IFT 615 – Intelligence artificielle

Recherche locale

Hugo Larochelle
Département d'informatique
Université de Sherbrooke
http ://www.dmi.usherb.ca/~larocheh/cours/ift615.html

Objectifs

- Comprendre :
  - la différence entre une recherche heuristique et une recherche locale
  - la méthode *hill-climbing*
  - la méthode *simulated annealing*
  - les algorithmes génétiques

Motivations pour une recherche locale

- Rappel de quelques faits saillants de A* :
  - un noeud ou état final (le but) à atteindre est donné comme entrée
  - la solution est un chemin et non juste un noeud final
  - idéalement on veut un chemin optimal
  - les noeuds rencontrés sont stockés pour éviter de les revisiter
- Pour certains types de problèmes impliquant une recherche dans un espace d'états, on pourrait avoir les caractéristiques suivantes :
  - il y une **fonction objectif** (*objective function*) à optimiser (possiblement avec une fonction but qui identifie un noeud final)
  - la solution recherchée est juste le noeud optimal (ou proche) et **non le chemin qui y mène**
  - l'espace d'états est trop grand pour enregistrer les noeuds visités
- Pour ce genre de problèmes, une recherche locale peut être la meilleure approche

Principe d'une recherche locale

- Une recherche locale garde juste certains noeuds visités en mémoire :
  - le cas le plus simple est *hill-climbing* qui garde juste **un noeud** (le noeud courant) et l'améliore itérativement jusqu'à converger à une solution
  - le cas le plus élaboré est celui des algorithmes génétiques qui gardent **un ensemble de noeuds** (appelé *population*) et le fait évoluer jusqu'à obtenir une solution
- En général, il y a une fonction objectif à optimiser, c.-à-d. maximiser ou minimiser
  - dans le cas de *hill-climbing*, elle permet de déterminer le noeud visité suivant
  - dans le cas des algorithmes génétiques, on l'appelle la **fonction de *fitness*** : elle intervient dans le calcul de l'ensemble des noeuds successeurs de l'ensemble courant
- En général, **une recherche locale ne garantie pas de solution optimale**
  - son attrait est surtout sa capacité de trouver une solution acceptable rapidement

Méthode *hill-climbing*

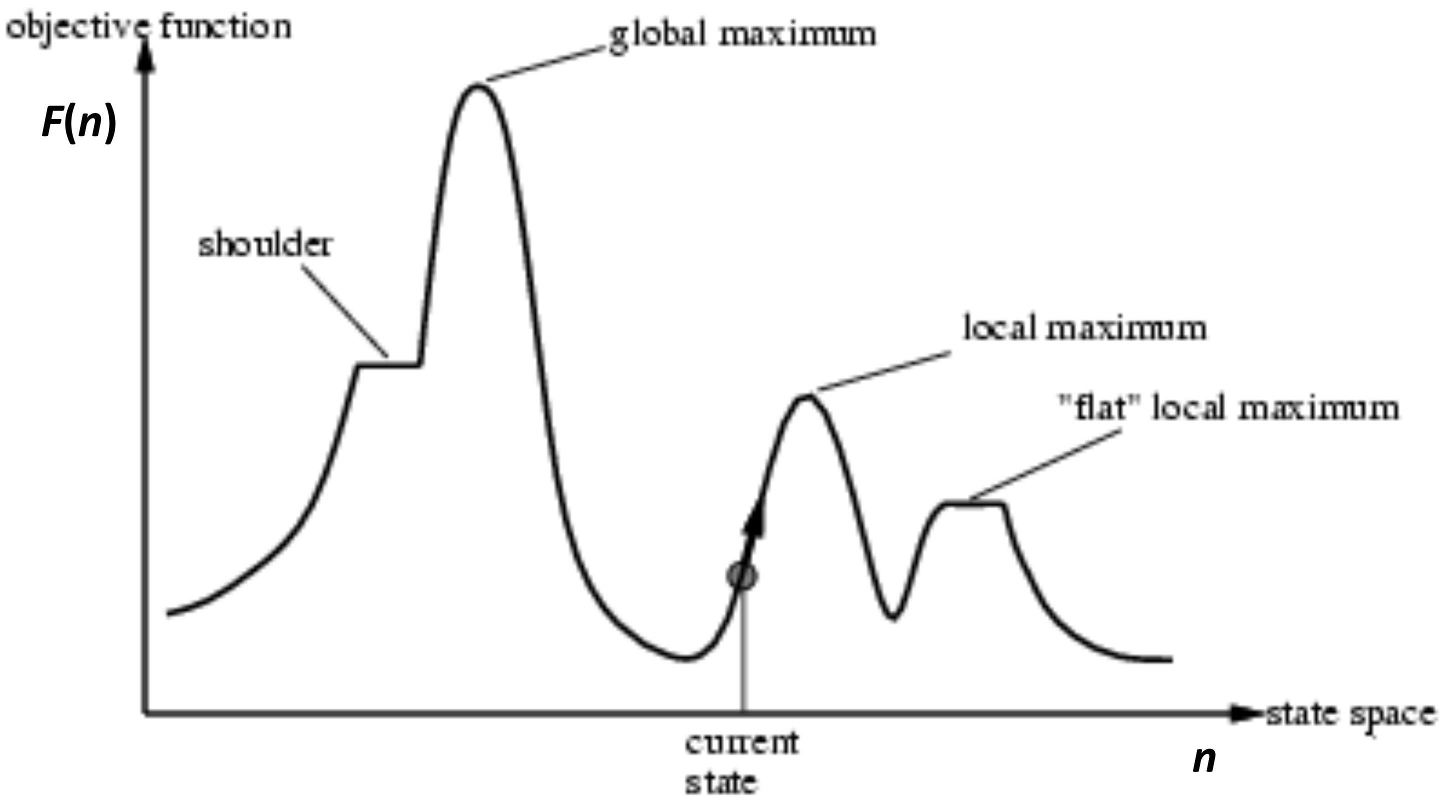
- Entrées :
  - noeud initial
  - fonction objectif à optimiser :
    - » notée *F(n*) dans les algorithmes
- Méthode :
  - le nœud courant est initialisé au noeud initial
  - itérativement, le nœud courant est comparé à ses successeurs immédiats
    - » **le meilleur voisin *n'* immédiat** et ayant la plus grande valeur *F*(*n*) que le nœud courant, devient le nœud courant
    - » **si un tel voisin n'existe pas, on arrête** et on retourne le nœud courant comme solution

Algorithme *hill-climbing*

Algorithme HILL-CLIMBING(*noeudInitial*) // *cette variante maximise*

1. déclarer deux nœuds : *n*, *n'*
2. *n = noeudInitial*
3. tant que (1) // *la condition de sortie (exit) est déterminée dans la boucle*
    4. *n'* = noeud successeur de *n* ayant la plus grande valeur *F(n')*
    5. si *F*(*n'*) ≤ *F*(*n*) // *si on minimisait, le test serait F(n') ≥ F(n)*
        6. retourner *n* // *on n'arrive pas à améliorer p/r à F(n)*
    7. *n = n'*

Illustration de l'algorithme *hill-climbing*



Imaginez ce que vous feriez pour arriver au (trouver le) sommet d'une colline donnée, en plein brouillard et soufrant d'amnésie.

Exemple de simulation de *hill-climbing*

- Soit la fonction objectif suivante, définie pour les entiers de 1 à 16 :

***n* =**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
$F(n)$ =	4	6	15	5	3	2	4	5	6	7	8	10	9	8	7	3

- Quelle valeur de *n* trouverait la méthode *hill-climbing* si la valeur initiale de *n* était 6 et que les successeurs (voisins) utilisés étaient *n-1* (seulement si *n>1*) et *n+1* (seulement si *n<16*)?

Exemple de simulation de *hill-climbing*

- Soit la fonction objectif suivante, définie pour les entiers de 1 à 16 :

?

n =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
F(*n*) =	4	6	15	5	3	2	4	5	6	7	8	10	9	8	7	3

- Quelle valeur de *n* trouverait la méthode *hill-climbing* si la valeur initiale de *n* était 6 et que les successeurs (voisins) utilisés étaient *n-1* (seulement si *n>1*) et *n+1* (seulement si *n<16*)?

- **Réponse**:
  - suite des valeurs de *n* parcourues: 6

Exemple de simulation de *hill-climbing*

- Soit la fonction objectif suivante, définie pour les entiers de 1 à 16 :

?

n =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
F(*n*) =	4	6	15	5	3	2	4	5	6	7	8	10	9	8	7	3

- Quelle valeur de *n* trouverait la méthode *hill-climbing* si la valeur initiale de *n* était 6 et que les successeurs (voisins) utilisés étaient *n-1* (seulement si *n>1*) et *n+1* (seulement si *n<16*)?

- **Réponse**:
  - suite des valeurs de *n* parcourues: 6 → 7

Exemple de simulation de *hill-climbing*

- Soit la fonction objectif suivante, définie pour les entiers de 1 à 16 :

?

n =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
F(*n*) =	4	6	15	5	3	2	4	5	6	7	8	10	9	8	7	3

- Quelle valeur de *n* trouverait la méthode *hill-climbing* si la valeur initiale de *n* était 6 et que les successeurs (voisins) utilisés étaient *n-1* (seulement si *n>1*) et *n+1* (seulement si *n<16*)?

- **Réponse**:
  - suite des valeurs de *n* parcourues: 6 → 7 → 8

Exemple de simulation de *hill-climbing*

- Soit la fonction objectif suivante, définie pour les entiers de 1 à 16 :

?

n =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
F(*n*) =	4	6	15	5	3	2	4	5	6	7	8	10	9	8	7	3

- Quelle valeur de *n* trouverait la méthode *hill-climbing* si la valeur initiale de *n* était 6 et que les successeurs (voisins) utilisés étaient *n-1* (seulement si *n>1*) et *n+1* (seulement si *n<16*)?

- **Réponse**:
  - suite des valeurs de *n* parcourues: 6 → 7→ 8→ 9→ 10→ 11→ 12
  - *hill-climbing* termine et retourne *n*=12

Exemple : *N-Queen*

- Problème : Placer *N* reines sur un échiquier de taille *N*× *N* de sorte que deux reines ne s'attaquent pas mutuellement :
    - c-à-d., jamais deux reines sur la même diagonale, ligne ou colonne

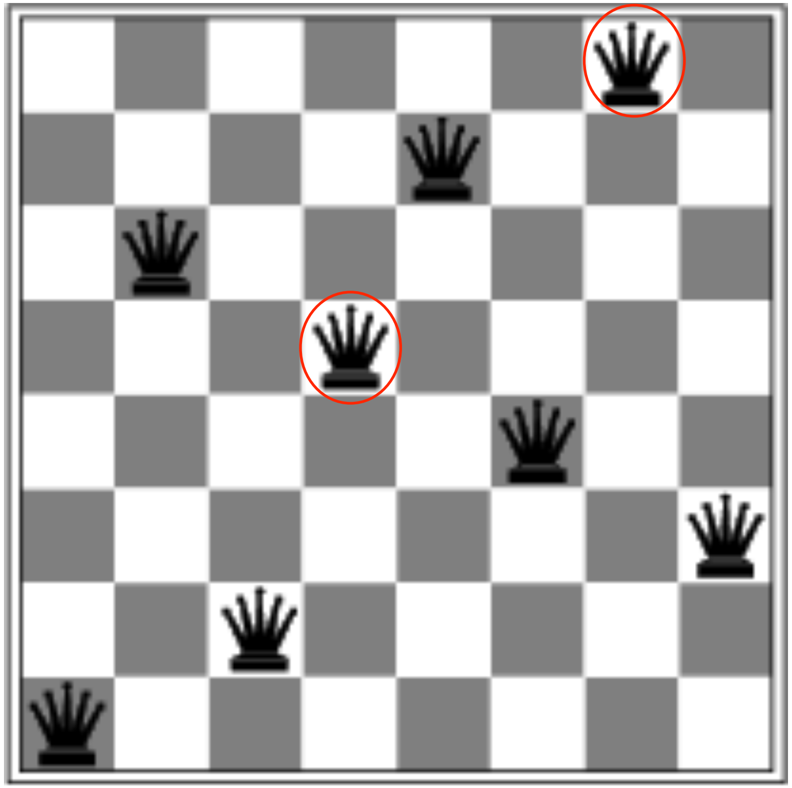
Hill-climbing avec 8 reines

- *n* : configuration de l'échiquier avec *N* reines
- *F*(*n*) : nombre de paires de reines qui s'attaquent mutuellement directement ou indirectement dans la configuration *n*
- On veut le **minimiser**

- *F*(*n*) pour l'état (noeud) affiché : 17
- Encadré : les meilleurs successeurs, si on bouge une reine dans sa colonne

Hill-Climbing avec 8 reines

- Un exemple de minimum local avec $F(n)$=1



Méthode *simulated annealing* (recuit simulé)

- C'est une amélioration de l'algorithme *hill-climbing* pour **minimiser le risque d'être piégé dans des maxima/minima locaux**
  - au lieu de regarder le meilleur voisin immédiat du nœud courant, **avec une certaine probabilité on va regarder un moins bon voisin immédiat**
    - on espère ainsi s'**échapper des optima locaux**
  - au début de la recherche, la **probabilité de prendre un moins bon voisin** est plus élevée et **diminue graduellement**

- Le nombre d'itérations et la diminution des probabilités sont définis à l'aide d'un schéma (*schedule*) de « températures », en ordre décroissant
  - ex.: schéma [2^{-0}, 2^{-1}, 2^{-2}, 2^{-3}, ... , 2^{-99}], pour un total de 100 itérations
  - la meilleure définition du schéma va varier d'un problème à l'autre

Algorithme *simulated annealing*

Algorithme SIMULATED-ANNEALING(*noeudInitial, schema*) // *cette variante maximise*

1. déclarer deux nœuds : *n*, *n'*
2. déclarer : *t*, *T*, *ΔE*,
3. *n* = *noeudInitial*
4. pour *t* = 1 ... taille(*schema*)
    5. *T* = *schema*[*t*]
    6. *n'* = successeur de *n* choisi au hasard
    7. *ΔE* = *F*(*n'*) – *F*(*n*) // *si on minimisait, ΔE = F(n) – F(n')*
    8. si *ΔE* > *0* alors assigner *n* = *n'* // *amélioration p/r à n*
    9. sinon assigner *n* = *n'* seulement avec probabilité de $e^{\Delta E / T}$
10. retourner *n*

plus *T* est grand,
plus $e^{\Delta E / T}$ est petite

D'autres améliorations: *tabu search*

- L'algorithme *simulated annealing* minimise le risque d'être piégé dans des optima locaux
- Par contre, il n'élimine pas **la possibilité d'osciller indéfiniment** en revenant à un noeud antérieurement visité
- **Idée**: On pourrait **enregistrer les noeuds visités**
  - on revient à A* et approches similaires!
  - mais c'est impraticable si l'espace d'états est trop grand

- **L'algorithme *tabu search*** (recherche taboue) enregistre seulement les ***k* derniers noeuds visités**
  - l'**ensemble taboue** est l'ensemble contenant les *k* noeuds
  - le paramètre *k* est choisi empiriquement
  - cela n'élimine pas les oscillations, mais les réduit
  - il existe en fait plusieurs autres façon de construire l'ensemble taboue...

D'autres améliorations: *beam search*

- **Idée**: plutôt que maintenir un seul noeud solution *n*, en pourrait maintenir un ensemble de *k* noeuds différents
  1. on commence avec un ensemble de *k* noeuds choisis aléatoirement
  2. à chaque itération, tous les successeurs des *k* noeuds sont générés
  3. on choisit les *k* meilleurs parmi ces noeuds et on recommence

- Cet algorithme est appelé ***local beam search*** (exploration locale par faisceau)
  - à ne pas confondre avec *tabu search*
  - variante *stochastic beam search* : plutôt que prendre les *k* meilleurs, on assigne une probabilité de choisir chaque noeud, même s'il n'est pas parmi les k meilleurs (comme dans *simulated annealing*)

Algorithme génétique

- Très similaire à *local* ou *stochastic beam-search*
- **Algorithme génétique**
  - on commence aussi avec un ensemble de *k* noeuds choisis aléatoirement : cet ensemble est appelé une **population**
  - un successeur est généré en combinant deux parents
  - un noeud est représenté par un mot (chaîne) sur un alphabet (souvent l'alphabet binaire)
  - la fonction d'évaluation est appelée **fonction de *fitness*** (fonction d'adaptabilité, de survie)
  - la prochaine génération est produite par **(1) sélection, (2) croisement et (3) mutation**

Algorithme génétique

- Inspiré du processus de l'évolution naturelle des espèces :
  - après tout l'intelligence humaine est le résultat d'un processus d'évolution sur des millions d'années :
    - théorie de l'évolution (Darwin, 1858)
    - théorie de la sélection naturelle (Weismann)
    - concepts de génétiques (Mendel)
  - la simulation de l'évolution n'a pas besoin de durer des millions d'années sur un ordinateur

Algorithme génétique

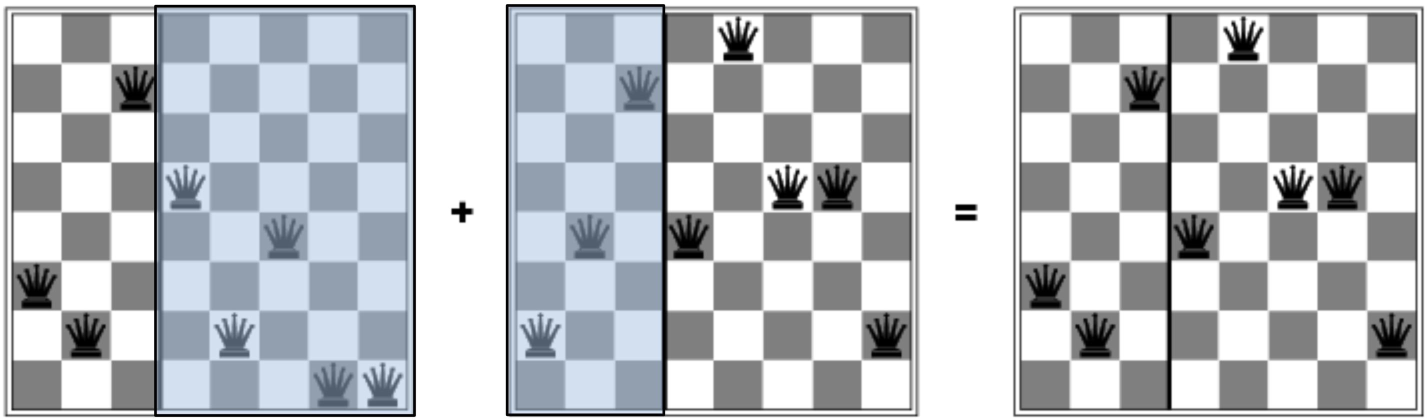
- On représente l'espace des solutions d'un problème à résoudre par une population (ensemble de **chromosomes**).
  - un chromosome est une chaîne de bits (**gènes**) de taille fixe
  - par exemple : 101101001
- Une population génère des enfants par un ensemble de procédures simples qui manipulent les chromosomes
  - **croisement de parents**
  - **mutation d'un enfant généré**
- Les enfants sont conservés en fonction de leur **adaptabilité** (*fitness*) déterminée par une fonction d'adaptabilité donnée *F*(*n*)

Algorithme génétique

Algorithme ALGORITHME-GÉNÉTIQUE(k, *nb_iterations*) // *cette variante maximise*

1. *population* = ensemble $\{n_1, n_2, ..., n_k\}$ généré aléatoirement de k chromosomes
2. pour t = 1 ... *nb_iterations*
   3. *nouvelle_population* = {}
   4. pour i = 1 ... k
      5. n = chromosome pris dans *population* avec probabilité proportionnelle à $F(n)$
      6. n' = chromosome différent pris dans *population* – { n } de la même façon
      7. n^* = résultat du croisement entre n et n'
      8. avec petite probabilité, appliquer une mutation à n^*
      9. ajouter n^* à *nouvelle_population*
   10. *population* = *nouvelle_population*
11. retourner chromosome n dans *population* avec valeur de $F(n)$ la plus élevée

Croisement : exemple avec 8 reines



672**47588** + **752**51448 = 67251448

Exemple avec 8 reines

- Fonction de *fitness* : nombre de paires de reines qui ne s'attaquent pas (min = 0, max = 8 × 7/2 = 28)
- Probabilité de sélection du premier chromosome : **pourcentage de *fitness* parmi la population**
  - 24/(24+23+20+11) = 31%
  - 23/(24+23+20+11) = 29%
  - 20/(24+23+20+11) = 26%
  - 11/(24+23+20+11) = 14%

Exemple avec 8 reines

- Plusieurs autres choix de processus de sélection seraient valide
  - ex.: on pourrait ne jamais sélectionner les chromosomes faisant partie des 25% pires
- **L'important est que la probabilité qu'un chromosome *n* soit choisi augmente en fonction de sa valeur *F*(*n*)**

Autre Exemple

- Calculer le maximum de la fonction $F(n) = 15n - n^2$
- Supposons n entre [0, 15] :
  - on a besoin de seulement 4 bits pour représenter la population

Integer	Binary code	Integer	Binary code	Integer	Binary code
1	0 0 0 1	6	0 1 1 0	11	1 0 1 1
2	0 0 1 0	7	0 1 1 1	12	1 1 0 0
3	0 0 1 1	8	1 0 0 0	13	1 1 0 1
4	0 1 0 0	9	1 0 0 1	14	1 1 1 0
5	0 1 0 1	10	1 0 1 0	15	1 1 1 1

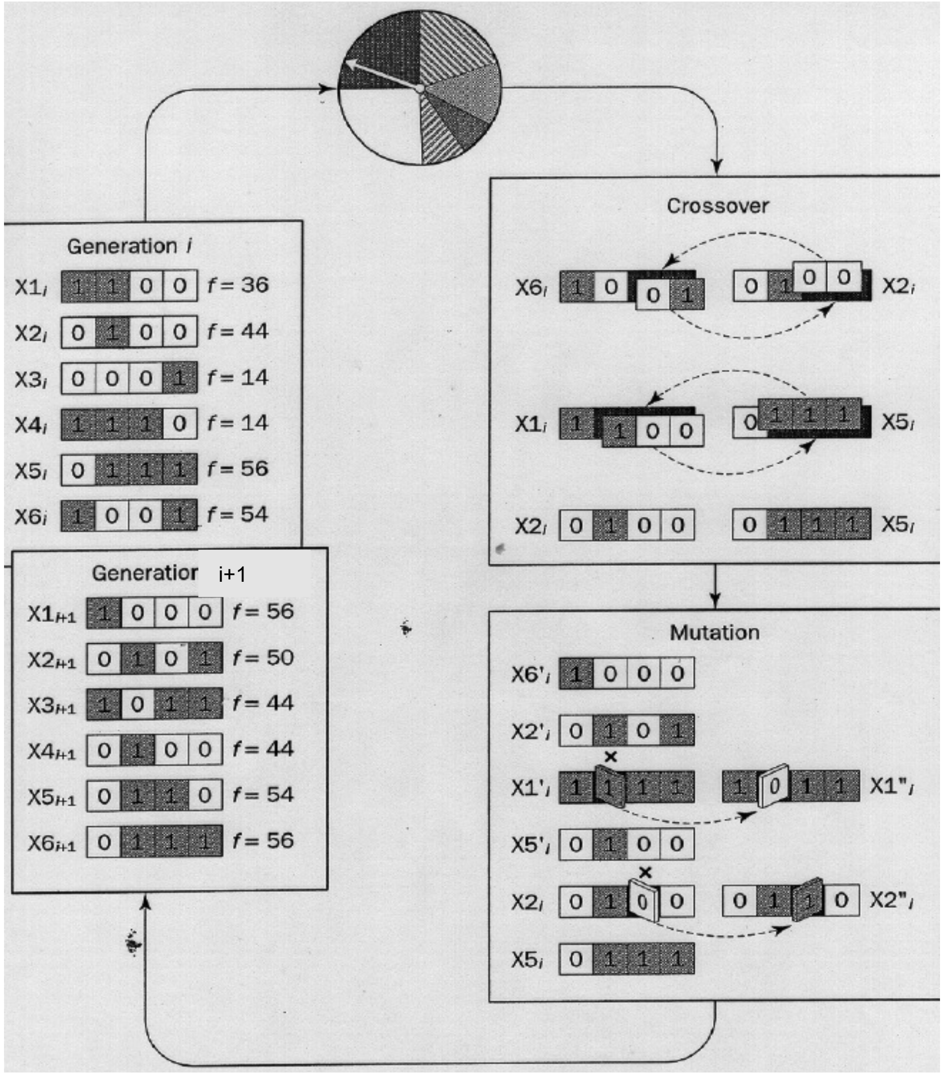
[Michael Negnevitsky. Artificial Intelligence. Addison-Wesley, 2002. Page 222.]

Autre Exemple (suite)

- Fixons la taille de la population à 6
- Et la probabilité de mutation à 0.001
- La fonction d'adaptabilité à $F(n) = 15n - n^2$
- L'algorithme génétique initialise les 6 chromosomes de la population en les choisissant au hasard

Chromosome label	Chromosome string	Decoded integer	Chromosome fitness	Fitness ratio, %
X1	1 1 0 0	12	36	16.5
X2	0 1 0 0	4	44	20.2
X3	0 0 0 1	1	14	6.4
X4	1 1 1 0	14	14	6.4
X5	0 1 1 1	7	56	25.7
X6	1 0 0 1	9	54	24.8

Autre Exemple (illustration des étapes)



Programmation génétique

- Même principes que les algorithmes génétiques sauf que les populations sont des programmes au lieu des chaînes de bits

[Michael Negnevitsky. Artificial Intelligence. Addison-Wesley, 2002. Page 247.]

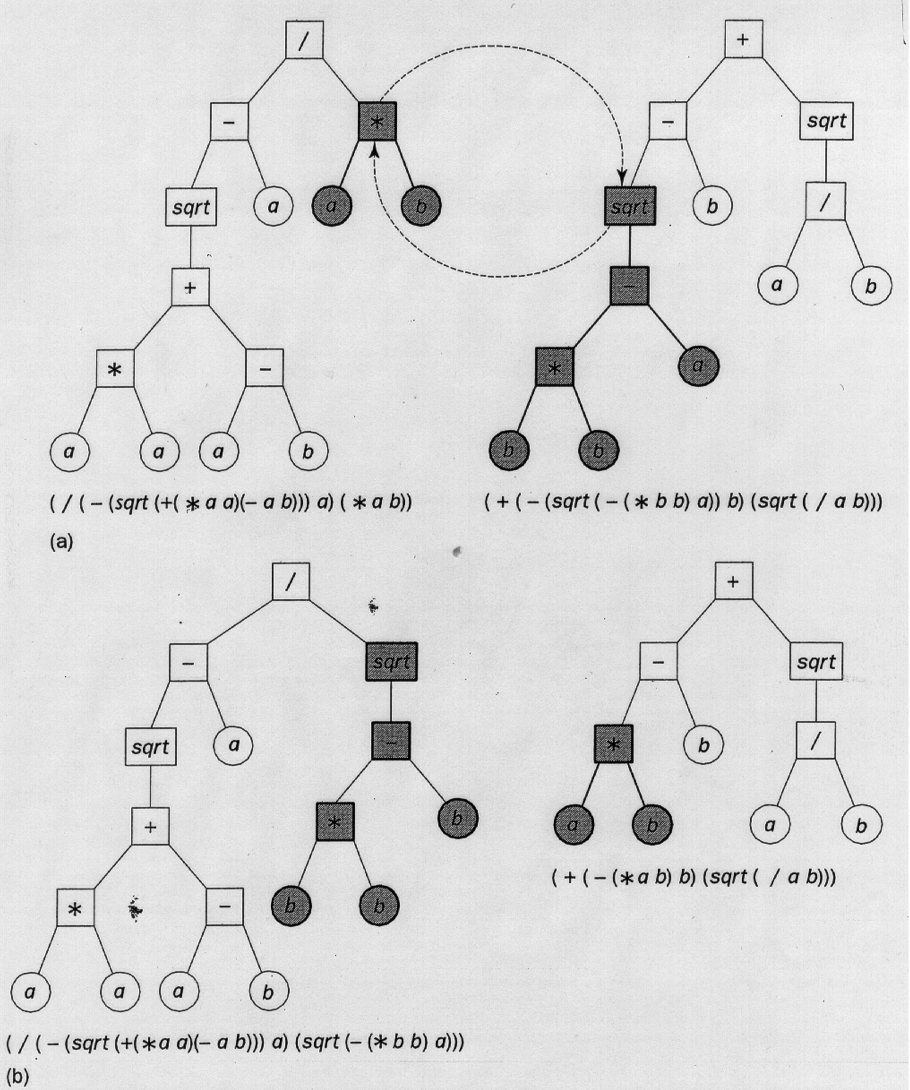


Figure 7.15 Crossover in genetic programming: (a) two parental S-expressions; (b) two offspring S-expressions

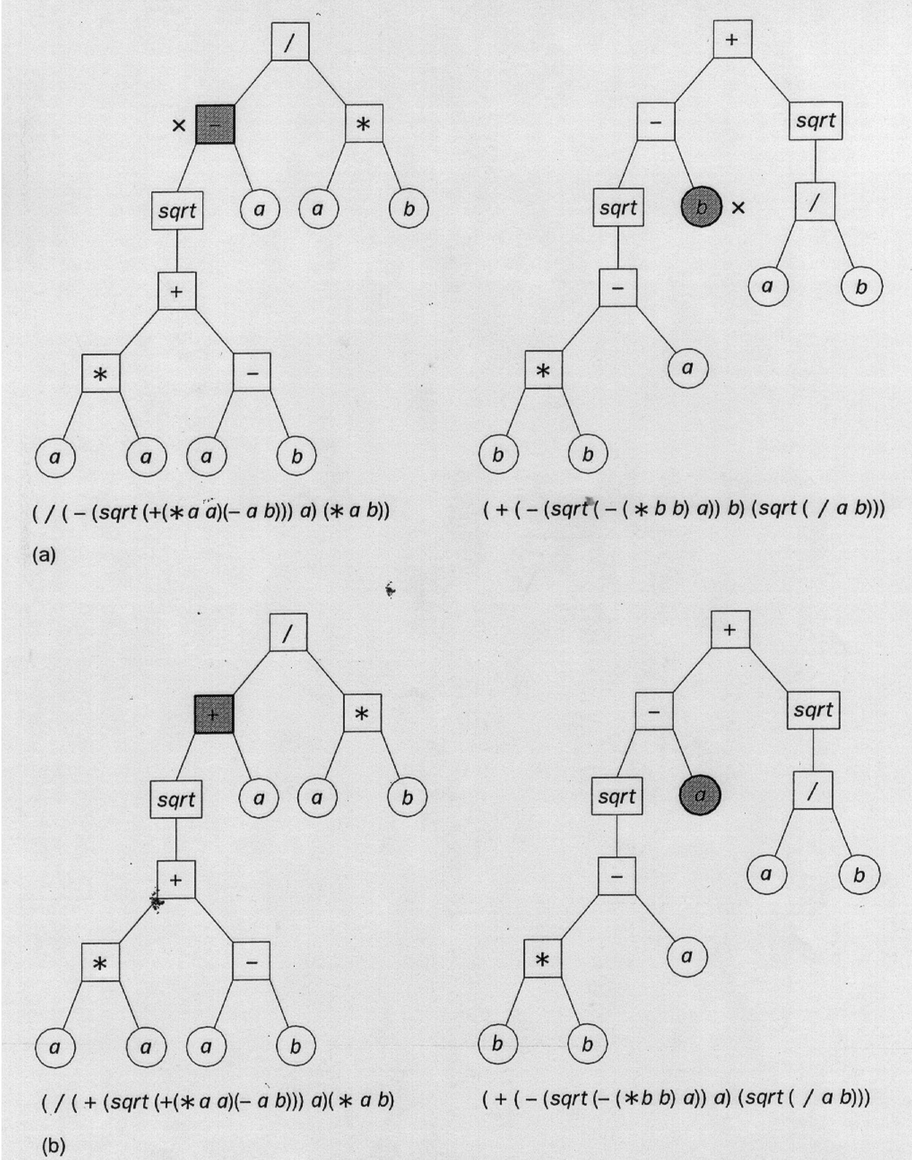


Figure 7.16 Mutation in genetic programming: (a) original S-expressions; (b) mutated S-expressions

Objectifs du cours

Algorithmes et concepts

agents intelligents

recherche locale

appropri√© si :
- ‚úì le chemin qui m√®ne √Ý la solution n‚Äôest pas important
- ‚úì l‚Äôespace de recherche trop grand pour A*
- ‚úì on peut se satisfaire d‚Äôune solution qui n‚Äôest pas optimale

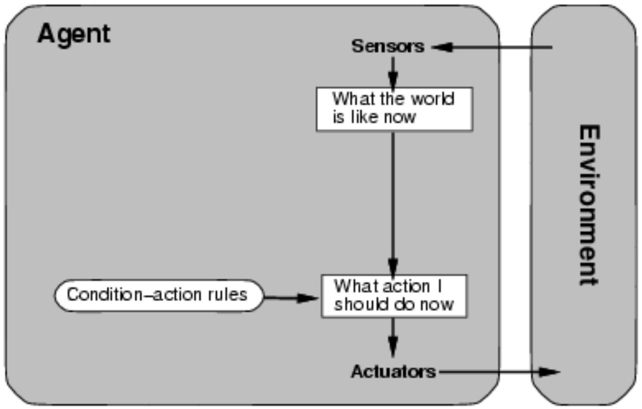
- ‚úì servent √Ý optimiser une fonction
- ‚úì retourne une solution qui optimise cette fonction

recherche heuristique

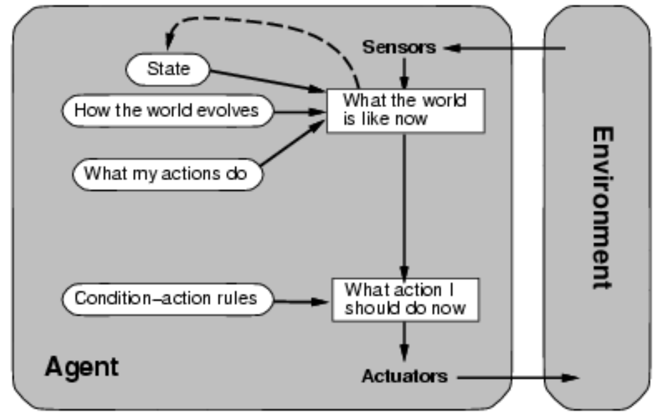
appropri√© si :
- ‚úì on a besoin du chemin vers la solution
- ‚úì on veut des garantis sur l‚Äôoptimalit√© de la solution

Recherche locale : pour quel type d'agent?

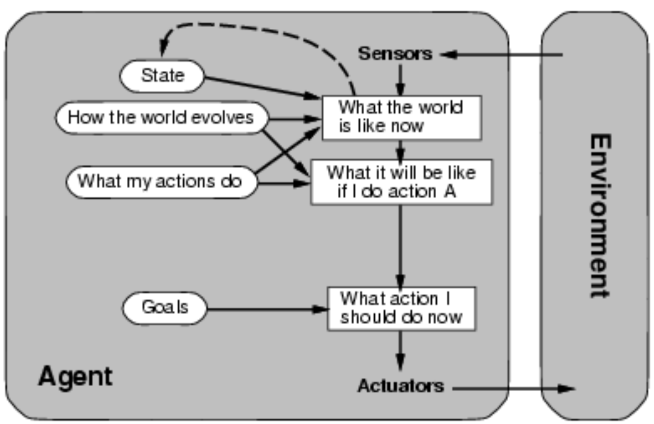
Simple reflex



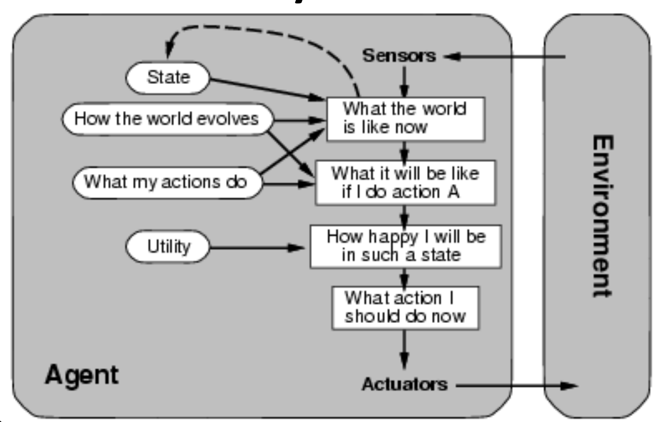
Model-based reflex



Goal-based



Utiliy-based



Conclusion

- La recherche locale est parfois une alternative plus intéressante que la recherche heuristique

- J'ai ignoré le cas où on a également une fonction but *goal*(*n*)
  - dans ce cas, lorsqu'on change la valeur de n, on arrête aussitôt que *goal*(*n*) est vrai
    - » ex.: *goal*(*n*) est vrai si n est un optimum global de *F*(*n*)

- La recherche locale va s'avérer utile plus tard dans le cours
  - satisfaction de contraintes
  - apprentissage par renforcement

Vous devriez être capable de...

- Décrire ce qu'est la recherche locale en général
- Décrire les algorithmes :
  - *hill-climbing*
  - *simulated annealing*
  - algorithme génétique
- Savoir simuler ces algorithmes
- Connaître leurs propriétés (avantages vs. désavantages)