Solveurs et générateurs pour des jeux de logique

Analyse des besoins

Martial DUVERNEIX, Florian GAUTIER, Pierre LORSON, Teiki PEPIN

1er février 2018

Table des matières

1 Introduction

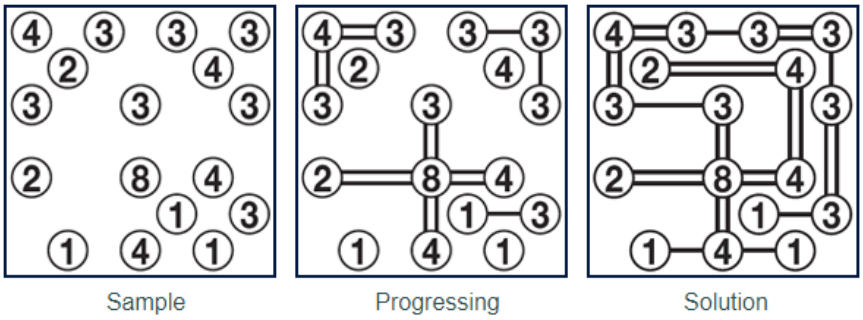
Ce projet consiste à concevoir des programmes pouvant générer et résoudre des instances des jeux de logique *Hashiwokakero* (figure 1) et *Inshi no heya* (figure 2). De manière similaire au *Sudoku*, ces deux jeux proviennent du journal japonais Nikoli [1] et sont structurés sous forme d'une grille qui 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.

De plus, la résolution de ces puzzles devra être optimisée. Il sera nécessaire d'utiliser des méthodes proches de celles des solveurs SAT : il faut déterminer une valuation de variables booléennes exprimants les contraintes de la grille de jeu telle que toutes les contraintes soient satisfaites.

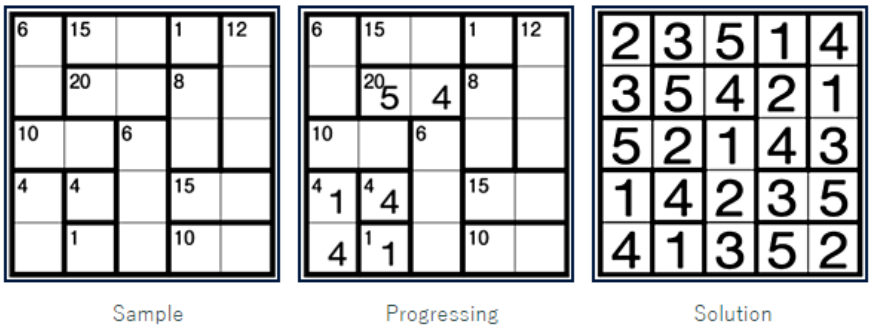
Hashiwokakero est composé d'îles contenant des chiffres allant de 1 à 8. Il faut relier toutes les îles par des "ponts" pour former un seul groupe (figure 1). Les ponts doivent être en ligne droite, ne peuvent pas se croiser entre eux et ne peuvent pas survoler une île. Chaque île doit avoir exactement autant de ponts que le nombre qu'elle indique. Dans la variante originale du jeu, deux îles ne peuvent pas être connectées directement entre elles par plus de deux ponts.

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



Inshi no heya contient des "salles" délimitées par des traits épais avec un nombre écrit en haut à gauche. Ces salles ont l'une de leurs dimensions (la largeur ou la longueur de la salle) fixée à une case. Le but est de remplir la grille en insérant des nombres de 1 à N (où N est la taille de la grille) dans chaque case de façon à ce que le produit de tous les nombres d'une salle soit égal au nombre indiqué originalement dans cette salle (figure 2). Un nombre présent dans une case ne peut pas être répété dans une autre case de la même rangée ou de la même colonne.

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



Nous avons choisi deux jeux plutôt différents afin d'augmenter la diversité des structures de données et algorithmes mis en place au sein du projet. Cela nous permet aussi de fournir au client deux exemples de générations et résolutions de problèmes distincts ce qui pourrait être utile pour de prochains projets similaires.

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.

On y observe notamment :

— L'utilisation de *uint64_t* pour stocker les valeurs des cellules. Ce type provenant de la bibliothèque standard *stdint.h* permet de définir le nombre de bits qui seront utiliser par une variable. On y retrouvera d'ailleurs *uint_fast64_t* qui est un type spécialisé dans la rapidité des opérations.

— L'implémentation d'une liste chaînée d'une structure conservant chacune la grille d'origine et les modifications qui lui ont été apportées à chaque instant. Cela permet d'effectuer rapidement du backtracking lorsque nécessaire.

— L'application d'opération bit à bit au sein d'algorithmes SWAR lors du traitement des données au sein de la grille. Ceci permet d'appliquer une instruction à de multiples données en parallèle afin de réduire le nombre d'opérations nécessaires.

Nous pourrons nous servir d'un court document rédigé par D. Andersson [2] qui démontre que le problème *Hashiwokakero* est un problème NP-complet. Nous savons donc qu'il n'existe pas de moyen de trouver une solution à une grille en un temps polynomial. Nous devrons donc tester un ensemble de valeurs sur la grille et effectuer un backtracking lorsque la grille devient impossible à résoudre jusqu'à obtenir une solution, ou alors, jusqu'à que l'on puisse admettre l'absence de solution.

De même, nous disposons également d'un article de T. Morsink [3] sur le puzzle *Hashiwokakero*. Celui-ci décrit :

— Les contraintes basiques qui nous permettent de connecter des îles entres elles sans avoir recours au backtracking.

— Les étapes de résolution effectuées par le solveur réalisé par l'auteur.

— Les étapes de génération pour implémenter un générateur de nouvelles grilles.

— Une comparaison des temps pris par le générateur de l'auteur, son sol-

veur et un autre solveur externe pour générer et résoudre des grilles de taille allant de 7x7 à 15x15.

Afin de représenter les îles et les ponts du jeu *Hashiwokakero*, nous pourrons prendre appui sur une représentation sous forme d'arbre couvrant où chaque noeud représente une île et les ponts sont représentés par des arêtes. Cette implémentation est détaillé au sein de la publication "Rooted Tree and Spanning Tree Constraints" [4].

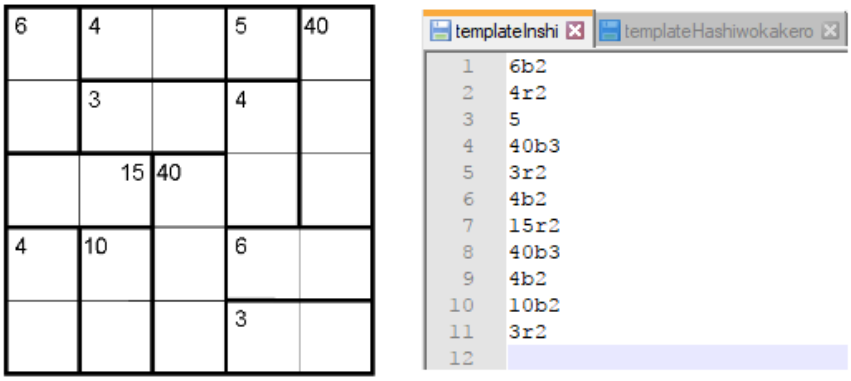
3 Description des besoins

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 soit lisible et compréhensible par tout le monde.
  - 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 excessives pour s'afficher correctement en console.
  - Ces fonctionnalités sont essentielles au bon fonctionnement du programme du fait de leur nécessité pour vérifier l'exécution des autres fonctionnalités. Cela sera assuré par l'utilisation de grilles de test dont les sorties sur console et sur fichier auront été prédéterminées.

- 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 via un parseur que nous implémenterons.
  - De part la nature du programme, la lecture des fichiers est un besoin essentiel de ce projet. Le bon fonctionnement de cela sera aussi assuré par la lecture de grilles dont les valeurs qui devront être lues par le programmes seront prédéterminées.

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



- Mettre en place, pour chaque jeu, un système de résolution de grilles :
  - La résolution s'effectuera en étapes dans l'ordre suivant afin d'améliorer sa rapidité :
    - Détermination des éléments évidents de la grille de jeu.
    - Construction des variables de la grille dont les valeurs restent à déterminer.
    - Recherche d'une valuation de ces variables qui valide la grille.
  - 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.
  - La résolution devra se terminer et afficher un message correspondant si la grille fournie est dans format incorrect.
  - Il sera possible d'afficher à chaque étape de la résolution l'état actuel de la grille.
  - La résolution est un point essentiel du programme. Le suivi des étapes n'est pas nécessaire au bon fonctionnement du programme, il est considéré comme conditionnel.
  - La validité de la résolution sera vérifié à l'aide de grilles de tests dont les résultats auront été prédéterminés.

- Mettre en place, pour chaque jeu, un système de génération de grilles :
  - Le programme générera par défaut une grille ayant au moins une

solution qu'il affichera à la sortie standard. Ceci est essentiel au programme.
  - L'utilisateur aura la possibilité de décider si la grille générée ne doit admettre qu'une unique solution. Cette fonctionnalité est une amélioration supplémentaire apportée au programme après la réalisation de ses fonctionnalitées clés, elle est conditionnelle.
  - Permettre la sauvegarde des grilles générées dans des fichiers au même format que celui utilisé pour la lecture de grilles en entrée. La génération de grilles ayant pour but la réutilisation de ces grilles dans la résolution, cette fonctionnalité est essentielle.
  - Diverses grilles générées seront résolues à la main afin de vérifier la validité de la génération.

- 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.
    - Pour *Hashiwokakero*, le minimum valide est une grille 3x1 (ou bien 1x3). Le maximum dépendra de notre implémentation mais une taille idéale serait de 64x64.
    - Pour *Inshi no heya*, le minimum valide est une grille 1x1. Le maximum sera fortement limité par les types utilisés au sein de l'implémentation car la valeur attribuée à une salle est, au pire des cas, $N!$ où N est la taille de la grille.
  - Détecter automatiquement les dimensions des grilles fournies aux programmes.
  - Permettre de choisir les dimensions désirées lors de la génération d'une nouvelle grille.
  - La robustesse du programme doit être assurée : il ne doit pas être possible d'aller aux delà des bornes déterminées pour les dimensions.
  - Pour chacun des programmes, des grilles de test correspondant aux limites des dimensions seront utilisées afin de vérifier la correspondance de la résolution à des résultats préalablement déterminés.
  - Pour chacun des programmes, la validité des grilles générées aux limites des dimensions sera vérifié par la résolution de ces grilles.
  - Ces fonctionnalités de scalabilité sont essentielles.

- Chaque programme devra disposer de plusieurs options d'exécution :

— Comportement par défaut :
  — Prend un paramètre : le nom du fichier contenant la grille à résoudre.
  — Lance la résolution d'une grille contenue dans un fichier et affiche en sortie standard la grille initiale puis la grille résolue.
  — Ceci est, bien évidemment, essentiel au projet.

— *-generate* :
  — Prend un paramètre : la taille de la grille à générer.
  — Permet de générer une nouvelle grille avec au moins une solution dont la taille est détérminée par l'utilisateur.
  — Est compatible avec toutes les autres options d'exécution.
  — Cette fonctionnalité est un des points clés du programme et est donc essentielle.

— *-strict* :
  — Force la grille générée à n'avoir qu'une unique solution.
  — Cette option doit être utilisée avec l'option de génération et est compatible avec toutes les autres options d'exécution.
  — Tel que vu précédemment, cette fonctionnalité est conditionnelle.

— *-save* :
  — Prend un paramètre : le nom du fichier de sortie.
  — Si le format de fichier d'entrée et différent de celui utilisé pour l'affichage de grilles, cette option permet d'enregistrer une grille générée au format d'entrée.
  — Cette option doit être utilisée avec l'option de génération et est compatible avec toutes les autres options d'exécution.
  — Cette fonctionnalité est essentielle à implémenter pour les tests de génération.

— *-output* :
  — Prend un paramètre : le nom du fichier de sortie.
  — Permet d'enregistrer un affichage compréhensible de la grille résolue ou générée dans un fichier sans l'afficher au préalable en sortie standard.
  — Est compatible avec toutes les autres options d'exécution.

    - Cette fonctionnalité est essentielle.

- *-verbose* :
    - Permet d'afficher chaque étape de la résolution ou de la génération de la grille.
    - Est compatible avec toutes les autres options d'exécution.
    - Tel que vu précédemment, cette fonctionnalité est conditionnelle.

- *-help* :
    - Permet d'afficher des informations sur le programme et ses options d'exécution.
    - Cette option peut être utilisée avec toute autre option.
    - Ceci apporte des informations supplémentaires non requis pour l'utilisation du programme, son implémentation est donc conditionnelle.

- *-version* :
    - Permet d'afficher la version courante du programme.
    - Cette option peut être utilisée avec toute autre option.
    - Ceci apporte des information supplémentaires non nécessaires à l'utilisation du programme, son implémentation est donc conditionnel.

3.2 Besoins non fonctionnels

- Un besoin essentiel du projet est que 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 faible en consommation de mémoire et efficace 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.
    - Il est essentiel de mettre en place des algorithmes de résolution

optimaux afin de déterminer tous les éléments certains d'une grille en moins d'une seconde.
  - Utiliser de manière intensive des algorithmes SWAR. Cela désigne le fait d'appliquer une seule instruction sur de multiples données contenu dans un registre. Dans notre cas, on utilisera ces algorithmes pour réduire le temps de calcul en regroupant des données et en utilisant une seule instruction sur celles-ci ce qui est plus rapide que de réitérer la même instruction au sein d'une boucle. Ceci est essentiel au projet.
  - Il faut se servire de structures de données concises utilisant des types optimisés pour du traitement bit à bit tels que *uint64_t* et *uint_fast32_t*. Ceci est aussi essentiel.
  - Exécuter plusieurs threads afin de répartir les tâches effectuées lors de la résolution et la génération de grilles. Ceci a été vu comme une fonctionnalité supplémentaire pouvant être ajoutée au projet afin de l'améliorer au delà de ce qui est attendu. Son implémentation est donc optionnelle.

- 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 Diagrammes de Gantt

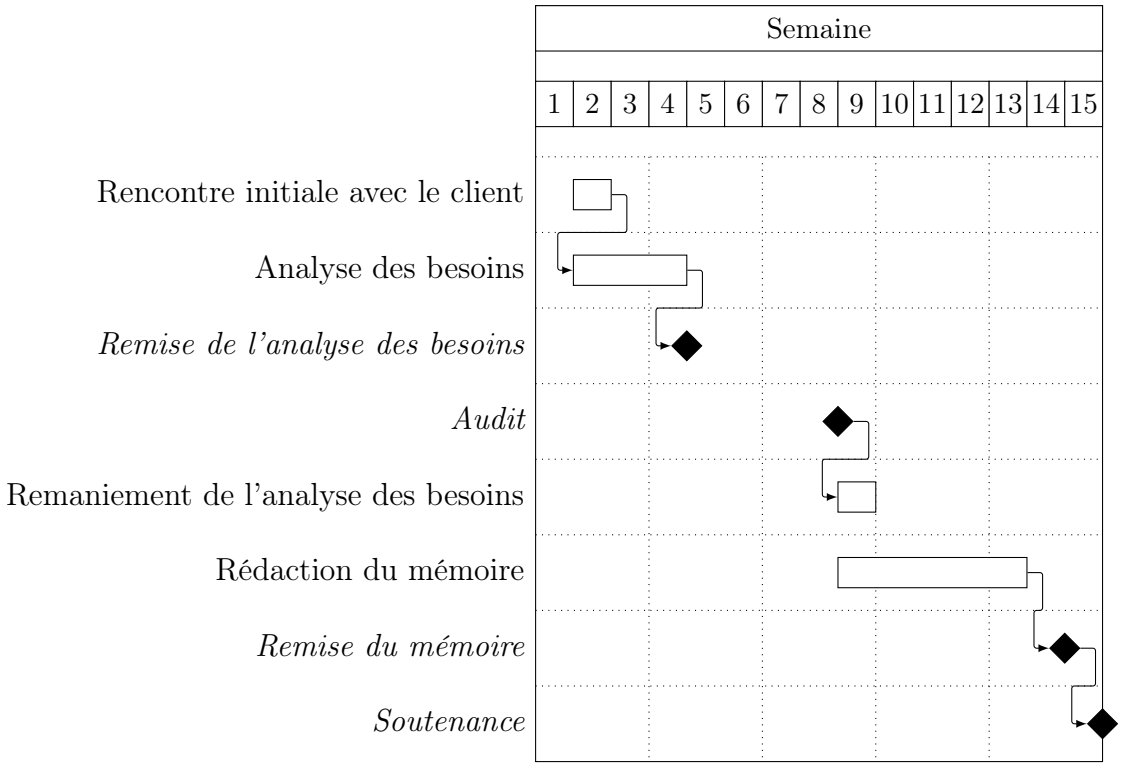


FIGURE 4 – Planning de la rédaction des documents propres au projet.

La répartition des tâches suivante s'applique à chacun des deux puzzles de manière identique. Les deux programmes seront développés en parallèle.

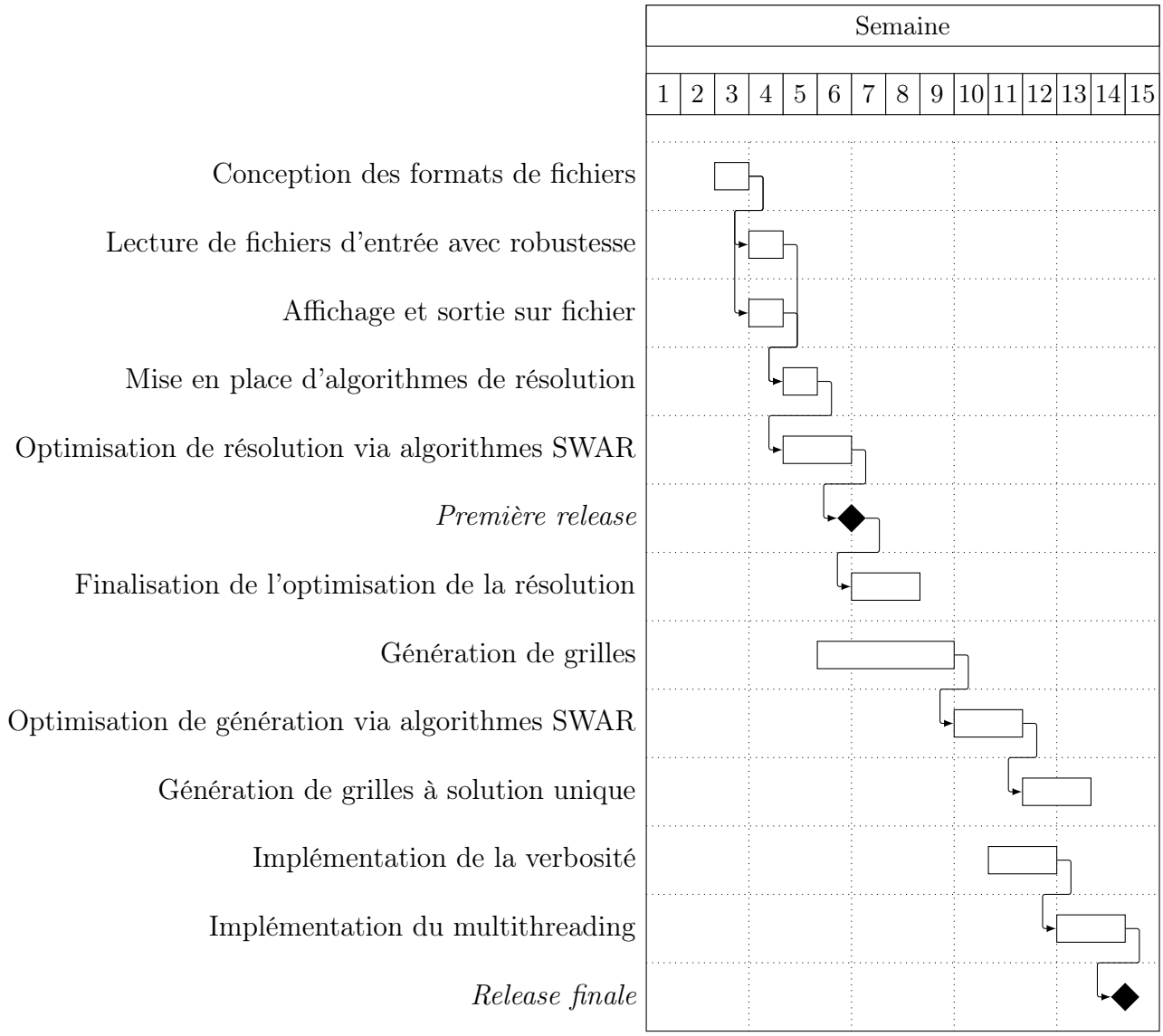


FIGURE 5 – Distribution de la charge de travail au cours du semestre.

5 Références

[1] N. Co., "Présentation et description des puzzles publiés par Nikoli." `http://www.nikoli.co.jp/en/puzzles/index.html`. [Visité le 24 Janvier 2018].

[2] D. Andersson, "Hashiwokakero is NP-complete," *Information Processing Letters*, vol. 109, no. 19, pp. 1145–1146, 2009.

[3] T. Morsink, "Hashiwokakero." `http://www.liacs.nl/assets/Bachelorscripties/2009-11TimoMorsink.pdf`. [Visité le 31 Janvier 2018].

[4] P. Prosser and C. Unsworth, "Rooted tree and spanning tree constraints," in *17th ECAI workshop on modelling and solving problems with constraints*, 2006.