FACULTÉ DES SCIENCES DE MONTPELLIER

Projet CSP – Génération de benchmark et évaluation de méthode

Participants :

Laurencia ***DOVI LATE***

Marta ***BOSHKOVSKA***

Introduction

Ce rapport explore la méthodologie d'évaluation des algorithmes de résolution de problèmes en trois parties distinctes.Dans la première étape, nous construisons un jeu d'essai pertinent avec des paramètres clés, tout en analysant la transition de phase. La deuxième phase se concentre sur l'évaluation des performances d'un algorithme spécifique sur ce jeu, en fournissant des mesures détaillées, des courbes et une analyse comparative des modifications apportées à l'algorithme Choco. Enfin, la troisième partie consiste à évaluer expérimentalement la méthode par défaut de Choco, en incluant des mesures telles que le temps de calcul et la taille de l'arbre, suivies éventuellement d'un ajustement facultatif de l'heuristique du solveur par défaut, avec des comparaisons et des analyses approfondies.Dans notre rendu zip contenant les sources nous avons les fichiers Expe.java est pour le code des graphes 1, 2,3 et 4 de la partie C, Expe1.java pour la partie B et Expe2.java pour le code des noeuds.

A. Matériel fourni

Question 8:

Pour le fichier "benchInsat.txt" , aucun des réseaux générés n'a eu au moins une solution. Pour le fichier "benchSatisf.txt", les trois réseaux générés ont tous eu au moins une solution.

Cela suggère que les réseaux dans "benchSatisf.txt" sont plus faciles à satisfaire que ceux dans "benchInsat.txt".

Cette différence peut être due aux paramètres de génération des réseaux, notamment le nombre de contraintes et le nombre de tuples.

Ainsi, on pourrait conclure qu'un nombre plus élevé de tuples par contrainte peut accroître la probabilité de trouver une assignation des variables satisfaisant toutes les contraintes.

B. Construction d'un jeu d'essais conséquent et identification de la transition de phase

Dans cette section, nous avons créé quatre jeux d'essai, chacun visant à explorer l'impact de différents paramètres de contrôle tels que le nombre de tuples et le nombre de contraintes. L'objectif

principal de chaque jeu d'essai est d'analyser comment ces paramètres influent sur le pourcentage de réseaux ayant au moins une solution.

Afin d'obtenir des résultats un peu plus fiables, nous avons fixé le nombre de réseaux à 10 pour chaque jeu d'essai. Bien que l'utilisation d'un nombre encore plus élevé aurait pu conduire à des résultats plus précis, cela aurait également entraîné des temps d'expérimentation considérablement plus longs.

Pour éviter des temps d'exécution excessivement longs sur certaines instances, nous avons introduit un mécanisme de timeout.La limite de temps est établie à 10 secondes, ce qui implique que si le processus de calcul excède cette durée, nous interrompons la recherche d'une solution pour ce réseau.

Pour gérer les situations dans lesquelles aucune solution n'a été trouvée en raison de la limite de temps(timeout), nous avons trouvé deux approches sont envisageables:

-Considérer que le réseau est dépourvu de solutions

-Ne pas prendre en compte ces cas dans nos calculs

Dans notre tp,nous avons choisi d'adopter la première approche ,considérant ainsi que le réseau n'a pas de solution dans le cas d'un timeout .

1er jeu d'essai:

Script bash pour génération de jeux d'essai: "jeu_essai_1.sh".

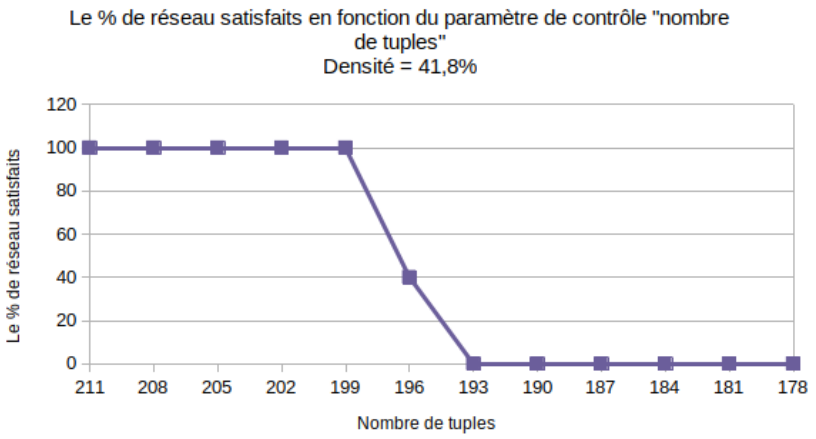
Dans ce jeu d'essai on a une densité fixe de: $D = \frac{2c}{n^2 - n} = \frac{2 \times 249}{35^2 - 35} = 41,8\%$

Les paramètres fixes sont:

- 35 variables
- 17 la taille de domaine de chaque variable
- 249 contraintes
- 10 réseaux

Nous avons fixé la densité et fait varier le nombre de tuples afin d'analyser l'impact sur le pourcentage de réseaux ayant au moins une solution. Ainsi en variant le nombre de tuples, nous avons modifié la dureté des contraintes dans le réseau en fonction du nombre de tuples.

Les résultats que nous avons obtenus sont représentés sur le graphe ci-dessous:



Avec cette approche, nous pouvons observer comment la variation du nombre de tuples influence la capacité des réseaux à avoir au moins une solution. On observe une stabilité du pourcentage de

réseaux satisfaits, soit 100%, pour les tuples variant de 211 à 199. Cependant, à mesure que le nombre de tuples diminue de 199 à 193, le pourcentage de réseaux satisfaisants décline progressivement pour atteindre zéro à partir de 193 tuples.

2ème jeu d'essai:

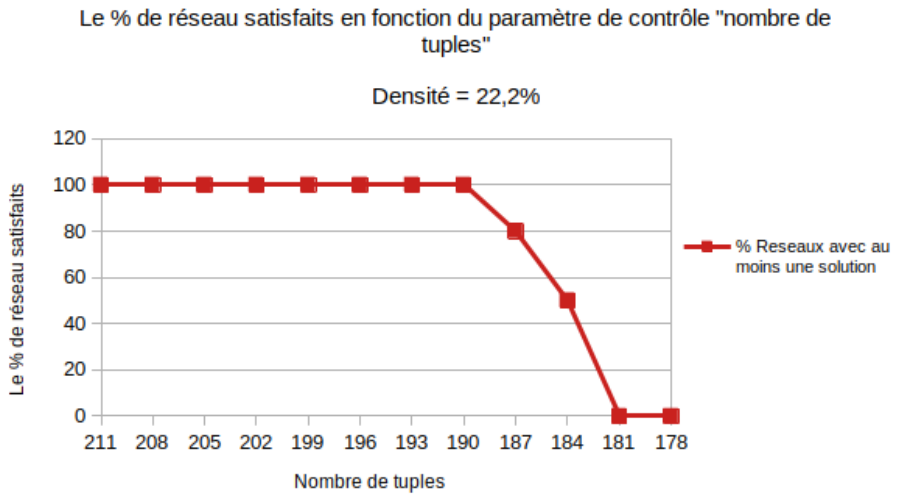
Script bash pour génération de jeux d'essai: "jeu_essai_2.sh".

Dans ce jeu d'essai on a une densité fixe de: $D = \frac{2c}{n^2 - n} = \frac{2 \times 220}{45^2 - 45} = 22,2\%$

Les paramètres fixes sont:

- 45 variables
- 17 la taille de domaine de chaque variable
- 220 contraintes
- 10 reseaux

Dans ce jeu,la densité est toujours fixée comme dans le dernier mais cette fois ,cette dernière a pour valeur 22.2%.Les résultats obtenus sont représentés sur le graphe suivant:



<u>3ème jeu d'essai:</u>

Script bash pour génération de jeux d'essai: "jeu_essai_3.sh".

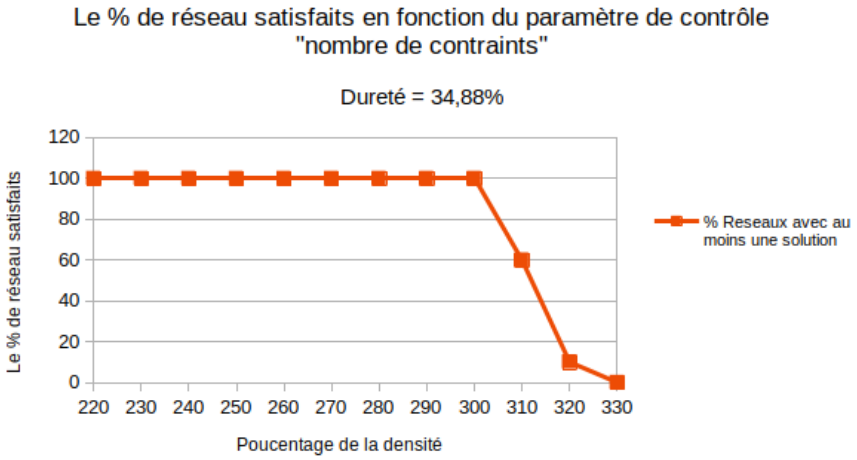
Dans ce jeu d'essai on a une dureté fixe de: $T = \frac{d^2 - t}{d^2} = \frac{289 - 211}{289} = 34,88\%$

Les paramètres fixes sont:

- 50 variables
- 18 la taille de domaine de chaque variable
- 211 tuples
- 10 reseaux

Nous avons fait varier le nombre de contraintes pour observer comment cela affecte le pourcentage de réseaux ayant au moins une solution. En ajustant le nombre de contraintes nous avons fait évoluer la densité.

Les résultats que nous avons obtenus sont représentés sur le graphique suivant:



4ème jeu d'essai:

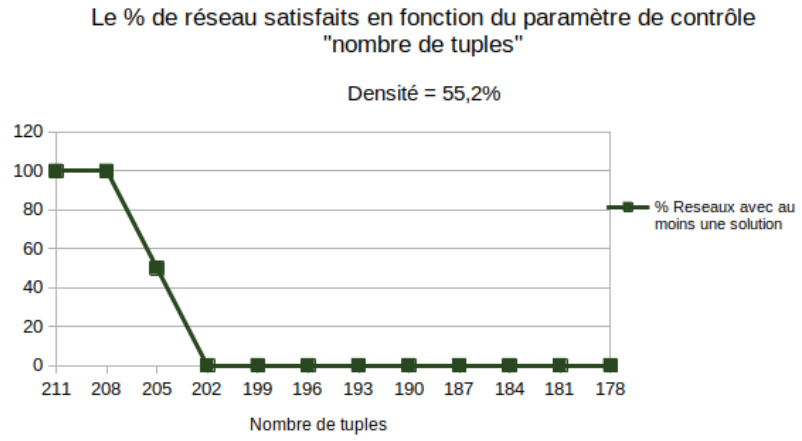
Script bash pour génération de jeux d'essai: "jeu_essai_2.sh".

Dans ce jeu d'essai on a une densité fixe de: $D = \frac{2c}{n^2 - n} = \frac{2 \times 240}{30^2 - 30} = 55,2\%$

Les paramètres fixes sont:

- 30 variables
- 17 la taille de domaine de chaque variable
- 240 contraintes
- 10 reseaux

Les résultats que nous avons obtenus sont représentés sur le graphe suivant:

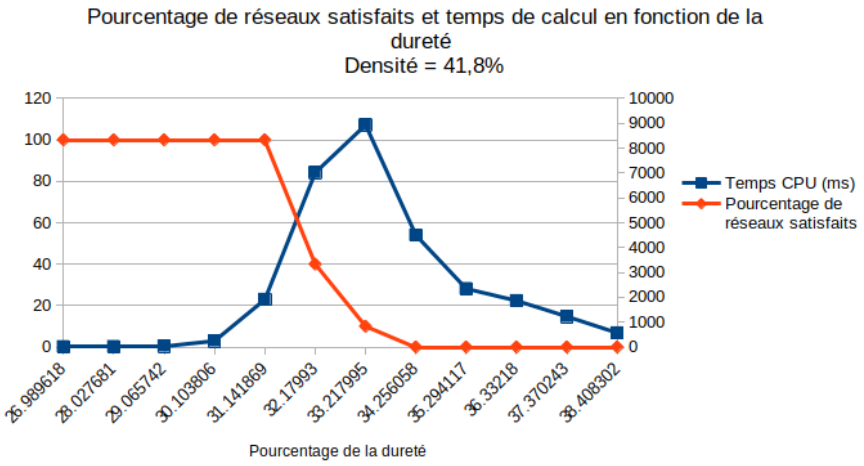


C. Évaluation de la méthode de résolution par défaut de Choco sur vos jeux d'essais

On évalue la méthode de résolution par défaut de Choco sur le premier jeu d'essai en mesurant le temps d'exécution du programme. Dans le tableau ci-dessus, les paramètres incluent le pourcentage de dureté variant de 26,9% à 38,4%, le temps CPU du processus de résolution en millisecondes, et le pourcentage de réseaux satisfaits. La densité est fixée à 41,8%.

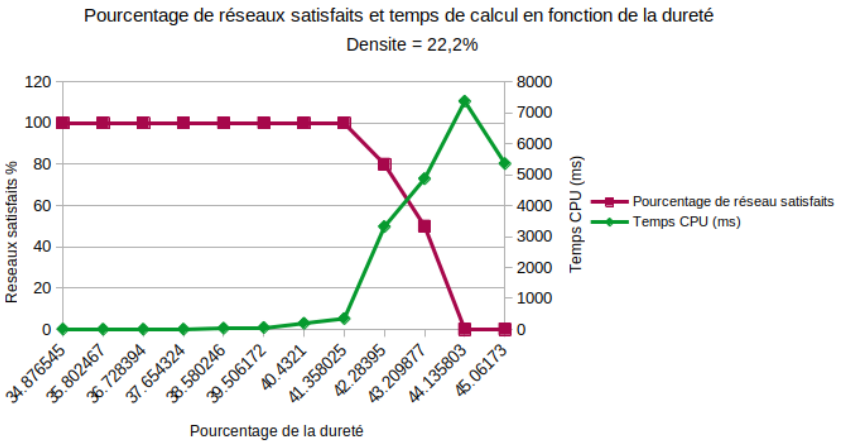
Dureté	Temps CPU (ms)	Pourcentage de réseaux satisfaits
26.989618	9	100
28.027681	15	100
29.065742	37	100
30.103806	244	100
31.141869	1912	100
32.17993	7013	40
33.217995	8933	10
34.256058	4494	0
35.294117	2345	0
36.33218	1861	0
37.370243	1225	0
38.408302	566	0

En se basant sur les données de ce tableau, nous avons généré le graphique ci-dessous. Il est notable qu'un pic de temps de calcul survient au niveau de la transition de phase.

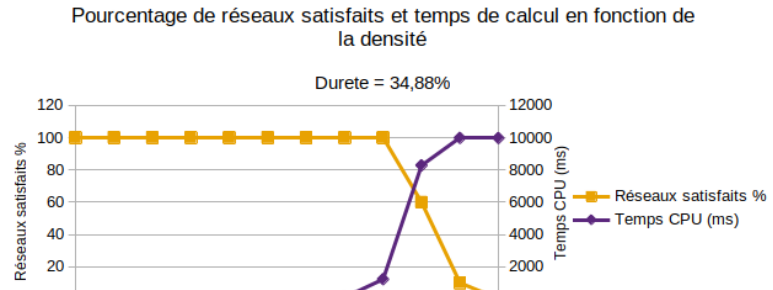


Pour le deuxième jeu d'essai, nous avons effectué les mêmes mesures, mais cette fois-ci, la densité a été fixée à 22,2%.

Les résultats que nous avons obtenus sont représentés sur le graph suivant:

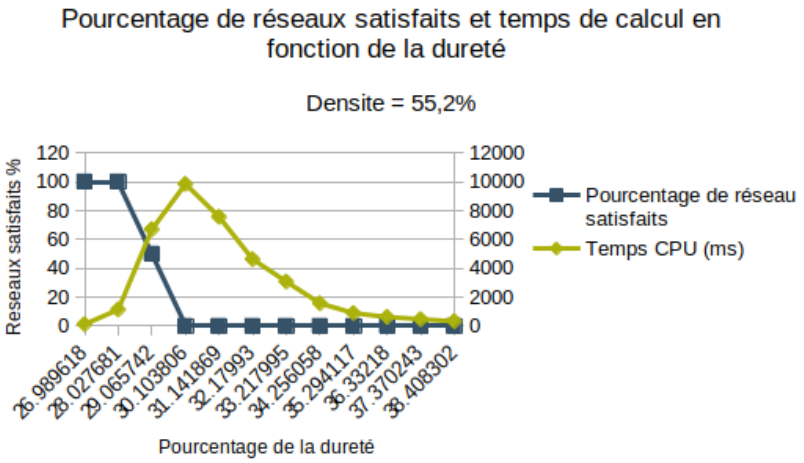


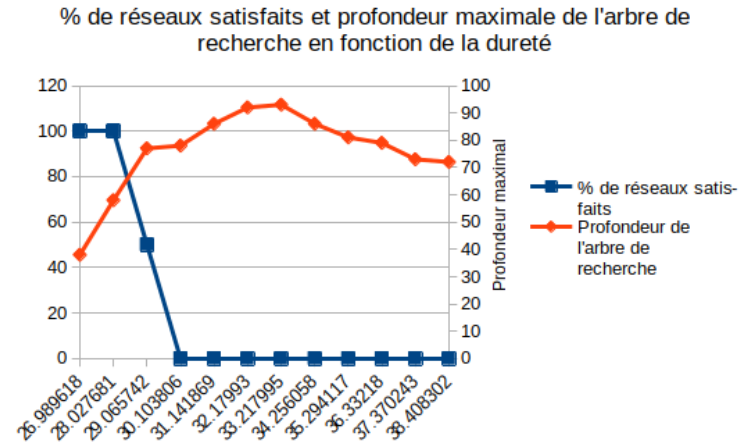
Pour le troisième jeu d'essai, la dureté a été fixée à 34,88%. Vous pouvez observer l'évolution du pourcentage de réseaux satisfaits et le temps CPU en millisecondes en fonction de la densité, qui



augmente de 34,88% à 45,07%. Les résultats que nous avons obtenus sont représentés sur le graph suivant:

Pour ce dernier jeu d'essai, on a fixé la densité à 55,2% et observé le pourcentage de résolution pour une dureté variant entre 26.98% et 37.37%. On constate alors sur le graphique que le phase de transition est atteinte aux alentours de 30% de dureté pour cette densité, là où le temps CPU monte à 10 000 (ms) et le nombre de réseaux satisfaits chute à 0.





On s'aperçoit sur le graphique précédent que lorsque que la dureté du problème augmente, la profondeur de l'arbre augmente aussi. Elle augmente fortement et ne décroît que peu car si un réseau n'est pas résoluble il faut parcourir toutes les possibilités. De même, lorsque le timeout est atteint car on pourrait aller de plus en plus profondément très longtemps.