Solveurs et générateurs pour des jeux de logique

Analyse des besoins

Martial DUVERNEIX, Florian GAUTIER, Pierre LORSON, Teiki PEPIN

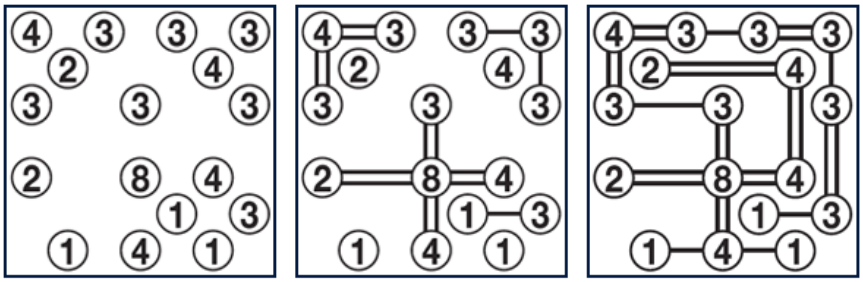
24 janvier 2018

Table des matières

1 Introduction

Ce projet consiste à concevoir des programmes pouvant générer et résoudre des instances des jeux de logique *Hashiwokakero* (figure 1) et *Inshi no heya* (figure 2). De manière similaire au *Sudoku*, ces deux jeux proviennent du journal japonais Nikoli [NC] sont structurés sous forme d'une grille qui se complète à l'aide d'un crayon. Dans le cadre de ce projet, on souhaite généraliser les méthodes implémentées pour la résolution et la génération de telle façon à ce que la grille de jeu puisse avoir des dimensions variables.

FIGURE 1 – Étapes de résolution du puzzle *Hashiwokakero*.



De plus, la résolution de ces puzzles devra être optimisée. Il sera nécessaire d'utiliser des méthodes proches de celles des SAT-solvers en se basant sur le traitement d'heuristiques. On pourra aussi mettre en place des structures de données optimisées, une architecture SIMD pour appliquer une instruction à de multiples données et une exécution sur plusieurs threads dans le but de réduire le temps requis pour la résolution des grilles.

FIGURE 2 – Étapes de résolution du puzzle *Inshi no Heya*.

2 Description et analyse de l'existant

Nous disposons dans le cadre de ce projet de code source en C similaire à celui qui est attendu de nous pour la résolution et la génération de grilles de Sudoku. Ce code nous a été directement fourni par le client dans le but de nous guider. Il met en place la grande majorité des fonctionnalités que nous devons implémenter mais dans le contexte de la résolution du Sudoku : des structures de données compactes, des opérations bit à bit, des fichiers d'entrées pour des tests de cas particuliers, etc...

Nous pourrons nous servir d'un court extrait d'un document rédigé par Daniel Andersson [And09] qui représente le puzzle *Hashiwokakero* en tant que problème SAT et démontre que celui-ci est NP-complet.

De même, nous disposons d'un document de T. Morsink [Has09] qui présente une méthode de résolution de *Hashiwokakero* par solveur SAT.

Il existe également un article écrit par Patrick Prosser and Chris Unsworth [PU06] traitant de la représentation des données de *Hashiwokakero* sous forme d'arbre couvrant où chaque noeud représente une île et les ponts sont représentés par les arêtes.

Enfin, nous avons également un article [MEP12] qui explore la résolution de Hashiwokakero par un parcours en profondeur du graphe le représentant. Les techniques de résolutions applicables y sont détaillées.

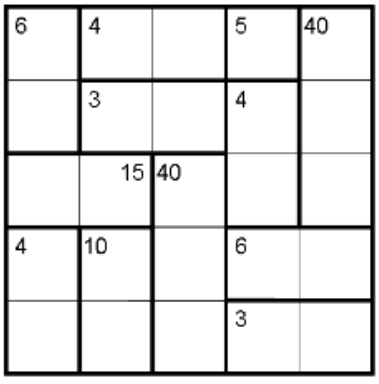
3 Description des besoins

3.1 Besoins fonctionnels

— Permettre l'affichage d'une grille de jeu :
  — Concevoir pour chaque puzzle un format d'affichage qui représente tous les éléments qui le compose et qui satisfait un bon niveau de lisibilité pour un humain.
  — Implémenter pour chaque programme une fonctionnalité d'affichage permettant d'afficher le puzzle en question sur la sortie standard.
  — Implémenter pour chaque programme la possibilité d'enregistrer la représentation de la grille dans un fichier, particulièrement pour les cas où la grille aurait des dimensions trop larges pour s'afficher correctement en console.

— Permettre la lecture de grilles enregistrées dans des fichiers :
  — Définition d'un format de représentation des grilles pour chaque jeu optimisé pour la lecture par ordinateur (figure 3). Ce format peut donc être différent du format de représentation destiné à l'affichage.
  — Permettre à chaque programme de lire les fichiers du format qui lui correspond et de convertir les données lues en une structure de données utilisable par le reste du programme.

FIGURE 3 – Prototype de format de fichier pour *Inshi no Heya* et sa grille correspondante.



templateInshi | templateHashiwokakero

```
1  6b2
2  4r2
3  5
4  40b3
5  3r2
6  4b2
7  15r2
8  40b3
9  4b2
10 10b2
11 3r2
12
```

— Mettre en place, pour chaque jeu, un système de résolution de grilles :
  — La résolution devra proposer une solution à chaque grille qui lui est fournie, et ce, que la grille admette une unique solution ou plusieurs.
  — La résolution devra se terminer et afficher un message correspondant si la grille fournie n'admet aucune solution.
  — Il sera possible d'afficher à chaque étape de la résolution un message explicatif de ce qu'il s'y passe.

— Mettre en place, pour chaque jeu, un système de génération de grilles :
  — L'utilisateur aura la possibilité de décider si la grille générée admet au moins une solution ou une unique solution.
  — Permettre la sauvegarde de grilles générées dans des fichiers au même format que celui utilisé pour la lecture de grille en entrée.

— Mettre en place une taille de grille variable pour chaque jeu :
  — Définir les dimensions minimums et maximums qui seront supportées par les structures de données utilisées.
  — Détecter automatiquement les dimensions des grilles fournies aux programmes.
  — Permettre de choisir les dimensions désirées lors de la génération d'une nouvelle grille.

— Chaque programme devra disposer de plusieurs options d'exécution dont :
  — *-verbose* pour afficher chaque étape de la résolution de la grille.
  — *-output* pour enregistrer la grille résolue dans un fichier sans l'afficher préalablement en sortie standard.
  — *-generate* pour générer une nouvelle grille avec au moins une solution.
  — *-strict* pour générer une nouvelle grille avec une unique solution.
  — *-help* pour afficher des informations sur le programme et ses options d'exécution.
  — *-version* pour afficher la version courante du programme.
  — Aucune option d'exécution nous permettra de lancer la résolution sur le fichier transmis en paramètre et d'afficher en sortie standard la grille résolue.
  — Certaines de ces options pourront être compatibles entre elles tan-

dis que d'autres seront mutuellement exclusives.

3.2 Besoins non fonctionnels

— Chacun des deux jeux de logique fera l'objet d'un fichier exécutable permettant de réaliser les opérations de résolution et de génération correspondant à son jeu.

— Les grilles de jeu pouvant être de grande taille, la résolution et la génération doivent être optimisées :
  — Il est nécessaire d'utiliser un langage de programmation efficace en utilisation de la mémoire et surtout en temps de calcul. Il a donc été convenu avec le client lors du premier entretien que les programmes seraient implémentés en C dans la mesure du possible.
  — Mettre en place des algorithmes de résolution optimaux afin de trouver une solution en un minimum de temps.
  — Utiliser de manière intensive des algorithmes SWAR pour réduire le temps de calcul.
  — Exécuter plusieurs threads afin de répartir les calculs.

— Des grilles de tests devront être fournies. Ces grilles devront permettre de tester pour la résolution des cas généraux ainsi que des cas particuliers tels que : grille avec aucune solution, grille avec une unique solution, grille de taille minimum, grille de taille maximum, grille vide, etc...

— La réalisation du projet devra respecter le calendrier prévisionnel de l'UE Projet de Programmation :
  — Une première version du livrable devra être fournie pour le 16 février.
  — Un audit lors du 28 février.
  — Une version final du livrable ainsi qu'un mémoire devra être fournie pour le 5 avril.
  — Une soutenance aura lieu le 11 avril.

4 Diagramme de Gantt

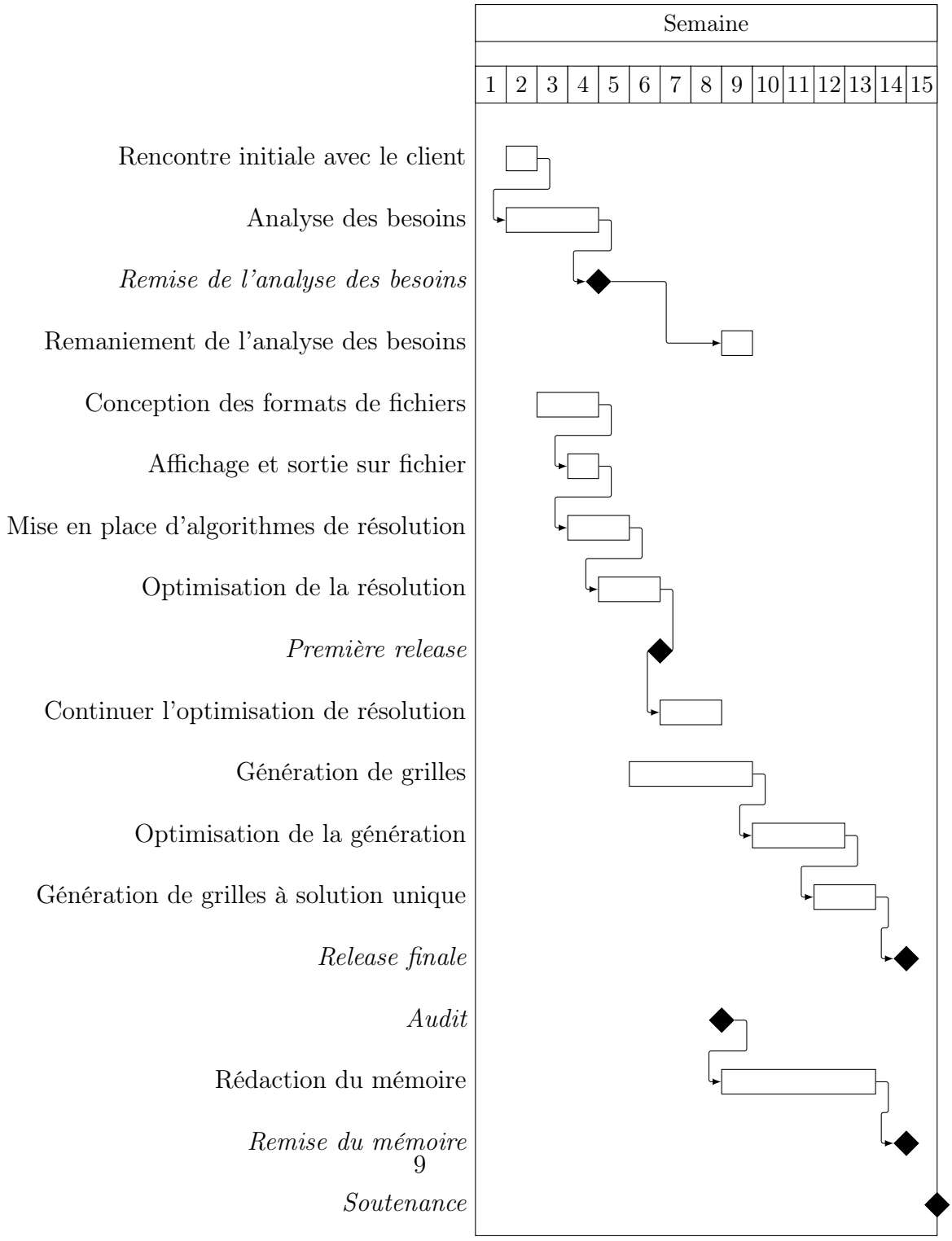


FIGURE 4 – Distribution de la charge de travail au cours du semestre.

5 Table des figures

6 Références

[And09] Daniel Andersson. Hashiwokakero is np-complete. *Information Processing Letters*, 109(19) :1145–1146, 2009.

[Has09] Morsink T Hashiwokakero. Available on : http ://www. liacs. nl/assets/bachelorscripties/2009-11timomorsink. pdf. *Last access*, 20, 2009.

[MEP12] Reza Firsandaya Malik, Rusdi Efendi, and Eriska Amrina Pratiwi. Solving hashiwokakero puzzle game with hashi solving techniques and depth first search. *Bulletin of Electrical Engineering and Informatics*, 1(1) :61–68, 2012.

[NC] Ltd. Nikoli Co. Présentation et description des puzzles publiés par nikoli. `http://nikoli.co.jp/en/puzzles/index.html`. [Visité le 24 Janvier 2018].

[PU06] Patrick Prosser and Chris Unsworth. Rooted tree and spanning tree constraints. In *17th ECAI workshop on modelling and solving problems with constraints*, 2006.