31/12/2016

Jean-Baptiste DURIEZ Jordane QUINCY

Rapport de TP

Programmation par contraintes

Table des matières

[Introduction 1](#_Toc471051400)

[Génération des CSP 1](#_Toc471051401)

[Développement des algorithmes 2](#_Toc471051402)

[Résultats 2](#_Toc471051403)

[1. CSP des n reines 2](#_Toc471051404)

[a. Algorithmes recherchant une seule solution 3](#_Toc471051405)

[b. Algorithmes recherchant toutes les solutions 6](#_Toc471051406)

[2. CSP aléatoire 8](#_Toc471051407)

[a. Algorithmes cherchant une solution 8](#_Toc471051408)

[b. Algorithmes recherchant toutes les solutions 9](#_Toc471051409)

# Introduction

Le but de ce TP est de développer 3 algorithmes de résolution de CSP (Problème de satisfaction de contraintes), l’algorithme BackTracking, l’algorithme BackJumping orienté graphe et enfin l’algorithme ForwardChecking. Ainsi nous allons pouvoir comparer ces trois algorithmes sur différents CSP pour voir quel algorithme est le plus efficace en termes de temps mais aussi en termes de nombre de nœuds visités.

Toute la programmation pour ce TP a été réalisée en Java.

# Génération des CSP

Avant de développer les algorithmes nous avons dû dans un premier temps générer de manière aléatoire des CSP afin d’avoir des problèmes à faire résoudre par nos algorithmes de résolution.

Pour ce faire, nous avons modélisé un CSP par une classe (CSP) ayant comme attribut une liste de variables, une liste de contraintes et un nombre de variables.

Les variables sont représentées par la classe Variable, qui a pour attribut un id (qui va être le nom de la variable), un domaine (liste d’entier) et une connectivité (qui sera utilisé lors de la génération du CSP).

Les contraintes sont quant à elles représentées par la classe Contraintes. Elle a pour attribut variable1 et variable2 qui correspondent aux variables de la contrainte et une liste de couple d’entier qui correspondent aux couples acceptées par cette contrainte.

La génération aléatoire de CSP va se faire ensuite par différents paramètres permettant de créer des CSP plus ou moins complexes. Ainsi la génération va dépendre :

* Du nombre de variables qu’il y aura dans le CSP, plus il y a de variables, plus le CSP sera complexe
* De la taille maximale des domaines des variables. En effet chaque variable à un domaine de définition qui va être générée aléatoirement et sa taille sera inférieure ou égale à la taille maximale des domaines des variables. Plus la taille des domaines est grande et plus le CSP sera difficile.
* De la densité, qui est le ratio du nombre de contraintes pour le CSP qu’on va générer sur le nombre de contrainte totale si toutes les variables du CSP étaient contraintes entre elle. Grâce à cette donnée on va pouvoir savoir combien notre CSP va avoir de contraintes. Plus il y a de contrainte, plus le CSP sera complexe.
* De la dureté, qui est le ratio du nombre de couples autorisés sur le nombre de couples total pour une contrainte. Grâce à la dureté on va pouvoir déterminer la liste des couples autorisés par contrainte. Plus la dureté est élevée et plus le problème sera facile, inversement plus la dureté est faible et plus le problème sera difficile.
* De la connectivité, cela correspond au nombre maximal de contrainte pour une variable donnée.

Une fois qu’on a tous ces éléments, la génération de CSP est assez simple, on génère d’abord les variables, puis on génère toutes les contraintes possibles (sans générer leur liste de couples). On vient ensuite retirer les contraintes de manière aléatoire pour satisfaire la connectivité et le nombre de contraintes de notre CSP (déterminé grâce à la densité). Il ne reste plus qu’à générer la liste des coupes pour chaque contrainte qu’on a conservée en fonction de la dureté.

Afin de savoir si nos algorithmes fonctionnent correctement nous avons également dû générer des CSP connus. Pour cela, nous avons choisi de créer un générateur de CSP du problème des n reines que nous connaissons bien. Ainsi nous pouvons créer n’importe quel problème des n reines et tenter de le résoudre.

# Développement des algorithmes

Pour chaque algorithme nous avons développé 2 versions. Une première version recherchant une seule solution du CSP et s’arrêtant dès que cette solution a été trouvée. Et une deuxième version recherchant toutes les solutions du CSP.

La première version des algorithmes est fortement basée sur les algorithmes donnés en cours alors que la deuxième version est un peu modifiée afin de parcourir toutes les solutions.

Tous les algorithmes fonctionnent parfaitement sauf l’algorithme de BackJumping qui permet de trouver toutes les solutions. En effet, même si nous arrivons à avoir plusieurs solutions avec ce dernier algorithme, nous ne les récupérons pas forcément toutes, cela dépend des CSP. Cela est dû à la gestion après avoir trouvé une solution pour continuer le parcours comme si de rien n’était qui ne doit pas bien faire le travail...

# Résultats

Nous avons réalisé différentes résolutions de CSP afin de voir les différences de nos algorithmes.

## CSP des n reines

Tout d’abord pour être sûr que nos algorithmes fonctionnent, nous les avons testés avec le problème des n Reines et ils ont tous donnés de bons résultats. Les voici :

## Algorithmes recherchant une seule solution

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Problème n reines | Algorithme utilisé | Nombre de nœuds parcourus | Temps d’exécution en ms | Première solution |
| 1 Reine | BackTracking | 1 | 2 | [{1 = 1}] |
| 1 Reine | BackJumping | 1 | 1 | [{1 = 1}] |
| 1 Reine | ForwardChecking | 1 | 0 | [{1 = 1}] |
| 2 Reines | BackTracking | 1 | 2 | Pas de solution |
| 2 Reines | BackJumping | 1 | 0 | Pas de solution |
| 2 Reines | ForwardChecking | 1 | 0 | Pas de solution |
| 3 Reines | BackTracking | 9 | 3 | Pas de solution |
| 3 Reines | BackJumping | 9 | 1 | Pas de solution |
| 3 Reines | ForwardChecking | 1 | 0 | Pas de solution |
| 4 Reines | BackTracking | 12 | 5 | [{1=2, 2=4, 3=1, 4=3}] |
| 4 Reines | BackJumping | 12 | 1 | [{1=2, 2=4, 3=1, 4=3}] |
| 4 Reines | ForwardChecking | 8 | 1 | [{1=2, 2=4, 3=1, 4=3}] |
| 5 Reines | BackTracking | 5 | 7 | [{1=1, 2=3, 3=5, 4=2, 5=4}] |
| 5 Reines | BackJumping | 5 | 2 | [{1=1, 2=3, 3=5, 4=2, 5=4}] |
| 5 Reines | ForwardChecking | 5 | 1 | [{1=1, 2=3, 3=5, 4=2, 5=4}] |
| 6 Reines | BackTracking | 56 | 11 | [{1=2, 2=4, 3=6, 4=1, 5=3, 6=5}] |
| 6 Reines | BackJumping | 56 | 4 | [{1=2, 2=4, 3=6, 4=1, 5=3, 6=5}] |
| 6 Reines | ForwardChecking | 32 | 4 | [{1=2, 2=4, 3=6, 4=1, 5=3, 6=5}] |
| 7 Reines | BackTracking | 11 | 7 | [{1=1, 2=3, 3=5, 4=7, 5=2, 6=4, 7=6}] |
| 7 Reines | BackJumping | 11 | 3 | [{1=1, 2=3, 3=5, 4=7, 5=2, 6=4, 7=6}] |
| 7 Reines | ForwardChecking | 7 | 5 | [{1=1, 2=3, 3=5, 4=7, 5=2, 6=4, 7=6}] |
| 8 Reines | BackTracking | 218 | 41 | [{1=1, 2=5, 3=8, 4=6, 5=3, 6=7, 7=2, 8=4}] |
| 8 Reines | BackJumping | 218 | 32 | [{1=1, 2=5, 3=8, 4=6, 5=3, 6=7, 7=2, 8=4}] |
| 8 Reines | ForwardChecking | 98 | 31 | [{1=1, 2=5, 3=8, 4=6, 5=3, 6=7, 7=2, 8=4}] |
| 9 Reines | BackTracking | 73 | 24 | [{1=1, 2=3, 3=6, 4=8, 5=2, 6=4, 7=9, 8=7, 9=5}] |
| 9 Reines | BackJumping | 73 | 12 | [{1=1, 2=3, 3=6, 4=8, 5=2, 6=4, 7=9, 8=7, 9=5}] |
| 9 Reines | ForwardChecking | 33 | 21 | [{1=1, 2=3, 3=6, 4=8, 5=2, 6=4, 7=9, 8=7, 9=5}] |
| 10 Reines | BackTracking | 194 | 62 | [{1=1, 2=3, 3=6, 4=8, 5=10, 6=5, 7=9, 8=2, 9=4, 10=7}] |
| 10 Reines | BackJumping | 194 | 39 | [{1=1, 2=3, 3=6, 4=8, 5=10, 6=5, 7=9, 8=2, 9=4, 10=7}] |
| 10 Reines | ForwardChecking | 96 | 72 | [{1=1, 2=3, 3=6, 4=8, 5=10, 6=5, 7=9, 8=2, 9=4, 10=7}] |
| 11 Reines | BackTracking | 93 | 53 | [{1=1, 2=3, 3=5, 4=7, 5=9, 6=11, 7=2, 8=4, 9=6, 10=8, 11=10}] |
| 11 Reines | BackJumping | 93 | 27 | [{1=1, 2=3, 3=5, 4=7, 5=9, 6=11, 7=2, 8=4, 9=6, 10=8, 11=10}] |
| 11 Reines | ForwardChecking | 37 | 57 | [{1=1, 2=3, 3=5, 4=7, 5=9, 6=11, 7=2, 8=4, 9=6, 10=8, 11=10}] |
| 12 Reines | BackTracking | 510 | 244 | [{1=1, 2=3, 3=5, 4=8, 5=10, 6=12, 7=6, 8=11, 9=2, 10=7, 11=9, 12=4}] |
| 12 Reines | BackJumping | 510 | 182 | [{1=1, 2=3, 3=5, 4=8, 5=10, 6=12, 7=6, 8=11, 9=2, 10=7, 11=9, 12=4}] |
| 12 Reines | ForwardChecking | 210 | 199 | [{1=1, 2=3, 3=5, 4=8, 5=10, 6=12, 7=6, 8=11, 9=2, 10=7, 11=9, 12=4}] |
| 13 Reines | BackTracking | 209 | 129 | [{1=1, 2=3, 3=5, 4=2, 5=9, 6=12, 7=10, 8=13, 9=4, 10=6, 11=8, 12=11, 13=7}] |
| 13 Reines | BackJumping | 209 | 118 | [{1=1, 2=3, 3=5, 4=2, 5=9, 6=12, 7=10, 8=13, 9=4, 10=6, 11=8, 12=11, 13=7}] |
| 13 Reines | ForwardChecking | 95 | 236 | [{1=1, 2=3, 3=5, 4=2, 5=9, 6=12, 7=10, 8=13, 9=4, 10=6, 11=8, 12=11, 13=7}] |
| 14 Reines | BackTracking | 3784 | 1424 | [{1=1, 2=3, 3=5, 4=7, 5=12, 6=10, 7=13, 8=4, 9=14, 10=9, 11=2, 12=6, 13=8, 14=11}] |
| 14 Reines | BackJumping | 2784 | 1084 | [{1=1, 2=3, 3=5, 4=7, 5=12, 6=10, 7=13, 8=4, 9=14, 10=9, 11=2, 12=6, 13=8, 14=11}] |
| 14 Reines | ForwardChecking | 1736 | 1639 | [{1=1, 2=3, 3=5, 4=7, 5=12, 6=10, 7=13, 8=4, 9=14, 10=9, 11=2, 12=6, 13=8, 14=11}] |
| 15 Reines | BackTracking | 2703 | 1113 | [{1=1, 2=3, 3=5, 4=2, 5=10, 6=12, 7=14, 8=4, 9=13, 10=9, 11=6, 12=15, 13=7, 14=11, 15=8}] |
| 15 Reines | BackJumping | 2703 | 957 | [{1=1, 2=3, 3=5, 4=2, 5=10, 6=12, 7=14, 8=4, 9=13, 10=9, 11=6, 12=15, 13=7, 14=11, 15=8}] |
| 15 Reines | ForwardChecking | 1195 | 1704 | [{1=1, 2=3, 3=5, 4=2, 5=10, 6=12, 7=14, 8=4, 9=13, 10=9, 11=6, 12=15, 13=7, 14=11, 15=8}] |
| 16 Reines | BackTracking | 20088 | 9855 | [{1=1, 2=3, 3=5, 4=2, 5=13, 6=9, 7=14, 8=12, 9=15, 10=6, 11=16, 12=7, 13=4, 14=11, 15=8, 16=10}] |
| 16 Reines | BackJumping | 20088 | 9596 | [{1=1, 2=3, 3=5, 4=2, 5=13, 6=9, 7=14, 8=12, 9=15, 10=6, 11=16, 12=7, 13=4, 14=11, 15=8, 16=10}] |
| 16 Reines | ForwardChecking | 8740 | 13544 | [{1=1, 2=3, 3=5, 4=2, 5=13, 6=9, 7=14, 8=12, 9=15, 10=6, 11=16, 12=7, 13=4, 14=11, 15=8, 16=10}] |
| 17 Reines | BackTracking | 10731 | 6844 | [{1=1, 2=3, 3=5, 4=2, 5=8, 6=11, 7=15, 8=7, 9=16, 10=14, 11=17, 12=4, 13=6, 14=9, 15=12, 16=10, 17=13}] |
| 17 Reines | BackJumping | 10731 | 6544 | [{1=1, 2=3, 3=5, 4=2, 5=8, 6=11, 7=15, 8=7, 9=16, 10=14, 11=17, 12=4, 13=6, 14=9, 15=12, 16=10, 17=13}] |
| 17 Reines | ForwardChecking | 4959 | 10577 | [{1=1, 2=3, 3=5, 4=2, 5=8, 6=11, 7=15, 8=7, 9=16, 10=14, 11=17, 12=4, 13=6, 14=9, 15=12, 16=10, 17=13}] |
| 18 Reines | BackTracking | 82580 | 58907 | [{1=1, 2=3, 3=5, 4=2, 5=8, 6=15, 7=12, 8=16, 9=13, 10=17, 11=6, 12=18, 13=7, 14=4, 15=11, 16=9, 17=14, 18=10}] |
| 18 Reines | BackJumping | 82580 | 59139 | [{1=1, 2=3, 3=5, 4=2, 5=8, 6=15, 7=12, 8=16, 9=13, 10=17, 11=6, 12=18, 13=7, 14=4, 15=11, 16=9, 17=14, 18=10}] |
| 18 Reines | ForwardChecking | 35232 | 88400 | [{1=1, 2=3, 3=5, 4=2, 5=8, 6=15, 7=12, 8=16, 9=13, 10=17, 11=6, 12=18, 13=7, 14=4, 15=11, 16=9, 17=14, 18=10}] |

Tout d’abord les résultats des solutions sont correctes (vérifié sur internet).

Ensuite on voit très clairement ici qu’il n’y a pas de différence entre l’algorithme de BackTracking et l’algorithme de BackJumping, ils parcours exactement le même nombre de nœuds et ont un temps d’exécution très proche. Ce résultat est tout à fait normal puisque dans le problème des n reines toutes les variables sont reliées entre-elles donc faire un « backjump » revient exactement à la même chose que de faire un « backtrack ».

Par contre l’algorithme ForwardChecking a lui des résultats différents en termes de temps d’exécution et de parcours de nœuds. En effet ce dernier parcours nettement moins de nœuds, il est donc bien plus efficace à ce niveau-là, en revanche il prend généralement plus de temps… Ce qui n’est pas très normal car en parcourant moins de nœuds le temps d’exécution devrait être moindre. Cela est donc sûrement dû à notre algorithme qui fait bien du ForwardChecking (l’algo parcoure bien que les nœuds nécessaires), mais qui ne doit pas être très bien optimisé et qui prend donc plus de temps.

## Algorithmes recherchant toutes les solutions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Problème n reines | Algorithme utilisé | Nombre de nœuds parcourus | Temps d’exécution en ms | Nombre de solutions |
| 1 Reine | BackTracking | 2 | 2 | 1 |
| 1 Reine | BackJumping | 2 | 0 | 1 |
| 1 Reine | ForwardChecking | 2 | 0 | 1 |
| 2 Reines | BackTracking | 1 | 2 | 0 |
| 2 Reines | BackJumping | 1 | 1 | 0 |
| 2 Reines | ForwardChecking | 1 | 0 | 0 |
| 3 Reines | BackTracking | 9 | 2 | 0 |
| 3 Reines | BackJumping | 9 | 1 | 0 |
| 3 Reines | ForwardChecking | 1 | 0 | 0 |
| 4 Reines | BackTracking | 31 | 6 | 2 |
| 4 Reines | BackJumping | 31 | 2 | 2 |
| 4 Reines | ForwardChecking | 23 | 2 | 2 |
| 5 Reines | BackTracking | 97 | 10 | 10 |
| 5 Reines | BackJumping | 97 | 4 | 10 |
| 5 Reines | ForwardChecking | 89 | 5 | 10 |
| 6 Reines | BackTracking | 301 | 24 | 4 |
| 6 Reines | BackJumping | 301 | 12 | 4 |
| 6 Reines | ForwardChecking | 169 | 16 | 4 |
| 7 Reines | BackTracking | 1063 | 81 | 40 |
| 7 Reines | BackJumping | 1063 | 37 | 40 |
| 7 Reines | ForwardChecking | 627 | 80 | 40 |
| 8 Reines | BackTracking | 4021 | 206 | 92 |
| 8 Reines | BackJumping | 4021 | 146 | 92 |
| 8 Reines | ForwardChecking | 2237 | 212 | 92 |
| 9 Reines | BackTracking | 16435 | 958 | 352 |
| 9 Reines | BackJumping | 16435 | 802 | 352 |
| 9 Reines | ForwardChecking | 9087 | 1061 | 352 |
| 10 Reines | BackTracking | 70353 | 5754 | 724 |
| 10 Reines | BackJumping | 70353 | 5617 | 724 |
| 10 Reines | ForwardChecking | 34253 | 7044 | 724 |
| 11 Reines | BackTracking | 331171 | 38606 | 2680 |
| 11 Reines | BackJumping | 331171 | 38697 | 2680 |
| 11 Reines | ForwardChecking | 154827 | 43706 | 2680 |
| 12 Reines | BackTracking | 1698177 | 302491 | 14200 |
| 12 Reines | BackJumping | 1698177 | 301366 | 14200 |
| 12 Reines | ForwardChecking | 780413 | 282402 | 14200 |

Comme pour les algorithmes cherchant une solution, les résultats sont bons. Et de même l’algorithme de BackJumping a exactement les mêmes résultats que l’algorithme de BackTracking. Et l’algorithme de ForwardChecking parcoure une nouvelle fois beaucoup moins de nœuds mais prend plus de temps (sûrement dû à notre implémentation non optimisée).

## CSP aléatoire

## Algorithmes cherchant une solution

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithme utilisé | Nombre Variables | Taille Max Domaine | Densité | Dureté | Connectivité | Nombre Nœuds | Temps exécution (en ms) | Réalisable |
| BackTracking | 20 | 10 | 50 | 70 | 10 | 418 | 47 | Oui |
| BackJumping | 20 | 10 | 50 | 70 | 10 | 404 | 32 | Oui |
| ForwardChecking | 20 | 10 | 50 | 70 | 10 | 20 | 20 | Oui |
| BackTracking | 20 | 10 | 50 | 70 | 10 | 3442 | 97 | Oui |
| BackJumping | 20 | 10 | 50 | 70 | 10 | 456 | 15 | Oui |
| ForwardChecking | 20 | 10 | 50 | 70 | 10 | 20 | 28 | Oui |
| BackTracking | 20 | 10 | 50 | 70 | 10 | 46816 | 1105 | Oui |
| BackJumping | 20 | 10 | 50 | 70 | 10 | 19360 | 438 | Oui |
| ForwardChecking | 20 | 10 | 50 | 70 | 10 | 8294 | 2042 | Oui |
| BackTracking | 20 | 10 | 50 | 50 | 10 | 240 | 29 | Oui |
| BackJumping | 20 | 10 | 50 | 50 | 10 | 212 | 15 | Oui |
| ForwardChecking | 20 | 10 | 50 | 50 | 10 | 46 | 27 | Oui |
| BackTracking | 20 | 10 | 50 | 50 | 10 | 14767 | 635 | Non |
| BackJumping | 20 | 10 | 50 | 50 | 10 | 10998 | 369 | Non |
| ForwardChecking | 20 | 10 | 50 | 50 | 10 | 957 | 573 | Non |
| BackTracking | 20 | 10 | 50 | 50 | 10 | 230834 | 8981 | Oui |
| BackJumping | 20 | 10 | 50 | 50 | 10 | 185664 | 6965 | Oui |
| ForwardChecking | 20 | 10 | 50 | 50 | 10 | 8740 | 2919 | Oui |
| BackTracking | 20 | 10 | 20 | 40 | 10 | 14734 | 246 | Oui |
| BackJumping | 20 | 10 | 20 | 40 | 10 | 382 | 6 | Oui |
| ForwardChecking | 20 | 10 | 20 | 40 | 10 | 3240 | 228 | Oui |
| BackTracking | 20 | 10 | 20 | 40 | 10 | 1170 | 45 | Oui |
| BackJumping | 20 | 10 | 20 | 40 | 10 | 67 | 2 | Oui |
| ForwardChecking | 20 | 10 | 20 | 40 | 10 | 616 | 102 | Oui |
| BackTracking | 20 | 10 | 20 | 40 | 10 | 20 | 7 | Oui |
| BackJumping | 20 | 10 | 20 | 40 | 10 | 20 | 3 | Oui |
| ForwardChecking | 20 | 10 | 20 | 40 | 10 | 20 | 18 | Oui |

Cette fois-ci on voit bien la différence entre l’algorithme BackTracking et celui de BackJumping, en effet en génération aléatoire toutes les variables ne sont pas connectées entre elles, de ce fait le « backjump » permet réellement de ne pas passer pas des nœuds inutiles.

Quant à l’algorithme de ForwardChecking il est à nouveau un peu plus long à l’exécution mais passe quasi tout le temps par moins de nœuds.

## Algorithmes recherchant toutes les solutions

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithme utilisé | Nombre Variables | Taille Max Domaine | Densité | Dureté | Connectivité | Nombre Nœuds | Temps exécution (en ms) | Nombre de solutions trouvées |
| BackTracking | 10 | 10 | 50 | 70 | 10 | 218141 | 2782 | 71450 |
| BackJumping | 10 | 10 | 50 | 70 | 10 | 218141 | 1953 | 71450 |
| ForwardChecking | 10 | 10 | 50 | 70 | 10 | 218017 | 1815 | 71450 |
| BackTracking | 10 | 10 | 50 | 70 | 10 | 998362 | 17672 | 615411 |
| BackJumping | 10 | 10 | 50 | 70 | 10 | 483800 | 4282 | 297537 |
| ForwardChecking | 10 | 10 | 50 | 70 | 10 | 998360 | 14681 | 615411 |
| BackTracking | 10 | 10 | 50 | 70 | 10 | 609457 | 9594 | 326906 |
| BackJumping | 10 | 10 | 50 | 70 | 10 | 560917 | 6911 | 298734 |
| ForwardChecking | 10 | 10 | 50 | 70 | 10 | 606879 | 7600 | 326906 |
| BackTracking | 10 | 10 | 50 | 50 | 10 | 224296 | 6364 | 128473 |
| BackJumping | 10 | 10 | 50 | 50 | 10 | 98128 | 1694 | 55492 |
| ForwardChecking | 10 | 10 | 50 | 50 | 10 | 221602 | 3992 | 128473 |
| BackTracking | 10 | 10 | 50 | 50 | 10 | 24875 | 263 | 11952 |
| BackJumping | 10 | 10 | 50 | 50 | 10 | 1347 | 10 | 600 |
| ForwardChecking | 10 | 10 | 50 | 50 | 10 | 24875 | 253 | 11952 |
| BackTracking | 10 | 10 | 50 | 50 | 10 | 3691 | 70 | 432 |
| BackJumping | 10 | 10 | 50 | 50 | 10 | 437 | 6 | 51 |
| ForwardChecking | 10 | 10 | 50 | 50 | 10 | 2207 | 36 | 432 |
| BackTracking | 10 | 10 | 20 | 40 | 10 | 971359 | 11427 | 555750 |
| BackJumping | 10 | 10 | 20 | 40 | 10 | 10685 | 72 | 6065 |
| ForwardChecking | 10 | 10 | 20 | 40 | 10 | 959239 | 10367 | 555750 |
| BackTracking | 10 | 10 | 20 | 40 | 10 | 1768457 | 13511 | 519696 |
| BackJumping | 10 | 10 | 20 | 40 | 10 | 1264142 | 4960 | 392364 |
| ForwardChecking | 10 | 10 | 20 | 40 | 10 | 1636337 | 7349 | 519696 |
| BackTracking | 10 | 10 | 20 | 40 | 10 | 543457 | 6977 | 408240 |
| BackJumping | 10 | 10 | 20 | 40 | 10 | 434626 | 4586 | 327915 |
| ForwardChecking | 10 | 10 | 20 | 40 | 10 | 543457 | 3880 | 408240 |

On voit à nouveau que généralement il y a moins de parcours de nœuds pour l’algorithme ForwardChecking que pour le BackTracking en revanche le BackJumping ne trouve pas toutes les solutions, on ne peut donc pas exploiter ces résultats…