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 0xk. |
| **Candidat S** | Tout vecteur ligne de longueur k, à valeurs dans [0 ; v-1]. |
| **Score d’un candidat** | Nombre de contraintes satisfaites parmi celles que 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 Nxk 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 Nxk 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 Nxk obtenue aléatoirement. |
| **Mouvement aléatoire** |  |
| **Paliers** |  |
| **Décroissance de la température** |  |
| **Critère d’arrêt** |  |
| **Paramètres de l’algorithme** |  |
|  |  |
|  |  |

## E – Recherche avec tabou (TS)

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## F – Tabou avec diversification (TSD)

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## G – Algorithme évolutionnaire (EA)

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# 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 de la machine de test

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 |

## C – Pseudocode

Blabla

## D – Caractéristiques

Blabla

## E – Liste des paramètres

Blabla

# IV – Remarques et conclusion

Blabla