

# Optimisation stochastique - 2

Late Acceptance Hill Climbing



# Late Acceptance Hill Climbing



# Late Acceptance Hill Climbing (LAHC)

- Proposée par Edmond K. Burke & Youri Bykov (2008) ;
- Algorithme mono-agent, adapté aux problèmes discrets ;
- Utilisation d'une **mémoire de scores** de candidats déjà visités ;
- Basé sur l'heuristique Greedy Hill Climbing : "Accepter un nouveau candidat s'il est meilleur que le candidat courant" ;
- "Accepter un nouveau candidat s'il n'est pas pire qu'un précédent candidat d'il y a un certain nombre d'itérations" ;
- Un unique paramètre : la taille de la mémoire.

# Greedy Hill Climbing : principe

- Heuristique de recherche locale ;
- Le candidat en cours est remplacé par son meilleur voisin selon :
  - Pour un candidat quelconque,
  - On évalue tous ses voisins,
  - On le remplace par celui qui l'améliore le plus,
  - On itère jusqu'à ce qu'il n'y ait plus d'amélioration.
- Permet de trouver un *optimum local*.

# Greedy Hill Climbing : illustration

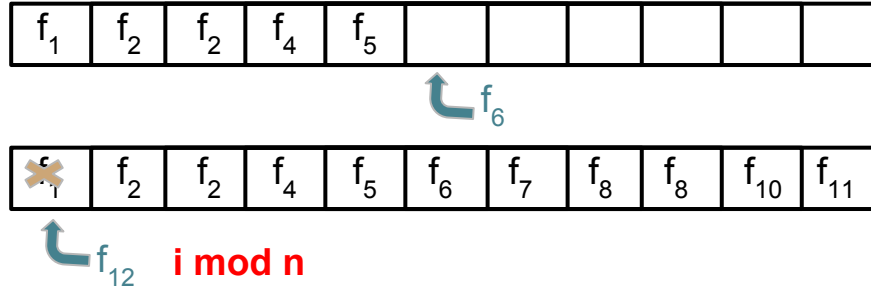


# LAHC : principe

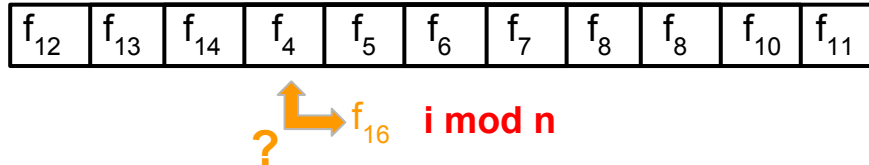
- Le candidat courant en cours génère un nouveau voisin ;
- On ajoute le coût du candidat courant à la fin de la mémoire (modulo  $n$ ) ;
- On évalue le coût de ce voisin et on le compare à la performance du  $n^e$  candidat passé ( $n$  : taille de la mémoire) ;
- S'il l'améliore, alors le voisin devient le candidat courant.

# LAHC : illustration

- Le candidat courant est ajouté dans la mémoire ;



- La comparaison se fait entre l'évaluation du voisin et la  $n^e$  ancienne évaluation.



# Algorithme du LAHC (minimisation)

```
X, un candidat, f(X) son évaluation
fmin <- f(X), Xmin <- X, tabMemoire <- [fmin | ... | fmin]
n <- 1
Tant que non critèreConvergence()
  Xvois <- genereVoisin(X)
  fmem <- tabMemoire[n % tailleMem]
  Si fmem ≥ f(Xvois) alors
    X <- Xvois
  Fin si
  tabMemoire[n % tailleMem] <- f(X)
  Si fmin > f(Xvois) alors
    Xmin <- Xvois
    fmin <- f(Xvois)
  Fin si
  n++
Fin tant que
```



# LAHC : critères de convergence

Permet de terminer l'algorithme selon plusieurs conditions :

- Le nombre d'évaluations atteint une limite ;
- Il n'y a pas eu d'amélioration depuis un certain nombre d'itérations.

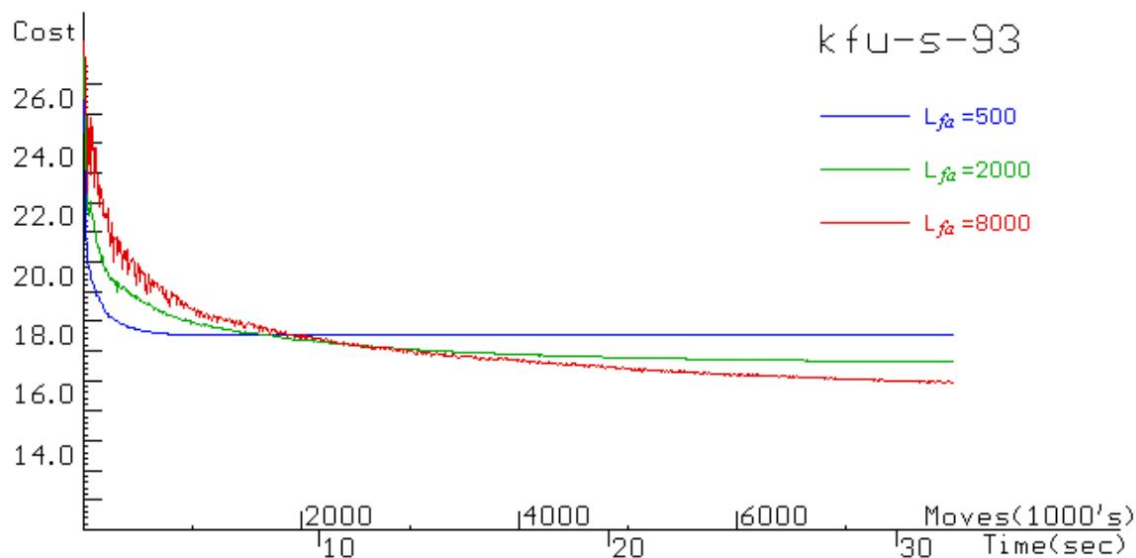
# LAHC : gestion de la mémoire

La mémoire des déplacements précédents est gérée selon :

- Structure tableau contenant  $n$  évaluations de candidats visités ;
- La longueur de la mémoire est un paramètre critique :
  - $n$  petit : peu de valeurs retenues, comportement proche du hill-climbing d'origine = convergence prématurée,
  - $n$  grand : beaucoup de mouvements aléatoires diversifie la recherche.
- On fixe, à l'initialisation, toutes les valeurs à celle d'origine. Cela empêche des mouvements trop aléatoires lors des premières itérations.

# LAHC : gestion de la mémoire

Influence de la taille de la mémoire sur le comportement dans le cas d'un problème de planification :



# LAHC : perturbation

- Problématique identique à celle du recuit simulé ;
- Génère un nouveau candidat au **voisinage** du candidat en cours ;
- Influence de la distance entre ces deux solutions (hypervolume du voisinage) ;
- Spécifique au problème à résoudre (discret/continu).

# LAHC : voisinage

- Ensemble des transformations locales appliquées à une solution ;
- Selon la taille ou le type du problème, il peut être très grand ;
- Problème d'échantillonnage aléatoire de l'ensemble : "manquer" l'optimum.

# LAHC : améliorations

Comparaison à un candidat passé :

- Dépend de la taille de la mémoire (version classique) ;
- On compare à l'optimum de la mémoire ;
- On compare à un membre aléatoire de la mémoire ;
- On compare à une moyenne/médiane des valeurs de la mémoire.

# LAHC : améliorations

Le critère d'acceptation est :

- Déterministe :
  - évaluation inférieure ou égale au candidat de la mémoire (par défaut),
  - évaluation strictement inférieure au candidat de la mémoire,
  - évaluation inférieure ou égale au candidat de la mémoire ou bien au candidat courant,
  - ...
- Probabiliste :
  - Metropolis-like :  $p = e^{-(c_{\text{vois}} - c_{\text{cur}}) / (\text{mem}[i] - c_{\text{cur}})}$ ,
  - linéaire :  $p = 1 - (c_{\text{vois}} - c_{\text{cur}}) / [2(\text{mem}[i] - c_{\text{cur}})]$ .

# LAHC : améliorations

Mémorisation de l'information :

- Descente seulement :
  - on remplace dans la liste si  $c < \text{mem}[i]$  ;
- Stratégie de réchauffement :
  - après convergence, on ré-hausse les valeurs de la mémoire de  $p\%$  ( $p=20$ ).





That's all folks !