Techniques de résolution

MTH6311

S. Le Digabel, École Polytechnique de Montréal

H2018

(v3)

Plan

1. Méthodes de trajectoire
2. Méthodes basées sur des populations
3. Conseils pratiques
4. Présentation de tests numériques

Rappels et définitions

- f est la fonction objectif.
- Ω est l'ensemble des points admissibles.
- $\mathcal{X}$ est l'ensemble des points pouvant être visités durant la recherche ($\Omega \subseteq \mathcal{X}$).
- La structure de voisinage N donne les règles de déplacement dans l'espace de recherche.
- Processus d'intensification : Mouvements dans $\mathcal{X}$ visant à baisser rapidement la valeur de f.
- Processus de diversification : Mouvements dans $\mathcal{X}$ visant à explorer l'espace. Permettent de s'échapper d'optima locaux, et d'explorer de nouvelles régions de $\mathcal{X}$.

Méthode de recherche locale, ou descente

Descente

[1] Choisir un point $x \in \Omega$

[2] Déterminer un point x'
qui minimise f dans $N(x) \cap \Omega$

[3] Si $f(x') < f(x)$
Poser $x \leftarrow x'$
Aller en [2]
Sinon STOP

Descente (suite)

- On peut définir `x := descente(x)` comme une fonction de Ω dans Ω.
- Dans l'étape [2], on peut stopper la recherche dès qu'on trouve $x' \in N(x) \cap \Omega$ tel que $f(x') < f(x)$ (stratégie opportuniste).
- Une descente est en général très rapide.

Descente : limitations

- Une descente s'arrête au premier minimum local rencontré.
- Or, un minimum local pour une structure de voisinage ne l'est pas forcément pour une autre.
- Le choix de N est donc crucial.
- Pour éviter d'être bloqué au premier minimum local rencontré, on peut décider d'accepter, sous certaines conditions, de se déplacer d'un point x vers un point $x' \in N(x) \cap \Omega$ tel que $f(x') \geq f(x)$.

Méthode du recuit simulé

- *Simulated annealing* (Kirkpatrick, Gelatt et Vecchi, 1983)
- Wikipedia : *méthode [...] inspirée d'un processus utilisé en métallurgie. On alterne dans cette dernière des cycles de refroidissement lent et de réchauffage (recuit) qui ont pour effet de minimiser l'énergie du matériau. [...] La méthode vient du constat que le refroidissement naturel de certains métaux ne permet pas aux atomes de se placer dans la configuration la plus solide. La configuration la plus stable est atteinte en contrôlant le refroidissement et en le ralentissant par un apport de chaleur externe.*

Recuit simulé : Ingrédients

- Paramètre de température T, avec une méthode pour le mettre à jour.
- Nombre aléatoire r dans $[0; 1]$.
- Fonction $p(T, x, x')$ qui dépend de T et de deux points de Ω, et qui est généralement choisie comme $e^{\frac{f(x)-f(x')}{T}}$:
  - Si $f(x') < f(x)$ ou si T est grand, alors presque toujours $r < p(T, x, x')$ (et on bougera de x à x').
  - Si $f(x') > f(x)$ et T est petit, alors presque toujours $r > p(T, x, x')$.

Recuit simulé : Algorithme

Recuit simulé

[1] Choisir un point $x \in \Omega$
Choisir une température initiale T_1
Poser $k \leftarrow 1$

[2] Tant qu'aucun critère d'arrêt n'est satisfait
  Choisir aléatoirement $x' \in N(x) \cap \Omega$
  Générer un nombre réel aléatoire r dans $[0; 1]$
  Si $r < p(T_k, x, x')$
    Poser $x \leftarrow x'$
  Poser $T_{k+1} \leftarrow g(T_k)$
  Poser $k \leftarrow k + 1$

Recuit simulé : mise à jour de T

- Plus T est petite et moins on a de chance d'accepter un mouvement qui dégrade f.
- Au début, on choisit T grand pour une plus grande liberté d'exploration.
- On fait ensuite tendre T vers 0 ce qui empêche de détériorer une solution.
- Décroître la température est donc une stratégie d'instensification.
- Convergence : Aarts, Korst, et Laarhoven (1997) : Si on définit T_k tel que $\sum_{k=1}^{\infty} e^{\frac{L}{T_k}} = \infty$ avec $L \in \mathbb{R}$, alors $\lim_{k \to \infty} P[\text{optimum trouvé après } k \text{ itérations}] = 1$.

Recuit simulé : mise à jour de T (suite)

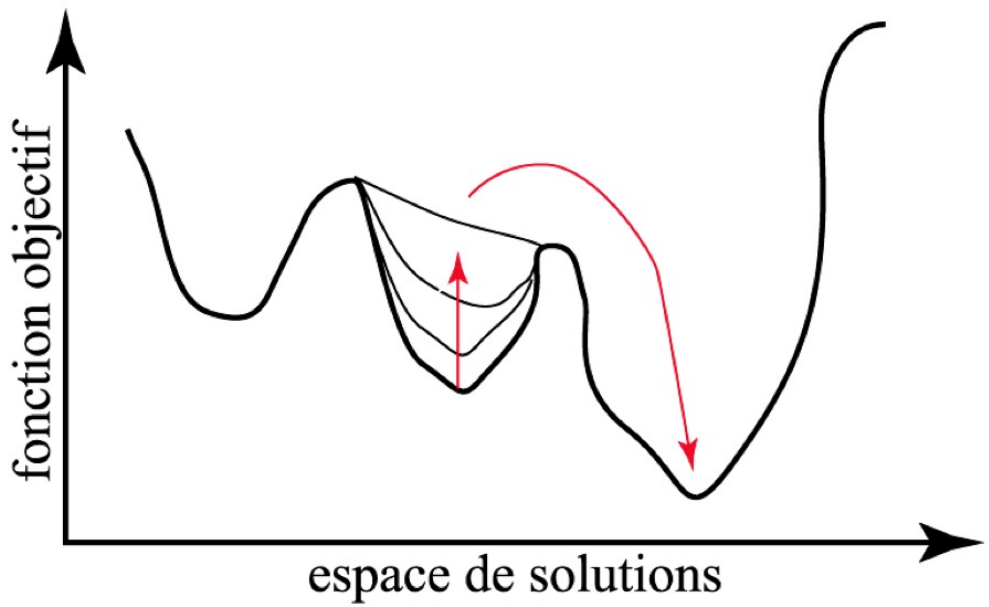
- Fonctionne avec $g(T) = \frac{L}{\log(T+c)}$ avec $c \in \mathbb{R}$, mais implique une convergence pratique trop lente.
- On sacrifie donc la convergence théorique pour une convergence pratique plus rapide vers des optima locaux.
- Choix populaire : $g(T) = \alpha T$ avec $0 < \alpha < 1$ (typiquement $\alpha = 0.95$).
- On peut aussi décroître T par paliers.
- On peut aussi utiliser une fonction non-monotone : ré-hausser la température est alors une stratégie de diversification.

Recuit simulé : améliorations

- Méthode sans mémoire, mais on peut ajouter une mémoire à long terme qui stocke le meilleur point rencontré.
- Comme critère d'arrêt, on peut choisir une limite sur le temps, ou une limite sur le nombre d'itérations sans modification du meilleur point courant. Lorsqu'on stocke le meilleur point rencontré, on peut décider de stopper la recherche dès qu'un certain nombre d'itérations a été effectué sans amélioration de ce point.
- L'un des principaux défauts de la méthode est le choix aléatoire dans $N(x)$. On peut donc être proche de l'optimum et passer juste à côté sans le voir. On peut corriger ceci en ajoutant une descente.

Recherche locale guidée (guided local search)

- C. Voudouris, 1997.
- Consiste à utiliser une technique de recherche locale dans laquelle la fonction objectif varie durant le processus de recherche, le but étant de rendre les minima locaux déjà visités moins attractifs.



Recherche locale guidée (suite)

- On note $\{A_1, A_2, \ldots, A_m\}$ un ensemble de m attributs utilisés pour discriminer les solutions de Ω. Par exemple :
  - Pour la coloration des sommets d'un graphe, on peut associer un attribut à chaque arête du graphe et on dira qu'une coloration a l'attribut A_e si et seulement l'arête e est en conflit dans la solution.
  - Pour le problème du voyageur de commerce, on peut par exemple associer un attribut à chaque arête du graphe et dire qu'une tournée possède l'attribut A_e si l'arête e fait partie de la tournée.

Recherche locale guidée (suite)

- Soit w_i le poids de l'attribut A_i et soit $\delta_i(x)$ une fonction qui vaut 1 si x possède l'attribut A_i et 0 sinon ($i \in \{1, 2, \ldots, m\}$). On modifie la fonction objectif comme ceci : $f'(x) = f(x) + \lambda \sum_{i=1}^{m} w_i \delta_i(x)$, avec $\lambda \in \mathbb{R}$.
- On va modifier les poids w_i au cours de l'algorithme. En règle générale, lorsqu'on se déplace d'un point x vers un point voisin x', on augmente le poids des attributs de x'.

Recherche locale guidée : Algorithme

GLS

[1] Choisir un point $x \in \Omega$
Poser $x^* \leftarrow x$
Initialiser les poids w_i à 0 ($f' = f$)

[2] Tant qu'aucun critère d'arrêt n'est satisfait
  $x' \leftarrow$ descente(x) sur f'
  Mettre à jour les poids w_i
  Poser $x \leftarrow x'$
  Si $f(x) < f(x^*)$
    Poser $x^* \leftarrow x$

Recherche tabou (Tabu Search)

- F. Glover, 1986.
- À chaque itération, on choisit le meilleur point $x' \in N(x)$, même si $f(x') > f(x)$.
- Lorsqu'on atteint un minimum local du voisinage $N(x)$, l'algorithme va donc se déplacer vers un point x' pire que x.
- Pour éviter de revenir à x si $x \in N(x')$, on crée une liste taboue T qui mémorise le derniers points visités et interdit tout déplacement vers un point de cette liste.

Recherche tabou : Algorithme

Tabou

[1] Choisir un point $x \in \Omega$
Poser $T \leftarrow \emptyset$
Poser $x^* \leftarrow x$

[2] Tant qu'aucun critère d'arrêt n'est satisfait
Déterminer x' qui minimise f dans $N_T(x)$
Si $f(x') < f(x^*)$
poser $x^* \leftarrow x'$
Poser $x \leftarrow x'$
Mettre à jour T

Recherche tabou (suite)

- Les points ne demeurent dans T que pour un nombre limité d'itérations, car on limite sa taille à $|T|_{max}$, ce qui en fait donc une mémoire à court terme.

- Si un point est dans T on dit que c'est un point tabou. De même, tout mouvement qui nous mène du point courant à un point de T est appelé mouvement tabou.

- Gérer la liste taboue T peut s'avérer difficile en pratique, surtout au niveau de la mémoire. Au lieu de mémoriser des points, on peut mémoriser des attributs, des mouvements, ou des modifications.

Recherche tabou (suite)

- Cette gestion de T, efficace en mémoire, induit cependant quelques inconvénients :
  - Points multiples obtenus à partir de modifications différentes :

$$abcd \to bacd \to bcad \to dcab \to acdb \to abdc \to abcd \ .$$

    On n'a pas effectué deux fois la même permutation de deux lettres (ab, ac, bd, ad, bc, et cd), mais on retombe toutefois sur le même point $abcd$.

  - Points inaccessibles :

$$abcd \to bacd \to dacb \to dcab \ .$$

    $T = \{$ab,bd,ac$\}$, et si $|T|_{max} \geq 3$, alors on interdit le point $dcba$ qui pourrait être intéressant.

Recherche tabou (suite)

- Pour éviter ce dernier comportement, on peut lever le statut tabou d'un point si celui-ci satisfait certains critères d'aspiration. Par exemple, le statut tabou d'un point est levé si celui-ci est meilleur que le meilleur point x^* rencontré jusqu'ici.
- On définit l'ensemble $N_T(x)$ comme l'ensemble des points de $N(x)$ vers lesquels la liste taboue permet de se diriger, en considérant le critère d'aspiration. Par exemple :

$$N_T(x) = \left\{x' \in N(x) \text{ tel que } x' \notin T \text{ ou } f(x') < f(x^*)\right\}.$$

Recherche tabou (suite)

- Comme critère d'arrêt on peut par exemple fixer un nombre maximum d'itérations sans amélioration de x^*, ou on peut fixer un temps limite après lequel la recherche doit s'arrêter.
- Certains préconisent l'utilisation d'une liste taboue de longueur variable (Taillard, 1991).
- D'autres préfèrent les listes taboues réactives (Battiti et Tecchiolli, 1994) dont la taille varie selon les résultats de la recherche. Ainsi, si des points sont visités trop souvent, on peut augmenter la longueur de la liste, ce qui aura pour effet de diversifier la recherche. Par contre, si le point courant n'est que rarement amélioré, cela peut signifier qu'il est temps d'intensifier la recherche en évitant d'interdire trop de points dans le voisinage du point courant, et en diminuant donc la longueur de la liste taboue.

Recherche tabou (suite)

La liste taboue est une mémoire à court terme. On peut rajouter une mémoire à plus long terme, de quatre façons :

1. Les mémoires basées sur la récence mémorisent pour chaque point l'itération la plus récente qui l'impliquait. On peut par exemple favoriser les mouvements vers des points contenant des attributs peu récents, afin d'éviter de rester bloqué dans une même région de l'espace de recherche.

2. Les mémoires basées sur la fréquence mémorisent le nombre de fois que les points ont été visités. Cette information permet d'identifier les régions de l'espace de recherche qui ont été le plus visitées. On peut exploiter cette information pour diversifier la recherche.

Recherche tabou (suite)

3 Les mémoires basées sur la qualité mémorisent les meilleurs points rencontrés afin d'identifier les composantes communes de ces points. On peut ensuite faire usage de cette information pour créer de nouveaux points qui contiennent autant que possible ces composantes prometteuses.

4 Les mémoires basées sur l'influence mémorisent les choix qui se sont avérés les plus judicieux ou les plus critiques durant le processus de recherche. Ceci permet d'essayer de répéter les bons choix et d'éviter de répéter les mêmes erreurs.

GRASP

- *Greedy Randomized Adaptive Search Procedure*, Feo et Resende, 1995.
- Suppose qu'un point est constitué d'un ensemble de composantes.
- Méthode itérative en deux phases :
  - Phase constructive : Génère un point en ajoutant de nouvelles composantes en plusieurs étapes. Chaque composante candidate est évaluée à l'aide d'un critère heuristique qui permet de mesurer le bénéfice qu'on peut espérer en rajoutant cette composante au point partiel courant. La liste de candidats, notée `RCL` (*Restricted Candidate List*) contient les R meilleures composantes selon ce critère.
  - Phase d'amélioration : Appliquer une autre méthode (descente ou autre).

GRASP : Algorithme

GRASP

- [1] Poser $f^* \leftarrow +\infty$ et $x^* \leftarrow \emptyset$
- [2] Tant qu'aucun critère d'arrêt n'est satisfait
  - Poser $x \leftarrow \emptyset$
  - Tant que le point courant x n'est pas complet
    - Évaluer l'ensemble des composantes pouvant être rajoutées à x
    - Déterminer la liste RCL des R meilleures composantes
    - Choisir aléatoirement une composante dans RCL et l'ajouter à x
  - $x' \leftarrow$ descente(x)
  - Si $f(x') < f^*$
    - Poser $x^* \leftarrow x'$ et $f^* \leftarrow f(x')$

GRASP (suite)

- La longueur R de la liste RCL joue un rôle important. Si on pose $R = 1$, la phase constructive est un algorithme glouton qui choisit à chaque étape la meilleure composante. Si on fixe R égal au nombre de composantes pouvant être rajoutées, on a alors un algorithme constructif purement aléatoire.
- R peut être fixe ou variable (adaptative).
- GRASP est assez simple à mettre en place. Mais pour qu'elle soit efficace, il est important d'utiliser une méthode constructive capable de générer des points dans des régions différentes de l'espace de recherche.
- Il se pose aussi le problème de la réalisabilité des points. Définir une procédure de réparation.

Voisinages variables

- On considère un ensemble de voisinages $\{N^1, N^2, \ldots, N^{k_{max}}\}$ paramétrés par un indice k.
- Plus k est grand, et plus on a des voisinages "larges" et éloignés. Par exemple, k peut correspondre à un nombre de composantes différentes.
- L'idée derrière les méthodes à voisinages variables est de considérer successivement les voisinages, des plus proches au plus lointains.

Voisinages variables (suite)

- On change de voisinage ($k \leftarrow k+1$) à chaque fois que l'on est "bloqué" dans l'un d'entre eux.
- En effet, un minimum local par rapport à un voisinage ne l'est plus forcément par rapport à un autre.
- En cas de succès (baisse de f), on revient à N^1.
- En général, on permet de cycler plusieurs fois de $k = 1$ à $k = k_{max}$.

Descente à voisinages variables (VND)

VND

[1] Choisir un point $x \in \Omega$
Poser $k \leftarrow 1$

[2] Tant qu'aucun critère d'arrêt n'est satisfait
  Déterminer x' qui minimise f dans $N^k(x) \cap \Omega$
  Si $f(x') < f(x)$
    Poser $x \leftarrow x'$
    Poser $k \leftarrow 0$
  Si $k < k_{max}$
    Poser $k \leftarrow k + 1$
  Sinon
    Poser $k \leftarrow 1$

Recherche à voisinages variables (VNS)

- VNS : **V**ariable **N**eighborhood **S**earch.
- N. Mladenović et P. Hansen, 1997.
- Nécessite une méthode de recherche locale (`descente`) et une méthode de perturbation (`shaking`) permettant de s'éloigner des optima locaux.
- La méthode de perturbation est paramétrée par k (amplitude de perturbation) et change de plus en plus la solution courante à mesure que k grandit.
- Chaque valeur de k correspond à un voisinage différent N^k.
- La recherche est ainsi de plus en plus globale et diversifiée lorsqu'aucun progrès n'est effectué.

Recherche à voisinages variables (VNS)

VNS

[1] Choisir un point $x \in \Omega$
Poser $k \leftarrow 1$

[2] Tant qu'aucun critère d'arrêt n'est satisfait
- $x' \leftarrow$ shaking(x, k) *choisir aléatoirement un point x' dans $N^k(x)$*
- $x'' \leftarrow$ descente(x')
- Si $f(x'') < f(x)$
  - Poser $x \leftarrow x''$
  - Poser $k \leftarrow 0$
- Si $k < k_{max}$
  - Poser $k \leftarrow k + 1$
- Sinon
  - Poser $k \leftarrow 1$

Illustration de VNS

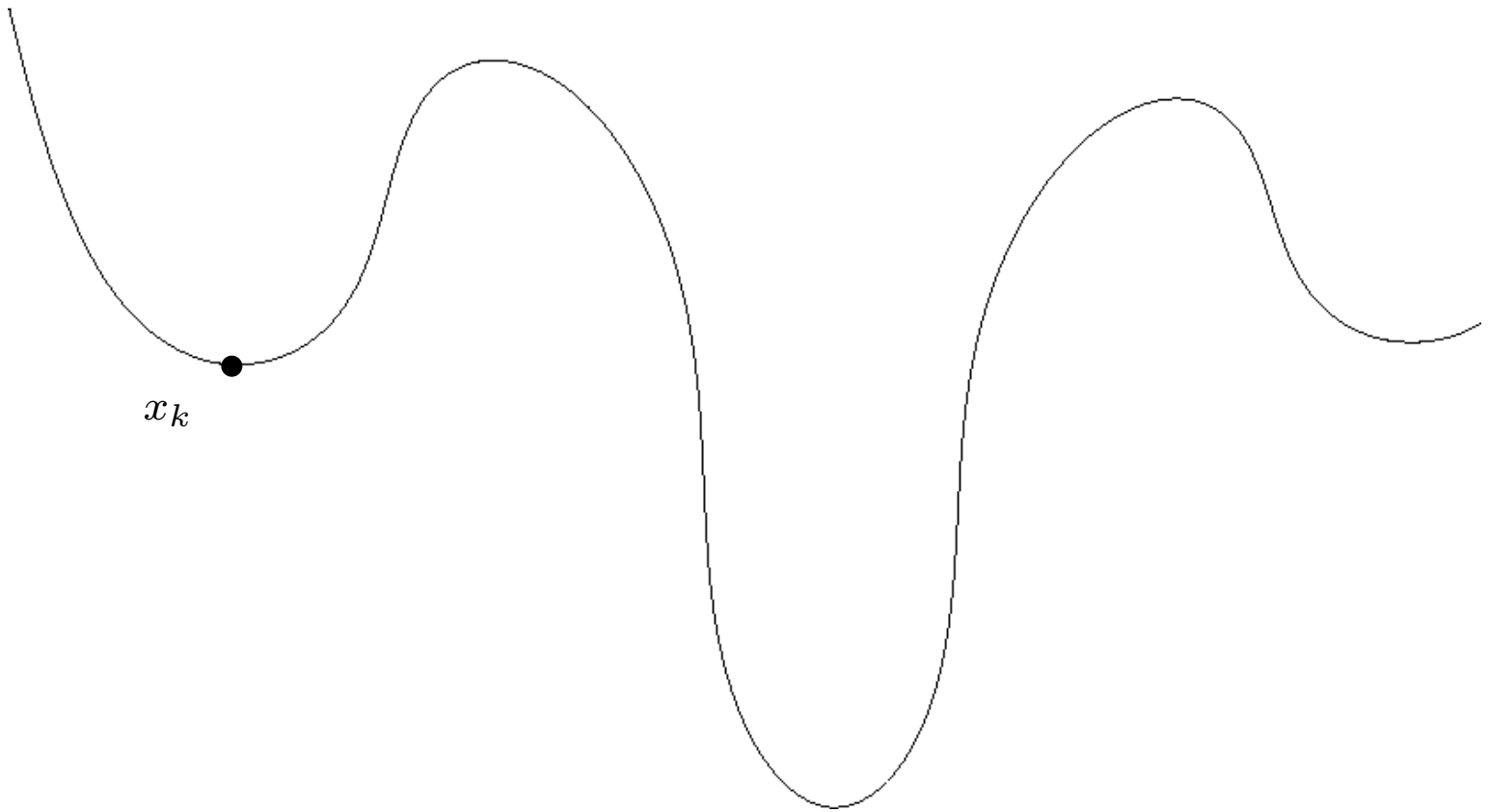


Illustration de VNS

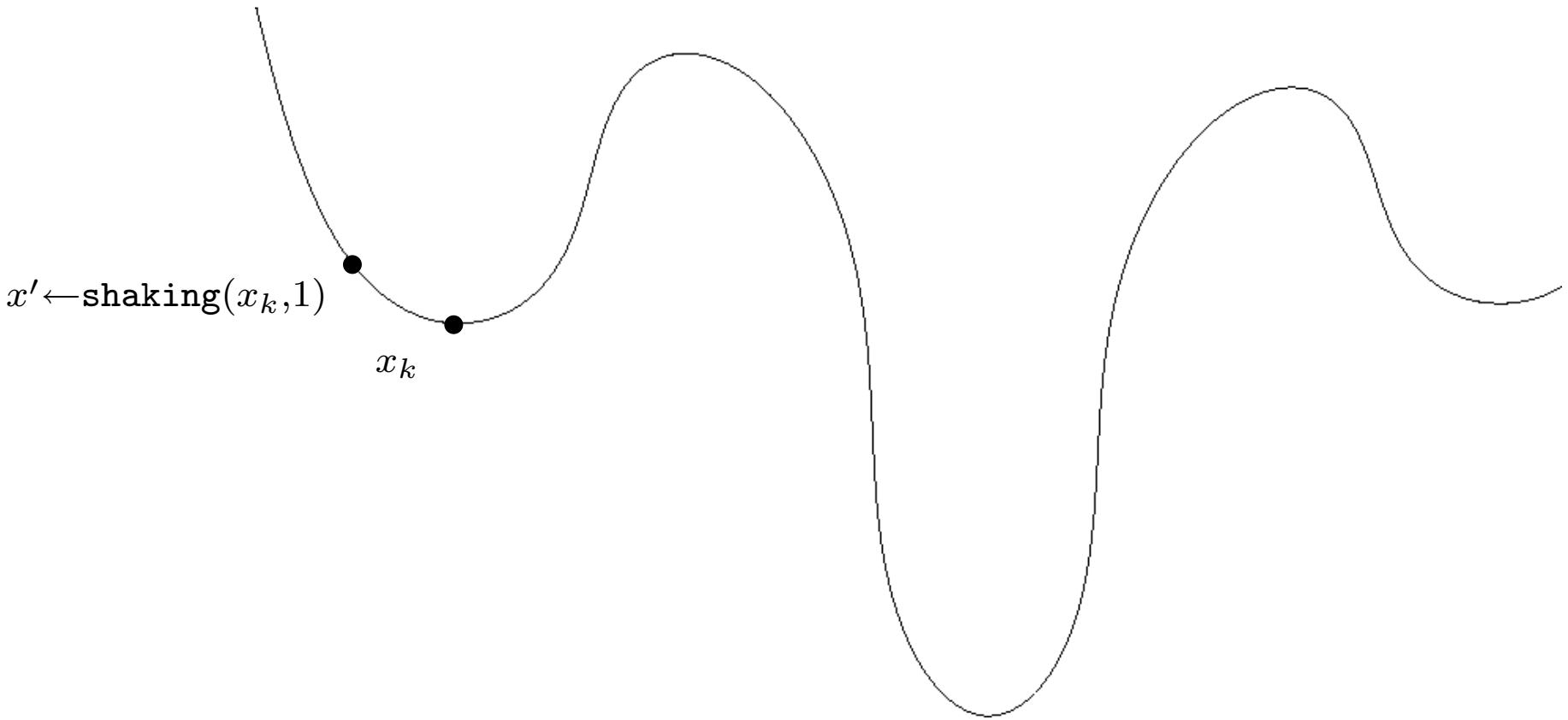


Illustration de VNS

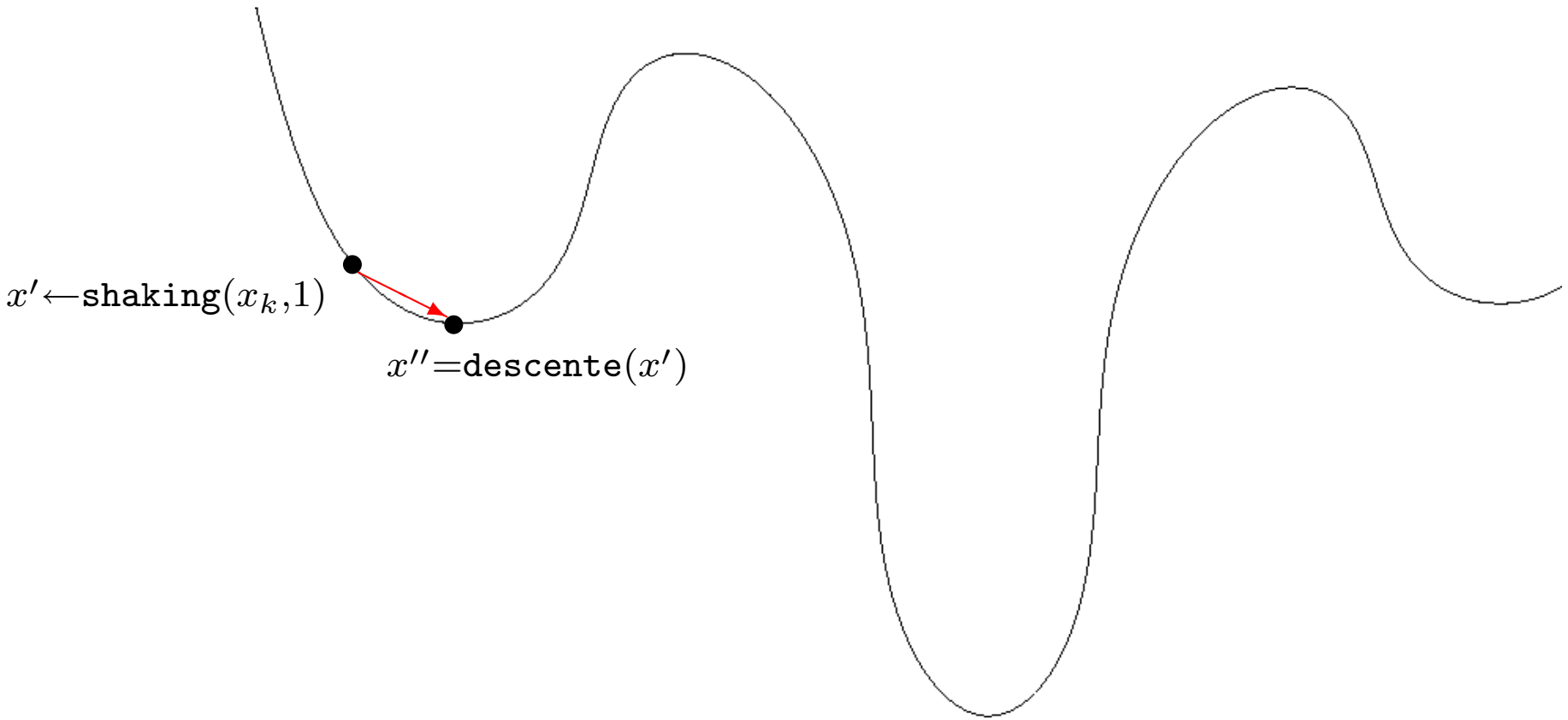


Illustration de VNS

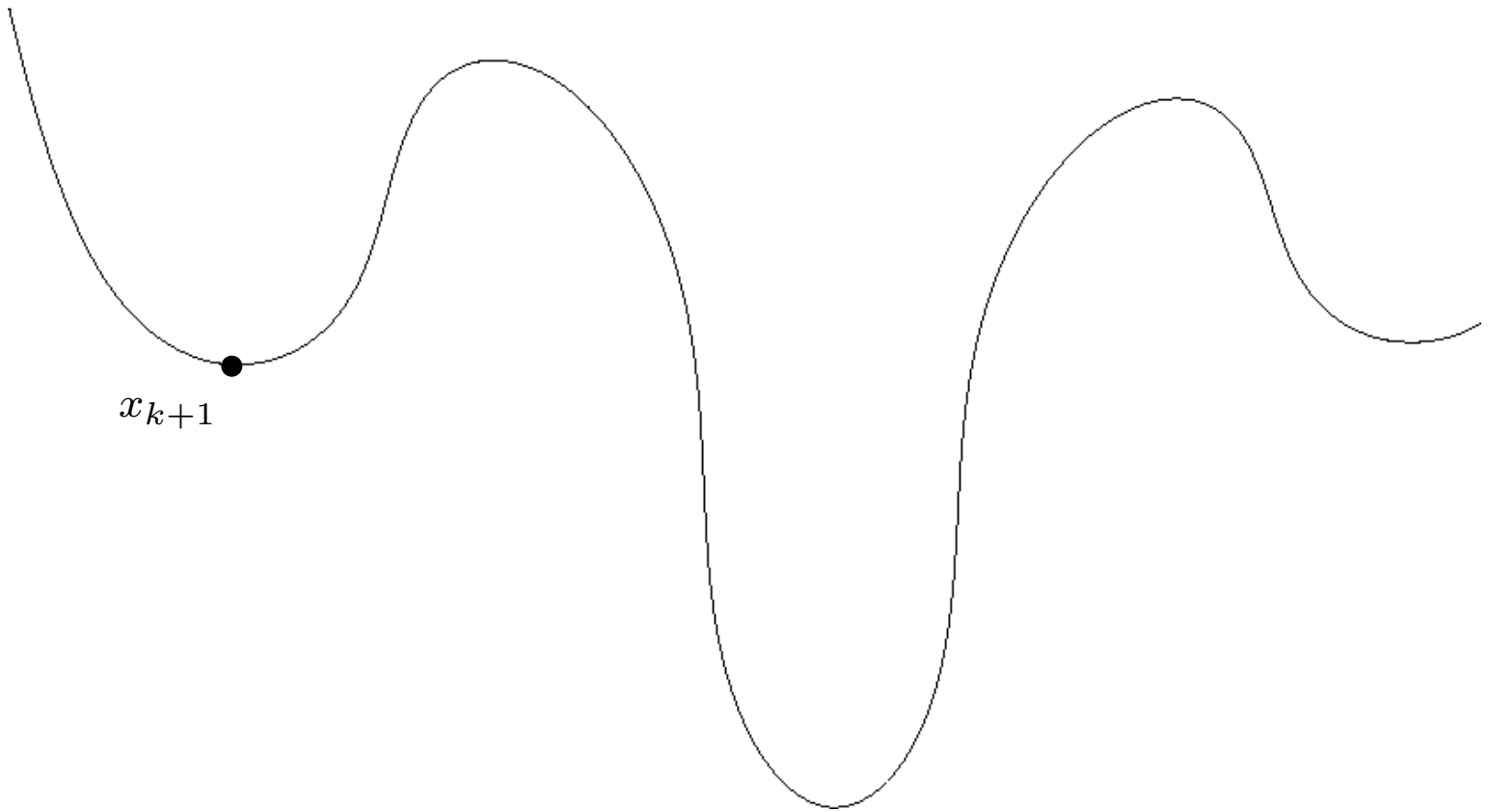


Illustration de VNS

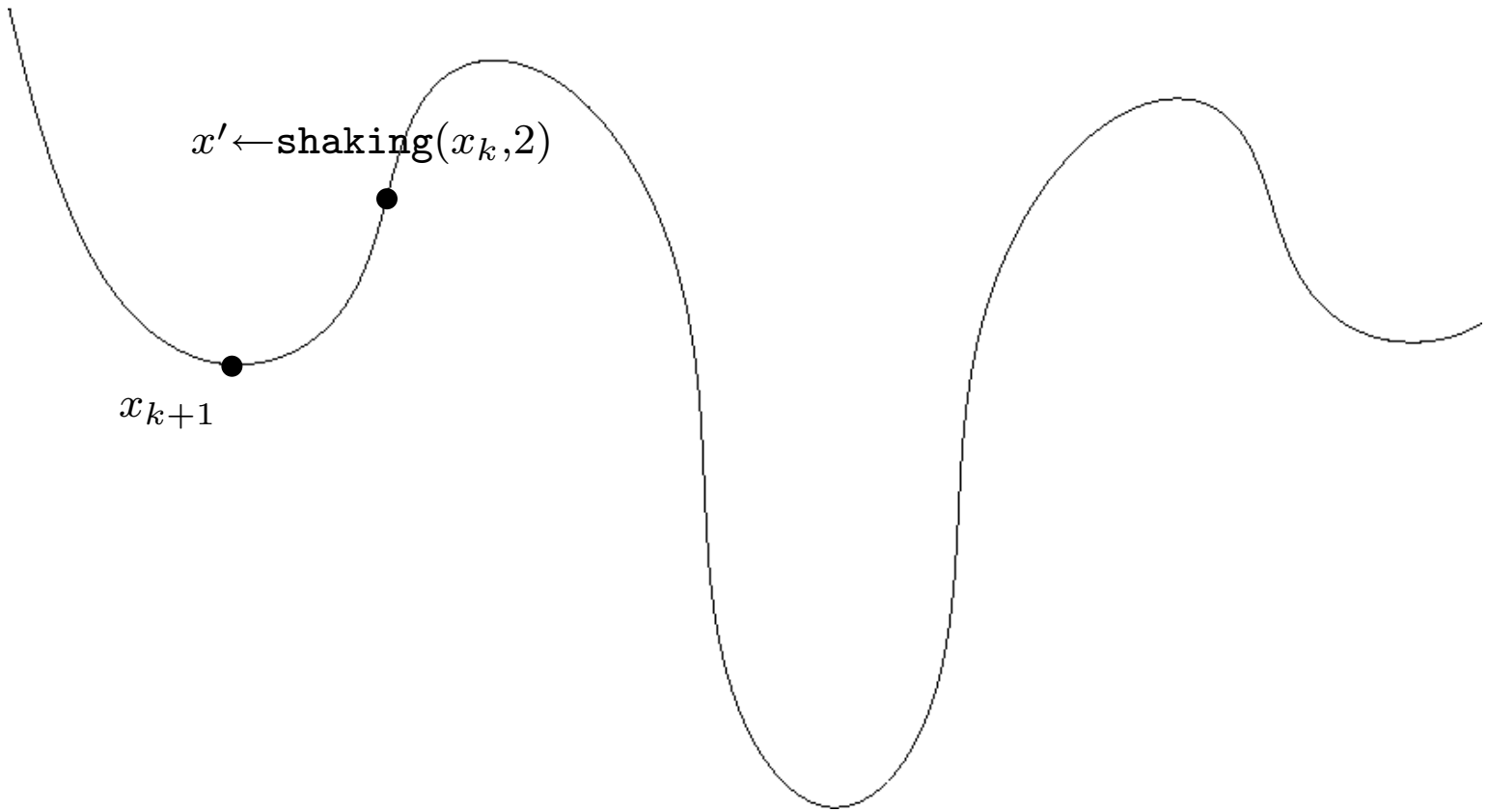


Illustration de VNS

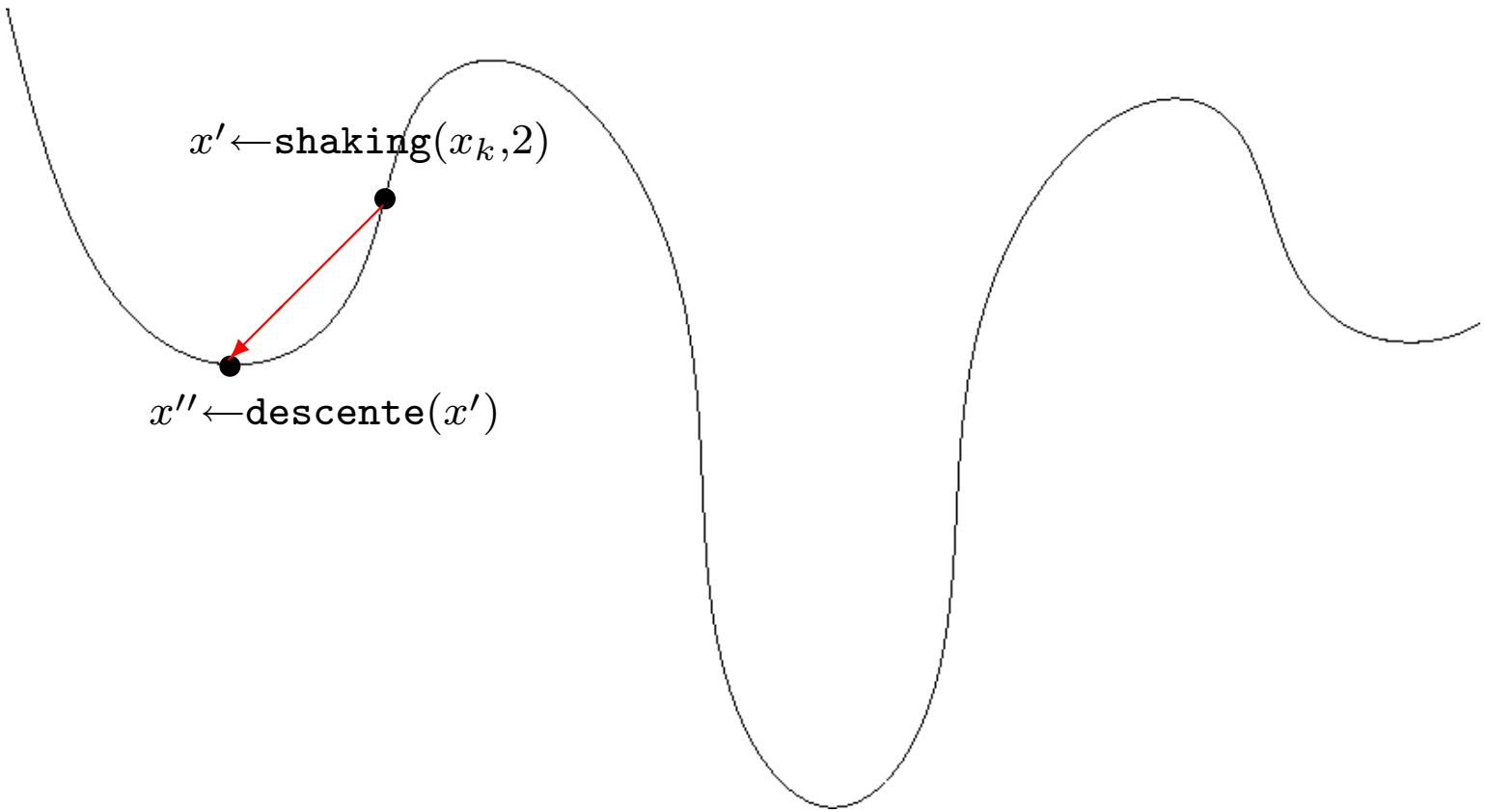


Illustration de VNS

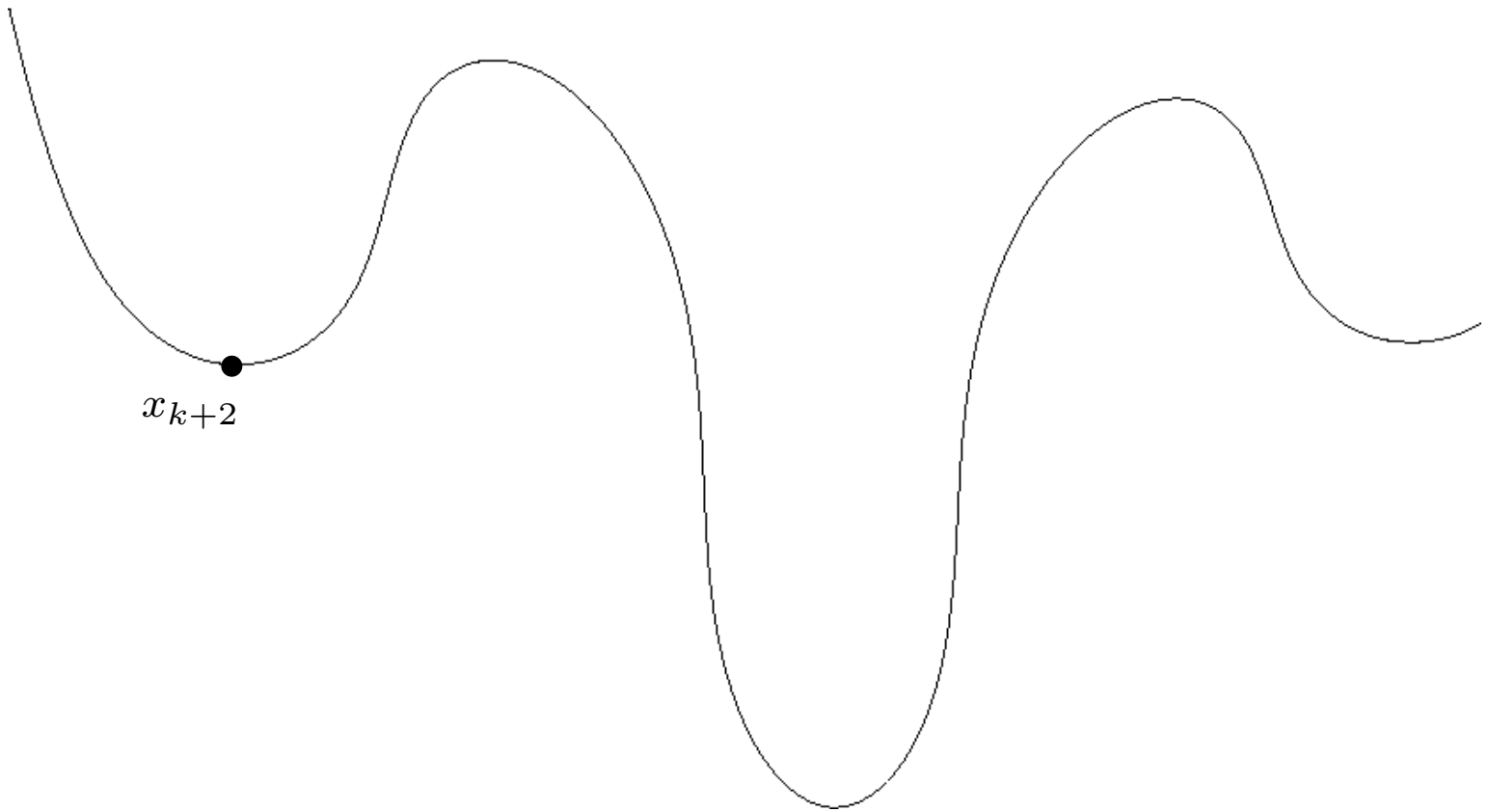


Illustration de VNS

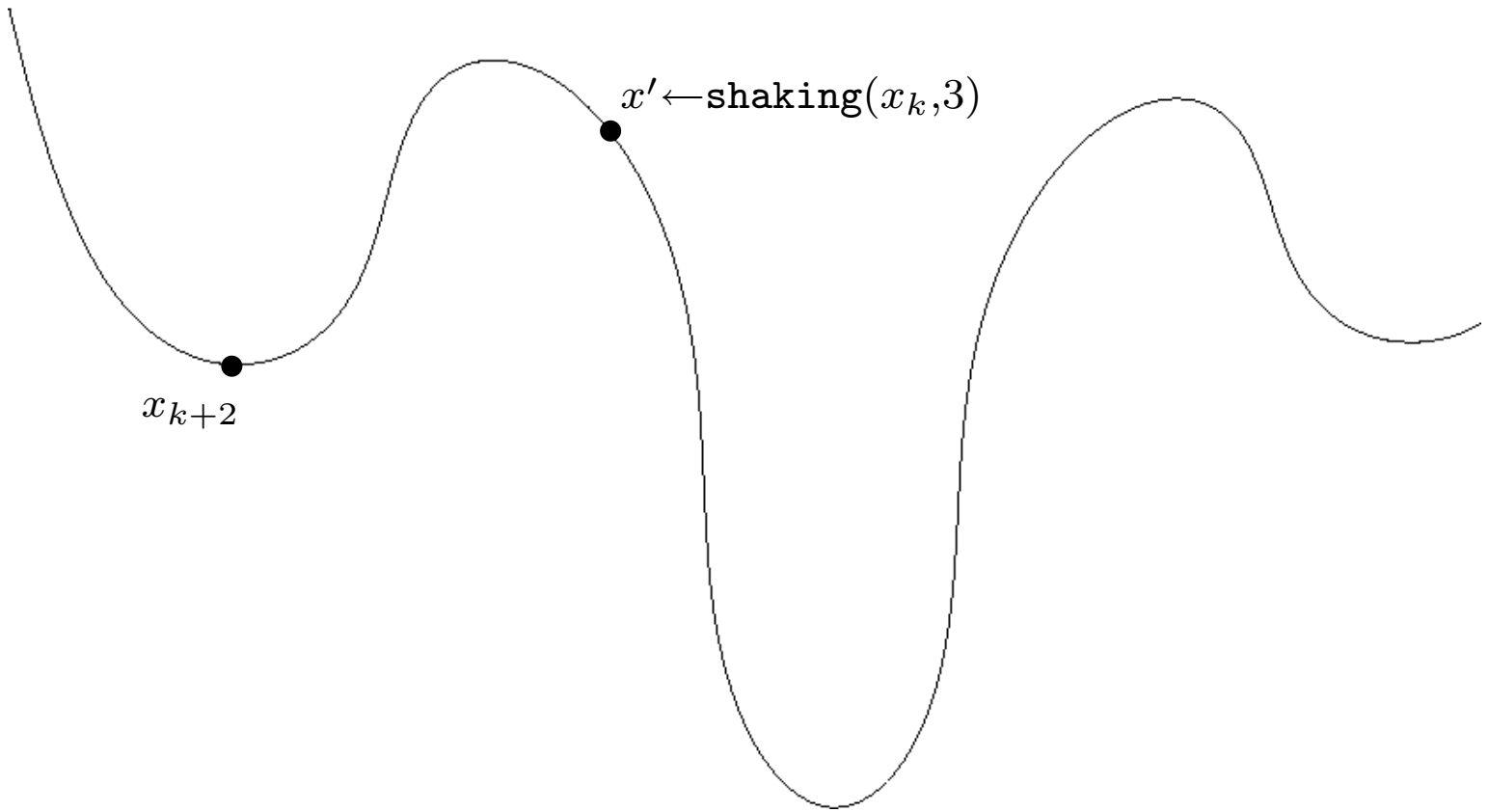


Illustration de VNS

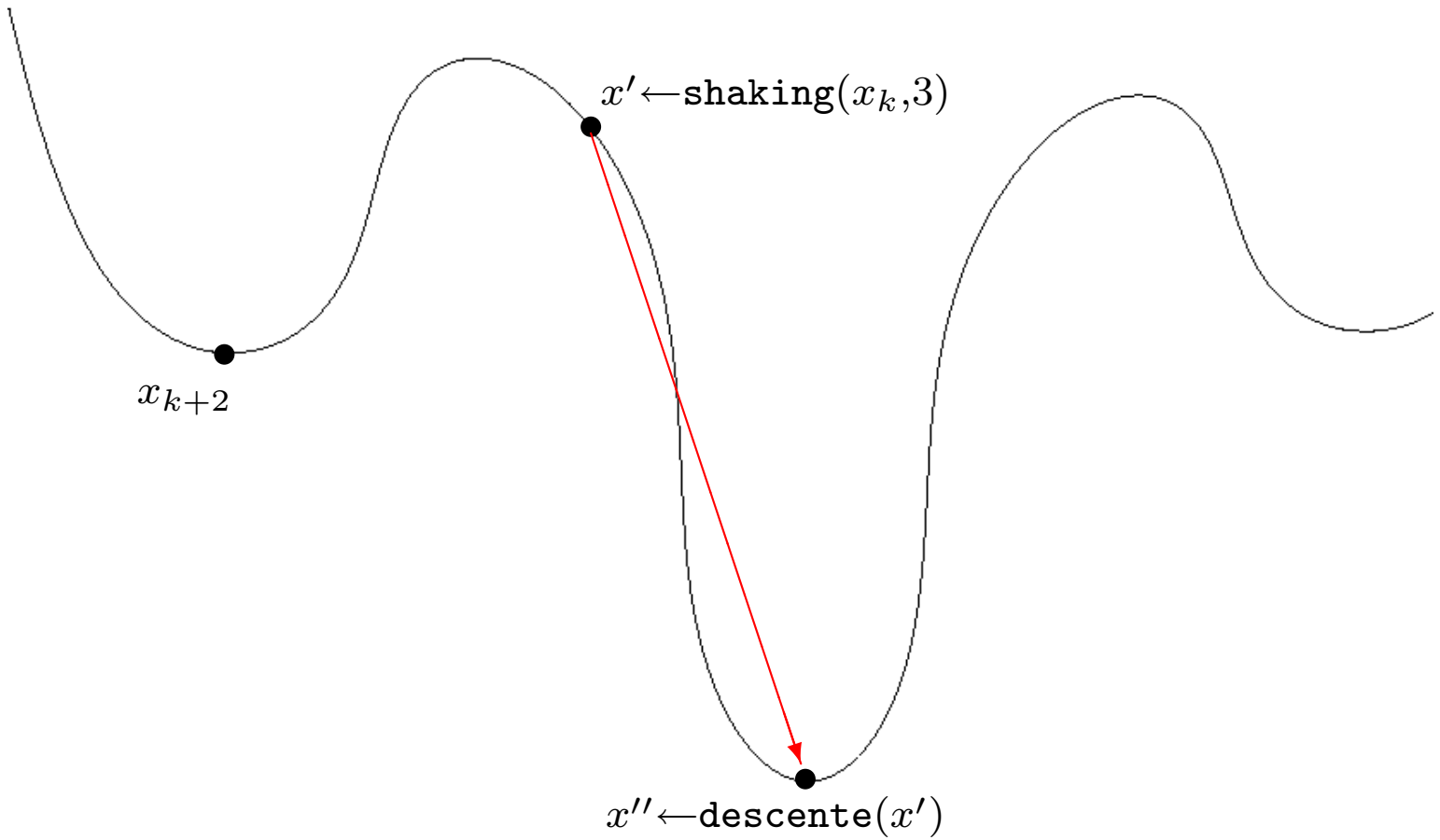
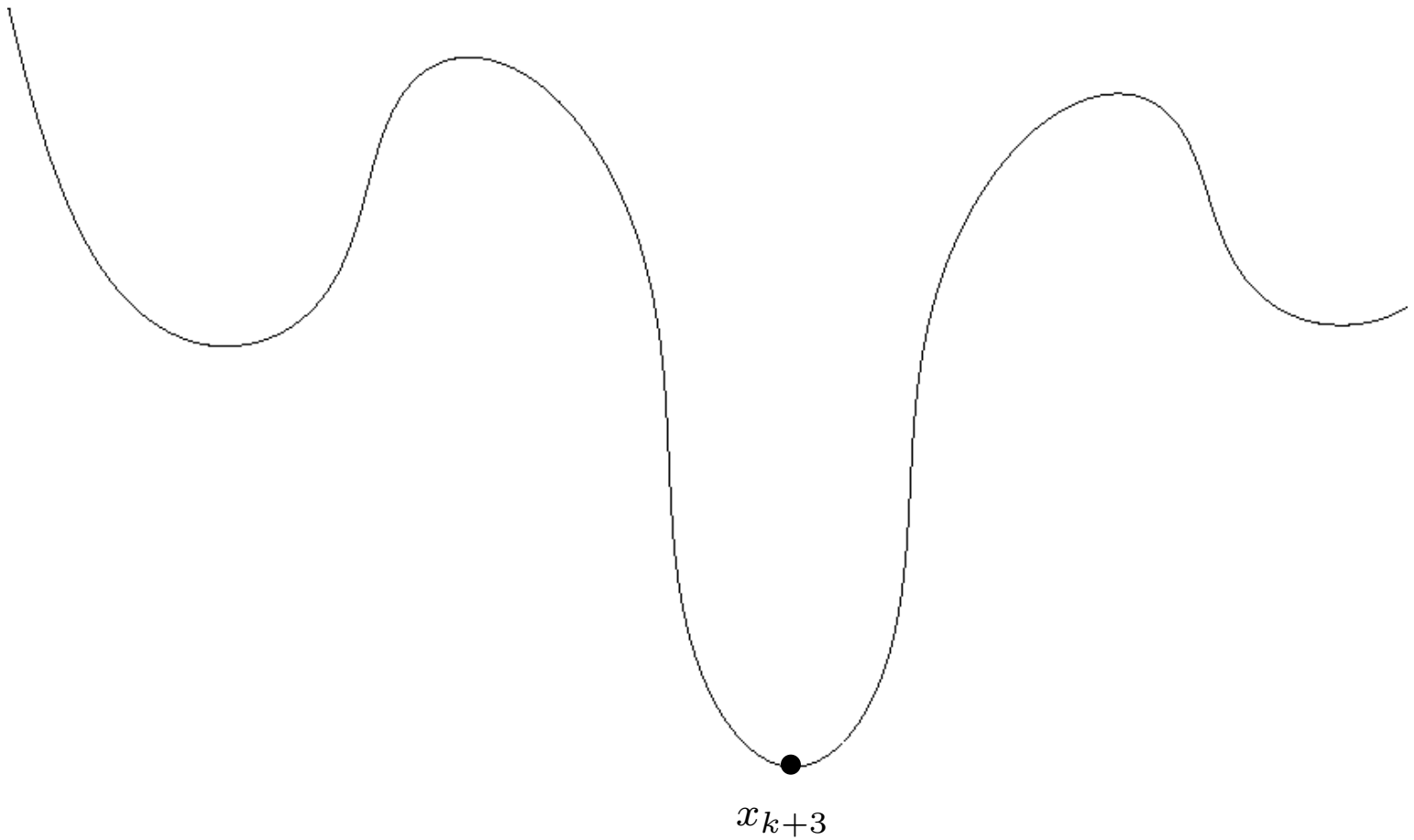


Illustration de VNS



Formulation Search Space (FSS)

- *Reformulation Descent (RD)* : Mladenović, Plastria, et Urosević (2005).
- *Formulation Search Space (FSS)* : mêmes auteurs, 2007.
- Équivalents de la VND et de la VNS mais en enrichissant les voisinages avec des façons différentes et mesurables de changer la formulation du problème.
- Liens exacts entre FSS et VNS ?
- **Exemples :** *Circle packing* : mixer les coordonnées cartésiennes et polaires. *Pooling* : mixer représentation par flots et par proportions. Autres exemples ?

Méthodes basées sur des populations

- Parfois appelées méthodes évolutives.
- Ces méthodes font évoluer une population d'individus ($\simeq$ des points) selon des règles bien précises. Ces méthodes alternent entre des périodes d'adaptation individuelle et des périodes de coopération durant lesquelles les individus peuvent échanger de l'information.

Méthode évolutive générique

```
Méthode évolutive
    |
    | [1] Générer une population initiale d'individus
    |
    | [2] Tant qu'aucun critère d'arrêt n'est satisfait
    |     | Exécuter une procédure de coopération
    |     | Exécuter une procédure d'adaptation individuelle
```

Types d'individus

Les individus qui évoluent dans une méthode basée sur les populations ne sont pas nécessairement des points. Il peut aussi s'agir de morceaux de points ou tout simplement d'objets que l'on peut facilement transformer en points. Considérons par exemple un problème de tournées de véhicules. Un individu peut être la tournée d'un véhicule qui visite un sous-ensemble de clients alors qu'un point est un ensemble de tournées (et donc d'individus) qui visitent tous les clients. Pour la coloration de graphes, on peut considérer toute permutation des sommets comme un individu. En appliquant l'algorithme séquentiel de coloration, on peut transformer une permutation en une coloration et donc un individu en point.

Type d'évolution

La population d'une itération est constituée d'anciens individus (qui auront survécu) et de nouveaux individus. La méthode évolutive doit indiquer comment décider de la survie des individus et comment choisir les nouveaux individus qui vont entrer dans la population. Lorsqu'on change totalement de population d'une itération à l'autre (c'est-à-dire que seuls les nouveaux individus sont conservés pour l'itération suivante), on parle de remplacement générationnel. Par contre, lorsque seulement une partie de la population peut varier d'une itération à la suivante, on parle de remplacement stationnaire (*steady state*). La plupart des méthodes évolutives utilisent des populations de taille fixe, et on décide généralement de garder les p meilleurs individus (parmi les anciens et les nouveaux). Des populations de taille variable (où on décide par exemple de manière aléatoire de la survie des individus) sont cependant également possibles.

Structure de voisinage

À chaque individu on associe un sous-ensemble d'autres individus avec lesquels il peut échanger de l'information. Si chaque individu peut communiquer avec tous les autres individus de la population, on parle de population non-structurée. Par contre, si chaque individu ne peut communiquer qu'avec un sous-ensemble d'individus, on parle de population structurée. Les structures les plus communes sont la grille et le cercle.

Sources d'information

Le nombre d'individus qui coopèrent pour créer un nouvel individu est souvent égal à 2. On parle alors de parents qui génèrent des enfants. On peut cependant également combiner plus de 2 points pour créer des enfants. Certaines méthodes évolutives utilisent par exemple l'information contenue dans toute la population pour créer un nouvel individu. D'autres méthodes utilisent même toutes les populations de toutes les itérations précédentes pour créer des enfants : on dit alors que la source d'information est l'historique de la recherche. Par historique on entend généralement toute information qui ne peut pas être obtenue en analysant les individus de la population courante ; la connaissance des compositions des populations précédentes est nécessaire pour accéder à cette information.

Irréalisabilité

Un individu est un objet défini avec des règles bien précises. En combinant des individus pour en créer des nouveaux, il peut arriver que le nouvel objet résultant de l'échange d'information n'est pas un individu admissible. Par exemple, définissons un individu comme une coloration sans conflit des sommets d'un graphe. Étant données deux colorations C_1 et C_2, on peut combiner celles-ci pour en créer une nouvelle en choisissant la couleur de chaque sommet aléatoirement dans C_1 ou C_2. Une telle coloration peut avoir des conflits et n'est donc pas nécessairement un individu.

Irréalisabilité (suite)

Dans une telle situation, on peut réagir d'au moins 4 façons :

- Rejeter l'enfant.
- *Réparer* l'enfant pour le rendre réalisable.
- Accepter l'enfant et rajouter une composante à la fonction objectif qui pénalise les violations de contraintes.
- Développer des procédures de combinaisons qui garantissent l'obtention d'enfants admissibles (sans violation).

Intensification

Idéalement, on devrait pouvoir localiser l'information pertinente qui rend un individu meilleur qu'un autre. Lorsque ceci est possible, il reste alors à développer une procédure de coopération qui crée des enfants en combinant adéquatement les informations pertinentes de chacun des parents. La phase d'adaptation individuelle n'est alors pas vraiment indispensable au bon fonctionnement de la méthode évolutive. C'est le cas par exemple pour les problèmes de tournées de véhicules où la bonne qualité d'une solution peut être expliquée par l'utilisation d'arcs de faible coût.

Intensification (suite)

Il existe cependant de nombreux problèmes pour lesquels une telle information pertinente est difficile à localiser. Par exemple, si Π_1 et Π_2 sont deux permutations des sommets d'un graphe et si C_1 et C_2 sont les deux colorations résultant de l'algorithme séquentiel de coloration, avec C_1 utilisant moins de couleurs que C_2, il est difficile de localiser dans Π_1 les raisons qui font que Π_1 est meilleure que Π_2. Il est donc également difficile de transmettre une information pertinente lors de la phase de coopération.
Il est alors important d'utiliser une bonne procédure d'adaptation individuelle. Il est courant de faire appel à une technique de recherche locale qui est appliquée à chacun des individus afin d'explorer les régions de l'espace de recherche qui leur sont proches (on parle alors d'un algorithme mimétique).

Diversification

Une difficulté majeure rencontrée lors de l'utilisation des méthodes évolutives est leur convergence prématurée. Certains utilisent des procédures de bruitage qui modifient légèrement les individus de manière aléatoire. Ce bruitage est appliqué indépendamment sur chaque individu. Il diffère de l'utilisation d'une recherche locale par le fait que son effet sur la qualité de la solution n'est pas prévisible. L'opérateur de bruitage le plus connu est la mutation dans les algorithmes génétiques. Au lieu de modifier les individus aléatoirement, certains préfèrent créer de nouveaux individus différents des individus déjà rencontrés en faisant usage d'une mémoire à long terme basée par exemple sur la récence et la fréquence.

Algorithmes génétiques (GAs)

- Holland, 1962 ; Rechenberg, 1965 ; Fogel et al, 1966.
- Inspirés de la théorie de l'évolution et des processus biologiques qui permettent à des organismes de s'adapter à leur environnement.

Algorithme génétique avec remplacement générationnel

GA

[1] Générer une population initiale P_0 de p individus

Poser $i \leftarrow 0$

[2] Tant qu'aucun critère d'arrêt n'est satisfait

Poser $i \leftarrow i + 1$

Poser $P_i \leftarrow \emptyset$

Répéter p fois

Créer e en croisant 2 individus de P_{i-1}

Muter e et l'ajouter à P_i

Algorithme génétique (suite)

- **Types d'individus :** Points, en général.
- **Type d'évolution :** Remplacement générationnel avec une population de taille constante.
- **Structure de voisinage :** Population possiblement structurée.
- **Sources d'information :** Deux parents.
- **Irréalisabilité :** L'opérateur de croisement évite la génération de points non admissibles.
- **Intensification :** Aucune.
- **Diversification :** Mutation qui est une procédure de bruitage.

Recherche dispersée (scatter search)

- F. Glover, 1977.
- Consiste à générer un ensemble D_i de points à partir d'un ensemble R_i de points de références. Ces points sont obtenus en effectuant des combinaisons linéaires des points de référence. Ces combinaisons peuvent avoir des coefficients négatifs, ce qui veut dire que les points résultant peuvent être à l'extérieur de l'enveloppe convexe des R_i. L'ensemble C_i des points résultants n'est pas forcément un ensemble de points admissibles. On applique donc une procédure de réparation sur chaque point de C_i pour obtenir un ensemble A_i de points admissibles. Les points de A_i sont finalement optimisés à l'aide d'une recherche locale pour obtenir l'ensemble D_i des points dispersés. Le nouvel ensemble R_{i+1} de points de référence est obtenu en sélectionnant des points dans $R_i \cup D_i$.

Recherche dispersée : Algorithme

Scatter Search

[1] Générer une population initiale R_0
Poser $i \leftarrow 0$

[2] Tant qu'aucun critère d'arrêt n'est satisfait
- $C_i \leftarrow$ combinaisons linéaires des points de R_i
- $A_i \leftarrow$ procédure de réparation(C_i) (admissibles)
- $D_i \leftarrow$ recherche locale (A_i)
- $R_{i+1} \leftarrow$ points dans $R_i \cup D_i$
- Poser $i \leftarrow i + 1$

Recherche dispersée (suite)

- **Types d'individus :** Points admissibles.
- **Type d'évolution :** Remplacement stationnaire avec une population de taille généralement constante.
- **Structure de voisinage :** Population non structurée.
- **Sources d'information :** Au moins deux parents.
- **Irréalisabilité :** Les points non admissibles sont réparés.
- **Intensification :** Recherche locale.
- **Diversification :** Combinaison non convexe de points de référence.

Colonies de fourmis (ACO)

- *Ant Colony Optimization*, M. Dorigo, 1992.
- Méthode fortement basée sur l'historique de la recherche.
- Méthode évolutive inspirée du comportement des fourmis à la recherche de nourriture. Lorsqu'une fourmi doit choisir entre deux directions, elle choisit avec une plus grande probabilité celle comportant une plus forte concentration de phéromone. C'est ce processus coopératif qu'ACO tente d'imiter.
- Chaque fourmi est un algorithme constructif capable de générer des points. Soit D l'ensemble des décisions possibles que peut prendre une fourmi pour compléter un point partiel. La décision $d \in D$ qu'elle choisira dépendra de deux facteurs, à savoir la force gloutonne et la trace.

Colonies de fourmis (suite)

- La force gloutonne est une valeur η_d qui représente l'intérêt qu'a la fourmi à prendre la décision d. Plus cette valeur est grande, plus il semble intéressant de faire ce choix. En général, cette valeur est directement proportionnelle à la qualité du point partiel obtenu en prenant la décision d.
- La trace τ_d représente l'intérêt historique qu'a la fourmi de prendre la décision d. Plus cette quantité est grande, plus il a été intéressant dans le passé de prendre cette décision.
- Étant donnés deux paramètres α et β, la fourmi va prendre la décision d avec une probabilité

$$\frac{(\eta_d)^\alpha (\tau_d)^\beta}{\sum_{r \in D} (\eta_r)^\alpha (\tau_r)^\beta} .$$

Colonies de fourmis (suite)

- Lorsqu'une fourmi termine la construction de son point, elle laisse une trace sur le chemin emprunté. Cette trace est proportionnelle à la qualité du point construit.
- Il est important de mettre en place un processus d'évaporation de la trace afin d'oublier les choix réalisés dans un lointain passé et de donner plus d'importance aux choix réalisés récemment.

Colonies de fourmis (suite)

- Soit ρ un paramètre d'évaporation choisi dans $]0;1[$.
- Soit A l'ensemble des fourmis, et soit $f(a)$ la valeur du point produit par la fourmi a.
- La mise à jour de la trace sur la décision d est réalisée comme suit :

$$\tau_d = (1-\rho)\tau_d + c\sum_{a\in A}\Delta_d(a)$$

où c est une constante et où

$$\Delta_d(a) = \begin{cases} 1/f(a) & \text{si la fourmi } a \text{ a réalisé le choix } d, \\ 0 & \text{sinon.} \end{cases}$$

Colonies de fourmis : Algorithme

ACO

[1] Initialiser les traces τ_d à 0
pour toute décision possible d

[2] Tant qu'aucun critère d'arrêt n'est satisfait
Construire $|A|$ points en tenant compte
de la force gloutonne et de la trace
Recherche locale sur chacun de ces points
Mettre à jour les traces τ_d
Mettre à jour le meilleur point rencontré
Poser $i \leftarrow i + 1$

Colonies de fourmis (suite)

- **Types d'individus :** Points admissibles obtenus à l'aide d'un algorithme constructif.
- **Type d'évolution :** Remplacement générationnel avec une population de taille constante.
- **Structure de voisinage :** Population non structurée.
- **Sources d'information :** Hstorique de la recherche (mémorisé dans la trace).
- **Irréalisabilité :** L'algorithme constructif ne produit pas de point non admissible.
- **Intensification :** Aucune.
- **Diversification :** Aucune.

Méthode à mémoire adaptative (MMA)

- Rochat et Taillard, 1995.
- Extension de la Recherche Tabou qui permet de réaliser automatiquement une diversification et une intensification de la recherche.
- Mémoire centrale stockant les composantes des meilleurs points rencontrés.
- Ces composantes sont combinées afin de créer de nouveaux points.
- Si la combinaison ne produit pas un point admissible, un processus de réparation est mis en œuvre.

Méthode à mémoire adaptative (suite)

- Une descente est ensuite appliquée et les composantes du point ainsi obtenu sont considérées pour éventuellement faire partie de la mémoire centrale.
- Au début de la recherche, la mémoire centrale contient des composantes provenant de points très diverses et le processus de combinaison aura donc tendance à créer une diversité de nouveaux points.
- Plus la recherche avance et plus la mémoire centrale aura tendance à ne mémoriser que les composantes d'un sous-ensemble très restreint de points. La recherche devient donc petit à petit un processus d'intensification.

Méthode à mémoire adaptative : Algorithme

```
MMA
|
|  [1] Générer un ensemble de points et introduire leurs
|       composantes dans la mémoire centrale
|
|  [2] Tant qu'aucun critère d'arrêt n'est satisfait
|      |  Combiner les composantes de la mémoire centrale
|      |      pour créer de nouveaux points
|      |  Réparer les points non admissibles
|      |  Recherche locale sur chaque point admissible
|      |  Mettre à jour la mémoire centrale en ôtant
|      |      certaines composantes et en y insérant
|      |      de nouvelles provenant des nouveaux points
```

Méthode à mémoire adaptative (suite)

- **Types d'individus :** Composantes de points admissibles.
- **Type d'évolution :** Remplacement stationnaire avec une population de taille constante.
- **Structure de voisinage :** Population non structurée.
- **Sources d'information :** Au moins deux parents.
- **Irréalisabilité :** Les points non admissibles sont réparés.
- **Intensification :** Recherche locale et intensification implicite durant les dernières itérations.
- **Diversification :** Diversification implicite durant les premières itérations.

Méthodes hybrides

La tendance actuelle est d'avoir recours à des algorithmes dits hybrides. Il existe plusieurs types d'hybridations possibles :

- On peut utiliser une recherche locale dans les méthodes évolutives : On obtient alors des algorithmes mimétiques. L'avantage est que la recherche locale réduit le danger de passer à côté d'un optimum sans le voir. En règle générale, une méthode évolutive est excellente pour détecter de bonnes régions dans l'espace de recherche alors qu'une recherche locale explore efficacement les régions prometteuses.

Méthodes hybrides (suite)

- On peut exécuter en parallèle diverses métaheuristiques, voire même plusieurs fois la même métaheuristique mais avec divers paramètres. Ces processus parallèles communiquent entre eux régulièrement pour échanger de l'information sur leurs résultats partiels.

- On peut aussi alterner entre plusieurs métaheuristiques lorsqu'une est bloquée.

Méthodes hybrides (suite)

- Une autre forme d'hybridation consiste à combiner les métaheuristiques avec des méthodes exactes. Une métaheuristique peut par exemple fournir des bornes à une méthode de type *branch and bound*. Au contraire, une méthode exacte peut donner lieu à une technique efficace pour la détermination du meilleur voisin d'un point (ce qui peut s'avérer plus judicieux que de choisir le meilleur point parmi un échantillon de voisins) : Matheuristiques.

Conseils généraux

- Ne pas s'attendre à ce qu'une méthode soit efficace pour tous les problèmes/instances.
- Il convient en priorité de considérer des méthodes **simples** possédant peu de paramètres. Selon ce critère, toujours privilégier les méthodes de trajectoires à celles basées sur des populations.
- Les maximes suivantes correspondent à quelques principes qui permettent de guider toute personne intéressée à adapter une des méthodes décrites ci-dessus à un problème concret. Ces principes ont éte rédigés dans la thèse de doctorat de N. Zufferey.

Conseils pour les méthodes de trajectoire

Max. 1 Afin de faciliter la génération d'un point voisin, il ne faut pas hésiter à travailler dans l'espace des points non admissibles ou partiels, en modifiant si besoin est la fonction objectif à optimiser (pénalités).

Max. 2 À partir de n'importe quel point, il devrait être possible d'atteindre un optimum en générant une suite de points voisins.

Max. 3 Afin de bien guider la recherche d'un optimum dans Ω, il faut que les points voisins d'un point x ne soient pas trop différents de x.

Conseils pour les méthodes de trajectoire (suite)

Max. 4 Plutôt que d'optimiser directement la fonction objectif relative à un problème, mieux vaut optimiser une fonction objectif auxiliaire qui donne du relief à l'espace de recherche. De plus, cette fonction doit permettre d'évaluer rapidement un point voisin, l'idéal étant d'évaluer un point voisin en ne calculant que l'écart de valeur qui le distingue du point courant (calcul incrémental).

Max. 5 Pour générer un point voisin, il faut tenir compte des propriétés du problème : Un point voisin x' ne doit pas contenir les mêmes défauts que le point courant x, même si x' est de bien moins bonne qualité que x.

Conseils pour les méthodes basées sur des populations

- Méthodes surévaluées grâce (ou à cause) des métaphores utilisées pour leur nom. Il faut cependant être très prudent.
- Très difficile de considérer des jeux de paramètres génériques.
- De gros efforts seront nécessaires afin de bien choisir les paramètres.
- En cas de succès, il est alors aussi très difficile de comprendre pourquoi.
- À ne considérer que comme méthodes de *dernier recours*.

Conseils pour les méthodes basées sur des populations (suite)

Max. 6 Durant la phase de coopération, des informations pertinentes qui tiennent compte des propriétés du problème traité doivent être échangées.

Max. 7 Il faut disposer d'un outil d'intensification performant. Le plus souvent, cet outil est une procédure de recherche locale efficace (algorithme mimétique).

Max. 8 Afin de ne pas réduire à néant trop rapidement l'effet de l'opérateur de combinaison, il est important de préserver la diversité des informations contenues dans la population.

Max. 9 L'opérateur d'échange d'information doit permettre, s'il combine de bonnes informations, de transmettre ces bonnes informations à la solution enfant.

Introduction

- Comment mesurer la performance d'un algorithme ?
- Comment tester le comportement d'une méthode non-déterministe ?
- Choisir un ensemble de problèmes ou d'instances.
- Tâcher de diviser les problèmes en groupes de problèmes cohérents.
- Faire des tables et/ou des graphes.

Mesures

- Qualité d'un point obtenu :
  - Valeur de l'objectif.
  - Écart par rapport au meilleur point connu : le *gap*.
- Qualité de la convergence :
  - Temps CPU, temps réel. Problème : dépend de la machine, difficilement comparable.
  - Il faut essayer d'utiliser un autre critère que le temps, si possible, indépendant et universel. Exemple : nombre d'évaluations d'une boîte-noire.
- Est-ce qu'on s'intéresse à la qualité du point obtenu après l'exécution de l'algorithme, ou bien durant son déroulement ?

Tables

- Indiquer les caractéristiques de l'instance.
- Les tables sont importantes car elles donnent des valeurs précises, contrairement aux graphes.
- Problème : Une table devient vite illisible (*too much information*).

Tables (exemple)

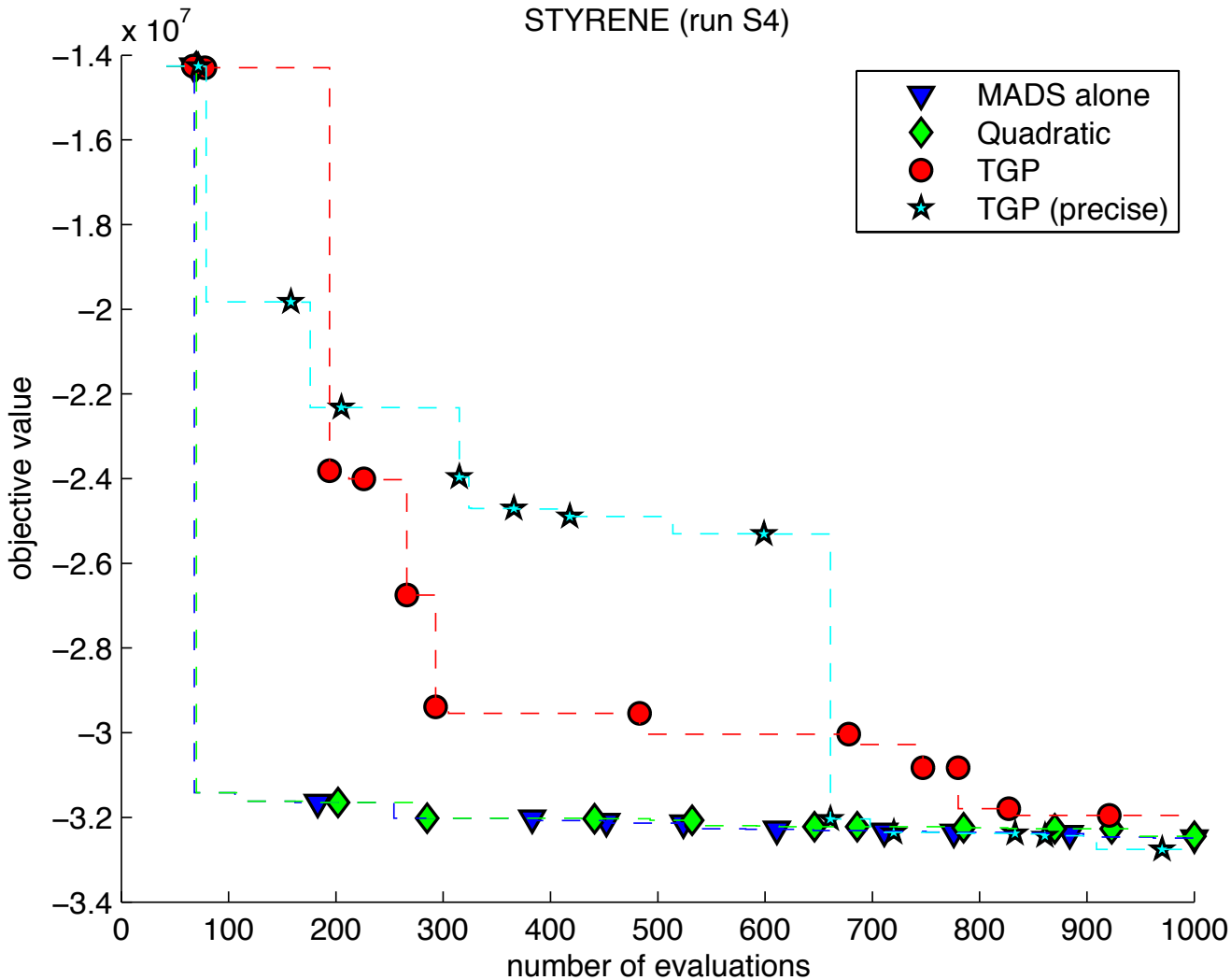
Table 6 **Pooling Test Problems from the Literature**

	Parameters			Solution				CPU time (sec)			Error (%)		
Example	*nsp*	k_{max}	*nt*	Exact	MSLP	MALT	VNS	MSLP	MALT	VNS	MSLP	MALT	VNS
Flow model													
AST1	1,000	10	1	549.803	276.661	532.901	545.27	2.20	2.45	2.81	49.68	3.07	0.82
AST2	1,500	10	1	549.803	284.186	535.617	543.909	9.18	5.21	5.68	48.31	2.58	1.07
AST3	1,000	10	1	561.048	255.846	397.441	412.145	18.71	4.96	5.34	54.35	29.09	26.47
AST4	230	0	0	877.649	—	876.206	876.206	0.82	0.77	1.01	—	0.16	0.16
BT4	5	0	0	45	39.6970	45	45	0.01	0.01	0.01	11.78	0	0
BT5	10	15	2	350	327.016	324.077	350	0.03	0.09	1.11	6.57	7.41	0
F2	120	10	1	110	100	107.869	110	0.07	0.44	0.57	9.09	1.94	0
H1	5	0	0	40	40	40	40	0.02	0.01	0.01	0	0	0
H2	5	0	0	60	60	60	60	0.02	0.01	0.01	0	0	0
H3	5	3	1	75	60.7332	70	75	0.02	0.01	0.03	19.02	6.67	0
RT1	5	0	0	4,136.22	126.913	4,136.22	4,136.22	1.34	0.04	0.04	96.93	0	0
RT2	5	5	1	4,391.83	—	4,330.78	4,391.83	0.04	0.47	0.60	—	1.39	0
GP1	50	5	1	60.5	28.732	35	46	0.01	0.04	0.08	52.51	42.15	23.97
Proportion model													
AST1	1,000	10	1	549.803	544.307	532.901	533.783	1.14	2.38	2.61	1	3.07	2.91
AST2	1,500	10	1	549.803	548.407	535.617	542.54	3.04	4.97	5.37	0.25	2.58	1.32
AST3	1,000	10	1	561.048	551.081	397.441	558.835	4.98	4.98	5.93	1.68	29.09	0.3
AST4	230	0	0	877.649	—	876.206	876.206	1.19	1.21	1.55	—	0.16	0.16
BT4	5	0	0	45	39.7019	45	45	0.01	0.02	0.02	11.77	0	0
BT5	10	15	2	350	292.532	323.12	350	0.12	0.16	1.53	16.42	7.68	0
F2	120	0	0	110	110	110	110	0.15	0.49	0.49	0	0	0
H1	5	0	0	40	40	40	40	0.02	0.01	0.01	0	0	0
H2	5	0	0	60	60	60	60	0.02	0.01	0.01	0	0	0
H3	5	3	1	75	69.9934	70	75	0.02	0.01	0.02	6.68	6.67	0
RT1	5	0	0	4,136.22	3,061.03	4,136.22	4,136.22	0.07	0.03	0.03	25.99	0	0
RT2	5	5	1	4,391.83	4,391.02	4,330.77	4,391.82	0.04	0.58	0.72	0.02	1.39	0

Graphe de convergence

- On trace l'historique d'une exécution sur un graphe "qualité v.s. convergence".
- Problème : valide uniquement pour une exécution d'une méthode sur une instance particulière.

Graphe de convergence (exemple)



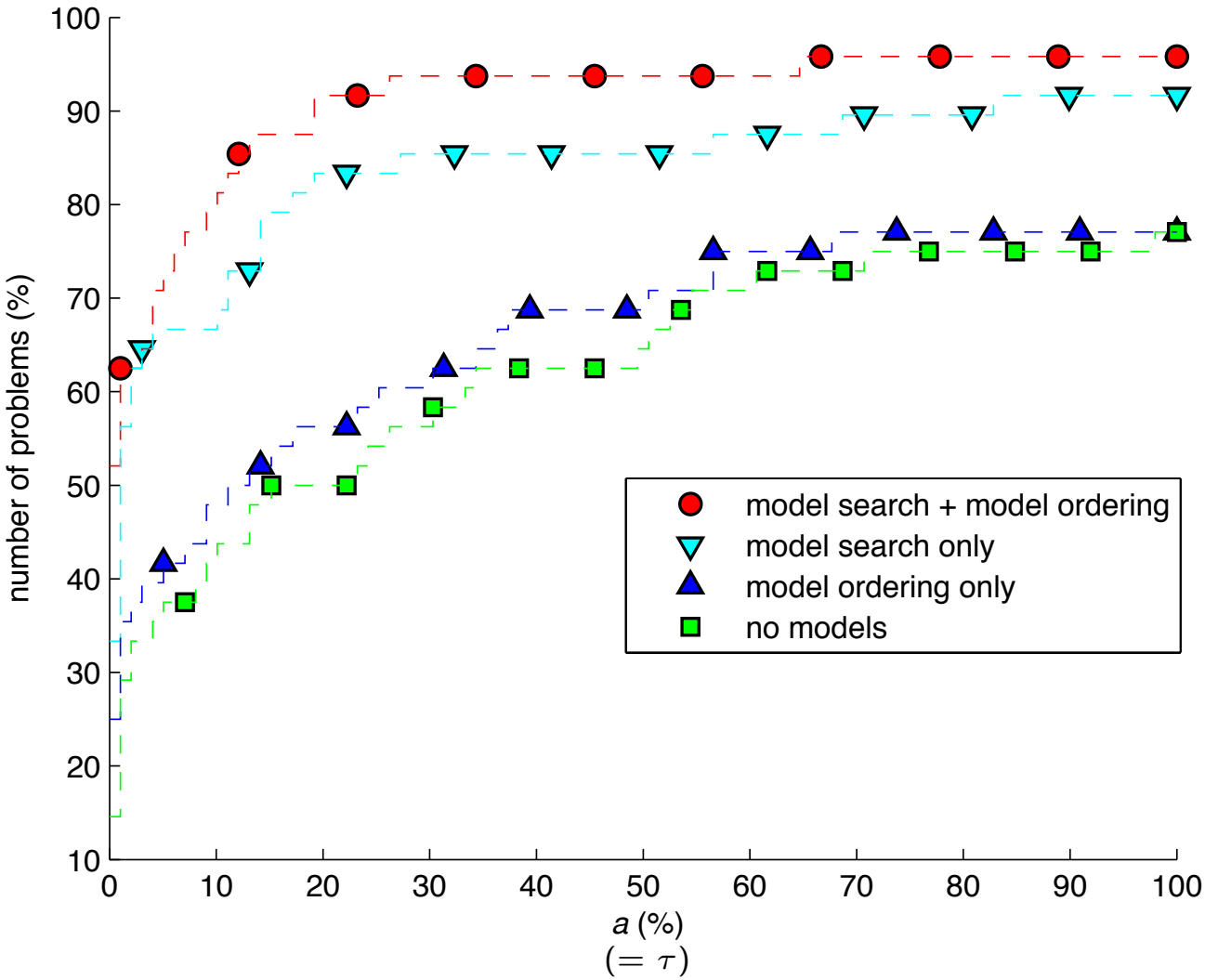
Profils de performance et de données (version simplifée)

- Introduits par Dolan, Moré, Wild (2002 et 2009).
- Beaucoup plus parlants qu'une table. Permettent de comparer des variantes algorithmiques d'un seul coup d'oeil.
- On considère :
  - Un ensemble d'algorithmes (ou de variantes algorithmiques), à comparer.
  - Un ensemble d'instances (les problèmes).
- On a le résultat de l'exécution de tous les algorithmes sur tous les problèmes. À partir de ceci, on mémorise, pour chaque problème, le meilleur point obtenu.

Profil de performance

- *Performance profile (pp)*.
- Considère uniquement les points finaux. Les critères d'arrêt doivent donc être similaires pour tous les algorithmes (même temps max, même critère de terminaison si on étudie le même algorithme, etc.)
- Axe horizontal : Une précision τ.
- Axe vertical : Proportion des problèmes résolus à τ près.
- Pour $\tau = 0$, on voit donc directement les algorithmes qui ont donné le meilleur point.

Profil de performance (exemple)



Profil de données

- *Data profile (dp)*.
- On considère l'historique du déroulement des algorithmes, donc la convergence.
- On fixe la précision τ.
- Axe horizontal : La convergence (temps, nombre d'évaluations, etc.).
- Axe vertical : Proportion des problèmes résolus à τ près.
- Pour $x = 0$, on devrait observer $y = 0$. Pour $x = x_{max}$, on doit voir les résultats finaux qui correspondent à l'abscisse τ d'un pp.

Data profile (exemple)