Université de Jijel
Faculté des Sciences Exactes et Informatique
Département Informatique

2014

Introduction aux Métaheuristiques

Support de Cours

Introduction aux Métaheuristiques

Spécialités

Système d'Information et Aide à la Prise de Décision (SIAD)

Intelligence Artificielle (IA)

Auteur

Dr. LEMOUARI ALI

Ministère de l'Enseignement Supérieure et Recherche Scientifique
Université de Jijel

Faculté des Sciences Exactes et Informatique
Département Informatique

Support de Cours

Introduction Aux Métaheuristiques

Spécialité

Système d'Information & Aide à la Prise de Décision (SIAD)

Intelligence Artificielle

Présenté par

Dr. LEMOUARI Ali

Introduction Aux Métaheuristiques

Le cours d'introduction aux Métaheuristiques préparé pour servir comme support pédagogique d'étudiants inscrits en première année Master de spécialités système d'information et aide à la prise de décision (SIAD) et intelligence artificielle (IA) filière informatique. Le support peut être utile pour d'autre filière des sciences technologiques et exactes,… et autres. Plus précisément dans des disciplines où une résolution d'un problème non polynomial s'impose.

Sommaire

Chapitre 1. Introduction Générale	1
1 Introduction	2
2 Optimisation combinatoire	3
3 Intensification et diversification	3
4 Classification des méthodes de résolution	4
Chapitre 2. Algorithmes et Complexités	6
1 Introduction	7
2 Complexité Algorithmique	8
2.1 Notion d'algorithme	8
2.2 Analyse d'algorithme	9
2.3 Quelques Propriétés	9
3. Notation asymptotique	10
3.1 Propriétés de la Notation	
4. Classes des problèmes	12
4.1 Classe des problèmes P et NP	13
4.2 Classe des problèmes NP complet et NP difficile	13
5. Problèmes de décisions	14
6. Problème de satisfiabilité	15
7. Réduction des problèmes	15
8. Série d'Exercice	16
Chapitre 3. Heuristiques Basées Solution	18
1. Introduction	19
2. Méthode de la descente	20
3. Méthode Hill Climbing	22
4. Méthode de recuit simulé	23
5. Méthode de recherche taboue	25
6. Exercices	27
Chapitre 4. Approche Evolutionnaire : Algorithme génétique	29
1. Introduction	30
2. Principes d'algorithme génétique	31
2.1 Codage des variables	31
2.1.1 Codage binaire	31
2.1.2 Codage réel	33
2.2 Population Initiale	33
2.3 Fonction d'adaptation	33
2.4 Sélection : Algorithmes et méthodes	34
2.4.1 Méthode de sélection par roulette	34

2.4.2 Méthode de sélection par élitisme	36
2.4.3 Méthode de sélection par tournoi	37
2.4.4 Méthode de sélection par rang de classement	38
2.5 Croisement	39
2.5.1 Croisement binaire	39
2.5.2 Croisement réel	41
2.6 Mutation	44
2.6.1 Mutation binaire	44
2.6.2 Mutation réel	44
3. Algorithme génétique	46
4. Convergence et mesure de performance d'un AG	46
5. Exercices	47
Chapitre 5. Optimisation par Colonie de Fourmis	50
1. Introduction	51
2. Intelligence collective des fourmis	52
2.1 Communication dans une colonie des fourmis	52
2.2 Principes de la stigmergie	53
2.2.1 La Phéromone	54
2.2.2 La Phéromone	54
2.2.3 La Phéromone	54
2.3 Le Fourragement	54
3. Optimisation par colonie des fourmis	56
3.1 Principe de l'algorithme ACO	56
3.1.1 Evaporation de la trace de phéromone	57
3.1.2 Renforcement des traces	58
3.2 Algorithme ACO	58
4 Application au voyageur de commerce	58
Chapitre 6. Algorithmes à base d'essaims particulaires	62
1. Introduction	63
2. Algorithme de base PSO	64
3. Essaims particulaire et notion de voisinage	65
3.1 Modèle topologique de base	65
3.2 Autres topologies	65
3.3 Essaim de particule entièrement informé	66
4. Application	68
4.1 Cas continue	68
4.2 Cas discret	73
Annexes	
1. Travaux Pratiques	79
2. Partiel 2009-2014	81

Chapitre I

Chapitre I
Introduction Générale

Chapitre 1
Introduction Générale

Les métaheuristiques forment un ensemble de méthodes utilisées en recherche opérationnelle et en intelligence artificielle pour résoudre des problèmes d'optimisation réputés difficiles. Résoudre un problème d'optimisation combinatoire, c'est trouver l'optimum d'une fonction, parmi un nombre fini de choix, souvent très grand. Les applications concrètes sont nombreuses, que ce soit dans le domaine de la production industrielle, des transports ou de l'économie – partout où se fait sentir le besoin de minimiser des fonctions numériques, dans des systèmes où interviennent simultanément un grand nombre de paramètres.
De nombreuses définitions ont été faites dans la littérature, dans ce chapitre nous retiendrons que deux définitions "*A metaheurisic is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework which can be applied to different optimization problems with relatively few modifications to make them adapted to a specific problem."* http://www.metaheuristics.net

"*A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a constructive method."* (S. Voß, et al. 1999).

Alors une métaheuristique est une méthode algorithmique capable de guider et d'orienter le processus de recherche dans un espace de solution, souvent très grand à des régions riches en solutions optimales. Le fait de rendre cette méthode abstraite et plus générique conduite à un vaste utilisation pour des champs d'applications différents.

A ces applications, les métaheuristiques permettent, de trouver des solutions, peut-être pas toujours optimales, en tout cas très proches de l'optimum et en un temps

raisonnable. Elles se distinguent en cela des méthodes dites exactes, qui garantissent certes la résolution d'un problème, mais au prix de temps de calcul prohibitifs.

Optimisation Combinatoire

En mathématique, l'optimisation combinatoire recouvre toutes les méthodes qui permettent de déterminer l'optimum d'une fonction avec ou sans contraintes. Soit S un ensemble de solutions à un problème d'optimisation et f une fonction objective qui mesure la valeur $f(s)$ avec $s \in S$. Pour un problème de minimisation on cherche à déterminer une solution s qui minimise la fonction objective. Un minimum est une solution qui fait partie des solutions réalisable, dans le domaine d'optimisation on distingue deux types de minimums :

- **Minimum Local** : une solution s est minimum local par apport à une structure de voisinage N si $\forall s' \in N(s), f(s) \leq f(s')$.
- **Minimum Global** : une solution s est minimum global si $\forall s' \in S, f(s) \leq f(s')$.
- **Voisinage** : le voisinage est une fonction notée N qui associe un sous ensemble de S à toute solution s, les voisins de s sont $s' \in N(s)$.

Intensification et diversification

Le principe d'intensification et de diversification est un point critique pour tout métaheuristique, il consiste à trouver un compromis entre les deux tendances duales suivantes :

- Il s'agit d'une part d'intensifier l'effort de recherche vers les zones les plus prometteuses de l'espace de solutions.
- Il s'agit d'autre part de diversifier l'effort de recherche de façon à être capable de découvrir de nouvelles zones contenant de meilleur es combinaisons.

La façon d'intensifier ou de diversifier la recherche dépend d'une métaheuristique à une autre et dans la plus part des cas la modification des paramètres, dont le métaheuristique dépond, permettent à celle-ci d'échapper à une convergence prématurée, autrement dit d'échapper d'un minimum local. Pour les approches dites perturbatives, l'intensification de la recherche se fait en favorisant l'exploration des meilleurs voisins d'une solution. La diversification d'une approche perturbative se fait généralement en introduisant une part d'aléatoire, par exemple autorisé avec une faible probabilité la recherche à choisir des voisins de moins bonne qualité.

Pour les approches constructives, l'intensification de la recherche se fait en favorisant, à chaque étape de la construction, le choix de composants appartenus aux meilleurs combinaisons précédemment construites. La diversification se fait en introduisant une part d'aléatoire permettant de choisir avec une faible probabilité de moins bons composants.

En général, plus on intensifie la recherche d'un algorithme en l'incitant à explorer les combinaisons proches des meilleures combinaisons trouvées, et plus il converge rapidement. Cependant, si l'on intensifie trop la recherche, l'algorithme risque d'être stagner autour d'optima locaux.

L'équilibre entre intensification et diversification dépend du temps de calcul dont on dispose pour résoudre un problème donné. Plus ce temps est petit et plus on a intérêt à favoriser l'intensification pour converger rapidement, quitte à converger vers des combinaisons de moins bonne qualité. Cet équilibre dépend également de l'instance du problème à résoudre, plus particulièrement de la topologie de répartition des solutions réalisables.

Différentes approches ont été proposés pour adapter les valeurs des paramètres manipulés. Les plus répondues sont ceux qui adoptent dynamiquement les valeurs au cours de la recherche de solution. Enfin, difficile de trouver les valeurs adéquates permettant d'intensifier et de diversifier la recherche à la fois.

Classification des Métaheuristiques

Les problèmes d'optimisation combinatoire sont souvent des problèmes très difficiles dont la résolution par des méthodes exactes peut s'avérer très longue ou peu réaliste. L'utilisation de méthodes heuristiques permet d'obtenir des solutions de bonne qualité en un temps de résolution raisonnable. Les heuristiques sont aussi très utiles pour le développement de méthodes exactes fondées sur des techniques d'évaluation et de séparation (Branch and Bound).

Une heuristique est un algorithme qui a pour but de trouver une solution réalisable, sans garantie d'optimalité, contrairement aux méthodes exactes qui garantissent des solutions exactes. Comme les algorithmes de résolution exacte sont de complexité exponentielle pour les problèmes difficiles, il peut être plus judicieux de faire appel aux heuristiques pour calculer une solution approchée d'un problème ou aussi pour accélérer le processus de résolution exacte. Généralement une heuristique est conçue pour un problème particulier, mais les approches peuvent contenir des principes plus généraux. On parle de métaheuristique.

Une manière de classifier les métaheuristiques est de distinguer celles qui travaillent avec une population de solutions de celles qui ne manipulent qu'une seule solution à la fois. Les méthodes qui tentent itérativement d'améliorer une solution sont appelées méthodes de recherche locale ou méthodes de trajectoire.

La méthode Tabou, le Recuit Simulé et la Recherche à Voisinages Variables sont des exemples typiques de méthodes de trajectoire. Ces méthodes construisent une trajectoire dans l'espace des solutions en tentant de se diriger vers des solutions optimales. L'exemple le plus connu de méthode qui travaille avec une population de solutions est l'algorithme génétique. La figure suivante donnera un panorama des méthodes les plus utilisées.

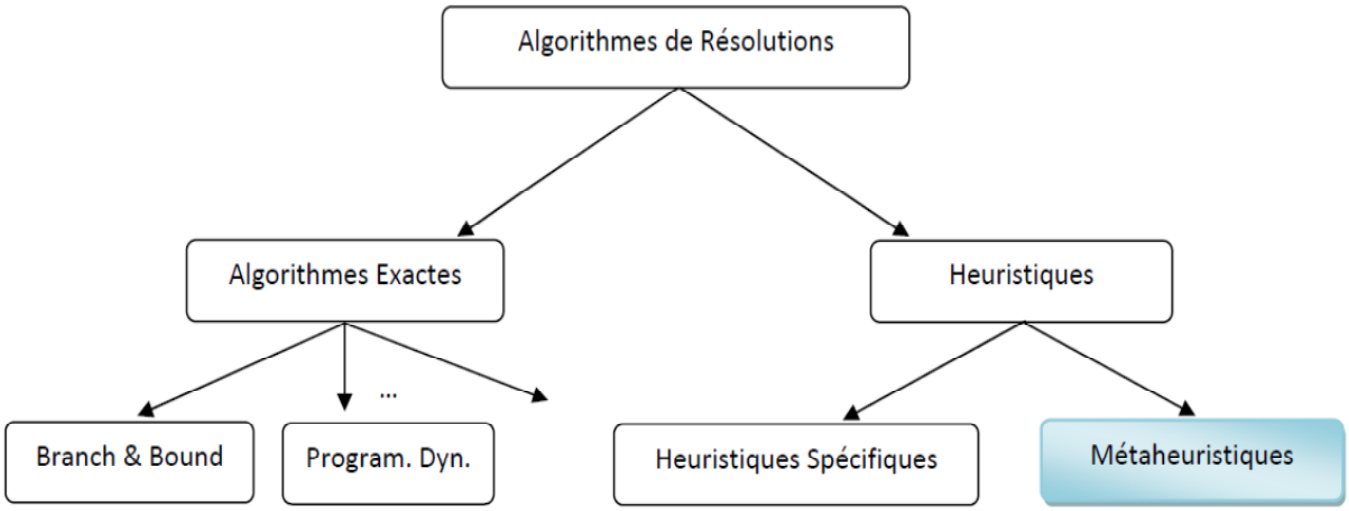


Fig 1. Classes des méthodes de résolutions.

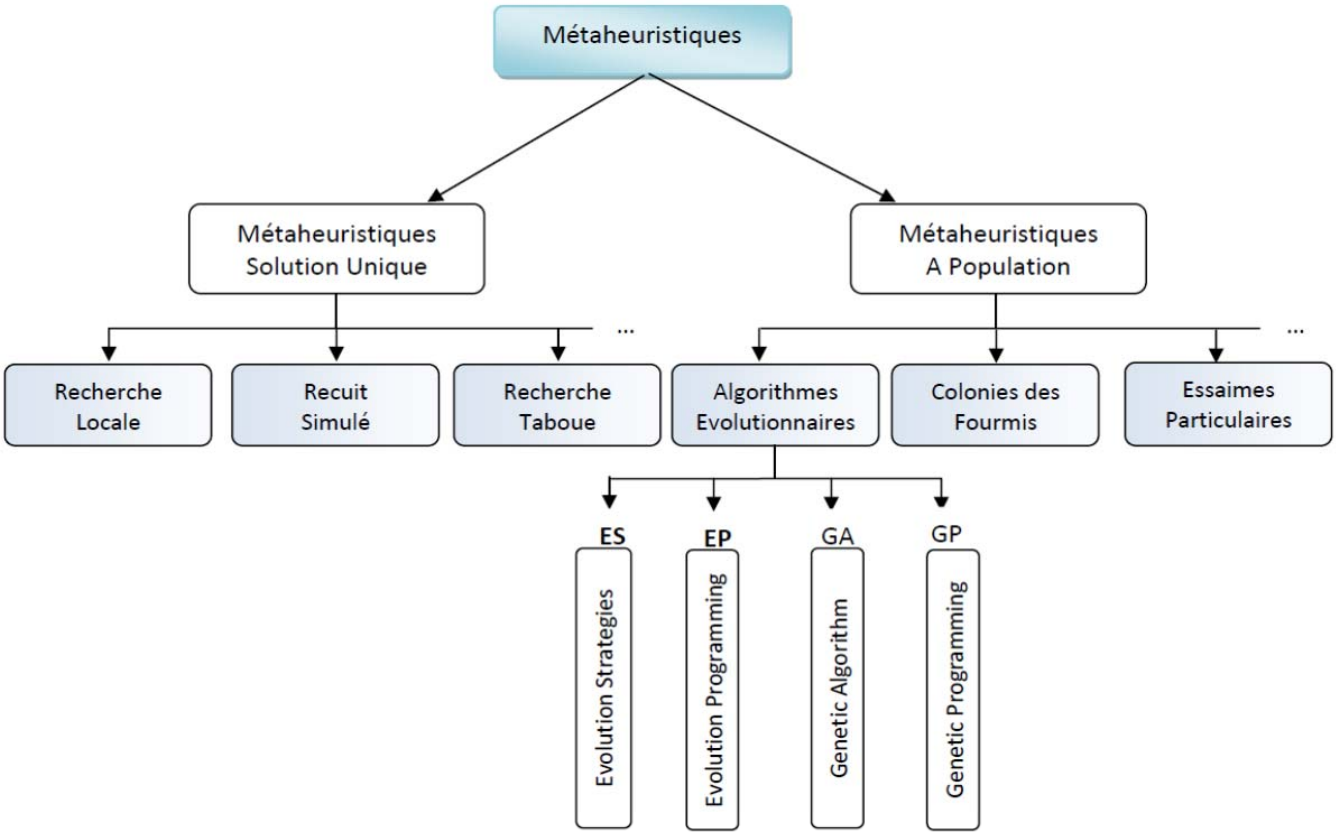


Fig 1. Classes des méthaheuristiques.

Chapitre 2

Chapitre 2
Algorithmes et Complexités

Chapitre 2
Algorithmes et Complexités

1. Introduction

La recherche de l'algorithme ayant la plus faible complexité, pour résoudre un problème donné, est souvent une tâche difficile pour ceux qui n'ont pas subis une formation rigoureuse en algorithmique et structure de données. Développer un algorithme excessivement très compliqué en structure de contrôle, ayant une complexité en temps exponentielle, c'est-à-dire dont le temps de résolution s'étale à des heures voir des jours, n'a pas de sens comme méthode de résolution. Alors qu'il existe un algorithme simple et clair de complexité polynomial, c'est-à-dire qui peut s'exécuter et donne des résultats satisfaisants dans une durée du temps suffisante pour l'utilisateur. Cependant la conception d'algorithme de résolution traitant un problème, qui respecte les critères de clarté et de simplicité doivent être considérés comme aussi importants que celui de l'efficacité.

Le temps d'exécution d'un programme dépend de plusieurs facteurs : de la représentation des données en mémoire, proprement dit de la structure de données représentant le problème, il dépend aussi de la puissance du processeur exécutant les instructions du programme. Un autre facteur d'importance décisif est la complexité algorithmique du programme qui mesure l'efficacité en termes de nombre d'itérations effectuées pour résoudre le problème.

L'efficacité de l'exécution mesurée en fonction de l'utilisation des ressources de l'ordinateur :

- Le temps de calcul pour exécuter les opérations composants les instructions.
- La taille mémoire nécessaire pour contenir et manipuler les programmes ainsi que leurs structures de données.

En programmation structurée et séquentielle, il s'agit de trouver le meilleur compromis entre l'utilisation de ces deux ressources principales, temps d'exécution et taille mémoire.

L'analyse de la complexité a pour objectif de quantifier les deux grandeurs physiques citées ci-dessus : temps d'exécution, et taille mémoire dans le but de comparer différents techniques de résolutions algorithmiques qui résolvent un problème *P* donnée, indépendamment de l'environnement de programmation. La grandeur taille mémoire nécessaire pour les données ne représente pas un défi dans l'analyse de la complexité, surtout avec les progrès technologiques d'industrialisation des mémoires. Alors le seul défi en termes d'analyse de complexité est le temps d'exécution d'un algorithme.

Souvent la phase d'analyse de la complexité n'est pas aussi importante, pour des problèmes simples où une résolution mathématique a été prouvée. Par exemple trouver la complexité d'un algorithme calculant le plus grand commun diviseur de deux nombres n'a pas de sens, dans le sens où l'algorithme d'Euclide est sur place. Alors cette phase d'analyse de la complexité s'impose pour des problèmes où, l'espace de solutions est très grande, peu d'informations heuristiques pour décider comment déplacer et aucune information sur la solution optimale à quoi se ressemble. Enfin le calcul de la complexité est utile pour des problèmes où une résolution métaheuristique à laquelle on ne peut échapper.

2. Complexité algorithmique

2.1 Notion d'algorithme et Complexité

La première définition de la notion d'algorithme remonte au IXème siècle ap. J.-C., Al Khuwarizmi, un mathématicien perse, publie un ouvrage consacré aux algorithmes : l'étymologie du terme « algorithme» vient du nom de ce mathématicien. Selon AlKuwarizmi Un algorithme est composé d'un nombre fini d'étapes, chaque étape est composée d'un ou de plusieurs opérations.

Un algorithme est une méthode précise utilisable par ordinateur pour déterminer la solution d'un problème *P*, à condition que l'algorithme soit ***correct*** et qu'il ***converge***.

Différents algorithmes ont des coûts différents en termes de :

- Temps d'exécution (nombre d'opérations effectuées par l'algorithme).
- Taille mémoire (Taille nécessaire pour stocker les différentes structures de données pour l'exécution).

Ces deux concepts sont appelés la complexité en temps et en espace de l'algorithme.

La complexité algorithmique est un concept fondamental pour tout informaticien, elle permet de déterminer si un algorithme *A* et meilleur qu'un algorithme *B* et s'il est **optimal** ou s'il ne doit pas être utilisé.

2.2 Analyse d'un algorithme

L'analyse d'un algorithme est l'ensemble des techniques qui déterminent le temps d'exécution d'un algorithme (CPU time) et espace mémoire nécessaire pour stocker ces données.

Généralement le temps d'exécution noté t_e dépend de :

- La taille mémoire pour stocker la structure de données.
- La taille de code de l'algorithme.
- Type d'ordinateur utilisé, c'est-à-dire le processeur.
- La complexité en temps de l'algorithme

Soit n la taille de données concernant un problème P et soit T(n)n le temps d'exécution de l'algorithme résolvant le problème, on distingue :

- Le temps au plus mouvait cas noté $T_{max}(n)$ qui correspond au temps maximum pris par l'algorithme pour le problème *P* ayant des données de taille n.
- Le temps moyen noté T_{moy} , qui représente le temps moyen d'exécution sur des données de taille n.

2.3 Quelques Propriétés

- Le temps d'exécution t_e d'une affectation ou d'un test est considéré comme constant, noté c.
- Le temps d'une séquence d'instructions est la somme des temps t_e des instructions qui la composent.
- Le temps d'un branchement conditionnel est égal au temps t_e du test plus le maximum des deux temps t_e correspondant aux deux alternatives.
- Le temps associé à une boucle est égal à la somme du temps de test plus le temps de corps de la boucle.

Exemple

Soit l'algorithme suivant qui implémente la somme :

$$s = \sum_{i=1}^{n} i^2$$

Algorithme 1 Somme

1. $s \leftarrow 0$;
2. ***Pour*** $(i = 1, .., n)$ ***Faire***
3. $s \leftarrow s + i * i$;
4. ***FinPour***

Soit c_1, c_2, c_3, c_4 et c_5 sont les temps associés aux opérations d'affectation, d'incrémentation, de test, d'addition et de multiplication respectivement. Le temps maximum pris par l'algorithme est calculé comme suit :

$$T_{max(n)} = c_1 + c_1 + n(c_1 + c_2 + c_3 + c_4 + c_5)$$
$$= 2c_1 + n\sum_1^5 c_i \approx n * C$$

Le temps maximum est asymptotiquement ayant le comportement d'équation $n \times C$,

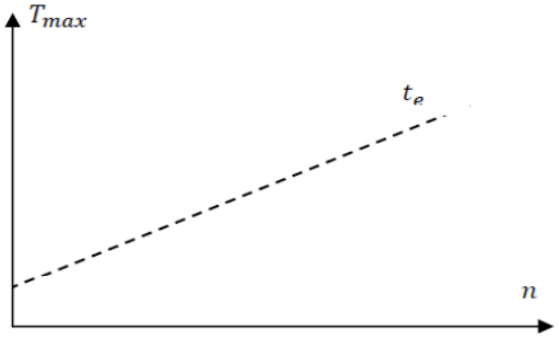


Fig. 3 forme asymptotique du temps.

3. Notation Asymptotique

C'est l'ordre de magnitude de la fréquence de l'exécution d'une instruction, noté $f(n) = O(g(n))$, en d'autre terme combien de fois l'instruction est exécuté. Mathématiquement la notation asymptotique est définie comme suit :

Soit f et g deux fonctions de $\mathbb{N}$ dans $\mathbb{R}^{*+}$, $f(n) = O(g(n))$ ou f est en grand O de g si et seulement si, $\exists\, c \in \mathbb{R}^{*+},\ \exists\, n_0 \,/\, \forall\, n > n_0,\ f(n) \leq c * g(n)$.

Exemple

Soit le polynôme :

$$f(n) = a_m n^m + a_{m-1} n^{m-1} + \cdots + a_1 n + a_0$$

La complexité de polynôme est de n^m, c'est-à-dire que $f(n) \approx O(n^m)$, en effet,

$$f(n) \leq |a_m| n^m + |a_{m-1}| n^{m-1} + \cdots + |a_1| n + |a_0|,$$
avec $n > 0$ on a
$$f(n) \leq n^m (|a_m| + \frac{|a_{m-1}|}{n} + \cdots + \frac{|a_1|}{n^{m-1}} + \frac{|a_0|}{n^m})$$
$$\leq n^m (|a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|)$$
$$\leq C n^m$$

C une constante égal à $|a_m| + |a_{m-1}| + \cdots + |a_1| + |a_0|$, donc la complexité de polynôme $f(n)$ est n^m.

3.1 Propriétés de la Notation O

- Les constantes ne sont pas importantes.
- Les termes d'ordres inferieurs sont négligeables, en effets

$$Si \lim_{n\to\infty} \frac{f(n)}{g(n)} \to 0, Alors\ f(n) + g(n) \approx O(g).$$

- Si $T_1(n) = O(g_1),\ T_2(n) = O(g_2),\ Alors$
  $T_1(n) + T_2(n) = O(g_1 + g_2)$ et $T_1(n) * T_2(n) = O(g_1 * g_2)$.

Les principales complexités sont :

- $O(1)$: Complexité à temps constant,
- $O(\log_2(n))$: Complexité logarithmique,
- $O(n)$: Complexité linéaire,
- $O(n \log_2(n))$: Complexité onn log (n),
- $O(n^2),\ O(n^3),\ O(n^k)$: Complexité polynomiale,
- $O(2^n)$: Complexité exponentielle (temps infini),

La figure suivante montre l'allure des différentes fonctions asymptotiques :

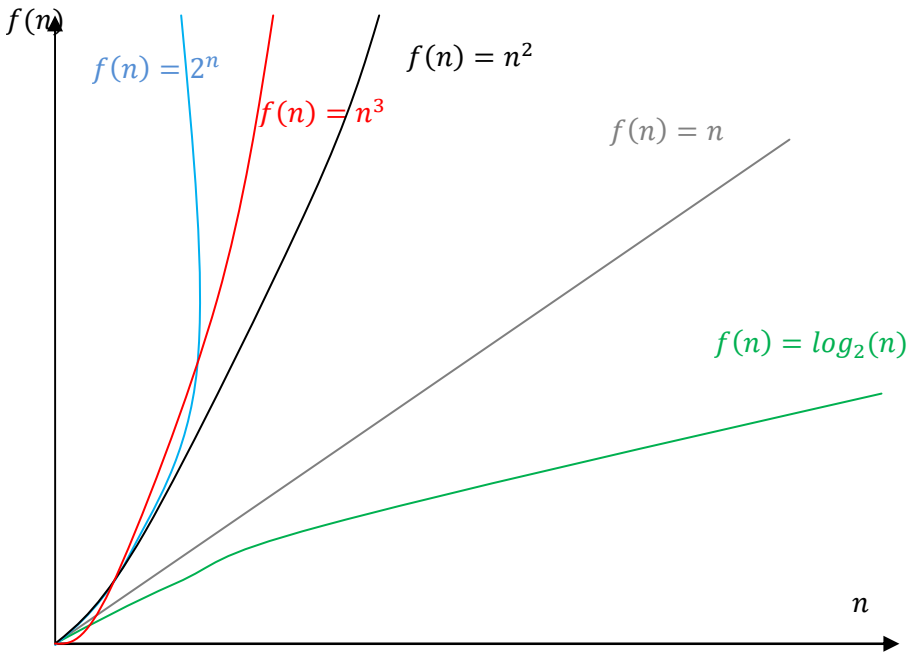


Fig. 4 Quelques fonctions asymptotique.

Exemple

Soit l'algorithme **Algorithme 1**

Algorithme 2 fact(n :entier)

1. $s \leftarrow 0$;
2. ***Si*** $n=0$ ***Alors*** $fact \leftarrow 1$
3. ***Sinon*** $fact \leftarrow n * fact(n-1)$

On suppose que le coût de test est c_1, c_2 le coût de l'opération de multiplication le temps d'affectation est supposé nul, alors le temps est exprimé pour la fonction récursive comme suit :

$$T(0) = c_1$$
$$T(n) = c_1 + c_2 + T(n-1)$$
$$= n(c_1 + c_2) + c_1$$

Alors la complexité de l'algorithme Algorithme 1 est de O(n).

Le tableau suivant nous montre une comparaison entre les différentes fonctions en termes des valeurs :

	2	**16**	**64**	**256**
$\boldsymbol{loglog\, n}$	0	2	2.58	3
$\boldsymbol{\log n}$	1	4	6	8
$\boldsymbol{n}$	2	16	64	256
$\boldsymbol{nlog\, n}$	2	64	384	2048
$\boldsymbol{n^2}$	4	256	4096	65536
$\boldsymbol{2^n}$	4	65536	1.84467 e+19	1.15792 e+77
$\boldsymbol{n!}$	2	2.0923 e +13	1.26887 e+89	8.57539 e+506 !!!

4. Classe des Problèmes

4.1 Classe des Problèmes

Il existe deux classes des problèmes :

Classe P : la classe des problèmes *P* est la classe de tous les problèmes qui peuvent être résolu par un algorithme déterministe en temps polynomial.

Classe NP : la classe des problèmes *NP* est la classe de tous les problèmes qui peuvent être résolu par un algorithme non déterministe dans un temps polynomial.

Algorithme déterministe : dans un algorithme déterministe chaque étape de l'algorithme n'a qu'un seul choix pour aller à l'étape suivante.

Algorithme non déterministe : dans un algorithme non déterministe à chaque étape, l'algorithme a plusieurs choix pour aller à l'étape suivante. L'algorithme doit deviner le meilleur choix. Pour cela en peut dire que la classe des problème IP est inclus dans la classe des problèmes *NP* et en note P $\subset$ NP.

Exemple

1. Recherche d'un élément x dans une liste $A[j],\ j = 1,..,n$

- E_1: $A[1]$, en test si $A[1] = x$ alors **succès**, sinon Aller à E_2,
- E_2: $A[2]$, en test si $A[2] = x$ alors **succès**, sinon Aller à E_3,
- ….
- E_n: $A[n]$, en test si $A[n] = x$ alors succès, sinon échec.
- C'est un algorithme déterministe de complexité $O(n)$.

2. L'algorithme non déterministe consiste à prendre une variable aléatoire j, $j \in [1, n]$ et en test :

Si $A[j] = x$ alors **succès**, sinon **échec**.

L'algorithme est non déterministe de complexité $O(1)$.

4.2 NP difficile et NP complet

Définition 1

Un problème est NP-Complet est résolu en un temps polynomial si et seulement si tous les problèmes NP-complet pouvant être résolu dans un temps polynomial.

Définition 2

Si un problème NP-difficile peut être résolu dans un temps polynomial alors tous les problèmes NP-complet peuvent être résolu dans un temps polynomial.

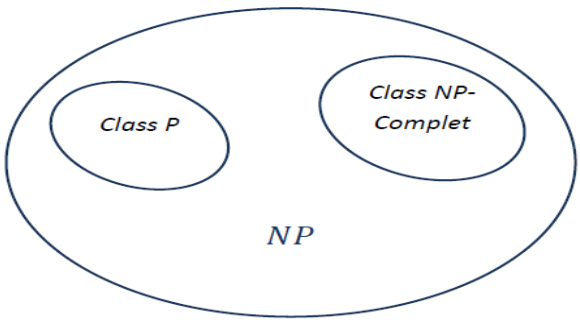


Fig. 5 Classes des Problèmes

Exemple

Soit E l'ensemble de n éléments tel que $E = \{a_1, a_2, .., a_n\}$,

Ecrire un algorithme non déterministe (NP) de complexité O(n) qui détermine s'il existe ou pas un sous ensemble s, $S \subset E$ telque

$$\sum_{a_i \in S} a_i = M$$

Solution

Soit $X = \{x_1, x_2, .., x_n\}$ à chaque liste en associe une variable X, avec X un vecteur booléen tel que :

$$x_i = \begin{cases} 1 \; si \; a_i \in S \\ 0 \quad sinon \end{cases}$$

L'algorithme non déterministe est le suivant

Algorithme 3 Sac non déterministe

1. $som \leftarrow 0$;
2. ***Pour*** $(i = 1, .., n)$ ***Faire***
3. $som \leftarrow som + a_i X_i$
4. ***Si*** $som = M$ ***Alors*** *Succès*
5. ***Sinon*** *Echec*

5. Problèmes de Décision

Définition

Un problème de décision est un problème dont la solution est formulé en termes de **Oui / Non.**

Exemple

- Etant donné un graphe $G(X, E)$, existe-t-il un chemin de longueur inférieur ou égal à L.
- Etant donné un graphe $G(X, E)$, les sommets de X peuvent-il être colorés par au plus K couleurs de telle manière que les sommets adjacents soient de couleurs différents.
- Soit $s = \{a_1, a_2, .., a_n\}$ un ensemble de n entiers, existe-t-il $R \subset S$ tel que : $\sum_{i \in R} a_i = \sum_{i \in S-R} a_i$ (Problème de sac à dos).

6. Problèmes de Satisfiabilité (SAT)

Soit $F(x_1, x_2, .., x_n)$ une expression logique de n variables. Le problème SAT consiste à trouver des valeurs vrai ou faux (1,0) pour chacune des variables x_i de telle manière à rendre vrai l'expression $F(x_1, x_2, .., x_n)$.

Un algorithme non déterministe résolvant SAT peut être écrit comme suit :

Algorithme 4 SAT non déterministe

1. ***Pour*** $(i = 1, .., n)$ ***Faire***
2. $X_i \leftarrow aléa(0,1)$;
3. ***Si*** $F(X_1, X_2, .., X_k) = 1$ ***Alors*** F *est satisfiable*
4. ***Sinon*** F *n'est pas satisfiable*

S.Cook [1971] a formulé le théorème suivant théorème :

Théorème

La satisfiabilité est dans P si et seulement si P=NP

Cook à donner une démonstration direct que la satisfiabilité est un problème NP-Complet. La théorie de NP-Complitude concerne la reconnaissance des problèmes les plus durs de la classe NP. Le concept central relié à la définition de la NP-Complétude est celui de la réduction entre problème. Cette réduction est en temps polynomial.

7. Réduction des Problèmes

Soit deux problèmes P_1 et P_2, P_1 est réduit à P_2 écrit $P_1 \propto P_2$ si chaque algorithme qui résout P_1 capable de résoudre P_2. Selon la démonstration de S. Cook tous les problèmes de la classe NP sont réductibles au problème de satisfiabilité.

Exemple

- Considérons P_1 le problème de voyageur de commerce, soit un ensemble de ville E est un ensemble de distance D entre villes. La question est de trouver un tour contenant les villes dont la distance est inférieure ou égale à M.
- Considérons P_2 le problème de cycle hamiltonien. Soit un graphe $G(X, E)$ avec Xensemble des sommets et E l'ensemble des arcs. La question est de trouver un cycle simple qui contient toutes les sommets du graphe. Alors en peut dire que le problème P_1 est réduit à P_2 c'est-à-dire $P_1 \propto P_2$.

8. Exercices

Exercice N°1

Calculer la complexité des algorithmes qui implémente le produit matriciels et le produit de deux polynômes :

- $C_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj},\ i,j=1,n$

- $P = \sum_{i=0}^{m} a_i x^i, \qquad Q = \sum_{i=0}^{n} b_i x^i$

Exercice N°2

Soit l'algorithme de tri par sélection :

```
Début
   Min, t : entier
   Pour (i = 0, i < n-1) faire
            Min ←i ;
            Pour (j = i+1, j < n) Faire
               Si ( a[j] < a[min] ) Alors
                     Min ←j ;
               t ← a[min]; a[min] ← a[i];
               a[i] ← t;
            Fin Pour
   Fin Pour
```

Calculer la complexité de l'algorithme ?

Exercice 3

Quelle est la classe de complexité de la fonction f(n) suivante (démontrez-le)

$$f(n) = 2 \times n^3 + 4 \times n^2 + 2^3$$

Exercice 4

Quelle est la complexité de la fonction *fct()* suivante :

```
Début
   Pour  i= 1,  n ∗ n Faire
     u ←i,
     Tantque (u > 1) Faire
        u ← u/2,
     Fin Tantque
```

Fin Pour
Fin

Exercice 5

Ecrire un algorithme qui prend un tableau d'entiers et calcule les valeurs minimale et maximale qu'il contient. Evaluez la complexité de votre algorithme.

Exercice 6

Calculer la complexité de la fonction qui calcule x^n à partir de la formule suivante :

$$x^n = \begin{cases} (x^2)^{n/2} & si\ n\ est\ pair \\ x * x^{n-1} & si\ n\ est\ impair \end{cases}$$

En utilisant ce principe, quel est le coût de l'élévation d'une matrice à la puissance n.

Exercice 7

Calcul de l'élément majoritaire d'un tableau.

Etant donné un ensemble E de n éléments, on veut savoir s'il existe un élément majoritaire et quel est-il (un élément majoritaire apparaître plus d'une fois sur deux, i.e. si k est son nombre d'apparitions, 2*k > n).

Donnez une méthode naïve pour ce calcul. Quelle est sa complexité ?

Une autre méthode est divisée pour régner. On considère maintenant l'approche diviser pour régner suivante : pour calculer l'élément majoritaire dans l'ensemble E s'il existe, on réparti les n éléments de E dans deux ensembles de mêmes tailles E1 et E2, et on calcule (récursivement) dans chacune de ces parties l'élément majoritaire. Pour que e soit majoritaire dans E il suffit que e soit majoritaire dans E1 et dans E2, ou que e soit majoritaire dans E1 et non dans E2 (ou inversement), mais qu'il apparaisse suffisamment dans E2.

Ecrivez une procédure de calcul correspondant à cette idée. Quelle est sa complexité ?

Chapitre 3

Chapitre 3
Heuristiques basées solution

Chapitre 3
Heuristiques basées solution

1. Introduction

L'Optimisation par la méthode du gradient est basée sur l'hypothèse que la fonction à optimiser est dérivable, c'est-à-dire facile à calculer la première dérivée et même le deuxième dérivé. C'est une grosse supposition. Mais dans la plupart des cas, vous ne pouvez pas calculer la pente de la fonction, parce que vous ne savez pas quelle est la fonction. Le problème dont vous cherchez à résoudre possède plusieurs paramètres, plusieurs contraints, et dans la plus part des cas multi-objective.

Par exemple le calcul d'un chemin minimal passant par tous les sommets du graphe, avec la contrainte qu'un sommet ne sera visité qu'une seule fois, alors le seul moyen dont vous disposez est une simple matrice de distance entre villes. La fonction d'évaluation est la somme des distances entre les villes composantes du chemin. Tout ce que vous avez est une façon de créer ou modifier les entrées de la fonction, de les tester et d'évaluer leur qualité.

Dans des situations pareils S. Luke (S. luke, 2010) propose les quatre étapes suivantes comme démarche de résolution :

- Initialiser une ou plusieurs solutions candidats. Ceci est connu en tant que procédure d'initialisation.
- Évaluer la qualité d'une solution. Ceci est connu en tant que procédure d'évaluation.
- Créer une copie de la solution courante.

- Modifier la copie de la solution courante en produisant une autre légèrement différente. Ceci est connu sous le nom de la procédure de modification ou fonction de voisinage.

Enfin selon le principe de métaheuristique utilisée on décidera des solutions à retenir dans l'itération suivante.

En général, l'efficacité des méthodes de recherche locale simples (descente, ou plus grande descente) est très peu satisfaisante. D'abord, par définition, la recherche s'arrête au premier minimum local rencontré, c'est là leur principal défaut. Pour améliorer les résultats, on peut lancer plusieurs fois l'algorithme en partant d'un jeu de solutions initiales différentes, mais la performance de cette technique décroît rapidement.

En revanche, autoriser de temps à autre une certaine dégradation des solutions trouvées, afin de mieux explorer tout l'espace des configurations, a conduit au développement des deux méthodes que nous allons exposer dans les sections suivantes de ce chapitre, le recuit simulé et la méthode de recherche taboue.

Il est important de remarquer également l'importance du choix de la fonction de voisinage N : un minimum local pour une certaine structure de voisinage ne l'est pas forcément pour une autre. C'est d'ailleurs ce constat qui est à l'origine de la méthode dite de recherche par voisinage variable, qui repose sur la construction de solutions s parmi plusieurs voisinages Ni, plutôt que dans un seul. On peut décider soit d'examiner toutes les solutions du voisinage et prendre la meilleure de toutes (ou prendre la première trouvée), soit d'examiner un sous-ensemble du voisinage.

2. Méthode de la Descente

L'**algorithme du gradient** est un algorithme d'optimisation destiné à minimiser une fonction définie sur un espace euclidien. L'algorithme est itératif et procède donc par améliorations successives. Un déplacement est effectué dans la direction *opposée* au gradient, de manière à faire décroître la fonction. (http://fr.wikipedia.org/wiki/Direction_de_descente)

Par définition une **direction de la descente** est une direction le long de laquelle la fonction à minimiser a une dérivée directionnelle strictement négative. Ces directions sont utilisées par les méthodes à directions de descente. C'est le long de ces directions qu'un déplacement est effectué afin de trouver la solution suivante. Les directions de descente peuvent être calculées par de nombreuses techniques.

Mathématiquement, une direction *de descente* de f en x est un vecteur $d \in E$ où E est un espace euclidien tel que

$$f'(x;d) < 0$$

Où

$$f'(x;d) = \lim_{t \to 0} \frac{f(x+\alpha d) - f(x)}{t}$$

On en déduit pour un cas de minimisation

$$\forall\, \alpha > 0\ , f(x + \alpha d) < f(x)\ ,$$

α suffisament petit. Alors si bien que f décroît en x dans la direction d

Dans un espace hilbertien E de dimension infinie et selon le théorème de Riesz-Fréchet il existe un vecteur $\nabla f(x) \in E$ appelé le gradient de f en *x,* définie par :

$$\forall\, d \in E, \qquad f'(x).d = < \nabla f(x), d >$$

La direction du gradient est l'opposé du gradient, alors

$$d = -\nabla f(x).$$

En résumé la méthode de la ***descente de gradient***. Consiste à identifier la pente et d'essayer de déplacer selon cette pente. Supposons que nous allons minimiser la fonction *f (x).*

La technique est très simple. Nous commençons avec une valeur arbitraire utilisée pour *x*. Nous ajoutons ensuite d'une façon répétée une petite valeur de sa pente, qui est, $x \leftarrow x + \alpha f(x)$, où α est une très petite valeur positive.

Si la pente est positive, *x* augmente. Si la pente est négative, *x* diminuer. La figure Fig. 5, illustre à peu près cela. En fin de compte les déplacements en *x* permettent de ramener les valeurs de la fonction jusqu'à ce qu'il soit au sommet, point auquel la pente est nulle et le *x* ne changera plus.

Dans le cas pratique nous sommes confrontés à des situations où une fonction continue à une seule variable est insuffisant pour représenter l'objectif cherché Autrement dit nous aurons aucun intérêt à développer un algorithme pour trouver le minimum d'une telle fonction, un simple calcul mathématique permet de trouver la solution optimale. Dans le cas général, nous aimerions trouver la maximum d'une fonction à plusieurs dimensions. Pour le faire, nous remplaçons la variable *x* par le vecteur *x*, et remplaçant la pente de *f (x)* par le gradient de *x*, $\nabla f(x)$. Pour rappel: le gradient est simplement une vecteur où chaque élément correspond à la pente de *x* dans cette dimension, c'est-à-dire $<\partial f/\partial x_1, \partial f/\partial x_2, ..., \partial f/\partial x_n>$.

Donc, nous allons déplacer selon la direction de la pente dans toutes les dimensions à la fois. Voici l'algorithme de la descente de gradient :

Algorithme 5 Descente de Gradient

1. $x \leftarrow$ *solution initiale généré aléatoire,*
2. ***Tantque*** (x est la meilleure) **ou** ($Nombre\ Iter. = \max$) ***Faire***
3. $x \leftarrow x - \alpha f(x)$; $x \leftarrow x - \alpha \nabla f(x)$ *pour le cas plusieurs dimensions.*
4. ***Fin Tantque***

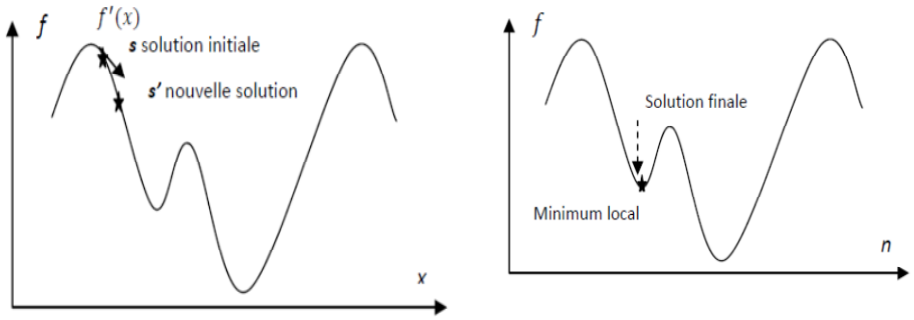


Fig.5 Exemple de la descente de gradient

3. Hill Climbing Méthode

La méthode de la descente est appelé dans la littérature aussi la méthode de recherche locale ou *Hill Climbing*. Cette technique est liée à la descente de gradient. Mais cela ne nécessite pas à connaître la force du gradient ni même sa direction: il vous suffit itérativement tester de nouvelles solutions potentielles dans la région de la solution courante, et de prendre la meilleure dans le voisinage.

La méthode est une généralisation de la méthode de la descente de gradient elle consiste à partir d'une solution *s* à choisir une solution *s'* dans un voisinage de s, noté $N(s)$. La nouvelle solution choisie est meilleure que la précédente sous la fonction objective. Cela nous permet d'explorer l'espace d'une manière itérative jusqu'à convergence à un optimum local.

Algorithme 6 Hill Climbing

1. $s \leftarrow$ *solution initiale,*
2. ***Tantque*** (x est la meilleure) **ou** ($Nombre\ Iter. = \max$) ***Faire***
3. $r \leftarrow \boldsymbol{s'}$ *où* $\boldsymbol{s'} \in N(s)$
4. ***Si*** $f(r) < f(s)$ ***Alors***
5. $s \leftarrow r$
6. ***Fin Tantque***

Nous pouvons remarquez directement les ressemblances entre l'algorithme Hill-Climbing et la descente de gradient. La véritable différence est que la fonction de voisinage est plus générale dans Hill-Climbing et elle doit s'appuyer sur une approche de recherche stochastique (partiellement aléatoire) des meilleurs voisins de la solution courante.

Parfois, des mauvaises voisines seront sélectionnées (cas où tous les voisins sont faibles par apport à la solution courante). L'algorithme développé dans la section précédente peut être modifié on introduisant la notion de fonction de voisinage N(s), où s est la solution courante, puis sélectionner la meilleurs solution voisins qui va participer dans l'itération qui suive. Cet algorithme est connu sous le nom de ***Hill Climbing*** avec la grande descente.

Algorithme 7 Hill Climbing Grande Ascent

1. $n \leftarrow$ *nombre voisins servant pour gradient*
2. $s \leftarrow$ *solution initiale,*
3. ***Tantque*** (x est la meilleure) **ou** ($Nombre\ Iter. = \max$) ***Faire***
4. $r \leftarrow \boldsymbol{s}' \text{ où } \boldsymbol{s}' \in N(s)$
5. ***Pour*** $n-1\ fois$ ***Faire***
6. $r' \leftarrow \boldsymbol{s}' \text{ où } \boldsymbol{s}' \in N(s)$
7. ***Si*** $f(r') < f(r)$ ***Alors***
8. $r \leftarrow r'$
9. ***Fin Pour***
10. ***Si*** $f(r) < f(s)$ ***Alors***
11. $s \leftarrow r$
12. ***Fin Tantque***

Une variante de l'algorithme très populaire, est de modifier l'algorithme précédent par remplacement. La modification consiste à ne pas comparer *r* à *s,* à la place, nous remplaçons simplement *s* directement avec *r*. Bien sûr, cela risque de perdre la meilleure solution, mais nous tenons l'algorithme à garder le plus longtemps possible la solution découverte cachée, dans une variable appelé *Best*. A la fin de la course, nous revenons à la meilleure. L'algorithme Hill Climbing avec remplacement est le suivant :

Algorithme 8 Hill Climbing Grande Ascent avec Replacement

1. $n \leftarrow$ *nombre voisins servant pour gradient*
2. $s \leftarrow$ *solution initiale,*
3. $Best \leftarrow s$
4. ***Tantque*** (x est la meilleure) **ou** ($Nombre\ Iter. = \max$) ***Faire***
5. $r \leftarrow \boldsymbol{s}' \text{ où } \boldsymbol{s}' \in N(s)$
6. ***Pour*** $n-1\ fois$ ***Faire***
7. $r' \leftarrow \boldsymbol{s}' \text{ où } \boldsymbol{s}' \in N(s)$
8. ***Si*** $f(r') < f(r)$ ***Alors***
9. $r \leftarrow r'$
10. ***Fin Pour***
11. $s \leftarrow r$
12. ***Si*** $f(s) < f(Best)$ ***Alors***
13. $Best \leftarrow s$
14. ***Fin Tantque***

4. Recuit Simulé

Le recuit simulé en anglais « simulated annealing » est souvent présenté comme la plus ancienne des métaheuristiques. Il s'inspire d'une procédure utilisée

depuis longtemps par les métallurgistes. 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 maîtrisant le refroidissement et en le ralentissant par un apport de chaleur externe, ou bien par une isolation. Elle a été mise au point par trois chercheurs Kirkpatrick, Gelatt et Vecchi en 1983, et indépendamment par Černy en 1985. (S. Kirkpatrick. et al. 1983) (V. Cerny, 1985)

Le recuit simulé s'appuie sur l'algorithme de Metropolis-Hastings, qui permet de décrire l'évolution d'un système thermodynamique. Par analogie avec le processus physique, la fonction à minimiser est l'énergie du système. On introduit également un paramètre fictif, la température du système.

Dans l'algorithme de Métropolis, on part d'une configuration donnée, et on fait subir au système une modification élémentaire. Si cette perturbation a pour effet de diminuer la fonction objectif (ou énergie) du système, elle est acceptée. Sinon, elle est acceptée avec une probabilité P. En appliquant itérativement cette règle, on engendre une séquence de configurations qui tendent vers l'équilibre thermodynamique.

L'algorithme de recuit simulé est différent du premier algorithme Hill Climbing dans la décision de remplacer la solution S, par une solution meilleure R. Plus précisément dans l'algorithme de recuit simulé, si R est meilleure que S, nous remplaçant la solution S par R comme d'habitude. Mais si R est pire que S, nous pouvons encore remplacer S avec R avec une probabilité, $P(t, R, S)$.

$$P(t, R, S) = e^{(f(R)-f(S))/t}$$

Algorithme 9 Recuit Simulé

1. $t \leftarrow t_0$; t_0 est la température initialiseée à une grande valeur
2. $s \leftarrow s_0$; s_0 une solution initiale
3. $best \leftarrow s$;
4. ***Tantque*** ($Best$ est la meilleure solution) & ($Nombre\ Iter. = \max$) & ($t \leq 0$) ***Faire***
5. $r \leftarrow s'$ où $s' \in N(s)$;
6. $\rho \leftarrow random(o, 1)$;
7. ***Si*** $\left(f(r) < f(s)\right)$ ***ou*** $\left(\rho < e^{\frac{f(r)-f(s)}{t}}\right)$ ***Alors***
8. $s \leftarrow r$;
9. ***Si*** $\left(f(s) < f(best)\right)$ ***Alors***
10. $best \leftarrow s$;
11. $décémenter\ t$;
12. ***Fin Tantque***

Interprétation

Cette équation est importante pour les raisons suivantes :
Notons d'abord que si la solution r est pire que s, alors la fraction $(f(r) - f(s))/t$ est négative

- **Premièrement**

Si r est bien pire que s, la fraction est plus grande, et si la probabilité est proche de 0. si r est très proche de s, la probabilité est près de 1. Ainsi, si r n'est pas plus pire que s, nous allons sélectionnez r avec une probabilité raisonnable.

- **Deuxièmement**, nous avons un paramètre température t, que l'on peut ajuster. Si la température t est proche de 0, la fraction $(f(r) - f(s))/t$ est de nouveau un grand nombre, alors la probabilité est proche de 0. Si t est très élevé, la probabilité est proche de 1.

L'idée est d'initialiser d'abord la température à un nombre élevé, ce qui donne à l'algorithme une exploration de l'espace de solutions très aléatoire. Ensuite, diminuer la température t d'une manière lente, jusqu'à ce que la température converge à zéro. Dans ce cas l'algoithme de recuit simulé converge à l'algorithme Hill-Climbing de base.

5. Méthode de Recherche Taboue

L'algorithme de Recherche taboue (Tabu Search) est une technique de recherche dont les principes ont été proposés pour la première fois par Fred Glover dans les années 80 (Glover, 1980). Elle se distingue des méthodes de recherche locale simples par le recours à un historique des solutions visitées, de façon à rendre la recherche un peu moins aléatoire. Il devient donc possible de s'extraire d'un minimum local, mais, pour éviter d'y retomber périodiquement, certaines solutions sont considérées taboues.

En effet l'originalité de la méthode est l'introduction d'une mémoire par l'intermédiaire de l'utilisation des tabous. Cela consiste à enregistrer ce qui s'est passé dans les étapes précédentes et à interdire que cela se reproduise. Le but est évidemment de favoriser une large exploration de l'espace des solutions et d'éviter de rester dans un optimum local ou d'y retourner trop rapidement.

Principe

C'est une approche facile à comprendre, le but est de conserver dans une liste taboue *L*, d'une certaine longueur *l*, les solutions récemment visitées. Chaque fois que nous choisissant une nouvelle solution, elle est insérée dans la liste taboue. Si la liste taboue est trop grande, on enlève la solution le plus âgé et elle ne sera plus taboue.

Le fait de créer une liste taboue c'est pour éviter d'être piégé et tourné ainsi en rond, et c'est pour ça qu'on crée la liste L qui mémorise les dernières solutions visitées et qui interdit tout déplacement vers une solution de cette liste. Cette liste L est appelée ***liste Tabou***, c'est la mémoire de la recherche et le savoir d'exploration de l'espace de solutions.

Mémoire

La taille de la liste taboue, dite aussi mémoire, est à déterminer empiriquement, elle varie avec les problèmes, mais c'est une donnée primordiale. En effet une liste trop petite peut conduire à un cycle, alors qu'une liste trop grande peut interdire des transformations intéressantes.

Voisinage

A l'inverse du recuit simulé qui génère de manière aléatoire une seule solution voisine $s' \in N(s)$ à chaque itération, Tabou examine un échantillonnage de solutions de $N(s)$ et retient la meilleure s' même si f(s')>f(s). La recherche Tabou ne s'arrête donc pas au premier optimum trouvé.

Algorithme tabou

Algorithme 10 Recherche Taboue

1. $l \leftarrow$ *longueur maximale de la liste taboue,*
2. $n \leftarrow$ number of tweaks desired to sample the gradient
3. $s \leftarrow$ *solution initial;*
4. $Best \leftarrow s$
5. $L \leftarrow \{\}$ *la liste taboue de longueur* l *de politique first in first out;*
6. *Insérer* s *dans* L *;*
7. ***Tantque*** ($Best$ est la meilleure) **ou** ($Nombre\ Iter. = \max$) ***Faire***
8. ***Si*** $longueur(L) > l$ ***Alors***
9. *Défiler un élément de la liste L*
10. $r \leftarrow r'$ où $r' \in N(s)$
11. **Pour** $(i \leftarrow 1, .., n-1)$ ***Faire***
12. $w \leftarrow w'$ où $w' \in N(s)$
13. ***Si*** $w \notin L\ et\ (f(w) < f(r) ou\ r \in L)$ ***Alors***
14. $r \leftarrow w$
15. ***Si*** $r \notin L\ et\ f(r) < f(s)$ ***Alors***
16. s $\leftarrow r$
17. $Enfiler\ r\ dans\ L$
18. **Si** $f(s) < f(Best)$ **Alors**
19. *Best* $\leftarrow s$
20. ***Fin Tantque***

6. Exercices

Exercice N°1 (http://www.igt.net/~ngrenon/)

En appliquant la méthode de recuit simulé pour certain problème, vous êtes arrivé à une itération où t =2 et la valeur de la fonction objectif de la solution courante est 30. Cette solution a 4 voisins dont la valeur de la fonction objective donne 29, 34, 31 et 24. Pour chacune de ces solutions vous voulez déterminer la probabilité qu'elle soit choisie pour être la prochaine solution courante.

(a) Déterminer ces probabilités pour un problème de maximisation.

(b) Même question pour un problème de minimisation.

Exercice N°2

Soit le problème de coloration des arrêtes d'un graphe $G(X, E)$ avec X l'ensemble des sommets et E l'ensemble des arrêtes. L'objectif est de colorer les arrêtes du graphe tel que deux arêtes ayant une extrémité commune soient de couleurs différentes. Soit c_i la couleur de l'arrête i. Le but de l'exercice est d'appliquer la méthode de recherche taboue pour le problème.

- Donnez la représentation d'une solution du problème, montrer à l'aide d'un graphe cette représentation.
- Considérons trois couleurs c_1, c_2 et c_3 et selon votre représentation, donnez une solution s initiale. Puis déroulez l'algorithme tabou pour deux itérations.
- Pour les deux itérations, donnez le contenu de la liste taboue.

Exercice N°3

Mêmes questions de l'exercice 2 si on considère maintenant l'objectif est de trouver un chemin minimale reliant les sommets du graphe $G(X, E)$ avec X l'ensemble des sommets et E l'ensemble des arrêtes. Avec la contrainte qu'un sommet est visité une et une seule fois.

Exercice N°4 (J.-F. Scheid 2011)

Une entreprise dispose de plusieurs dépôts (Di) contenant chacun un certain nombre de containers. Différents magasins (Mj) commandent des containers. On connaît le coût de transport de chaque dépôt aux magasins. exemple,

	M1	M2	M3	Dépôt
D1	5	3	4	8
D2	6	7	2	9

Les demandes magasins sont 4, 5 et 8 containers.

Quelle est l'organisation des livraisons des containers pour minimiser le coût total de transport ?

Ce problème peut se modéliser par programmation linéaire en nombres entiers et sa résolution peut se faire par séparation et évaluation (Chercher une solution).

Proposez une solution pour le problème en se basant sur la méthode de recherche taboue.

Chapitre 4

Chapitre 4
Approche Evolutionnaire : Algorithme Génétique

Chapitre 4
Approche Evolutionnaire : Algorithme Génétique

1. Introduction

Les Algorithmes Evolutionnaires (AE) sont inspirés du concept de la sélection naturelle développé par « Charles Darwin » dans « The Origin of Species », Darwin montre que l'apparition d'espèces distinctes se fait par le biais de la sélection naturelle.

Le principe de la sélection est fondé sur la lutte pour la vie, due à une population tendant à s'étendre mais disposent d'un espace et ressources finis. Il en résulte que les individus les plus adoptés tendant à survivre plus longtemps et à se reproduire plus aisément.

Le principe d'adaptation d'une population d'individus aux conditions naturelles définis par environnement, inclut les lois de variations telles que le croisement et la mutation qui expliquent l'apparition des variations aux niveaux individuelles. Cette appariation explique bien le phénomène d'évolution et d'adaptation sans avoir besoin à une modification directe des individus. Le principe ainsi défini conduit une population à s'évoluer et s'adapter de génération en génération sans tendre vers un objectif dicté.

Différents types d'algorithmes évolutionnaires ont été développés au début des années soixante :

Stratégies d'Evolution (SE) : les (SE) ont été développés par deux ingénieurs Rechenberg et Schwefel au cours de leurs travaux de recherches sur des problèmes numériques pour l'optimisation paramétrique.

Programmation Evolutionnaire (EP) : la (PE) a été développée aux années 60 par L. Fogel dans le contexte de la découverte d'automates d'états finis pour l'approximation des séries temporelles, puis utilisé plus tard par D.B. Fogel aux années 91.

Algorithmes Génétiques (AG) : les (AG), développées par J. Holland en 75 comme outils de modélisation de l'adaptation et qui travaillent dans un espace de chaînes de bits. Les plus populaires aux chercheurs de différentes disciplines. Largement utilisées et développées par D.E. Goldberg en 1989.

Programmation Génétique (PG) : la (PG) a été développée par J. Koza en 90 dans le but d'attendre un des rêves des programmeurs : faire écrire un programme automatiquement. La représentation est basée sur des arbres représentant les programmes.

Dans la suite de ce chapitre nous nous traitons beaucoup plus en détaille plus particulièrement les algorithmes génétiques.

2. Principe d'Algorithme Génétique

2.1 Codage des Variables

L'étape clef dans un algorithme génétique est de définir et coder convenablement les variables d'un problème donnée. On retrouve différents techniques de codages.

Le codage est un processus de représentation des gènes. Le processus peut être effectué par utilisation des : bits, nombres, arbres, tableaux, listes ou tous autres objets. La littérature définit deux types de codage : binaire et réel.

2.1.1 Codage binaire

C'est la représentation la plus fréquente, soit f une fonction à optimiser de paramètres x. La variable x représente un individu de la population, il est codé sous forme d'une chaîne de n bits. Soit $x \in [x_{min}, x_{max}]$ avec $x \in R$ et x a un nombre de décimale noté d.

Dans une représentation binaire, la taille de l'individu n vérifie l'inéquation suivante :

$$|x_{max} - x_{min}| * 10^d \leq 2^n$$

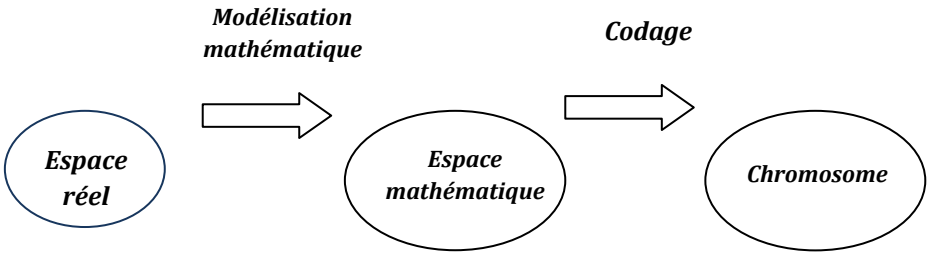


Fig. 6 Les phases de définition d'un codage

Exemple

Soit à coder les individus d'une population pour mettre en place un AG minimisant une fonction dont les solutions sont définies dans l'intervalle $x \in [-1{,}28,\ 1{,}28]$. La précision d est de 2 décimales.

- La taille nécessaire pour coder l'individu est :

$$|1{,}28 + 1{,}28| * 10^2 \leq 2^n \Longrightarrow 8 * \ln 2 \leq n * \ln 2 \Longrightarrow n = 8$$

La chaîne 00000000 permet de coder la variable réel $x = -1{,}28$.

La chaîne 11111111 pour coder la variable réel $x = 1{,}28$.

Dans ce type de codage pour chaque variable réel x de précision d correspond un entier long g telque $g_{max} = 2^n - 1$ où n est la taille de la chaine de bits (taille de code de l'individu).

g est codé sur n bits suivant la représentation suivante :

b_{n-1}	b_{n-2}	b_{n-3}	.....	b_2	b_1	b_0

b_0 est le bit de plus faible poids et b_{n-1} de plus fort poids. Alors l'entier long g, tel que $0 \leq g \leq g_{max}$, est calculé par la relation suivante :

$$g = \sum_{j=0}^{n-1} b_j 2^j \quad (2)$$

Enfin, il est facile de trouver la correspondance entre la variable réel x et l'entier long g, les relations (3) et (4) permettent le passage entre les deux représentations.

- **Réel – Entier long**

$$g = \frac{x - x_{min}}{x_{max} - x_{min}} * g_{max} \quad (3)$$

- **Entier long – Réel**

$$x = x_{min} + \frac{(x_{max} - x_{\min}) * g}{g_{max}} \quad (4)$$

Exemple

Reprenons l'exemple précédent avec $x \in [-1{,}28,\ 1{,}28], n = 8$. Alors l'entier long 250 correspond à la variable réel :

$x = -1.28 + \frac{2.56 * 250}{255} = 1.23$, soit en binaire 11111010.

2.1.2 Codage Réel

La représentation des solutions dans le cadre des AG n'est pas nécessairement réduite à un alphabet de faible cardinalité (0,1), il existe toute une école pour laquelle la représentation la plus efficace est celle qui s'appuie sur des nombres réels. Cette représentation est à la base de l'approche évolutionnaire « Evolution stratégie ». Ce type de codage présente certains avantages par apport au codage binaire :

- Le codage réel est robuste pour les problèmes considérés comme difficile pour le codage binaire.
- Ce codage nécessite une adaptation des opérateurs de croisement et mutation.

2.2 Population Initiale

Plusieurs mécanismes de génération de la population initiale sont utilisés dans la littérature. Le choix de l'initialisation se fera en fonction des connaissances que l'utilisateur a sur le problème. S'il n'a pas d'informations particulières, alors une initialisation aléatoire, la plus uniforme possible afin de favoriser une exploration de l'espace de recherche maximum, sera la plus adaptée. Par ailleurs, cette étape présente un problème principal qui est celui de choix de la taille de la population. En effet une taille de population trop grande augmente le temps de calcul et nécessite un espace mémoire considérable, alors qu'une taille de population trop petite conduit à l'obtention d'un optimum local.

2.3 Adaptation

La fonction d'adaptation, ou fitness, associe une valeur pour chaque individu. Cette valeur a pour but d'évaluer si un individu est mieux adapté qu'un autre à son environnement. Ce qui signifie qu'elle quantifie la réponse fournit au problème pour une solution potentielle données. Ainsi les individus peuvent être comparés entre eux.

Cette fonction, propre au problème, est souvent simple à formuler lorsqu'il ya peu de paramètres. Au contraire, lorsqu'il ya beaucoup de paramètres ou lorsqu'ils sont corrèles, elle est plus difficile à définir. Dans ce cas, la fonction devient une somme pondérée de plusieurs fonctions. Un ajustement des coefficients est nécessaire.

2.4 Sélection : Algorithme et Méthodes

En biologie, la **sélection naturelle** est l'un des mécanismes qui cause l'évolution des espèces. Ce mécanisme est particulièrement important du fait qu'il explique l'adaptation des espèces aux milieux au fil des générations. La théorie de la sélection naturelle permet d'expliquer et de comprendre comment l'environnement influe sur l'évolution des espèces et des populations en sélectionnant les individus les plus adaptés et elle constitue donc un aspect fondamental de la théorie de l'évolution.

La sélection s'effectue sur un ensemble d'individus appelé reproducteur qui est composé :

- Soit de l'ensemble de la population.
- Soit d'un sous ensemble construit par sélection : Seuls les individus les mieux adaptés (dont les évaluations sont les plus fortes) vont « survivre », les autres sont éliminés. On fixe généralement à ½ la part de la population qui survie, dans ce cas le tri de la population selon la fonction objective est obligatoire.
- Soit d'un sous ensemble construit par seuillage : Tous les individus qui possèdent une évaluation en dessus d'un seuil prédéfini survivent. S'il n'y en a aucun, une nouvelle population aléatoire est créée, ce qui ne nécessite pas le tri de la population selon la fonction objective.

On trouve **essentiellement** quatre types de méthodes de sélection différentes :

- La méthode de la "loterie biaisée" (roulette wheel) de Goldberg,
- La méthode "élitiste",
- La sélection par tournois,
- La sélection universelle stochastique.

2.4.1 Méthode de la roulette

Est la plus célèbre des sélections stochastiques. Supposant un problème de maximisation avec uniquement des performances positives, elle consiste à donner à chaque individu une probabilité d'être sélectionné proportionnelle à sa performance. Alors Un individu avec une forte évaluation a une grande chance d'être sélectionné alors qu'un individu avec un plus faible coût en a moins.

Si la population d'individus est de taille égale à N, alors la probabilité de sélection d'un individu x_i notée $p(x_i)$ est égale à : $p(x_i) = f(x_i)/\sum_{i=1}^{n} f(x_i)$ en

pratique, on calcul pour chaque individu x_i sa probabilité cumulée $q_i = \sum_{j=1}^{i} p(x_j)$ et on choisie aléatoirement un nombre r compris entre 0 et 1.

L'individu retenu est x_1 si $q_1 \geq r$ ou x_i $(2 \leq i \leq N)$ si $q_{i-1} \prec r \leq q_i$. Ce processus est répété N fois. Avec une telle sélection, un individu fort peut être choisi plusieurs fois. Par contre, un individu faible aura moins de chance d'être sélectionné. Une illustration de la roulette est donnée dans l'exemple suivant.

Exemple

L'exemple est dû à Goldberg (1989). Il consiste à trouver le maximum de la fonction f(x) = x sur l'intervalle [0, 31] où x est un entier naturel.

Ind. (x_i)	***Code***	***Adaptation*** $(f(x) = x)$	P_i
1	01011	11	0.186
2	10011	19	0.322
3	00101	5	0.85
4	11000	24	0.407
Total		59	1

Une sélection par la méthode la loterie biaisée nous donne la roue suivante :

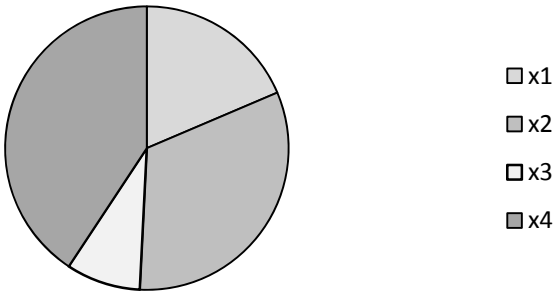


Fig. 7 Représentation à l'aide de la roue de loterie.

Un tournement de la roue quatre fois de suite nous donne la population suivante

Ind. (x_i)	***Code***
1	11000
2	00101
3	11000
4	01011

Algorithme de sélection par roulette

Algorithme 11 Select Roulette

1. $P \leftarrow$ Vecteur d'individus $< P_1, P_2, .., P_n >$ n est la taille de la population
2. $f \leftarrow < f_1, f_2, .., f_n >$ vecteur d'évaluation
3. ***Pour*** $(i = 2, .., n)$ ***Faire*** $f_i \leftarrow f_{i-1} + f_i$;
4. ***Répéter***
5. $\rho \leftarrow$ random$(0, f_n)$
6. ***Pour*** $(i = 2, .., n)$ ***Faire***
7. ***Si*** $f_{i-1} < \rho \leq f_i$ ***Alors*** Retourne P_i
8. ***Sinon*** Retourne P_1
9. ***FinPour***
10. ***Jusqu'à*** *(nombre d'individus sélectionné atteint le Max cherché)*

2.4.2 Sélection par élitisme

Cette méthode consiste à sélectionner les n individus dont on a besoin pour la nouvelle génération P' en prenant les n meilleurs individus de la population P après l'avoir triée de manière décroissante selon la fonction d'adaptation (fitness) de ses individus. Il est inutile de préciser que cette méthode est encore pire que celle de la loterie biaisée dans le sens où elle amènera à une convergence prématurée encore plus rapidement et surtout de manière encore plus sûre que la méthode de sélection de la loterie biaisée ; en effet, la pression de la sélection est trop forte, la variance nulle et la diversité inexistante.

Exemple

Selon l'exemple précédent, une sélection par élitisme après avoir trié la population selon la fonction d'adaptation permet de nous donner la population suivante :

Ind. (x_i)	***Code***
1	11000
2	10011
3	11000
4	11000

On remarque directement que les meilleurs individus sont sélectionnés, alors c'est pour ça on dit que la pression de sélection est forte, et c'est l'inconvénient de la méthode par élitisme.

Algorithme de sélection par élitisme

Algorithme 12 Select Elitisme

1. ***Algorithme de sélection de n/2 individus, population triée {suivant l'ordre croissant des fitness.***
2. $P \leftarrow$ Vecteur d'individus $< P_1, P_2, .., P_n >$ n est la taille de la population
3. ***Pour*** $(i = \frac{n}{2} + 1, .., n)$ ***Faire***
4. *Retourne* P_i
5. ***FinPour***
6. ***Jusqu'à*** *(nombre d'individus sélectionné atteint le Max cherché)*

2.4.3 Sélection par tournoi

Cette méthode n'utilise que des comparaisons entre individus et ne nécessite pas le tri de la population. Elle possède un paramètre, taille du tournoi. Pour sélectionner un individu, on en tire t uniformément dans la population, et on sélectionne le meilleur de ces t individus.

Si $t = 2$ alors on effectue un tirage avec remise de deux individus de la population, et on les fait "combattre". Celui qui a l'adaptation la plus élevée l'emporte avec une probabilité p comprise entre 0.5 et 1. On répète ce processus n fois de manière à obtenir les n individus de la population qui serviront de parents.

Le choix de t permet de faire varier la pression sélective, c'est-à-dire les chances de sélection des plus performants par rapport aux plus faibles.

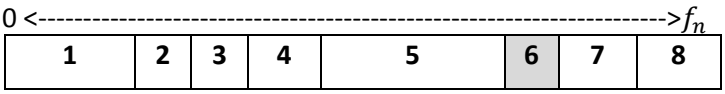
Algorithme 13 Select Tournement

1. $P \leftarrow$ Vecteur d'individus $< P_1, P_2, .., P_n >$
2. $t \leftarrow tournement\ ,\ \ t > 1$
3. *Best* ← tirage avec remise d'un individu
4. ***Pour*** $(i = 2, .., t)$ ***Faire***
5. *Next* ← tirage avec remise d'un individu
6. $\rho \leftarrow$ random(0,1)
7. ***Si*** $Adaptation\ (Next) > Adaptation\ (Best)$ ***Alors*** $Best \leftarrow Next$
8. ***FinPour***
9. ***Si*** $\rho > 0.5$ ***Alors*** $retourne\ Best$

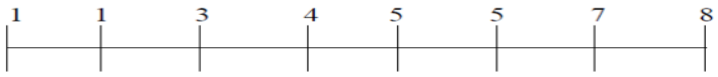
2.4.4 Sélection par rang

La sélection par rang trie d'abord la population par fitness. Chaque individu se voit associé un rang en fonction de sa position. La valeur du rang est calculée selon le total des adaptations des individus divisé par la taille de la population, autrement rang $= \sum f_i/n$, alors le plus mauvais individu aura le premier rang, le suivant le deuxième rang, ainsi de suite. La sélection par rang d'un individu est la même que par roulette, mais les proportions sont en relation avec le rang plutôt qu'avec la valeur de l'évaluation. Avec cette méthode de sélection, tous les individus ont une chance d'être sélectionnés. Cependant, elle conduit à une convergence plus lente vers la bonne solution.

Pour expliquer soit une population composée de huit (08) individus, dont l'adaptation est représentée selon la répartition représentée selon la grille suivante :



Le rang est calculé selon la relation $r = f_n/n$, où n est le nombre d'individus. Alors générer une variable aléatoire $\mu \in [0, r]$ et c'est selon cette valeur et le rang qu'on va sélectionner les individus participants à la sélection.



Les n individus sélectionnés sont 1,1,3,4,5,5,7 et 8

Algorithme 14 Select Rang

1. $P \leftarrow$ Vecteur d'individus $< P_1, P_2, .., P_n >$ n est la taille de la population
2. $f \leftarrow < f_1, f_2, .., f_n >$ vecteur d'évaluation
3. ***Pour*** $i = 2, .., n$ ***Faire*** $f_i \leftarrow f_{i-1} + f_i$;
4. $r \leftarrow random(0, \frac{f_n}{n})$
5. ***Répéter***
6. ***Tantque*** $(f_{index} < r)$ ***Faire*** $index \leftarrow index + 1$;
7. $r \leftarrow r + \frac{f_n}{n}$
8. Retourne P_{index}
9. ***Jusqu'à*** *(nombre d'individus sélectionné atteint le Max cherché)*

2.5 Croisement

La naissance d'un nouvel individu, nécessite la prise aléatoire d'une partie des gènes de chacun des deux parents. Ce phénomène, issu de la nature est appelé croisement (crossover).

Le croisement est le processus de prendre deux parents et de produire à partir d'elles des enfants. Il s'agit d'un processus essentiel pour explorer l'espace des solutions possibles.

La littérature définit plusieurs opérateurs de croisement. Ils différent selon le type de codage adapté et la nature du problème traité.

2.5.1 Croisement binaire

Ce croisement peut avoir recourt à plusieurs types en occurrence :

Croisement en 1-point

C'est le croisement le plus simple et le plus connu dans la littérature. Il consiste à choisir au hasard un point de croisement pour chaque couple de chromosomes. Les sous-chaînes situées après ce point sont par la suite inter-changées pour former les fils, (Fig. 8).

Parent1	**1 1 0 1 0 0 0 1 1 0 1 0**
Parent2	1 0 0 1 1 0 0 1 1 0 1 1

Fils 1	**1 1 0 1 0 0** 0 1 1 0 1 1
Fils 2	1 0 0 1 1 0 **0 1 1 0 1 0**

Fig. 8 Croisement en un point de deux chromosomes

Croisement en 2-points

Dans ce type de croisement, deux points de coupure sont choisis au hasard et le contenu entre ces points est inter-changé pour former les fils, Fig. 9.

Parent1	**1 1 0 1** **0 0 0 1** **1 0 1 0**
Parent2	1 0 0 1 1 0 0 1 1 0 1 1

Fils 1	**1 1 0 1** 1 0 0 1 **1 0 1 0**
Fils 2	1 0 0 1 **0 0 0 1** 1 0 1 1

Fig. 9 Croisement en 2-points de deux chromosomes

Croisement en n-points

Ce type de croisement s'énonce par un choix aléatoirement de n-points de coupure pour dissocier chaque parent en n+1 fragments. Pour former un fils, il suffit de concaténer alternativement n+1 sous chaînes à partir des deux parents.

Croisement uniforme

Il existe des versions distinctes de ce croisement. La plus connue consiste à définir de manière aléatoirement un masque, c'est-à-dire une chaîne de bits de même longueur que les chromosomes des parents sur lesquels il sera appliqué. Ce masque est destiné à savoir, pour chaque locus, de quel parent le premier fils devra hériter du gène s'y trouvant; si face à un locus le masque présente un 1, le fils héritera le gène s'y trouvant du parent1, s'il présente un 0 il en héritera du parent2.

La création du fils2 se fait de manière symétrique: si pour un gène donné le masque indique que le fils1 devra recevoir celui-ci du parent1 alors le fils2 le recevra du parent2, et si le fils1 le reçoit du parent2 alors le fils 2 le recevra du parent1, comme le montre la figure **Fig. 10.**

Parent1	**1 1 0 1 1 0 0 1 1 0 1 0**
Parent2	0 1 0 0 1 1 1 1 0 0 1 1
Masque	**1 1 0 0 1 1 0 0 1 0 0 0**

***Fils* 1**	**1 1** 0 0 **1 0** 1 1 **1** 0 1 1
Fils 2	0 1 **0 1** 1 1 **0 1** 0 **0 1 0**

Fig. 10 Croisement uniforme

Croisement avec trois parents

Dans cette technique, trois parents sont aléatoirement choisis. Chaque bit du premier parent est comparé avec le bit du deuxième parent. Si tous les deux sont les mêmes le bit est pris pour le résultat, sinon le bit du troisième parent est pris pour le résultat (figure Fig. 11).

Parent1	**1 1 0 1 1 0 0 1 1 0 1 0**
Parent2	**0 1 0 0 1 1 1 1 0 0 1 1**
Parent3	1 1 0 0 1 1 0 0 1 0 0 0

Fils	1 **1 0** 0 **1** 1 0 **1** 1 **0 1** 1

Fig. Croisement avec trois parents

Fig. 11 Croisement trois parents

2.5.2 Croisement réel

Le codage réel requiert des opérateurs génétiques spécifiques pour la manipulation des chromosomes. Il est de plusieurs types :

L'opérateur de Croisement PMX (Partially Mapped Crossover)

PMX est un opérateur de croisement à deux points de coupure qui définit un segment de même longueur dans chacun des parents P1 et P2. Les segments sont indiqués avec une trame foncée dans la partie (a) de la figure Fig. 12. Ces segments sont copiés vers les enfants E1 et E2. E1 a hérité du segment de P2 et E2 de P1, comme le montre la partie (b) de la figure.

A partir de l'un de ces segments, on établit à chaque gène une liste de correspondance. Cette liste va servir à placer les gènes redondants et elle est formée de la manière suivante: pour chaque position du segment on note x le gène qui s'y trouve et y celui de l'autre enfant dans la même position. Tant que y est retrouvé ailleurs dans le segment de départ, on note y' son correspondant dans l'autre enfant et on remplace y par y'. Par exemple, le gène correspondant à "1" de El est "6" mais ce gène existe aussi dans El et son correspondant est "3". Ainsi dans la liste de correspondance, on note que "1" a pour correspondant "3". La liste complètement formée se trouve à la marge de la partie (b) du schéma.

On procède ensuite au placement des gènes hors segment en les copiant des parents respectifs. Par exemple, copier "l" de P1 dans E1 provoque un conflit puisque ce gène existe déjà. On utilise alors son correspondant dans la liste qui est "3". En procédant de manière itérative, on arrive à former les enfants E1 et E2 illustrés dans le partie (c) de la figure.

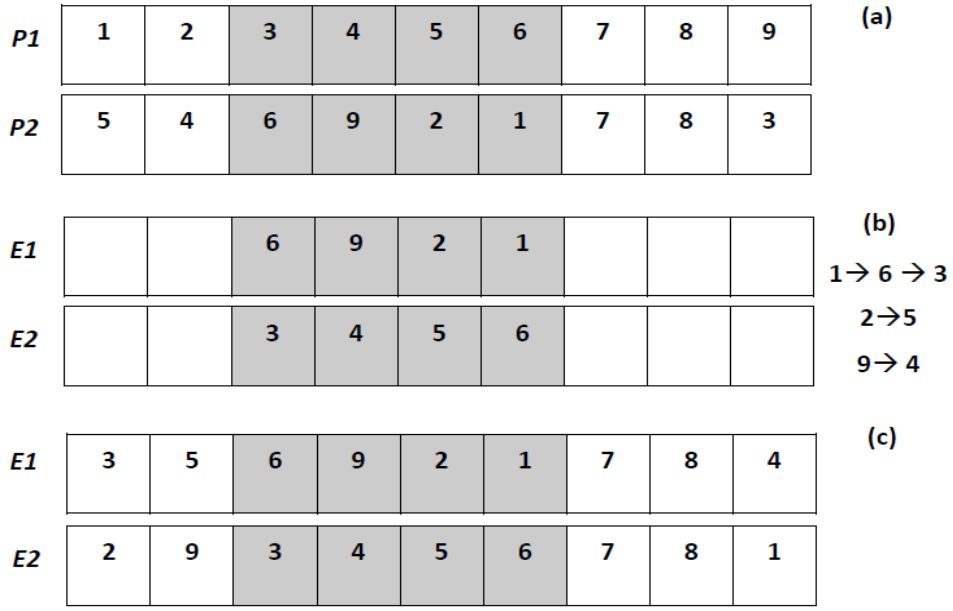


Fig. 12 Opérateur de Croisement PMX

L'opérateur de Croisement CX (Cycle Crossover)

CX est un opérateur qui satisfait la condition suivante: chaque gène d'un enfant provient de l'un des parents à la même position. Les enfants sont donc formés en copiant un gène d'un parent et en éliminant l'autre à la même position puisqu'il va appartenir au deuxième enfant. Une fois que les positions occupées sont copiées par élimination, on a complété un cycle. Les places restantes des deux enfants sont complétées par les parents opposés.

Dans l'exemple présenté dans la figure Fig. 13 partie (a), la première position de E1 est attribuée à "1" provenant de P1. Le gène de P2 situé à la même position est "4", il est recherché dans P1 et retrouvé à la quatrième position. Le gène "4" est copié dans E1 et son correspondant "8" est recherché dans P1 et retrouvé à la huitième position. "8" est copié dans El à cette même position et son correspondant "3" est recherché dans Pl et retrouvé à la troisième position. "3" est copié dans El et son correspondant "2" est recherché dans Pl et retrouvé à la deuxième position. "2" est copié et son correspondant "1" est retrouvé dans Pl à la première position où cette position est déjà occupée dans El donc le cycle est terminé. Le cycle de El est "1, 4, 8, 3,2" et de manière analogue, le cycle "4, 1, 2, 3,8" est formé pour E2.

En dernier lieu, il faut combler les positions vacantes à partir des parents opposés. Par exemple, les gènes "7 ,6 ,9" de El proviennent de P2. On obtient ainsi les enfants illustrés dans la partie (b) de la figure, Fig. 13.

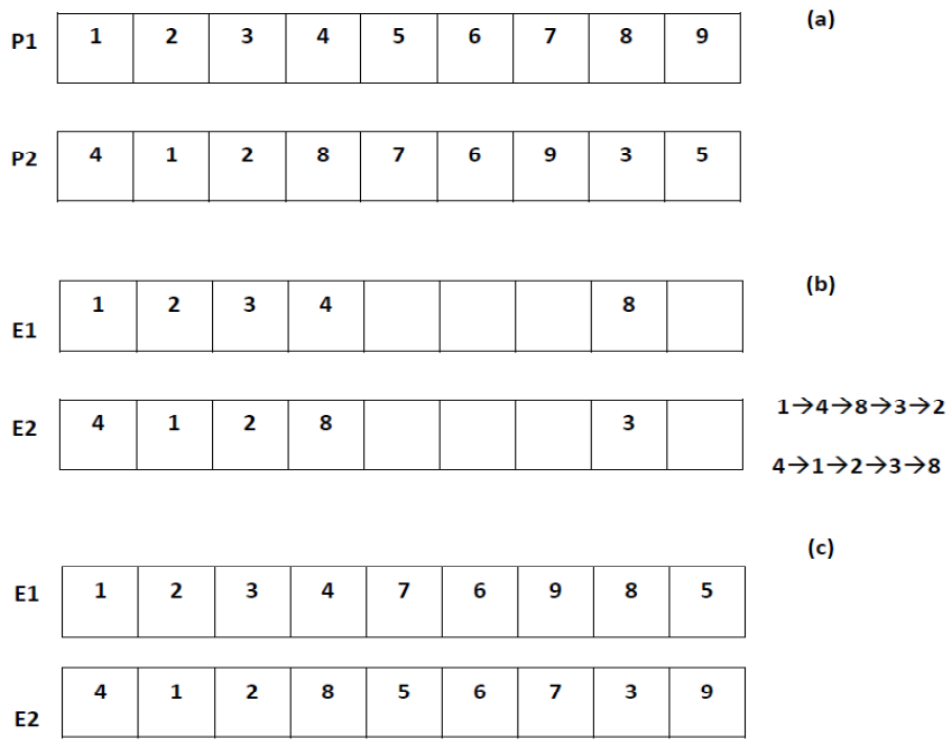


Fig. 13 Opérateur de Croisement CX

L'opérateur de Croisement OX (Order Crossover)

OX est un opérateur avec deux points de coupure qui copie le segment formé par ces deux points de coupure dans les deux enfants. Par la suite, il recopie, à partir du deuxième point de coupure ce qui reste des gènes dans l'ordre du parent opposé en évitant les doublons.

La figure (Fig. 14) présente un exemple du déroulement du croisement avec l'opérateur OX. Dans la partie (a), un segment est formé par les points de coupure dans les deux parents et ces segments sont copiés tels quels dans les enfants El et E2 comme le montre la partie (b).

Enfin, on procède à la copie des gènes situés hors du segment copié. Pour cela, on se place à partir du deuxième point de coupure et on choisit les gènes non redondants provenant du parent opposé. Par exemple, dans la partie (c) de la Figure ?, on essaie de placer le gène "6" de P2 après le deuxième point de coupure dans El mais ce gène existe déjà à l'intérieur du segment. Il est donc ignoré et on passe au suivant. Le gène "2" ne présente pas de conflit, il est donc copié et ainsi de suite jusqu'à où on forme les deux enfants El et E2 tel qu'elle est illustré dans la partie (c) de la figure.

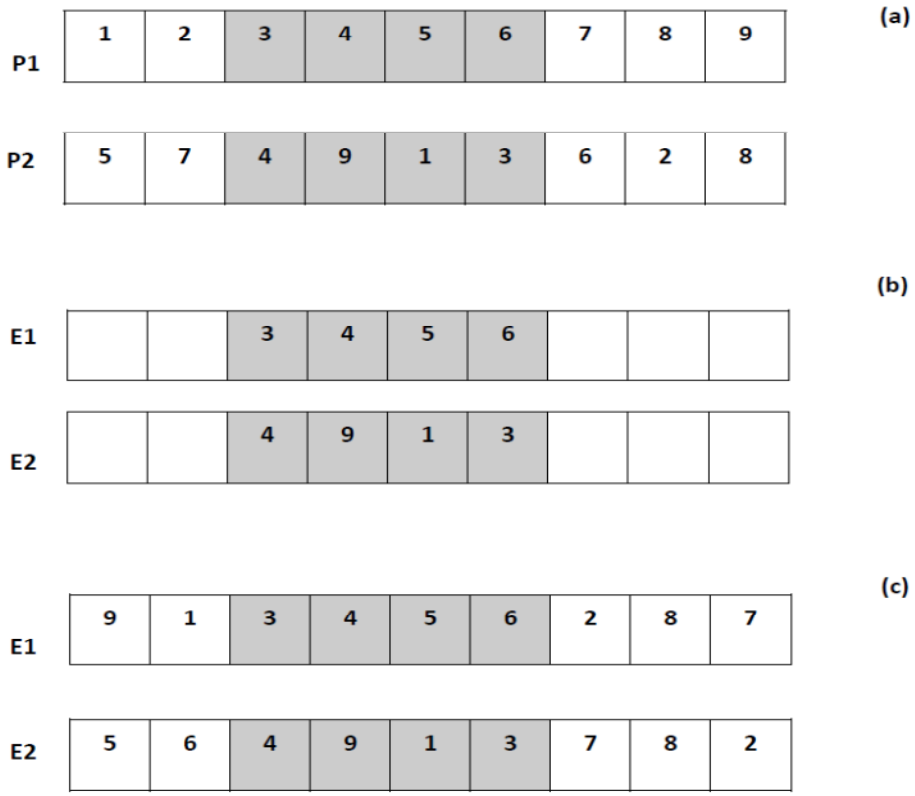


Fig. 14 Opérateur de Croisement OX

2.6 Mutation

C'est un processus où un changement mineur de code génétique est appliqué à un individu pour introduire de la diversité et ainsi d'éviter de tomber dans des optimums locaux. Différentes manières de mutation d'un chromosome sont aussi définies dans la littérature.

2.6.1 Mutation en codage binaire

Dans un algorithme génétique simple, la mutation en codage binaire est la modification aléatoire occasionnelle (de faible probabilité) de la valeur d'un caractère de la chaîne.

2.6.2 Mutation en codage réel

Pour le codage réel, les opérateurs de mutation les plus utilisés sont les suivants :

Mutation par inversion

Deux positions sont sélectionnées au hasard et tous les gènes situés entre ces positions sont inversés. La figure Fig. 15, montre un exemple de mutation par inversion. L'individu *i* avant mutation est représenté dans la partie (a) avec le

segment formé par les deux positions sélectionnées. L'inversion est effectuée à l'étape (b) afin de produire l'individu *i'*.

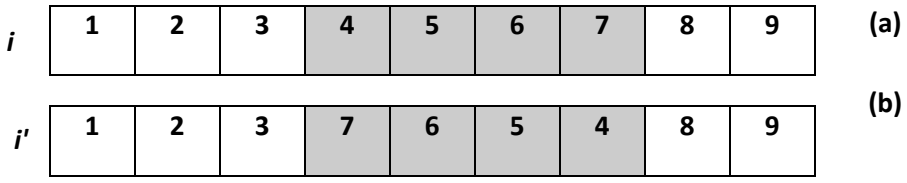


Fig. 15 Mutation par inversion

Mutation par insertion

Deux positions sont sélectionnées au hasard et le gène appartenant à l'une est inséré à l'autre position. Dans la Figure Fig. 16 partie (a), les gènes "3" et "6" de l'individu i sont sélectionnés. La deuxième étape, illustrée par la partie (b), montre l'insertion de "6" avant "3" et le décalage de tous les autres gènes.

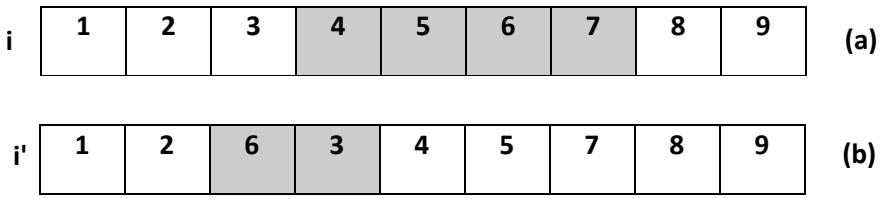


Fig. 16 Mutation par insertion

Mutation par déplacement

Une séquence est sélectionnée au hasard et déplacée vers une position elle-même tirée au hasard. Un exemple de mutation par déplacement est illustré à la figure Fig. 17. Dans la partie (a), la séquence "3-4-5-6" est sélectionnée et déplacée à la première position pour former l'individu i' représenté dans la partie (b).

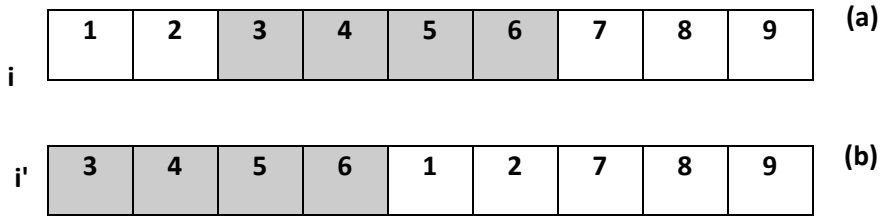


Fig. 17 Mutation par déplacement

Mutation par permutation

Deux positions sont sélectionnées au hasard et les gènes situés dans ces positions sont permutés. Comme il est illustré à la Figure Fig. 18, les éléments "3" et "6" sont sélectionnés dans i (partie (a)) et permutés dans i' (partie (b)).

Fig. 18 Mutation par permutation

3. Algorithme AG

Algorithme 15 Algorithme Génétique

1. *Initialiser la population initiale* $P(t)$.
2. *Evaluer* $P(t)$.
3. ***TantQue*** *(Critère arrêt non atteint)* ***Faire***
4. $P(t+1) \leftarrow$ *Sélection des Parents dans* $P(t)$
5. $P(t+1) \leftarrow$ *Appliquer Opérateur de Croisement sur* $P(t+1)$
6. $P(t+1) \leftarrow$ *Appliquer Opérateur de Mutation sur* $P(t+1)$
7. $P(t) \leftarrow$ *Remplacer les Anciens de* $P(t)$ *par leurs Descendants de* $P(t+1)$
8. *Evaluer* $P(t)$
9. ***FinTantQue***

4. Convergence et mesure de performance d'un AG

Le critère de convergence peut être de nature diverse, Les paramètres qui conditionnent la convergence d'un algorithme génétique sont :

- la taille de la population d'individus
- le nombre maximal de génération
- la probabilité de croisement ;
- la probabilité de mutation.

Les valeurs de tels paramètres dépendent fortement de la problématique étudiée. Ainsi il n'existe pas de paramètres qui soient adaptés à la résolution de tous les problèmes qui peuvent être posés à un algorithme génétique. Cependant certaines valeurs sont souvent utilisées (définies dans la littérature) et peuvent être

de bons points de départ pour démarrer une recherche de solutions à l'aide d'un AG.

- La probabilité de croisement est choisie dans l'intervalle [0.5, 0.9] ;
- La probabilité de mutation est choisie dans l'intervalle [0.05, 0.1].

5. Exercices

Exercice N°1

Codage binaire

Soit à calculer le maximum de la fonction $f(x) = x^2$ définie sur l'intervalle $x \in [0, 31]$. on suppose une taille de population soit égale à cinq individus.

- Donner la taille en nombre de bits d'un individu.
- Appliquer les différents opérateurs de l'algorithme génétique pour deux générations.

Pour le cas de codage binaire, donner les procédures qui implémentent les opérateurs :

- De croisement
- De mutation
- De sélection par tournoi.

Exercice N°2

Soit la fonction f(x)= 1- $(x-1)^2$, on cherche à trouver le maximum de cette fonction à l'aide d'un AG sur l'intervalle $[-10\ \ 10]$.

- Donner le nombre de bits nécessaire pour coder un individu de la population.
- Donner la représentation en binaire des valeurs de variable x=-4, x=-2 x=0, x=6, x=8 et x=10.
- Donner sous une forme tabulaire les valeurs de la fonction d'évaluation, puis les probabilités de sélection de chaque individu.
- Démontrer que la probabilité de sélection d'un individu est de la forme $\frac{1}{n-1}\left(1 - \frac{f(x_i)}{\sum_1^n f(x_i)}\right)$.
- Donner un algorithme qui implémente la méthode de sélection par roue de loterie.

Exercice N°3

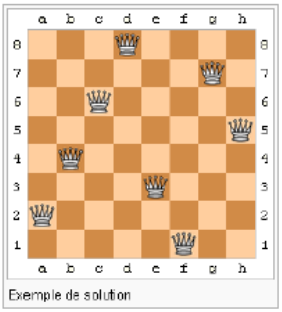
Les algorithmes génétiques réels sont adaptés à la recherche du minimum d'une fonction f définie de $R^n \to R$. Par rapport aux algorithmes génétiques binaires, le seul changement concerne les deux principes de croisement et de mutation.

- Pour le croisement, cas réel, à partir de deux parents p_1 et p_2, deux enfants e_1 et e_2 sont crées avec une probabilité p_c en sélectionnant aléatoirement $\alpha \in [0 \quad 1]$ et en écrivant : $e_1 = \alpha p_1 + (1-\alpha) p_2$, $e_1 = (1-\alpha) p_1 + \alpha p_2$. Ecrire un algorithme de croisement réel.
- Pour la mutation, à partir d'un enfant e, on crée, avec une probabilité p_m un nouvel élément $e' = e + \sigma\mu$ où μ est un vecteur aléatoire suivant une loi normale centrée réduite et σ¾ est un réel strictement positif fixé. Ecrire un algorithme de mutation réel.
- Ecrire un algorithme génétique réel permettant de rechercher le minimum d'une fonction $g(x) = \|x\|^2$ définie sur $[-2,2]^4$.

Reprendre les questions précédentes, si on considère un algorithme génétique binaire qui cherche à minimiser une fonction à n variable définie dans $[a \quad b]^n$.

Exercice N°4

Le but du problème des huit reines, est de placer huit reines d'un jeu d'échecs sur un échiquier de 8×8 cases sans que les reines ne puissent se menacer mutuellement, conformément aux règles du jeu d'échecs. Par conséquent, deux reines ne devraient jamais partager la même rangée, colonne, ou diagonale (voir dessin ci-contre).



Exemple de solution

L'algorithme naïf de recherche exhaustive teste toutes les manières possibles de placer une reine par colonne, pour retirer toutes celles pour lesquelles des reines se menacent mutuellement. Il y a 88 = 16777216 placements à explorer.

On peut le traiter comme un problème d'optimisation si l'on considère qu'il faut minimiser le nombre de conflits. Il s'agira ici de placer les 8 reines sans aucun conflit, en partant d'une solution avec une seule reine par ligne et par

colonne. L'objectif est de mettre en place un AG permettant de résoudre le problème.

- Donner une représentation des solutions pour le problème.
- Proposer une fonction objective permettant l'évaluation des solutions.
- Ecrire les deux procédures implémentant les deux opérateurs génétiques : de croisement et de mutation.

Exercice N°5

On considère le problème du voyageur de commerce pour un graphe complet. On suppose que les villes sont numérotées de 1 à n. Le problème est de trouver le tour (circuit qui visite toutes les villes une fois) le plus court.

Pour utiliser l'algorithme génétique on doit coder le problème permettant de réaliser le croisement et la mutation.

Nous considérons le codage suivant : On écrit un "tour" comme une suite des n villes. On combine deux tours en les coupant de manière aléatoire, au même endroit, et en recollant les morceaux. On appelle ce type de croisement « croisement avec un point de coupure ».

Exemple : n=5

Parent 1	4	5	1	2	3	⇉	Enfant 1	4	5	1	3	5
Parent 2	4	2	1	3	5		Enfant 2	4	2	1	2	3

- Proposer un algorithme permettant d'implémenter ce type de croisement et qui permet de résoudre les problèmes qui posent.
- Proposer aussi un algorithme pour l'opérateur de mutation.
- Même questions si on considère un type de croisement avec deux points de coupure.

Chapitre 5

Chapitre 5
Optimisation par Colonie des fourmis

Chapitre 5
Optimisation par Colonies des Fourmis

1. Introduction

Dans ces dernières années, l'intelligence en essaims avait gagnée une grande popularité. L'intelligence en essaims est souvent définie comme (Bonabeau et al., 1999) :

Toute tentative de concevoir des algorithmes ou des dispositifs de résolution des problèmes, distribuées, inspirés des comportements collectifs des insectes sociaux ou d'autres sociétés animales.

L'approche est motivée par le fait que la richesse en comportements dans les insectes sociaux, émerge, non pas des entités prises séparément, mais à travers les interactions entre elles. Il a été proposé que même avec des sociétés plus complexes, les individus sont moins compliqués et plus simples (Delgado et al. 1997).

Kennedy et autres dans leurs volume «Swarm Intelligence » (Kennedy et al., 2001), avaient remis en cause cette dernière définition, en se basant sur une définition plus ancienne, donnée pour décrire les systèmes de simulation des essaims : « Nous utilisons le terme "Essaims" dans un sens général pour référencer toute structure de collection d'agents en interaction. L'exemple classique d'essaims est l'essaim des abeilles, mais la métaphore d'essaim peut être élargie aux autres systèmes avec une architecture similaire. Une colonie de fourmis peut être vue

comme un essaim dans lequel ces entités et ces agents sont des fourmis, un essaim d'oiseaux est un essaim et pour lequel ces agents sont des oiseaux, … ».

Quatre principes gouvernant l'intelligence en essaims :

Feedback positif : Permet de renforcer les meilleurs choix dans le système.

Feedback négatif : Permet l'ignorance et la suppression des mauvais choix dans le système.

Aspect aléatoire : Permet la bonne exploration de l'espace de solution, d'une manière indépendante de la qualité, ce qui donne de la créativité.

Interaction multiple : Principe de base permettant la construction des meilleures solutions et choix.

Les essaims de fourmis par exemple offrent une grande diversité de comportements et de morphologies. L'étude précise de leur comportement (l'éthologie) est souvent limitée aux espèces les moins populeuses pour des raisons pratiques et évidentes d'étude en laboratoire. Cette diversité exubérante est une mine d'inspiration fascinante pour les systèmes informatiques. C'est ainsi que les capacités des fourmis en matière de coopération, de communication, de compétition et d'apprentissage, peuvent être mises à profit pour la conception de robots ou d'algorithmes de résolution de problèmes, qui constitue l'objet de ce chapitre.

2. Intelligence Collectives des Fourmis

Les fourmis offrent une grande diversité de comportements et de morphologies, l'étude précise de leurs comportements est souvent limitée aux espèces les moins populeuses pour des raisons pratiques. Cette diversité est une mine d'inspiration fascinante pour les systèmes informatiques, c'est ainsi que les capacités des fourmis en matière de coopération, de communication, de compétition et d'apprentissage peuvent être mise à profit pour la conception des algorithmes de résolution des problèmes.

Les principales caractéristiques des fourmis que l'on pourra retrouver dans les systèmes informatiques sont :

2.1 Communication

Les fourmis ont développés des mécanismes de communication très élaborés, il à été définie plusieurs types de réponse mettant en œuvre une forme de communication :

- L'alarme.
- L'attraction simple.
- Le recrutement.
- L'échange d'aliments solides.
- Le marquage de territoire et du nid.

- …etc.

La communication chimique est la plus présente chez les fourmis. Les phéromones, sont des mélanges hydrocarbures, sont à la base de la communication de nombreuses espèces.

Les ouvrières sont capables de déposer des traces chimiques sur le trajet qu'elles empruntent pour ramener de la nourriture. Ils sont capables aussi de déclencher des alarmes quand le nid est attaqué et ainsi mobiliser un grand nombre d'individus pour défendre la fourmilière.

La communication entre les individus peut se faire directement ou indirectement, l'utilisation des phéromones est majoritairement considérée comme une forme indirecte puisque l'échange d'information se fait grâce au support du sol, ce principe est appelé principe de la ***stigmergie.***

2.2 Principe de la Stigmergie

Un principe fondamental des comportements émergents à travers des interactions locales est la stigmergie. Le biologiste Grassé est le premier qui fait introduire ce concept dans les années cinquante (Grassè, 1959), devenue maintenant une voie de recherche et de conception dans les systèmes d'agents artificiel. Grassé fut découvrir le principe en étudiant les comportements des insectes sociaux. Le nome stigmergie dérivé du mot Grec « Stigma » veut dire « Signe » et le mot « Ergon » veut dire « Travail », indiquant que les activités des individus sont influencées par des signes externes, eux-mêmes générés par les activités des individus. Ainsi, des tâches simples doivent être coordonnées par le moyen de la communication indirecte et émanant à l'émergence de phénomènes.

Deux formes de la stigmergie ont été identifiées. Une stigmergie Sema-tectonique, entraînant un changement physique dans les caractéristiques de l'environnement. La construction du nid chez les fourmis est un exemple de la stigmergie sema-tectonique. Une fourmi observe une structure en développement et c'est à son tour d'y participer à la construction. La deuxième forme de la stigmergie est basée sur la trace. Ici les marques sont des traces, déposées par les fourmis sur l'environnement, émanant à une forme de contribution indirecte à la tâche en question et réalisant une influence sur le comportement désiré.

La stigmergie n'explique pas les mécanismes de coordination en détail, Mais pour des problèmes inverses pour concevoir des systèmes accomplissant des tâches globales, la stigmergie fournit le concept général permettant la coordination entre les individus et le comportement global au niveau de la colonie.

Les avantages de l'utilisation de l'approche par stigmergie sont :

- L'utilisation des individus simples.
- Réduction de la communication entre individus.
- Une approche incrémental dans la construction des meilleures solutions. Ce qui montre une méthode d'apprentissage incrémentale pour la résolution des problèmes.

- La flexibilité et la tolérance aux pannes.

Les fourmis sociales exhibent les deux formes de la stigmergie. La stigmergie sema-tectonique est observé dans la construction des piles de fourmis mortes.

La stigmergie à base de la trace de la phéromone est beaucoup plus observée dans la recherche de nourriture, avec l'émergence du chemin minimal entre la source de nourriture et le nid (Goss et al., 1989). Il a été démontré que les fourmis n'utilisent aucune information visuelle dans leurs activités, et sans aucun contrôle central indiquant aux fourmis le chemin à employer (Hölldobler et al., 1990).

2.2.1 Phéromones

La phéromone est une substance chimique qui jeu un rôle très important dans la réalisation des tâches définies. Il permet de refléter une caractéristique des systèmes complexes. La présence de la phéromone dans une piste jeu le rôle d'un moyen permettant l'attraction des fourmis à la piste renforcée. La phéromone est soumise à l'évaporation, due aux contraintes d'environnement. Le rôle inverse est joué à travers le processus de propagation et de sécrétion de la phéromone.

L'évaporation permet de réduire les amplifications et de créer des points de fluctuations, pour lesquels des nouvelles situations prometteuses peuvent être visité. Ce mécanisme conduit à la découverte des nouvelles régions riches en nourriture.

2.2.2 Propagation

La propagation de la phéromone est un processus qui permet de laisser passer une quantité de la phéromone d'un emplacement à un autre emplacement voisin. Si le taux d'évaporation est de ½ par exemple, la moitie de la quantité de la phéromone de l'emplacement actuel est propagée vers les emplacements voisins.

2.2.3 Evaporation

L'évaporation est un processus, qui permet de réduire la quantité de la phéromone dans un emplacement par un taux défini. Cette évaporation est due aux contraintes de l'environnement.

2.3 Le Fourragement

La recherche de la nourriture, appelée aussi « le fourragement », est une activité souvent plus dispersée spatialement que la construction du nid et qui peut aussi être mise en œuvre de façon très différente suivant les espèces de fourmis. La communication peut avoir un impact important, en particulier pour les mécanismes

de recrutement, dont le principal intérêt collectif est de rassembler les ouvrières sur les sources de nourriture rentables. D'un point de vue plus général, la communication mise en œuvre pour la recherche de nourriture peut être considérée comme une forme de mémoire collective.

Les étapes suivantes expliquent bien ce mécanisme,

Fig. 19 Fourmis suit la trace.

La figure Fig. 19, montre un scénario initial. Les fourmis emploient le chemin marqué par la trace de la phéromone. Le chemin joint le nid à la source de nourriture. La phéromone est une substance chimique générée et perçoive par les fourmis. La substance représente le moyen de communication indirecte dans le scénario. Les fourmis dans leurs retours au nid, déposent une quantité de la phéromone à chaque déplacement et préfèrent de se déplacer vers les emplacements les plus concentrés de la phéromone.

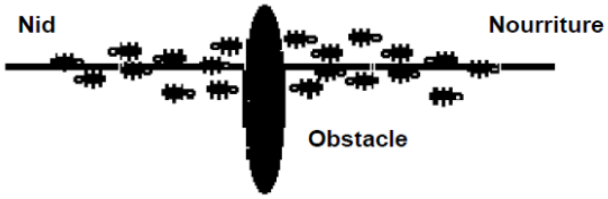


Fig. 20. Introduction d'un obstacle

Lorsque le chemin émergé est obstrué à l'aide d'un obstacle, comme le montre la figure Fig. 20, les fourmis prennent une décision aléatoire pour choisir l'un des deux branches droite ou gauche de l'obstacle. Au début et sans présence d'aucune concentration de la phéromone, le flux des fourmis se subdivise en deux flux approximativement égales. Le choix aléatoire donne une exploration des deux choix, figure 3.

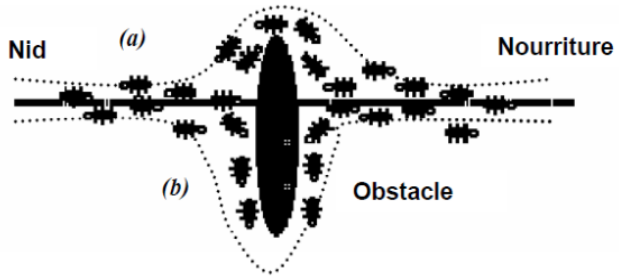


Fig. 21 Les fourmis se divisent et explorent deux chemins.

Le chemin représenté dans la figure Fig. 21 (a), est le plus concentré en phéromone. Ainsi, les fourmis qui circulent sur ce chemin, arrivent en premier à la destination et permettent la formation d'un flux entre le nid et la source de nourriture. Tandis que dans le deuxième chemin, représenté à l'aide de la figure Fig. 21 (b), la quantité secrétée est moins concentré. En effet avec la considération du caractère d'évaporation de la phéromone, le chemin le plus court devient le plus concentré, ce qui conduit à une attraction des fourmis jusqu'au recrutement de toutes les fourmis, comme il est indiqué dans la figure Fig.22.

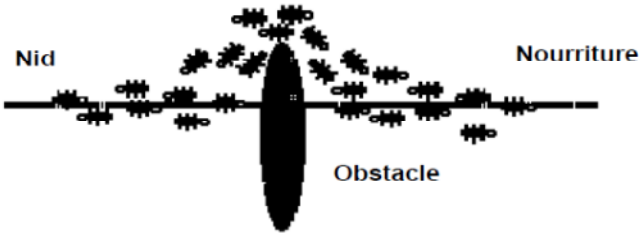


Fig. 22 Emergence du chemin minimal.

3. Optimisation par Colonies des Fourmis

La capacité des fourmis à trouver le plus court chemin entre une source de nourriture et leur nid, a été utilisée pour résoudre des problèmes d'optimisation combinatoire. Les traces de la phéromone, représentent une attirance pour un arc du graphe modélisant le problème. Chaque fourmi construit une solution pour le problème et l'évaluation de chaque solution est utilisée pour mettre à jour les traces de la phéromone. Ces principes ont été appliqués en premier au problème de voyageur de commerce (Colorni et al., 1991) (Colorni et al., 1992). Après des variations de la méthode ont été proposées par (Dorigo et al., 1996a) (Gambardella et al., 1996). La même technique est appelée pour d'autres problèmes combinatoires comme l'affectation quadratique et autres problèmes (Gambardella et al., 1999) (Maniezzo et al., 1999).

3.1 Principes de l'algorithme ACO

La méthode ACO « Ant Colony Optimisation » est une métaheuristique inspirée du comportement des fourmis dans la recherche des nourritures. La première version d'algorithme a été développée par Dorigo (Dorigo et al. 1997).

Lorsqu'une fourmi doit prendre de décision sur la direction à prendre, elle doit choisir le chemin ayant la plus forte concentration en phéromone, c'est-à-dire la décision dépend de la probabilité de transition d'un emplacement à une autre. Cette probabilité dépend de la concentration en phéromone. C'est exactement le principe utilisé par les algorithmes d'optimisation à base des fourmis.

Chaque fourmi est considérée comme un agent capable de générer des solutions, la décision à prendre une fourmi pour construire une solution dépend de deux facteurs :

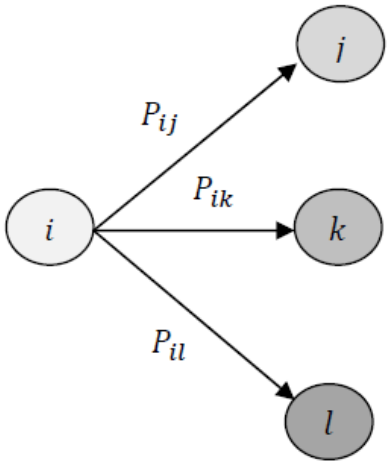
- **Facteur de visibilité**, appelé aussi ***force gloutonne*** noté $\eta_d$où d est la décision à prendre, cette force représente l'apport qu'à la fourmi lorsque elle prend une décision. Cette valeur est directement proportionnelle à la qualité de la solution obtenue en prenant la décision d.
- **Facteur de trace**, τ_d où d est la décision à prendre, cette trace représente l'apport historique qu'à la fourmi lorsque elle prendre la décision. Plus cette valeur est grande, plus il a été intéressant dans le passé de prendre cette décision.

D'où la relation suivante qui représente la probabilité qu'un élément où une décision d est choisie. α et β sont deux paramètres qui mesure l'importance de deux facteurs de visibilité et de trace.

$$P_d = \frac{\tau_d^\alpha \, \eta_d^\beta}{\sum_{e \in D} \tau_e^\alpha \, \eta_e^\beta}, \quad \alpha \geq 0, \beta \geq 0$$

Exemple

Soit i, j, k et l quatre emplacements, supposant que la fourmi se trouve à l'emplacement i, le prochaine emplacement de la fourmi dépend de la visibilité et de la trace, c'est-à-dire de la probabilité P_{ij}, P_{ik} et P_{il}



$$P_1 = \frac{\tau_{ij}^\alpha \eta_{ij}^\beta}{\sum_{p \in (j,k,l)} \tau_{ip}^\alpha \eta_{ip}^\beta}$$

$$P_2 = \frac{\tau_{ik}^\alpha \eta_{ik}^\beta}{\sum_{p \in (j,k,l)} \tau_{ip}^\alpha \eta_{ip}^\beta}$$

$$P_3 = \frac{\tau_{il}^\alpha \eta_{il}^\beta}{\sum_{p \in (j,k,l)} \tau_{ip}^\alpha \eta_{ip}^\beta}$$

3.1.1 Evaporation

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. Soit ρ un paramètre appelé taux d'évaporation choisi dans l'intervalle $]0,1[$. La mise à jour de la trace sur la décision d est réalisée comme suit :

$$\tau_d = (1-\rho)\tau_d + c \sum_{k \in K} \Delta_d^k$$

Où

$$\Delta_d^k = \begin{cases} 1/L_k \text{ si la fourmi k a réalisée la décision } d \\ 0 \quad \text{sinon} \end{cases}$$

c est une constante, K l'ensemble des fourmis et L_k est la solution construite par la fourmi k.

3.1.2 Renforcement

Le processus de renforcement consiste à laisser une quantité de la phéromone à l'emplacement visité par la fourmi. Pour imiter ce principe et pour chacune des K solutions construites à une itération et pour chaque décision d choisie de ces solutions ajouter la quantité $\frac{Q}{L_k}$ à $\tau_d$où Q est un paramètres.

3.3 Algorithme ACO

Algorithme 16 ACO

1. ***Initialiser*** la trace de la phéromone τ_dà 0 pour toute décision d.
2. ***Tant que*** (Critère d'arrêt non atteinte) ***Faire***
3. ***Pour*** Chaque fourmi $k \in K$ ***Faire***
4. Construire une solution en tenant compte de la visibilité et la trace
5. Mettre à jour la trace $\tau_d$ainsi que la solution trouvée
6. ***Finpour***
7. ***FinTantQue***

4. Application : Voyageur de Commerce

Le problème du voyageur de commerce consiste à trouver le trajet le plus court chemin reliant n villes données, chaque ville ne devant être visitée qu'une seule fois. Le problème est plus généralement représenté comme un graphe complètement connecté $G(N, A)$, où N sont les nœuds et N les arêtes.

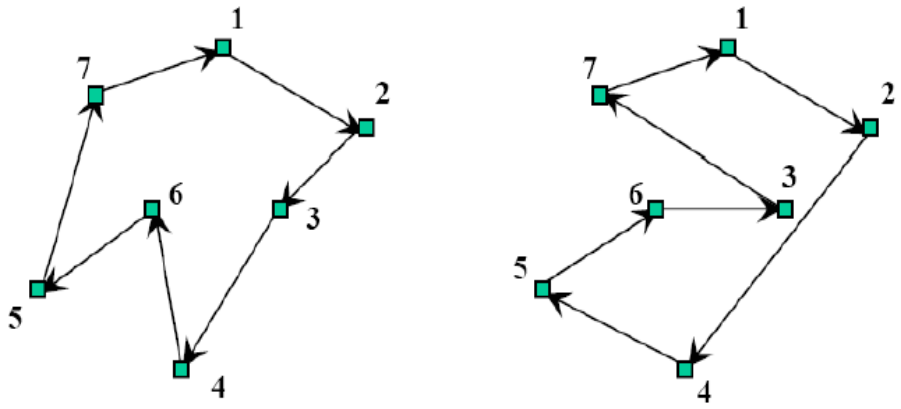


Fig. 23 Exemple d'un PVC de sept villes avec deux tournées différentes.

L'émergence de chemin minimale entre la source de nourriture et leur nid est utilisée en premier pour résoudre le problème de voyageur de commerce. Le travail qui suit est inspirée des travaux de (Colorni et al. 1991) (Colorni et al. 1992) (Dorigo et al. 1996a) (Gambardella et al. 1996).

4.2 Ant system (AS)

Dans l'algorithme "Ant System" (AS) (Colorni et al., 1992), pour chaque itération t avec $1 \leq t \leq t_{max}$, une fourmi k ($k = 1,..,m$) parcourt le graphe et construit un trajet complet. Une première variation de "Ant System" a été proposée dans (Dorigo et al., 1996a) (Gambardella et al., 1996). L'introduction de fourmis "élitistes" la meilleure fourmi, celle qui a effectué le trajet le plus court, dépose une quantité de la phéromone plus grande. Le but est d'accroître la probabilité aux autres fourmis d'explorer la solution la plus prometteuse. Du côté des fourmis artificielles, quelques modifications sont à apportées aux capacités de ces derniers : Elles possèdent une mémoire et elles ne sont pas totalement aveugles.

Dans (Colorni et al., 1991) trois algorithmes ont été introduits qui mettent à profit ce comportement collectif. Ils sont appliqués au PVC, dont voici la modélisation proposée, avec les autres variations de l'algorithme :

Les fourmis sont placées sur les sommets du graphe (i.e. sur chaque ville). Elles se déplacent d'un sommet. On note par $b_i(t)$ le nombre de fourmis dans la ville i à l'instant t et soit $m = \sum_{i=1}^{i=n} b_i(t)$ le nombre total de fourmis. Chaque fourmi possède les caractéristiques suivantes :

- La fourmi dépose une trace de la phéromone sur l'arête (i, j) quand elle se déplace de la ville i à la ville j ;
- Elle choisit la ville de destination suivant une probabilité, qui dépend de la distance entre cette ville et de la phéromone présente sur l'arête (règle de transition) ;
- Afin de ne passer qu'une seule fois par chaque ville, la fourmi ne peut se rendre sur une ville qu'elle a déjà traversée. Pour cela la fourmi est dotée d'une mémoire. Pour éviter qu'elle ne revienne sur ses pas, elle conserve la liste des villes qu'elle a déjà traversées. Cette liste, nommée ***liste-tabou*** est remise à zéro chaque fois que la fourmi a terminé un tour. La liste-tabou constitue la mémoire de la fourmi.

La règle de déplacement appelée "règle aléatoire de transition proportionnelle", utilisée est la suivante :

$$p_{ij}^{k}(t) = \frac{\left(\tau_{ij}(t)\right)^{\alpha}\left(\nu_{ij}\right)^{\beta}}{\sum_{l \notin L_k(i)}\left(\tau_{il}(t)\right)^{\alpha}\left(\nu_{il}\right)^{\beta}} \quad si\ j \notin L_k(i)$$

- $L_k(i)$ représente la *liste-tabou* de la fourmi k située sur le sommet i . C'est la liste des villes déjà visités par la fourmi k .

- ν_{ij} représente une mesure de ***visibilité*** qui correspond à *l'inverse de la distance* entre les villes i et j , $\nu_{ij} = \frac{1}{d_{ij}}$. Cette information statique est utilisée pour diriger le choix des fourmis vers des villes proches, et éviter les villes trop lointaines.
- τ_{ij} la quantité de la phéromone déposée sur l'arête, reliant les deux villes est appelée l'***intensité de la piste***. Ce paramètre définit l'attractivité d'une partie du trajet global, qui change à chaque passage d'une fourmi. C'est en quelque sorte une mémoire globale du système qui évolue par apprentissage.
- α et β sont deux paramètres contrôlant l'importance relative de l'***intensité*** de la piste, $\tau_{ij}(t)$, et de la ***visibilité***, ν_{ij}.

Avec $\alpha = 0$, seule la visibilité de la ville est prise en compte, la ville la plus proche est donc choisie à chaque pas. Au contraire, avec $\beta = 0$, seules les pistes de la phéromone sont prises en comptes. Pour éviter une sélection trop rapide d'un trajet, un compromis entre ces deux paramètres jouant sur les comportements de ***diversification*** et **d'*intensification*** sera nécessaire. Après un tour complet, chaque fourmi laisse une quantité de la phéromone $\Delta\tau_{ij}^{k}(t)$ sur l'ensemble de son parcours, quantité qui dépend de la qualité de la solution trouvée :

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \dfrac{Q}{L^{k}(t)} & si\ (i,j) \in T^{k}(t) \\ 0 & si\ (i,j) \notin T^{k}(t) \end{cases}$$

Où $T^{k}(t)$ est le trajet effectué par la fourmi k à l'itération , $L^{k}(t)$ la longueur de tour et Q une constante. L'algorithme ne sera pas complet sans le processus d'évaporation des pistes de la phéromone. En effet, pour éviter d'être piégé dans des solutions sous optimales, il est nécessaire de permettre au système "d'oublier" les mauvaises solutions. La mise à jour de la phéromone est effectuée une fois que toutes les fourmis sont passées par toutes les villes.

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t)$$

Où ρ , est le coefficient d'évaporation des traces de la phéromone et $\Delta\tau_{ij}^{k}$ représente le renforcement de l'arc (i,j) par la fourmi. La quantité initiale de la phéromone sur les arêtes, est une distribution uniforme d'une petite quantité $\tau_0 \geq 0$.

4.3 Algorithme AS pour e PVC

Les étapes essentielles de l'algorithme résolvant le voyageur de commerce (PVC) à base des fourmis, sont les suivantes :

Algorithme 17 Fourmis TSP

1. Initialisation()
2. ***Tantque*** $(t \leq t_{max})$ ***Faire***
3. ***Pour*** chaque fourmi k $(k = 1, .., m)$ ***Faire***
4. Choisir une ville au hasard
5. Pour chaque ville non visitée i ***Faire***
6. Choisir une ville j, selon la règle de transition.
7. ***Finpour***
8. Déposer une trace $\Delta\tau_{ij}^{k}(t)$ sur le trajet $T^{k}(t)$.
9. ***Finpour***
10. Vaporiser les traces selon la relation $\tau_{ij}(t+1)$.
11. Déplacement des fourmis (t←t+1)
12. ***Fin tantque***

Chapitre 6

Chapitre 6
Méthode d'optimisation par essaims particulaires

Chapitre 6
Optimisation par Essaims Particulaires

1. Introduction

Les réseaux de nuerons, le recueil simulé, les algorithmes culturels, l'optimisation par colonies des fourmis et les algorithmes évolutionnaires sont des instances pour lesquelles les théories en psychologie, en physique et en biologie avaient influenceraient le développement des méthodes de calculs pour la résolution des problèmes.

Les individus apprennent, l'un de l'autre, non seulement les faits mais aussi les méthodes et techniques, traitant ces faits. Le modèle de simulation appelé Modèle d'Adaptation Culturelle (MAC), montre à travers les simulations informatiques que la dissémination des propriétés culturelles, peut émerger à travers des interactions sociales (Axelrod, 1997). Ce modèle constitue l'élément de base de la méthode d'optimisation par essaims particulaires (OEP).

La méthode OEP, développée par Kennedy et al. (Kennedy, et al. 1995) , a été influencée par les travaux cités au-dessus, pour lesquels l'apprentissage social est l'élément de base de ces modèles. La méthode a été aussi influencée par les travaux de Reynold et Heppener dans la simulation des essaims, et plus particulièrement la simulation de comportement de vol d'oiseaux (Reynold, 1987) (Heppner, 1990).

L'OEP, est un outil d'optimisation basé sur une population d'individus. Il a été prouvé que l'OEP peut donner des bons résultats pour les problèmes d'optimisation combinatoires discrets (Sheloker et al. 2006), et continues avec une fonction mono-objective ou multi-objectives (Van der bergh et al. 2006). La méthode OEP est favorablement comparable aux autres méthodes d'optimisations,

comme les algorithmes évolutionnaires et l'évolution différentielle (Vesterstrom et al. 2004).

2. Algorithme de Base

L'algorithme de base d'OEP, consiste en un essaim de particules ou une population de solutions candidate. Ces particules coexistes et en évolution dans l'espace de recherche, basés sur leurs expérience et le savoir partagés avec le voisinage. Chaque particule possède deux paramètres, la position et la vitesse (x(t),v(t)) . La population vole dans l'espace de recherche à base des deux équations (1) et (2), suivantes (Clerc, 2006) :

$$x(t+1) = x(t) + v(t+1) \quad \textbf{(1)}$$

$$v(t+1) = wv(t) + R(c)(p(t) - x(t)) + R(c)(g(t) - x(t)) \quad \textbf{(2)}$$

Algorithme 18 Fourmis TSP

1. Initialisation()
2. t←0
3. Initialisation
4. ***Tant que*** True ***Faire***
5. ***Pour*** chaque Particule ***Faire***
6. Calculer X(t+1) et V (t+1), selon les équations(1)et(2)
7. ***SI*** f (X (t +1)) est meilleure que f (p(t)) ***Alors***
8. remplacer p(t +1) avec X(t +1).
9. ***FINSI***
10. Remplacer g (t +1) avec le meilleur des p(t +1) .
11. ***FINPOUR***
12. t ←t+1
13. 11. True ←(t=max t)
14. ***Fin Tantque***

Plusieurs versions du modèle d'équation de base de la méthode, ont été proposées dans la littérature. Une standardisation a été proposée en 2006 visant à donner un seul modèle d'équation. Un grand nombre des travaux réalisés à base de la méthode, vise à trouver des valeurs aux paramètres d'équations, afin d'adapter les particules dans l'espace des solutions.

Nous résumons les progrès de la méthode en trois catégories :

- Certains travaux étaient

dévouées aux avancements théoriques, comme la convergence de modèle d'équation ou l'adaptation du modèle aux environnements dynamiques (Kennedy et al. 1995) (Clerc, 2006) (Van der bergh et al. 2006).

- D'autres travaux, sont centrés sur l'aspect social des particules, proprement dit le voisinage informant les particules (Kennedy et al. 2003) (Mendes et al. 2004).
- Le reste des travaux a été concentré sur l'utilisation de la philosophie de la méthode, pour la résolution des problèmes d'optimisation dans les différentes disciplines.

3. Essaims particulaires et le voisinage

3.1 Modèle topologique de base

Deux modèles considérés comme ancêtres pour l'OEP, le modèle *Lbest* et le modèle *Gbest*. Ces deux modèles représentent permettent de préciser le voisinage informant des particules.

Dans la modèle *Lbest,* un individu est connecté à son voisin immédiat selon une topologie en anneau. L'individu d'index i est influencé par les deux individus, celui d'ordre $(i-1)$ et de $(i+1)$, respectivement. Typiquement, il n'y a que deux voisins pour un individu dans le modèle *Lbest*.

Dans la topologie *Gbest*, un individu est connecté à la totalité de la population selon une topologie en étoile. Le meilleur individu acte comme un attracteur, permettant d'attirer tous les individus de la population vers son bassin. Le mouvement des individus est influencé par l'attracteur.

Dans le modèle *Lbest,* l'essaim est divisé en groupes d'individus, qui permettent une meilleure exploration de l'espace de recherche (Kennedy et al. 2003) (Abraham et al. 2005). Par contre dans le modèle *Gbest,* il n y a qu'un seul groupe qui explore l'espace de recherche, ce qui donne une convergence prématurée de l'essaim vers des optimums locaux. Ce modèle est bien conseillé dans les problèmes où la fonction objective est unimodale, alors que l'approche *Lbest* est recommandée pour les fonctions objectives multimodales.

3.2 Autres topologies

La notion de topologie est divisée en deux types : spatiale, si le voisinage est mesuré à travers une distance euclidienne entre deux positions ou sociale, si il est déterminé à travers une structure de données comme les tableaux, listes, arbres ou hiérarchique. Dans la plus part des cas, les topologies de voisinage contrôlent les deux critères d'exploration et d'exploitation d'une population. Elles affectent ainsi, la capacité de communication et la performance des groupes (Kennedy et al. 2003) (Mendes et al. 2004). Il existe plusieurs topologies :

- **Voisinage de Von Neumann :** Utilisé dans les automates cellulaires. Les individus sont rangés dans une grille carrée, chaque individu est connecté aux voisins en haut, en bas, droit et gauche.

- **Voisinage de Moore :** Le voisinage d'un individu est composé de voisinage de Van Neumann plus les quatre emplacements diagonaux.

- **Topologie en étoile :** Tous les individus sont connectés à un seul individu superviseur. Ce dernier compare les performances de tous les individus.

- **Topologie pyramidale :** Le voisinage d'un individu est constitue de trois voisins : le premier, le deuxième et le troisième individu ayant la plus petite distance moyenne de l'individu, dans l'ordre des nœuds du graphe.

- **Cluster** topologie **:** Représente quatre cliques connectées entre eux par plusieurs chemins. Sociologiquement, se ressemble à des communautés pour lesquels une minorité d'individus ont une tendance en dehors de groupe (Mendes et al. 2004) .

- **Topologie** hiérarchique ***:*** le voisinage d'un individu est l'individu en lui-même, avec ces parents dans la hiérarchie. Cette topologie a été présentée dans la méthode optimisation par essaims particulaires hiérarchiques (HPSO) (Janson et al. 2005).

- ***Topologie*** Aléatoire ***:*** Proposée par Clerc, les k individus informant, y compris l'individu lui-même, sont choisies aléatoirement de l'ensemble S (S : la taille de la population) selon une distribution uniforme (Clerc, 2007). La probabilité de distribution est donnée par la formule N_F/S^{k-1}, où N_F , est un nombre favorable dans S_{k-1} .

3.3 L'essaim de particule entièrement informé

L'essaim de particule entièrement informé ou ''*Fully Informed Particle Swarm optimizer (FIPS)*'' étudié en détail par Mendes, Eberhart et Kennedy (Kennedy, 1999) (Kennedy et al. 2003) (Mendes et al. 2004) (Mendes, 2004). Chaque individu, profite des informations sur toutes les topologies de voisinage. Cette variante est basée sur le fait que le facteur de construction proposé par Clerc et Kennedy impose que le coefficient d'apprentissage $\varphi = \varphi 1 + \varphi 2$, utilise seulement des informations provenant de deux attracteurs, la meilleure position retrouvée par l'individu, et la meilleure position obtenue dans le voisinage de celui-ci (Clerc et al. 2002).

Le modèle canonique avec le facteur de construction proposé par Clerc, est représenté comme suit :

$$V(t+1) = \chi\,(V(t) + U[0,\varphi_1](P_I - X(t)) + U[0,\varphi_2](P_g - X(t))) \quad \textbf{(3)}$$

$$X(t+1) = X(t) + V(t+1) \quad \textbf{(4)}$$

Dans la littérature, l'algorithme est toujours utilisé avec $\chi = 0.7298$ et $\varphi 1 = \varphi 2 = 2{,}05$. χ dépend des coefficients d'apprentissage. Le système précédent est équivalent à :

$$V(t+1) = \chi(V(t) + \varphi(P_m - X(t))) \quad \textbf{(5)}$$
$$X(t+1) = X(t) + V(t+1) \quad \textbf{(6)}$$

avec $= \varphi_1 + \varphi_2$, et

$$P_m = \frac{\varphi_1 P_i + \varphi_2 P_g}{\varphi_1 + \varphi_2}, \qquad \varphi_1 + \varphi_2 > 4 \ (\mathbf{7})$$

Le facteur de construction est donné par :

$$\chi = \frac{2k}{\left|2 - \varphi - \sqrt{\varphi(\varphi - 4)}\right|}, k = 1 \ (\mathbf{8})$$

La pondération aléatoire des termes *Pi* et *Pg* , conduite à une recherche munie par l'individu entre et en dehors de la région définie par les deux points *Pi* et *Pg* . Mendes, propose une alternative pour calculer *Pm* en distribuant la pondération sur le voisinage entier (Mendes et al. 2004).

$$P_m = \frac{\sum_{k \in N} w(k) \varphi_k P_k}{\sum_{k \in N} w(k) \varphi_k} \ (\mathbf{9})$$

où

$$\varphi_k = U\left[0, \frac{\varphi_{max}}{|N|}\right], \forall k \in N \ (\mathbf{10})$$

Tous les voisins d'un individu contribuent dans l'ajustement de la vitesse. Avec cette stratégie l'individu est entièrement informé.

w(k) dans l'équation (9) représente le facteur de pondération. Trois approches ont été considérées dans (Mendes, 2004) :

- La première est lorsque les solutions contribuent également.
- La deuxième lorsque les solutions sont pondérés par qualité.
- La dernière lorsque les individus sont pondérés à base d'une distance. Par exemple lorsque la qualité est utilisée, Pm est calculé à l'aide de l'équation :

$$P_m = \frac{\sum_{k \in N} \frac{\varphi_k P_k}{f_k}}{\sum_{k \in N} \frac{\varphi_k}{f_k}} \quad (11)$$

Différents FIPS ont été proposés et comparés avec le modèle canonique. Les topologies étaient crées à partir d'un graphe aléatoire. Les résultats obtenus montrent que les performances dépendent des topologies de voisinages.

Plusieurs topologies ont été testées pour arriver à cerner la différence entre l'OEP canonique et FIPS. Dans l'OEP canonique lorsque seul le meilleur voisin n'a aucun effet, l'influence est importante, et une convergence prématurée est produite. Dans FIPS, diverses régions sont découvertes, et l'individu est attiré vers une région entre optimums.

La vitesse dans la version canonique d'OEP, est ajustée par la somme des différences entre la position courante et celle de la meilleure dans l'expérience et la meilleure dans le voisinage. Dans FIPS, la vitesse est ajustée par une somme, de sorte d'une différence moyenne entre les meilleurs voisins antérieurs et la position courante de l'individu concerné.

L'importante différence entre l'OEP canonique et FIPS est la contribution de l'expérience dans l'apprentissage, en contraste avec l'OEP canonique dans laquelle l'expérience contribue par la moitie. Dans FIPS l'individu est complètement informé.

Différentes expérimentations ont été faites par Kennedy et Mendes, à base de différentes topologies on utilisant plusieurs types de FIPS, avec différents coefficients de contribution dans l'apprentissage. Pour conclure, dans l'ordre de comparer, il serait nécessaire de trouver des topologies qui s'adaptent bien à chaque algorithme. Alors on peut penser à quelque chose comme « le meilleur voisin possède un grand poids » (Clerc, 2006).

4. Applications

4.1 Optimisation Continue

Soit les fonctions :

$$Schaffer\ f6, \qquad z = f(x,y) = \frac{1}{2} + \frac{sin^2(\sqrt{x^2+y^2}) - \frac{1}{2}}{1 + \frac{1}{10}(x^2+y^2)^2}$$

$$Rosenbrock, \qquad f(x) = \sum_{i=0}^{n} 100(x_{i+1} - {x_i}^2)^2 + (x_i - 1)^2$$

$$Rastrigin, \qquad f(x) = \sum_{1}^{D} (x^2 - 10\cos(2\pi x)) + 10$$

Ces fonctions et d'autres sont des benchmarks utilisées pour tester les méthodes d'optimisation global pour le cas de minimisation.

Nous utilisons la fonction Rosenbrock pour l'implémenter par la méthode d'optimisation par essaimes, nous utilisons le standard PSO définie par Clerc (Clerc 2006).

x est de dimension D, donc chaque individu ou position de l'individu est représenté sous forme d'un vecteur de dimension D. Le minimum de la fonction est connu, $\min = 0$.

Le standard PSO est

$$x(t+1) = x(t) + v(t+1)$$
$$v(t+1) = wv(t) + R(c)(p(t) - x(t)) + R(c)(g(t) - x(t))$$

où

R(c): Une distribution aléatoire enc,

p(t): Meilleur position obtenu par l'individu dans la mémoire.

g(t): Meilleur position obtenu par l'informant (dans le voisinage).

4.1.1 Paramètres d'Algorithme

La vitesse et la position seront déclarées des structures :

```
Position = structure {
                  Taille : Entier ;
                  x : vecteur [1..Dmax] de Réel ;
                  f : Réel
                 }

Vitesse  = structure {
                  Taille : Entier ;
                  v : vecteur [1..Dmax] de Réel ;
                 }
```

S : taille d'essaime ; % $S = 10 + 2 * \sqrt{D}$;

V : Vecteur [1..S_{max}] de Vitesse ; % S_{max} : taille maximum d'essaime,

X : vecteur [1..S_{max}] de Position ;

K : nombre d'informant, K = 3; w = $\frac{1}{2\log 2}$; c = $\frac{1}{2} + \log 2$.

P : vecteur [1..S_{max}] de Position ; % ce vecteur contient la meilleure position de chaque individu obtenu dans l'expérience (c'est le Lbest dans le modèle d'équation). La meilleure des meilleurs positions est un rang noté Best,(c'est le Gbest dans le modèle d'équation).

4.1.2 Initialisation de l'essaime (position et vitesse)

$Pour\ (s = 0,..,S)\ Faire$
$X[s].taille \leftarrow D; V[s].taille \leftarrow D;$
$Pour\ (d = 0,..,D)\ Faire$
$X[s].x[d] \leftarrow aléa(x_{min}[d], x_{max}[d])$;
V$[s].v[d] \leftarrow (x_{min}[d] - x_{max}[d]) * (\frac{1}{2} - aléa(0,1)$;
$FinPour$
$FinPour$

$aléa(a,b)$: fonction de distribution uniforme sur $[a,b], x \leftarrow aléa(a,b) \Leftrightarrow x = a + \rho(b-a)$.

4.1.3 Première Evaluation

$Pour\ (s = 0,..,S)\ Faire$
$X[s].f \leftarrow Eval(s) - f_{min}$ % *le minimum* f_{min} *cherché est connu.*
$P[s] \leftarrow X[s]$ % *la meilleure dans l'expérience est elle-même la première évaluation*
$FinPour$
$best \leftarrow 0$;
$Pour\ (s = 0,..,S)\ Faire$
Si (P$[s].f < P[best].f\ Alors\ Best \leftarrow s$;
$FinPour$
$Error \leftarrow P[best].f,\qquad Error_{prec} \leftarrow P[best].f;$

4.1.4 Fonction d'Evaluation Eval()

$fonction\ Eval\ (s: entier): réel;$
$k \leftarrow 10; f \leftarrow 0;$
$Pour\ (d = 0,..,D) Faire$
$f \leftarrow f + X[s].x[d] * X[s].x[d] - k * \cos(2\pi * X[s].x[d])$;
$Finpour$

$f \leftarrow f + k * D;$
$Eval \leftarrow f;$

4.1.5 Topologie d'Informant

La structure de données la plus performante pour représenter la topologie d'informant dans ce cas, est d'utiliser une matrice carrée appeléeLink. Chaque ligne de rang s dans la matrice représente l'ensemble d'informant d'individu s. Si Link[s, j] = 1 alors j est un informant de s, sinon Link[s, j] = 0.

Nous utilisons une procédure inflink() permettant l'initialisation et la mise à jours de la liste d'informant de chaque individu. La procédure permet la mise à jours d'informant uniquement lorsque il y aura une stagnation dans la solution cherchée. Nous notons que chaque individu est capable de s'auto-informer.

$Procedure\ inflink\ (link : Matrice\ [1..S_{max}, 1..S_{max}] de\ (0,1));$

$Pour\ (s = 0,..,S-1)\ Faire$
$\quad link[s,s] \leftarrow 1; \quad \%\ s'auto-informer$
$\quad Pour\ (m = 0,..,S-1)\ Faire$
$\quad\ if\ s \neq m\ alors\ link[s,m] \leftarrow 0; \ \%\ aucun\ informant$
$\quad Finpour$
$Finpour$

$\%\ les\ autres\ informants;$
$Pour\ (s = 0,..,S-1)\ Faire$
$\quad Pour\ (i = 0,..,K-1)\ Faire \ \%\ K: nombre\ d'informant$
$\quad\ m \leftarrow alea\ (0, S-1);$
$\quad\ link[s,m] \leftarrow 1;$
$\quad Finpour$
$Finpour$

Avec ces instructions chaque particule comporte au maximum K informant. La fonction $alea(a,b)$ est une distribution uniforme dans (a,b) elle retourne $a + \rho(b+1-a), \rho \in [0,1]$.

4.1.6 Mouvements de l'essaim

$Procedure\ Move_P \begin{pmatrix} X: vecteur\ [1..S_{max}] de\ position, \\ V: vecteur\ [1..S_{max}] de\ vitesse \end{pmatrix};$

$Pour\ (s = 0,..,S-1)\ Faire \ \%\ pour\ chaque\ particule$
$\quad \%\ trouver\ le\ meilleur\ informant$

$g \leftarrow s;$
$Pour\ (m = 0, .., S-1)\ Faire$
$if\ (link[s,m] = 1) et (P[m].f < P[g].f)\ alors\ g \leftarrow m;$
$Finpour$
$\%\ vitesse\ de\ déplacement$
$Pour\ (d = 0, .., D-1)\ Faire$
$V[s].v[d] \leftarrow w * V[s].v[d] + alea(0,c) * (P[s].x[d] - X[s].x[d]);$
$V[s].v[d] \leftarrow V[s].v[d] + alea(0,c) * (P[g].x[d] - X[s].x[d]);$
$X[s].x[d] \leftarrow X[s].x[d] + V[s].v[d];$
$Finpour$
$\%\ Contrôle\ des\ excess\ de\ vitesse\ et\ de\ position\ x\ et\ v\ dans\ [x_{min}, x_{max}]$
$Pour\ (d = 0, .., D-1)\ Faire$
$Si\ X[s].x[d] < x_{min}[d]\ alors$
$X[s].x[d] \leftarrow x_{min}[d];$
$V[s].v[d] \leftarrow 0;$
$Finsi$

$Si\ X[s].x[d] > x_{max}[d]\ alors$
$X[s].x[d] \leftarrow x_{max}[d];$
$V[s].v[d] \leftarrow 0;$
$Finsi$
$Finpour$

$\%\ Evaluation\ de\ la\ nouvelle\ poistion$
$X[s].f \leftarrow Eval(s) - f_{min;}$
$\%\ Modification\ de\ l'historique\ des\ poistions\ p_{best}$
$Si\ X[s].f < P[s].f\ \ alors\ P[s] \leftarrow X[s];$
$\%\ Modification\ de\ g_{best}$
$Si\ P[s].f < P[Best].f\ \ alors\ Best \leftarrow\ s;$
$\%\ Calcul\ d'erreur$
$error\ \leftarrow P[Best].f;$
$Si\ error\ \geq error_{prec}\ alors\ \ P[s].f < P[Best].f\ \ alors\ inflink(link);$
$error_{prec} \leftarrow error;$

$Finpour$

Les variables error et $error_{prec}$ sont utilisées pour réinitialisation de la matrice d'informant dans le cas où une stagnation au niveau d'un optimum local aura lieu.

4.2 Optimisation Discret

4.2.1 Problématique

Le problème de Collectes et Livraisons Général - PCLG (GPDP : General Pick-up and Delivery Problem), est un problème d'optimisation, qui consiste à trouver un ensemble des tournées satisfont un ensemble de demandes de transport. Pour réaliser ces tournées, une flotte de véhicules est disponible. Chaque demande de transport est caractérisée par une charge de transport, un ou plusieurs sites d'origine ou de collecte (*pick-up*), et un ou plusieurs sites de destination ou de livraison (*delivery*). Lorsqu'une demande de livraison est prise en charge, elle ne peut être effectuée que par un seul véhicule et sans livraison intermédiaire de cette charge sur un site autre que la destination.

Le problème général (PCL) est défini de la façon suivante : Soit *n* le nombre de demandes de transport. Chaque demande de transport *i* est caractérisée par une charge q_i à transporter de l'ensemble des origines N_I^+ vers l'ensemble des destinations N_I^- . On suppose que cette charge est répartie sur les origines et les destinations de la façon suivante : $q_i = \sum_{j \in N_i^+} q_j = \sum_{j \in N_i^-} q_j$. De cette façon, les sites de collecte ont une charge positive et les sites de livraison une charge négative.

Soit $N^+ = \bigcup_{i \in N} N_i^+$ l'ensemble de toutes les origines, $N^+ = \bigcup_{i \in N} N_i^-$ l'ensemble de toutes les destinations, et $N = N + \cup N -$. Soit M l'ensemble des véhicules disponibles. Chaque véhicule $m \in M$ a une capacité maximale Q_m. Soit $M^+ = \{m^+ | m \in M\}$ l'ensemble des sites de départ (dépôts) des véhicules, $M^- = \{m^- | m \in M\}$ l'ensemble des sites d'arrivée des véhicules, et $W = M^+ \cup M^-$. Pour tout $(i, j) \in N \cup W$, on note d_{ij} la distance, t_{ij} le temps et c_{ij} le coût associés au trajet (i, j) .

4.2.2 Description et Contraintes liés au problème

Soit *n* le nombre de demandes de transport à satisfaire. Chaque demande de transport *k* concerne un seul passager qui doit être transporté d'un site d'origine p_k à un site de destination d_k. Le temps nécessaire pour embarquer ou débarquer un passager sur un site *i* est appelé temps de service et est noté s_i. Pour satisfaire les *n* demandes de transport, un seul hélicoptère est disponible.

Le véhicule est caractérisé par une capacité maximale Q en nombre de passagers, une durée maximale de vol T au bout de laquelle il doit se ravitailler en carburant à la base. Une capacité d'approche a_i est affectée à chaque site. Elle correspond au nombre maximal de passagers dans l'hélicoptère au moment de son atterrissage sur un site, ceci pour des raisons de sécurité.

Dans ce problème les distances sont euclidiennes. Ainsi pour deux sites *i* et *j* , la distance d_{ij} respecte l'inégalité triangulaire $d_{ij} \leq d_{ik} + d_{kj}$ et $d_{ij} = d_{ji}, \forall i, j, k$. Le temps de trajet t_{ij} entre deux noeuds *i* et *j* est pré-calculé à partir de la distance entre les sites et de la vitesse moyenne de véhicule.

Ce problème contient les contraintes classiques du PCL qui sont :

- **Couplage :** Pour chaque demande de transport k , le site de collecte p_k et le site de livraison d_k doivent être visités dans la même tournée.
- **Précédence :** Pour une demande de transport k , le site de collecte p_k doit être visité avant le site de livraison d_k .
- **Capacité :** Le nombre de passagers dans l'hélicoptère ne doit pas dépasser Q. Les deux contraintes spécifiques suivantes doivent de plus être respectées :
- **Capacité d'approche :** Pour des raisons de sécurité, le nombre maximal de passagers dans l'hélicoptère lors de son atterrissage sur un site peut être limité.
- **Durée maximale des tournées :** Étant donné que l'hélicoptère a une durée de vol limitée T au bout de laquelle il doit retourner se ravitailler en carburant à sa base et qu'il ne peut retourner à la base s'il a des passagers à son bord, la durée des tournées est elle aussi limitée à T .

L'objectif de ce problème est de satisfaire toutes les demandes de transport en minimisant la distance totale parcourue. Ce problème est donc un problème de transport à la demande avec des contraintes de capacité d'approche et de durée maximale des tournées.

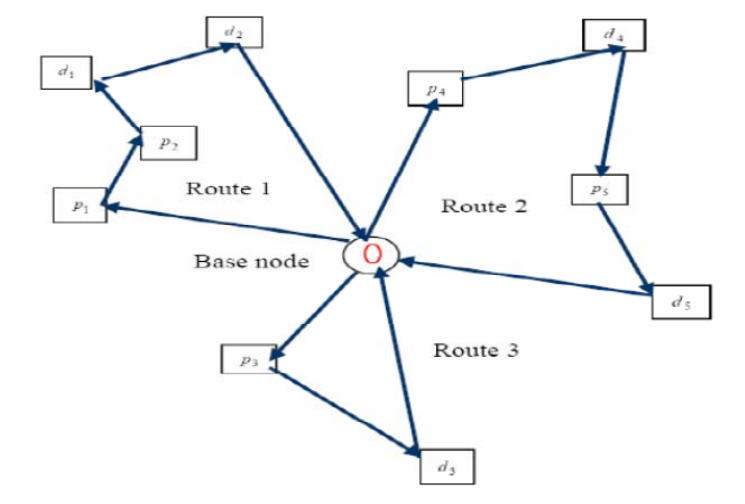


Fig. 24 Représentation graphique d'un Problème PCL.

4.2.3 Méthode de représentation

Méthode 1

Une représentation directe de problème de tournées de véhicules avec ou sans fenêtre horaire à un ou plusieurs véhicules, est d'utiliser un vecteur composé de séquence de nœuds pour lequel les tournées sont séparées par le nœud représentant la base.

Supposons que la base est représentée par le nœud 0 un exemple de solution pour 6 clients est donnée comme suit :

0	5	0	4	3	6	0	2	1	0

Une séquence de nœuds solutions.

Dans (Qing et al. 2006), la représentation d'un individu est basée sur ce principe, pour lequel les nœuds représentant la base en extrémité sont supprimés et ceux au milieu sont décalés à gauche. Par conséquent un individu est représenté à l'aide d'un vecteur de dimension n + k -1. Voici la représentation qui correspond à l'individu précédent.

1	2	3	4	5	6	0	0
8	7	4	3	1	5	2	6

Fig. 25 Représentation d'un individu.

Les deux derniers emplacements dans index de la figure Fig. 25, représentent les positions de séparation des tournées. La méthode a été appelée seulement pour des problèmes de petites tailles, voir quelques nœuds et non pas à des jeux de données de la littérature. Il est bien évident que la recherche de la solution à base de la méthode proposée, consiste à trouver les meilleures positions dans le vecteur index. Ce qui n'est pas facile avec l'utilisation directe de la méthode OEP.

Méthode 2

Nous savions que la méthode OEP permet l'optimisation des fonctions continues dans un espace de recherche de dimension d à base d'une population d'individus. Chaque individu possède des paramètres de vitesse et d'accélération, qui lui permettent de voler dans un espace de recherche en ajustant la position en fonction des deux mémoires : sa meilleure position obtenue jusqu'à l'instant considéré, et la meilleure position dans la population.

Appliquer directement les équations de base de l'OEP pour l'optimisation discrète s'avère impossible, et il est nécessaire de procéder à la modélisation de problème en tenant compte de la philosophie de la méthode. C'est pour cette raison nous cherchons d'abord à réécrire les équations de base de l'OEP à base des deux mémoires : meilleure expérience dans la mémoire de l'individu et la meilleure expérience dans la mémoire de l'essaim. Reprenons le modèle classique de l'OEP :

$$x(t+1) = x(t) + v(t+1)$$
$$v(t+1) = wv(t) + c_1\varphi_1(pbest - x) + c_2\varphi_2(gbest - x)$$

Il est évident que les deux parties de l'accélération, permettent le changement dans la position de l'individu vers la meilleure position dans l'expérience, et la meilleure position obtenue dans la population par $c_1\varphi_1(\text{pbest} - x)$ et $c_2\varphi_2(\text{gbest} - x)$, respectivement.

Soit x, pbest, et gbest trois vecteurs, représentant des tournées de véhicule pour un problème de collecte et de livraison. Avec x : la solution actuelle. pbest : La meilleure expérience obtenue par l'individu. gbest : La meilleure solution obtenue par la totalité de l'essaim. Les deux équations seront modifiées comme suit :

- La solution actuelle x obtenue par un individu sera modifiée, en déplaçant l'individu vers sa meilleure position visitée pbest , c'est-à-dire son expérience par la grandeur suivante :

$$V_1 = c_1\varphi_1 d_1$$

- La nouvelle position de l'individu sera soumise à des changements à base de la deuxième part de l'accélération. Par conséquent l'individu est déplacé vers la meilleure position obtenue par l'essaim gbest , comme suit :

$$V_2 = c_2\varphi_2 d_2$$

Où d_1 et d_2 sont deux scalaires appartenant à l'intervalle $[0\ 1]$, qui implémentent la différence entre les couples de vecteurs (*pbest*, *x*) et (*gbest*, (*pbest*, *x*)) , et V_1 , V_2 sont les vitesses de déplacements.

Exemple

Soit X_1, X_2 deux vecteurs représentant les nœuds d'un graphe, supposant qu'un nœud est codé en un entier positif :

$$X_1 = (2,1,3,4,6,7,9,8,10,5), X_2 = (9,5,6,7,3,4,1,8,10,2)$$

Trouver une mesure de ressemblance ou de différence entre deux vecteurs peut s'effectuer à l'aide de la distance de Hamming. L'inconvénient majeur de cette distance est lorsque l'un des vecteurs est obtenu à travers le décalage circulaire de l'autre vecteur (pour un problème de tournées de véhicules, les deux vecteurs représentent les mêmes tournées). La distance de Hamming entre ces deux vecteurs est maximale, pourtant que ces deux derniers représentent la même solution.

Pour cela nous cherchons tout d'abord à trouver une mesure de ressemblance $R(X_2, X_1)$ entre deux vecteurs, cette mesure représente le pourcentage des mêmes

arrêtes appartenant au deux vecteurs. Formellement soit A_1 , A_2 les arrêtes représentant X_1 , et X_2.

$$A_1 = \{(2,1)\ (1,3)\ (3,4)\ (4,6)\ (6,7)\ (7,9)\ (9,8)\ (8,10)\ (10,5)\}$$
$$A_2 = \{(9,5)\ (5,6)\ (6,7)\ (7,3)\ (3,4)\ (4,1)\ (1,8)\ (8,10)\ (10,2)\}$$

$$R(X_2, X_1) = \frac{1}{N-1}\sum_{i=1}^{n} n_i \quad n_i = \begin{cases} 1\ si\ l'arrête\ i \in A_1\ et\ A_2 \\ 0 \qquad\qquad\qquad Sinon \end{cases}$$

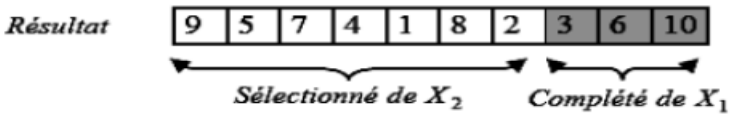
Dans l'exemple $R(X_2, X_1) = 1/3$, la différence entre les deux vecteurs est donc:

$$d(X_2, X_1) = 1 - R(X_2, X_1)$$

Nous dénommons cette distance la distance des paires. Dans l'exemple la différence entre les deux vecteurs est 2 / 3. Une fois la différence entre les deux vecteurs est calculée, la vitesse de changement de vecteur X_1 envers le vecteur X_2 est obtenue à l'aide de l'équation $V = c\varphi d$. Admettons que $c = 1$, et $\varphi = 1/2$, alors la vitesse de déplacement de vecteur X_1 envers X_2 est de $V = 1/3$.

Afin d'appliquer les changements sur le vecteur X_1 , une variable aléatoire ρ est générée pour chaque arrête dans X_2 . Si $\rho < V$ l'arrête en question est sélectionné pour participer dans le vecteur résultat. Si à la fin, le vecteur résultat n'est pas complet, celui-ci est complété à partir des éléments de X_1 , ce dernier est parcouru d'une manière circulaire à partir de dernier élément trouvé dans le vecteur résultat.

Admettons que ρ ={0.3, 0.5, 0.2, 0.9, 0.1, 0.2, 0.3, 0.7, 0.05}, le vecteur résultat sera :



Maintenant, pour calculer la nouvelle position X (déplacement vers pbest , gbest), il suffit d'appliquer la méthode sur les couples (pbest, x) et (gbest, (pbest, x)) .

4.2.4 Algorithme OEPPCL

Les étapes de l'algorithme qui suit, expliquent l'application de la méthode pour le problème de collecte et de livraison.

Algorithme

1. Initialisation
   a. Définir une taille pour l'essaim (nombre d'individus), et initialiser ($X.Pbest$) les individus aléatoirement.
   b. Initialiser les paramètres c_1, c_2, φ_1, φ_2
   c. Evaluer ces individus ; puis calculer le meilleur dans l'essaim, le $gbest$.

2. **While** (Nbe_Iteration ≠ Maximum) **Do**
3. **For** ($i=1$ *Taille Essaimes*) **Do** {
4. Evaluer $V_1 = c_1\varphi_1 d_1$, en déplaçant X vers $Pbest$ par V_1 .
5. Evaluer $V_2 = c_2\varphi_2 d_2$, en déplaçant le résultat obtenue, vers $Gbest$ par V_2 . Le résultat donnera X à l'instant $(t+1)$.
6. Modifier le vecteur résultat en ajoutant les nœuds de livraison à l'aide de la procédure ***Insertion_Noeuds***. Les résultats obtenus, représentent des solutions réalisables.
7. évaluer l'individu.
8. If $Eval(X) < Eval(pbest)$ alors Remplacer $pbest$ par X
9. If $Eval(X) < Min$ alors
10. $Min = Eval(X)$; $position = i$ (i : est l'individu d'ordre i)}
11. $Gbest$ = L'individu d'ordre $Position$.}

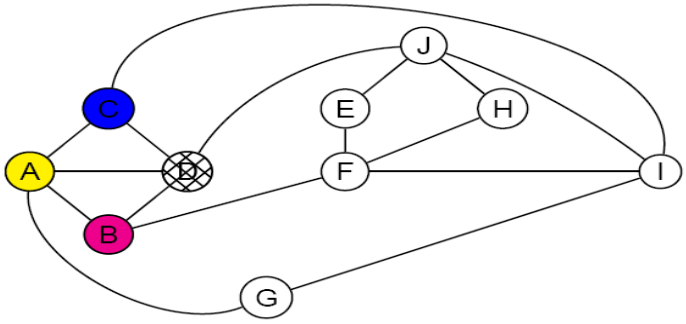
Annexe 1

Annexe I
Travaux Pratiques 2009-2014

Travaux Pratiques

TP1. K-Colorabilité

Etant donnée un graphe G et un entier K, peut-on colorier les sommets de G avec au plus K couleurs de façon à ce que deux sommets adjacents ne portent jamais la même couleur. L'exemple suivant nous donne une idée sur la colorabilité d'un graphe composé de dix sommets (A,B,…,G).



Il n'existe pas d'algorithme connu qui peut résoudre le problème en un nombre polynomial d'étapes. Soit K le nombre de couleur et n le nombre de sommet. Modéliser et implémenter le problème à base des AG.

TP2. Problème Sudoku

Le but de ce TP est de familiariser l'étudiant avec le concept d'algorithmes génétiques en appliquant le paradigme à la résolution du problème du Sudoku.

Le Sudoku est un jeu en forme de grille défini en 1979 par l'Américain Howard Garns, mais inspiré du carré latin, ainsi que du problème des 36 officiers du mathématicien suisse Leonhard Euler. Le but du jeu est de remplir la grille avec une série de chiffres allant de 1 à 9, qui ne se retrouvent jamais plus d'une fois sur une même ligne, dans une même colonne ou dans une même sous-grille en partant d'une grille partiellement vide. Ci-dessus un exemple de Sudoku rempli (à droite) et partiellement vide (à gauche).

		2		3	1			8
	4						1	3
8	1		7					
	3			5			6	2
4		7	3	8	6	9		1
6	5			2			8	
					5		7	9
2	7						3	
9			4	7		6		

4	1	2	3	6	7	8	5	9
6	5	8	9	4	1	3	2	7
3	9	7	5	2	8	1	6	4
9	6	4	2	3	5	7	8	1
7	3	5	1	8	4	6	9	2
8	2	1	7	9	6	4	3	5
1	4	9	6	5	3	2	7	8
5	8	3	4	7	2	9	1	6
2	7	6	8	1	9	5	4	3

Travail demandé

- Entraînez-vous sur une série de 10 Sudokus.
- Concevez et implémentez une structure de données pour une grille de Sudoku.
- Ecrivez une fonction capable de vérifier si une grille est correcte.
- Proposez un codage représentatif des grilles pour les individus d'un algorithme génétique.
- Proposez une fonction objective pour ce problème.
- Implémentez la sélection, le croisement et la mutation des individus.
- Modifiez les différents paramètres de votre algorithme de manière à avoir l'exécution la plus rapide possible (le minimum de génération).

TP3. Emploi du temps (time tableing)

La construction d'un emploi du temps est un travail long et difficile à mettre en place et qui doit être répéter à intervalle régulier. C'est pour cette raison que l'automatisation d'emplois du temps, est une tâche d'une grande importance car elle permet de gagner beaucoup d'heures de travail pour les employés, les institutions et les entreprises, et de fournir des solutions optimales avec satisfaction de contraintes en quelques minutes, ce qui peut accroître la productivité, la qualité de l'éducation, qualité des services et enfin, la qualité de vie. La difficulté de cette tâche fastidieuse et répétitive qui fait généralement intervenir de nombreux éléments d'informations, est liée à la nature des contraintes qu'il s'agit de satisfaire ces derniers.

L'existence d'un horaire respectant à la lettre toutes les contraintes n'est pas garantie dans le cas général. En réalité, il s'agit de trouver un horaire aussi bon que possible dans un temps de calcul raisonnable tout en tolérant la présence de conflits. Le nombre de ces conflits doit être minimisé pour que l'horaire soit jugé

satisfaisant par les différentes parties concernées (direction de l'université, Intervenant, Etudiants,…etc.).

Développez à l'aide du langage que vous préférez un programme permettant l'affectation des enseignants aux séances de cours, travaux dirigés et travaux pratiques.

TP4. Fourmis pour le voyageur de commerce

Implémentez en Java l'algorithme AS développé dans le chapitre n°5 pour le voyageur de commerce.

TP5. Emploi du temps (time tableing)

Implémentez en Java l'algorithme PSO développé dans le chapitre n°6 pour les fonctions cités dans le chapitre.

Annexe 2

EMD 2009-2014

Examen Final & Rattrapage

M é t a h e u r i s t i q u e

Exercice N°1

1. Donner une procédure qui cherche un élément x dans un tableau trié de taille N par la méthode de recherche dichotomique.
2. Démontrer que la complexité de l'algorithme est de l'ordre de $O(\log(n))$.

Exercice N°2

Soit la somme

$$s = C_n^0 + C_n^1 + ... + C_n^{n-1} + C_n^n$$

1. Proposer un algorithme (fonction) de complexité $O(nm)$ qui calcul la combinaison C_n^m en ne faisant aucune opération de multiplication.

2. utiliser cette fonction pour calculer la somme s, évaluer la complexité de votre calcul.

Problème

Dans ce problème, nous cherchons à implémenter un type de problème de tournées de véhicules à base des algorithmes génétiques. Le problème consiste à satisfaire un ensemble des demandes client, appelées requêtes. Plus exactement le transport des personnes à partir des sites dites « de collectes » et de les déposer dans des sites de destination dites « de livraison ». Le traitement d'une requête k d'un client, se résume dans la transportation du client d'un site de collecte noté p_k vers un site de livraison noté d_k. Le véhicule a une capacité limitée de places, soit a le nombre maximum de places. Le véhicule se déplace par une vitesse moyenne de 80 km/h, et possède une autonomie de circulation allant jusqu'à 150 km. Au delà de cette distance, le véhicule doit retourner à la base pour se ravitailler en carburant.

Ce problème contient les contraintes suivantes :

- Pour chaque demande de transport k, le site de collecte p_k et le site de livraison d_k doivent être visités dans la même tournée.

- Pour une demande de transport k, le site de collecte p_k doit être visité avant le site de livraison d_k

- Le nombre de passagers dans le véhicule ne doit pas dépasser a .

- Étant donné que le véhicule a une durée de circulation limitée T au bout de laquelle il doit retourner se ravitailler en carburant à sa base et qu'il ne peut retourner à la base s'il a des passagers à son bord, la durée des tournées est elle aussi limitée à T .

L'objectif de ce problème est de satisfaire toutes les demandes de transport en minimisant la distance totale parcourue.

Pour faciliter la tâche nous donnons en premier la codification qui sera utilisée dans la suite du problème. Un chromosome solution est représenté à l'aide de la liste des demandes de transport indexées dans laquelle chaque demande de transport k apparaît deux fois : une fois comme +k qui indique le nœud de collecte et l'autre fois comme -k pour le noeud de livraison. Par exemple avec quatre demande de transport, la représentation d'une solution est donnée comme suit :

0 +1 +3 -3 -1 0 +2 -2 0 +4 -4 0

0 représente la base, le véhicule commence à traiter la demande n°1 et n°3 dans la même tournée, puis la demande n°2 et n°4 dans deux tournées différentes, donc la solution est composée de trois tournées

Travail demandé

1. La représentation d'un chromosome solution, décrite ci-dessus (représentation par demande de transport avec séparateur de tournées) pose un problème lors de l'application des opérateurs génétiques. A partir de cette représentation donner la codification d'un chromosome uniforme (un chromosome sans découpage en tournées).

2. Donner une procédure qui permet d'initialiser une population de taille N .

3. Soit Eval(Chrom(i)), une procédure qui permet d'évaluer le chromosome d'ordre i . Expliquer et Donner une procédure de sélection à base de la méthode de sélection « roue de loterie ».

4. Donner deux procédures implémentant l'opérateur de croisement avec un (01) et deux (02) points de coupures.

5. En cherche maintenant à implémenter la procédure Eval, qui permet l'évaluation de la population. Présenter les différentes structures de données nécessaires, puis décrire brièvement la procédure Eval, en déduire la complexité.

Exercice N°3

Il s'agit de développer une application qui calcul la liste de tous les nombres premiers ≤N, par la méthode dite de « crible d'Erathosthène », dont on rappelle ici le principe :

1. On écrit la liste de tous les entiers allant de 2 jusqu'à N.

2. On garde 2 et on élimine tous les autres multiples de 2.

3. On garde 3 qui est le premier élément non éliminé après 2 et on élimine tous les autres multiples de 3.

4. On garde 5 qui est le premier élément non éliminé après 3 et on élimine tous les autres multiples de 5.

5. On réitère le procédé jusqu'à fin.

- Ecrire une procédure qui permet de calculer la liste de tous les nombres premiers ≤N. Evaluer la complexité de votre calcul ?

- Peut on trouver un algorithme qui nous permet d'améliorer la complexité de calcul trouvée dans la première question, si oui donner la procédure de calcul puis prouvez la complexité de l'algorithme ?

Problème

On cherche à développer dans ce problème un algorithme génétique binaire basé sur l'évolution d'une population de N individus, chacun d'eux formés d'une chaîne de N bits de 0 ou de 1.

La méthode de sélection est basée sur le rang de chaque individu dans la population et sur le principe de tirage aléatoire biaisé. En classant les individus de 1 à N en fonction de leur valeur par la fonction J (l'individu N correspond à celui qui a la plus petite valeur par J), une nouvelle population de N individus est créée en tirant à chaque fois l'individu i avec la probabilité :

$$P_i = \frac{i}{\sum_{i=1}^{N} i}$$

La probabilité de croisement et de mutation sont P_c et P_m respectivement.

1. Construire un algorithme génétique reprenant les principes décrits plus haut et ayant une populations de N individus et Nb générations et dont la taille de chaque individus est de L bits.

2. Appliquer cet algorithme génétique à la recherche du minimum de la fonction f(i) = le nombre de 1 dans un individu i dont le minimum global est évidemment (0,…,0) avec $N = 40,\quad Nb = 100,\quad L = 20,\quad P_c = 0.65,\quad P_m = 0.05$.

3. Montrer à l'aide d'un exemple de six (individu : 1 jusqu'à 6) individus et dont la taille de chaque individu est de dix bits. Le principe de la sélection montré ci-dessus.

Si on considère l'algorithme génétique précédent pour la minimisation d'une fonction réelle à n variables sur un intervalle $[a\ b]^n$.

1. Expliquez comment peut on représenter à l'aide d'un codage binaire un réel défini ainsi.

2. Donner une équation qui va permettre le passage entre les deux codages (binaire – réel).

3. Ecrire une version d'algorithme précédent permettant de minimiser la fonction $f(x) = \left\| x^2 \right\|$ avec $x \in [-2,\quad 2]$.

Exercice N°4

Ecrire un algorithme (fonction ou procédure) qui permet d'effectuer la multiplication de deux entiers non nuls par sommation. Quels est le nombre d'opération de (+) effectué dans votre algorithme. Evaluer la complexité de votre calcul.

11	10
5	20
2	~~40~~
1	80
	110

Il existe une méthode de multiplication appelée « Multiplication à la russe » : Dans la colonne gauche, on divise par deux en prenant la partie entière et on s'arrête à 1. Dans la colonne de droite, on double successivement chaque nombre.

On raye à droite tous les chiffres en face d'un nombre pair. On fait la somme des nombres de droite restants.

Ecrire un algorithme qui implémente la méthode, puis donner la complexité.

Exercice N°5

Soit à minimiser la fonction f(x)=x2-1, à l'aide d'un algorithme génétique, avec $x \in [0.0,\ 10.0]$, quelle est la taille nécessaire pour représenter un individu en binaire. Donner la représentation binaire des individus ayant pour valeur 0.5, 1.2, 2.2, 6.5, 7.6, 9.9. Donner l'évaluation puis l'adaptation de ces individus.

1. Ecrire l'algorithme qui implémente la méthode de sélection « Roue de loterie », puis dérouler cet algorithme sur les individus cités dans la première question.

2. Quelle est la complexité de votre algorithme de sélection ?

Exercice N°6

1. Ecrire un algorithme de sélection à base de la méthode de tournoi.

2. Ecrire un algorithme de sélection à base de la méthode de l'élitisme.

Problème

Les algorithmes génétiques réels sont adaptés à la recherche du minimum d'une fonction f définie de $R^n \to R$. Par rapport aux algorithmes génétiques binaires, le seul changement concerne les deux principes de croisement et de mutation.

Pour le croisement, cas réel, à partir de deux parents p_1 et p_2, deux enfants e_1 et e_2 sont crées avec une probabilité p_c en sélectionnant aléatoirement $\alpha \in [0\ \ 1]$ et en écrivant : $e_1 = \alpha p_1 + (1-\alpha) p_2$, $e_1 = (1-\alpha) p_1 + \alpha p_2$. Ecrire un algorithme de croisement réel.

Pour la mutation, à partir d'un enfant e, on crée, avec une probabilité p_m un nouvel élément $e' = e + \sigma\mu$ où μ est un vecteur aléatoire suivant une loi normale centrée réduite et σ ¾ est un réel strictement positif fixé. Ecrire un algorithme de mutation réel.

Ecrire un algorithme génétique réel permettant de rechercher le minimum d'une fonction $g(x) = \|x\|^2$ définie sur $[-2,2]^4$.

Reprendre les questions précédentes, si on considère un algorithme génétique binaire qui cherche à minimiser une fonction à n variable définie dans $[a\ \ b]^n$.

Exercice N°7

Soit $G(X,E)$ un graphe composer de n sommets, avec X l'ensemble des sommets et E l'ensemble des arcs. Donner un algorithme non déterministe qui test si les n sommets du graphe constituent un cycle hamiltonien. Il est demandé de traiter les deux cas où le graphe est orienté ou non orienté.

Exercice N°8

On demande d'écrire à l'aide d'une boucle tant que et de l'opération d'addition une fonction *racine* qui calcule la racine carrée entière par défaut d'un nombre entier positif n donné. La racine carrée entière r d'un nombre n vérifie la relation d'ordre $r^2 \le n \le (r+1)^2$. On rappelle que la somme des r premiers impaires est égale au carré de r.

- Ecrire une version récursive de la fonction *racine*.

- Pour les deux fonctions développées, calculer le nombre d'addition et de comparaisons effectuées.

Exercice N°9

Dans cet exercice on s'intéresse à l'implémentation de l'opérateur de croisement utilisé dans les algorithmes génétiques (AG). Le codage utilisé est le codage binaire.

1. Ecrire une fonction *One-Point Crossover* qui permet de croiser entre deux vecteurs parents pour produire deux vecteurs enfants. la taille des vecteurs est l.

2. Ecrire une fonction *Two-Point Crossover* qui permet de croiser entre deux vecteurs parents pour produire deux vecteurs enfants. la taille des vecteurs est l.

Le croisement uniforme *Uniform Crossover* est une autre technique qui permet de traiter le croisement avec appel au caractère aléatoire au niveau de chaque bit et non pas aux points des coupures.

3. Ecrire une fonction *Uniform Crossover* qui permet de croiser entre deux vecteurs parents pour produire deux vecteurs enfants.

4. Proposer un opérateur de croisement entre K vecteurs qui permet de produire K enfants.

Exercice N°10

On appelle distance d'édition, ou distance de ***Levenshtein*** entre deux mots : *M1* et *M2*, le coût minimal pour aller de *M1* à *M2* en effectuant les opérations de base suivantes : remplacement d'un caractère de M1 en un caractère de M2, Ajout

dans M1 d'un caractère de M2 et suppression d'un caractère de M1. On associe à chacune de ces opérations un coût égal à 1 (par exemple). Par exemple si M1= « Exercice1 » et M2= « Exercice2 » alors DE(M1,M2)=1 (remplacement de 1 par 2).

1. Ecrire un algorithme permettant de retourner la distance DE(M1,M2), puis évaluer la complexité de votre calcul. M1 et M2 sont deux chaines de caractère de longueur n et m respectivement.
2. Dérouler votre méthode sur les deux chaines M1= « NICHE » et M2=« CHIENS », intuitivement la première chaine peut se transformer en deuxième chaine en cinq (05) opérations : effacement de N puis effacement de I, Ajout de I de N puis de S. donc la distance DE(NICHE, CHIENS)=5.

Exercice N°11

Le jeu de la tour de Hanoï est constitue de 3 piquets et d'une série de n disques de diamètres différents. Initialement, tous les disques sont empilés les uns sur les autres autour d'un même piquet et forment ainsi une tour. L'objectif du jeu est d'arriver à transférer cette tour sur un autre piquet en respectant les règles suivantes : 1) on ne peut déplacer qu'un seul disque à la fois, d'un piquet vers un autre ; 2) seul le disque sur le dessus d'une

pile peut être déplacé ; 3) aucun disque ne peut reposer sur un autre disque de diamètre inférieur.

1. Dérouler les règles de jeux pour n = 3, et n=4.

2. Démontrer que la complexité de l'algorithme Hanoï est exponentielle.

Exercice N°12

Soit la fonction f(x)= 1- (x-1)2, on cherche à trouver le maximum de cette fonction à l'aide d'un AG sur l'intervalle $[-10\ \ 10]$.

Donner le nombre de bits nécessaire pour coder un individu de la population.

Donner la représentation en binaire des valeurs de variable x=-4, x=-2 x=0, x=6, x=8 et x=10.

Donner sous une forme tabulaire les valeurs de la fonction d'évaluation, puis les probabilités de sélection de chaque individu.

Démontrer que la probabilité de sélection d'un individu est de la forme $\frac{1}{n-1}\left(1-\frac{f(x_i)}{\sum_1^n f(x_i)}\right)$.

Donner un algorithme qui implémente la méthode de sélection par roue de loterie.

Bibliographies

(Vob et al. 1999) S. Voß, S. Martello, I.H. Osman and C. Roucairol (Eds), "Meta-Heuristics - Advances and Trends in Local Search Paradigms for Optimization". Kluwer Academic Publishers, Dordrecht, The Netherlands, (1999)

(Goldberg, 1989) D.E. Goldberg, " Genetic Algorithms in Search, Optimization and Machine Learning", Addison-Wesley, 1989.

(Perifel, 2013) Sylvain Perifel, "Théorie de la complexité" 2013. http://creativecommons.org/licenses/by-nc-sa/3.0/fr/

(Gilbert, 2006) Jean-Charles Gilbert « Chap 6. Méthodes à direction de la descente ». pages 241-272, https://who.rocq.inria.fr/Jean-Charles.Gilbert/ensta/06-rl.pdf

(Blum et al., 2003) Christian Blum, Andrea Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. ACM Computing Surveys, Vol. 35, No. 3, September 2003, pp. 268–308.

(Mendes, 2004) R. Mendes, "Population topologies and their influence in particle

(Mendes et al. 2004) R. Mendes , J. Kennedy , J. Neves, "The fully informed particle swarm: Simpler maybe better". In IEEE, Evolutionary Computation, Vol. 8 (3), 2004, pp: 204-210.swarm performance". Ph.D. Thesis Universidade do Minho, Lisbon, Portugal.

(Clerc et al., 2002) M. Clerc, J. Kennedy "The particle swarm: Explosion, stability, and convergence in a muli-dimensional search space" IEEE, Computational Intelligence, Vol 6, pp. 58-73.

(Clerc, 2006) M. Clerc M. "some idea about particle swarm optimization", http://www.mourice.free.fr/PSO .

(Colorni et al. 1991) A. Colorni, M. Dorigo, V. Maniezzo, "Distributed Optimization by Ant Colonies" . In Varela and Bourgine, pp. 134–142.

(Colorni et al. 1992) A. Colorni, M. Dorigo, V. Maniezzo, "Investigations of some properties of an "Ant algorithm"". In Männer and Manderick, pp. 509-520.

(Dorigo et al., 1996a) M. Dorigo, V. Maniezzo, and A. Colorni. "The ant system: Optimization by a colony of cooperating ants". IEEE, Transactions on Systems, Management and Cybernetics, 26, pp.1–13.

(Gambardella et al., 1996) L. Gambardella, M. Dorigo. "Solving Symmetric and Asymmetric TSPs by Ant Colonies". Proceeding of the IEEE, Transactions on Evolutionary Computation, Japan.

(Gambardella et al. 1999) L. M. Gambardella, E. Taillard, and M. Dorigo. "Ant Colonies for the QAP". Journal of Operational Research Societies, Vol. 50(2), pp.167-176.

(Maniezzo et al., 1999) V. Maniezzo , A. Colorni, "The Ant System Applied to the Quadratic Assignment problem". In IEEE Transactions on Knowledge and Data Engineering. Vol. 11(5), pp.769-778.

(Goss et al., 1989) S. Goss, S. Aron, J. Deneubourg, and J. Pasteels. "Self-Organized Shortcuts in the Argentine Ant". Naturwissenchaften, Vol. 76, pp.579–581.

(Holldobler et al., 1990) B. Holldobler, E. Wilson, "The Ants". Springer, Berlin.

(Gendreau et al. 2003) M. Gendreau, J-Yves, Potvin « Tabou Search » Cours de Département d'informatique et de recherche operationnelle, Centre de recherche sur les transports, Universite de Montreal, Canada 2003.

(Glover 1986), Fred Glover "Future Paths for Integer Programming and Links to Artificial Intelligence". Computers and Operations Research" , v. **13**,5, pages 533–549

(Scheid 2011). J.-F. Scheid « Graphes et Recherche Opérationnelle » , notes de cours 2011.

(Luke 2013). Sean Luke « Essentials of Meta heuristics » , Department of Computer Science George Mason University, Second Edition, Online Version 2.0, June, 2013

(Grassé, 1959) P. Grassé. "La reconstruction du nid et les coordinations interindividuelles chez bellicositermes et cubitermes. La théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs". Insectes Sociaux , 6, pp.41–81.

(Delgado et al., 1997) J. Delgado and R. Sole. "Collective-induced computation". Physical Review , Vol. 55, pp.2338-2344.

(Bonabeau et al., 1999) E. Bonabeau, M. Dorigo, G. Theraulaz, "Swarm Intelligence: From Natural to Artificial Systems". New York: Oxford University Press.

(Axelrod, 1997) R. Axelrod "The dissemination of culture: A model with local convergence and global polarization". Journal of Conflict Resolution, Vol. 41, pp.203–226.

(Heppner et al., 1990) F. Heppner, U. Grenander "A stochastic nonlinear model for coordinated bird flocks". In The Ubiquity of Chaos. American Association for the Advancement of Science, pp.233-238.

(Sheloker et al. 2006) P. S. Shelokar, P. Siarry , V. Jayaraman , B. D. Kulkarni. "Particle swarm and ant colony algorithm hybridized for improved continuous optimization". Applied Mathematics and Computation. Vol. 188(1), pp.129-142.

(Reynolds, 1987) C.W. Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioral Model". Computer Graphics, 21, pp. 25–34.

(Van der bergh et al., 2006) F. Van der bergh , A. Engelbercht , "A study of particle swarm optimization particle trajectories". Information Science, Vol. 176, pp.937-971.

(Kennedy et al., 1995) J. Kennedy , and R. Eberhart R., "Particle swarm optimization". In IEEE, International Conference on Neural Network, pp.1942-1948.

(Kennedy, 1997) J. Kennedy. "The particle swarm: Social adaptation of knowledge" Proceedings of the International Conference on Evolutionary Computation Indianapolis, Indiana. pp. 303–308.

(Kennedy, 1999) J. Kennedy. "Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance". Proceedings of the Congress on Evolutionary Computation, pp.1931–1938.

(Kennedy et al., 2001) J. Kennedy, R.C. Eberharte, et Y. Shi. "Swarm Intelligence". Morgan Kaufmann Publishers.

(Kennedy et al., 2003) J. Kennedy , R. Mendes R., "Neighborhood topologies in fully informed and best of neighborhood particle swarm". In IEEE, Proceedings of the International Workshop, Soft Computing in Industrial Applications, pp. 45-50.

(Janson et al., 2005) Janson S., and Middendorf M., "A hierarchical Particle swarm optimizer and its adaptive variant". In IEEE System Managment and Cybernetic, Vol.35 (6) , pp.1272-1282.

(Qing et al. 2006) Z. Qing, Q. Limin, L. Yingchun, Z. Shanjun. "An improved particle swarm optimization algorithm for vehicle routing problem with time window". In IEEE, Congress on Evolutionary Computation, CEC'06, pp.1386-1390.

http://www.metaheuristics.net

http://fr.wikipedia.org/wiki/Direction_de_descente