INF6102 – Métaheuristiques
Travaux pratiques n°4 – 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 – Opérateurs de variation

A – Description de l'opérateur de mutation

L'opérateur de mutation implémenté travaille sur les symboles présents dans la solution. Pour faire muter un individu, il suffit de sélectionner au hasard quelques coefficients de la matrice puis de le remplacer par des symboles différents tirés aléatoirement.

La mutation doit garder un faible impact sur la qualité de la solution, c'est pourquoi le nombre de symbole modifié ne peut être trop important. De ce fait, on choisit lors d'une phase de mutation de modifier 0.01% des symboles de la configuration.

B – Description de l'opérateur de croisement de base

L'opérateur de croisement de base construit un enfant à partir de deux parents en transmettant les symboles de l'un ou l'autre individuellement. Concrètement, lors de la génération de l'enfant, pour chaque symbole, un tirage au sort décidera si le coefficient est hérité du premier ou du second parent.

Ce processus est explicité dans le pseudocode suivant :

- POUR i entre 1 et N
  - POUR j entre 1 et k
    - a = parent1[i][j]
    - b = parent2[i][j]
    - Affectation c = a ou c = b avec une probabilité de 50%
    - enfant[i][j] = c
  - FIN POUR
- FIN POUR

C – Description de l'opérateur de croisement spécialisé

L'opérateur de croisement spécialisé s'adapte aux caractéristiques du problème et à la structure des données puisqu'il travaille sur les lignes des configurations, en évaluant leur performance au sens de la fonction de coût. En d'autres termes, il va s'agir d'un opérateur de croisement heuristique.

Comme précédemment, cet opérateur utilise deux parents pour générer un enfant unique. La construction de l'enfant se fait alors ligne par ligne. La première ligne est choisie aléatoirement en provenance de l'un ou l'autre des deux parents. Par la suite, pour chaque ligne, on sélectionne celle entre les deux parents qui résout le plus de nouvelles contraintes élémentaires.

La construction de l'enfant suit donc une logique gloutonne. Mais, contrairement à l'algorithme glouton développé dans le premier laboratoire, la solution est ici construite ligne par ligne, avec un choix entre deux attributs différents uniquement. Cette implémentation répond au pseudocode suivant :

- Générer solution vide enfant
- Choisir aléatoirement (uniformément) parentX = parent1 ou parent2
- enfant[1][*] = parentX[1][*] (M[i][*] désigne toute la ligne i de la matrice M)
- POUR l entre 2 et N
  - candidat1 = parent1[l][*]
  - candidat2 = parent2[l][*]
  - enfant[l][*] = candidat1
  - cout1 = verifierSolution(enfant)
  - enfant[l][*] = candidat2
  - cout2 = verifierSolution(enfant)
  - SI cout1 < cout2

            - enfant[l][*] = candidat1
        - FIN SI
- FIN POUR

D – Evaluation de la diversité

Afin de mesurer la diversité et son évolution, à titre indicatif, au sein d'une population d'individus, il a été nécessaire de définir la notion de distance entre deux configurations. Puisque ce concept n'a rien d'évident concernant les solutions du problème de covering array, il a fallu tâcher d'implémenter quelque chose d'arbitraire mais intuitif.

De façon assez basique, la distance entre deux configurations correspond au nombre de symboles différents aux mêmes emplacements. La diversité d'une population est alors la somme des distances entre tous les individus pris deux à deux.

Cette fonction est décrite pour une population « pop » de taille M par le pseudocode suivant :

- d = 0
- POUR i entre 0 et M-2
    - POUR j entre i+1 et M-1
        - POUR c entre 0 et k*N-1
            - SI pop[i]->matrice[c] != pop[j]->matrice[c]
                - d++
            - FIN SI
        - FIN POUR
    - FIN POUR
- FIN POUR
- RETOURNER d

III – Description de l'algorithme évolutionniste

A – Pseudocode

- Créer le tableau population de taille tailleParents+tailleEnfants
- POUR i entre 0 et tailleParents-1
    - population[i] = configurationAleatoire(v, k, N)
- FIN POUR
- meilleureSolution = population[0]
- coutMeilleure = verifierSolution(meilleureSolution)
- TANT QUE ((coutMeilleure > 0) ET (tempsCalcul < 60s))
    - POUR e entre 0 et tailleEnfants-1
        - Choisir aléatoirement parent1 dans la population parmi les parents admissibles (tailleParents premiers indices)
        - Choisir aléatoirement parent2 dans la population parmi les parents admissibles (tailleParents premiers indices)

            - Générer l'enfant suite au croisement de parent1 et parent2
            - Appliquer l'opérateur de mutation à l'enfant
            - population[tailleParents+e] = enfant
        - FIN POUR
        - POUR i entre 0 et tailleParents+tailleEnfants-1
            - couts[i] = verifierSolution(population[i])
        - FIN POUR
        - Trier population et couts par ordre croissant de coût (les tailleParents premiers sont les meilleures configurations et pourront être parents à l'itération suivante)
        - Calculer la diversité de la population
        - meilleureSolution = population[0]
        - coutMeilleure = couts[0]
- FIN TANT QUE
- RETOURNER meilleureSolution

B – Caractéristiques

L'algorithme génétique proposé se rapproche du schéma d'évolution (μ+λ)-ES. Un ensemble de test préliminaires (détaillés ci-après) a permis de fixer le nombre de parents m=20 et le nombre d'enfants n=20. L'implémentation donnée est purement génétique, n'utilisant donc pas d'opérateur de recherche locale, on aurait pu penser préférable de travailler sur des populations assez élargies. Cependant, des tests préliminaires ont montré que les résultats obtenus étaient meilleurs lorsque le couple (m,n) vaut (20,20) que lorsqu'il est fixé à (50,50). Cela peut s'expliquer par le fait qu'une population plus grande occasionne des temps de croisement et de mutation plus longs pour délivrer une génération. Avec des populations de tailles plus petites, la sélection se fait plus vite, sur de meilleurs parents, et produit ainsi de meilleurs enfants.

Lors d'une itération génétique, les parents admissibles pour générer la population suivante sont les m meilleures configurations, c'est-à-dire les individus violant le moins de contraintes élémentaires. On génère alors n enfants par croisement de parents sélectionnés aléatoirement deux à deux avant de faire muter les enfants sur 0.01% de leurs symboles. Si ce pourcentage correspond à moins d'un symbole, on effectue tout de même la mutation sur un coefficient. La population subsistant à cette étape est alors la réunion des parents et des enfants, ce qui implique que la technique implémentée utilise le recouvrement.

La diversité de la population est alors calculée à titre indicatif, mais on ne cherche pas particulièrement à la favoriser, d'où la non utilisation d'une technique de fitness sharing par exemple lors de la sélection des individus survivant.

L'algorithme implémenté n'impose pas de forte pression sélective. Certes, seuls m individus parmi les m+n configurations de la population peuvent prétendre à être parents pour la création de la génération suivante, mais le tirage des parents pour un croisement est totalement uniforme lors de cette étape.

C – Liste des paramètres

- v : nombre de symboles admissibles dans la matrice de configuration
- k : nombre de colonnes de la matrice
- N : nombre de lignes de la matrice
- 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
- natureCroisement : indique si l'opérateur de croisement utilisé est l'opérateur générique ou bien le spécialisé

IV – 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.
- v = 3, k = 20 - au mieux, avec l'algorithme tabou, on a pu avoir N = 17.
- v = 3, k = 60 - au mieux, avec l'algorithme tabou, on a pu avoir N = 22.
- v = 5, k = 10 - au mieux, avec l'algorithme tabou, on a pu avoir N = 37.
- v = 5, k = 15 - au mieux, avec l'algorithme tabou, on a pu avoir N = 43.
- v = 8, k = 10 - au mieux, avec l'algorithme tabou, on a pu avoir N = 94.
- v = 8, k = 15 - au mieux, avec l'algorithme tabou, on a pu avoir N = 109.

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 – Présentation des tests préliminaires

Les tests préliminaires réalisés ont eu pour objet de fixer les paramètres évoqués ci-dessus. En essayant plusieurs valeurs pour le couple (m,n) représentant le nombre de parents et d'enfants d'une génération, on a pu se rendre compte que les meilleurs résultats étaient atteints pour (20,20).

Par ailleurs, on s'est attaché à trouver le taux τ de symboles mutés dans une configuration enfant. Ce paramètre est tel que le nombre s de symboles mutés vaut s = τ * k * N. La valeur de τ optimisant les résultats vaut ainsi 0.0001, soit 0.01% de symboles mutés. Si le taux de mutation est trop haut, la diversité ne s'épuise jamais, mais la qualité des solutions est médiocre. A l'inverse, un τ trop élevé signifie une diversité éphémère, et donc une stagnation de la qualité des solutions.

D – Résultats obtenus

Les résultats présentés font état, pour chaque exemplaire du problème, du coût de la solution (nombre d'erreurs subsistant pour le premier N tel que la solution n'est pas valide), le nombre d'itérations de l'algorithme, et le temps d'exécution. Une exécution de l'algorithme ne parvenant pas à trouver de solution satisfaisante durera toujours une minute, puisqu'il s'agit là d'un des critères d'arrêt du calcul. Cependant, il peut arriver qu'une solution valable soit exposée, réduisant ainsi le temps d'exécution de l'algorithme à moins d'une minute.

Pour chacune de ces trois grandeurs, les valeurs minimale, moyenne, et maximale sur 10 exécutions sont données. Par ailleurs, on considère d'une part les résultats obtenus avec l'opérateur de croisement générique, et d'autre part l'opérateur spécialisé.

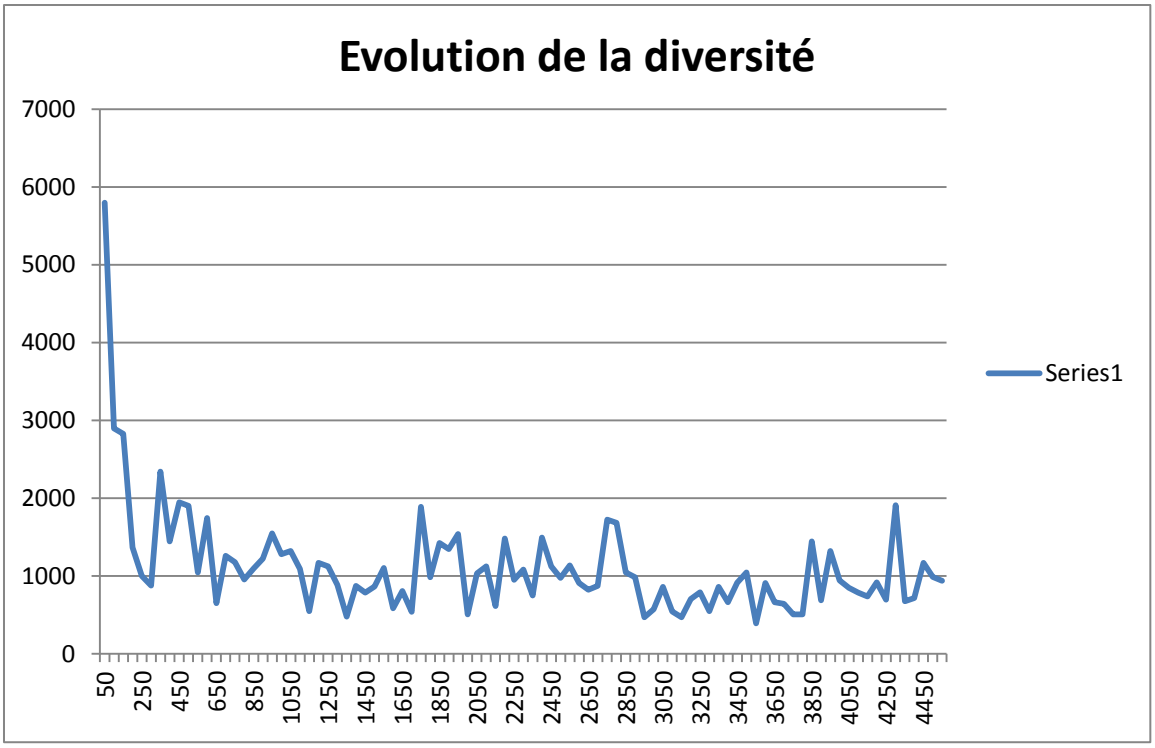
			Cout			Nombre itérations			Temps (s)		
v	k	N	Min	Moy	Max	Min	Moy	Max	Min	Moy	Max
2	4	4	2	2	2	653121	661511	679041	60	60	60
3	20	17	4	6.4	9	20730	20901.5	21104	60	60	60
3	60	24	6	10	14	1973	1978.9	1986	60	60	60
5	10	39	0	2.4	4	42780	44169.1	44506	58	59	60
5	15	45	2	6.5	11	19127	19247.3	19390	60	60	60
8	10	104	0	1.5	4	10921	19793.3	22345	39	54	60
8	15	130	0	0.3	2	4944	6387.4	8586	35	45	60

Opérateur de croisement générique (sur les symboles)

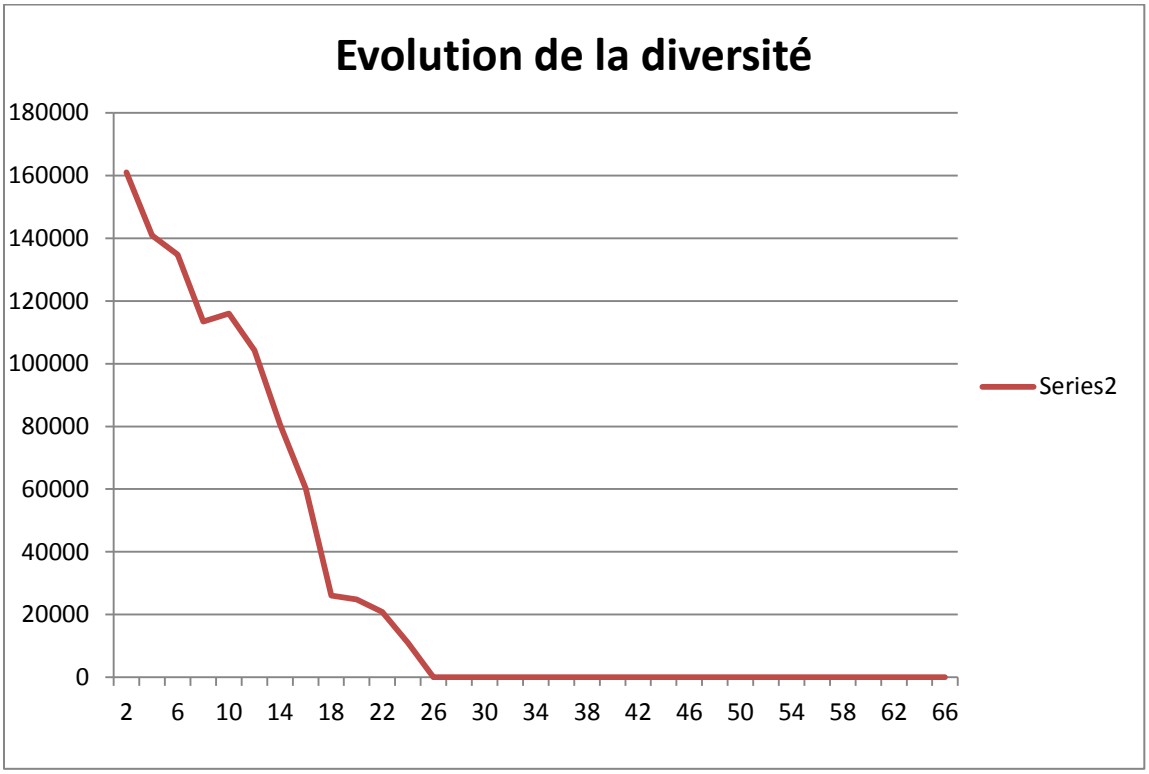
			Cout			Nombre itérations			Temps (s)		
v	k	N	Min	Moy	Max	Min	Moy	Max	Min	Moy	Max
2	4	4	2	2	2	725816	734934	742918	60	60	60
3	20	17	2	5.8	9	22208	22407.8	22545	60	60	60
3	60	24	6	9.4	15	2033	2043	2058	60	60	60
5	10	39	0	2.1	5	17476	48695	52512	20	56	60
5	15	45	4	6.5	13	21372	21639	21858	60	60	60
8	10	104	0	1	3	16531	25819	27852	37	56	60
8	15	130	0	0.1	1	5293	7168	10211	31	42	60

Opérateur de croisement spécialisé (sur les lignes)

Les graphiques suivants représentent l'évolution de l'indicateur de diversité en fonction du nombre d'itérations.



Evolution de la diversité au cours des générations – taux τ = 0.0001



Evolution de la diversité au cours des générations – taux τ = 0

E – Commentaires

Les résultats précédents montrent que la domination de l'opérateur de croisement spécialisé n'a rien d'évident. Le coût moyen des solutions trouvées est plus sensiblement inférieur sur les gros problèmes. Dans le reste des cas, certes l'opérateur sur les lignes paraît plus optimal, mais le coût des solutions et les temps d'exécutions restent semblables.

L'indicateur de diversité nous permet d'observer l'épuisement radical de la diversité de la population pour un taux de symboles mutés nul. On peut également se faire une idée de la vitesse d'épuisement de la diversité pour des taux de mutations plus raisonnables. Comme dans notre implémentation, on mute toujours au moins un symbole par enfant, la diversité ne s'épuise jamais.

Enfin, les performances de l'algorithme purement génétique sont bien moins bonnes que celles de la méthode taboue. Cela était prévisible du fait qu'aucun opérateur de recherche locale n'a été utilisé dans ce laboratoire. Il nous semblait cependant intéressant de constater cela directement en choisissant d'implémenter une solution purement génétique, et non pas une méthode hybride utilisant la recherche locale, comme par exemple la recherche taboue.