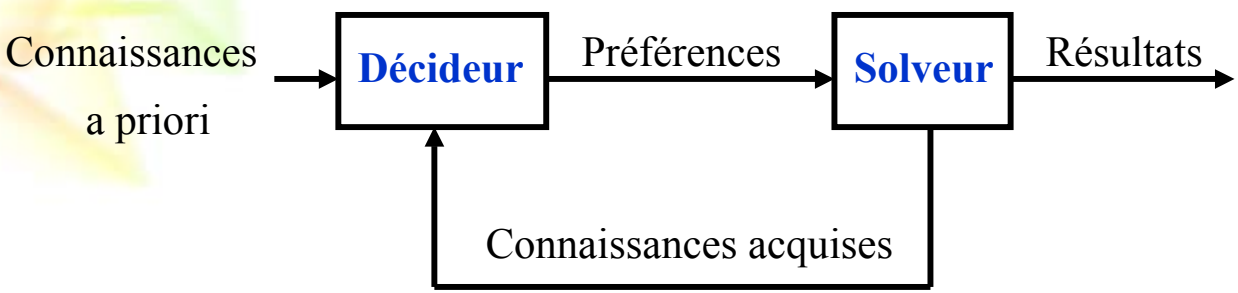
  **Connaissances a priori du problème.**

- **A posteriori :** Recherche ⟶ Préférences

  **Cardinalité de l 'ensemble PO réduite.**

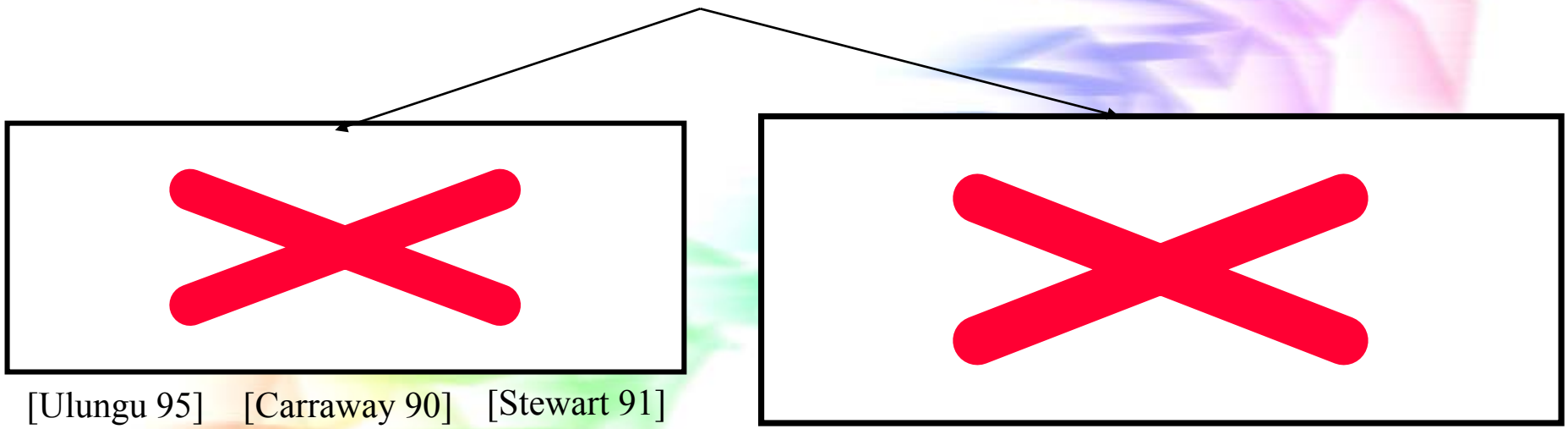
- **Interactive :** Recherche ⟷ Préférences

  **Aspect aide à la décision n 'est pas adressé.**



# Algorithmes de recherche

**Algorithmes de résolution**



[Ulungu 95] [Carraway 90] [Stewart 91]

Algorithmes exacts : problèmes bi-critères de petites tailles

Métaheuristiques :

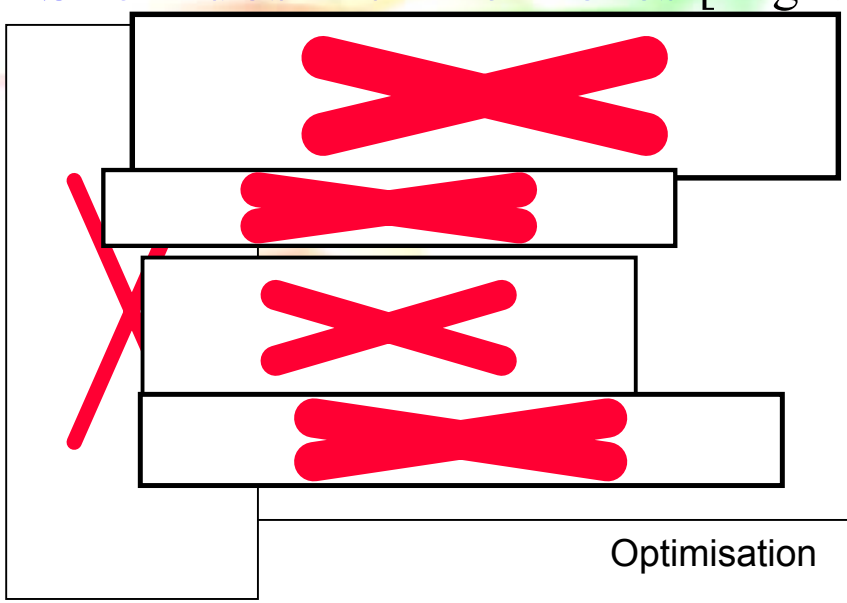
- solution unique : recuit simulé, recherche tabou, ...
- population : algorithmes génétiques, recherche dispersée, colonies de fourmis, ...

**PO* : Approximation de PO**

# PMO académiques

- Ordonnancement [Sayin et al. 99]
- Cheminement [Warburton 87], Arbre recouvrant [Zhou et Gen 99]
- Voyageur de commerce [Serafini 92], Affectation [Teghem 95]
- Routage de véhicules [Park et Koelling 89], ...

**Sac à dos multi-critères** [Teghem et al. 97]



$x_j$ = 1 si l 'élément *j* est dans le sac
0 sinon

$p_j$ : poids de l 'élément j

$u_{ij}$ : utilité de l 'élément j / critère i

# PMO industriels

- **Télécommunications** : Design d 'antennes [Sandlin et al. 97], affectation de fréquences [Dahl et al. 95], ...
- **Aéronautique** : ailes d 'avions [Obayashi 98], moteurs [Fujita 98], ...
- **Environnement** : gestion de la qualité de l 'air [Loughlin 98], distribution de l 'eau, …
- **Transport** : tracé autoroutier, gestion de containers [Tamaki 96], …
- **Finances, Productique, Robotique, ...**

## Design de réseaux de radiocommunications mobiles

[P. Reininger et A. Caminada, CNET]

**Objectifs et/ou contraintes** :

- Min (Nombre de sites candidats utilisés)
- Min (Interférence)
- Min (Overhead)
- Max (Trafic)
- Couverture, Handover, Connectivité, ...

# Classification des méthodes

- Transformation de PMO en uni-objectif
  - Méthode d'agrégation
  - Méthode *E*-contrainte
  - Programmation par but

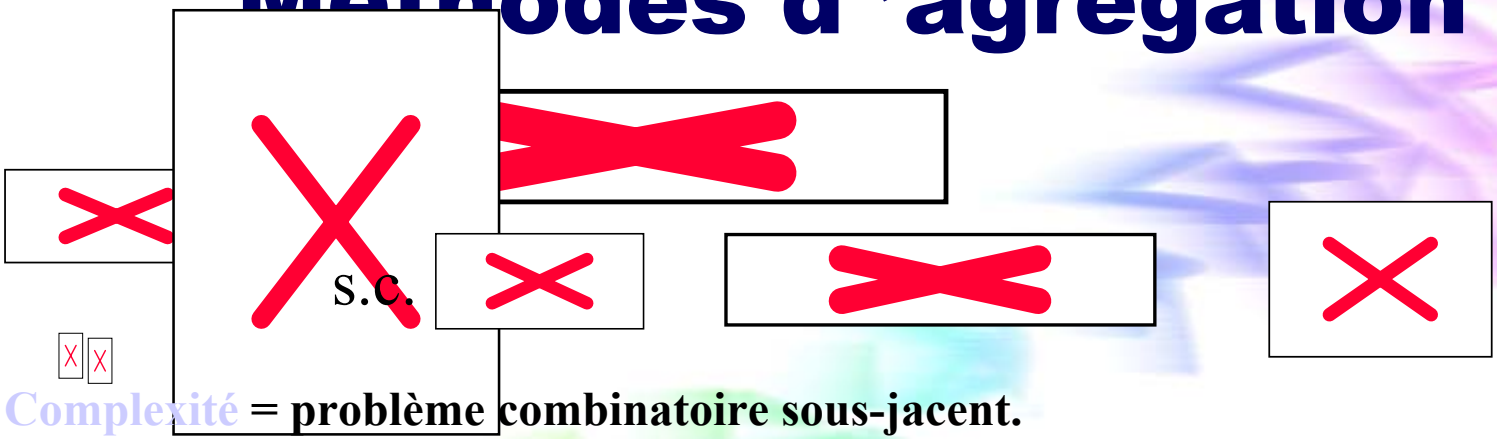
- Approches non Pareto

  Traitent séparément les différents objectifs

- Approches Pareto

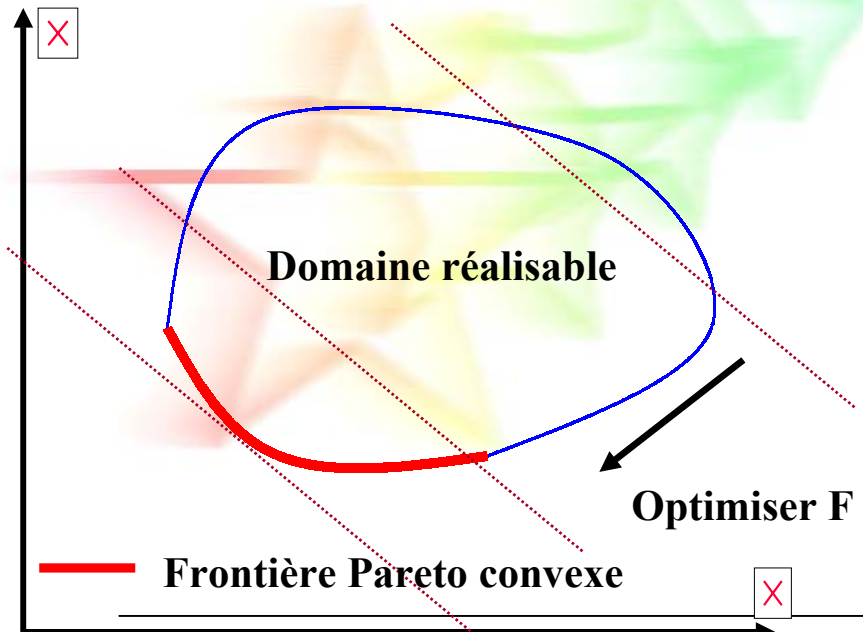
  Utilisent la notion de dominance

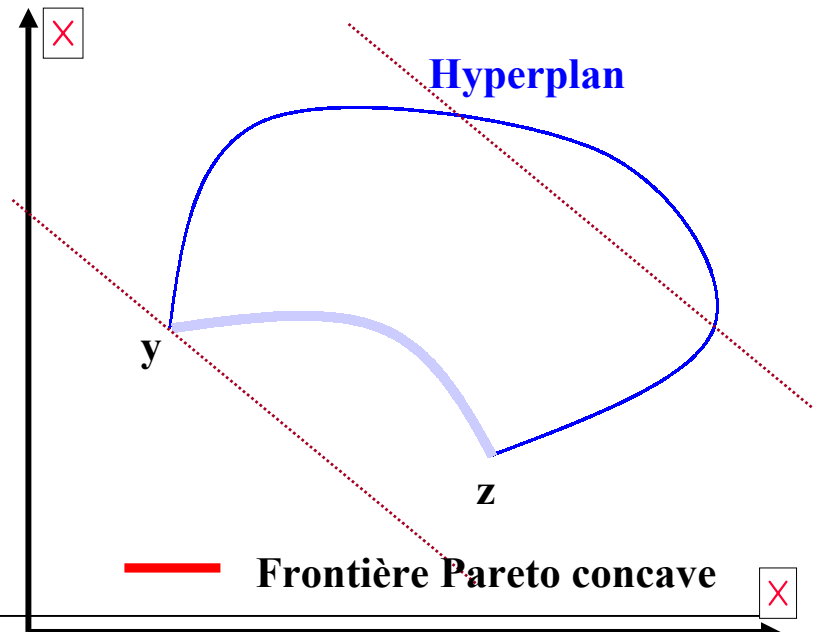
# Méthodes d ’agrégation



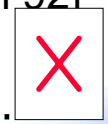

Complexité = problème combinatoire sous-jacent.

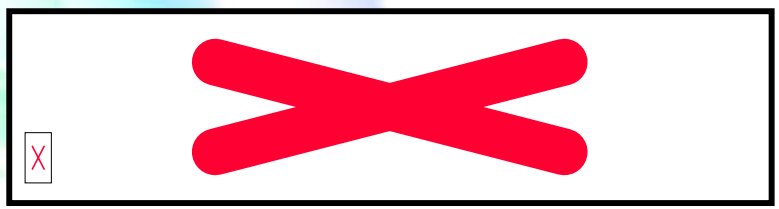
Ex : Polynomial = flot, cheminement, …  NP-complet = routage, affectation, ...



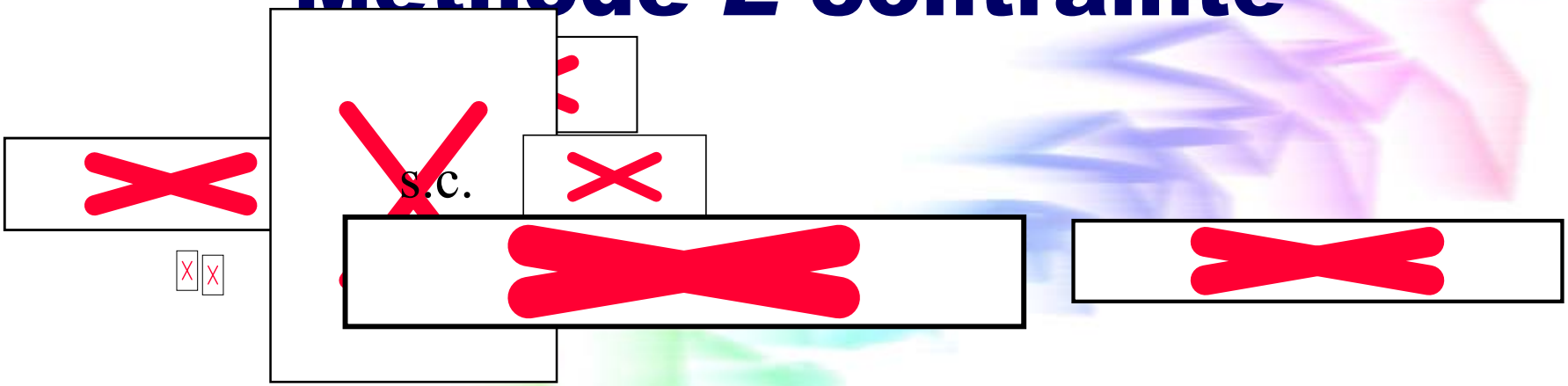
# Métaheuristiques

- Algorithmes génétiques [Hajela et Lin 92]
  - Codage d'un individu = Solution + 
  - But = Générer une variété de solutions Pareto
- Recuit simulé [Serafini 92]
  - Fonction d 'acceptation



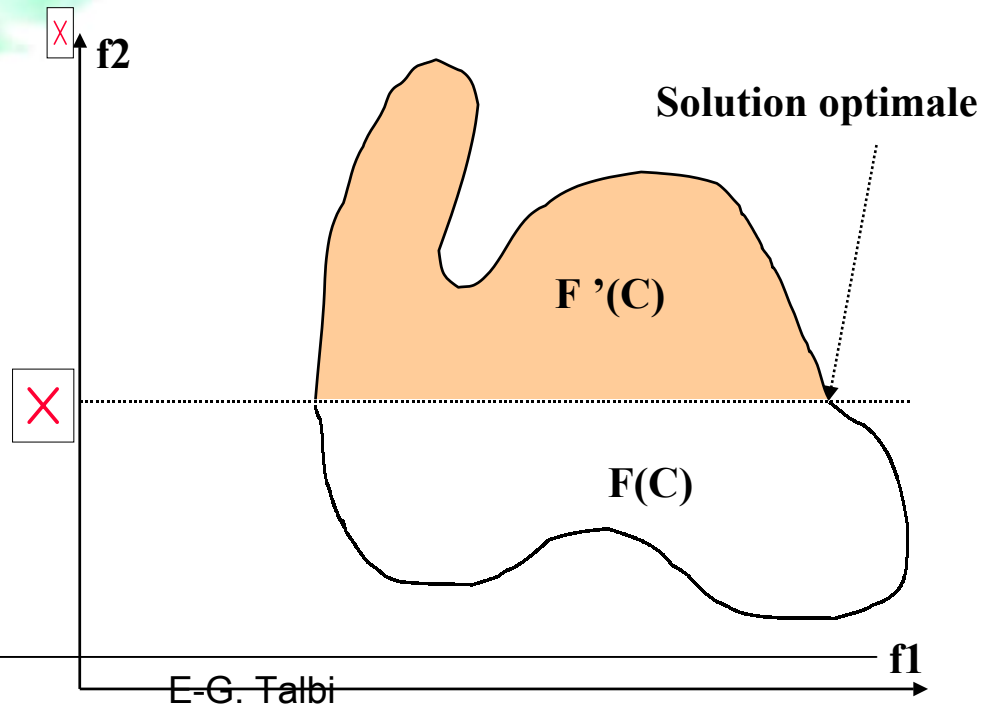
- Recherche tabou [Dahl et al. 95]
  - Poids = priorité de l 'objectif
- Métaheuristiques hybrides [Talbi 98]
  - Algorithme glouton + Recuit simulé [Tuyttens 98]
  - Algorithme génétique(Recherche locale) [Ishibuchi et Murata 98]
    - Sélection avec des poids différents
    - Recherche locale sur l 'individu produit (même poids)

# Méthode *E*-contrainte



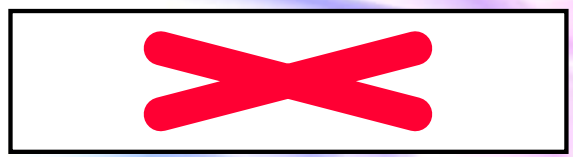

- Variation de

- F(C) = espace objectifs PMO
- F ’(C) = espace objectifs du problème transformé



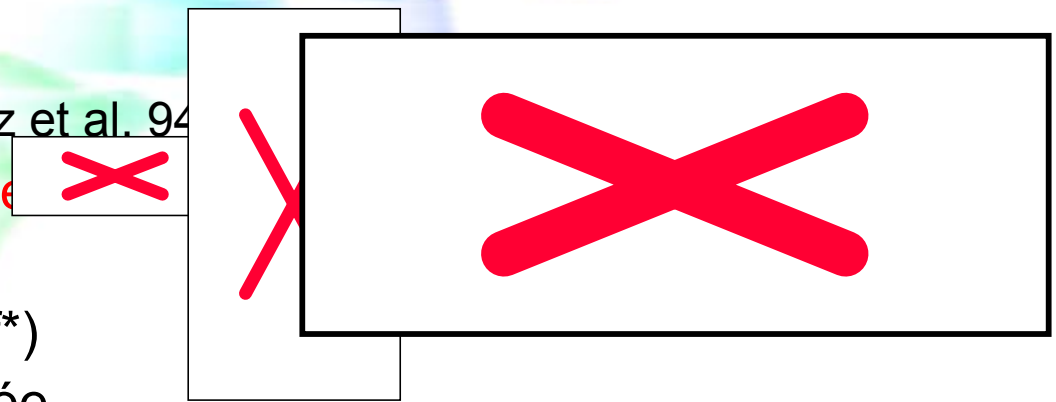
# Métaheuristiques

- Algorithmes génétiques
  - Individu = [Loughlin 98]



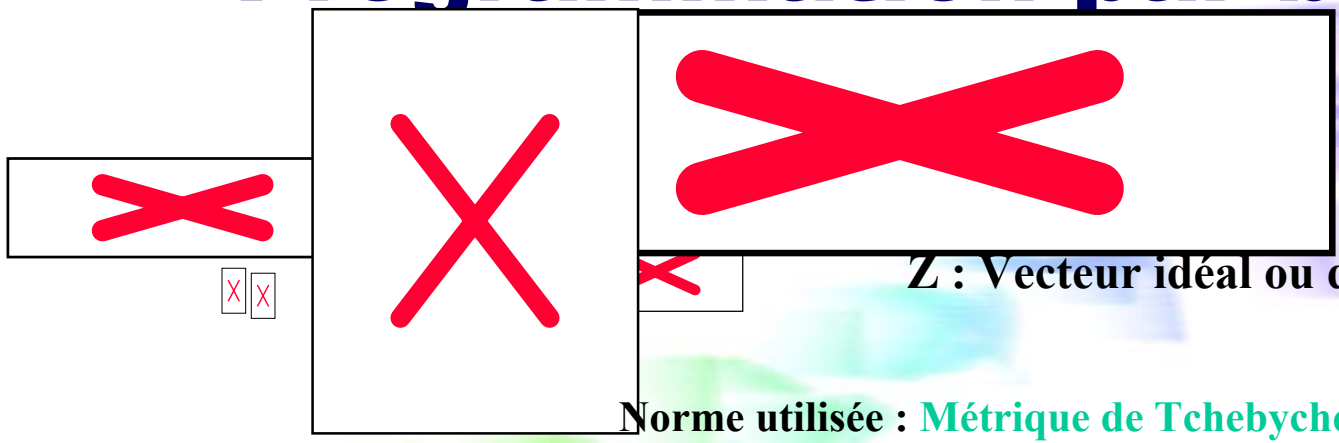
k : taille de la population

- Recherche tabou [Hertz et al. 94
  - Suite de sous-problème
  - Ordre de priorité
  - Seuil (f ') = Optimum (f*)
    + Déterioration acceptée



- Métaheuristiques hybrides [Quagliarella et al. 97]
  - Algorithme génétique + Recherche locale

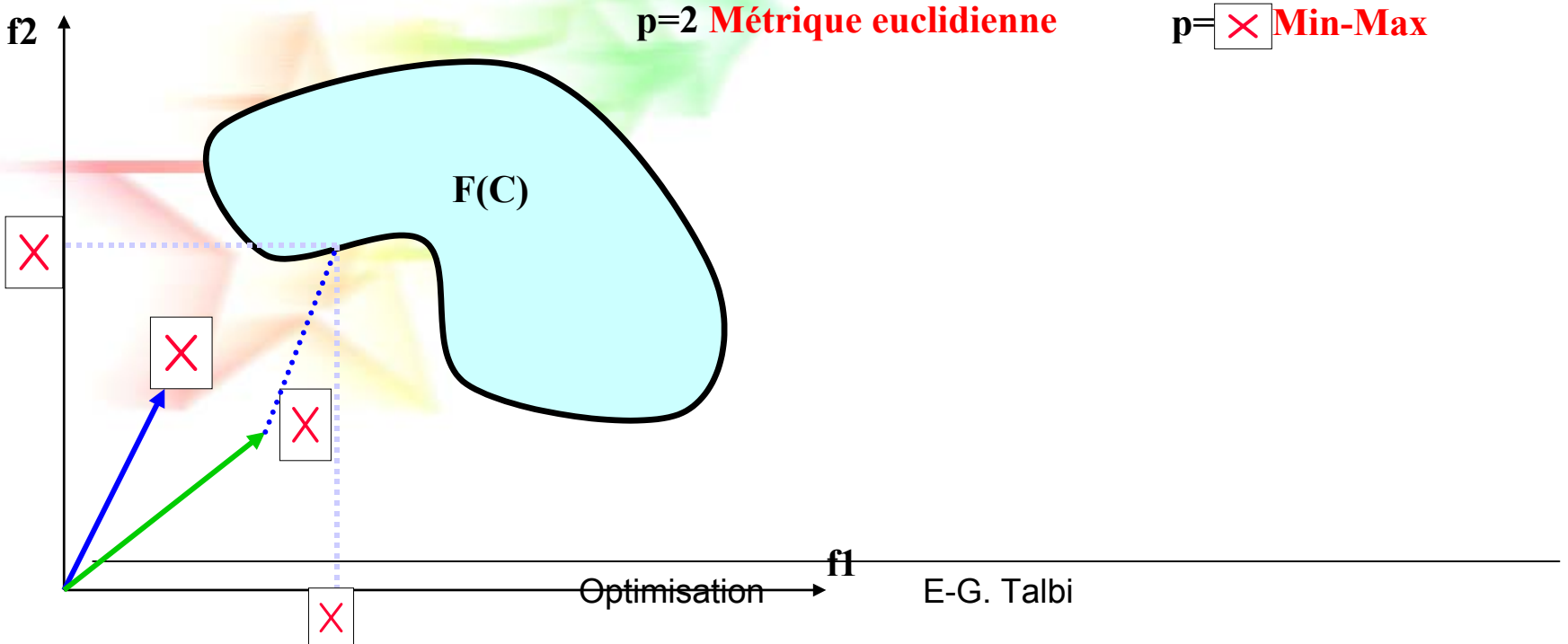
# Programmation par but



Z : Vecteur idéal ou de référence

Norme utilisée : Métrique de Tchebycheff métrique

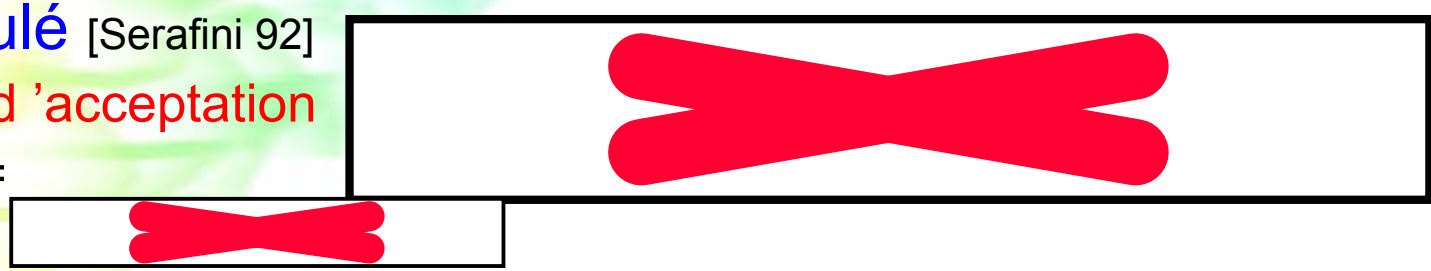
p=2 Métrique euclidienne p= Min-Max



# Métaheuristiques

- Algorithmes génétiques
  - Fonction min-max, AG parallèle avec différents poids [Coello 98]
  - Règles floues dans l ’évaluation de F [Reardon 98]

- Recuit simulé [Serafini 92]
  - Fonction d ’acceptation
  - Itération =



- Recherche tabou [Ga[illegible]bleux 96]
  - Meilleur voisin = Meil[illegible] compromis non tabou
  - Compromis = Norme de Tchebycheff par rapport au vecteur idéal
  - Mémorisation des M meilleures solutions

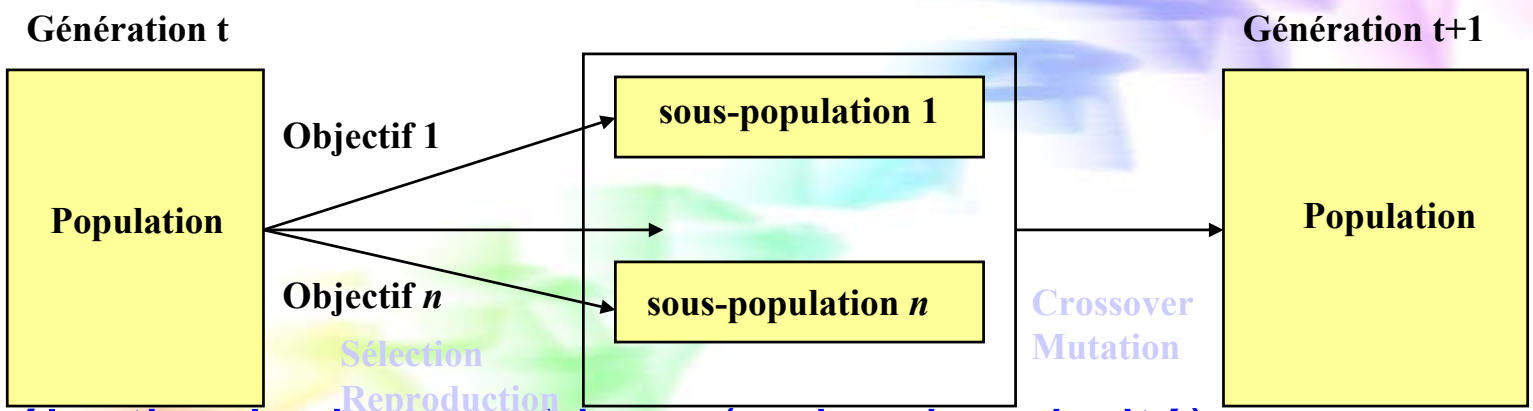
# Analyse critique

- Espace de recherche <> Problème initial. Le problème perd ses éventuelles propriétés.
- Une exécution produit une seule solution.
- Connaissances a priori (détermination des paramètres).
- Sensibles au paysage de la frontière Pareto (convexité, discontinuité, …), et à différents paramètres (poids, contraintes, buts, …).
- Objectifs bruités ou données incertaines.
- Coût associé aux algorithmes (exécution multiple).

# Approches non Pareto

**Traitent séparément les différents objectifs non commensurables.**

- Sélection parallèle (VEGA) [Schaffer 85]




- Sélection lexicographique (ordre de priorité)
  - Algorithmes génétiques [Fourman 85]
  - Stratégies évolutionistes [Kursawe 91]
- Reproduction multi-sexuelle [Lis et Eiben 96]
  - Plusieurs classes. Une classe = Un objectif
  - Reproduction (crossover) sur plusieurs individus



Tendance à ignorer les solutions compromis

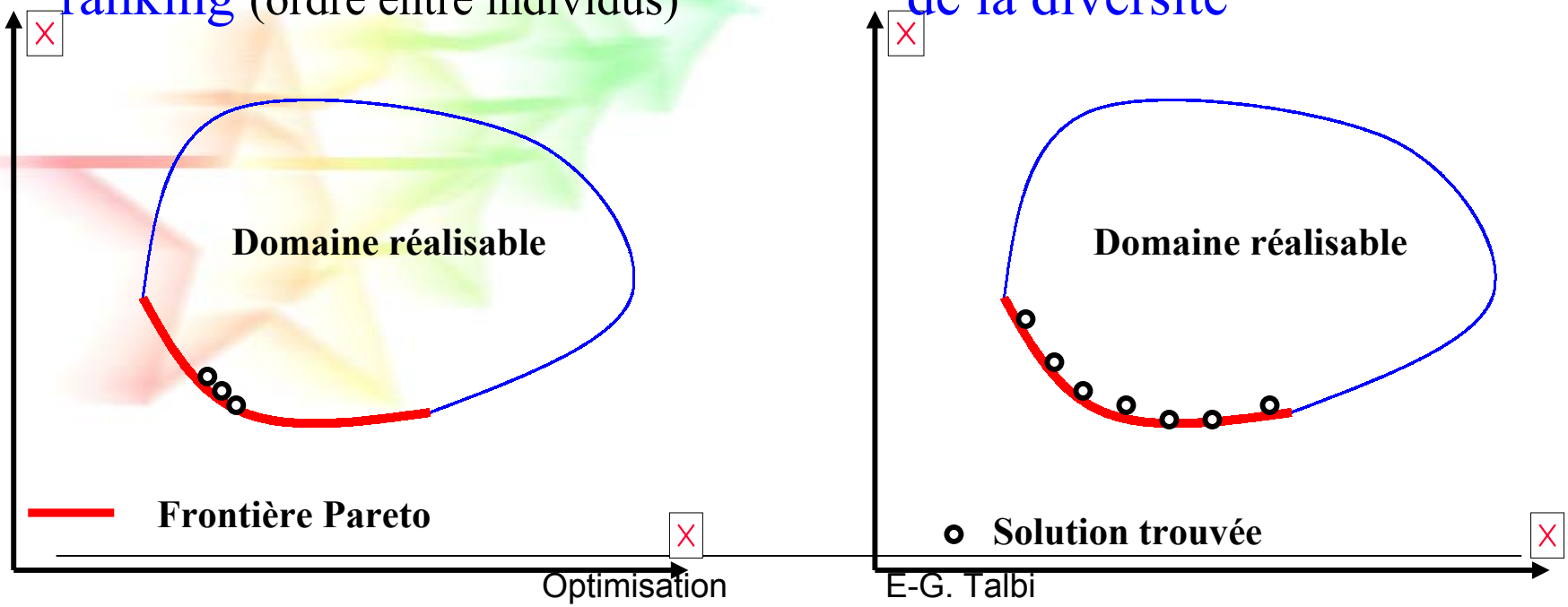
# Approches Pareto

- **Utilisent la notion de dominance dans la sélection des solutions.**
- **Capables de générer des solutions Pareto dans les portions concaves.**

## Objectifs

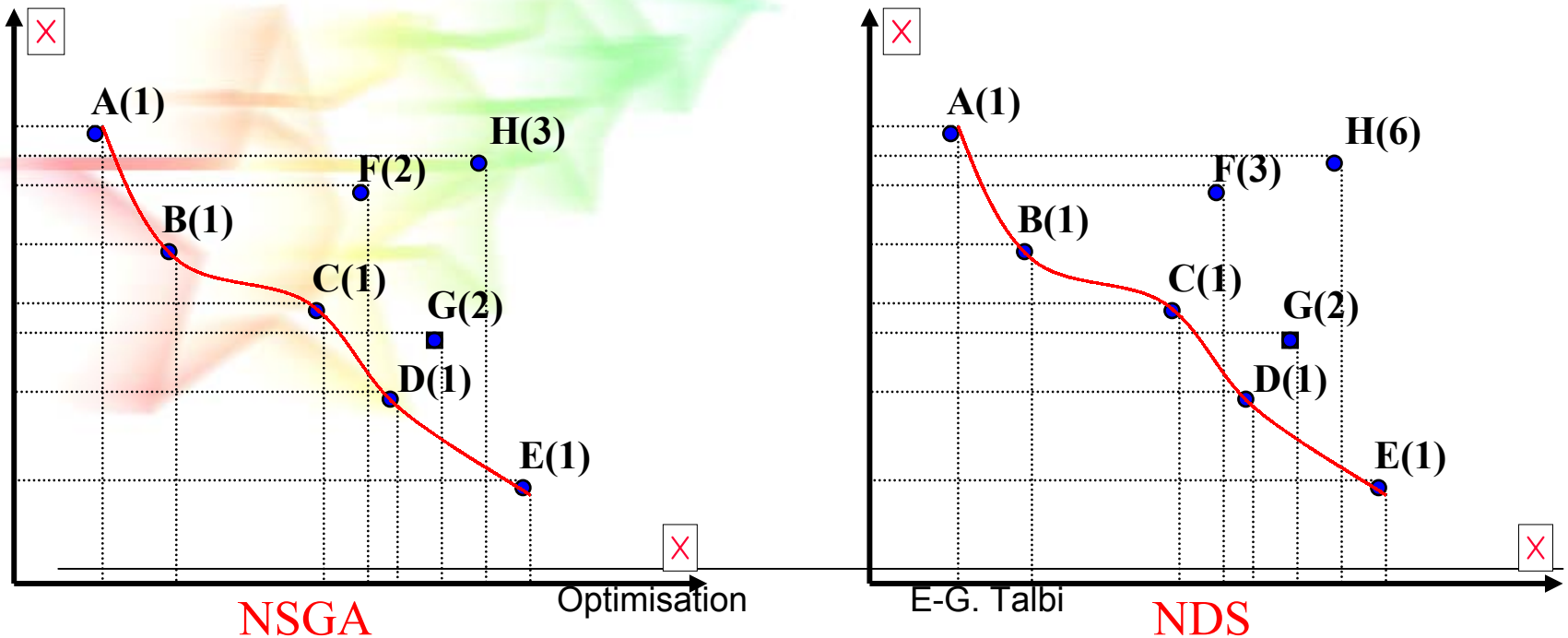
Convergence : Méthodes de ranking (ordre entre individus)

Diversification : Maintenance de la diversité



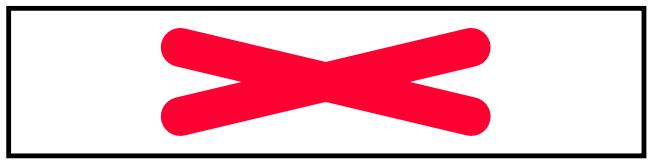
# Méthodes de ranking

- **NSGA** [Srinivas et Deb 95]
  - Rang Individus non dominés = 1. Pop=Pop-Individus non dominés
- **NDS** [Fonseca et Fleming 95]
  - Rang Individu = nombre de solutions le dominant
- **WAR** [Bentley 97], …
  - Rang Individu = moyenne des rangs / objectifs

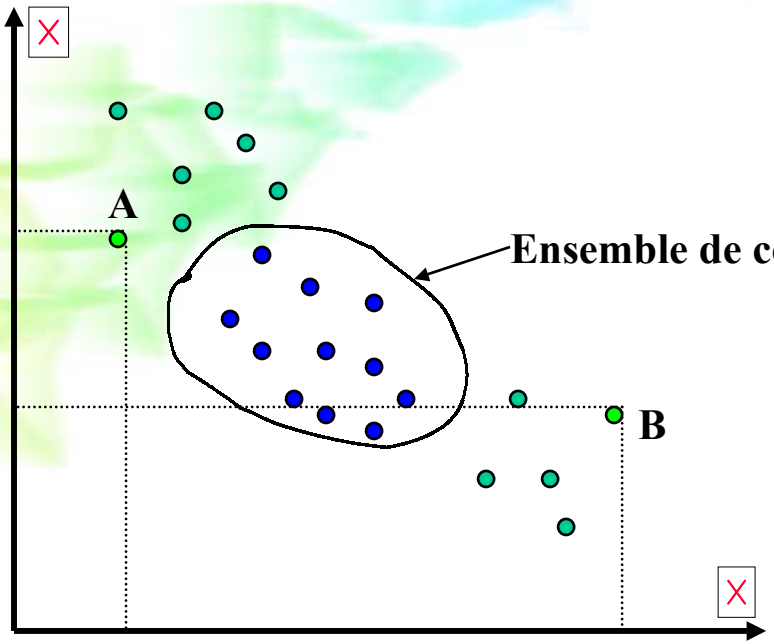


# Sélection

- Rang [Baker 85]



- Tournoi [Horn et al. 97]




- Autres : Roulette biaisée, etc ;

X = Probabilité de sélection

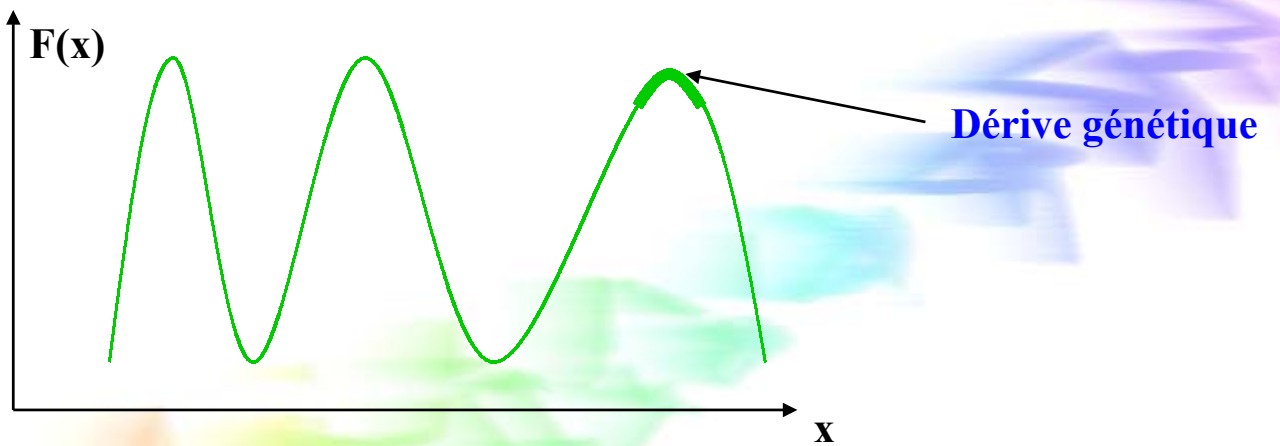
S = Pression de sélection

$$R_i = 1 + r_i + 2\sum_{i=1}^{n-1} r_i$$

$r_i$ = Nombre d 'individus de rang i

# Maintenance de la diversité

**Optimisation multi-modale : Localiser tous les optima d 'un problème**

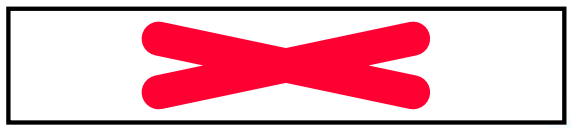


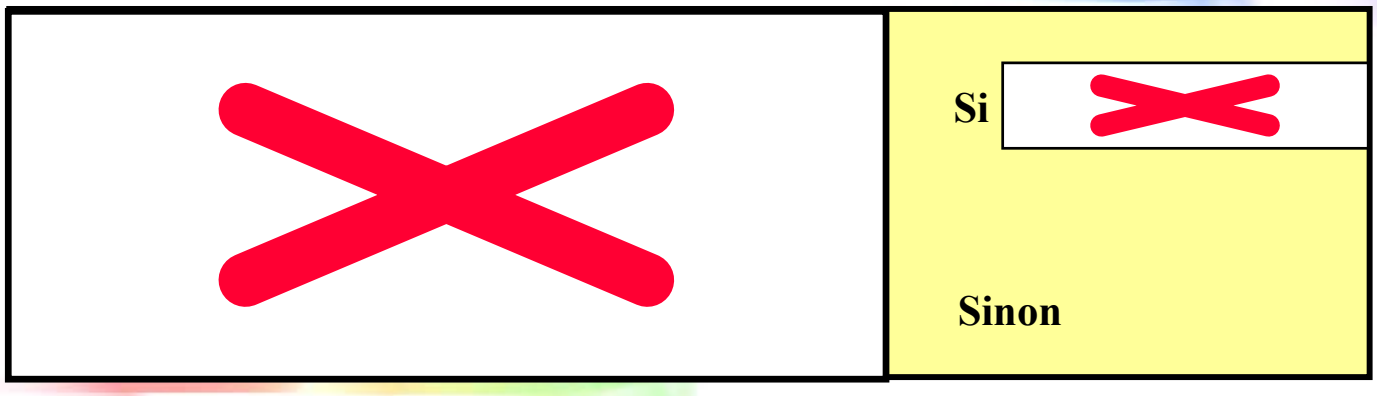

- Exécutions itératives indépendantes
- Niching séquentiel
  - Exécution itérative avec pénalisation des optima déjà trouvés
- Niching parallèle (sharing, crowding)
  - Une seule exécution

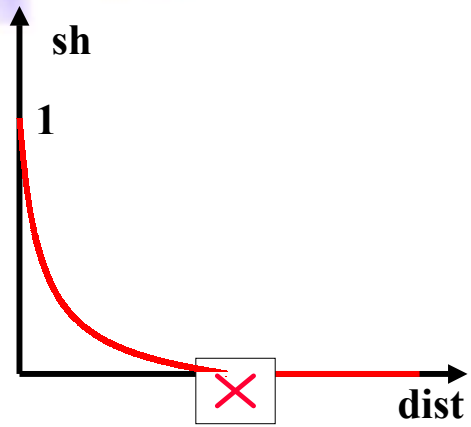
# Fonction de partage (sharing)

**Dégrade** le coût d ’un individu / nombre d ’individus **similaires**

**Compteur de niche**

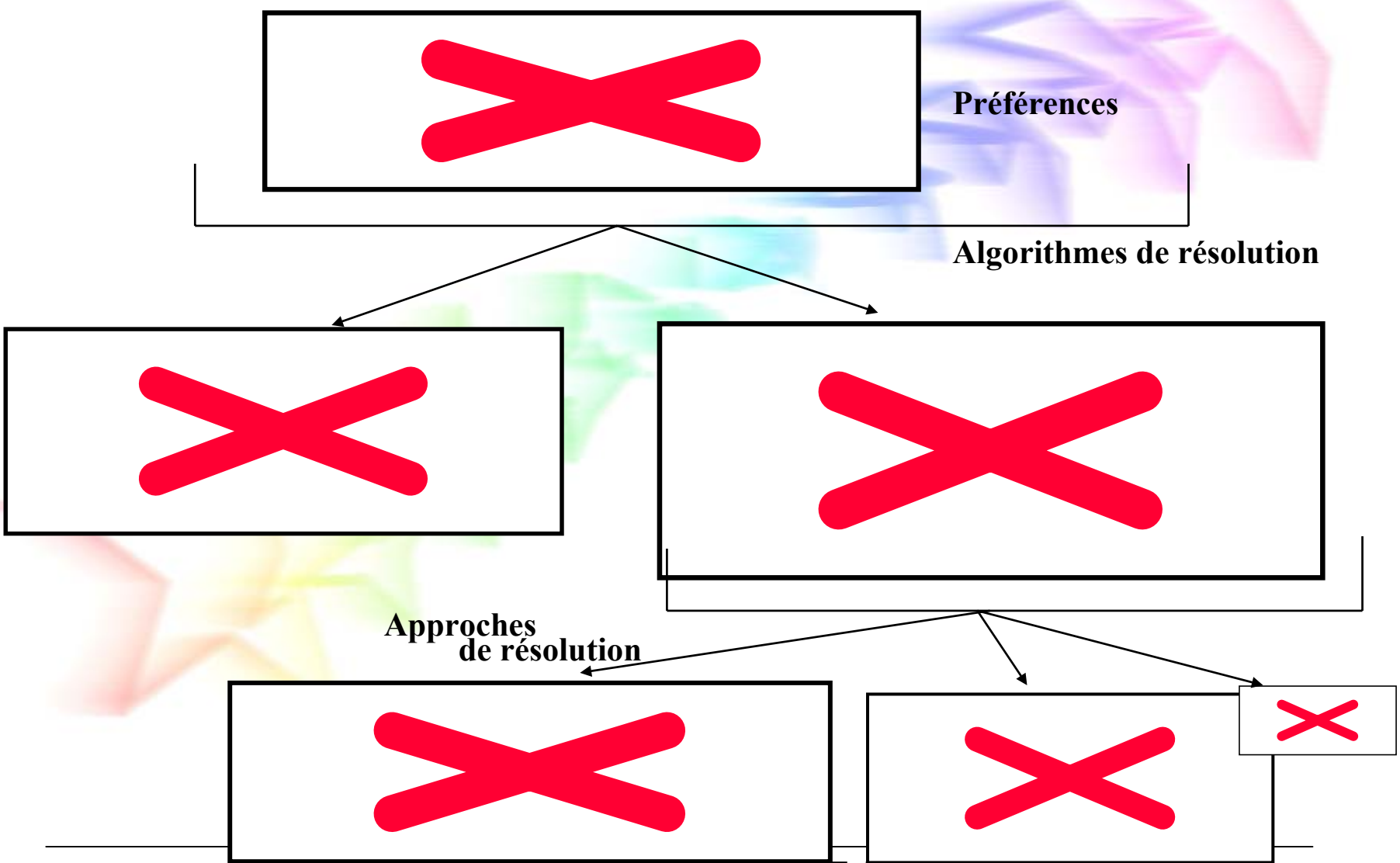





- Espace objectif et/ou Espace de décision (x,y) ?
- Métrique utilisée (dist) ?
- Taille des niches  ?

# Classification

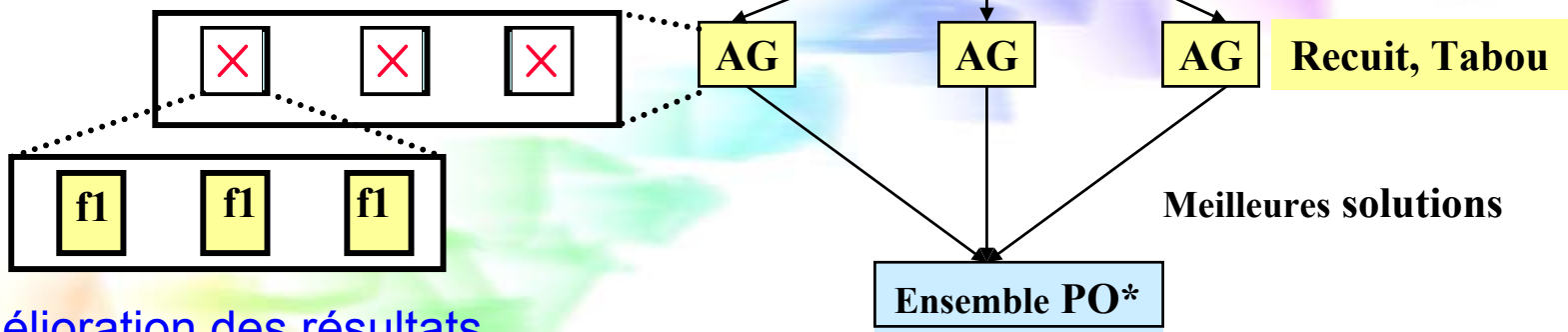


# Techniques avancées

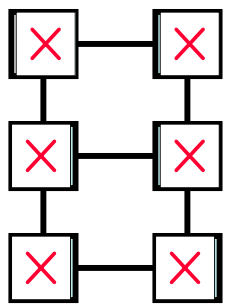
- Modèles parallèles

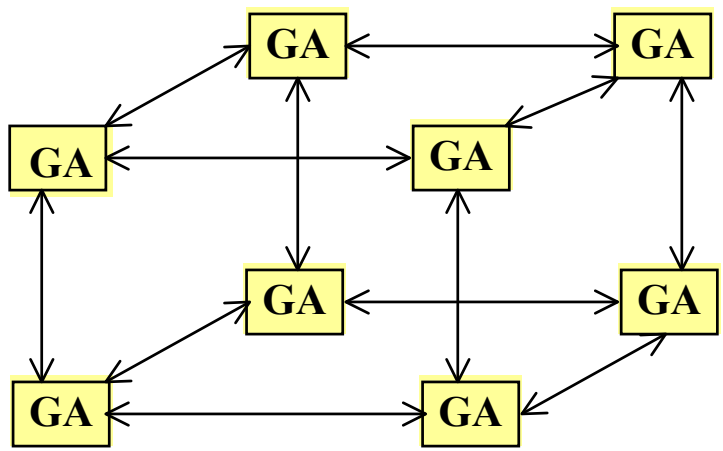
(Algorithmes génétiques)

- Accélération de la recherche




- Amélioration des résultats (coopération)
  - Modèle **insulaire**
  - Modèle **cellulaire**



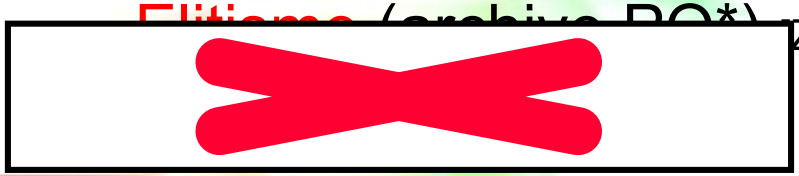


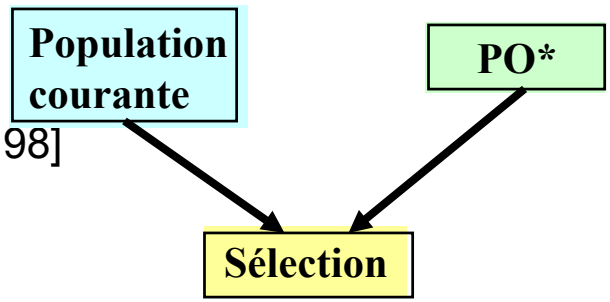

- Restriction de voisinage

# Techniques avancées

- Hybridation
  - Algorithme Génétique + Recherche Locale

- Elitisme (archive PO*) [Zitzler et Thiele 98]





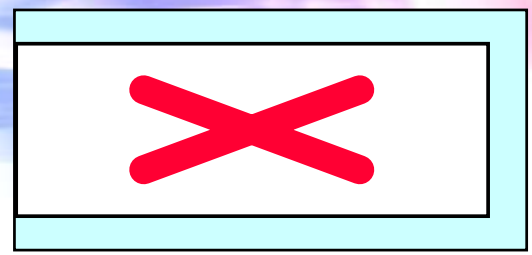

**A : Nombre d 'individus sélectionnés à partir de PO***

- Clustering de l 'ensemble PO* [Roseman et Gero 85]

# Evaluation des performances

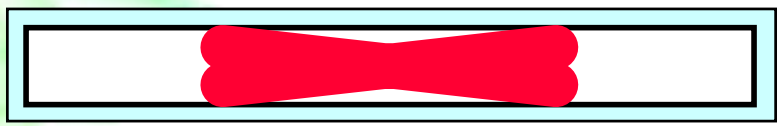
- **Ensemble PO connu** [Teghem et al.]
  - **Efficacité Absolue**

**Proportion des solutions Pareto dans PO***

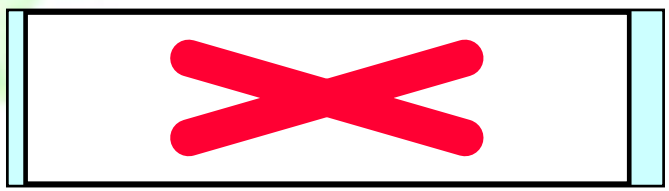


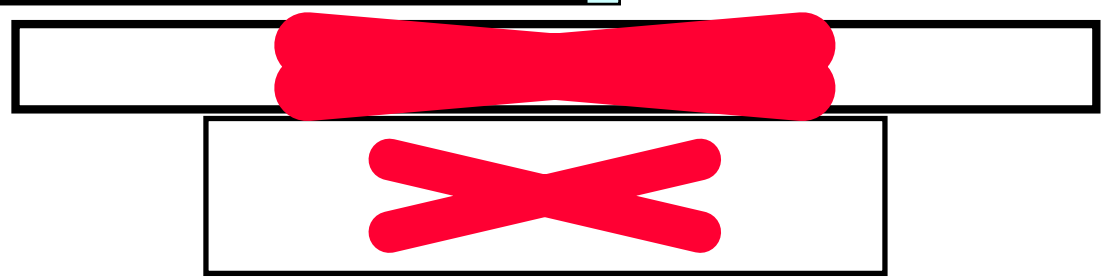
  - **Distance (PO*, PO)**

**Plus mauvaise distance**

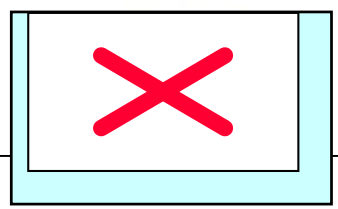


**Distance moyenne**





  - **Uniformité**



# Evaluation des performances

- Ensemble PO inconnu
  - Efficacité relative (A, B) : Nombre de solutions de A dominées par des solutions de B