

# Algorithmes stochastiques

Vincent Berry

Master IC

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Approche gloutonne</b>	<b>3</b>
<b>3</b>	<b>Algorithmes stochastiques</b>	<b>4</b>
3.1	Amélioration itérative . . . . .	6
3.2	Garanties d’optimalité . . . . .	8
3.3	Le Recuit Simulé . . . . .	8
3.3.1	Chaînes (ou modèles) de Markov . . . . .	10
3.4	Algorithmes Génétiques . . . . .	10
3.4.1	Théorie de l’évolution . . . . .	11
3.4.2	Calcul (r)évolutionnaire! . . . . .	11
3.4.3	Algorithme générique . . . . .	12
3.4.4	Un exemple . . . . .	12
3.4.5	Codage d’une solution et combinaison de solutions . . . . .	13
<b>4</b>	<b>Méthode du bruitage [Sharon 93]</b>	<b>14</b>
4.1	1 <sup>re</sup> variante . . . . .	14

4.2	2 <sup>e</sup> variante . . . . .	14
4.3	Résultats sur le TSP . . . . .	15
4.3.1	TSP avec valuation aléatoire (ie dans un graphe) . . . . .	15
4.3.2	TSP euclidien (ie, dans un espace à $k$ dimensions) . . . . .	15
<b>5</b>	<b>Conclusion - Méthodologie</b>	<b>15</b>
<b>6</b>	<b>Annexe : Le problème CSOAM</b>	<b>17</b>
6.1	Algorithme génétique pour CSOAM . . . . .	17

# 1 Introduction

Beaucoup de problèmes quelque soit leur champ d'application peuvent être formulés comme des **problèmes d'optimisation**.

Face à un problème d'optimisation, les méthodes les plus naturelles sont

- approche diviser pour régner
- programmation dynamique
- algorithme glouton
- programmation linéaire ou en nombre entiers
- algorithme de satisfaction de contraintes

La plupart des problèmes d'optimisation sont **NP-difficiles**, aussi on optera rarement en pratique pour un algo exact, en raison du temps astronomique qu'il peut requérir pour traiter la majorité des instances.

## 2 Approche gloutonne

Il s'agit d'algorithmes cherchant à résoudre le problème en un petit nombre d'étapes. Les étapes fixent successivement la valeur de toutes les variables du problème sans jamais reconsidérer les choix des étapes précédentes. On dit qu'ils "avalent" les variables (d'où le nom "glouton").

Ces algorithmes se résument à la forme suivante :

```
Tant qu'il y a des variables à fixer faire
  début
    [Xa ...Xb] <- Choisir-variable(s)-non-encore-instanciée(s)
    valeur[Xa ...Xb] <- Fixer-au-mieux (Xa ...Xb , Xa-1, .... ,X1)
  fin "tant que"
valmin <- f ( X1, ... , Xn )
Renvoyer ( valmin )
```

Les avantages des algorithmes gloutons sont principalement une grande simplicité de mise en oeuvre ainsi qu'un temps d'exécution raisonnable. En contrepartie, ils peuvent parfois aboutir à une solution relativement éloignée de la solution optimale. La qualité de la solution fournie dépend énormément (mais pas entièrement) des fonctions Choisir et Fixer. On essaye habituellement plusieurs variantes de ces fonctions pour équilibrer au mieux qualité de la solution et temps d'exécution.

Pour le problème CSOAM (cf annexe), on peut penser aux algorithmes gloutons suivants :

**1ère idée** : fixer les coordonnées d'un atome au hasard, puis à chaque étape décider les nouvelles coordonnées d'un nouvel atome possédant une liaison en commun avec un des atomes précédemment placés, ceci en utilisant la restriction  $f'$  de  $f$  ne tenant pas compte des atomes non placés. Étant donné l'expression de  $f$  on peut mathématiquement placer le nouvel atome de façon optimale par rapport aux atomes voisins déjà placés. On espère ainsi qu'une suite de placements localement optimaux conduira à obtenir une configuration finale proche de l'optimum global.

**2ème idée** : décomposer la molécule en plusieurs "paquets" d'atomes adjacents de petite taille, pour lesquels on connaît la configuration optimale, puis essayer de recombinaison au mieux les différents paquets, en les ajoutant successivement les uns aux autres, jusqu'à reformer l'intégralité de la molécule.

### 3 Algorithmes stochastiques

Une alternative aux approches ci-dessus est d'avoir recours à un **algorithme stochastique**.

DÉFINITION : on dit qu'un algorithme est **stochastique** si son comportement est déterminé par l'entrée mais aussi par des valeurs produites par un générateur de nombres aléatoires<sup>1</sup>. Donc, plusieurs exécutions successives de tels programmes ne produisent pas forcément le même résultat.

En vertu de ce principe, il s'agit d'algorithmes **heuristiques**, *i.e.*, ne fournissant pas forcément une solution optimale. Toutefois, certains ont des **garanties de convergence**, au sens où la probabilité qu'ils trouvent une solution optimale augmente avec le temps d'exécution qu'on leur octroie.

La plupart des algorithmes stochastiques consistent à **explorer l'espace des solutions**, passant de l'une à l'autre suivant une logique propre à chacun.

Ces techniques s'appliquent essentiellement aux **problèmes d'optimisation combinatoire**, bien qu'ils puissent aussi s'appliquer à l'optimisation de fonctions numériques.

Tout problème d'optimisation combinatoire  $\pi$  peut être défini par un couple  $(\mathcal{R}, f)$  où  $\mathcal{R}$  est un ensemble fini de *configurations* ou solutions de  $\pi$  ( $\mathcal{R}$  est appelé **espace des configurations**) et  $f : \mathcal{R} \mapsto \mathfrak{R}$  associe une valeur à toute configuration.

Résoudre  $\pi$  consiste à trouver  $r_{min} \in \mathcal{R}$  t.q.

$$f(r_{min}) = \text{Min}_{r \in \mathcal{R}} f(r) \quad (1)$$

$\mathcal{R}$  peut être schématisé par un **paysage** dans un espace à  $x + 1$  dimensions, où  $x$  est le nombre de variables décrivant une configuration  $r$  et la dernière coordonnée est  $f(r)$ , donne la **hauteur** de  $r$  dans le paysage .

Les configurations voisines,  $\mathcal{R}_r$  d'une configuration  $r$ , sont les points du paysage entourant  $r$ , ie les configurations qui ne diffèrent que peu de  $r$  (par exemple, par la valeur d'un paramètre).

Il est généralement impossible de représenter ce paysage, en raison du **gd nombre de variables** qui interviennent pour décrire une configuration mais aussi du fait que toutes les **variables ne sont pas ordonnées**, ie il n'existe pas d'ordre total sur les valeurs d'une variable, et il est difficile d'en imposer un qui soit arbitraire (en utilisant une seule dimension) *e.g.*, la couleur d'un sommet dans le cas d'un problème de coloration (une couleur est à distance 1 de toute autre couleur, impossible d'étaler les couleurs le long d'un seul axe, tq la distance entre deux couleurs soit significative d'une distance entre les configurations).

**QUESTION** : il existe toutefois une solution pour faire que le codage de toute couleur soit à même distance que le codage de tout autre couleur, en acceptant d'augmenter le nombre de dimensions. Quelle est cette solution ? ?

**SOLUTION** : il suffit d'ajouter  $k$  dimensions binaires où  $k$  est le nombre de couleurs. On ordonne arbitrairement les couleurs  $c_1 \prec \dots \prec c_k$  puis on code chaque couleur  $c_i$  comme étant le vecteur de  $k$  bits où seul le  $i^{eme}$  bit est à 1. La distance entre toute couleur est une autre est la même : 2 bits de différence.

En raison de ces problèmes, on schématisera le paysage des configurations par un simple graphique à deux dimensions, où l'axe des abscisses représente l'ensemble des configurations  $r \in \mathcal{R}$  et l'axe des ordonnées représente  $f(r)$ . Ex :

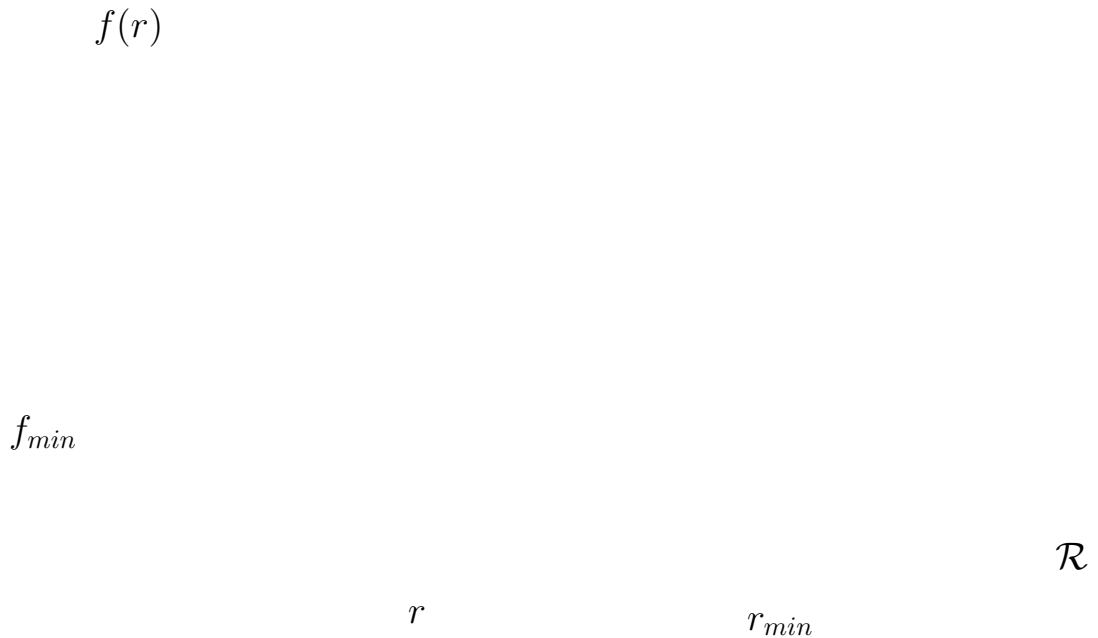


FIG. 1 – Paysage des configurations  $\mathcal{R}$  d'un problème  $\pi$ .

Le principe des algorithmes stochastiques est de se faire une idée de ce paysage pour déterminer le point du paysage ayant l'altitude la plus basse. Cet examen ne couvre qu'une fraction du paysage (qui est de taille exponentielle), mais les endroits examinés doivent couvrir le mieux possible l'étendu du paysage.

Les différents algorithmes stochastiques que nous verrons dans le cours sont **l'amélioration itérative**, **le recuit simulé**, **le tabou**, **les algorithmes génétiques**.

### 3.1 Amélioration itérative

PRINCIPE :

1. Partir d'une configuration initiale  $r$
2. Tant que  $\exists r' \in \mathcal{R}_r, f(r') \geq f(r)$  faire
3. Choisir au hasard une configuration  $r' \in \mathcal{R}_r : f(r') < f(r)$ .
3.  $r'$  devient la configuration courante  $r$
4. Renvoyer  $r$

L'application de ce principe consiste, dans le paysage, à descendre au fond de la

vallée où est située la configuration initiale. En d'autres mots, cela revient à laisser tomber une bille sur le paysage, elle va descendre autant que possible en suivant la ligne de plus grande pente jusqu'au fond d'une vallée. On a ainsi la garantie d'obtenir une **solution localement optimale**, aussi appelé un "*minimum local*", ie une solution qui ne peut être améliorée en se rendant à une de ses voisines, autrement dit qui correspond à l'altitude minimale dans la vallée du paysage où elle se trouve. Rappelons que ce qui nous intéresse est d'obtenir un **minimum global** (le point le plus bas de tout le paysage).

Cette approche marchera bien dans le cas d'un paysage à une seule vallée, mais plus le paysage devient cahotique, ie plus il possède de minima locaux, moins cette méthode a de chance de trouver un minimum global.

Cette approche présuppose

- qu'on sait obtenir une **configuration initiale** : le plus souvent on aura recours à un algorithme glouton afin de démarrer avec une solution pas trop mauvaise.
- qu'on sait **générer le voisinage  $\mathcal{R}_r$  d'une configuration  $r$** .

EXEMPLE : Problème TSP.

DONNÉES :  $G = (V, E)$ ,  $L : E \mapsto \mathfrak{R}$ , un entier

QUESTION : trouver un cycle hamiltonien de longueur minimale

Une configuration initiale pas trop mauvaise peut être obtenue par une **méthode gloutonne**. Par exemple on trie les arêtes par longueur croissante et on les ajoute dans l'ordre de taille croissante que si elles ne créent pas de cycle tant que l'ensemble des sommets ne sont pas reliés.

Pour une configuration donnée  $C$  on peut considérer que son voisinage est l'ensemble des configurations  $C'$  qui peuvent être obtenues en permutant les extrémités de deux arêtes  $e = (x, y)$ ,  $e' = (z, t)$  de  $C$  :  $C' = C - \{(x, y), (z, t)\} \cup \{(x, z), (y, t)\}$ . Avantage de cette transposition :  $f(C')$  peut être calculé en  $O(1)$  depuis  $f(C)$ .

FIG. 2 – Transposition d’une configuration (un circuit) à une configuration voisine (un circuit voisin) pour le problème du TSP.

Pour avoir plus de chances de trouver un minimum global et non local, ...

Partir de plusieurs points de départ dans le paysage (plusieurs billes qui roulent)

### 3.2 Garanties d’optimalité

Quand il y a un matroïde orienté dans la structure du problème étudié, la technique de l’amélioration itérative conduit à une solution optimale.

Ex d’un tel cas : algorithme du *Simplex*, qui fonctionne par améliorations successives de la solution proposée (cf dessin A. Dicky).

Ex où on n’a pas cette garantie : TSP, BIN-PACKING (découpe de verre).

Echanger un sommet :  $O(n^2)$ , mais on peut échanger plus de sommets pour passer d’une solution à l’autre : ... cf page AIA 5 (dos) ... définir la *k*-optimalité...

### 3.3 Le Recuit Simulé

Cette technique algorithmique est issue d’une analogie avec le monde de la sidérurgie, où pour amener un amas de métal à sa forme finale, on procède par paliers de températures décroissantes (”recuits”). À chaque palier le métal subit des déformations allant globalement dans le sens de la forme finale qu’on veut donner à l’objet.

L’algorithme du recuit simulé diffère principalement de l’amélioration itérative en ce qu’il accepte des dégradations de la configuration avec une certaine probabilité, ceci afin d’éviter d’être prisonnier d’un minimum local. On simule les ”recuits” à

l'aide d'une variable représentant la température, qu'on utilise pour faire varier la probabilité d'accepter une configuration moins bonne que la configuration courante : par analogie avec le travail du métal, plus la température est élevée, plus l'objet peut subir des déformations importantes y compris dans le mauvais sens. A l'inverse, plus on le refroidi, moins les déformations négatives sont possibles, donc plus la probabilité d'accepter une mauvaise configuration doit diminuer. Ainsi en fin d'exécution l'algorithme de recuit simulé se comportera comme l'algorithme d'amélioration itérative.

Cet algorithme possède l'étonnante propriété de converger de façon sûre vers l'optimum global (ie, avec une probabilité de 1), .... mais seulement pour un temps d'exécution infini, et sous certaines conditions. On peut toutefois arrêter la recherche au bout d'un nombre raisonnable d'étapes et obtenir une solution très proche de l'optimal. Les conditions de convergence portent principalement sur la règle d'acceptation d'une nouvelle configuration et sur l'espace de recherche ; par exemple il faut que toute configuration soit accessible depuis n'importe quelle autre en un nombre fini d'étapes.

FIG. 3 – Algorithme du recuit simulé

L'algorithme du recuit simulé est détaillé sur la figure 3. La règle d'acceptation d'une nouvelle configuration employée ici est celle de Metropolis. Elle possède les propriétés requises pour que la convergence de l'algorithme soit assurée. Elle exprime la probabilité d'accepter une configuration moins avantageuse en fonction de la température et de la différence énergétique avec la configuration précédente.

La température peut être décrementée de façon géométrique ( $T \leftarrow T \times 0.9$ ) ou de façon constante ( $T \leftarrow T - cte$ ). La notion d'*équilibre*, qui signifie en métallurgie que pour une température donnée on a effectué toutes les déformations voulues sur l'objet, n'a pas d'équivalent précis dans le recuit simulé. Généralement l'algorithme travaille par paliers de température, c'est-à-dire que la température reste fixe pendant l'exploration d'un certain nombre de configurations. La taille de chaque palier est fixée en fonction du temps qu'on juge suffisant à l'algorithme pour explorer le voisinage d'une configuration et trouver la plus intéressante vallée du paysage énergétique, au voisinage de la configuration courante. La finesse de cette exploration du paysage dépend de la température : plus celle-ci est basse, plus l'exploration est concentrée autour de la configuration courante, donc plus elle est de grain fin.

A chaque fois qu'on atteint l'équilibre, on "refroidit" le système en baissant la température. L'algorithme se termine quand le système est considéré "gelé", ie aucune configuration voisine n'est plus susceptible d'être acceptée.

Cet algorithme explore le paysage énergétique en effectuant des "bonds" d'une configuration à l'autre. Au départ, on accepte qu'il remonte pour cela à un niveau du relief plus haut que celui atteint précédemment, mais plus la température diminue, plus les "bonds" acceptés sont de petite taille. En lui laissant parcourir une large partie du paysage énergétique on espère échapper à un minimum local, bien qu'on ne dispose d'aucune garantie théorique puisque on limite l'exécution à un temps fini.

### 3.3.1 Chaînes (ou modèles) de Markov

## 3.4 Algorithmes Génétiques

Technique d'optimisation basée sur la théorie de l'évolution : la vie s'adapte à tous les milieux en sélectionnant les formes de vies les plus adaptées, ie, en **optimisant** une *fonction de survie*.

### 3.4.1 Théorie de l'évolution

Les algorithmes génétiques furent inventés pour simuler les processus observés lors de l'évolution naturelle des organismes vivants. Même si on est loin de connaître les mécanismes de l'évolution, on sait cependant qu'elle agit sur les chromosomes qui sont la partie codante des caractéristiques d'un organisme, et que la sélection naturelle est fonction de l'adaptation de l'organisme au milieu où il vit. Le phénotype d'un organisme (ses caractéristiques) est l'expression de son génotype, codé dans les chromosomes. On sait aussi que la reproduction est le point auquel l'évolution a lieu : des mutations ainsi qu'un processus de combinaison des chromosomes parentaux amènent les chromosomes de l'enfant à diverger de ceux de ses parents. Enfin il est généralement admis que l'évolution naturelle n'a pas de mémoire, tout ce qui est nécessaire pour engendrer de nouveaux individus est présent dans le génotype des individus vivants.

### 3.4.2 Calcul (r)évolutionnaire !

Dans les années 70, J. Holland s'inspira d'idées proches de ce Darwinisme caricaturé, pour proposer un nouveau type d'algorithmes : les algorithmes génétiques, simulant le processus de l'Evolution. Dans son idée, chaque solution d'un problème combinatoire considéré est codée par un chromosome, et il simule l'évolution d'un ensemble de solutions, une *population*, en combinant de façon répétée ces chromosomes tout en permettant de petites variations aléatoires (mutations).

Analogie :

<b>Nature</b>	<b>Ordinateur</b>
individu	solution à un problème
population	ensemble de solutions
adéquation au milieu	qualité de la solution
chromosome	représentation codée d'une solution
gène	partie de représentation d'une solution
croissance	décodage d'une solution depuis sa représentation
cross-over	opérateurs d'exploration de l'espace $\mathcal{R}$
mutation	modification d'une solution
sélection naturelle	réutilisation d'une bonne (sous-)solution

Les algorithmes génétiques (AG) sont un des types d'algorithmes évolutionnaires (il y en a d'autres). Ils ont pour caractéristiques :

- les chromosomes sont des chaînes de bits ou de lettres
- distinction importante du phénotype % génotype.

- les AG agissent uniquement sur les génotypes des individus
- les phénotypes sont typiquement des listes de paramètres ou de choix.

### 3.4.3 Algorithme générique

Tous les algorithmes génétiques dérivent du schéma suivant :

1. GÉNÉRER ALÉATOIREMENT UNE POPULATION DE CHROMOSOMES
2. DÉCODER CHQ CHROMOSOME POUR OBTENIR UN INDIVIDU
3. EVALUER L'ADÉQUATION DE CHAQUE INDIVIDU
4. GÉNÉRER UNE NOUVELLE POPULATION EN PARTIE PAR COPIE (CLONAGE), EN PARTIE PAR RECOMBINAISON, ET EN PARTIE PAR MUTATION, DEPUIS LES CHROMOSOMES DES MEILLEURS INDIVIDUS.
5. RÉPÉTER 2,3 ET 4 JUSQU'À UNE CONDITION D'ARRÊT

:

NOTES : il n'y a pas d'évolution conjointe de plusieurs générations de populations (une génération en remplace une autre). Le clonage permet malgré ce de simuler la survie d'individus pendant plusieurs générations.

### 3.4.4 Un exemple

$\pi$  : trouver le nombre parmi  $\{0, 1, \dots, 255\}$  dont la représentation binaire contient le maximum d'alternances de bits  $0 \rightarrow 1$ . Solution connue : 85 (01010101).

Décisions préalables à l'exécution :

- chromosomes : 8 bits.
- décodage : conversion classique binaire vers entier
- fonction objectif : conversion de décimal vers binaire et on compte le nombre de transitions  $0 \rightarrow 1$ .
- population : 10 individus. Population initiale choisie aléatoirement.
- sélection : utiliser seulement les 5 meilleurs chromosomes pour la reproduction
- clonage : chaque individu sélectionné est copié
- recombinaison : couper deux chromosomes sélectionnés à une position aléatoire et recomposer un chromosome avec la partie gauche d'un parent et la partie droite de l'autre parent.
- mutation : aucune

### 3.4.5 Codage d'une solution et combinaison de solutions

Il y a deux mécanismes qui lient l'algorithme au problème particulier auquel il est appliqué : le codage d'une solution (d'une configuration) en un chromosome et l'évaluation d'un chromosome. Le codage employé est souvent une traduction binaire de la configuration.

Pour la reproduction, on peut utiliser un croisement uniforme : chaque partie du code des nouveaux chromosomes est tirée au hasard parmi celles de ses deux parents. Les mutations éventuelles sont appliquées sur les nouveaux chromosomes suivant une probabilité faible, mais on peut autoriser un chromosome à supporter plusieurs mutations. Une mutation est réalisée sur un bit en changeant sa valeur (un 0 devient 1, et inversement). Pour simuler la sélection naturelle les parents  $X_i$  sont souvent choisis avec une probabilité proportionnelle à leur valeur pour le problème à résoudre, suivant le principe d'une roulette :

(figure)

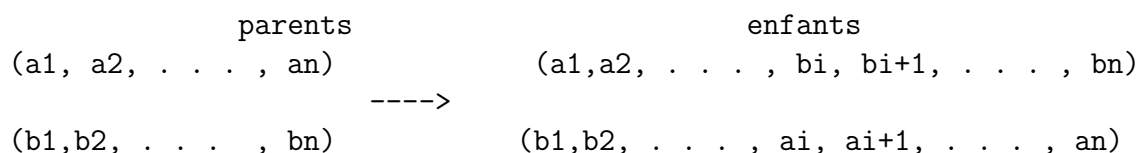
ainsi pour l'exemple  $p = 4$  et  $f(X_i) = 50\%, 25\%, 15\%, 10\%$  :

(figure)

Le nombre d'étapes de l'algorithme est souvent fixé à l'avance. Selon les problèmes il peut varier de 50 à 200 pour des populations de 100 à 1000 chromosomes. On peut bien sûr traduire le fonctionnement d'un algorithme génétique, comme une méthode considérant plusieurs points sur le relief du paysage énergétique et essayant à chaque étape de croiser leurs caractéristiques pour essayer d'obtenir de nouveaux points ayant une valeur énergétique plus faible.

La taille  $p$  de la population est généralement gardée constante pour tout l'algorithme, on peut la renouveler entièrement à chaque étape ou décider de garder les meilleurs parents (au sens de  $f$ ).

La combinaison de deux parents pour créer de nouveaux chromosomes se fait souvent en fixant un (ou plusieurs) point de croisement :



## 4 Méthode du bruitage [Sharon 93]

Basée sur le principe d'**amélioration itérative**.

Spécificité : pour évaluer la valeur  $f(j)$  d'une configuration  $j$  voisine de la configuration  $i$  courante, on ajoute une **erreur** à cette valeur.

L'erreur est en fait un **bruit** choisi aléatoirement avec une distribution uniforme, sur un intervalle centré autour de 0.

La largeur de cet intervalle, appelé **taux**, diminue progressivement au cours de l'algorithme.

Le **test d'arrêt** doit changer ; quand un nombre fixé d'évaluations ont été faites, on stoppe.

*Fausse idée* : on peut penser que le taux doit décroître jusqu'à 0 lorsqu'on s'approche de la fin de l'algorithme. Toutefois la pratique montre qu'on ne change alors plus de configuration et que les résultats sont moins bons que si on s'arrête à un *taux min*.

**Paramètres** : *taux max* et *taux min*, on navigue entre les deux avec une décroissance arithmétique ( $t = t - val$ ).

### 4.1 1<sup>re</sup> variante

*Idée* : rester près de la fonction réelle.

Alterner les périodes bruitées avec des phases d'amélioration itérative non-bruitées. D'une période bruitée à l'autre le taux d'erreur est diminué.

*Avantage* : permet de passer en revue beaucoup de minima locaux.

### 4.2 2<sup>e</sup> variante

*Idée* : revenir à une région prometteuse pour l'explorer plus.

Revenir périodiquement à la meilleure solution rencontrée (qui redevient artificiellement la configuration courante).

## 4.3 Résultats sur le TSP

On appelle *bruitage1* la méthode de bruitage simple et *bruitage2* la méthode qui intègre les 2 variantes décrites ci-dessus.

Les résultats sont obtenus sur la base d'un très petit nombre de graphes choisis dans une bibliothèque de tests.

### 4.3.1 TSP avec valuation aléatoire (ie dans un graphe)

Si on compare les *bruitage1*, *bruitage2* et recuit simulé sur la base de la valeur de la meilleure configuration obtenue en fonction du nombre de configurations qu'ils évaluent, on observe :

- *bruitage2* est toujours le plus efficace.
- *bruitage1* est plus efficace que recuit simulé pour un petit nb d'évaluations (il converge plus rapidement), mais pas si un gd nb d'évaluations sont effectuées.
- à nombre égal d'évaluations, le **recuit simulé est beaucoup plus lent** : utilisation plus importante du générateur aléatoire, de la fonction  $e$  (calcul de la probabilité d'accepter une configuration).

### 4.3.2 TSP euclidien (ie, dans un espace à $k$ dimensions)

- *bruitage1* est plus efficace que le recuit, pour tous les nb d'évaluations essayés.
- l'introduction des deux variantes de bruitage détériore le résultats de la méthode.

## 5 Conclusion - Méthodologie

Pour résoudre un problème d'optimisation (de complexité exponentielle), on dispose de tout un arsenal de méthodes approchées. Bien souvent une connaissance fine du domaine ( donc de la forme de l'espace de recherche) permettra d'orienter les différents algorithmes de façon très favorable. Cette connaissance préalable du domaine n'est toutefois pas nécessaire. On pourra en effet très bien se faire une idée approximative et intuitive sur "la façon dont les choses s'organisent" en commençant par programmer un ou deux algorithmes gloutons (la simplicité des idées initiales est garante d'un très court temps de programmation). Les résultats rapidement obtenus (grâce à une complexité faible) permettent généralement de localiser les points sensibles du domaine. On pourra alors s'orienter vers un algorithme stochastique qui explore l'espace de recherche de façon plus large et néanmoins

convergente. Un tel algorithme aboutit généralement à de très bons résultats en des temps d'exécution raisonnables (polynomiaux). Cet algorithme sera paramétré en fonction des connaissances acquises préalablement sur la base des heuristiques.

## 6 Annexe : Le problème CSOAM

**Problème** : configuration spatiale optimale des atomes d'une molécule (CSOAM)

**Donnée** :

- une molécule, ie un ensemble d'atomes identifiés par l'élément auquel ils appartiennent, ainsi qu'un certain nombre de liaisons (simples, doubles, ...).
- une fonction  $f$  permettant de calculer l'énergie d'une molécule en fonction de la configuration spatiale des atomes qui la compose.

**Question** : quelle est la valeur minimum de  $f$  et la configuration spatiale correspondante des atomes ? Autrement dit, quelle est la forme la plus stable de la molécule ?

La fonction d'énergie  $f$  sera généralement exprimée en tenant compte de plusieurs facteurs comme les énergies de liaisons, de valence, de torsion,... Si pour des molécules "simples" comme H<sub>2</sub>O ou NH<sub>3</sub> on connaît la configuration des atomes la plus stable (d'énergie la plus faible), ce n'est pas le cas pour des molécules comportant un nombre plus conséquent d'atomes.

Les variables dont dépend la fonction d'énergie de la molécule sont les coordonnées spatiales de chaque atome. Ainsi pour  $n$  atomes, on dispose de  $3n$  variables. L'espace de recherche correspond donc aux différentes configurations que peuvent adopter les atomes dans l'espace tridimensionnel. Comme mentionné précédemment, on peut discrétiser cet espace de recherche en fixant par exemple une précision de 3 chiffres après la virgule pour les coordonnées des atomes. Etant donné une configuration solution, il existe une infinité de solutions équivalentes correspondant aux translations de cette configuration dans une quelconque direction. C'est pourquoi on limite le nombre de solutions possibles en contraignant la position dans l'espace des atomes à l'intérieur d'une certaine "boîte". Les dimensions de cette boîte doivent toutefois être suffisamment grandes pour que la molécule puisse adopter toutes les configurations possibles sans entrave. On peut aussi fixer la position d'un atome, ou d'une liaison,... pour toute la durée des calculs.

### 6.1 Algorithme génétique pour CSOAM

Dans le cadre du problème CSOAM, un chromosome représente une configuration spatiales des atomes de la molécule. Pour coder cette configuration on décidera par exemple d'attribuer un même nombre fixe de bits à chaque atome, partagé en trois parties pour chacune des coordonnées de l'atome (figure).

#### Codage possible pour un chromosome

La portion de chromosome dédiée à chaque coordonnée doit être suffisante pour pouvoir coder la précision décidée au départ du problème. La fonction utilisée pour évaluer les

chromosomes est bien sûr la fonction  $f$  qu'on cherche à minimiser. On peut l'appliquer à chaque chromosome après décodage de la configuration qu'il représente.