Sommaire

4.3.1 Métaheuristiques

Problème d'optimisation difficile

$\min_x f(x)$ → problème d'optimisation (PO)

- On distingue 2 types de problèmes difficiles.
  Problème **discret** → nombre exponentiel de combinaisons à explorer
  Problème **continu** → minima locaux, aucune caractérisation du minimum global

- On ne peut pas trouver la solution exacte en un temps de calcul raisonnable.
  Il faut se satisfaire d'une **solution approchée** « suffisamment bonne ».

- Une métaheuristique est une méthode de résolution approchée mimant un processus physique.
  Recuit simulé → thermodynamique
  Algorithme évolutionnaire → sélection naturelle
  Essaim particulaire → mouvement collectif d'essaims
  Colonie de fourmis → mouvement organisé de fourmis

- Heuristique = méthode empirique spécialisée à un problème particulier
  Métaheuristique = principe général applicable à différents problèmes
  → nécessite un travail d'adaptation pour chaque problème

4.3.1 Métaheuristiques

Principales métaheuristiques

- Algorithmes

Algorithme	Acronyme	Type de problème	Extensions
Recuit simulé	SA	Discret (chemin)	Continu
Recherche tabou	TS	Discret (affectation)	
Essaims particulaires	PSO	Continu	Discret
Colonies de fourmis	ACO	Discret (chemin)	
Algorithme évolutionnaire	GA	Discret – Continu	
Adaptation de covariance	CMA	Continu	
Affine shaker	AS	Continu	Discret
Réseaux de neurones	NN	Identification	

- Applications
  - Optimisation **difficile** → minima locaux, minimum global
  - Optimisation **multi-objectifs** → front de Pareto
  - Identification de modèles → apprentissage + optimisation

4.3.1 Métaheuristiques

Principes généraux

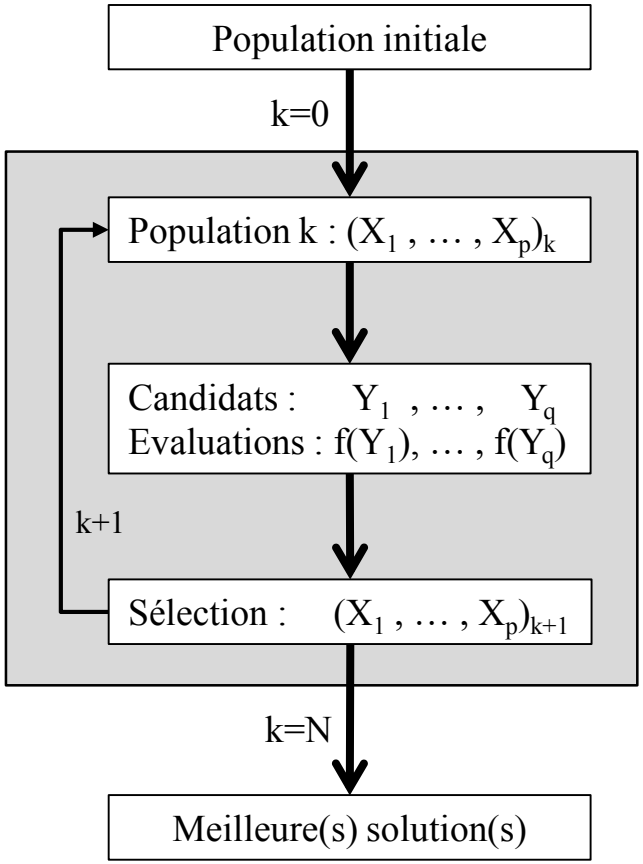
$\min_x f(x)$ → trouver le minimum global ou des minima locaux

- Population de p individus : X_i , $f(X_i)$
  Ordre de grandeur : p = 1 à 100 (selon algorithme)

- Exploration du voisinage : q candidats Y_j , $f(Y_j)$
  Perturbations aléatoires
  → **intensification** pour affiner les minima trouvés
  → **diversification** pour trouver d'autres minima

- Règles de sélection
  Acceptation aléatoire de dégradations
  → échappement de minima locaux

- Arrêt sur nombre d'essais N
  → généralement très grand



4.3.2 Recuit simulé

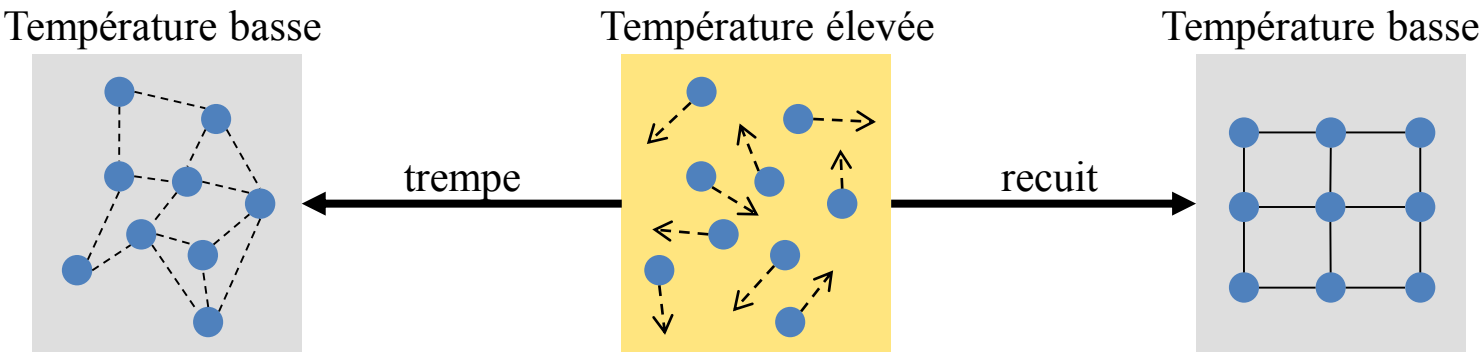
Principe

La méthode du recuit simulé (« Simulated Annealing », Kirkpatrick et co 1983) s'inspire de la thermodynamique d'un système de particules.

- Le recuit est un processus utilisé en métallurgie pour obtenir un alliage sans défaut.
  A très haute température, le métal est à l'état liquide et les atomes se déplacent librement.
  On procède à un refroidissement pour revenir à l'état solide.

- Si le refroidissement est rapide (**trempe**), les atomes se figent dans un état désordonné.
  L'alliage obtenu a une structure irrégulière et présente des défauts (énergie élevée).

- Si le refroidissement est lent (**recuit**), les atomes se réorganisent de façon régulière.
  L'alliage obtenu a une structure cristalline sans défaut (énergie minimale).



4.3.2 Recuit simulé

Niveau d'énergie

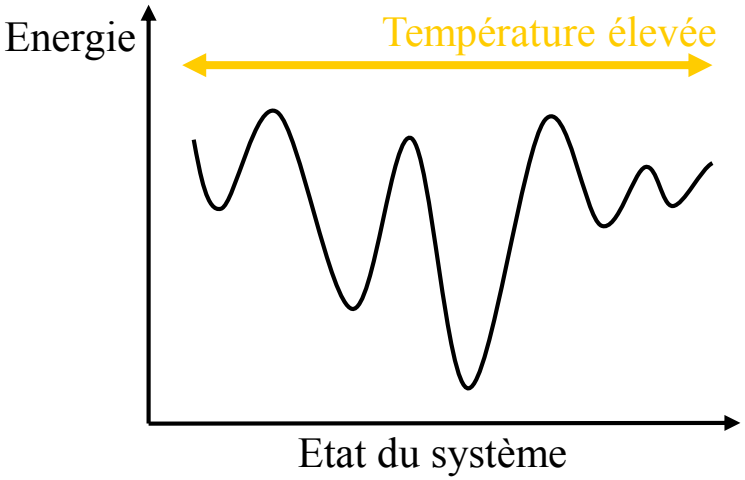
- Le niveau d'énergie E du système dépend de la disposition des atomes.

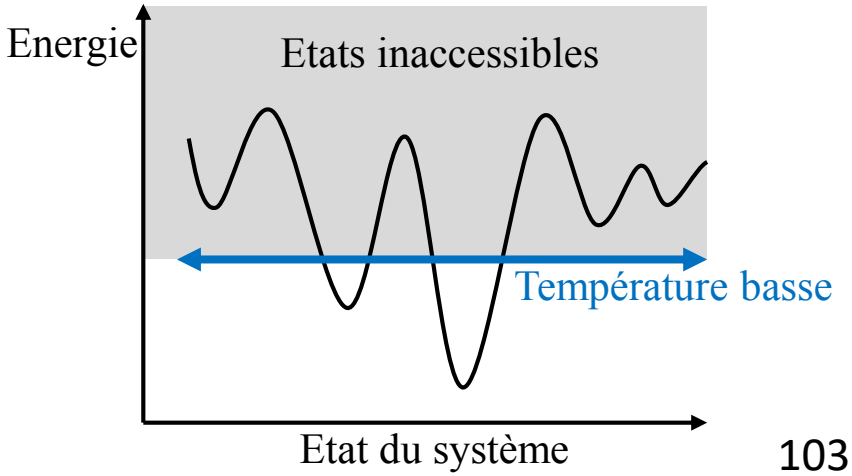
- A la température T, la probabilité $P_T(E)$ que l'énergie du système soit égale à E est

$$P_T(E) = \alpha e^{-\frac{E}{kT}}$$

→ loi de Gibbs (k = constante de Boltzmann)
(α= constante de normalisation : $\alpha = \int_E e^{-\frac{E}{kT}} dE$)

- A température élevée, tous les états d'énergie ont quasiment la même probabilité.
  A température basse, les états d'énergie élevée ont une probabilité quasiment nulle.





4.3.2 Recuit simulé

Algorithme de Metropolis

L'algorithme de Metropolis (1953) simule l'évolution du système vers l'équilibre.
L'équilibre thermodynamique correspond à l'état d'énergie minimale $\rightarrow \min_x E(x)$

- On part d'un état initial du système noté x_0 (= disposition des atomes) d'énergie $E_0 = E(x_0)$.

- On perturbe de façon aléatoire l'état du système : $x_0 \rightarrow x = x_0 + \delta x$
  $E_0 \rightarrow E = E_0 + \delta E$

- Le nouvel état x est accepté avec la probabilité $\begin{cases} P = e^{-\frac{\delta E}{T}} & \text{si } \delta E \geq 0 \quad \text{(augmentation d'énergie)} \\ P = 1 & \text{si } \delta E < 0 \quad \text{(diminution d'énergie)} \end{cases}$

- Le **paramètre de température T** règle le niveau d'acceptation des remontées d'énergie.

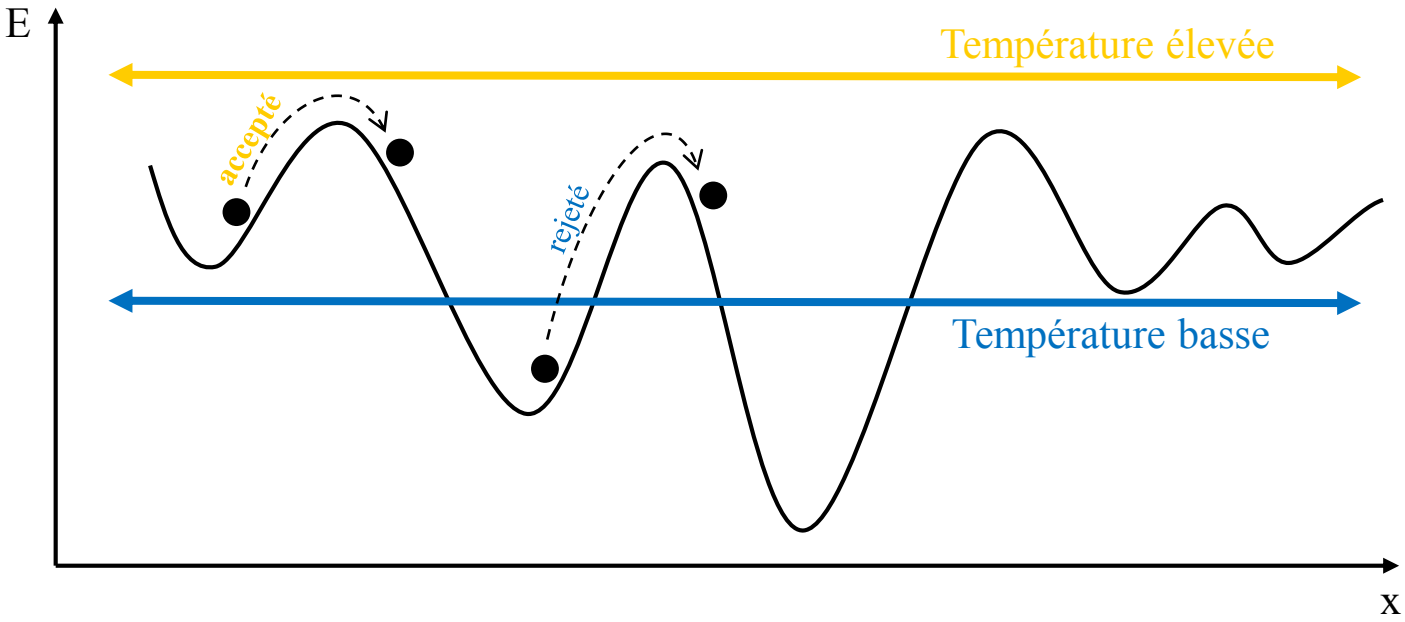
- A température élevée, un état d'énergie supérieure peut être accepté avec une probabilité forte.
  $\rightarrow$ Le système peut explorer l'ensemble des états possibles.
  A température basse, un état d'énergie supérieure est rejeté de façon quasi-systématique.
  $\rightarrow$ Le système se stabilise dans un état de basse énergie.

4.3.2 Recuit simulé

Convergence

- L'acceptation de solutions moins bonnes permet d'explorer l'espace des solutions. Le système peut s'extraire d'un minimum local si le nombre d'essais est suffisant.

- La convergence vers un minimum global nécessite :
  - une température initiale élevée → pour autoriser l'accès à tous les états
  - une décroissance de température lente → pour échapper au minima locaux
  - un nombre d'essais élevé

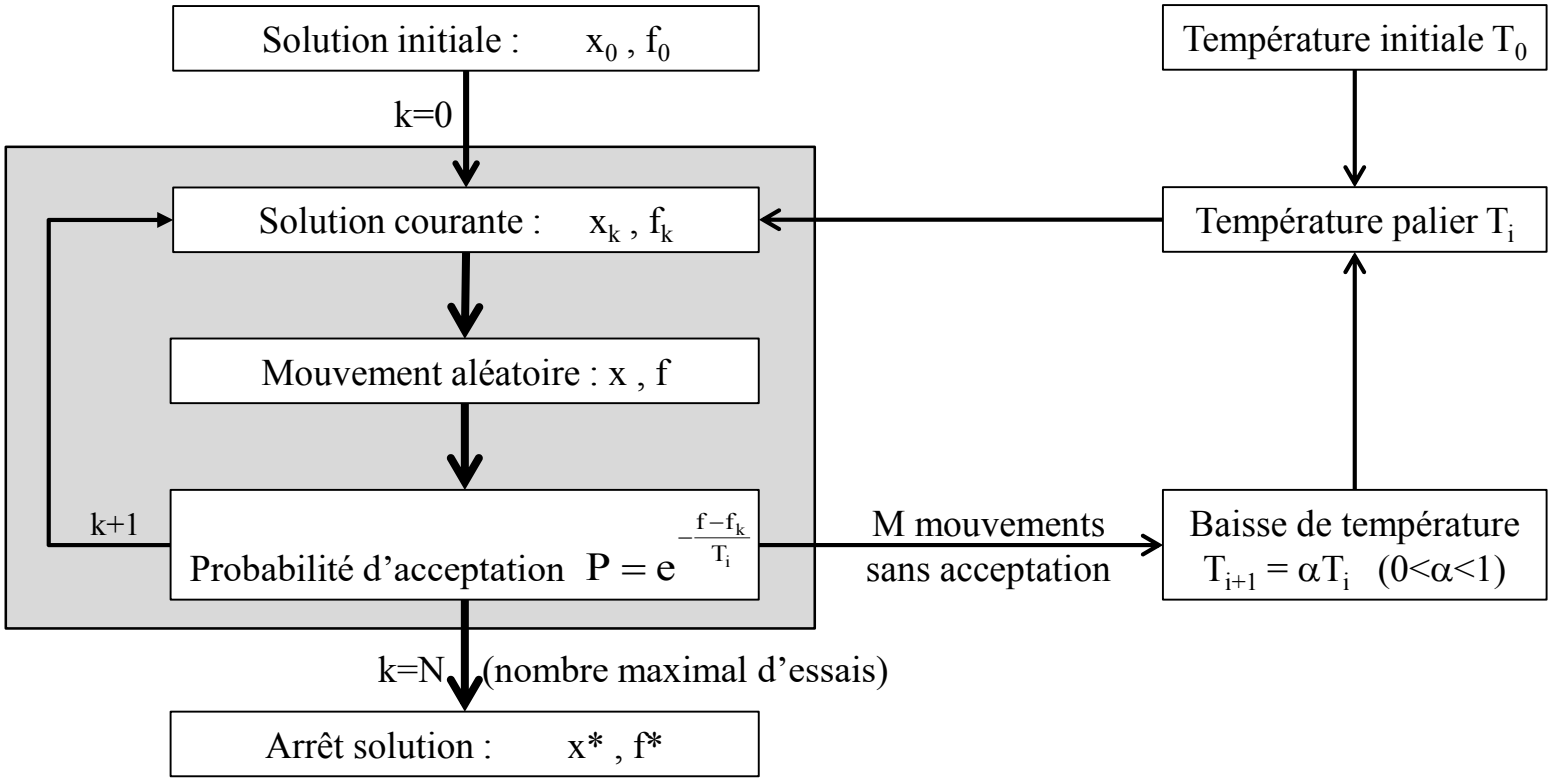


4.3.2 Recuit simulé

Algorithme

On applique l'algorithme de Metropolis à un problème de minimisation.

$$\min_x f(x) \quad \rightarrow \quad \begin{cases} \text{variables } x = \text{« état du système »} \\ \text{fonction } f = \text{« énergie du système »} \end{cases}$$



4.3.2 Recuit simulé

Réglages

- La **température initiale** T_0 doit être élevée pour accepter quasiment tous les mouvements.
  On choisit par exemple T_0 de façon à accepter 90% des mouvements aléatoires à partir de x_0.

- La **décroissance de température** doit être lente pour éviter le blocage dans un minimum local.
  On choisit par exemple une décroissance géométrique avec α=0.999 et M assez grand.

- L'algorithme s'arrête sur le nombre d'essais N ou sur un nombre de paliers sans amélioration.
  La qualité de la solution et le temps de calcul dépendent des réglages T_0, α, M, N.
  → à adapter expérimentalement au cas par cas

Mouvements aléatoires

- Les perturbations doivent générer une solution « localement proche » de la solution courante.
  Le choix des perturbations aléatoires dépend de la nature du problème (discret, continu).
  → Il faut définir le « **voisinage** » de la solution courante.

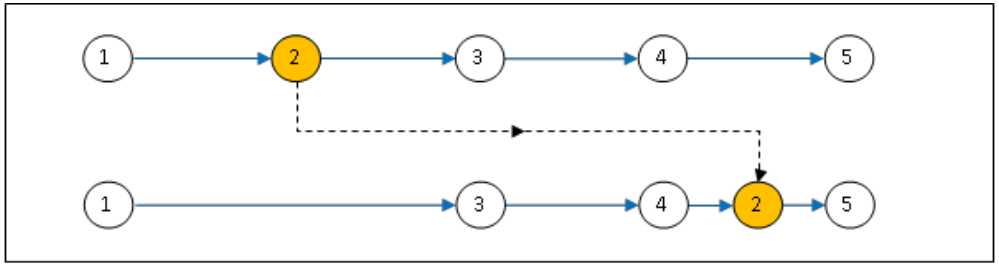
- Application au PVC : les variables sont les numéros des nœuds le long du chemin.
  On définit 3 types de perturbations élémentaires : insertion, échange, permutation.
  → Tirage aléatoire : - de la nature de la perturbation
  - des nœuds auxquels la perturbation est appliquée

4.3.2 Recuit simulé

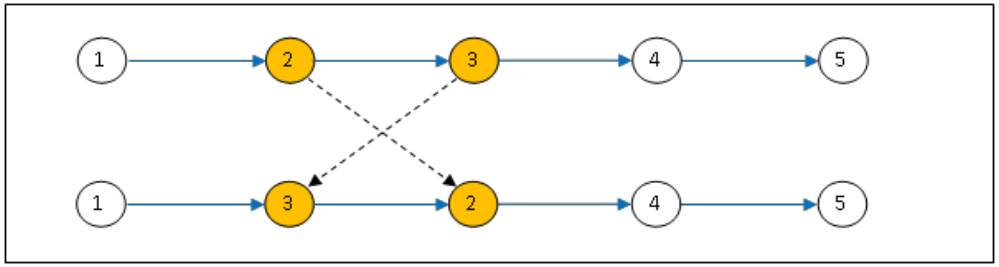
Voyageur de commerce

Le voisinage d'une solution est défini par 3 mouvements élémentaires.

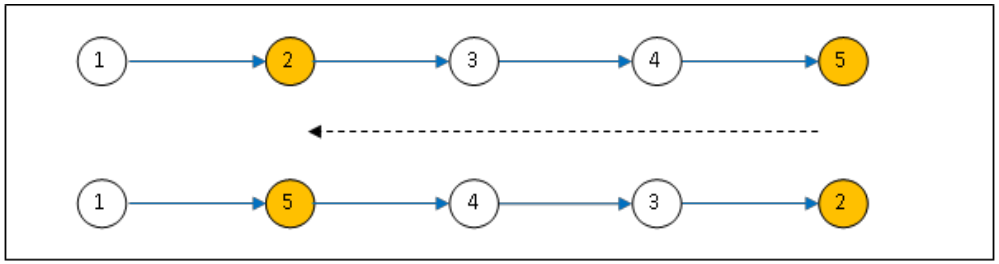
- **Insertion**
  2 est inséré après 4.



- **Echange**
  2 et 3 sont échangés.



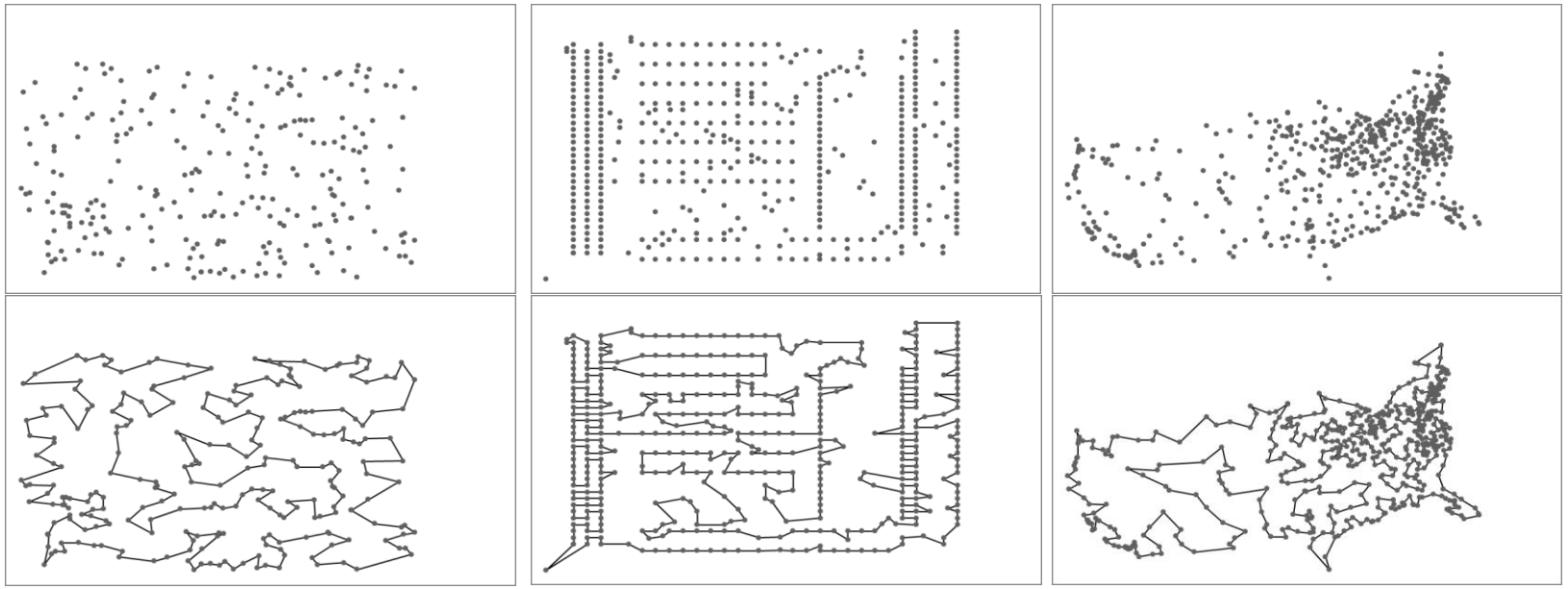
- **Permutation**
  Le chemin de 2 à 5 est inversé.



4.3.2 Recuit simulé

Voyageur de commerce

Defi250	Pcb442	Att532
250 villes fictives	Circuit imprimé à 442 trous	532 villes des USA



Coût = 11,809	Coût = 50778	Coût = 27686
Essais : 5.10^8	Essais : 8.10^8	Essais : 2.10^9
Acceptés : 1.10^4	Acceptés : 2.10^4	Acceptés : 2.10^4

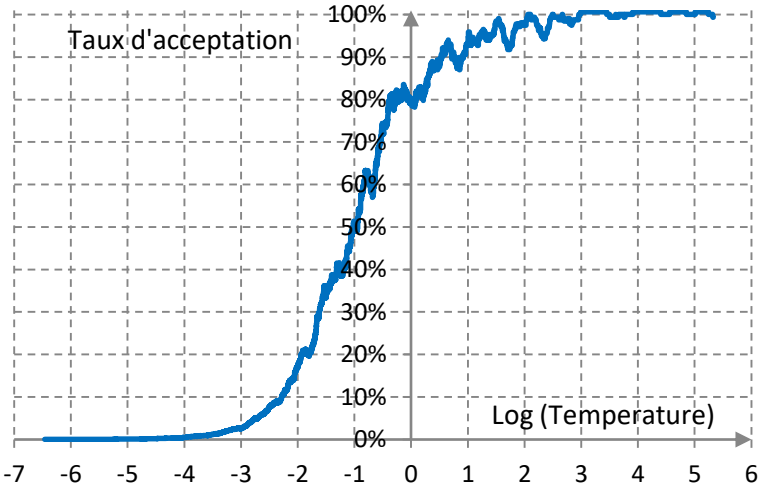
4.3.2 Recuit simulé

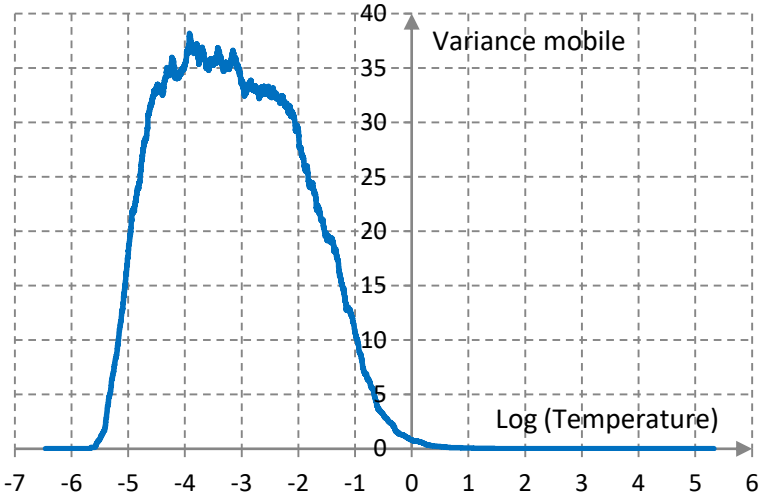
Voyageur de commerce

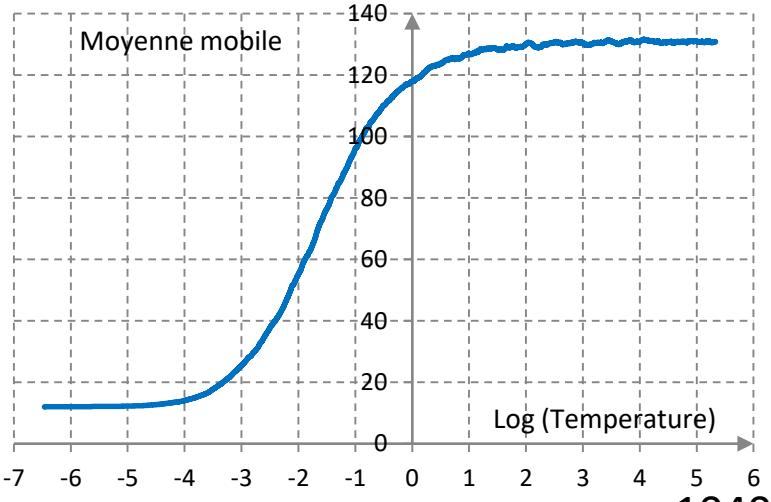
- Indicateurs de convergence
  - taux d'acceptation
  - moyenne mobile (par palier de température)
  - variance mobile

- Température de stabilisation
  → Variance maximale

- Exemple : problème Défi250







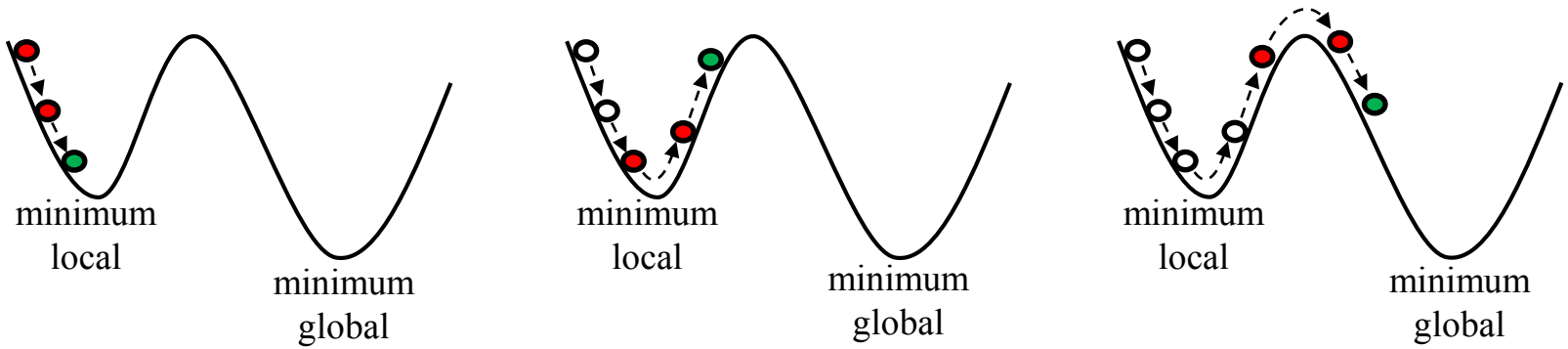
4.3.3 Recherche tabou

Principe

La méthode de recherche avec tabou (« Tabu Search », Glover 1986) est une méthode locale avec une mémoire des solutions visitées.

- La liste tabou mémorise les p dernières solutions visitées (= liste des solutions « taboues »)
  ou les p derniers mouvements pour limiter l'encombrement en mémoire.
  La taille de la liste est généralement de l'ordre de p=10 → mémoire à court terme

- On cherche une amélioration dans un voisinage (à définir) de la solution courante
  en excluant les solutions précédentes pour forcer la sortie d'un minimum local.
  Solution courante (itération k) : x_k
  Solution précédentes (itérations k–p à k–1) : $x_{k-p}, \ldots, x_{k-2}, x_{k-1}$

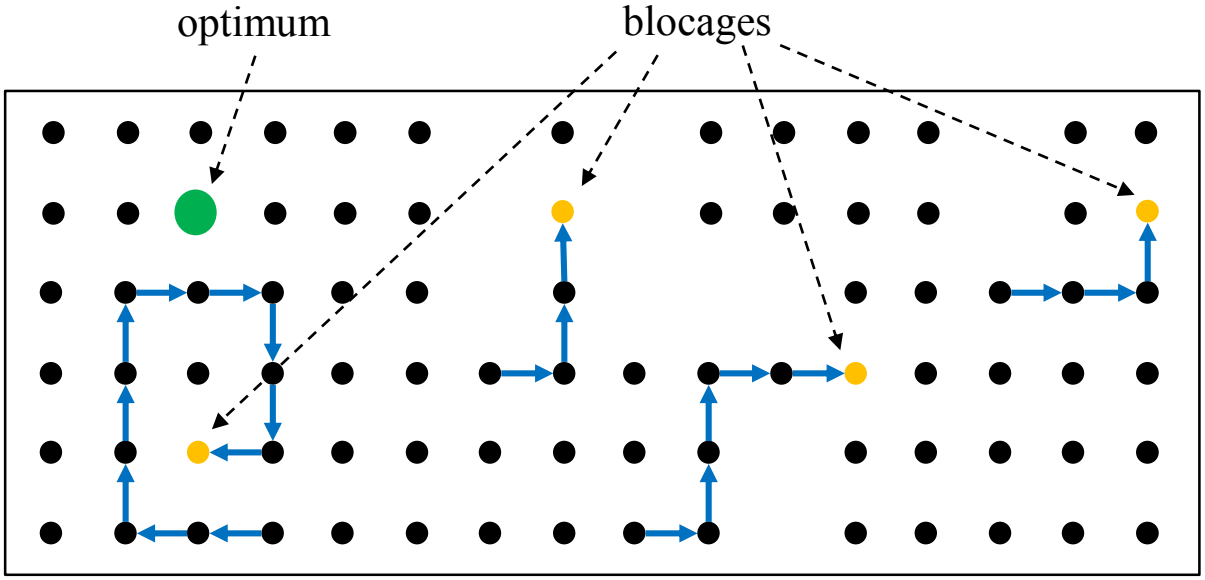


4.3.3 Recherche tabou

Liste tabou

- La liste tabou interdit temporairement le retour à des solutions visitées.

- La **taille** de la liste tabou et la **durée** des interdictions sont à définir.
  Une liste trop grande peut déconnecter la solution courante de la solution optimale.



La solution optimale (en vert) ne peut plus être atteinte à partir des points en orange si les interdictions ne sont pas levées → blocage de la recherche.

4.3.3 Recherche tabou

Voisinage

- Le voisinage V_k de la solution courante x_k est l'ensemble des solutions accessibles à partir de x_k par des modifications (ou mouvements) élémentaires.
  La définition du voisinage est à adapter selon le problème.

- De nombreux problèmes se ramènent à la recherche d'une **permutation optimale**.
  Pour ce type de problème, on peut définir le voisinage par 2 mouvements élémentaires.

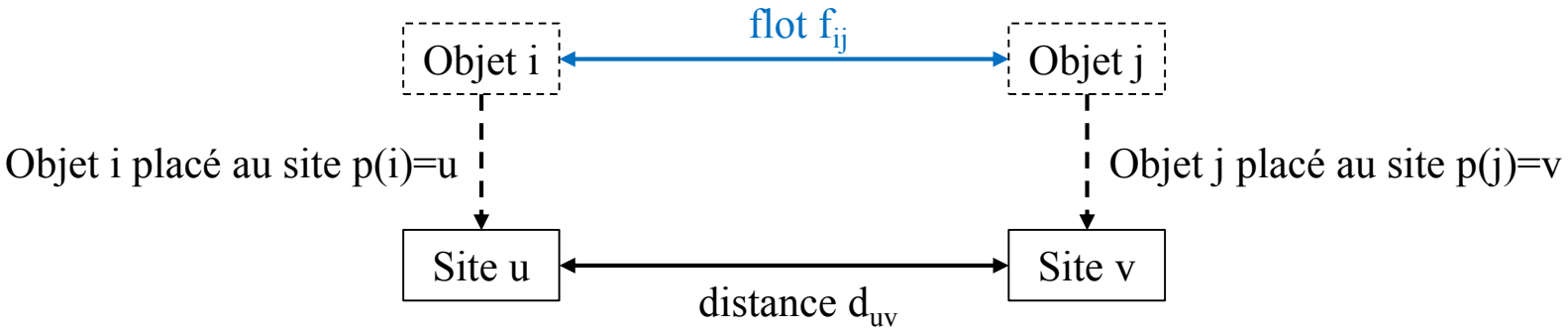
- **Echange**

- **Insertion**

4.3.3 Recherche tabou

Affectation quadratique

Le problème d'affectation quadratique (« Quadratic Assignment Problem » ou QAP) consiste à affecter n objets sur n sites.



- Le flot entre les objets i et j est : f_{ij} → matrice F des flots
- La distance entre les sites u et v est : d_{uv} → matrice D des distances
- Le coût est le produit flot × distance : $f_{ij}d_{uv}$ = coût du flot entre l'objet i placé au site u et l'objet j placé au site v

- Une solution est représentée par une **permutation P** donnant les affectations des n objets.
  P = (p(1) ,…, p(i) ,…, p(j) ,…, p(n)) avec p(i) = numéro du site où est placé l'objet i

- On cherche la permutation P minimisant le coût total : $C_P = \sum_{i,j=1 \text{ à } n} f_{ij} d_{p(i)p(j)}$

4.3.3 Recherche tabou

Affectation quadratique

- Le voisinage de la solution courante consiste à échanger les positions de 2 objets.
  Solution courante : $P = (p(1) ,\ldots, p(u) ,\ldots, p(v) ,\ldots, p(n))$
  Solution voisine : $P' = (p(1) ,\ldots, p(v) ,\ldots, p(u) ,\ldots, p(n))$
  obtenue en échangeant les positions des objets u et v → taille $\approx n^2$

- La variation de coût associée à l'échange des objets en u et en v est :

$$\Delta C_P = \sum_{j=1 \text{ à } n, j\neq u} \left(f_{uj} d_{p(v)p(j)} - f_{uj} d_{p(u)p(j)} \right)$$ → arcs partant de u : départ de p(v) au lieu de p(u)

$$+ \sum_{j=1 \text{ à } n, j\neq v} \left(f_{vj} d_{p(u)p(j)} - f_{vj} d_{p(v)p(j)} \right)$$ → arcs partant de v : départ de p(u) au lieu de p(v)

$$+ \sum_{i=1 \text{ à } n, i\neq u} \left(f_{iu} d_{p(i)p(v)} - f_{iu} d_{p(i)p(u)} \right)$$ → arcs arrivant en u : arrivée en p(v) au lieu de p(u)

$$+ \sum_{i=1 \text{ à } n, i\neq v} \left(f_{iv} d_{p(i)p(u)} - f_{iv} d_{p(i)p(v)} \right)$$ → arcs arrivant en v : arrivée en p(u) au lieu de p(v)

- La liste tabou contient les solutions précédentes P_k avec leur durée d'interdiction t_k.
  La durée t_k est le nombre d'itérations pendant lesquelles la solution P_k est taboue.
  La durée peut être fixée ou tirée aléatoirement pour chaque solution.

4.3.3 Recherche tabou

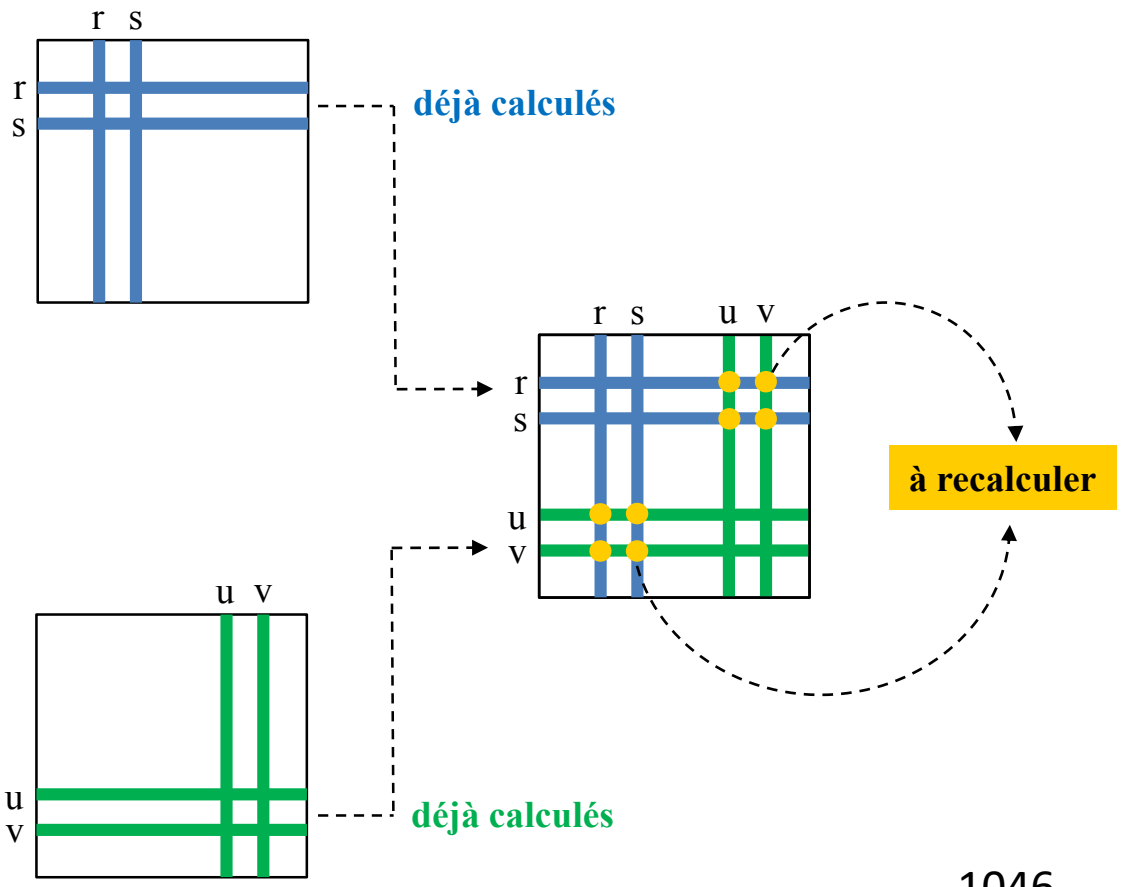
Affectation quadratique

L'évaluation d'une solution est la partie la plus coûteuse en temps de calcul.
On peut réduire le temps de calcul en évaluant les variations entre 2 solutions voisines.

- solution p $\xrightarrow{(r,s)}$ solution q
  → $C(q) = F \times D(q)$

- solution p $\xrightarrow{(r,s)}$ $\xrightarrow{(u,v)}$ solution q'
  → $C(q') = F \times D(q')$

- solution p $\xrightarrow{(u,v)}$ solution p'
  → $C(p') = F \times D(p')$



4.3.3 Recherche tabou

Exemple

Problème d'affectation quadratique NUG5

Matrice des flots

$$F = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 5 & 2 & 4 & 1 \\ 5 & 0 & 3 & 0 & 2 \\ 2 & 3 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 5 \\ 1 & 2 & 0 & 5 & 0 \end{pmatrix} \end{array}$$

Matrice des distances

$$D = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 2 & 3 \\ 1 & 0 & 2 & 1 & 2 \\ 1 & 2 & 0 & 1 & 2 \\ 2 & 1 & 1 & 0 & 1 \\ 3 & 2 & 2 & 1 & 0 \end{pmatrix} \end{array}$$

On cherche la permutation des objets (1 , 2 , 3 , 4 , 5) minimisant le coût $C = F \times D(p)$.

- Le nombre total de permutations est : $5! = 120$
- Il existe 2 permutations optimales : $(2 , 4 , 5 , 1 , 3) \rightarrow C = 50$
  $(3 , 4 , 5 , 1 , 2) \rightarrow C = 50$

4.3.3 Recherche tabou

Exemple : itération 0

On initialise la méthode tabou avec la solution : $p_0 = (5\ ,\ 4\ ,\ 3\ ,\ 2\ ,\ 1)$ de coût $C_0 = 64$.

- Le voisinage est défini comme l'ensemble des solutions accessibles en échangeant 2 objets. Tous les mouvements sont autorisés.

Solution		Mouvement									
		1-2	1-3	1-4	1-5	2-3	2-4	2-5	3-4	3-5	4-5
(5,4,3,2,1)		(4,5,3,2,1)	(3,4,5,2,1)	(2,4,3,5,1)	(1,4,3,2,5)	(5,3,4,2,1)	(5,2,3,4,1)	(5,1,3,2,4)	(5,4,2,3,1)	(5,4,1,2,3)	(5,4,3,1,2)
64	Coût	60	60	80	68	66	78	80	64	78	66
	Variation	-4	-4	16	4	2	14	16	0	14	2

- La meilleure solution dans le voisinage est : $p_1 = (\mathbf{4}\ ,\ \mathbf{5}\ ,\ 3\ ,\ 2\ ,\ 1)$ de coût $C_1 = 60$.
  → amélioration $\mathbf{\Delta C = -4}$

- La solution p_0 entre dans la liste tabou pendant un nombre d'itérations $t_0 = 3$.

Liste tabou : T_1

Solutions taboues	Durée d'interdiction
$p_0 = (5\ ,\ 4\ ,\ 3\ ,\ 2\ ,\ 1)$	$t_0 = 3$

4.3.3 Recherche tabou

Exemple : itération 1

On cherche une amélioration locale de la solution : $p_1 = (4 , 5 , 3 , 2 , 1)$ de coût $C_1 = 60$.

- Le voisinage est défini comme l'ensemble des solutions accessibles en échangeant 2 objets. Le mouvement (1–2) n'est pas autorisé, car il redonnerait p_0.

Solution		Mouvement									
		1-2	1-3	1-4	1-5	2-3	2-4	2-5	3-4	3-5	4-5
(4,5,3,2,1)		(5,4,3,2,1)	(3,5,4,2,1)	(2,5,3,4,1)	(1,5,3,2,4)	(4,3,5,2,1)	(4,2,3,5,1)	(4,1,3,2,5)	(4,5,2,3,1)	(4,5,1,2,3)	(4,5,3,1,2)
60	Coût	64	70	82	72	52	72	72	60	74	62
	Variation	4	10	22	12	-8	12	12	0	14	2

- La meilleure solution dans le voisinage est : $p_2 = (4 , \mathbf{3} , \mathbf{5} , 2 , 1)$ de coût $C_2 = 52$.
  → amélioration $\mathbf{\Delta C = -8}$

- La solution p_1 entre dans la liste tabou pendant un nombre d'itération $t_1 = 3$.

Liste tabou : T_2

Solutions taboues	Durée d'interdiction
$p_0 = (5 , 4 , 3 , 2 , 1)$	$t_0 = 2$
$p_1 = (4 , 5 , 3 , 2 , 1)$	$t_1 = 3$

4.3.3 Recherche tabou

Exemple : itération 2

On cherche une amélioration locale de la solution : $p_2 = (4 , 3 , 5 , 2 , 1)$ de coût $C_2 = 52$.

- Le voisinage est défini comme l'ensemble des solutions accessibles en échangeant 2 objets. Le mouvement (2–3) n'est pas autorisé, car il redonnerait p_1.

Solution		Mouvement									
		1-2	1-3	1-4	1-5	2-3	2-4	2-5	3-4	3-5	4-5
(4,3,5,2,1)		(3,4,5,2,1)	(5,3,4,2,1)	(2,3,5,4,1)	(1,3,5,2,4)	(4,5,3,2,1)	(4,2,5,3,1)	(4,1,5,2,3)	(4,3,2,5,1)	(4,3,1,2,5)	(4,3,5,1,2)
52	Coût	60	66	74	60	60	52	76	72	62	62
	Variation	8	14	22	8	8	0	24	20	10	10

- La meilleure solution dans le voisinage est : $p_3 = (4 , \mathbf{2} , 5 , \mathbf{3} , 1)$ de coût $C_3 = 52$.
  → coût identique $\mathbf{\Delta C = 0}$

- La solution p_2 entre dans la liste tabou pendant un nombre d'itération $t_2 = 3$.

Liste tabou : T_3

Solutions taboues	Durée d'interdiction
$p_0 = (5 , 4 , 3 , 2 , 1)$	$t_0 = 1$
$p_1 = (4 , 5 , 3 , 2 , 1)$	$t_1 = 2$
$p_2 = (4 , 3 , 5 , 2 , 1)$	$t_2 = 3$

4.3.3 Recherche tabou

Exemple : itération 3

On cherche une amélioration locale de la solution : $p_3 = (4\ ,\ 2\ ,\ 5\ ,\ 3\ ,\ 1)$ de coût $C_3 = 52$.

- Le voisinage est défini comme l'ensemble des solutions accessibles en échangeant 2 objets.
  Le mouvement (2–4) n'est pas autorisé, car il redonnerait p_2.

Solution		Mouvement									
		1-2	1-3	1-4	1-5	2-3	2-4	2-5	3-4	3-5	4-5
(4,2,5,3,1)		(2,4,5,3,1)	(5,2,4,3,1)	(3,2,5,4,1)	(1,2,5,3,4)	(4,5,2,3,1)	(4,3,5,2,1)	(4,1,5,3,2)	(4,2,3,5,1)	(4,2,1,3,5)	(4,2,5,1,3)
52	Coût	60	66	74	60	60	52	76	72	62	62
	Variation	8	14	22	8	8	0	24	20	10	10

- La meilleure solution dans le voisinage est : $p_4 = (\mathbf{2}\ ,\ \mathbf{4}\ ,\ 5\ ,\ 3\ ,\ 1)$ de coût $C_4 = 60$.
  → dégradation $\mathbf{\Delta C = +8}$

- La solution p_3 entre dans la liste tabou pendant un nombre d'itération $t_3 = 3$.

Liste tabou : T_4

Solutions taboues	Durée d'interdiction
$p_1 = (4\ ,\ 5\ ,\ 3\ ,\ 2\ ,\ 1)$	$t_1 = 1$
$p_2 = (4\ ,\ 3\ ,\ 5\ ,\ 2\ ,\ 1)$	$t_2 = 2$
$p_3 = (4\ ,\ 2\ ,\ 5\ ,\ 3\ ,\ 1)$	$t_3 = 3$

→ p_0 sort de la liste.
p_0 est à nouveau autorisée.

4.3.3 Recherche tabou

Exemple : itération 4

On cherche une amélioration locale de la solution : $p_4 = (2\ ,\ 4\ ,\ 5\ ,\ 3\ ,\ 1)$ de coût $C_4 = 60$.

- Le voisinage est défini comme l'ensemble des solutions accessibles en échangeant 2 objets.
  Le mouvement (1–2) n'est pas autorisé, car il redonnerait p_3.

Solution		Mouvement									
		1-2	1-3	1-4	1-5	2-3	2-4	2-5	3-4	3-5	4-5
(2,4,5,3,1)		(4,2,5,3,1)	(5,4,2,3,1)	(3,4,5,2,1)	(1,4,5,3,2)	(2,5,4,3,1)	(2,3,5,4,1)	(2,1,5,3,4)	(2,4,3,5,1)	(2,4,1,3,5)	(2,4,5,1,3)
60	Coût	52	64	60	72	70	74	72	80	70	50
	Variation	-8	4	0	12	10	14	12	20	10	-10

- La meilleure solution dans le voisinage est : $p_5 = (2\ ,\ 4\ ,\ 5\ ,\ \mathbf{1}\ ,\ \mathbf{3})$ de coût $C_5 = 50$.
  → amélioration $\mathbf{\Delta C = -10}$

- La solution p_4 entre dans la liste tabou pendant un nombre d'itération $t_4 = 3$.

Liste tabou : T_5

Solutions taboues	Durée d'interdiction
$p_2 = (4\ ,\ 3\ ,\ 5\ ,\ 2\ ,\ 1)$	$t_2 = 1$
$p_3 = (4\ ,\ 2\ ,\ 5\ ,\ 3\ ,\ 1)$	$t_3 = 2$
$p_4 = (2\ ,\ 4\ ,\ 5\ ,\ 3\ ,\ 1)$	$t_4 = 3$

→ p_1 sort de la liste.
p_1 est à nouveau autorisée.

4.3.3 Recherche tabou

Exemple : itération 5

On cherche une amélioration locale de la solution : $p_5 = (2 , 4 , 5 , 1 , 3)$ de coût $C_5 = 50$.

- Le voisinage est défini comme l'ensemble des solutions accessibles en échangeant 2 objets.
  Le mouvement (4–5) n'est pas autorisé, car il redonnerait p_4.

Solution		Mouvement									
		1-2	1-3	1-4	1-5	2-3	2-4	2-5	3-4	3-5	4-5
(2,4,5,1,3)		(4,2,5,1,3)	(5,4,2,1,3)	(1,4,5,2,3)	(3,4,5,1,2)	(2,5,4,1,3)	(2,1,5,4,3)	(2,3,5,1,4)	(2,4,1,5,3)	(2,4,3,1,5)	(2,4,5,3,1)
50	Coût	62	66	72	50	60	62	74	70	70	60
	Variation	12	16	22	0	10	12	24	20	20	10

- La meilleure solution dans le voisinage est : $p_6 = (3 , 4 , 5 , 1 , 2)$ de coût $C_5 = 50$.
  → coût identique $\Delta C = 0$
  On obtient les **2 permutations optimales** : $p_5 = (2 , 4 , 5 , 1 , 3)$ de coût $C = 50$.
  $p_6 = (3 , 4 , 5 , 1 , 2)$
  Les itérations suivantes ne donnent plus d'améliorations.

- La liste tabou a produit une dégradation à l'itération 3 : $\Delta C = +8$.
  Ceci a permis de **sortir d'un minimum local** : $p_3 = (4 , 2 , 5 , 3 , 1)$ de coût $C_3 = 52$
  puis d'obtenir à l'itération suivante la solution optimale de coût $C = 50$.

4.3.3 Recherche tabou

Exemple

- Problème NUG5 : récapitulatif des itérations

Itération	Solution		Mouvement 1-2	1-3	1-4	1-5	2-3	2-4	2-5	3-4	3-5	4-5
	(5,4,3,2,1)		(4,5,3,2,1)	(3,4,5,2,1)	(2,4,3,5,1)	(1,4,3,2,5)	(5,3,4,2,1)	(5,2,3,4,1)	(5,1,3,2,4)	(5,4,2,3,1)	(5,4,1,2,3)	(5,4,3,1,2)
0	64	Coût	60	60	80	68	66	78	80	64	78	66
		Variation	-4	-4	16	4	2	14	16	0	14	2
	(4,5,3,2,1)		(5,4,3,2,1)	(3,5,4,2,1)	(2,5,3,4,1)	(1,5,3,2,4)	(4,3,5,2,1)	(4,2,3,5,1)	(4,1,3,2,5)	(4,5,2,3,1)	(4,5,1,2,3)	(4,5,3,1,2)
1	60	Coût	64	70	82	72	52	72	72	60	74	62
		Variation	4	10	22	12	-8	12	12	0	14	2
	(4,3,5,2,1)		(3,4,5,2,1)	(5,3,4,2,1)	(2,3,5,4,1)	(1,3,5,2,4)	(4,5,3,2,1)	(4,2,5,3,1)	(4,1,5,2,3)	(4,3,2,5,1)	(4,3,1,2,5)	(4,3,5,1,2)
2	52	Coût	60	66	74	60	60	52	76	72	62	62
		Variation	8	14	22	8	8	0	24	20	10	10
	(4,2,5,3,1)		(2,4,5,3,1)	(5,2,4,3,1)	(3,2,5,4,1)	(1,2,5,3,4)	(4,5,2,3,1)	(4,3,5,2,1)	(4,1,5,3,2)	(4,2,3,5,1)	(4,2,1,3,5)	(4,2,5,1,3)
3	52	Coût	60	66	74	60	60	52	76	72	62	62
		Variation	8	14	22	8	8	0	24	20	10	10
	(2,4,5,3,1)		(4,2,5,3,1)	(5,4,2,3,1)	(3,4,5,2,1)	(1,4,5,3,2)	(2,5,4,3,1)	(2,3,5,4,1)	(2,1,5,3,4)	(2,4,3,5,1)	(2,4,1,3,5)	(2,4,5,1,3)
4	60	Coût	52	64	60	72	70	74	72	80	70	50
		Variation	-8	4	0	12	10	14	12	20	10	-10
	(2,4,5,1,3)		(4,2,5,1,3)	(5,4,2,1,3)	(1,4,5,2,3)	(3,4,5,1,2)	(2,5,4,1,3)	(2,1,5,4,3)	(2,3,5,1,4)	(2,4,1,5,3)	(2,4,3,1,5)	(2,4,5,3,1)
5	50	Coût	62	66	72	50	60	62	74	70	70	60
		Variation	12	16	22	0	10	12	24	20	20	10

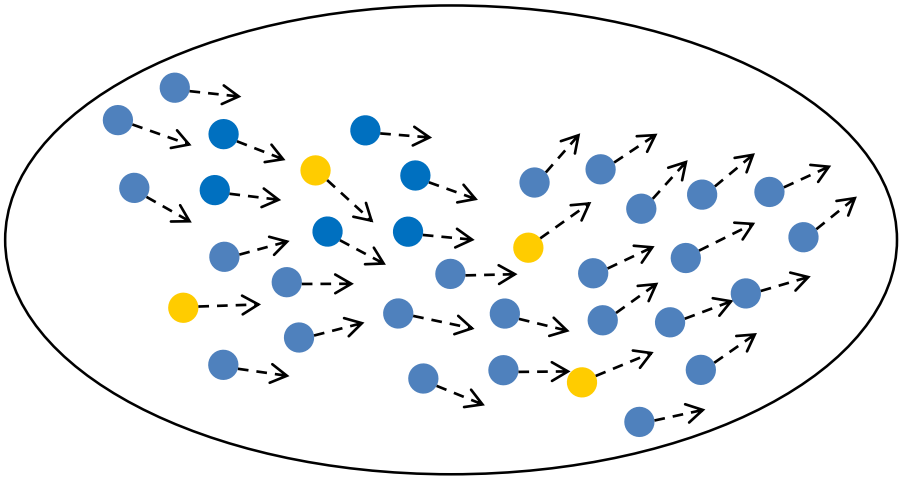
4.3.4 Essaims

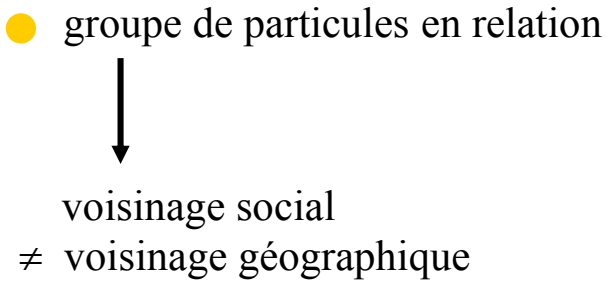
Principe

La méthode des essaims particulaires (« Particle Swarm », Eberhart et Kennedy 1995) s'inspire du comportement social des animaux (nuée d'oiseaux, banc de poissons, essaim d'abeilles).

- Un essaim est composé d'individus ou **particules** en mouvement.

- Le mouvement de chaque particule est influencé par :
  - sa vitesse actuelle → tendance « aventureuse »
  - son expérience personnelle → tendance « conservatrice » à revenir à sa meilleure position
  - l'expérience sociale → tendance « panurgienne » à suivre un groupe de voisins

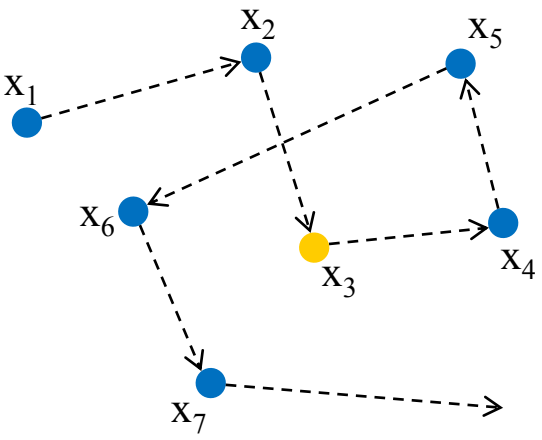




4.3.4 Essaims

Mouvement individuel

- Chaque particule est repérée par sa position x et sa vitesse v.
  La particule se déplace au cours du temps.
  x_k = position de la particule à la date t_k
  v_k = vitesse de la particule à la date t_k

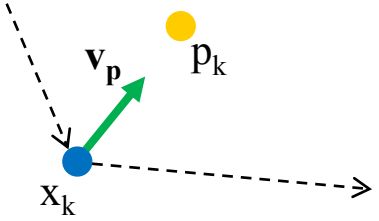


- La meilleure position rencontrée est gardée en mémoire.
  p_k = **meilleure position** connue de la particule à la date t_k
  (« p » = **personnelle**)

Date	t_1	t_2	t_3	t_4	…	t_k
Position	x_1	x_2	x_3	x_4	…	x_k

$p_k = x_3$ = meilleure position connue à t_k

- La particule est attirée vers sa meilleure position.
  La vitesse est déviée par une composante v_p orientée vers p_k.
  $v_k \rightarrow v_k + v_p$ avec $v_p = c_p.(p_k - x_k)$.

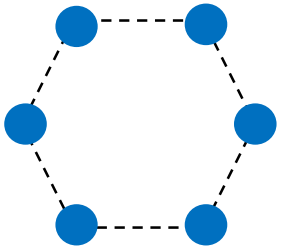


4.3.4 Essaims

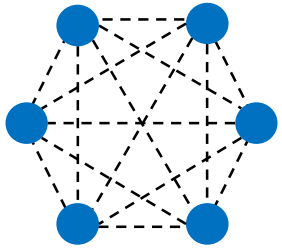
Voisinage

- Chaque particule communique avec un groupe d'autres particules ou **voisins**.
  V = ensemble des voisins de la particule
  g_k = **meilleure position** connue des particules de V à la date t_k (« g » = **groupe**)

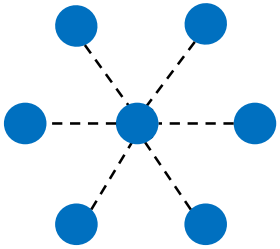
- Plusieurs topologies de voisinage sont possibles.



Anneau
V = 2 à m particules

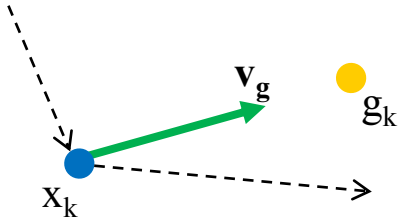


Étoile
V = toutes les particules



Rayon
V = une particule centrale

- La particule est attirée vers la meilleure position de son groupe.
  La vitesse est déviée par une composante v_g orientée vers g_k.
  $v_k \rightarrow v_k + v_g$ avec $v_g = c_g.(g_k - x_k)$.

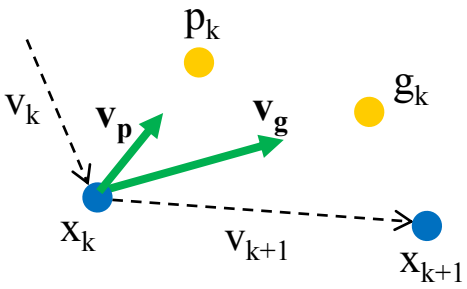


4.3.4 Essaims

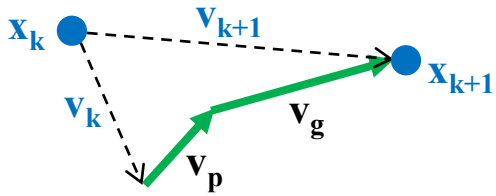
Mouvement des particules

- L'essaim est composé de N particules dont on simule le mouvement.
  La vitesse et la position de chaque particule sont mises à jour à chaque date.

Date t_k		Date t_{k+1}	
Vitesse :	v_k	Vitesse :	$v_{k+1} = v_k + v_p + v_g$
Position :	x_k	Position :	$x_{k+1} = x_k + v_{k+1}$



- La nouvelle vitesse est une pondération des 3 composantes.
  - composante aventureuse : v_k
  - composante conservatrice : $v_p = c_p.(p_k - x_k)$ avec $c_p = c_1 r_p$
  - composante panurgienne : $v_g = c_g.(g_k - x_k)$ avec $c_g = c_2 r_g$



- Les coefficients de pondération c_p et c_g comportent
  - une part fixe (c_1, c_2) → à régler expérimentalement selon le problème
  - une part aléatoire (r_p, r_g) → permet une exploration large de l'espace de recherche

4.3.4 Essaims

Algorithme

- On simule le mouvement d'un essaim de N particules cherchant le minimum d'une fonction.

$$\min_x f(x) \quad \rightarrow \text{ N particules} \quad \begin{cases} \text{positions } x^{(1)}, x^{(2)}, \dots, x^{(N)} \\ \text{vitesses } v^{(1)}, v^{(2)}, \dots, v^{(N)} \end{cases}$$

- La convergence vers le minimum global dépend :
  - du nombre de particules N → selon problème (N=10 à 100)
  - de la topologie du voisinage V → m particules (m=2 à 10), voisinage fixé ou aléatoire
  - des réglages de la vitesse → nombreuses variantes, réglage expérimental

- Mise à jour de la vitesse :

$$v_{k+1} = \omega v_k + r_p c_1.(p_k - x_k) + r_g c_2.(g_k - x_k) + c_3 v_a$$

$$\begin{cases} \omega & = \text{facteur d'inertie} & \in [0\ ,\ 1] \\ c_1, c_2, c_3 & = \text{constantes} & \in [0\ ,\ 1] \\ r_p, r_g & = \text{nombres aléatoires} & \in [0\ ,\ 1] \\ v_a & = \text{perturbation aléatoire} & \end{cases}$$

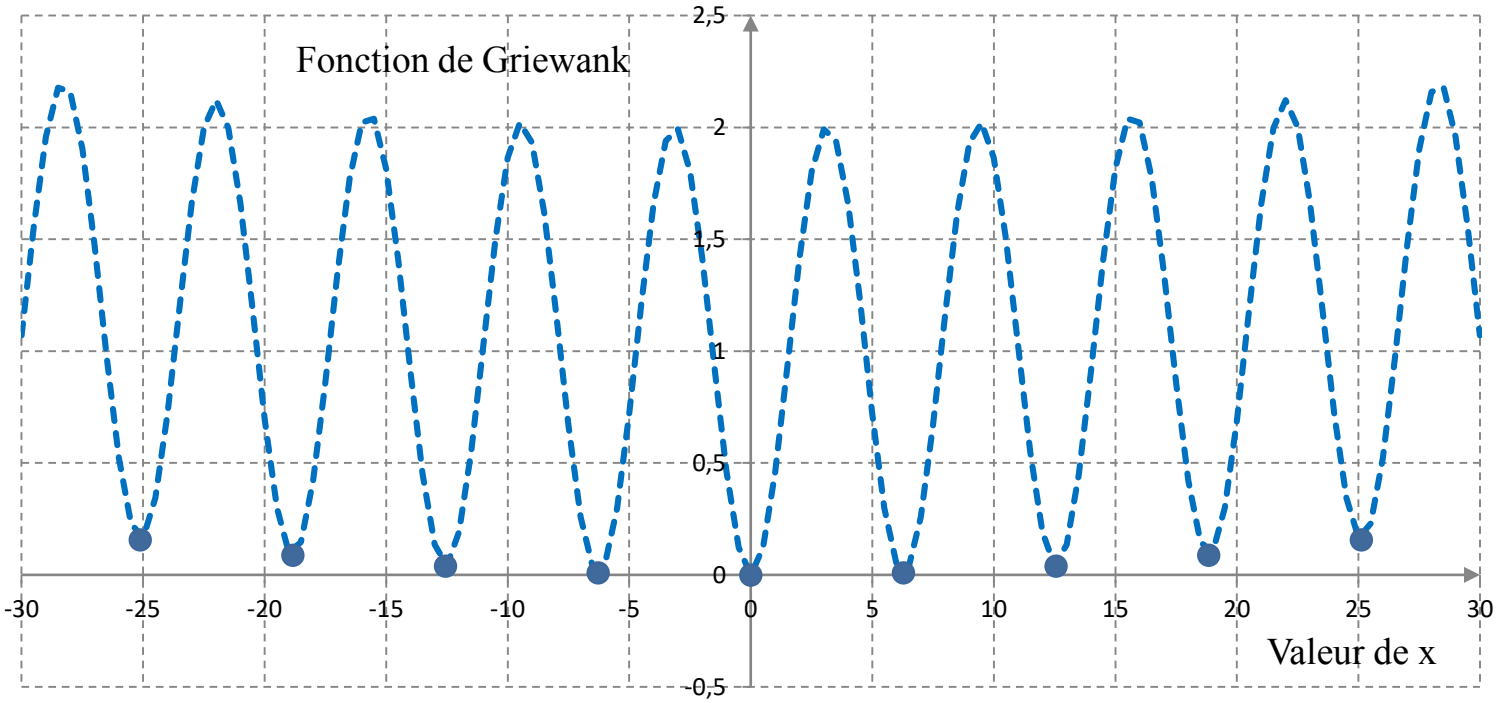
- On peut favoriser la convergence vers un optimum global
  - en imposant une borne v_{max} sur la vitesse
  - en réduisant progressivement le facteur d'inertie ω
  - en réinitialisant aléatoirement les particules de vitesse faible

4.3.4 Essaims

Exemple

Fonction de Griewank à une variable $f(x) = \frac{x^2}{4000} - \cos(x) + 1$

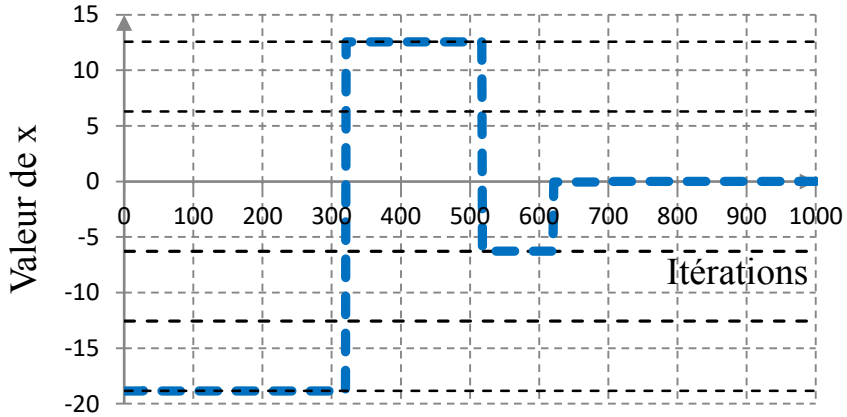
Minima locaux : $\sin(x) + \frac{x}{2000} = 0 \quad \rightarrow$ proche de $k.2\pi$ lorsque $|x| << 2000$

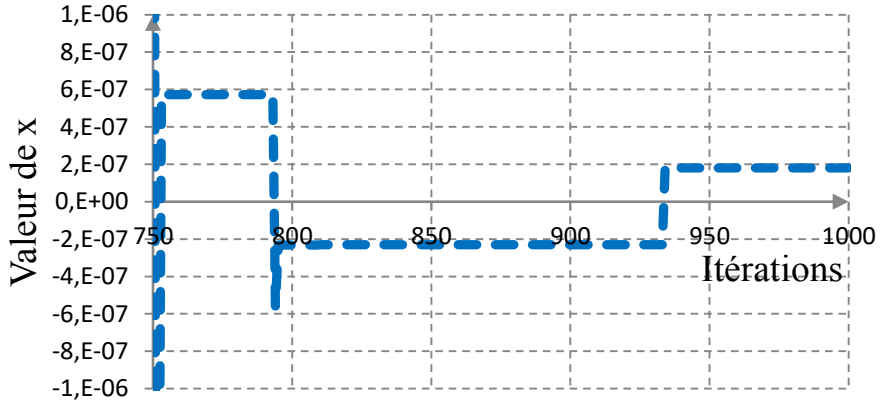


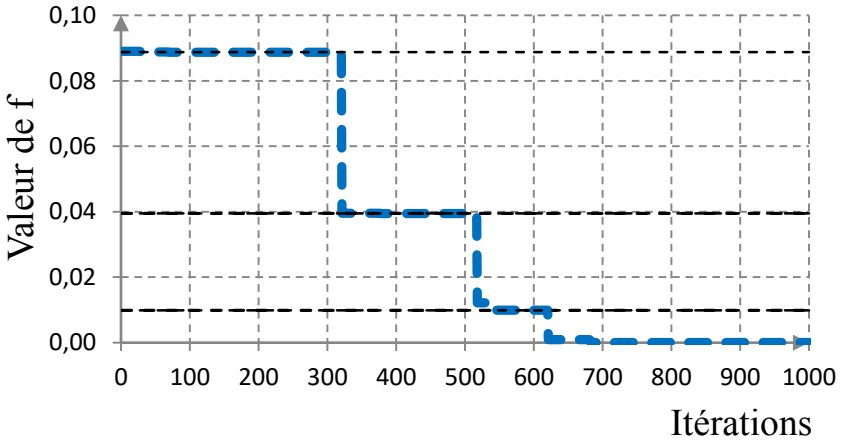
4.3.4 Essaims

Exemple

Fonction de Griewank à une variable $f(x) = \frac{x^2}{4000} - \cos(x) + 1$







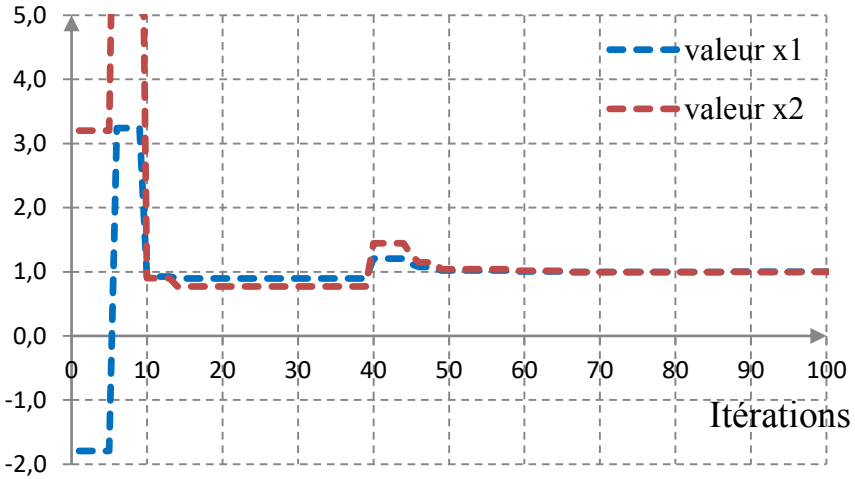
- Nombre de particules : 100
- Nombre d'itérations : 1000
- Nombre d'évaluations : 3.10^6

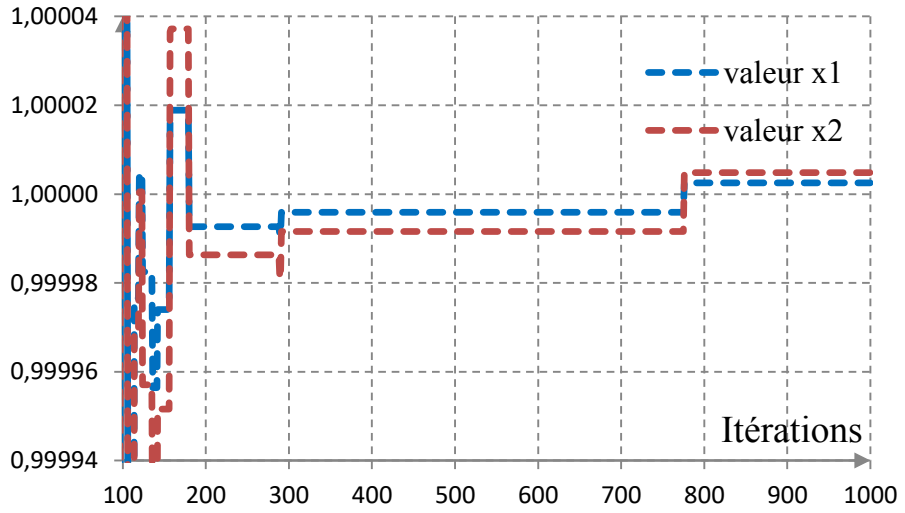
→ Solution : $x = 1.804\ 10^{-7}$

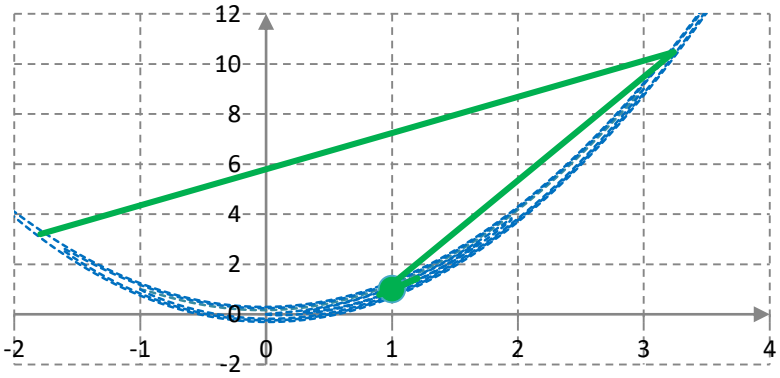
4.3.4 Essaims

Exemple

Fonction de Rosenbrock à deux variables : $f(x_1, x_2) = 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2$







- Nombre de particules : 100
- Nombre d'itérations : 1000
- Nombre d'évaluations : 3.10^6

→ Solution : $x_1 = 1.0000025$
$x_2 = 1.0000048$

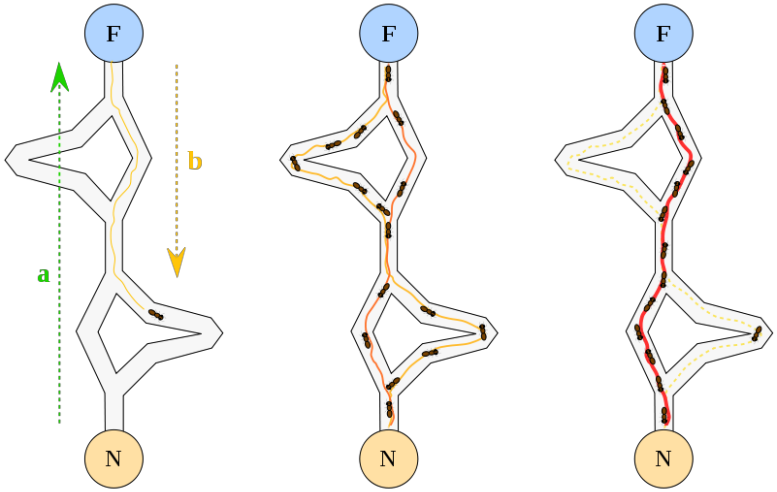
4.3.5 Fourmis

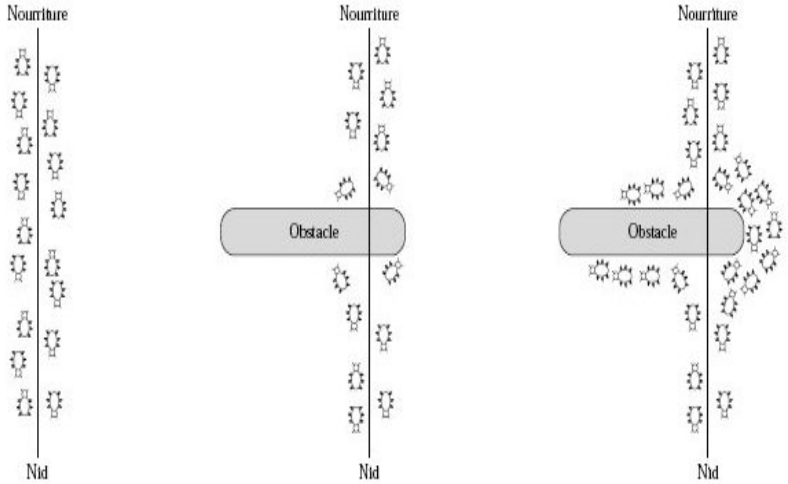
Principe

La méthode des colonies de fourmis (« Ant Colony », Dorigo et co 1990) est inspirée du comportement des fourmis à la recherche de nourriture.

- Les fourmis déposent sur leur passage des substances volatiles appelées phéromones.
  Plus un chemin est concentré en phéromones, plus il est attractif.
  Ce mode de communication indirecte par modification de l'environnement est la **stigmergie**.

- Le chemin le plus court entre la fourmilière et la nourriture est parcouru plus rapidement.
  Il reçoit davantage de phéromones au cours du temps et devient le chemin le plus emprunté.





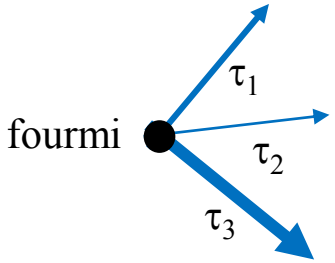
4.3.5 Fourmis

Mouvement individuel

- Chaque fourmi se déplace de manière aléatoire indépendamment des autres fourmis.
  Le choix de la direction dépend de la concentration locale en phéromones.

- Le trajet d'une fourmi est composé d'une succession de segments.
  A partir de sa position courante, une fourmi a le choix entre plusieurs segments.

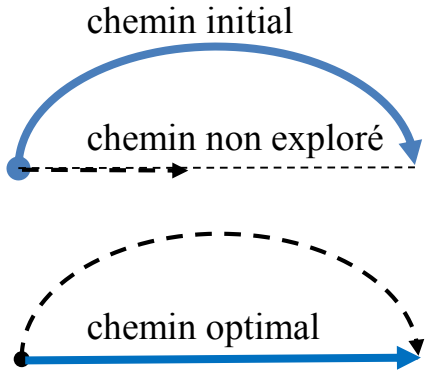
- L'**intensité** τ_i du segment i représente sa concentration en phéromones.
  La probabilité p_i de choisir le segment i dépend de son intensité τ_i.
  $\tau_2 < \tau_1 < \tau_3 \quad \Rightarrow \quad p_2 < p_1 < p_3$
  → Le segment 3 a une probabilité plus forte d'être sélectionné.



Evaporation

- Si un seul chemin est marqué, toutes les fourmis le suivent.
  Les autres solutions ne seront pas suffisamment explorées.
  → Le chemin sélectionné n'est pas forcément le plus court.

- L'**évaporation** des phéromones permet d'affaiblir le chemin initial.
  La probabilité d'explorer d'autres solutions est plus forte.

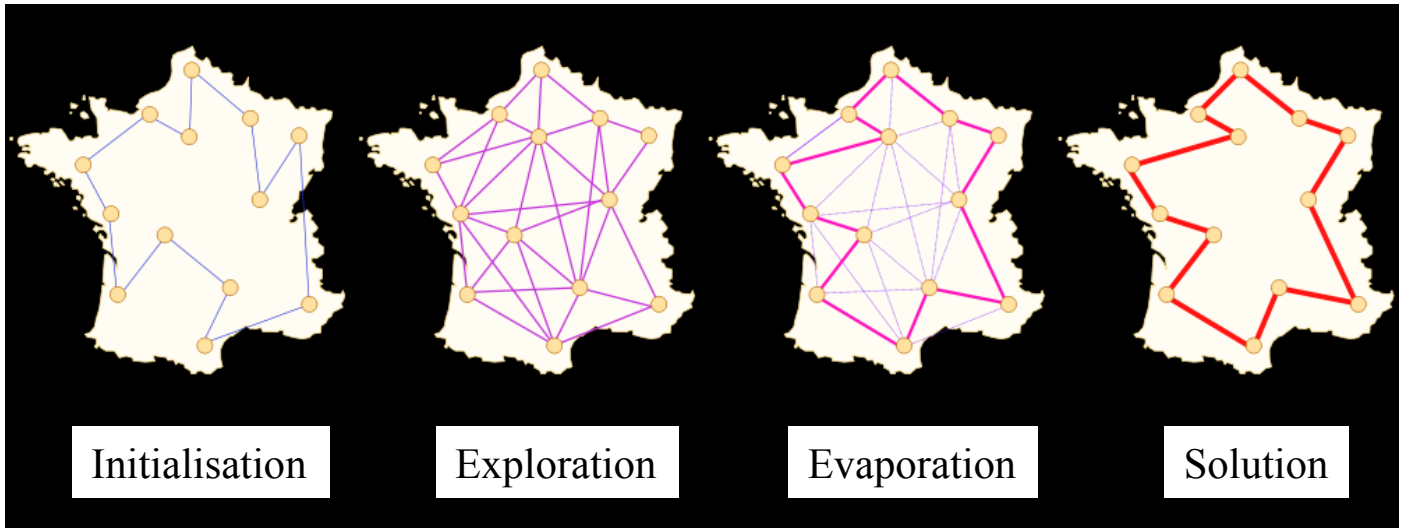


4.3.5 Fourmis

Voyageur de commerce

Les algorithmes de colonies de fourmis s'appliquent de façon naturelle au PVC.

- Une fourmi « éclaireuse » crée un chemin initial peu marqué en phéromones.
- Les fourmis suivantes explorent aléatoirement l'ensemble des chemins.
- Le chemin le plus court se renforce plus rapidement en phéromones et devient plus attractif. L'évaporation fait oublier les autres chemins.



4.3.5 Fourmis

Voyageur de commerce

- Lorsqu'une fourmi se trouve à la ville i, il reste un ensemble de villes non visitées J.
- La probabilité p_{ij} de choisir la ville $j \in J$ dépend des distances d_{ij} et des concentrations.

$$p_{ij} = (\tau_{ij})^{\alpha}.(\eta_{ij})^{\beta} \quad \rightarrow \text{ normalisée à 1 sur } j \in J$$

η_{ij} = **visibilité** entre les villes i et j $\rightarrow \eta_{ij} = \frac{1}{d_{ij}}$ privilégie les villes proches

τ_{ij} = **intensité** de l'arête i-j $\rightarrow$ concentration en phéromones

- Les paramètres de réglage α et β et contrôlent l'influence de la visibilité et de l'intensité.
- A la fin de son tour, la fourmi a parcouru une distance : $D = \sum d_{ij}$.

Elle dépose sur les arêtes i-j sélectionnées une quantité de phéromones $\Delta\tau_{ij}$.

$\Delta\tau_{ij} = \frac{D_0}{D}$ si i-j est sur le trajet $\rightarrow$ privilégie les chemins les plus courts

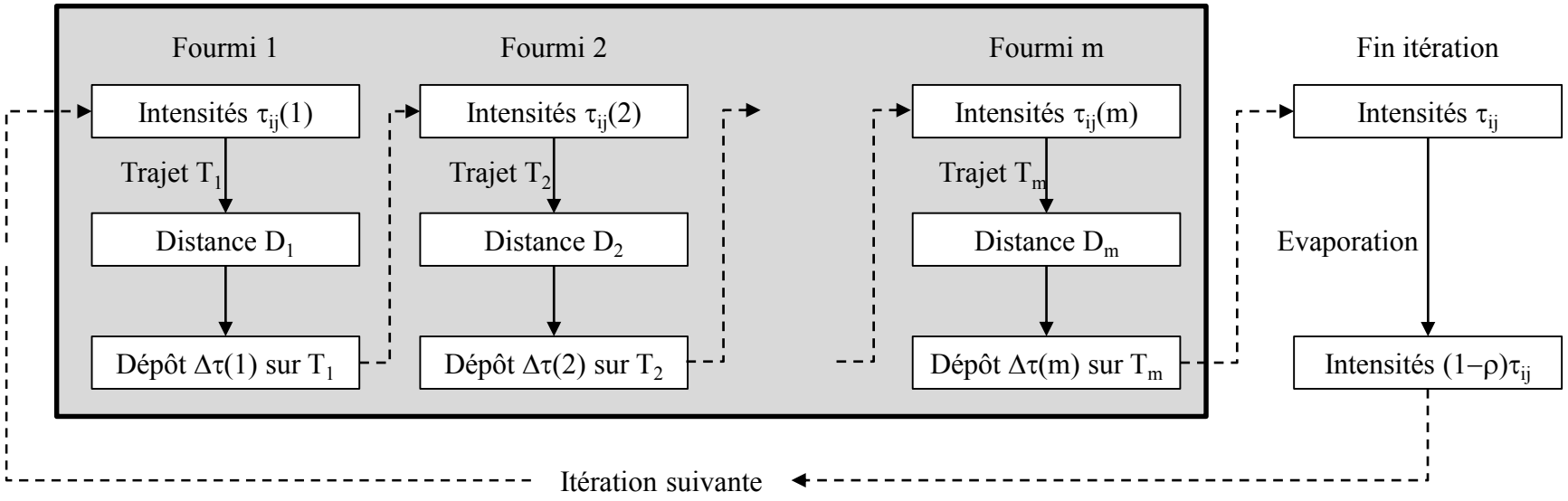
D_0 = distance de référence fixée

4.3.5 Fourmis

Voyageur de commerce

- A chaque itération, un ensemble de m fourmis parcourt les n villes.
  Chaque fourmi augmente les intensités τ_{ij} sur son trajet.
  $\tau_{ij} \rightarrow \tau_{ij} + \Delta\tau_{ij}(k)$ après passage de la fourmi k (=1 à m) sur les arêtes i-j sélectionnées

- Lorsque les m fourmis sont passées, on réduit toutes les intensités par évaporation.
  L'évaporation permet d'affaiblir l'attractivité des « mauvaises » solutions.
  $\tau_{ij} \rightarrow (1 - \rho)\,\tau_{ij}$ sur toutes les arêtes i-j (ρ = paramètre d'évaporation, $0 < \rho < 1$)



4.3.6 Algorithme évolutionnaire

Principe

Les algorithmes évolutionnaires ou génétiques (Holland 1975) sont inspirés des principes de l'évolution et de la sélection naturelle (Darwin 1859)

- Une **population d'individus** évolue de génération en génération.
  Chaque individu est plus ou moins « performant » selon ses caractéristiques.
  L'évolution naturelle favorise les caractéristiques qui rendent un individu performant.

- Le renouvellement d'une génération (parents) se déroule en 4 étapes :
  - sélection d'une partie des parents pour la reproduction
  - croisement des parents sélectionnés pour engendrer des enfants
  - mutation (variation) éventuelle d'une partie des enfants
  - sélection entre parents et enfants pour constituer la génération suivante.

- L'évolution fait apparaître des individus de plus en plus performants.
  La rapidité et l'efficacité du processus dépend :
  - de la taille de la population
  - de la représentation (codage) des individus → variables binaires, entières, réelles
  - des opérateurs de sélection et de variation → nombreuses variantes possibles

4.3.6 Algorithme évolutionnaire

Problème de minimisation

On applique un algorithme évolutionnaire à un problème de minimisation.

$$\min_x f(x) \quad \rightarrow \quad \begin{cases} \text{variables } x = \text{« phénotype » d'un individu} \\ \text{fonction } f = \text{« performance (fitness) » de l'individu} \end{cases}$$

- La population est constitué de p individus (p ≈ 100 à 1000 selon la nature du problème).
  Les mécanismes d'évolution ont pour objectif :
  - de conserver les meilleures solutions (sélection) → **intensification**
  - d'explorer de nouvelles solutions (variation) → **diversification**

- Les **opérateurs de sélection** sont basés sur la fonction de performance.
  → sélection déterministe, proportionnelle, par tournois, …
  → sélection pour la reproduction, pour le remplacement

- Les **opérateurs de variation** génèrent des solutions aléatoires.
  → modification d'une partie des variables
  → variation par croisement, par mutation

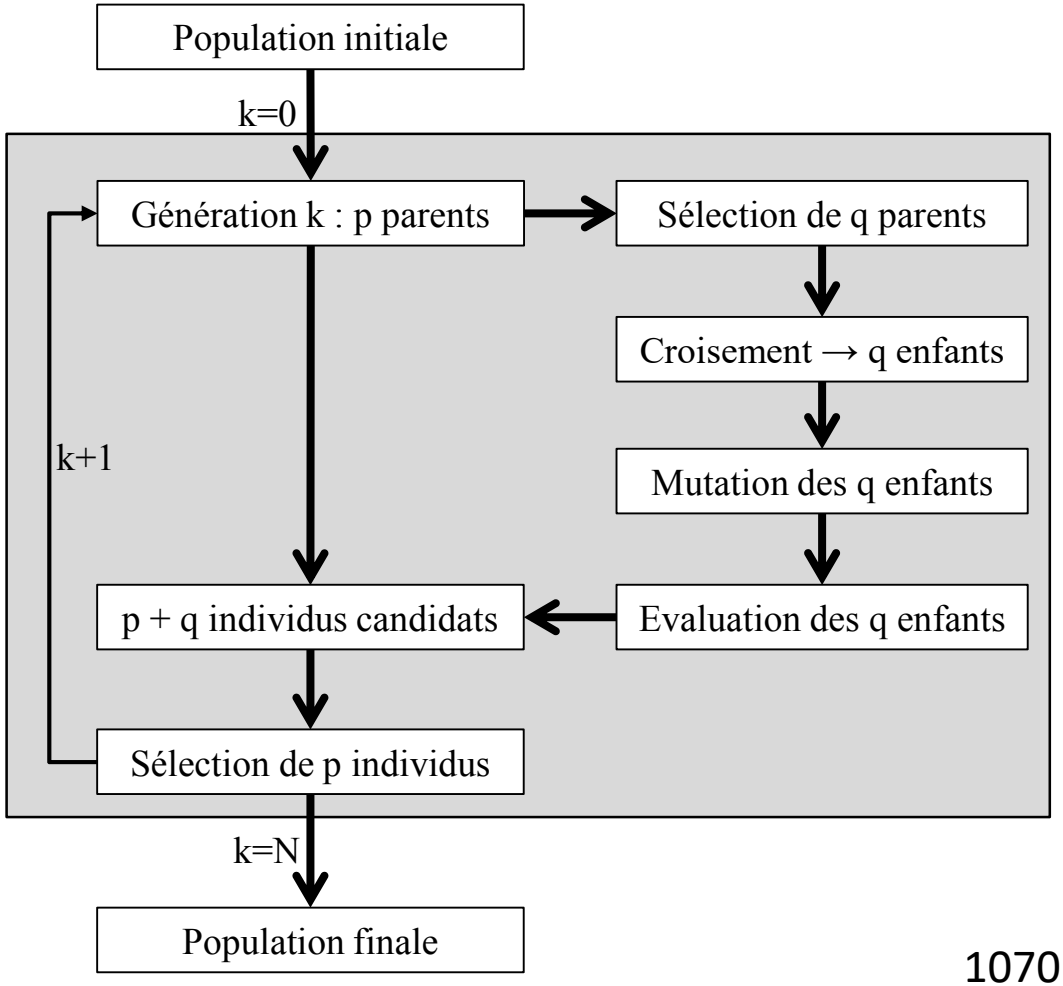
- Un algorithme « génétique » utilise une représentation binaire des individus (« génotype »).
  Le génotype binaire est transcrit en **phénotype** (variables x) avant évaluation.

4.3.6 Algorithme évolutionnaire

Algorithme

Eléments de l'algorithme

- Paramètres
  - p = taille population
  - q = nombre d'enfants
  - N = nombre de générations
- **Opérateurs**
  - Sélection
  - Croisement
  - Mutation
- **Codage** des solutions
  - Binaire (génotype)
  - Réel (phénotype)
- Fonction d'évaluation
  - Fitness



4.3.6 Algorithme évolutionnaire

Exemple

Minimisation de $f(x) = x^2$

- On considère une population de 10 individus, représentés chacun par :
  - une valeur $x_{i,\ i=1 \text{ à } 10}$ comprise entre −16 et +16
  - son évaluation $f(x_i)$

- L'étape de reproduction comprend 4 étapes :
  - sélection des 8 meilleurs parents parmi les 10 → 8 parents
  - appariement des parents sélectionnés par paires → 4 paires de parents
  - croisement de chaque paire de parents pour engendrer 2 enfants → 8 enfants
  - mutation de 2 enfants parmi les 8

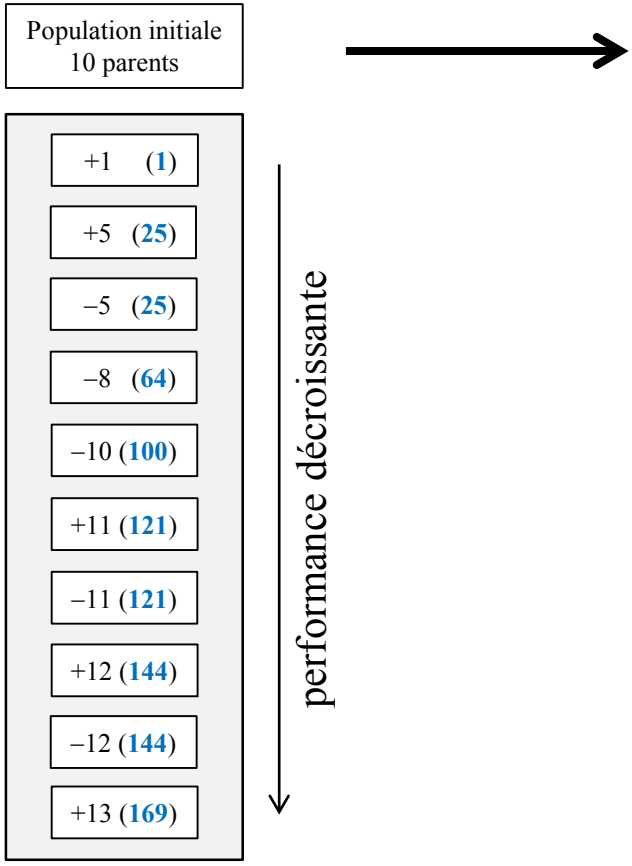
- Les 8 enfants $y_{j,\ j=1 \text{ à } 8}$ sont ensuite évalués → $f(y_j)$
  On dispose de 18 candidats pour la génération suivante :
  - 10 parents → x_i , $f(x_i)$ pour i = 1 à 10
  - 8 enfants → y_j , $f(x_j)$ pour j = 1 à 8

- L'étape de remplacement consiste à sélectionner les 10 meilleurs individus parmi les 18 pour constituer la génération suivante.

4.3.6 Algorithme évolutionnaire

Exemple

Minimisation de $f(x) = x^2$



Population initiale (génération k).

La population initiale comporte 10 individus (parents).

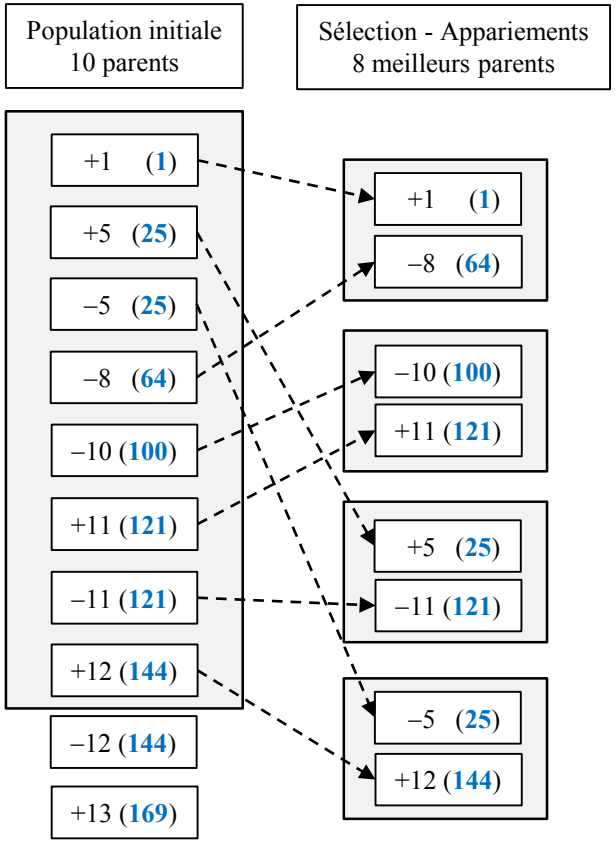
Chaque individu est représenté par
- une valeur $x_{i,\ i=1\ à\ 10}$ comprise entre −16 et +16
- son évaluation ou performance $\mathbf{f(x_i)}$

Les individus sont triés par performance décroissante (valeur de f croissante car f est à minimiser).

4.3.6 Algorithme évolutionnaire

Exemple

Minimisation de $f(x) = x^2$



Sélection pour la reproduction

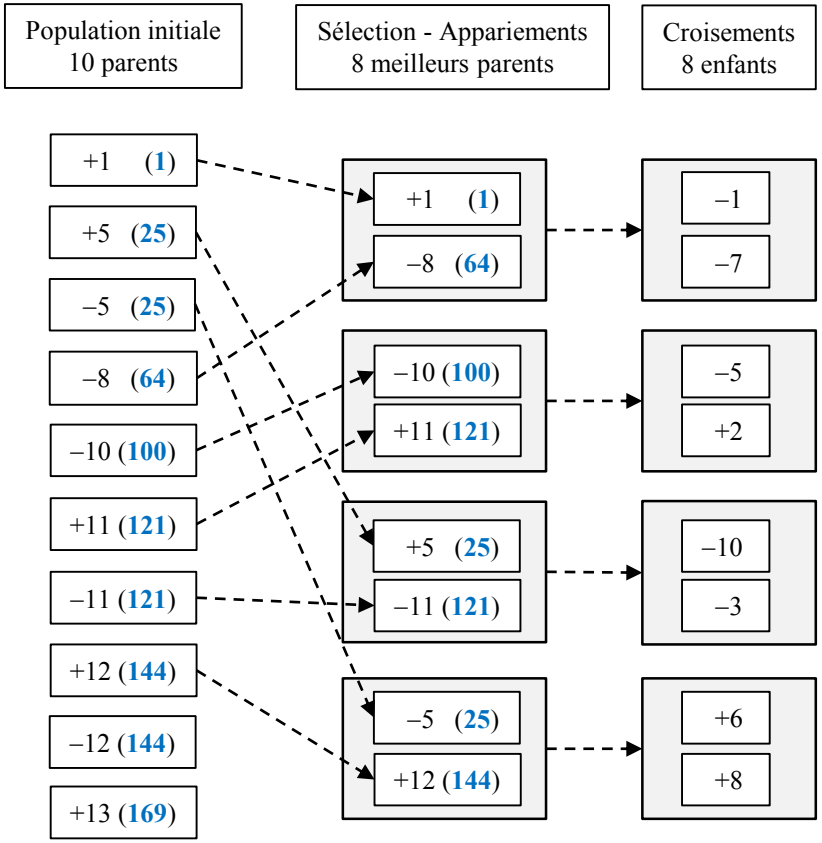
Les 8 meilleurs parents sont sélectionnés,
puis appariés aléatoirement 2 à 2.
→ sélection déterministe

Méthodes de sélection possibles
- Proportionnelle
  Un parent peut être sélectionné plusieurs fois proportionnellement à sa performance.
- Tournois
  Plusieurs parents sont tirés pour le tournoi.
  Le meilleur parent du tournoi est sélectionné.

4.3.6 Algorithme évolutionnaire

Exemple

Minimisation de $f(x) = x^2$



Croisement

2 parents engendrent 2 enfants par croisement.
Pour chaque enfant, la valeur de x est tirée aléatoirement entre celles des parents.

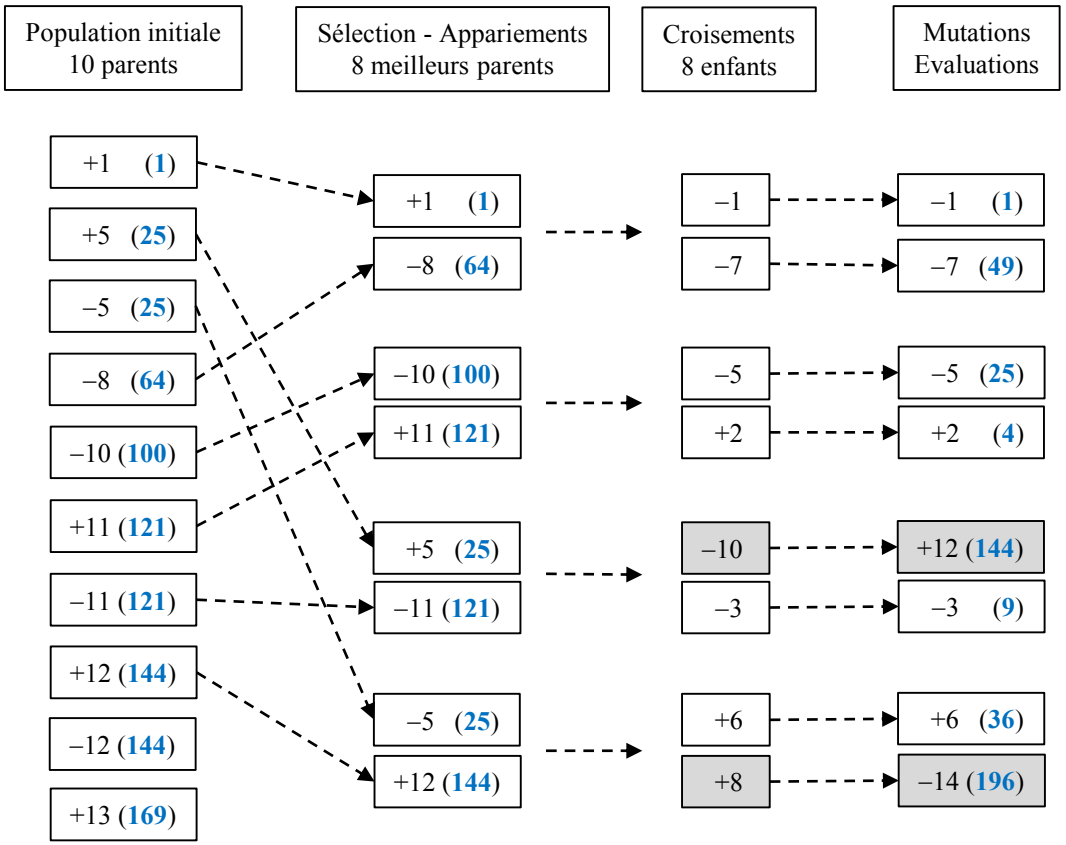
L'opérateur de croisement doit être adapté au problème :

- variables entières ou réelles
- fonction continue , minima locaux
- proximité parents / enfants

4.3.6 Algorithme évolutionnaire

Exemple

Minimisation de $f(x) = x^2$



Mutation

2 enfants sont modifiés aléatoirement.
Les 8 enfants sont ensuite évalués.

L'opérateur de mutation peut être uniforme ou gaussien.

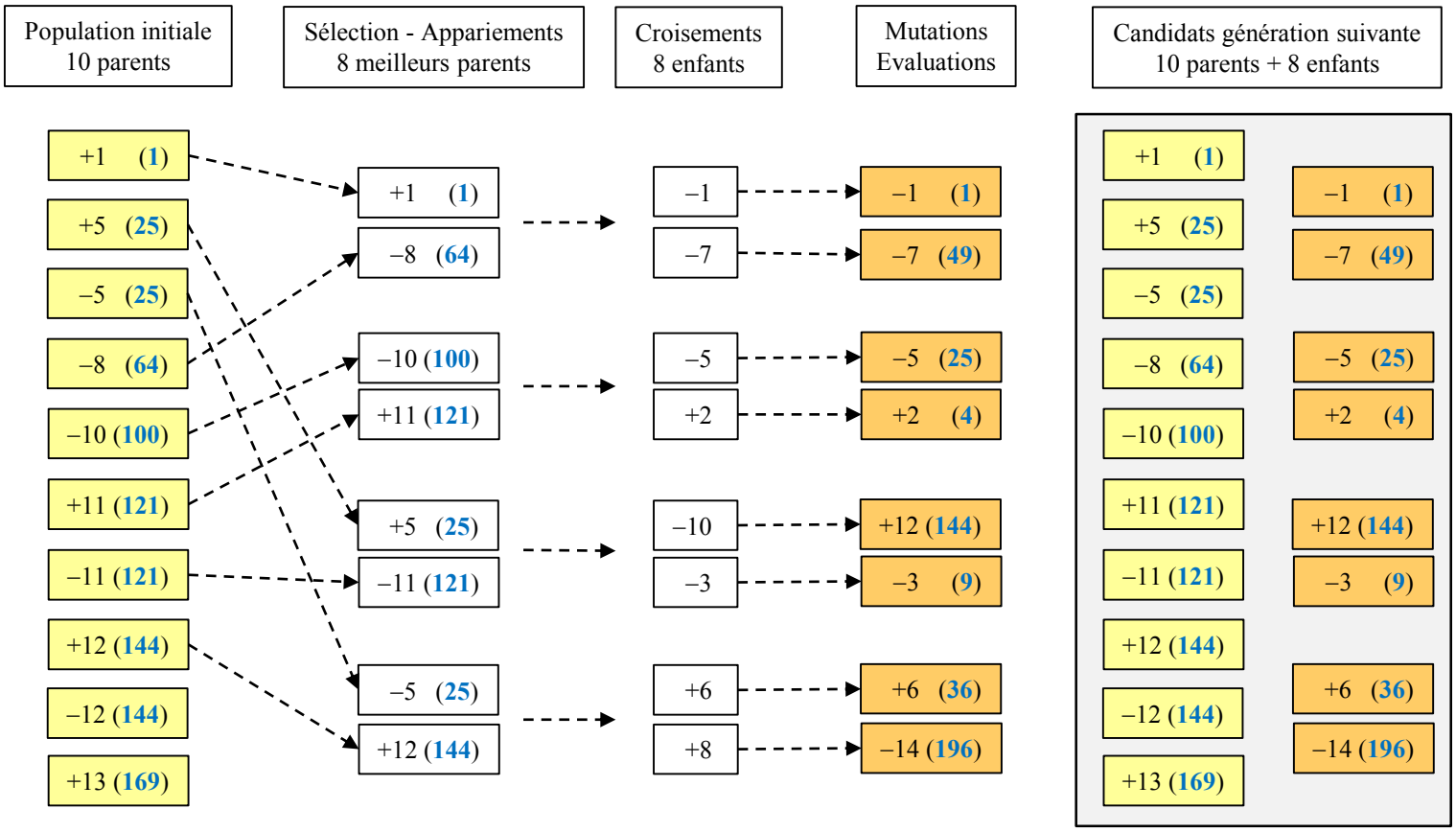
Pour une mutation gaussienne, l'écart-type σ évolue selon le taux τ de mutations favorables.

Règle des 1/5
- $\tau > 1/5$ → augmenter σ
- $\tau < 1/5$ → diminuer σ

4.3.6 Algorithme évolutionnaire

Exemple

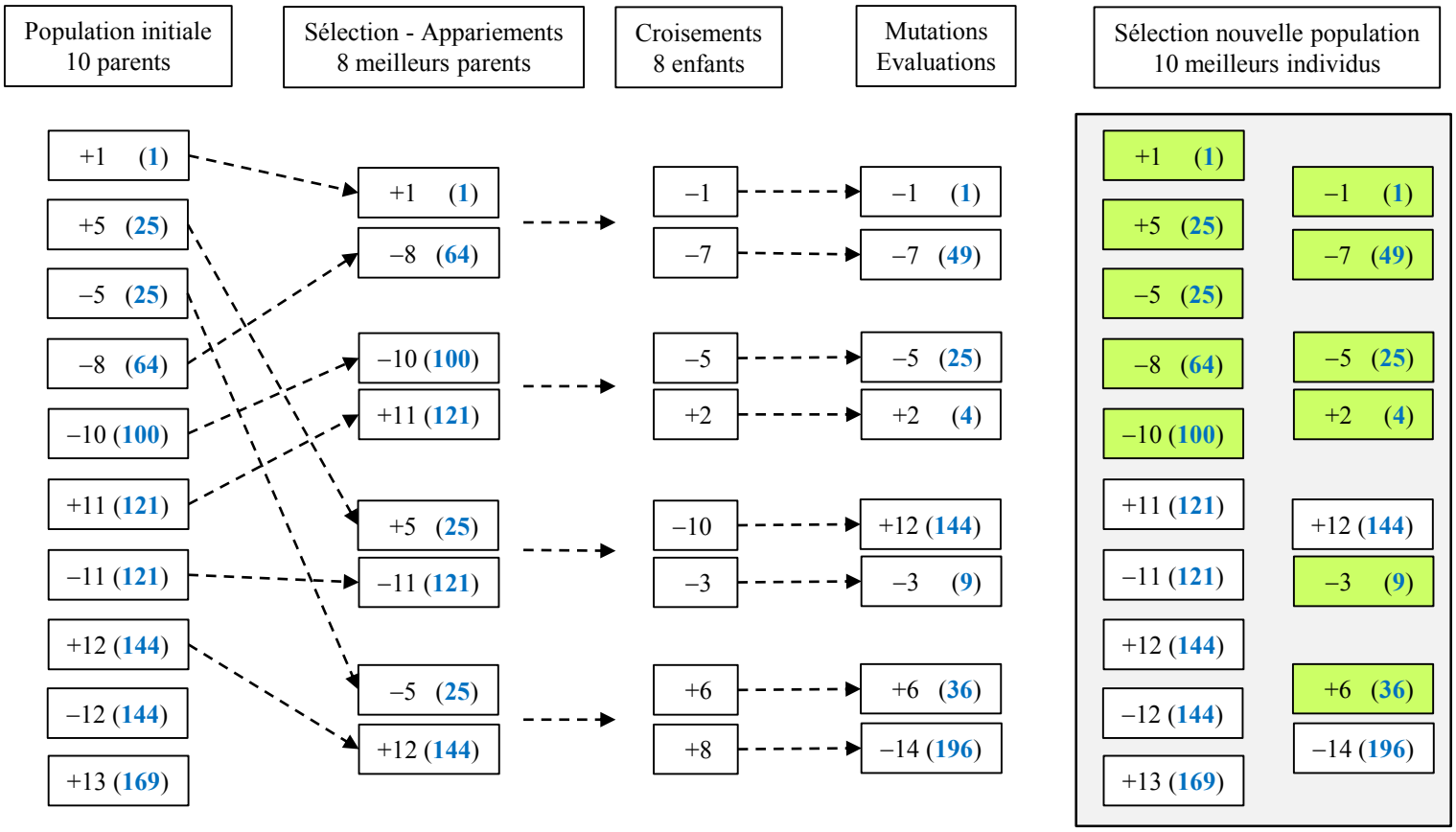
Minimisation de $f(x) = x^2$



4.3.6 Algorithme évolutionnaire

Exemple

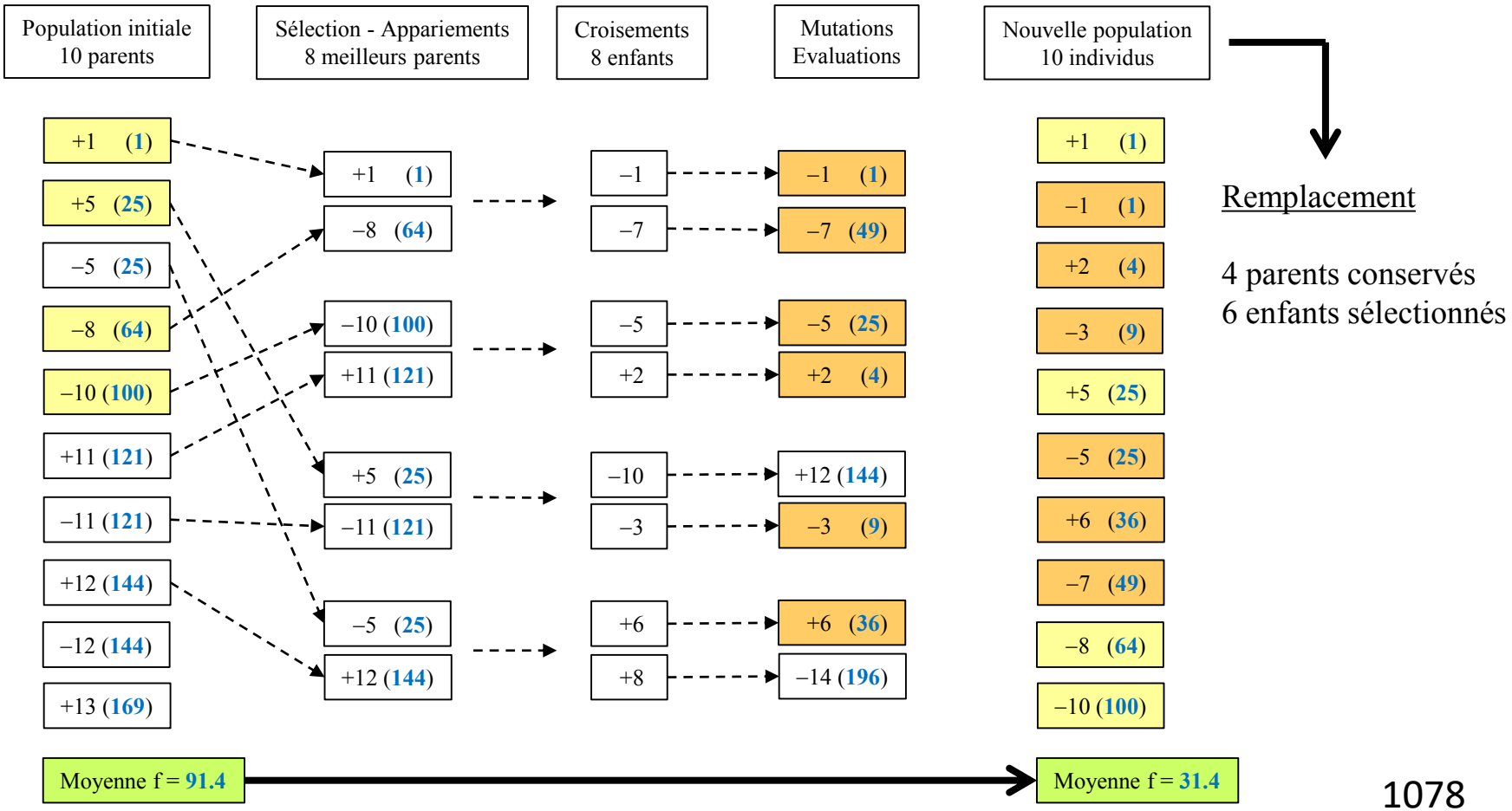
Minimisation de $f(x) = x^2$



4.3.6 Algorithme évolutionnaire

Exemple

Minimisation de $f(x) = x^2$



Remplacement

4 parents conservés
6 enfants sélectionnés

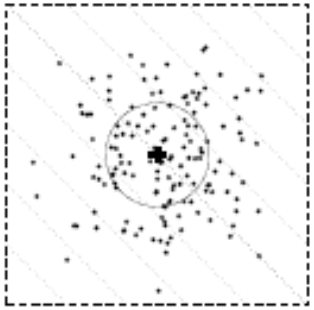
4.3.7 Adaptation de covariance

Principe

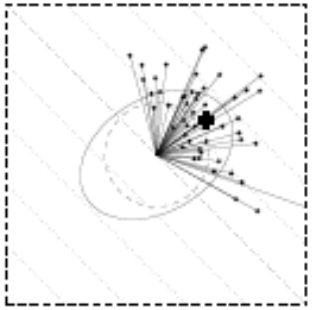
La méthode d'adaptation de covariance (« CMAES », Hansen et Ostermeier 2001) combine un algorithme évolutionnaire avec une méthode de descente.

- CMAES = « Covariance Matrix Adaptation Evolution Strategy »
  L'aspect **évolutionnaire** (ES) permet une **diversification** par génération aléatoire de candidats.
  L'aspect **descente** (CMA) permet une **intensification** vers le minimum local.

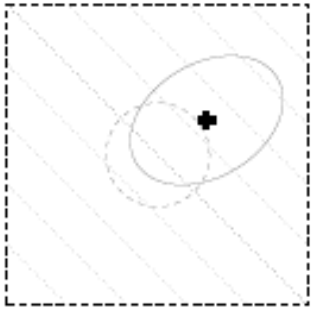
- Une itération se déroule en 3 étapes :
  - tirage de candidats suivant une loi normale
  - sélection des meilleurs candidats (minimisation de f(x))
  - mise à jour (adaptation) de la moyenne et la covariance



Tirage



Sélection

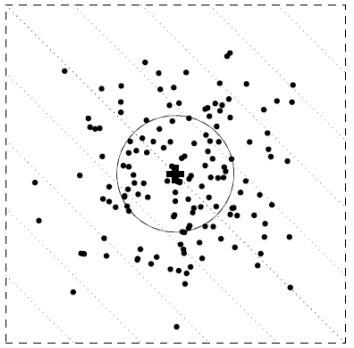


Adaptation

4.3.7 Adaptation de covariance

Echantillon de candidats

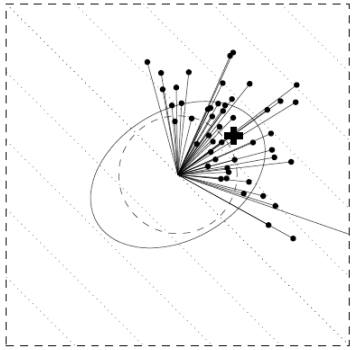
- Les paramètres de la distribution au début de l'itération sont :
  - la moyenne : m
  - la covariance : C
  - le pas d'exploration : σ



- On génère un **échantillon** de p candidats : $x_i = m + \sigma y_i$, i = 1 à p
  Les variables y suivent une loi normale : $y_i \leftarrow N(0,C)$

- Les q meilleurs candidats sont sélectionnés : $x_j = m + \sigma y_j$, j = 1 à q
  $f(x_1) \leq f(x_2) \leq \ldots \leq f(x_q) \leq \ldots \leq f(x_p)$

- Les variables de déplacement y_j associées aux q meilleurs candidats permettent de :
  - construire une direction de descente
  - adapter la matrice de covariance



- Les **paramètres m, C, σ** de la distribution sont mis à jour à partir :
  - des valeurs précédentes de m, C, σ
  - de la moyenne et de la covariance des y_j , j= 1 à q
  - du taux de réussite de l'échantillon → $f(x_i) < f(m)$

4.3.7 Adaptation de covariance

Moyenne et pas

- La **direction de descente** d est la moyenne pondérée des q meilleurs déplacements.

$$d = \sum_{j=1}^{q} w_j y_j \qquad \rightarrow \text{ poids } w_1, w_2, \ldots, w_q \text{ associés aux déplacements } y_1, y_2, \ldots, y_q$$

(y_1 = meilleur candidat)

- Mise à jour de la **moyenne m**

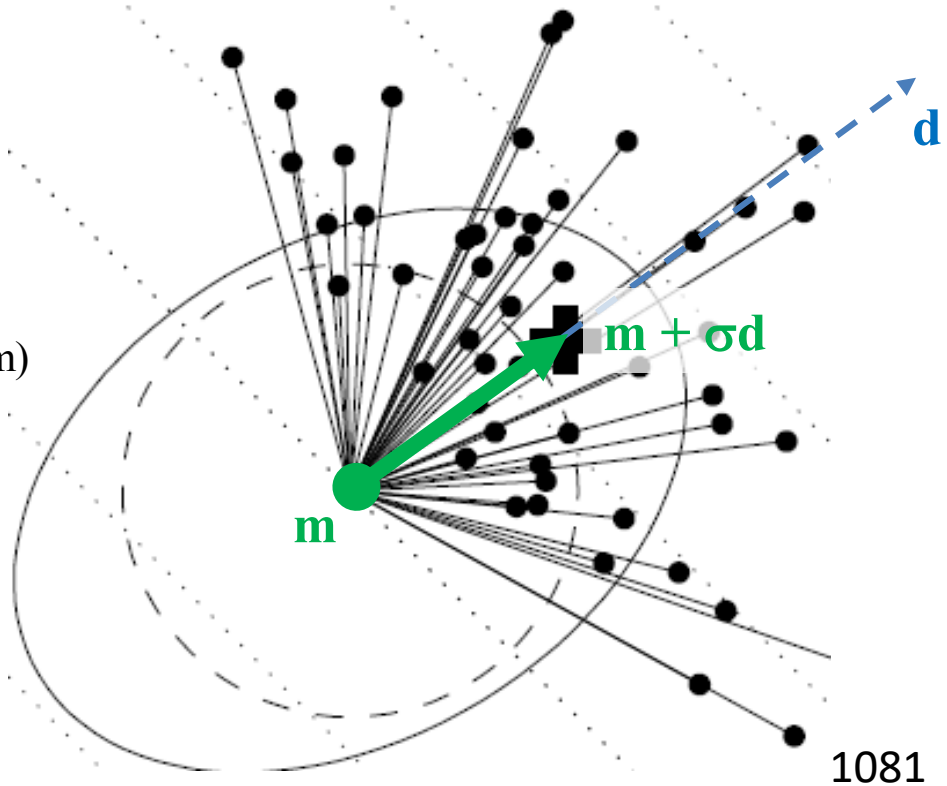
$$m \rightarrow m + \sigma d$$

- Mise à jour du **pas σ**

τ = taux de réussite de l'échantillon
= part de candidats x_i tels que $f(x_i) < f(m)$

Règle des 1/5
- Si $\tau > 1/5$, on augmente σ.
- Si $\tau < 1/5$, on diminue σ.

$$\sigma \rightarrow \alpha_\sigma \sigma \quad \text{avec} \quad \alpha_\sigma = e^{\frac{1}{3}\cdot\frac{\tau-\tau_s}{1-\tau_s}}, \tau_s = 1/5$$



4.3.7 Adaptation de covariance

Covariance

- La covariance pondérée des q meilleurs déplacements est C_q.

$$C_q = \sum_{j=1}^{q} w_j \left(y_j . y_j^T \right) \quad \rightarrow \text{ matrice de rang q}$$

- La direction d définit une matrice C_d.

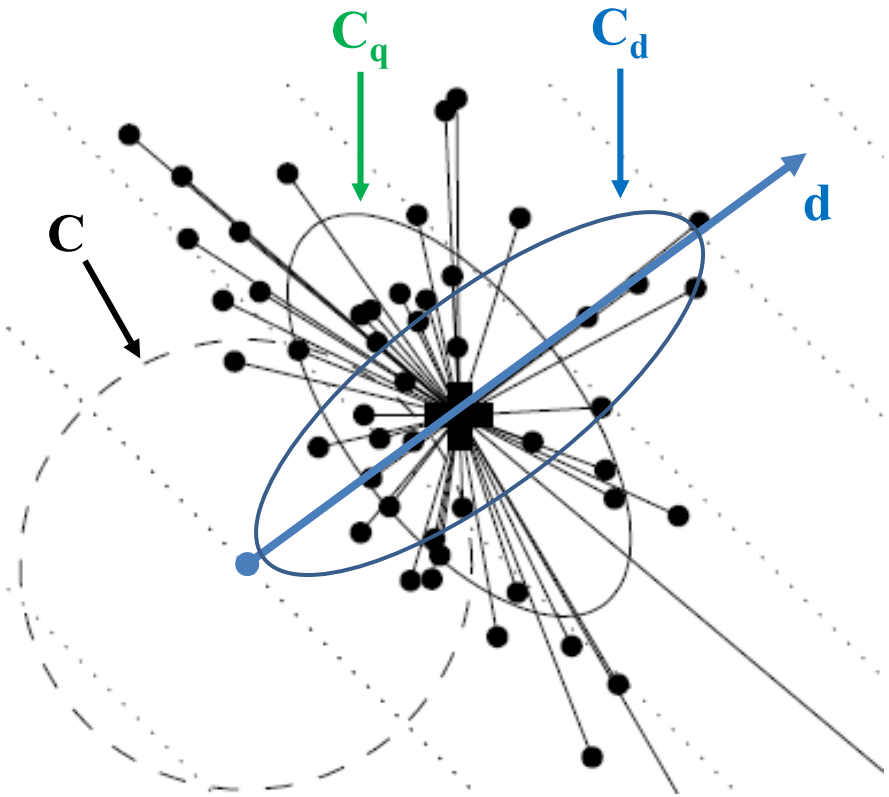
$$C_d = d.d^T \quad \rightarrow \text{ matrice de rang 1}$$

(≈ mise à jour hessien par méthode SR1)

- Mise à jour de la covariance C

$$C \rightarrow (1-\alpha_q-\alpha_d)C + \alpha_q C_q + \alpha_d C_d$$

avec des pondérations α_q , α_d



4.3.7 Adaptation de covariance

Itération CMAES

- Mise à jour de m, C, σ

$$m \rightarrow m + \sigma d \quad \text{avec } d = \sum_{j=1}^{q} w_j y_j$$

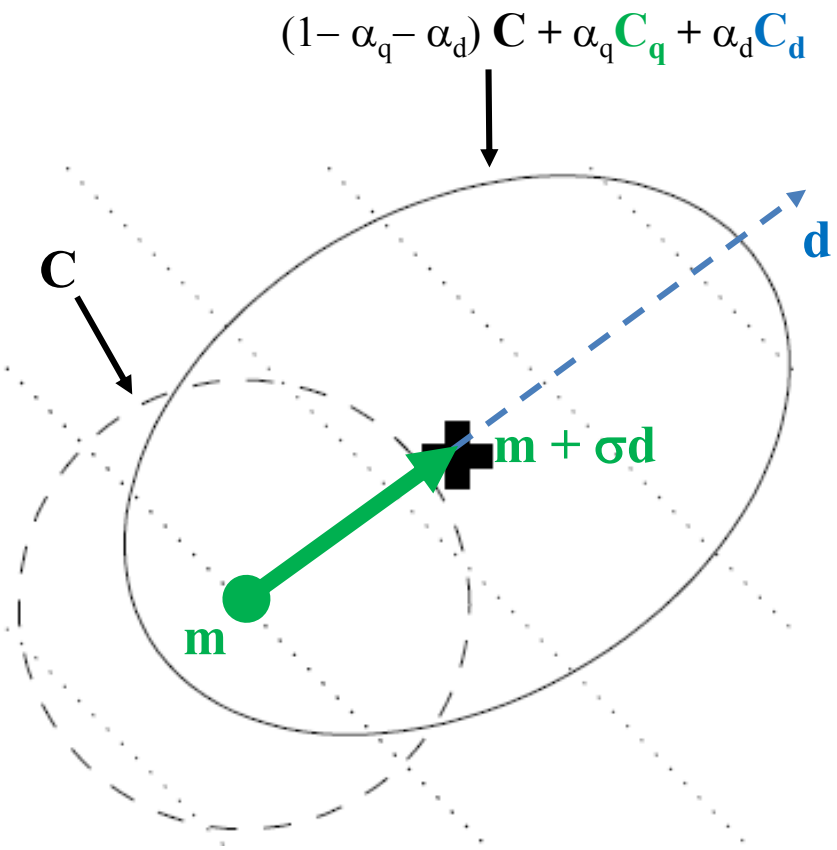
$$\sigma \rightarrow \alpha_\sigma \sigma \quad \text{avec } \alpha_\sigma = e^{\frac{1}{3}\cdot\frac{\tau-\tau_s}{1-\tau_s}}, \tau_s = 1/5$$

$$C \rightarrow (1-\alpha_q-\alpha_d)C + \alpha_q C_q + \alpha_d C_d$$

$$C_q = \sum_{j=1}^{q} w_j \left(y_j . y_j^T\right) \rightarrow \text{rang q}$$

$$C_d = d.d^T \rightarrow \text{rang 1}$$

- Direction moyenne
  On peut remplacer d par $(1-\alpha_p)d_p + \alpha_c d$
  → pondération entre direction précédente d_p et direction courante d_c
  → moyenne des directions successives



4.3.7 Adaptation de covariance

Algorithme

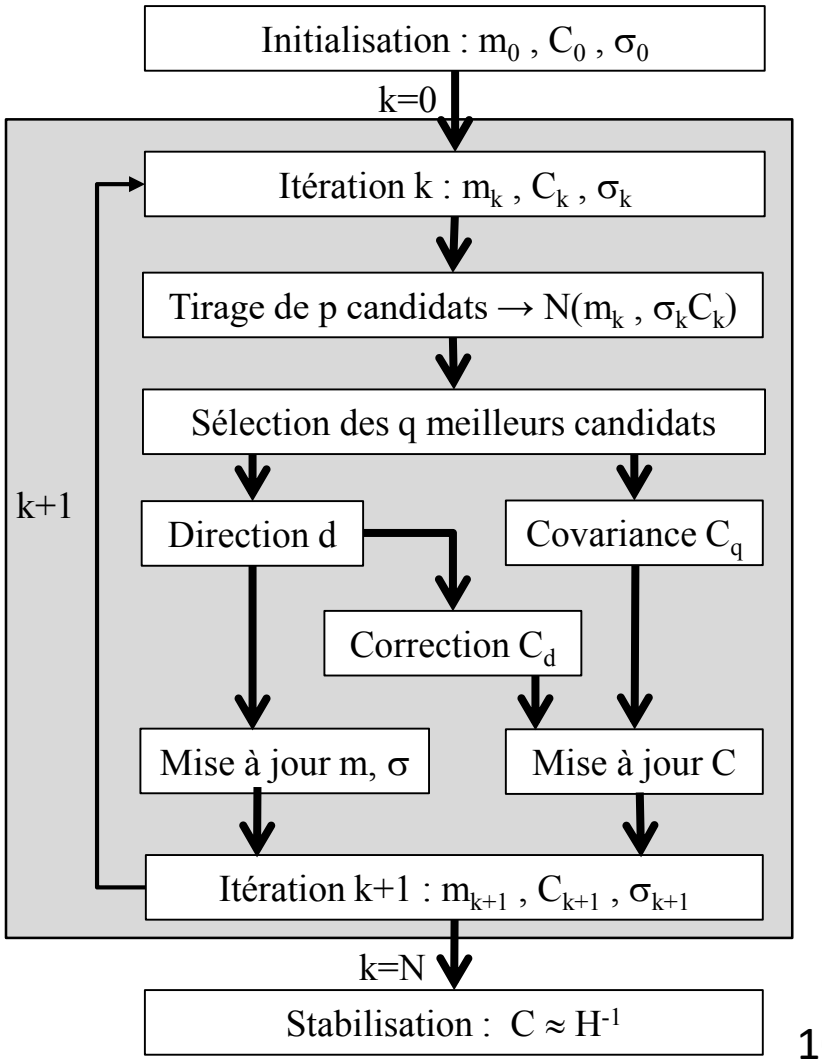
Paramètres de l'algorithme

- Tailles
  p = taille échantillon
  q = taille sélection

- Poids
  w_j = poids sur $y_{j\ ,\ j=1\ à\ q}$

- Pondérations
  $\alpha_\sigma \rightarrow$ pas σ
  $\alpha_q \rightarrow$ correction covariance C_q
  $\alpha_d \rightarrow$ correction covariance C_d
  $\alpha_p \rightarrow$ direction précédente d_p

  Réglages standards expérimentaux
  en fonction de la taille du problème



4.3.7 Adaptation de covariance

Relation covariance - hessien

- Au voisinage du minimum m de f : $f(x) \approx f(m) + \frac{1}{2}(x-m)^T \nabla^2 f(m)(x-m)$ avec $\nabla f(m) = 0$

  Les **lignes de niveau** de f vérifient : $\boxed{(x-m)^T H (x-m) = C^{te}}$ avec $H = \nabla^2 f(m)$

- La densité de la loi normale N(m , C) à n variables est : $p(x) = \frac{1}{(2\pi)^{\frac{n}{2}} |C|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-m)^T C^{-1} (x-m)}$

  Les **lignes d'iso-densité** vérifient : $\boxed{(x-m)^T C^{-1} (x-m) = C^{te}}$

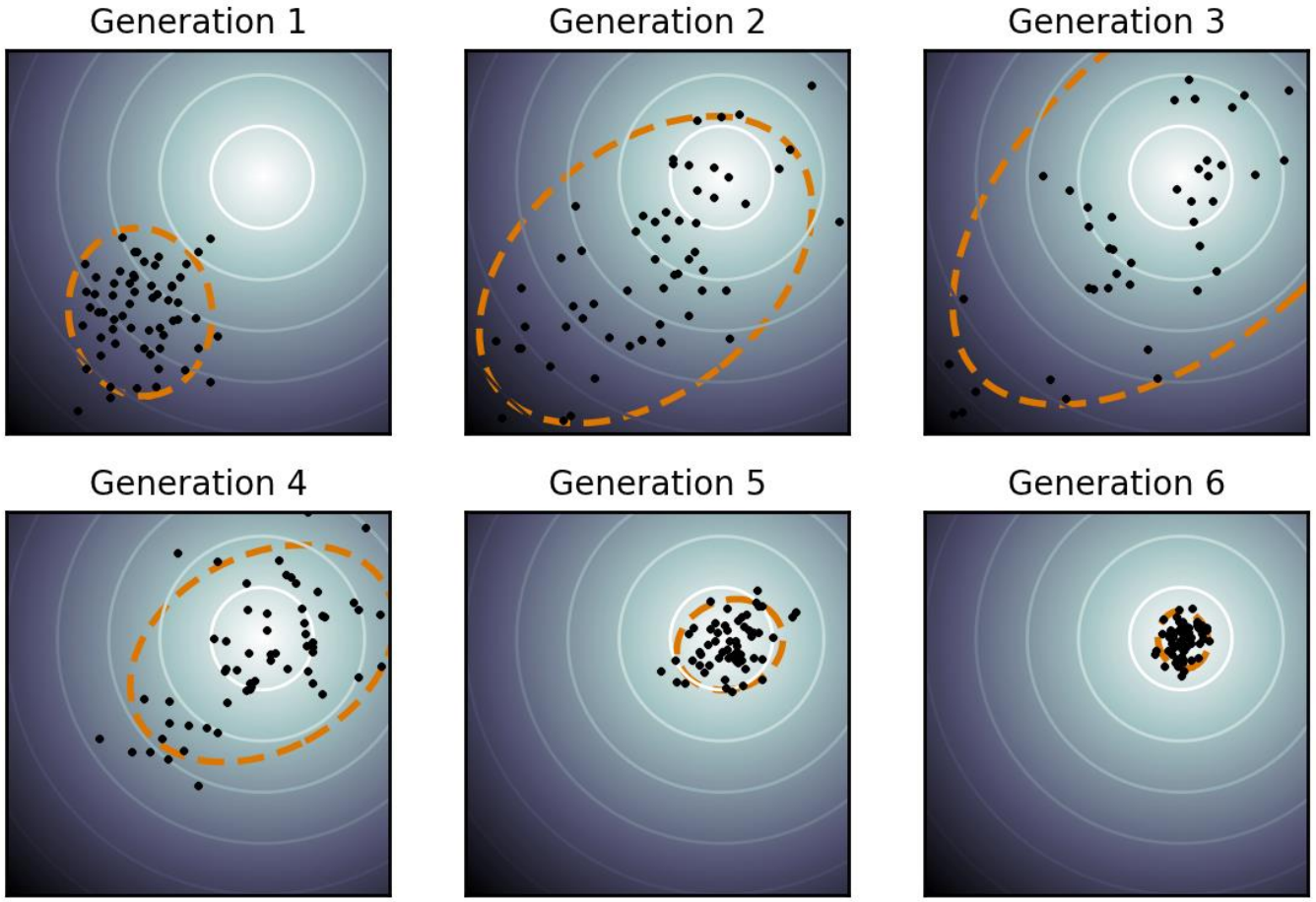
- La direction de déplacement d est définie à partir d'un échantillon de loi N(m , σC).
  Au voisinage du minimum, si la covariance C est proche de H^{-1} (à un facteur près),
  l'échantillon $x_{i,\ i = 1 \text{ à } p}$ suit les lignes de niveau de f.
  → pas de direction privilégiée (d ≈ 0)
  → stabilisation de m et C

- Lorsque les itérations CMAES se stabilisent, on observe en pratique que : **$C \approx \alpha H^{-1}$**
  → résultat expérimental, pas de preuve théorique de convergence

4.3.7 Adaptation de covariance

Relation covariance - hessien

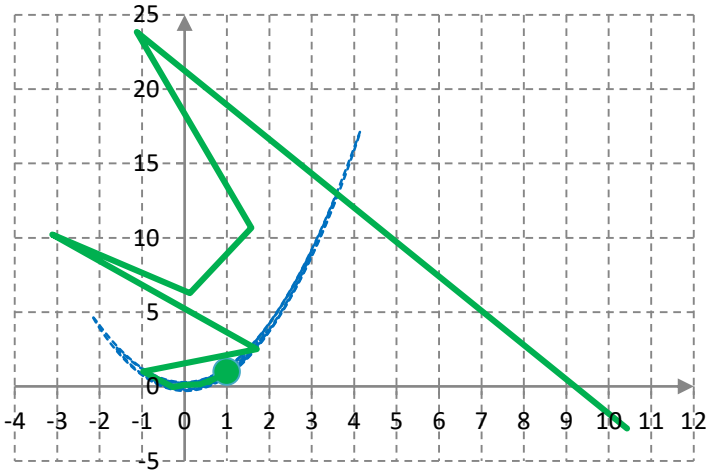
Convergence de la covariance vers un multiple du hessien : $C \approx \alpha H^{-1}$

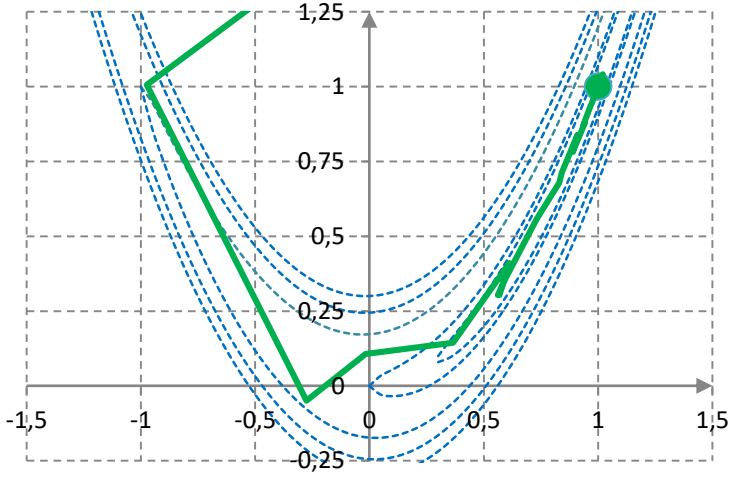


4.3.7 Adaptation de covariance

Exemple

Fonction de Rosenbrock à deux variables : $f(x_1, x_2) = 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2$





- Covariance : $C^{-1} = \begin{pmatrix} 240.5 & -119.4 \\ -119.4 & 59.5 \end{pmatrix} .10^9$
- Hessien : $H = \begin{pmatrix} 802 & -400 \\ -400 & 200 \end{pmatrix}$

- Nombre d'itérations : 140
- Nombre d'évaluations : 800

→ Solution : $x_1 = 0.99999936$
$x_2 = 0.99999938$

4.3.8 Affine shaker

Principe

La méthode Affine Shaker (Battiti et Tecchiolli 1994) est une **recherche aléatoire locale**.
La région de recherche est modifiée en fonction de l'amélioration constatée sur la fonction.

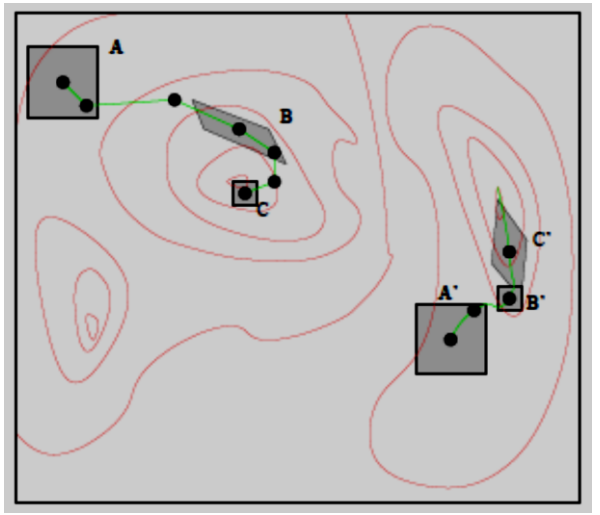
« shaker » ← mouvement aléatoire
« affine » ← transformation affine de la région de recherche

- La région de recherche B est centrée sur le point courant x_0 → $f_0 = f(x_0)$
  Un déplacement aléatoire d est tiré dans la région de recherche.

- La fonction est évaluée en $x_1 = x_0 + d$ → $f_1 = f(x_1)$
  Si $f_1 < f_0$, le point x_1 est sélectionné.
  La région B est **dilatée** suivant la direction d.

- Sinon on évalue le point $x_2 = x_0 - d$ → $f_2 = f(x_2)$
  Si $f_2 < f_0$, le point x_2 est sélectionné.
  La région B est **dilatée** suivant la direction d.

- Sinon la région B est **contractée** suivant la direction d.
  Le point x_0 est conservé.



4.3.8 Affine shaker

Région de recherche

- La région de recherche B est définie dans R^n par **n vecteurs indépendants** $b_1, \dots, b_n$.

$$B = (b_1, \cdots, b_n) \quad \text{avec} \quad b_i \in R^n$$

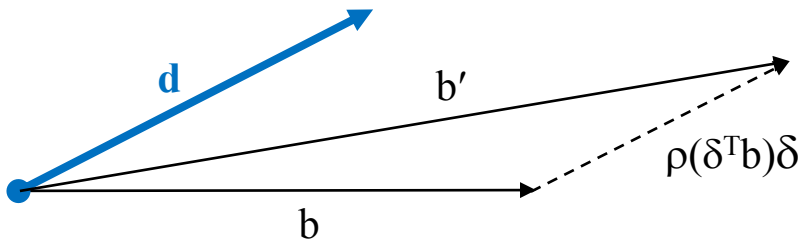
- Le déplacement d est une combinaison linéaire des b_i avec des coefficients aléatoires β_i.
  Les coefficients β_i sont tirés uniformément dans $[-1,+1]$.

$$d = \sum_{i=1}^{n} \beta_i b_i \quad \text{avec} \quad -1 \le \beta_i \le 1$$

- La région B est dilatée ou contractée avec un coefficient ρ suivant la direction d.
  On ajoute à chaque vecteur b une composante proportionnelle au produit scalaire $d^T b$.

$$b \rightarrow \boxed{b' = b + \rho(\delta^T b)\delta} \quad \text{avec} \quad \delta = \frac{d}{\|d\|} \quad \rightarrow \text{ vecteur unitaire suivant d}$$

$$\Rightarrow \quad b' = b + \rho \frac{d^T b}{\|d\|^2} d$$



- Dilatation : $\rho = \rho_d > 0 \quad (\rho_d = +0.5)$
  Contraction : $\rho = \rho_c < 0 \quad (\rho_c = -0.5)$

4.3.8 Affine shaker

Région de recherche

- On applique la transformation à chacun des vecteurs $b_1, \dots, b_n$ définissant la région B.

$$b'_i = b_i + \rho(\delta^T b_i)\delta \quad , \quad i = 1 \text{ à } n$$

- **Sous forme matricielle**, en notant B la matrice n×n des vecteurs colonnes $b_1, \dots, b_n$

$$B = \begin{pmatrix} b_1 & b_2 & \cdots & \cdots & b_n \end{pmatrix} = \begin{matrix} b_1 & b_2 & & & b_n \\ \end{matrix} \begin{pmatrix} b_{11} & b_{21} & \cdots & \cdots & b_{n1} \\ b_{12} & b_{22} & \cdots & \cdots & b_{n2} \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ b_{1n} & b_{2n} & \cdots & \cdots & b_{nn} \end{pmatrix} \quad \text{et} \quad \delta = \begin{pmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \vdots \\ \delta_n \end{pmatrix}$$

$$\begin{cases} b'_1 = b_1 + \rho\delta(\delta^T b_1) \\ b'_2 = b_2 + \rho\delta(\delta^T b_2) \\ \vdots \\ b'_n = b_n + \rho\delta(\delta^T b_n) \end{cases} \Leftrightarrow B' = B + \rho\delta(\delta^T B) = B + \rho(\delta\delta^T)B = (I + \rho\delta\delta^T)B$$

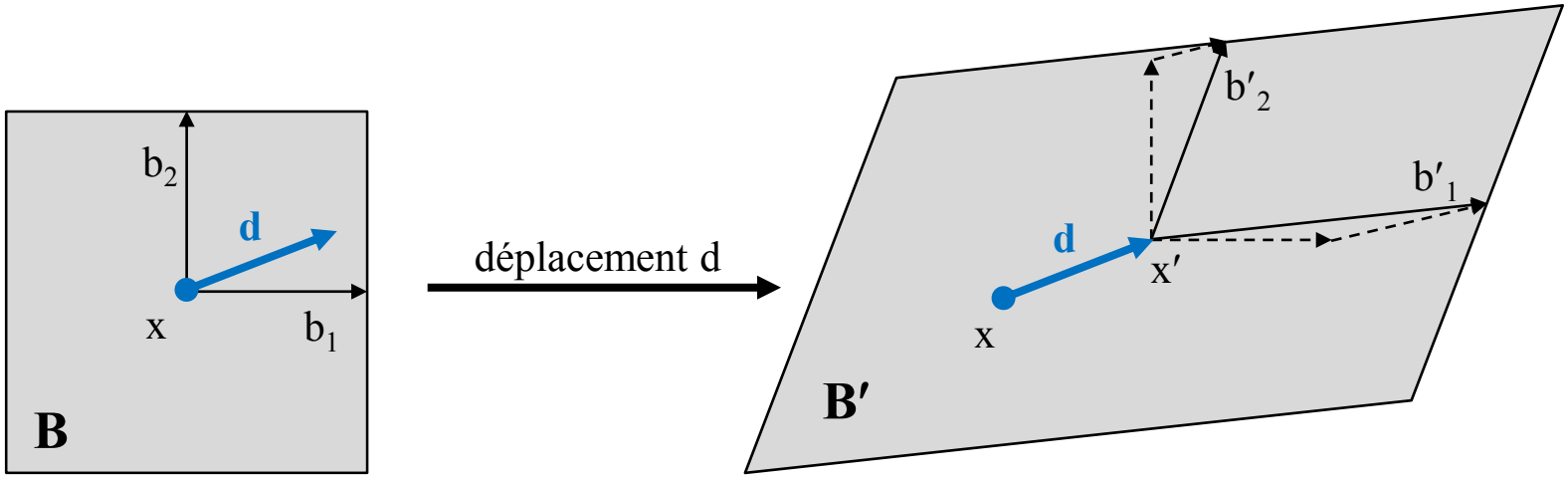
$$\Leftrightarrow \boxed{B' = PB \quad \text{avec} \quad P = I + \rho\delta\delta^T = I + \rho\frac{dd^T}{\|d\|^2}}$$

→ **matrice de transformation P**

4.3.8 Affine shaker

Illustration en 2D

- La région initiale B est un **carré** défini par (b_1 , b_2).
- Le déplacement aléatoire d donne un point $x' = x + d$.
- Les vecteurs b_1 et b_2 deviennent b'_1 et b'_2 : $\begin{cases} b'_1 = b_1 + \rho(\delta^T b_1)\delta \\ b'_2 = b_2 + \rho(\delta^T b_2)\delta \end{cases} \Leftrightarrow B' = \left(I + \rho \frac{dd^T}{\|d\|^2} \right) B$
- La nouvelle région B′ est un **parallélogramme** dilaté suivant la direction d.

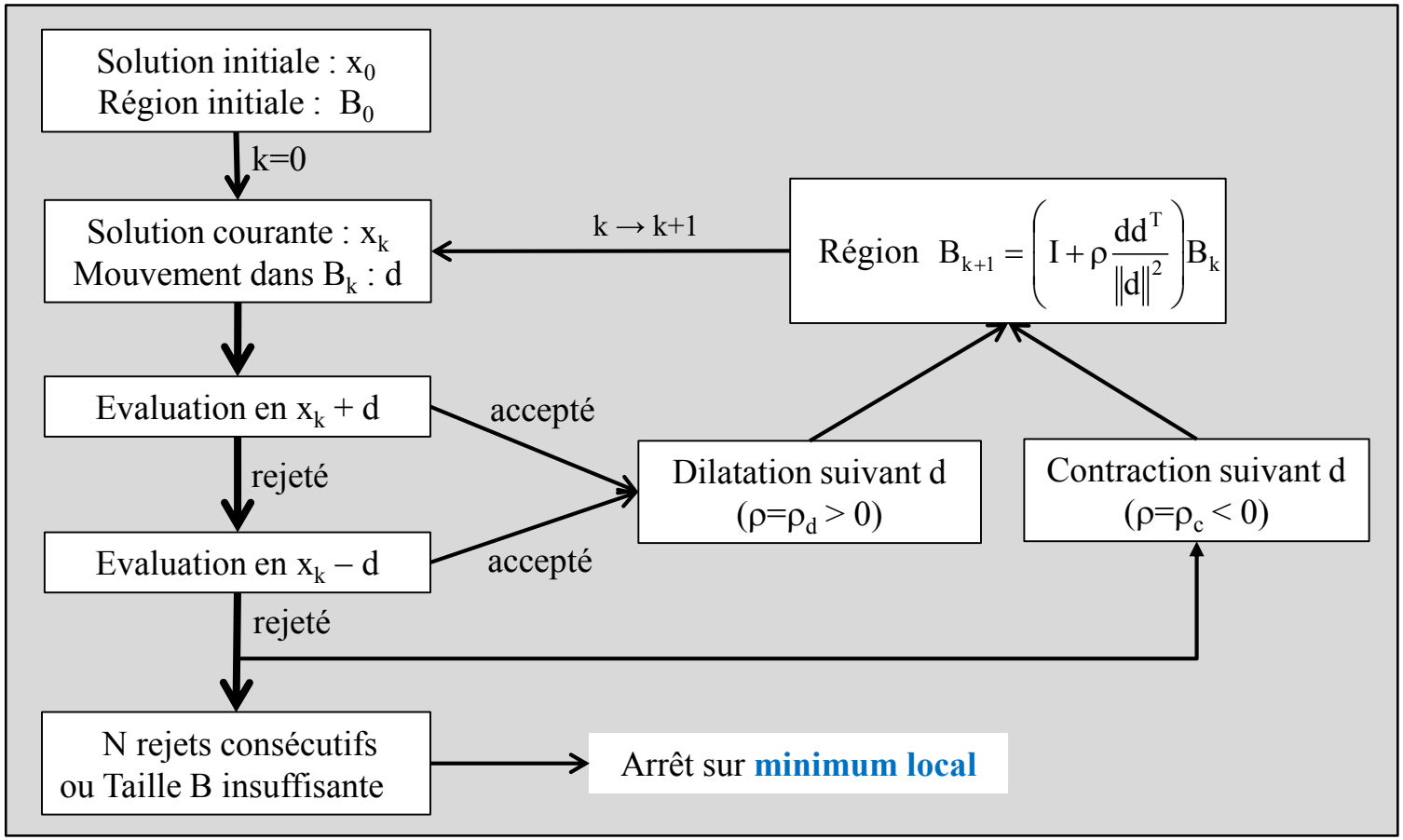


4.3.8 Affine shaker

Algorithme

L'algorithme s'applique à un problème de minimisation sans contrainte : $\min_{x \in R^n} f(x)$

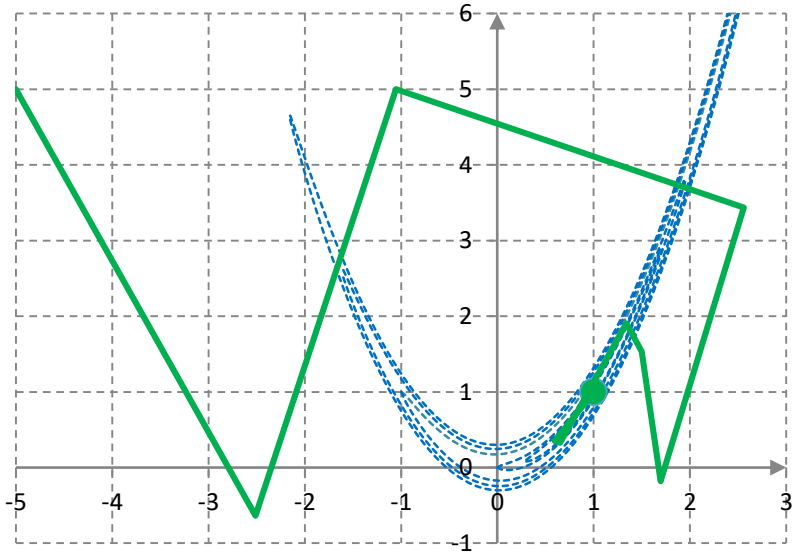
Il s'arrête sur un **minimum local** qui dépend de l'initialisation x_0 , B_0.



4.3.8 Affine shaker

Exemple

Fonction de Rosenbrock à deux variables : $f(x_1, x_2) = 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2$



N_f	f	x_1	x_2
1	40036,0	-5,0000	5,0000
3	4842,0	-2,5123	-0,6379
6	1517,8	-1,0533	5,0000
9	971,7	2,5591	3,4356
15	932,0	1,6944	-0,1809
24	51,5	1,4993	1,5322
39	1,2779	1,3401	1,9036
48	1,0386	0,6511	0,3282
402	0,1641	0,5952	0,3527
1206	0,000097	0,9903	0,9806
28944	0,000023	1,0040	1,0082
95676	0,000021	0,9978	0,9960

- Initialisation : $x_1 = -5$, $x_2 = 5$
- Progression initiale rapide : $x_1 = 0.99$ après 1200 appels fonctions, $x_2 = 0.98$
- Convergence précise lente → nécessite plusieurs milliers d'appels fonction

4.3.9 Reformulations

Problème d'optimisation difficile

$\min_x f(x)$ → problème d'optimisation (PO)

- Le problème est considéré comme « difficile » dans 2 cas.
  - Les variables sont discrètes → contrainte d'intégrité de la solution
  - La fonction admet des minima locaux → difficulté de garantir le minimum global

  Des techniques de reformulation permettent de favoriser la résolution.

- Les contraintes d'intégrité peuvent être prises en compte par pénalisation.
  → transforme le problème mixte en problème continu
  → permet d'appliquer des algorithmes sans contrainte (classiques ou métaheuristiques)

- Le minimum global peut être isolé par transformations successives.
  → dilatation de la fonction pour éliminer les minima locaux supérieurs à un seuil
  → poursuite de l'algorithme sur la fonction dilatée

- Les reformulations améliorent l'efficacité des métaheuristiques
  et/ou élargissent leur champ d'application.

4.3.9 Variables discrètes

Méthode de pénalisation

$\min_x f(x)$ → problème à une variable x

- On suppose que la variable x est astreinte à un ensemble discret de valeurs.

  $x \in D = \{d_1, d_2, ..., d_p\}$

- On définit une fonction de pénalisation φ : $\varphi(x) = 0$ si $x \in D$
  $\varphi(x) > 0$ sinon

Exemples de pénalisation : $\varphi(x) = -(x - d_j)(x - d_{j+1})$ avec j tel que $d_j \le x < d_{j+1}$

$$\varphi(x) = \sin\left(\pi \frac{x - d_j}{d_{j+1} - d_j}\right)$$

- La pénalisation crée des minima locaux pour les valeurs admissibles $x \in D$.
  On peut ensuite appliquer une méthode d'optimisation continue à la fonction pénalisée.

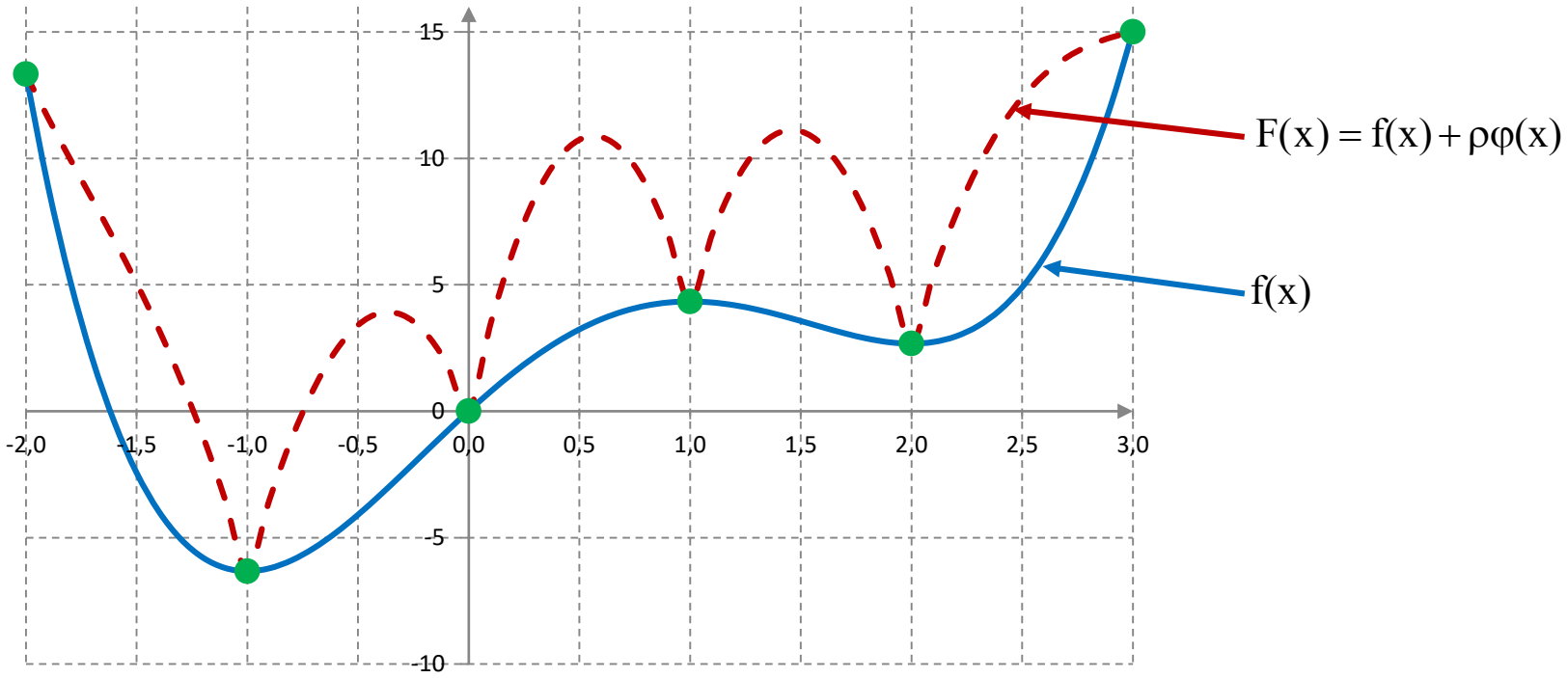
$\min_{x \in R} F(x) = f(x) + \rho\varphi(x)$ → minimisation sur la variable x continue
pénalisation ρ à régler pour forcer $x \in D$

4.3.9 Variables discrètes

Exemple

$$\min_x f(x) = x^4 - \frac{8}{3}x^3 - 2x^2 + 8x \quad \text{avec} \quad x \in \{-1, 0, 1, 2\}$$

- Fonction pénalisée avec $\varphi(x) = -(x - d_j)(x - d_{j+1})$ et $\rho = 30$

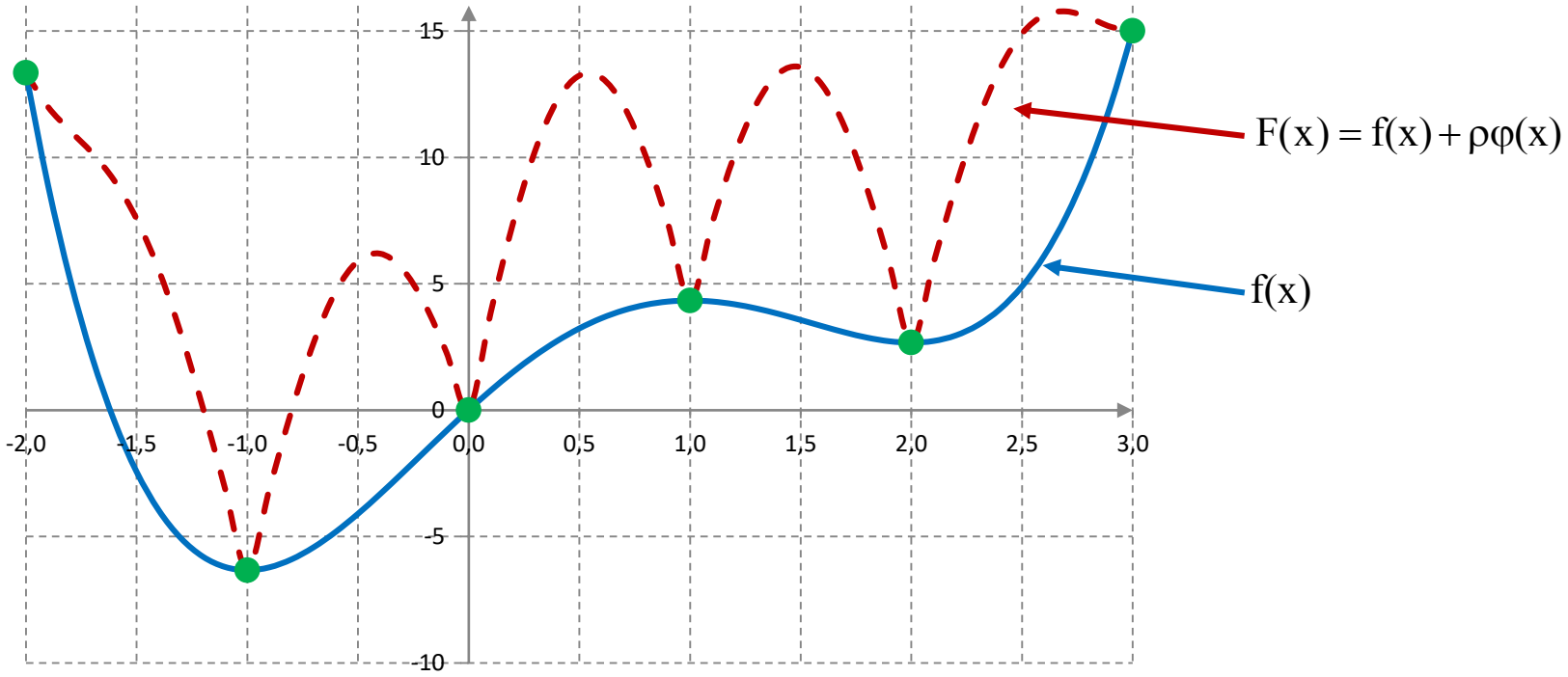


4.3.9 Variables discrètes

Exemple

$$\min_x f(x) = x^4 - \frac{8}{3}x^3 - 2x^2 + 8x \quad \text{avec} \quad x \in \{-1, 0, 1, 2\}$$

- Fonction pénalisée avec $\varphi(x) = \sin\left(\pi \frac{x - d_j}{d_{j+1} - d_j}\right)$ et $\rho = 10$



4.3.9 Minima locaux

Méthode de dilatation

$\min_x f(x)$ → problème à plusieurs minima locaux

- On suppose qu'un premier minimum local a été trouvé en $x=x_m$ → $f(x_m) = f_m$

- On définit la fonction dilatée F : $$F(x) = \begin{cases} f(x) + c_1\|x - x_m\| + \dfrac{c_2}{\|f(x) - f_m\|} & \text{si } f(x) > f_m \\ f(x) & \text{si } f(x) \le f_m \end{cases}$$

  Le terme en c_1 remonte les minima locaux supérieurs à f_m sans modifier le voisinage de x_m.
  Le terme en c_2 remonte la fonction au voisinage de x_m.
  Les minima inférieurs à f_m ne sont pas modifiés.

- Le réglage des coefficients c_1 et c_2 permet de pénaliser les régions où $f(x) > f_m$ et de favoriser l'exploration des minima restants.

- L'algorithme est redémarré sur la fonction dilatée F.
  Les dilatations successives éliminent les minima locaux et isolent le minimum global.

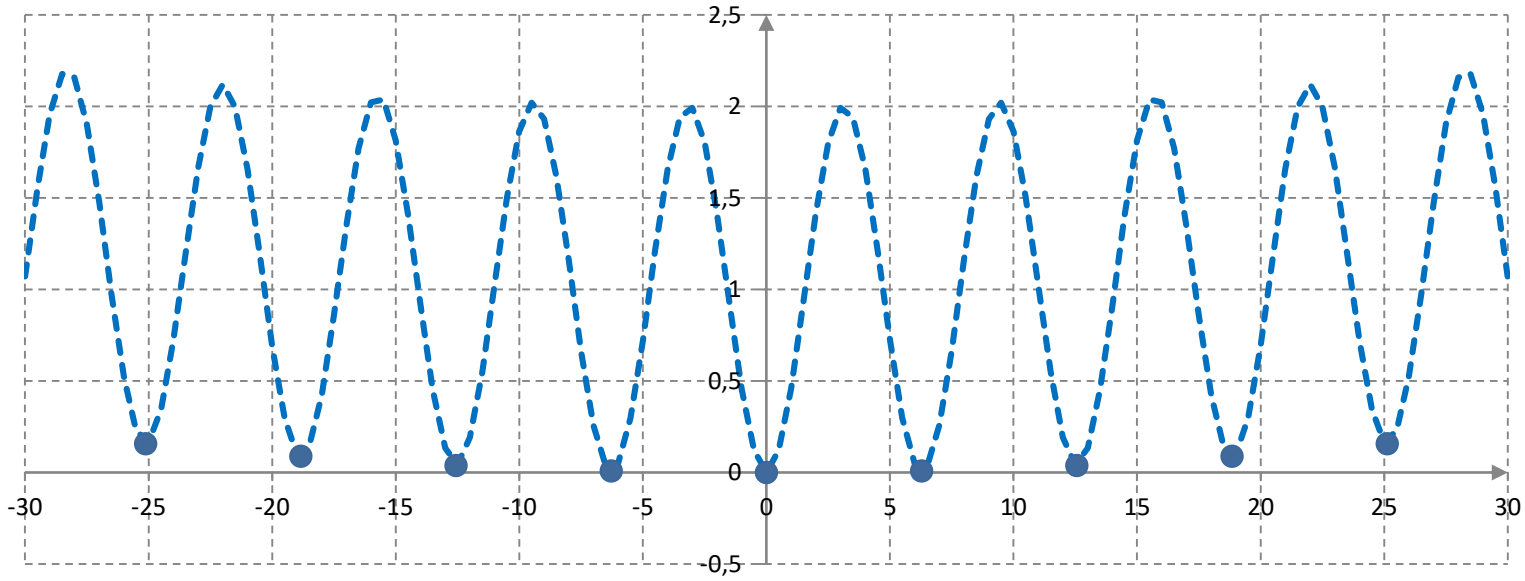
4.3.9 Minima locaux

Exemple

Fonction de Griewank à une variable $f(x) = \frac{x^2}{4000} - \cos(x) + 1$

Minima locaux : $\sin(x) + \frac{x}{2000} = 0 \quad \rightarrow$ proche de $k.2\pi$ lorsque $|x| << 2000$

x_m	-25,12	-18,84	-12,56	-6,28	0,00	6,28	12,56	18,84	25,12
f_m	1,578	0,887	0,394	0,098	0,0000	0,0986	0,3946	0,8878	1,578



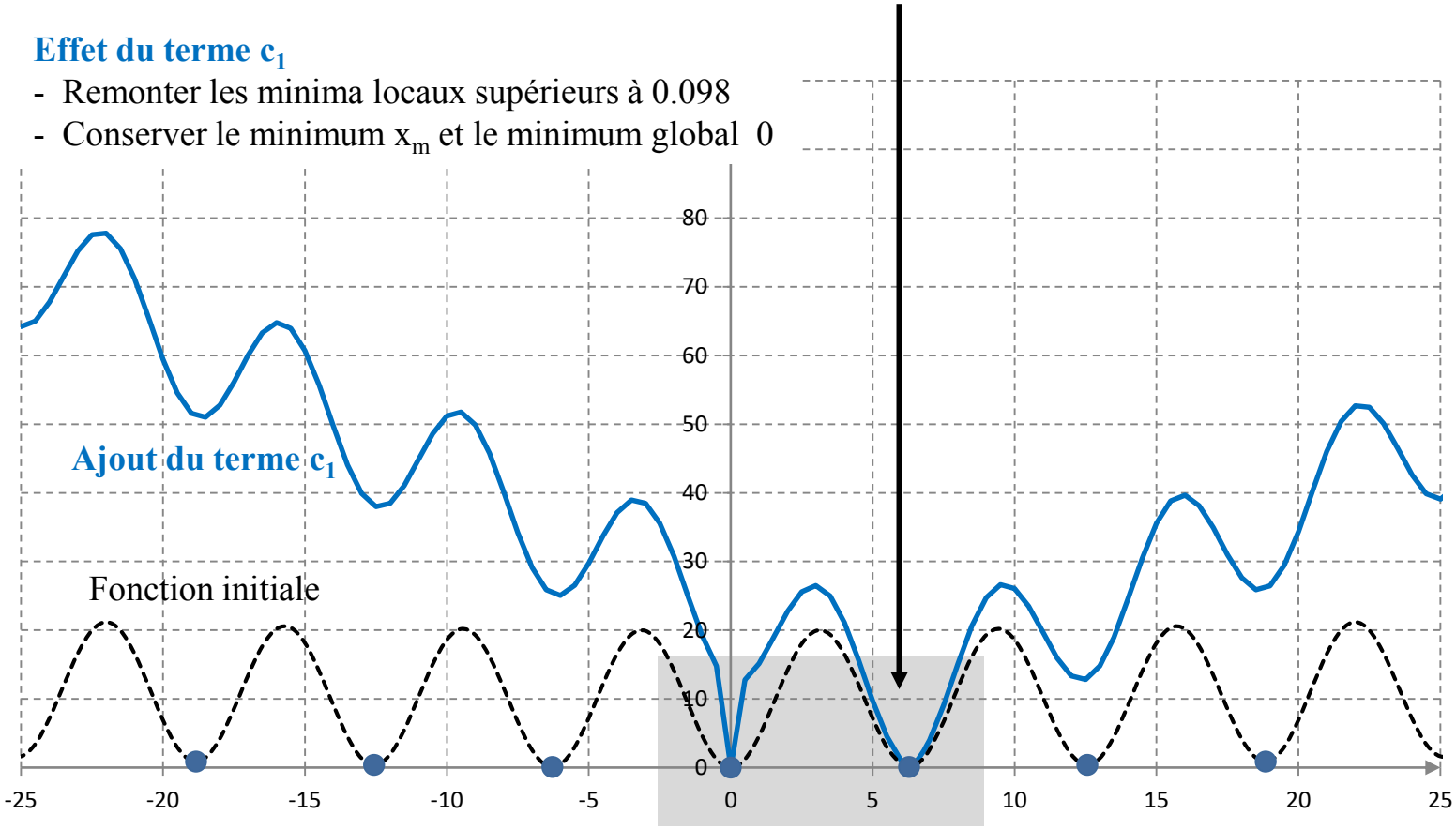
4.3.9 Minima locaux

Exemple

Fonction dilatée autour du minimum local $x_m = 6.28$, $f_m = 0.098$

Effet du terme c_1

- Remonter les minima locaux supérieurs à 0.098
- Conserver le minimum x_m et le minimum global 0



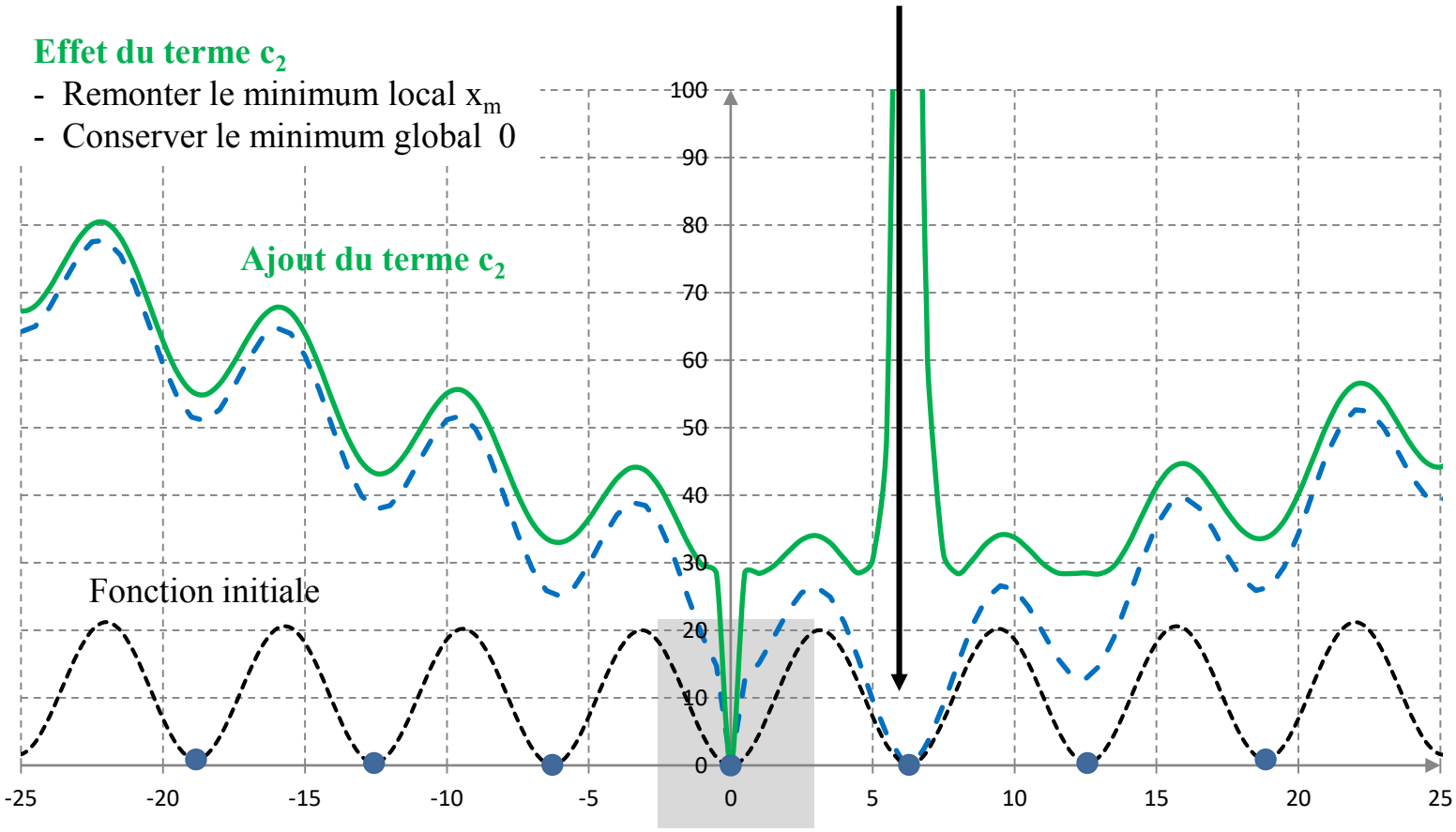
4.3.9 Minima locaux

Exemple

Fonction dilatée autour du minimum local $x_m = 6.28$, $f_m = 0.098$

Effet du terme c_2

- Remonter le minimum local x_m
- Conserver le minimum global 0

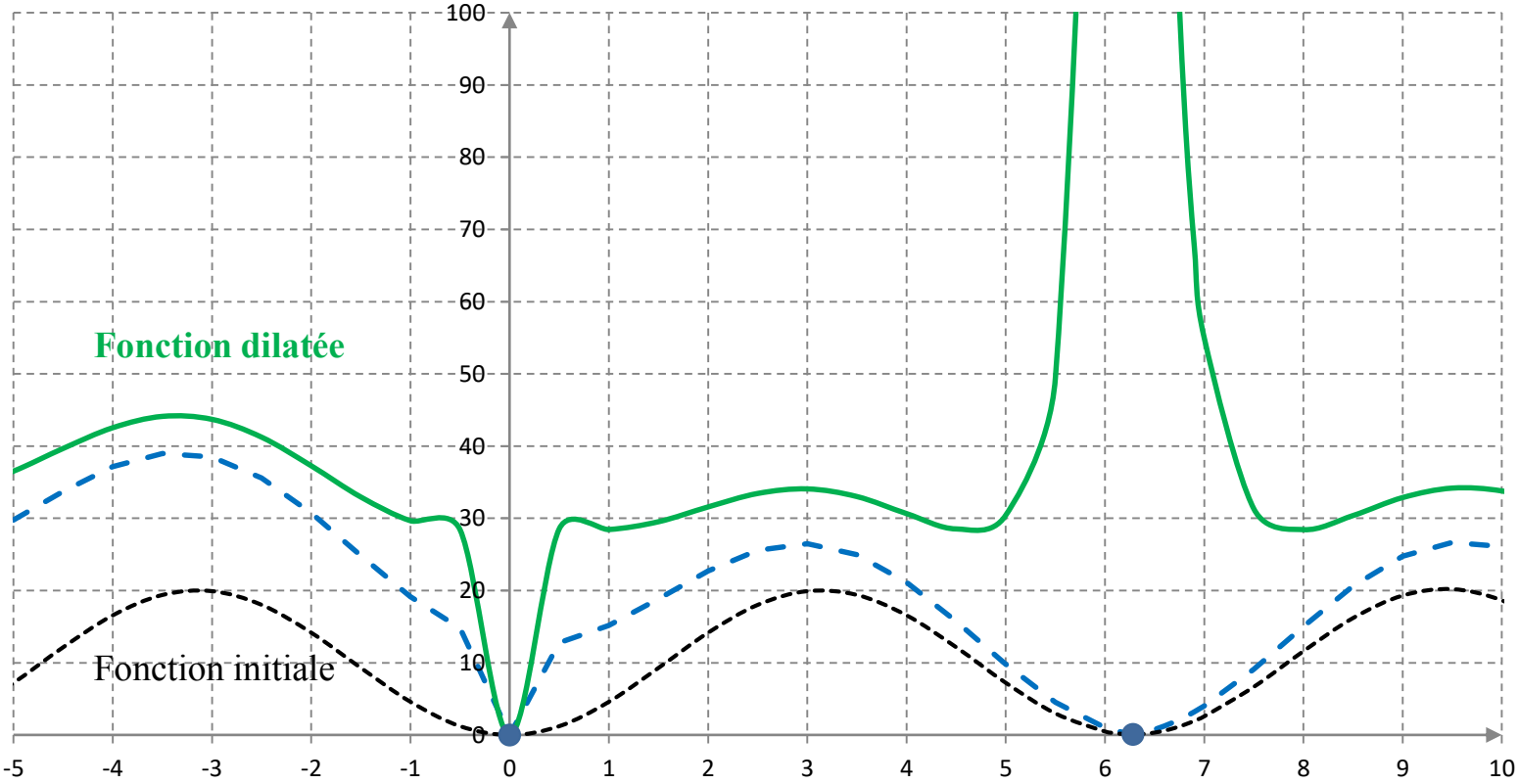


4.3.9 Minima locaux

Exemple

Fonction dilatée autour du minimum local $x_m = 6.28$, $f_m = 0.098$

Il ne reste que le minimum global en 0 (= seul minimum inférieur à f_m).



4.3.9 Minima locaux

Fonction signe

- La fonction dilatée F est de la forme : $F(x) = \begin{cases} f(x)+g(x) & \text{si } c(x) \geq 0 \\ f(x) & \text{sinon} \end{cases}$

  En utilisant la fonction signe : $F(x) = f(x) + \frac{1}{2}\text{sgn}[c(x)+1]g(x)$

- La condition sur le signe de c(x) crée des discontinuités.
  Selon l'algorithme utilisé, il peut être préférable de garder une fonction continue.

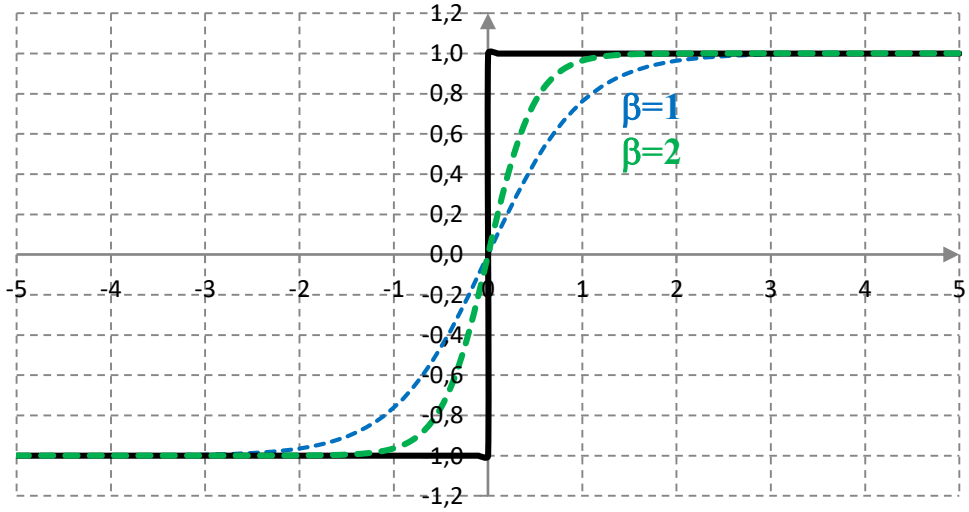
- On peut approcher la fonction signe par une fonction continue. (fonction sigmoïde).

$$\text{sgn}(y) \approx \frac{2}{1+e^{-\alpha y}} - 1$$

ou

$$\text{sgn}(y) \approx \tanh(\beta y) = \frac{e^{\beta y} - e^{-\beta y}}{e^{\beta y} + e^{-\beta y}}$$

avec α , β positifs assez grands



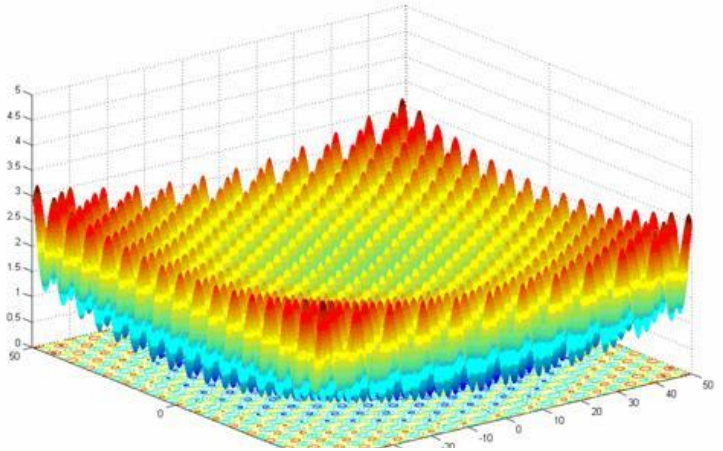
4.3.9 Fonctions tests

Exemples

- Fonction de Griewank

$$f(\mathbf{x}) = \sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

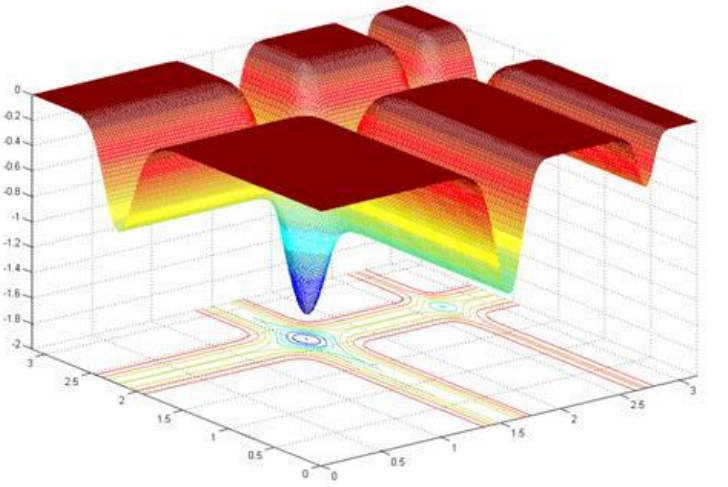
Minimum global unique
x = 0



- Fonction de Michalewicz

$$f(\mathbf{x}) = -\sum_{i=1}^{d} \sin(x_i)\sin^{2m}\left(\frac{ix_i^2}{\pi}\right)$$

Minimum global unique
x = (2.20 , 1.57) en dimension d=2



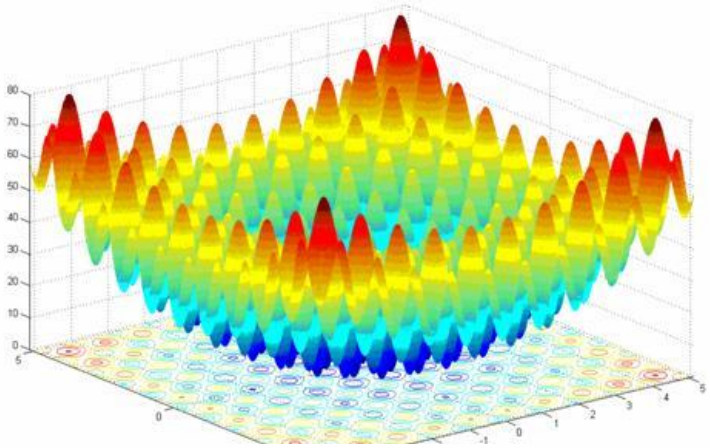
4.3.9 Fonctions tests

Exemples

- Fonction de Rastrigin

$$f(\mathbf{x}) = 10d + \sum_{i=1}^{d} \left[x_i^2 - 10\cos(2\pi x_i)\right]$$

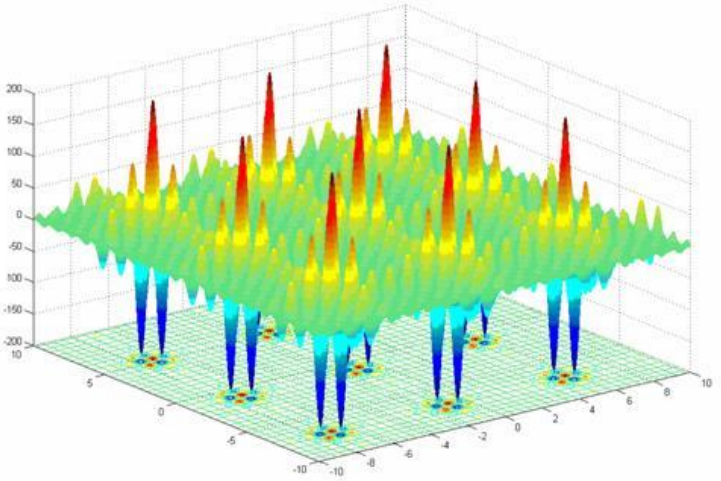
Minimum global unique
x = 0



- Fonction de Schubert

$$f(\mathbf{x}) = \left(\sum_{i=1}^{5} i\cos((i+1)x_1 + i)\right)\left(\sum_{i=1}^{5} i\cos((i+1)x_2 + i)\right)$$

18 minima globaux
f(x*) = –186.7309



Résumé : optimisation discrète

Méthodes exactes

• Algorithmes spécifiques :	chemin	(§4.1.2)
	flot	(§4.1.3)
	affectation	(§4.1.4)
• Programmation linéaire :	techniques de formulation	(§4.2.1)
	séparation et évaluation	(§4.2.3)

Méthodes approchées

• Métaheuristiques :	recuit simulé	(§4.3.2)
	essaims	(§4.3.4)
	algorithmes évolutionnaires	(§4.3.6)
• Applications :	minima locaux , minimum global	
	front de Pareto (optimisation multi-objectifs)	

Difficultés pratiques

- Combinatoire exponentielle
- Minima locaux
- Pas de conditions d'optimalité

Bibliographie : optimisation discrète

Livres en français

- **Programmation mathématique** (M. Minoux)
- **Métaheuristiques pour l'optimisation difficile** (J. Dréo, A. Pétrowski, P. Siarry, E. Taillard)
- **Optimisation discrète** (A. Billionnet)
- Algorithmes de graphes (P. Lacomme, C. Prins, M. Sevaux)
- Programmation linéaire (C. Guéret, C. Prins, M. Sevaux)
- Précis de recherche opérationnelle (R. Faure, B. Lemaire, C. Picouleau)

Livres en anglais

- **Stochastic optimization** (J.J. Schneider, S. Kirkpatrick)
- Integer and combinatorial optimization (L.A. Wolsey, G.L. Nemhauser)
- Heuristic search (S. Edelkamp, S. Schrödl)