Universite de Montpellier

Rapport Csp Optimisation

Chakib ELHOUITI
Massili KEZZOUL

6 novembre 2020

1 Introduction

Ce projet s'attache à mettre en place des optimisations au resolveur de CSP.

Il est demandé de réaliser au moins une des optimisations suivantes : - heuristique d'assignation des variables, - test de viol de contraintes même si toutes les variables de la contrainte ne sont pas assignées, - arc-consistance, - forward checking...

Le projet est suivi d'un travail d'évaluation complet des effets des différentes optimisations.

2 Construction du benchmark

On a commencé par réaliser un script Python, qui ne permet de construire un benchmark avec des valeurs de (n,c,d,t) différentes, ce qui veut dire, notre script Python utilise le csp generateur pour generer 10 instance de Networ pour un niveau de dureté (1–100) en les stockant dans des fichiers sous un format bien précis(ex : d1-I7.txt), pour pouvoir aprés automatiser la tache d'execution pour notre APP java.

2.1

3 Implementation des optimisations

Dans notre cas on a pu réaliser deux optimisations :

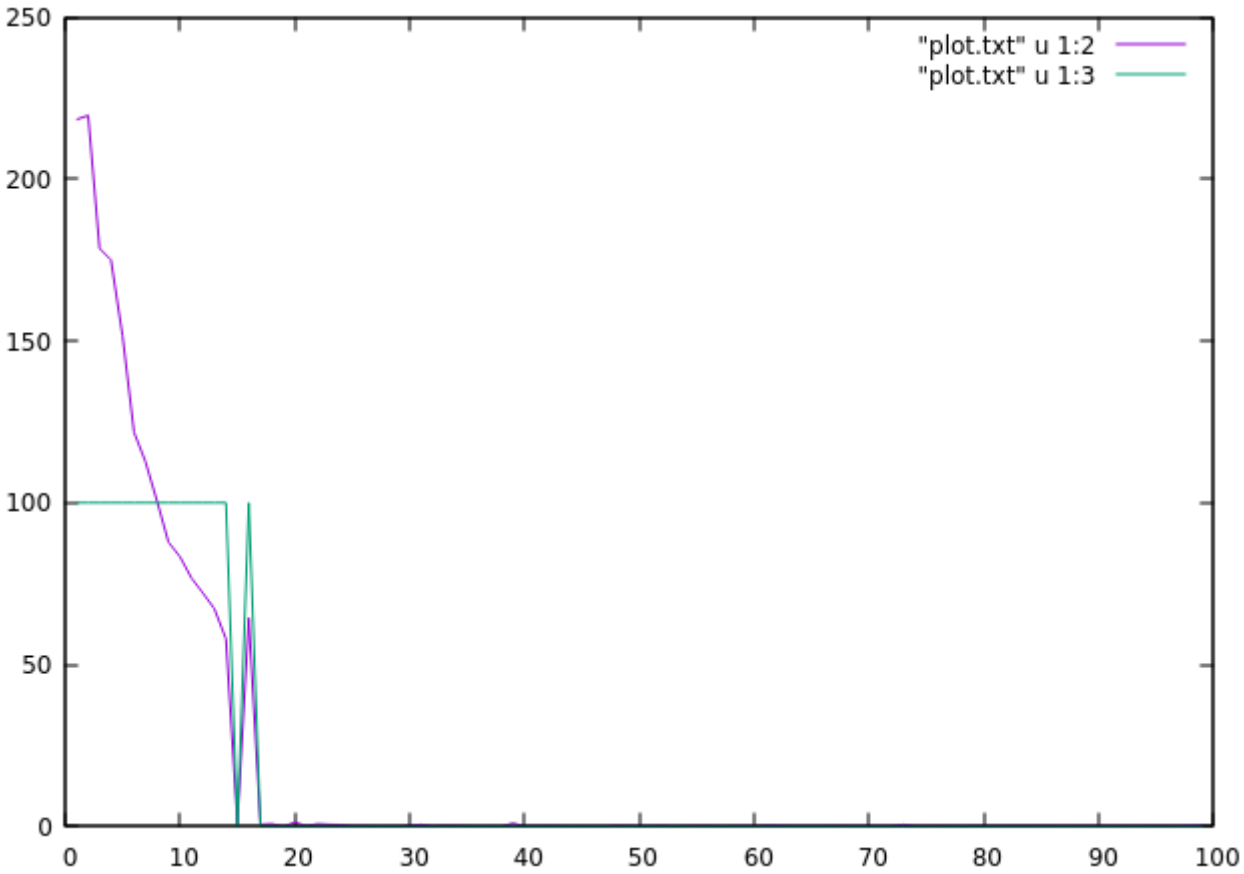
3.1 Forward Cheking

On a réalisé l'implémentation de l'algorithme Forward Cheking, pour pouvoir remplacer backtrack.

3.1.1 Resultats du Forward Cheking

On a pu constaté en variant les differents Valeur pour generer un réseau de contrainte aléatoire et sur un niveau de dureté de 1 à 100. L'algo du forward cheking permet en premier lieu de resoudre les réseaux de grande taille en fonction de (n,c,d,t).

Concernant le graphe obtenu pour une execution de réseaux de contrainte pour des niveaux de dureté de 1 à 100, on a calculé le temps moyen de 5 execution par instance, on a obtenu : un graph avec un temps élevé, c'est du à l'algo du forward cheking (plus de tuples avec plus de contraintes = plus de temps). Concernant la phase de transition, on remarque bien un pic pour le temps moyen par dureté Ce qui montre le graphe suivant : (n = 35, c=249, d = 17 et le t est calculé)



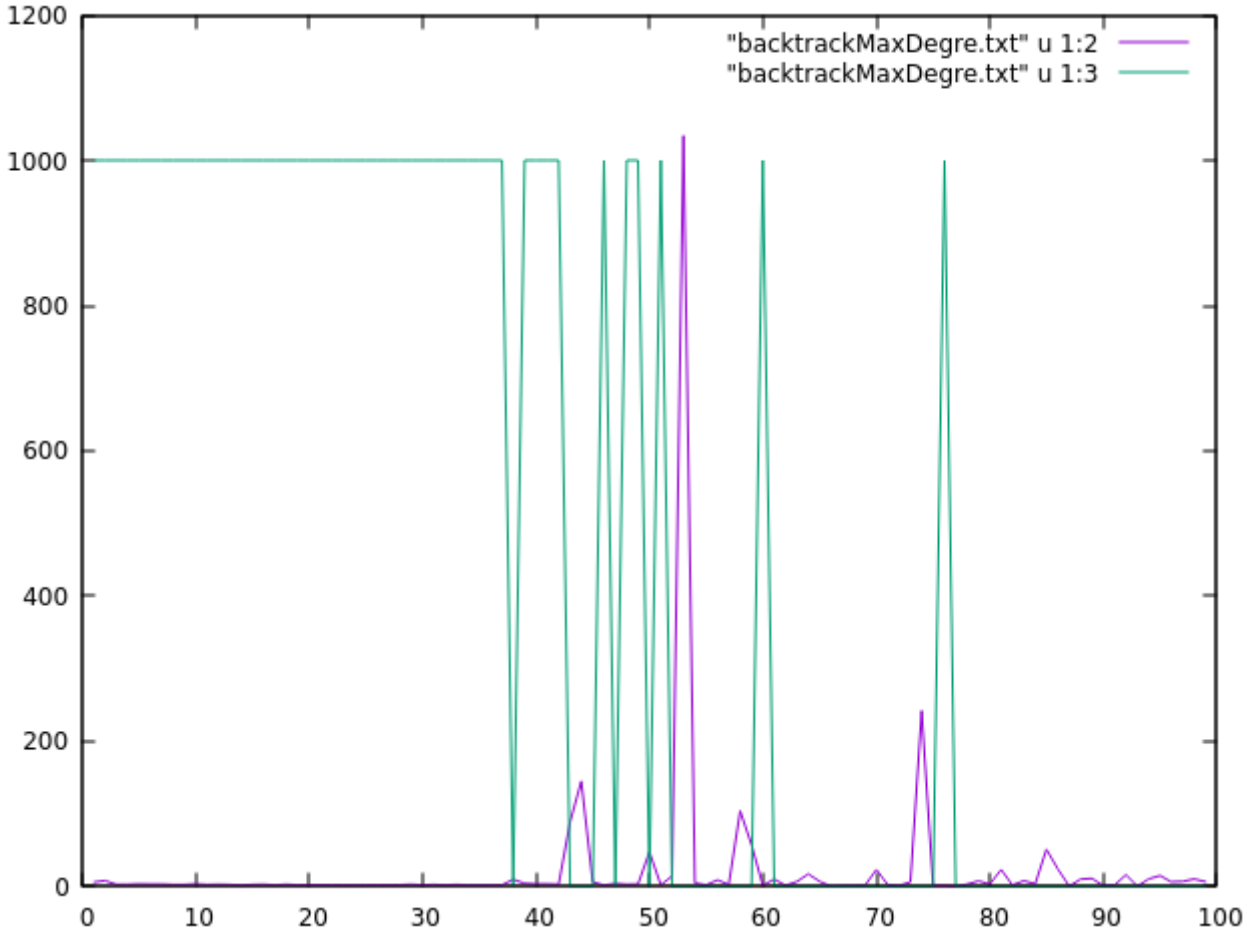
3.2 L'heuristique MaxDegré

Pour cette heuristique, on a commencé par stocker une Hashmap (string,int) qui permet de stocker les variables du réseau avec leurs nombre d'occurences dans les contraintes (le degré), ensuite on tri cette HashMap par ordre décroissant et on choisi a chaque fois la variable qui n'est pas assignée avec le plus grand degré.

3.2.1 Resultats du maxDegré

Cette heuristique ne permet pas de résoudre des CSP avec de grande taille comme il le fait FC, mais permet bien de constater à travers un graphe le pic lors de la phase de transition. Elle reste toujours meilleure que le backtrack avec chooseVar sans aucune heuristique mais performante que le forward cheking.

exemple de graphe : (n = 20, c=50 , d = 17)



3.3 Exécution des programmes

pour éxecuter le programme il suffit de :

— Ouvrire un terminal et se rendre au dossier src/

— Ajouter les droits d'éxecutions au programme exec.sh

— Puis l'éxecuter 'exec.sh'

Le script génère un benchmark puis éxecuter l'algorithme de résolution sur ce dernier et gnuplot sur le resultat. Vous aurez donc à la fin un fichier graphe.png qui contient la courbe du temps d'éxecution ainsi que le tracé de transition de phase.

4 materiels et systèmes :

On a utilisé deux script dans notre implementation, le premier en Python pour générer les fichiers de tests et le deuxiéme en Shell pour automatiser l'exécution.

— Processeur : Comme on est deux, on testé avec un I5 7ème génération et un I5 2ème génération, on a remarqué une petite différence sur le temps moyen mais les résultats étaient les mêmes.

— Ram : 8GB et 4GB

— OS : Pop'Os (20.04) c'est une distribution Linux similaire à ubuntu mais plus stable (à notre avis).

— Compilateur : Javac 11.0.9