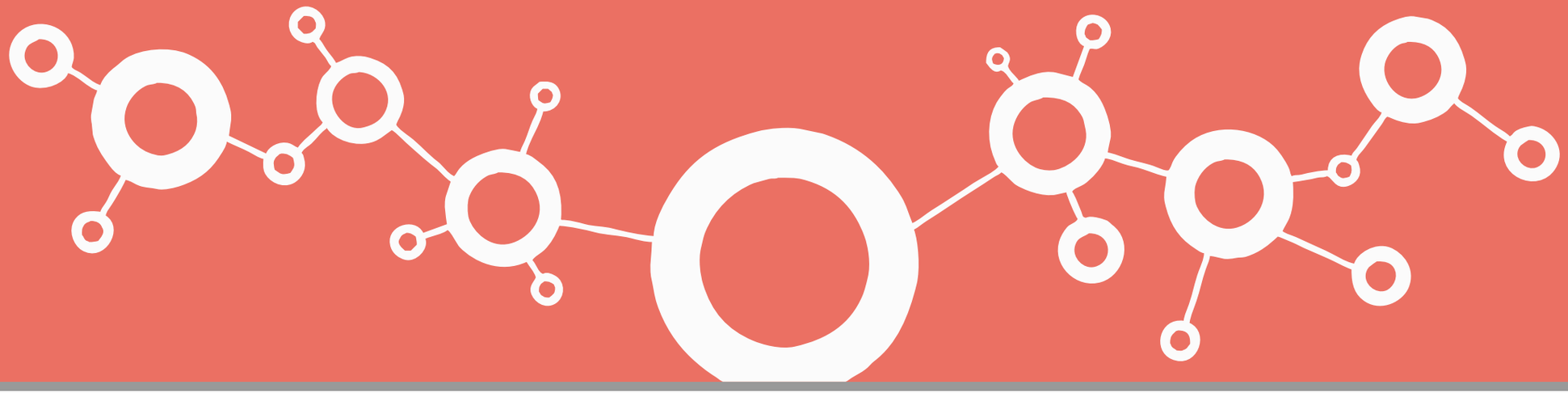


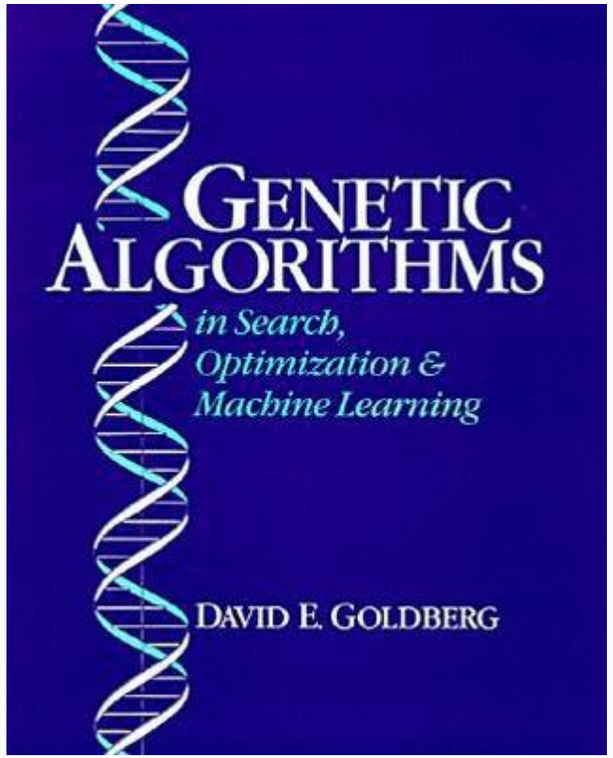
Ch. 12 : Algorithmes génétiques

Un peu de terminologie…

Famille d'algorithmes dont le principe s'inspire des mécanismes de la théorie de l'évolution (***reproduction***, ***mutation***, ***recombinaison*** et ***sélection***) pour résoudre des problèmes d'optimisation divers.

- Issus des travaux de **John Holland** et son équipe (Université du Michigan), dans les années 1960.

- Popularisés par l'ouvrage de **David Golberg**, *Genetic Algorithms in Search, Optimization, and Machine Learning* (1989)



Métaheuristiques

Les algorithmes génétiques sont un type de ***métaheuristique***

- Une ***heuristique*** est une « méthode », un « truc » qui permet d'obtenir rapidement une solution réalisable à un problème, ou d'améliorer une solution existante

  Exemple : si, à SUTOM / MOTUS, on a trouvé la séquence « CH » dans un mot, on sait qu'en français, il y a peu de chances qu'elle soit suivie par un B, un F, un G ou encore un X…

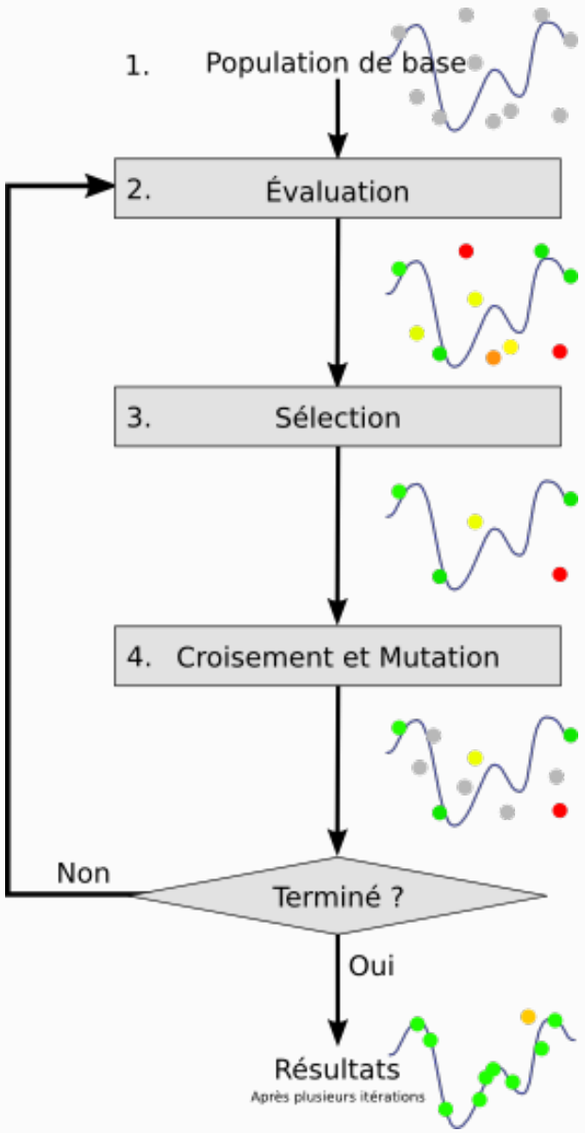
  Mais si on a la séquence « TIO », il y a de fortes chances que le mot se termine par « TION »

- ***méta*** signifie ici que ces algorithmes peuvent servir d'heuristiques à un grand nombre de problèmes très variés

4 Exemple d'application

Antenne mise au point par un algorithme génétique développé par la NASA pour le programme *Space Technology 5* en 2006, pour créer le meilleur diagramme de rayonnement

Principe général



Une *population* d'*individus* est générée aléatoirement ; chaque individu est représenté par un *chromosome*

On associe à chaque individu un *score*, ou *fitness*
(on verra plus loin comment trouver cette fonction de *fitness*)

Les meilleurs individus sont sélectionnés pour faire partie de la génération suivante et se reproduire ; les autres sont éliminés

On choisit des individus pour engendrer, par *croisement*, de nouveaux individus pour compléter la population. Tous les membres de la population sont susceptibles de subir une légère *mutation*.

On s'arrête au bout de *k* générations ou quand les solutions n'évoluent plus

Exemple : déchiffrer un mot de passe

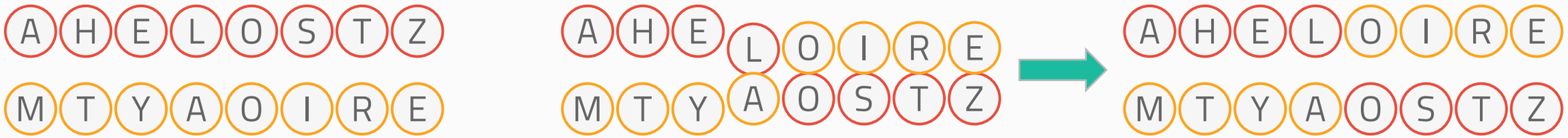
- On veut déchiffrer un mot de passe (par ex., un mot de huit lettres pris au hasard dans le dictionnaire)
- On dispose d'un « oracle », capable de nous dire si une lettre est bien placée, ou si elle appartient au mot à trouver (un peu comme dans Motus / Sutom)
- On veut trouver le mot de passe à l'aide d'un algorithme génétique

Chaque *individu* sera une combinaison aléatoire de 8 lettres : (A) (H) (E) (L) (O) (S) (T) (Z)

Fitness : +10 par lettre correctement positionnée, +2 par lettre correcte mais mal positionnée (par exemple !)

Exemple : déchiffrer un mot de passe

Un *croisement* / *cross-over* (ou *enjambement*) consiste à permuter deux individus à partir d'une certaine position aléatoire :



Une *mutation* consiste à remplace un caractère par un caractère aléatoire

Mise en œuvre et difficultés

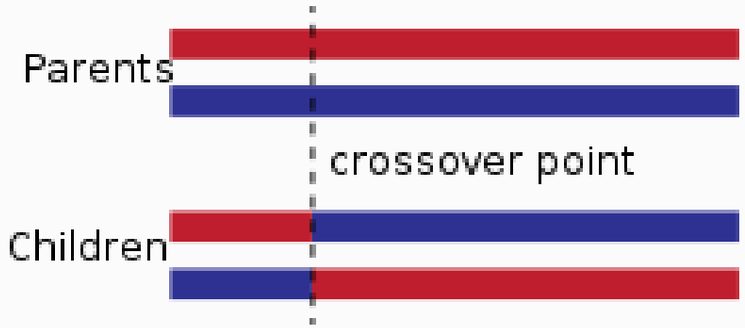
1. Trouver une représentation sous forme de chaîne (de caractères, de bits…) des « individus »
2. Trouver une fonction d'évaluation *(fitness)* pertinente

3. Comment effectuer la sélection des « meilleurs individus » ? Plusieurs méthodes :
   - *Roulette* : probabilité proportionnelle à la fitness
   - *Etat d'équilibre* : à chaque génération, seuls les quelques meilleurs individus sont sélectionnés pour donner naissance à de nouveaux individus, qui remplacent les plus mauvais ; le reste de la population survit
   - *Tournoi* : on tire *k* individus au hasard et on conserve le meilleur ; on répète l'opération *n* fois
   - *Elitisme* : les meilleurs individus participent à la génération suivante sans aucun changement

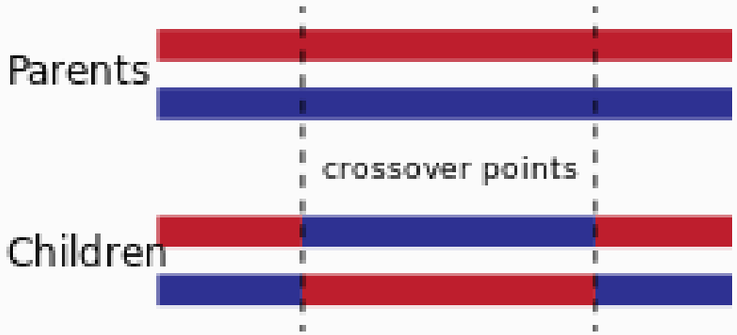
Mise en œuvre et difficultés

4. Différents types de croisements :

   - *Croisements 1-point* :



   - *Croisements k-points* :



Avantages et inconvénients

Avantages :

- Concept « bio-inspiré », facile à comprendre
- Permettent d'obtenir rapidement une solution « pas trop mauvaise »
- Facilement parallélisable

Inconvénients :

- Temps de calcul souvent plus élevé que d'autres métaheuristiques
- Difficiles à mettre en œuvre (choix des paramètres : taille de population, taux de mutation…)
- Aucune garantie qu'on atteint la solution optimale même après un grand nombre de générations
- On tombe souvent dans un *minimum local*, duquel il n'est pas toujours facile de s'échapper