Projet de Programmation par Contrainte

(Constraint Programming)

Dans le cadre de ce projet nous devons réaliser un algorithme de Branch and Prune. Dans un premier temps, nous réalisons une première version de cet algorithme qui n'effectue pas de pruning, mais qui fonctionne comme un simple algorithme de backtracking.

Nous analysons tout d'abord l'algorithme et son fonctionnement afin d'identifier quels sont les différents ensembles à représenter. Après cette analyse, nous remarquons que différentes structures sont nécessaires pour cet algorithme. Il faut pouvoir représenter l'ensemble des variables, leurs domaines (des ensembles ordonnés d'entiers), des nœuds (qui correspondent à des ensembles de nœuds), l'ensemble des derniers nœuds de notre arbre (L dans l'algorithme) et enfin les contraintes.

Nous choisissons donc des structures adéquates en fonction de l'utilisation de chacun de ces ensembles, afin de gagner en complexité. Nos choix sont les suivants :

- Les domaines sont représentés par des **List**. Il est nécessaire de pouvoir accéder à n'importe quelle valeur du domaine et de pouvoir la copier et dans la structure **List** ces deux opérations se font en O(n). De plus, lorsque nous améliorerons l'algorithme pour qu'il effectue du pruning, il sera nécessaire de pouvoir supprimer grace a une valeur passée en paramètre, au lieu d'un indice, ce qui est pratique avec **List**.

- Les nœuds sont représentés par des **vector**. La seule opération effectuée sur les nœuds est l'accès à ses valeurs. La structure **vector** permet cela en O(1).

- L'ensemble des variables n'est pas représenté par une structure. L'ensemble des variables étant supposé présenter des entiers triés par ordre croissant et correspondant toujours au nombre de domaines présents au sein d'un nœud, nous avons remarqué qu'il était plus simple de se contenter de prendre les indices des domaines contenus dans les nœuds. Ainsi la variable X_1 correspondra à l'indice du premier domaine contenu dans nœuds, autrement dit à l'indice de son domaine.

- L'ensemble de nœuds est représenté par une **Pile**. En effet on remarque qu'au sein de l'algorithme, on effectue uniquement des opérations d'ajout et de suppression sur cet ensemble, et ce toujours en tête. Une **Pile** est donc la structure toute indiquée pour cet ensemble.

- Les contraintes sont représentées directement par des fonctions au sein du code.

Cette version n'est pas totalement générique ou dynamique mais nous visons à implémenter cela pour la version finale. (Par exemple, nous avons pensé à un patron « Strategy » afin de pouvoir appliquer les contraintes correspondantes au problème choisi de façon dynamique).

Indications d'utilisation du programme :

Il faut compiler avec le standard c++11 (option : -std=c++11).

Il est possible de faire varier la valeur de « n » dans le problème des N-Queens en changeant la valeur de la variable globale « QUEENS_NUMBER » en début de fichier. Nous avons laissé en commentaires du code utilisé pour effectuer des tests (jusqu'à QUEENS_NUMBER = 12), il suffira donc de le dé-commenter (selon le « n » choisi) pour les effectuer a nouveau.