Aix-Marseille Université

Licence Informatique – Intelligence Artificielle

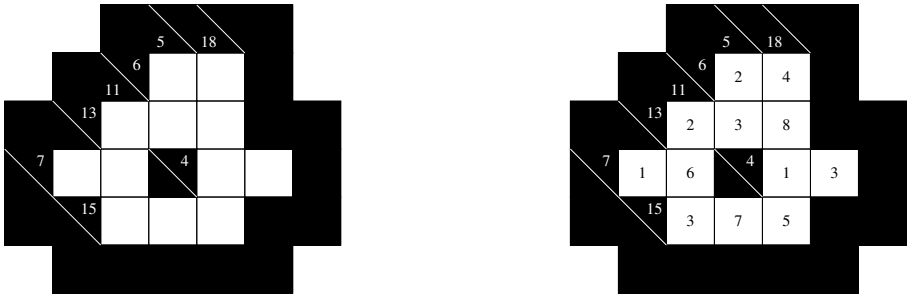
Projet

Dans ce projet, nous allons nous intéresser à un jeu logique mélangeant sudoku et mots croisés : le kakuro. À l'image du sudoku, le kakuro consiste à compléter une grille avec des chiffres compris entre 1 et 9 de sorte qu'une suite horizontale ou verticale de chiffres vérifie une définition comme pour les mots croisés. Les lettres étant remplacées par des chiffres, les définitions seront données par des sommes.

1 Présentation

Une instance kakuro se définit par la donnée d'une grille comportant des cases noires et des cases blanches (cette grille pouvant être de taille quelconque). Les cases noires permettent notamment de définir des plages consécutives de cases blanches horizontalement ou verticalement. Ces plages seront amenées à contenir les "mots numériques" (par analogie avec les mots croisés bien sûr). Parmi ces cases noires, certaines sont divisées en deux par une diagonale allant du coin supérieur gauche au coin inférieur droit. Les triangles inférieurs (respectivement supérieurs) ainsi définis peuvent contenir un nombre qui devra correspondre à la somme des chiffres du "mot numérique" inscrit dans les cases blanches situées sous la case noire (respectivement à sa droite). Ces sommes correspondent donc aux définitions des "mots numériques". Par ailleurs, chaque chiffre de 1 à 9 ne peut être utilisé qu'une seule fois par mot.

Un exemple d'instance kakuro est donnée dans la partie (a) de la figure ci-dessous. La solution de cette grille est présentée dans la partie (b).



(a) (b)

Les objectifs de ce projet sont d'utiliser les techniques d'IA (ici de programmation par contraintes) pour trouver une solution d'une grille donnée.

2 Modélisation d'une instance

Pour rechercher une solution d'une instance kakuro, nous allons modéliser le problème sous la forme d'une instance CSP. On associera une variable par case blanche de la grille. Chaque variable aura pour domaine initial $\{1, 2, \ldots, 9\}$. Quant aux contraintes, nous utiliserons deux types de contraintes :

- des contraintes binaires de différences (c'est-à-dire que si les variables x et y sont liées par une contrainte, celle-ci impose que x et y aient des valeurs différentes). En pratique, deux variables seront liées par une contrainte binaire si elles apparaissent dans un même mot.
- des contraintes n-aires qui lieront les variables correspondant à un même mot et qui garantiront que la somme des valeurs des variables a bien la bonne valeur.

3 Travail à réaliser

Pour des raisons d'efficacité, tous les algorithmes de résolution seront implémentés de façon itérative. Le travail qui vous est demandé est le suivant :

(i) Implémenter l'algorithme Backtrack,

(ii) Implémenter l'algorithme Forward-Checking n-aire (vous pouvez choisir la version que vous voulez parmi les différentes variantes),

(iii) Comparer l'efficacité de Backtrack et de Forward-Checking en termes de temps, de nombre de nœuds et de nombre de tests de contraintes. Une archive contenant quelques grilles de jeu est disponible sur la page AMeTICE de l'UE pour effectuer ces comparaisons. Vous pouvez bien entendu tester sur vos propres grilles.

(iv) Proposer des heuristiques de choix de variables adaptées au jeu du kakuro.

(v) Proposer des améliorations de l'algorithme Forward-Checking en tenant compte des spécificités du jeu.

(vi) Comparer l'efficacité des heuristiques et des améliorations proposées avec votre Forward-Checking initial doté d'une heuristique de type *dom/deg* (voir TD) en termes de temps, de nombre de nœuds et de nombre de tests de contraintes.

4 Format de fichier

La syntaxe des fichiers contenant des grilles de kakuro est la suivante :

— Chaque ligne du fichier représente une ligne de la grille.

— Chaque colonne est séparée par un espace.

— Le caractère "." désigne une case blanche vide.

— Étant donnés deux entiers `x` et `y`, "`x\y`" indique que le mot vertical aura pour somme `x` et le mot horizontal pour somme `y`. "`\`" représentera donc une case noire sans chiffre associé. "`x\`" et "`\y`" représenteront des cases noires ne contenant la définition que d'un mot respectivement vertical et horizontal.

5 Rendu du travail

Vous devrez fournir les code sources en C des différents algorithmes ainsi qu'un rapport. Ce rapport devra contenir :

1. Un manuel d'utilisation de votre programme.
2. Les explications concernant vos implémentations des algorithmes Backtrack et Forward-Checking.
3. Les résultats et/ou courbes de comparaisons entre Backtrack et Forward-Checking.
4. Une description des heuristiques de choix de variables que vous proposez.
5. Une description des améliorations proposées pour l'algorithme Forward-Checking.
6. Les résultats et/ou courbes de comparaisons entre les différentes heuristiques et/ou versions de Forward-Checking.
7. Pour chaque comparaison, vous donnerez une interprétation des résultats obtenus.

6 Quelques conseils

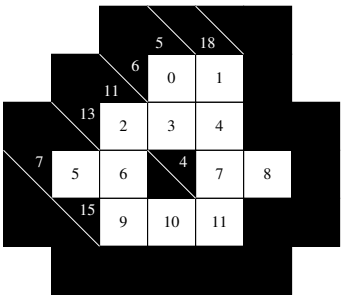
6.1 Structures de données

Il est fortement conseillé de prendre le temps de la réflexion avant d'arrêter un choix pour vos structures de données. En particulier, les structures de données choisies devront permettre une mise en œuvre aisée et efficace des algorithmes de résolution Backtrack et Forward-Checking. Il convient pour cela de représenter chaque variable avec son domaine (son domaine courant dans le cas de Forward-Checking) ainsi que les contraintes. Pour chaque contrainte, on a besoin de connaître l'ensemble des variables sur

lesquelles elle porte ainsi que la relation qui lui est associée. Les tests de consistances de Backtrack ou le filtrage de Forward-Checking nécessitent d'avoir un accès rapide aux contraintes dans lesquelles apparaît la variable courante. Ainsi, une représentation de l'ensemble des contraintes analogue à celle des listes d'adjacences pour les graphes peut être pertinente. Concernant les relations associées aux contraintes, elles peuvent être représentées en extension comme en intension. Cependant, compte tenu du jeu qui nous intéresse ici, une représentation en intension semble plus naturelle.

6.2 Lecture du fichier

L'objectif de la lecture du fichier est de prendre connaissance de la grille et de la représenter en mémoire sous la forme d'une instance CSP selon la modélisation donnée précédemment. À cette fin, il faut attacher une variable à chaque case blanche en numérotant, par exemple, les variables de 0 jusqu'au nombre de cases blanches moins un. La figure ci-dessous donne un exemple de numérotation possible des variables pour la grille donnée précédemment dans l'énoncé. Une telle numérotation permet ensuite de facilement créer les contraintes au fur et à mesure de la lecture de la grille.



Pour vous faciliter la lecture, un parser est mis à votre disposition. Il ne vous reste qu'à l'adapter à vos structures de données.

6.3 Implémentation de Backtrack et Forward-Checking

Les algorithmes Backtrack et Forward-Checking sont généralement présentés sous forme récursive. Toutefois, leurs implémentations dans des langages comme le langage C doivent être effectuées de manière itérative afin d'obtenir une meilleure efficacité pratique. Cela nécessite donc d'implémenter et gérer soi-même l'équivalent de la pile de récursion. Il est notamment important de mémoriser toutes les informations nécessaires (choix de variable ou de valeur, valeurs supprimées par filtrage, ...) afin de pouvoir revenir en arrière en cas d'échec.

Plusieurs heuristiques de choix de variables seront testées (l'heuristique dom/deg et au moins une heuristique que vous proposerez). Aussi, il est important de prévoir cette possibilité dès le départ dans votre implémentation de Backtrack ou de Forward-Checking.

6.4 Débogage et évaluation expérimentale

Durant la phase de débogage, je vous invite à compiler en utilisant les options `-Wall` et `-g3` du compilateur `gcc` et d'utiliser alternativement un débogueur comme `gdb` et un programme vérifiant tous les accès à la mémoire comme `valgrind`.

Une fois votre programme au point, il est conseillé de le compiler avec l'option d'optimisation `-O3` du compilateur `gcc` afin d'obtenir un exécutable le plus performant possible. Dans le cas de figure où vos expérimentations sont effectuées sur un ordinateur portable, il est important de s'assurer que la politique de gestion de l'énergie est réglée sur la puissance maximum afin de ne pas biaiser les comparaisons.