INF6102 – Métaheuristiques

Travaux pratiques n°5 – Covering Array

*Kévin Baumann – 1647505  
Florian Korsakissok - 1628087*

# I – Présentation du problème

Le problème traité est celui du « Covering Array », ou autrement dit d’une matrice de couverture. La résolution d’une instance d’un tel problème dépend de deux paramètres notés v et k. L’objectif est de remplir une matrice contenant k colonnes avec un minimum de lignes, à l’aide d’entiers compris entre 0 et v-1, de telle sorte que pour chaque paire de symboles, et pour chaque paire de colonnes, au moins une ligne de la matrice contienne cette paire de symboles sur cette paire de colonnes.

Plus formellement, le problème peut être défini comme suit :

* Données d’entrée : v, k
* Sortie de l’algorithme : une matrice M à coefficients dans [0 ; v-1] de taille N.k
* Contraintes : pour tout i, j dans [0 ; k-1], pour tout a, b dans [0 ; v-1], il existe une ligne l telle que M[l][i] = a et M[l][j] = b
* Objectif : minimiser N

N est la grandeur à minimiser : on veut se retrouver avec la matrice la plus petite possible, c’est-à-dire écrire aussi peu de lignes que possible dans le résultat. De ce fait, N peut être vu comme un coût dans le cadre de ce problème.

Dans l’implémentation de l’algorithme proposé, on utilisera souvent la notion de « contrainte élémentaire ». Une contrainte élémentaire est un quadruplet (i,j,a,b), et on dira que cette contrainte est satisfaite si il existe dans la matrice en cours de construction une ligne l telle que M[l][i] = a et M[l][j] = b.

# II – Description des techniques de résolution

## A – Glouton randomisé avec relances (Gr)

|  |  |
| --- | --- |
| **Solution partielle** | Toute matrice de largeur k à valeurs dans [0 ; v-1] qui ne remplit pas encore toutes les contraintes. |
| **Configuration initiale** | Matrice de taille 0.k. |
| **Candidat S** | Tout vecteur ligne de longueur k, à valeurs dans [0 ; v-1]. |
| **Score d’un candidat** | Nombre de contraintes satisfaites parmi celles qui ne sont pas encore résolues. |
| **Complétion de la configuration** | A chaque étape, on insère la meilleure ligne candidate en bas de la matrice S. |
| **Technique de randomisation** | Introduction d’un paramètre de tolérance α pour la constitution d’une liste restreinte de candidats. On fixe alors un seuil de résolution de nouvelles contraintes à partir duquel une ligne peut être candidate, tel que **seuil = scoreMax\*(1-α)**. Les candidats ainsi sélectionnés font l’objet d’un tirage uniforme. |
| **Paramètres de l’algorithme** | α : paramètre de tolérance pour la randomisation. |
| **Remarques** | Le glouton pur correspond à un α nul, tandis que le hasard parfait est atteint pour α=1. En dehors de ces valeurs, si, par exemple, α=1/2, et que lors d’une itération la meilleure ligne candidate résout 100 nouvelles contraintes, alors toutes les lignes satisfaisant au moins 50 contraintes seront candidates. |

## B – Algorithmes de recherche locale

|  |  |
| --- | --- |
| **Configuration** | Toute matrice de taille N.k contenant des symboles entiers dans [0 ; v-1]. |
| **Coût d’une configuration** | Nombre de contraintes élémentaires violées. |
| **Mouvement** | Un mouvement consiste à remplacer le symbole a contenu dans la matrice à la ligne l et à la colonne c par un nouveau symbole b. |
| **Calcul du coût d’un mouvement** | Sur la ligne l, pour chaque couple (a,d), où d est n’importe quel autre symbole, s’il y a exactement une occurrence, on compte une erreur en plus (une contrainte sera non satisfaite). Pour chaque couple (b,d), s’il n’y a aucune occurrence, on compte une erreur de moins. |
| **Complexité d’une itération** | O(k) : parcours de toutes les colonnes de la ligne de la case qui va changer. |

## C – Descente avec relances aléatoires (Dsc)

|  |  |
| --- | --- |
| **Configuration initiale** | Matrice de taille N.k obtenue aléatoirement. |
| **Sélection d’un mouvement** | Sélection du mouvement résolvant le plus de nouvelles contraintes. La nouvelle configuration doit avoir un coût strictement inférieur à la précédente. |

## D – Recuit simulé (SA)

|  |  |
| --- | --- |
| **Configuration initiale** | Matrice de taille N.k obtenue aléatoirement. |
| **Mouvement aléatoire** | Tirage au sort uniforme des paramètres a, b, c et l pour définir le mouvement effectué. |
| **Paliers** | Oui. |
| **Décroissance de la température** | Décroissance géométrique de facteur ½ au bout de 1000 itérations sans qu’une meilleure solution n’ait été trouvée. |
| **Critère d’arrêt** | - Exposition d’une solution de coût nul  - Temps d’exécution limite atteint |
| **Paramètres de l’algorithme** | T0 : température initiale  α : facteur de décroissance de la température  nbIterations : nombre d’itérations sans amélioration avant décroissance de la température |

## E – Recherche avec tabou (TS)

|  |  |
| --- | --- |
| **Configuration initiale** | Matrice de taille N.k obtenue aléatoirement. |
| **Sélection d’un mouvement** | Exécution de tous les mouvements possibles selon un ordre lexicographique. |
| **Principe de la liste taboue** | Interdiction de certains états pour la matrice pendant un nombre donné d’itérations. |
| **Contenu de la liste taboue** | Matrice cubique de dimensions N.k.v. Tabou[l][c][w] = i signifie que le symbole w est interdit dans la solution à la ligne l, colonne c, jusqu’à l’itération i. |
| **Mise à jour de la liste** | Lorsqu’à l’itération j un mouvement implique un passage de M[l][c] = a à M[l][c] = b, on bannit l’état que l’on vient de quitter pendant q itérations, c’est-à-dire Tabou[l][c][a] = j+q. |
| **Critère d’arrêt** | - Exposition d’une solution de coût nul  - Temps d’exécution limite atteint |
| **Paramètres de l’algorithme** | longueurListe : longueur de la liste taboue  K et K’ pour la diversification |
| **Diversification (pour TSD)** | Au cours d’un cycle, exécution de l’algorithme tabou pendant K secondes, puis phase de diversification pendant K’ secondes. Lors d’une phase de diversification, on prend en compte la fréquence d’utilisation du nouveau symbole dans la configuration en cours dans l’optique de favoriser les symboles les moins présents.  δ’(a,b) = δ(a,b) + h(b) – h(a) ; où h(x) est la fréquence d’apparition du symbole x. |

## F – Algorithme évolutionnaire (EA)

|  |  |
| --- | --- |
| **Type d’algorithme évolutionniste** | (μ+λ)-ES. |
| **Population initiale** | Construction aléatoire d’une population de m parents initiaux. |
| **Taille population** | Population constituée de m parents, générant n enfants, puis sélection des meilleurs individus pour se ramener à un effectif de m. |
| **Génération** | Deux parents génèrent un enfant. |
| **Sélection parents** | Les m meilleures configurations parmi les (m+n) individus sont sélectionnés. |
| **Opérateur de variation** | Application du croisement n fois, puis mutation de tous les enfants. |
| **Remplacement** | Recouvrement : oui. |
| **Opérateur de recherche locale** | Non. |
| **Opérateur de mutation** | 0.01% des symboles d’une configuration enfant, choisis aléatoirement, sont remplacés |
| **Opérateur de croisement** | Spécialisé : construction de l’enfant en choisissant, ligne par ligne, entre le parent A et le parent B, la ligne maximisant le nombre de nouvelles contraintes résolues en l’insérant dans l’enfant. |
| **Paramètres de l’algorithme** | tailleParents : nombre de parents sélectionnés dans une population  tailleEnfants : nombre d’enfants générés lors du croisement  pourcentMutation : pourcentage du nombre de symboles changeant lors d’une mutation |

# III – Résultats expérimentaux

## A – Présentation des données utilisées

Les données utilisées à des fins de test sont constituées de 7 exemplaires, chacun correspondant à un couple (v, k) différent. Les cas traités par la suite sont les suivants :

* v = 2, k = 4 - au mieux, avec l’algorithme tabou, on a pu avoir N = 5 en 60s.
* v = 3, k = 20 - au mieux, avec l’algorithme tabou, on a pu avoir N = 17 en 60s.
* v = 3, k = 60 - au mieux, avec l’algorithme tabou, on a pu avoir N = 22 en 60s.
* v = 5, k = 10 - au mieux, avec l’algorithme tabou, on a pu avoir N = 37 en 60s.
* v = 8, k = 10 - au mieux, avec l’algorithme tabou, on a pu avoir N = 94 en 60s.
* v = 8, k = 15 - au mieux, avec l’algorithme tabou, on a pu avoir N = 109 en 60s.

## B – Spécifications des machines de test

### 1 - Machine 1

Coefficient trouvé par dfmax : 5.40 🡺 coefficient d'ajustement : 8.6/5.4 = 1.59

|  |  |
| --- | --- |
| **Système d’exploitation** | Ubuntu 13.10 64 bits |
| **Mémoire vive** | 7.7 GiB |
| **Processeur** | Intel® CoreTM i7-3610QM CPU @ 2.30GHz × 8 |

### 2 - Machine 2

Coefficient trouvé par dfmax : 12.4 🡺 coefficient d'ajustement : 8.6/12.4 = 0.69

|  |  |
| --- | --- |
| **Système d’exploitation** | Windows 8.1 64 bits |
| **Mémoire vive** | 6.0 GiB |
| **Processeur** | Intel® CoreTM i7-2670QM CPU @ 2.20GHz |

## C – Valeurs des paramètres

|  |  |  |
| --- | --- | --- |
| **Paramètres Gr** | α = 0.3 | L’algorithme glouton pur étant plus efficace qu’un glouton avec une trop forte part d’aléatoire, un seuil de tolérance peu élevé a été retenu. |
| **Paramètres SA** | T0 = 0.1  α = 0.5  nbIterations = 1000 | Les paramètres liés à la température ont été calibrés suite à sept expériences à température constante. L’analyse de la fonction de coût selon le nombre d’itérations a permis de déterminer T0. |
| **Paramètres TS/TSD** | longueurListe = 20 ; 30 ; 15 ; 30 ; 50  K = 9 ; K’ = 1 | Des expériences préliminaires ont permis de fixer, pour chaque exemplaire, la meilleure valeur pour la longueur de la liste taboue. |
| **Paramètres EA** | tailleParents = 20  tailleEnfants = 20  pourcentMutation = 0.0001 | Des tests préliminaires ont montré qu’une population trop importante impliquait un temps de croisement trop long. Par ailleurs, le pourcentage de symboles mutés a été choisi de manière à préserver une certaine diversité, sans trop dégrader les solutions. |

## D – Résultats du glouton (Gr)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **v** | **k** | **N min** | **N moy** | **N max** | **Tps moy** | **It Moy** | **It/sec** | **EcartType** | **NbTests** |
| 3 | 20 | 25 | 26.4 | 27 | 200 | 400392 | 2001,96 | 0.663325 | 10 |
| 3 | 60 | 40 | 41.3 | 42 | 200 | 44621,2 | 223,106 | 0.640312 | 10 |
| 5 | 10 | 65 | 67.8 | 71 | 200 | 283757 | 1418,785 | 1.66132 | 10 |
| 8 | 10 | 209 | 215.3 | 222 | 500 | 168331 | 336,662 | 3.92556 | 10 |
| 8 | 15 | 240 | 252.8 | 260 | 500 | 81102,1 | 162,2042 | 5.6 | 10 |

## E – Résultats de la descente (Dsc)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **v** | **k** | **N min** | **N moy** | **N max** | **Tps moy** | **It Moy** | **It/sec** | **EcartType** | **NbTests** |
| 3 | 20 | 18 | 18.6 | 19 | 200 | 1437730 | 7188,65 | 0.489898 | 10 |
| 3 | 60 | 25 | 25.9 | 26 | 200 | 176674 | 883,37 | 0.3 | 10 |
| 5 | 10 | 43 | 43 | 43 | 200 | 891066 | 4455,33 | 0 | 10 |
| 8 | 10 | 115 | 115 | 115 | 500 | 523694 | 1047,388 | 0 | 15 |
| 8 | 15 | 134 | 134.7 | 135 | 500 | 252534 | 505,068 | 0.458258 | 10 |

## F – Résultats du recuit simulé (SA)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **v** | **k** | **N min** | **N moy** | **N max** | **Tps moy** | **It Moy** | **It/sec** | **EcartType** | **NbTests** |
| 3 | 20 | 17 | 17.7 | 18 | 200 | 9,28E+07 | 463913 | 0.458258 | 10 |
| 3 | 60 | 23 | 23 | 23 | 200 | 3,58E+07 | 178942,5 | 0 | 10 |
| 5 | 10 | 38 | 38 | 38 | 200 | 1,59E+08 | 792790 | 0 | 15 |
| 8 | 10 | 95 | 96.7 | 98 | 500 | 4,03E+08 | 805268 | 0.781025 | 10 |
| 8 | 15 | 110 | 111.3 | 112 | 500 | 3,01E+08 | 602764 | 0.640312 | 10 |

## G – Résultats de l’algorithme tabou (TS)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **v** | **k** | **N min** | **N moy** | **N max** | **Tps moy** | **It Moy** | **It/sec** | **EcartType** | **NbTests** |
| 3 | 20 | 17 | 17 | 17 | 200 | 3,49E+06 | 17458,9 | 0 | 10 |
| 3 | 60 | 22 | 22 | 22 | 200 | 714508 | 3572,54 | 0 | 10 |
| 5 | 10 | 37 | 37.7 | 38 | 200 | 2,68E+06 | 13406,7 | 0.458258 | 10 |
| 8 | 10 | 93 | 93.2 | 94 | 500 | 1,71E+06 | 3420,78 | 0.4 | 10 |
| 8 | 15 | 109 | 109.3 | 110 | 500 | 1,13E+06 | 2255,62 | 0.458258 | 10 |

## H – Résultats de l’algorithme tabou avec diversification (TSD)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **v** | **k** | **N min** | **N moy** | **N max** | **Tps moy** | **It Moy** | **It/sec** | **EcartType** | **NbTests** |
| 3 | 20 | 17 | 17 | 17 | 200 | 332614 | 1663,07 | 0 | 10 |
| 3 | 60 | 22 | 22 | 22 | 200 | 6866211 | 34331,055 | 0 | 19 |
| 5 | 10 | 38 | 38 | 38 | 200 | 315158 | 1575,79 | 0 | 10 |
| 8 | 10 | 94 | 94 | 94 | 500 | 183446 | 366,892 | 0 | 19 |
| 8 | 15 | 110 | 110.2 | 111 | 500 | 128706 | 257,412 | 0.4 | 10 |

## I – Résultats de l’algorithme évolutionnaire (EA)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **v** | **k** | **N min** | **N moy** | **N max** | **Tps moy** | **It Moy** | **It/sec** | **EcartType** | **NbTests** |
| 3 | 20 | 18 | 18.1 | 19 | 200 | 33033,2 | 165,166 | 0.3 | 10 |
| 3 | 60 | 25 | 25.4 | 27 | 200 | 3821,9 | 19,1095 | 0.663325 | 10 |
| 5 | 10 | 40 | 40.5 | 41 | 200 | 35597,3 | 177,9865 | 0.5 | 10 |
| 8 | 10 | 107 | 107.1 | 108 | 500 | 25226,1 | 50,4522 | 0.3 | 10 |
| 8 | 15 | 131 | 133.7 | 136 | 500 | 11731,7 | 23,4634 | 1.26886 | 10 |

## J – Synthèse

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Exemplaire** | | | **Gr** | | **Dsc** | | **SA** | | **TS** | | **TSD** | | **EA** | |
| **v** | **k** | **Best** | **Min** | **Avg** | **Min** | **Avg** | **Min** | **Avg** | **Min** | **Avg** | **Min** | **Avg** | **Min** | **Avg** |
| 3 | 20 | 17 | 25 | 26,4 | 18 | 18,6 | 17 | 17,7 | 17 | 17 | 17 | 17 | 18 | 18,1 |
| 3 | 60 | 22 | 40 | 41,3 | 25 | 25,9 | 23 | 23 | 22 | 22 | 22 | 22 | 25 | 25,4 |
| 5 | 10 | 37 | 65 | 67,8 | 43 | 43 | 38 | 38 | 37 | 37,7 | 38 | 38 | 40 | 40,5 |
| 8 | 10 | 93 | 209 | 215,3 | 115 | 115 | 95 | 96,7 | 93 | 93,2 | 94 | 94 | 107 | 107,1 |
| 8 | 15 | 109 | 240 | 252,8 | 134 | 134,7 | 110 | 111,3 | 109 | 109,3 | 110 | 110,2 | 131 | 133,7 |

# IV – Remarques et conclusion

Cette synthèse a permis de comparer, à temps d’exécution équivalents, l’ensemble des algorithmes développés en laboratoire sur un jeu de test donné. Il en ressort que l’algorithme le plus performant demeure la recherche taboue, qui obtient les meilleurs résultats pour chacun des exemplaires.

La version diversifiée de l’algorithme taboue offre des performances très proches mais néanmoins légèrement inférieures. La troisième position est occupée par l’algorithme évolutionnaire, qui est un processus génétique pur. L’absence de recherche locale dans l’algorithme ne l’a pas empêché de devancer la descente. Bien que moins performants, ces deux méthodes ne s’éloignent pas excessivement des performances taboues.

En revanche, la méthode gloutonne, avec une part d’aléatoire dégradant les résultats, propose des performances particulièrement médiocres, sans comparaison possible avec les autres algorithmes qui ont été développés par la suite.