Groupe numéro 5 : Florian Korsakissok

Rapport TP1

Covering Arrays

1. Présentation et définition du problème

1.1 Définition du problème

Le problème traité est “Covering Arrays”. Il s'agit de générer une matrice particulière pour des valeurs de (k, v) données, selon les propriétés suivantes :

- chaque paire de colonnes doit contenir toutes les combinaisons des v symboles
- la matrice a k colonnes
- la matrice générée doit avoir le moins de lignes possible.

1.2 Informations pertinentes

Aucune autre source d'informations que les transparents du cours n'a été utilisée pour réaliser ce travail pratique.

1.3 Jeux de données

Les combinaisons utilisées pour les tests sont celles-ci :

(v, k) = 2 4

(v, k) = 3 20

(v, k) = 3 60

(v, k) = 5 10

(v, k) = 5 15

(v, k) = 8 10

(v, k) = 8 15.

Le programme doit être exécuté dans le même dossier qu'un fichier texte appelé inputData contenant la liste de ces valeurs.

2. Description de l'algorithme

2.1 Caractéristiques de l'algorithme glouton

Solution partielle S	Matrice partiellement remplie, ayant k colonnes
Solution initiale	Matrice vide, de taille (0 lignes, k colonnes)
Candidats	Chaque symbole possible
Score d'un candidat	(Nombre de contraintes satisfaites) + (1/nombre de colonnes)*(nombre de contraintes pouvant être satisfaites dans le reste de la ligne)
A chaque étape	Ajout du candidat dans la case courante
Critère d'arrêt	Lorsque toutes les contraintes sont satisfaites

2.2 Discussion du critère glouton

On choisit toujours le symbole qui maximise le nombre de contraintes satisfaites. Ainsi, on peut espérer que les contraintes soient rapidement satisfaites et que la condition d'arrêt soit vraie avant que le nombre de lignes de la matrice atteigne des valeurs trop énormes. De plus, comme on fait aussi intervenir le nombre de contraintes pouvant potentiellement être satisfaites avec les colonnes de droite dans la ligne (encore à remplir), on est certain que l'algorithme finit par trouver une solution.

2.3 Contre-exemple

Voici un contre-exemple pour les paramètres (v,k) = (2, 4).

Matrice générée par l'algorithme glouton :

0 0 0 0

1 1 1 0

0 1 0 1

1 0 1 1

0 0 1 0

1 0 0 0

Meilleure matrice que celle générée par le glouton :

0 0 0 0

1 1 1 0

0 1 0 1

1 0 0 1

0 0 1 1

2.4 Randomisation de l'algorithme

Lors du choix du candidat, je rajoute de l'aléatoire en décidant du symbole selon une fonction dépendant du paramètre α (qui règle le niveau d'aléatoire) et du score de chaque symbole s_i . Je note p_i la quantité $\frac{s_i}{\sum_{k=1}^{v} s_k}$. On remarque que $\sum_{i=1}^{v} p_i = 1$. Je détermine aussi le symbole qui a le score maximum et le mets en position 1 : s_1 est le plus grand des s_i .

La fonction déterminant la probabilité qu'un symbole soit choisi à une certaine étape de l'algorithme est la suivante :

$$f_i(\alpha) = p_k - p_k . \alpha + \alpha \; pour \; i = 1$$

$$f_i(\alpha) = p_k - p_k . \alpha \; pour \; i \geq 2 \quad .$$

On remarque encore une fois que $\forall \alpha, \sum_{i=1}^{v} f_i(\alpha) = 1$.

Ainsi, pour α = 0, $f_i(0) = p_k$

Pour α = 1, $f_i(1) = 1 \; pour \; i = 1$

et $f_i(1) = 0 \; pour \; i \geq 2$.

La probabilité de choix d'un symbole varie donc entre le glouton pur (non stochastique) et une probabilité proportionnelle à son score. Néanmoins, nous voulons connaître l'effet d'un algorithme purement aléatoire sur les résultats. J'ai donc introduit une autre famille de fonctions g, telle que $g_i(0) = p_k = f_i(0)$ et $g_i(-1) = \frac{1}{v}$. Nous avons ainsi une fonction continue sur tout son intervalle de définition (nous nous intéressons en particulier à l'intervalle [-1, 1]). De plus, pour α=-1, les probabilités sont uniformes (glouton complètement aléatoire) et pour α=1, l'algorithme choisit toujours le symbole avec le meilleur score (glouton pur non stochastique).

2.5 Pseudo-code

- Initialiser une matrice vide
- Répéter ...
  - Rajouter une ligne à la matrice
  - Pour chaque colonne k, faire
    - Pour chaque symbole v, faire
      - calculer le nombre de contraintes satisfaites si on met le symbole v (son score)
    - Choisir le symbole qui a le meilleur score
    - L'ajouter dans la matrice à la colonne k
- ... Tant qu'il existe des contraintes non satisfaites
- Renvoyer la matrice

2.6 Complexité

Pour chaque itération de la boucle “tant que”, on remplit une ligne de la matrice (O(N) itérations). Chaque itération fait donc intervenir une boucle sur les colonnes de la matrice (O(k) itérations). Dans chaque itération de cette boucle, on choisit le symbole à insérer. Pour ce faire, on calcule le score de chaque symbole (O(v) itérations). Le score étant le nombre de contraintes satisfaites, on parcourt toutes les autres colonnes de la ligne pour voir quelles seraient ces contraintes (O(k) itérations). Pour les colonnes à droite de la case courante, soit les colonnes encore vides, on regarde pour tous les symboles quelles sont les contraintes qui pourraient être satisfaites avec le symbole à tester (O(v) itérations). La complexité globale d'une itération de la boucle “tant que” est donc en $O(k^2v^2)$. En ce qui concerne le nombre de lignes de la matrice générée, il est assez difficile d'avoir une bonne majoration en fonction de k et v. L'algorithme résultant est donc en $O(Nk^2v^2)$.

3. Implantation des algorithmes

3.1 Implantation des classes de base

J'ai implanté deux classes dans ce travail pratique, CA_Solution et CMatrix. CA_Solution permet de lire les fichiers de données, générer les matrices ainsi que vérifier la validité d'une solution. CMatrix est une représentation simple d'une matrice.

3.2 Implantation de l'algorithme glouton randomisé

L'ajout d'aléatoire se fait dans la méthode chooseSymbol de la classe CA_Solution, qui permet de définir un comportement différent selon l'algorithme (glouton pur ou glouton randomisé; j'ai aussi implanté d'autres variantes pour observer les différences dans les résultats).

3.3 Implantation d'une fonction de vérification

Le coût de la solution est obtenu de manière triviale, puisqu'il s'agit du nombre de lignes de la matrice. En revanche, la validité est un peu plus complexe à obtenir. La méthode int getErrorsNumber(); de la classe CA_Solution renvoie le nombre de contraintes non satisfaites par la dernière matrice générée, il faut donc qu'elle renvoie 0 pour que la solution soit valide.

4. Tests

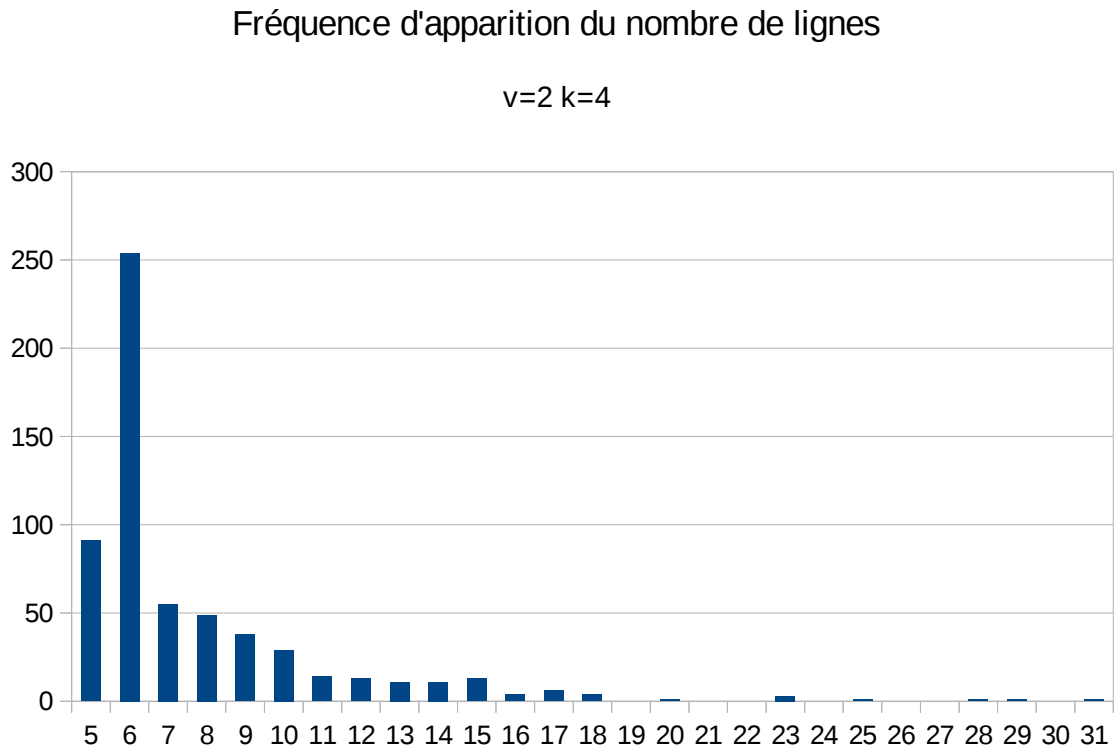
4.1 Tableau de résultats

k	v	alpha	min lignes	max lignes	ecart type	temps moyen(ms)
2	**4**	-1	5	31	4.853	0.075
		-0.66	5	17	2.349	0.057
		-0.33	5	10	1.317	0.048
		0	5	8	0.674	0.043
		0.33	5	7	0.533	0.041
		0.66	5	7	0.551	0.015
		1	5	5	0.000	0.013
3	**20**	-1	48	99	11.552	2.348
		-0.66	31	50	3.754	1.320
		-0.33	25	33	1.719	1.084
		0	23	26	0.695	0.963
		0.33	22	27	0.840	0.944
		0.66	21	25	0.786	0.921
		1	23	23	0.000	0.869
3	**60**	-1	65	111	9.463	21.043
		-0.66	40	60	3.498	13.070
		-0.33	34	41	1.348	10.392
		0	30	33	0.763	9.316
		0.33	29	32	0.730	9.124
		0.66	29	32	0.746	8.942
		1	30	30	0.000	8.067
5	**10**	-1	133	411	36.207	2.811
		-0.66	77	108	6.614	1.598
		-0.33	63	77	2.776	1.304
		0	54	62	1.429	1.174
		0.33	50	58	1.658	1.100
		0.66	49	54	1.108	1.036
		1	48	48	0.000	0.943
5	**15**	-1	156	309	27.526	6.742
		-0.66	89	126	5.987	3.800
		-0.33	72	82	2.347	3.119
		0	61	70	1.318	2.802
		0.33	58	66	1.445	2.656
		0.66	57	62	1.152	2.532
		1	58	58	0.000	2.443
8	**10**	-1	387	867	83.778	16.449
		-0.66	211	249	8.749	8.526
		-0.33	164	190	4.823	7.134
		0	146	159	2.810	6.596
		0.33	130	146	2.935	5.884
		0.66	122	134	2.186	5.355
		1	121	121	0.000	4.943
8	**15**	-1	467	837	83.993	41.387
		-0.66	232	290	10.379	21.518
		-0.33	191	207	3.768	18.049
		0	166	181	2.740	16.361
		0.33	153	165	2.199	15.631
		0.66	142	152	1.952	14.453
		1	143	143	0.000	13.807

Coefficient obtenu avec dfmax : 5,40.

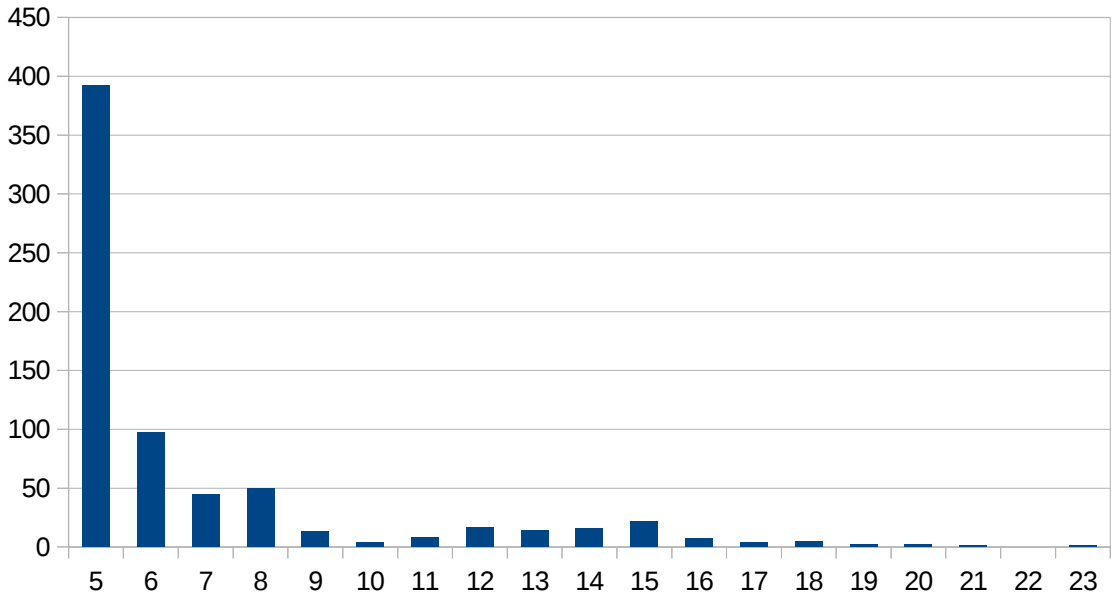
Coefficient multiplicateur pour les résultats des tests : 8,60 / 5,40 = 1.5926

Système d'exploitation	Ubuntu 13.10 64 bits
Mémoire vive	7.7 GiB
Processeur	Intel® Core™ i7-3610QM CPU @ 2.30GHz × 8



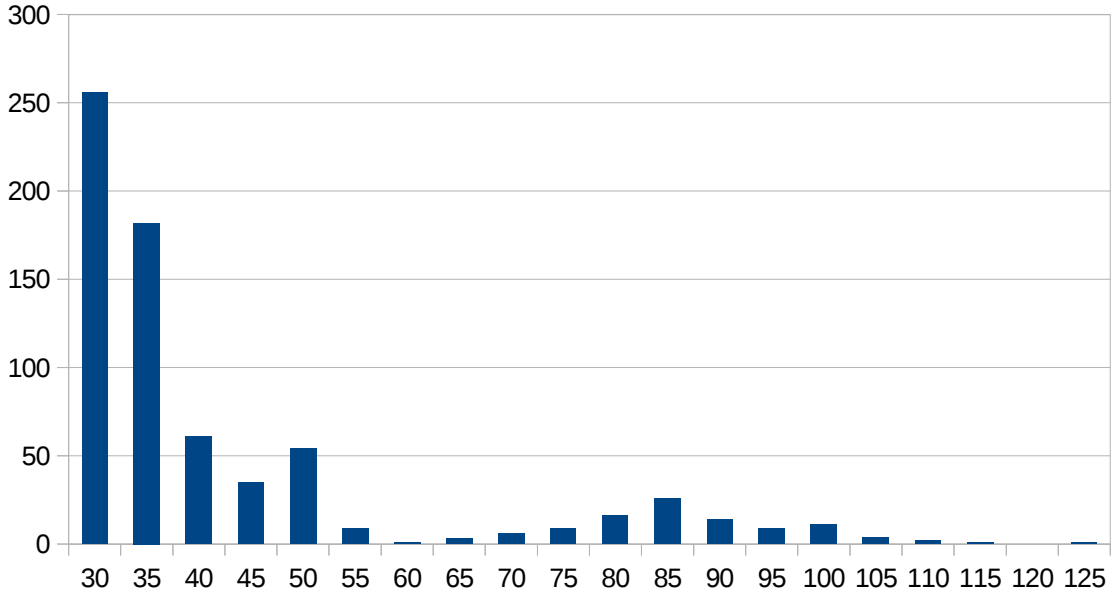
Fréquence d'apparition du nombre de lignes

v=3 k=20



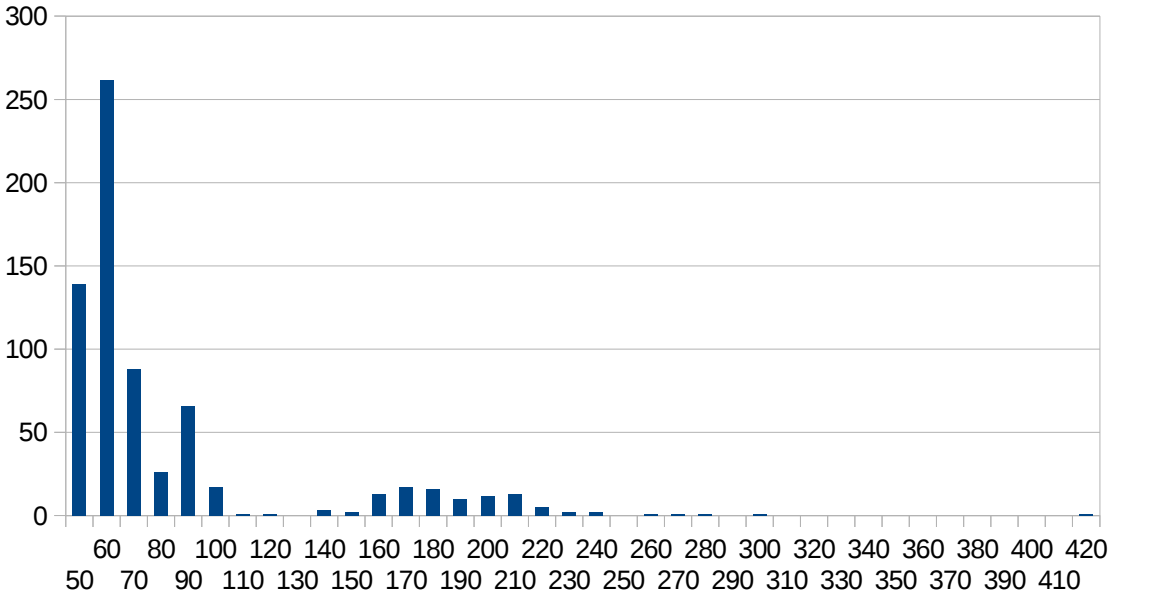
Fréquence d'apparition du nombre de lignes

v=3 k=60



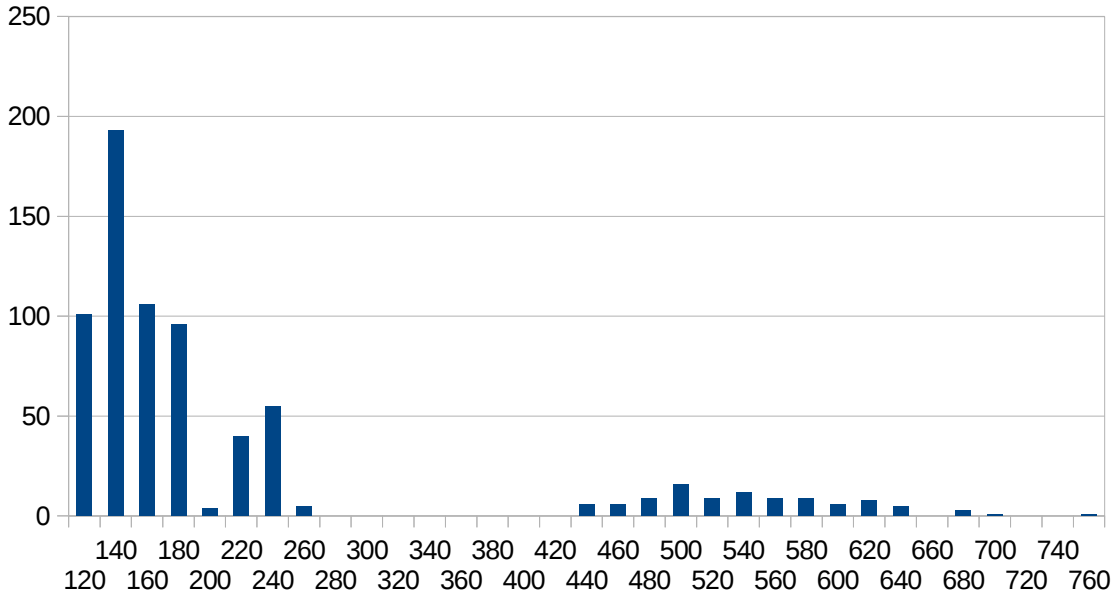
Fréquence d'apparition du nombre de lignes

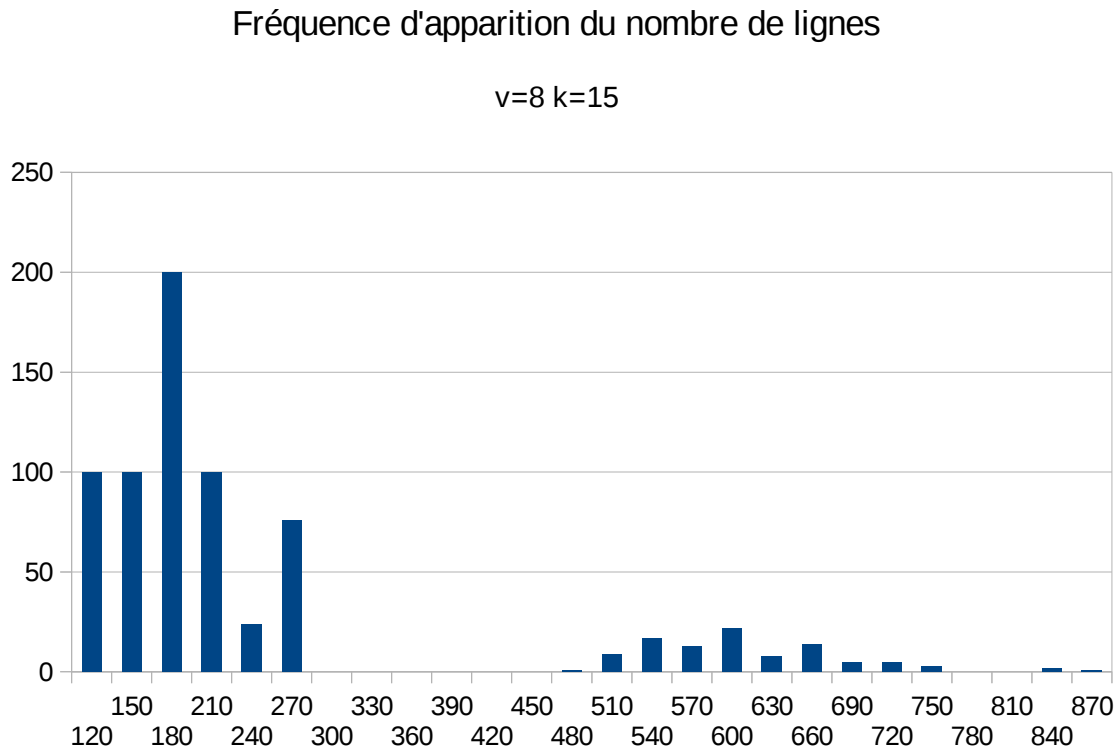
v=5 k=10



Fréquence d'apparition du nombre de lignes

v=8 k=10





4.2 Commentaires

Comme on peut s'y attendre, l'ajout d'aléatoire augmente l'écart-type et permet parfois (rarement...) de trouver de meilleures solutions que le glouton pur. On remarque que les seules fois où les solutions sont améliorées grâce à la part de hasard, c'est dans le cas où l'influence aléatoire est très faible ($\alpha = 0.66$). Il est en effet cohérent que pour un nombre d'itérations aussi faible, seule la méthode avec peu d'aléatoire donne de bons résultats. En effet, plus la part d'aléatoire est grande, plus le nombre d'itérations nécessaire à l'obtention d'un bon résultat est grand; néanmoins, la marge de manoeuvre augmente aussi avec la part d'aléatoire : plus on se rapproche du glouton pur, moins on permet d'avoir de bonnes solutions.