Optimisation Combinatoire (Méthodes approchées)

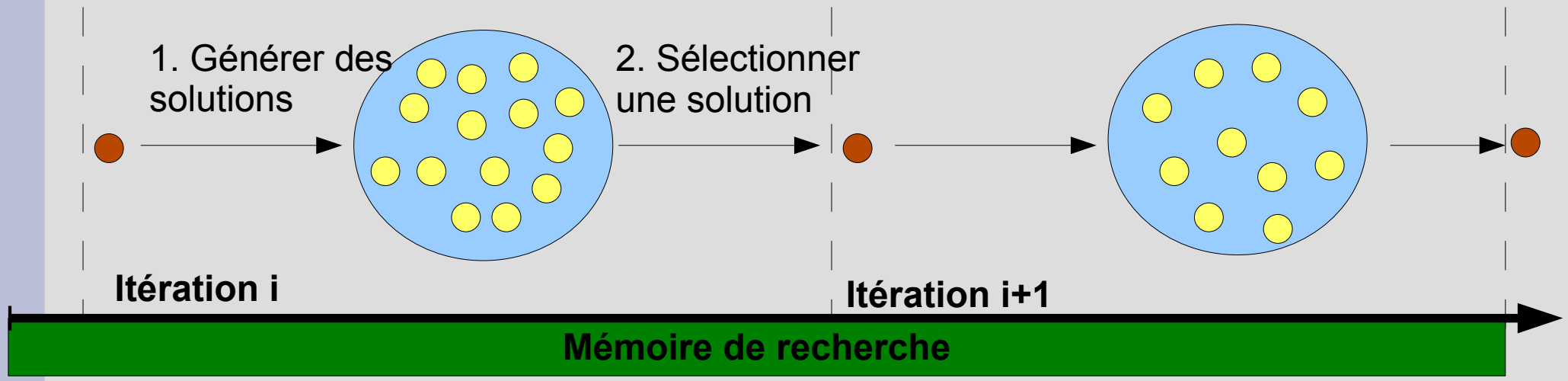
II. Recherche Locale simple (Les bases)

Heuristique Constructive

- Itérativement, ajoute de nouvelles composantes à une solution partielle candidate
  - Espace de recherche : solutions partielles
  - Étape de recherche : choisir de façon heuristique une composante 'optimale'
- Une seule solution au cours de la recherche

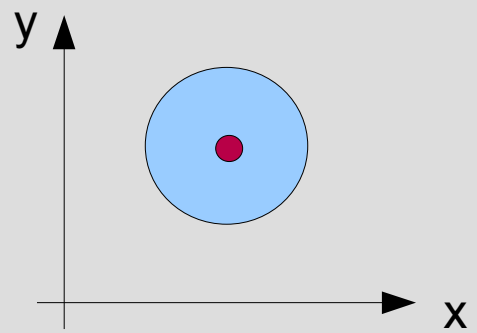
Recherche Locale

- Itérativement, parcourir l'espace de recherche **de proche en proche** à la recherche d'une 'bonne' solution
  - Une solution (candidate) initiale de départ
  - '**Améliorer**' cette solution en regardant dans son voisinage proche



Un voisinage ?

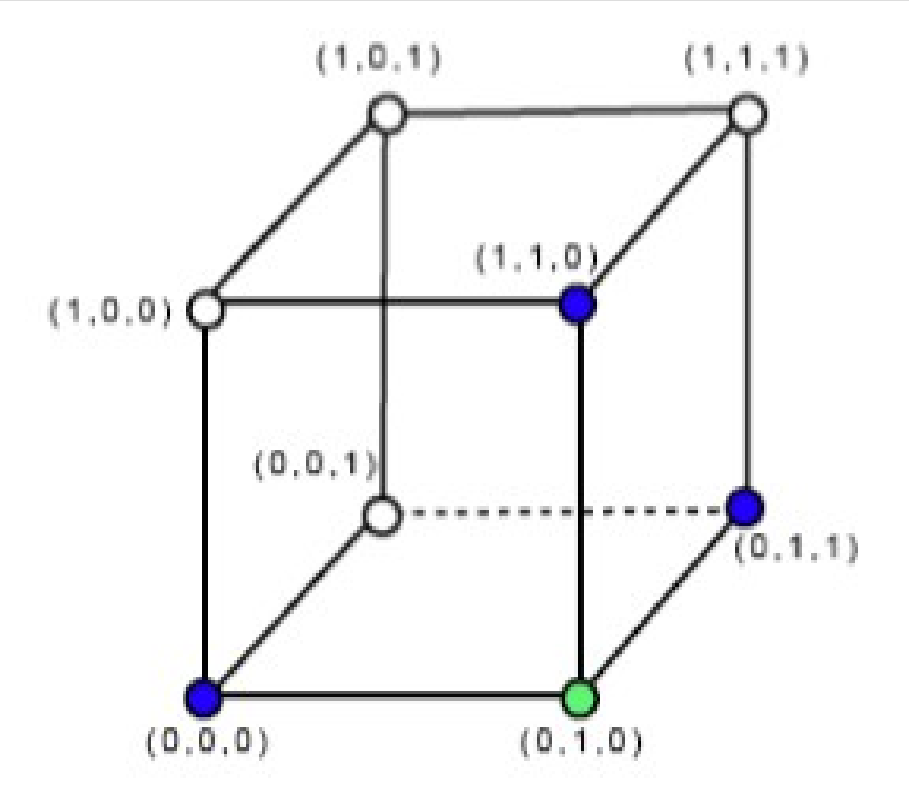
- Un ensemble de solutions qui diffèrent peu de s
- Voisinage d'un point dans le plan :

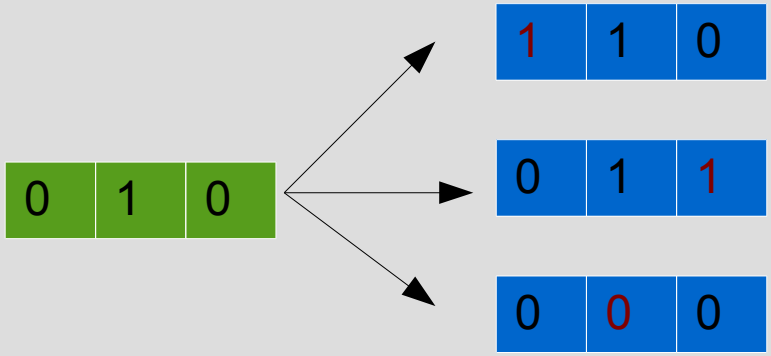


- Dans un espace (discret) de recherche S, le voisinage d'une solution est une fonction : ***N : S 6 N(S)***
  - **N est un operateur de voisinage**
  - Un opérateur de voisinage est par rapport à une **représentation** du problème/solution (**genotype**)

Exemples voisinages/représentation

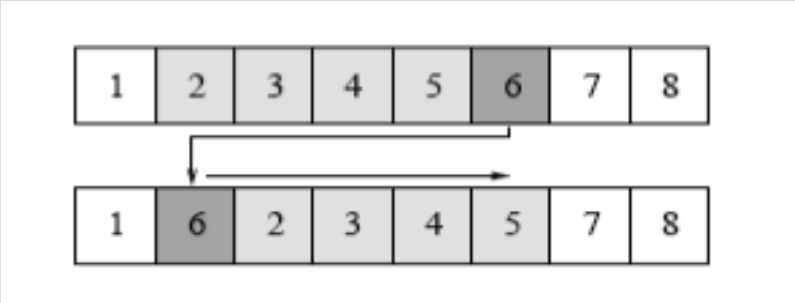
- Binaire (e.g. SAT, knapsack)
  - Hypercube des voisins à une distance de hamming 1



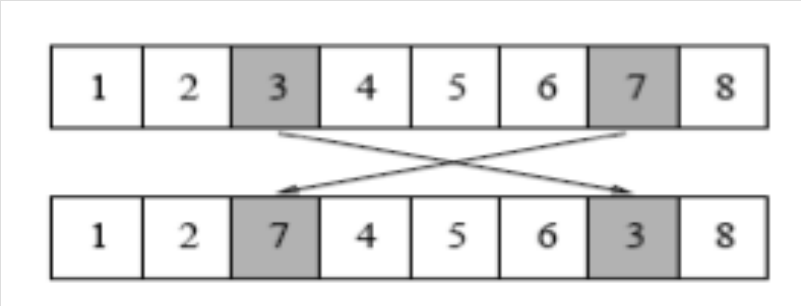


Exemples voisinages/représentation

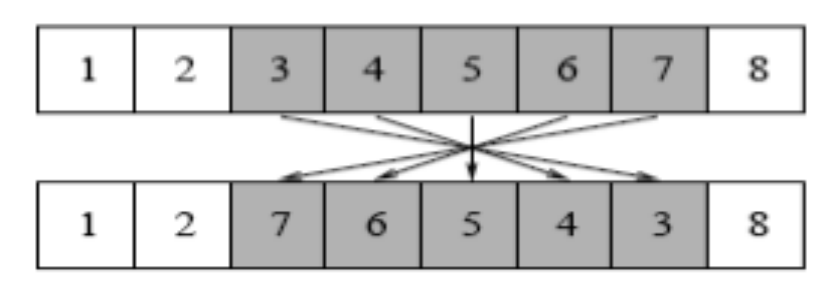
- Permutations (e.g., TSP, scheduling)
  - Insertion, swap, inversion, k-opt



insertion



Swap



inversion/échange

2-opt

Recheche Locale – Revisited

s ← solution initiale ;
Répéter
    N(S) ← générer des solutions voisines ;
    s' ← **choisir une solution dans N(s)** ;
    s ← s' ; // **remplacer** s
Jusqu'à **Condition d'arrêt**

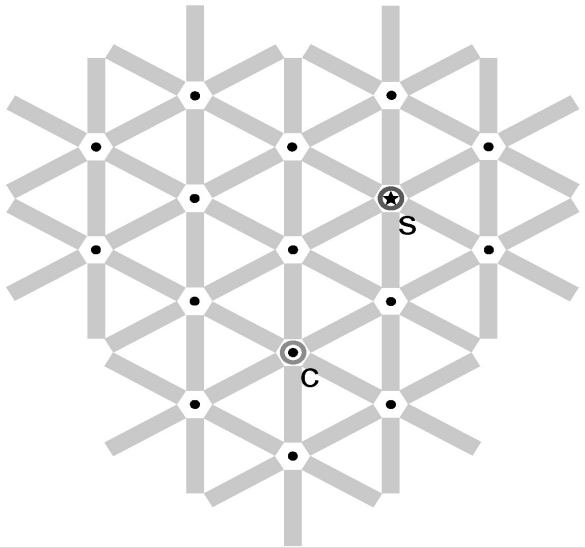
- Comment choisir/remplacer ?
- Comment générer une solution initiale ?
- Quelle condition d'arrêt ?

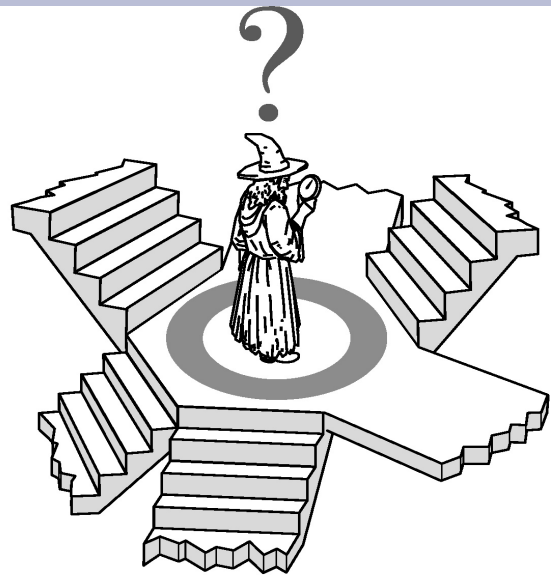
Recherche Locale – Aléatoire

s ← solution initiale ;
Répéter
    N(S) ← générer des solutions voisines ;
    s' ← **choisir aléatoirement une solution dans N(s)** ;
    s ← s' ; // **remplacer** s
Jusqu'à **Condition d'arrêt**

- Ineffectif en pratique
  - Mais possibilité de coupler avec d'autres types de recherches !

Vision globale Vs Vision Locale
Quelle boussoule utiliser ?





- **Sommets** : solutions candidates (positions)
- **Arêtes** : voisins d'une solution **selon l'opérateur N**
- s* : solution (optimale)
- s : solution candidate courante
- s' choisit **de façon locale** à partir de la position courante s

Fonction d'évaluation

- f : S → IR
  - Fonction d'évaluation ou fonction de **fitness**
  - Définit un score à chaque solution candidate
    - Définit la qualité d'une solution
  - *Pas toujours la même que la fonction objective qui est inhérente au problème*
    - e.g., SAT : nombre de clauses vérifiées
- f permet essentiellement de guider la recherche dans l'espace de recherche
  - L'objectif est de trouver un chemin qui aboutit à des bonnes solutions et ceci le plus rapidement possible

Recheche Locale (Revisted)

s ← solution initiale ;
Répéter
    N(s) ← générer des solution voisines ;
    s' ← choisir une solution dans N(s) **tel que f(s') < f(s)** ;
    s ← s' ; // remplacer s
Jusqu'à **Condition d'arrêt**

- Best Improvement
  - ***Choisir le meilleur voisins***
  - Il faut parcourir tous les voisins ! *Coût en temps*
- First improvement
  - ***Parcourir les voisins dans un ordre fixé***
    - Choisir le premier voisin s' tel que f(s') < f(s)
    - *Souvent plus efficace en pratique !*

Évaluation Incrémentale

- L'évaluation est en générale l'étape la plus coûteuse
- Calculer l'apport (la différence) d'une solution voisine s' (un move) par rapport à s (position courante)
  - L'évaluation d'une solution est une aggregation de ces composantes
  - Un opérateur de voisinage change en général quelques composantes seulement

    → calculer la différence de f(s') par rapport à f(s) et en déduire f(s') (au lieu de calculer directement f(s'))
- Exemple : 2-opt avec TSP :
  - w(p') = w(p) – arêtes dans p mais non dans p'
    + arêtes dans p' mais non dans p

Solution initiale

- Essentiellement deux stratégies
  - Random
  - Heuristic (e.g., greedy)
- Une solution initiale de meilleurs qualité n'est pas forcément meilleurs !
- Génération d'une solution candidate initiale de façon aléatoire peut être difficile

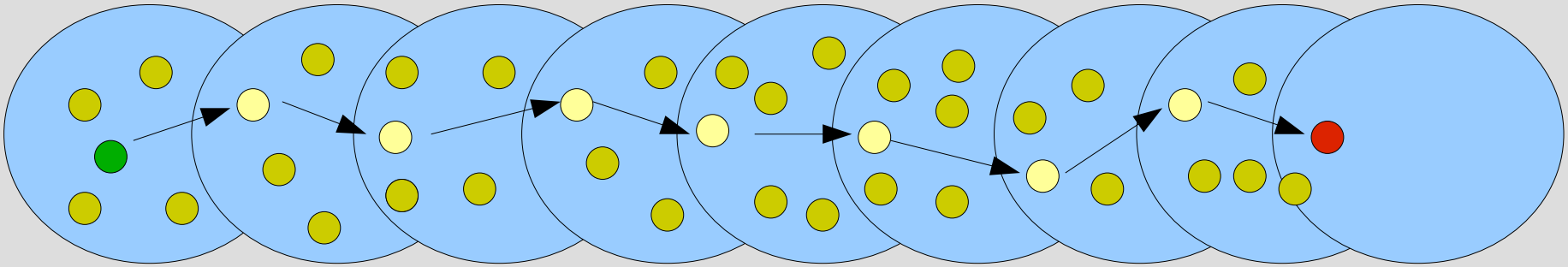
Hill Climbing – Descent – iterative improvement

```
s ← solution initiale ;
Répéter
      N(s) ← générer des solution voisines ;
      s' ← choisir une solution dans N(s) tel que f(s') < f(s) ;
      s ← s' ; // remplacer s
Jusqu'à Pas d'amélioration possible (Optimum local)
Retourner s ;
```

- Optimum local (Local Optimum)
  - Une solution s tel que f(s) **<=** f(s') pour tout s' dans N(s)
  - *Optimum local strict : f(s) **<** f(s') pour tout s' dans N(s)*
  - *Il peut en avoir plusieurs !*

Optima locaux

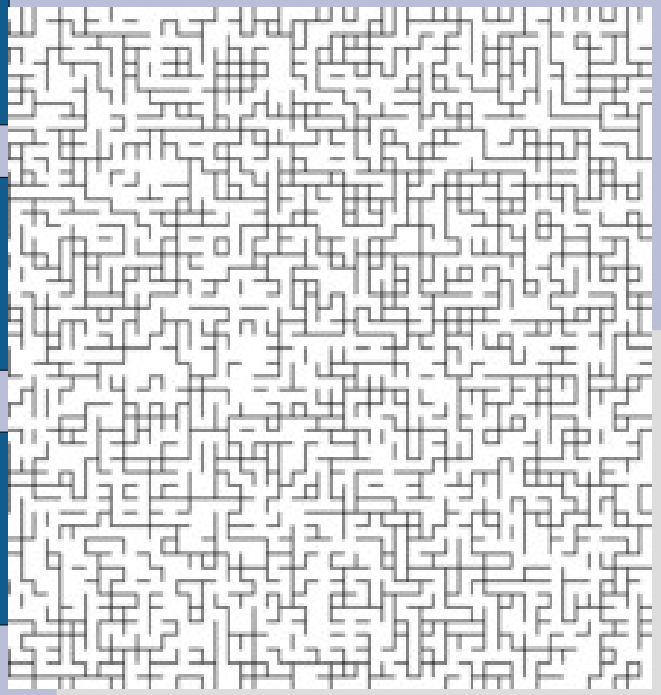
- *Trajectoire de recherche* : $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_k$
- s_k dépend de s_0, du voisinage N, de la fonction d'évaluation (fitness) et de la stratégie de sélection

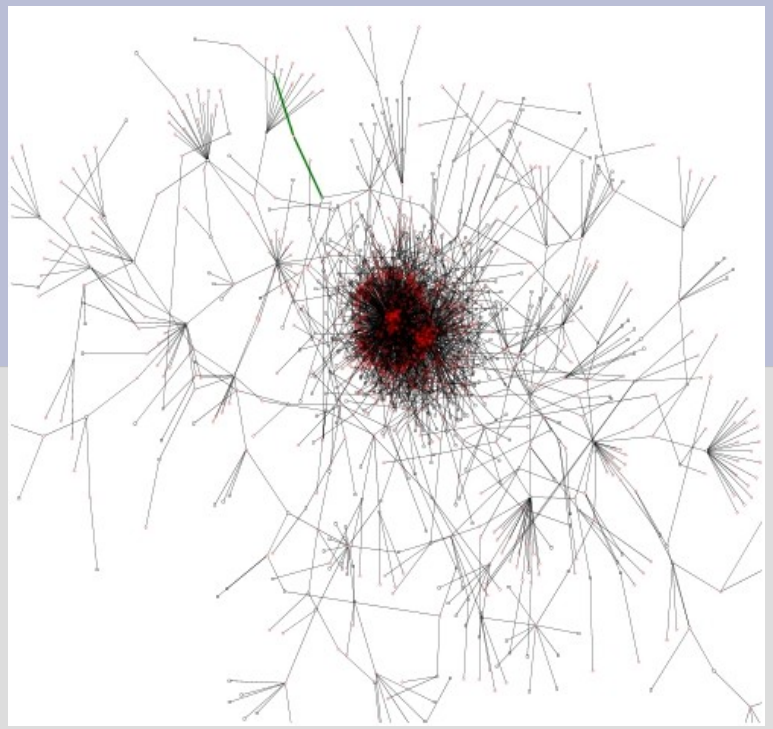


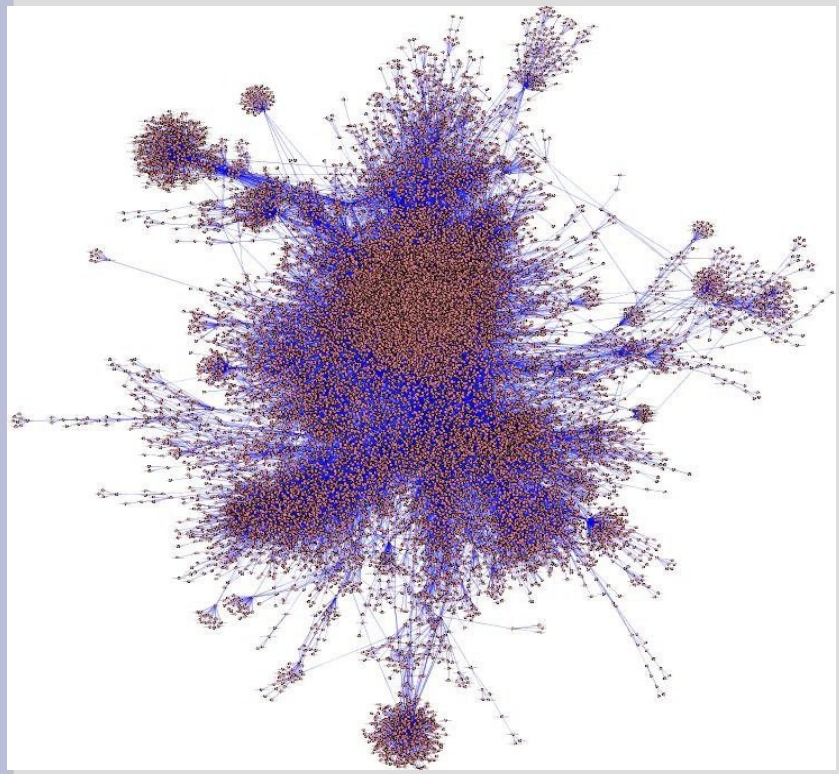
évaluation

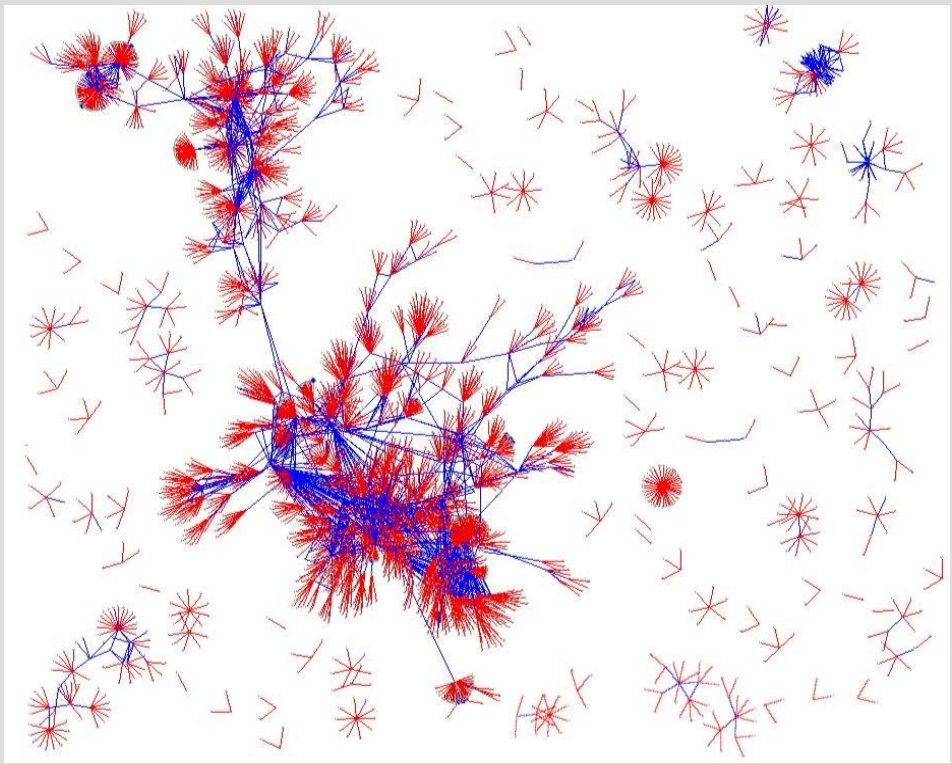
solutions

Trajectoire









Échapper les Optima locaux

- Quelques mécanismes simples
  - Restart : recommencer avec une nouvelle solution initiale
  - Non-improving steps : Quand arrivé sur un optimum local, accepter de faire des 'move' vers des solutions voisines ayant une fitness égale ou moins bonne
- Pas de garantie que cela soit effectif !

Intensification Vs Diversification

- Intensification
  - Itérativement, de façon gloutonne, augmenter la qualité de la solution ou la probabilité d'aller vers une bonne solution
- Diversification
  - Éviter de stagner dans une région (e.g. Optima locaux)
- Exemples :
  - Random local search : diversification
  - Hill-climbing (descent) : intensification
- **L'ingrédient de base d'une bonne recherche Locale est un bon équilibre entre Intensification et Diversification**