Rapport de Projet
Tabu Search

COUTABLE Guillaume, RULLIER Noémie
12 avril 2013

UNIVERSITÉ DE NANTES

Table des matières

1 Introduction

L'objectif de ce TP fut de comprendre et d'implémenter l'algorithme TabuSearch en utilisant la librairie JaCoP.

2 Présentation du TabuSearch

Tabu search est une heuristique de la méthode de recherche locale. Elle consiste dans un premier temps à générer affectation totale des variables. Puis calculer le coût de chaque mouvements possibles, pour ensuite effectuer le mouvement qui a un coût le moins important. Tabu search permet d'ajouter une contrainte, en effet on marque les p derniers mouvements effectués qui seront interdits. On répète cette dernière opération jusqu'à ce que le coût soit égal à 0 ou jusqu'à ce que le nombre maximum d'essai soit atteint.

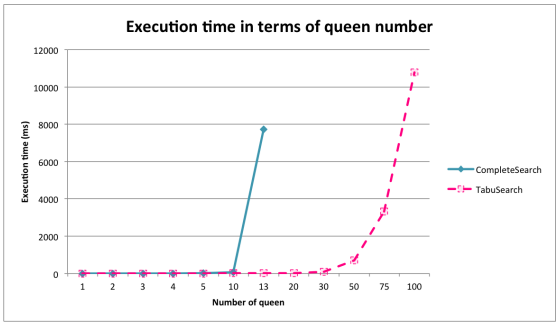
3 N-Queens avec TabuSearch et CompleteSearch comparaison

Ces comparaisons ont été effectuées pour différentes valeurs de n (représentant le nombre de reine). L'algorithme *CompleteSearch* a été exécuté avec l'heuristique *Depth First Search*.

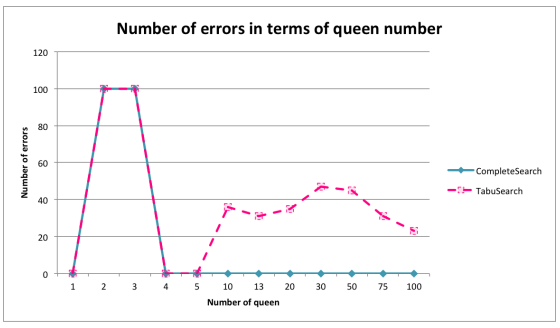
Pour chaque valeur n, l'algorithme a été lancé 100 fois. Le temps présenté est la moyenne de ces 100 résultats. Le *Number of errors* représente le nombre de fois où l'algorithme n'a pas trouvé de solution ; dans le cas du TabuSearch ce cas peut arriver lorsque le nombre maximum de tour de boucles est atteint sans qu'aucune solution soit trouvée.

CompleteSearch		
n	Number errors	Execution Time (ms)
1	0	0.2
2	100	0.45
3	100	0.77
4	0	1.6
5	0	2.77
10	0	60.22
13	0	7717.57
>16	No result (OutOfMemory)	

TabuSearch		
n	Number errors	Execution Time (ms)
1	0	0.13
2	100	0.83
3	100	1.01
4	0	0.47
5	0	0.56
10	36	2.98
13	31	10.52
20	35	20.55
30	47	102.39
50	45	692.37
75	31	3312.75
100	23	10776.28



(a) Execution time



(b) Number of errors

FIGURE 1 – Graph of comparaison between CompleteSearch and TabuSearch

On peut voir que pour n égal à 2 et 3, aucune solution n'est jamais trouvée. En effet, ce problème n'est pas consistent.

On peut de plus constater que pour $n > 13$, l'algorithme *CompleteSearch* ne permet pas de résoudre le problème. La façon dont celui-ci recherche les solutions engendre une exception *OutOfMemory*.

On peut remarquer que plus le nombre de n de reines augmente plus le temps d'exécution augmente (de façon exponentielle). Cette remarque est valable pour les deux algorithmes testés.

On peut cependant noter que le *TabuSearch* permet de trouver la solution pour un plus grand nombre de reines. Si l'on compare les deux algorithmes, pour le même nombre de reines donnés *TabuSearch* est plus rapide.

La dernière remarque que l'on peut faire, concerne le nombre d'erreurs. Ceux-ci ne sont pas réguliers pour le *TabuSearch* car en effet tout dépend de la première solution générée aléatoirement, du nombre d'itération maximum donnés (ici nous avons choisi 100) et de la taille de la liste des éléments tabous.

4 Ajout des conditions d'aspirations

On peut compléter l'heuristique de *TabuSearch* en ajoutant les conditions d'aspirations. Ces conditions d'aspirations permettent d'autoriser un élément tabou s'il améliore la meilleure solution. Pour ajouter les conditions d'aspirations à notre algorithme de *TabuSearch*. On effectue l'algorithme de *TabuSearch*, puis à chaque itération qui n'améliore pas la meilleure solution, on regarde dans la liste Tabou si un mouvement déjà effectué peut améliorer la meilleure solution.

5 TabuSearch algorithme avec procédure de redémarrage

L'ajout de la procédure de redémarrage, permet de relancer l'algorithme *TabuSearch* si aucune solution n'est trouvée avant que le nombre d'itérations maximum soit atteint.