UNIVERSITÉ DU QUÉBEC À MONTRÉAL

JEUX COMBINATOIRES ET FORMULES BOOLÉENNES QUANTIFIÉES (QBF)

ENCODAGES POUR LE JEU DE TIC-TAC-TOE D'HARARY

THÈSE

PRÉSENTÉE

COMME EXIGENCE PARTIELLE

DU DOCTORAT EN INFORMATIQUE

PAR

STEVE BOUCHER

JANVIER 2026

REMERCIEMENTS

Ce travail de recherche n'aurait pas été possible sans la collaboration de plusieurs personnes à qui j'aimerais adresser mes plus sincères remerciements.

Je tiens d'abord à remercier mon directeur de recherche, M. Roger Villemaire. Il a été disponible pour répondre à mes questions durant ces années, et ses précieux conseils m'ont guidé durant toute ma thèse.

Je veux aussi remercier la direction de Rheinmetall Canada inc. où je travaille depuis plus de vingt et un ans. Elle m'a offert le financement et accordé le temps nécessaire pour que je me consacre à mes études, et ce, depuis le début de mon parcours universitaire. Elle m'a aussi donné accès au matériel indispensable pour faire mes analyses. Elle offre un appui constant à ses employés dans la poursuite de leurs études, quels que soient leur parcours et leur niveau, et je lui en suis très reconnaissant.

Je tiens à remercier Mme Julie Lalancette pour ses suggestions en langue française.

Enfin, je tiens à remercier mes amis titulaires d'un doctorat, car chacun, à leur façon, m'a conseillé et encouragé tout au long de ma thèse.

Merci à tous.

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

ACRONYMES

ALO At-Least-One.

AMO At-Most-One.

CDCL Conflict Driven Clause Learning.

CEGAR Counterexample-guided Abstraction Refinement.

CNF Conjonctive Normal Form.

COR Corrective Encoding.

COR+ Polished Encoding.

DNF Disjonctive Normal Form.

DPLL Davis–Putnam–Logemann–Loveland.

DQBF Dependency Quantified Boolean Formula.

DYS QBF encoding of generalized Tic-Tac-Toe.

EO Exactly-One.

HTTT Harary's Tic-Tac-Toe (Tic-Tac-Toe d'Harary).

NP Nondeterministic polynomial time (Temps polynomial non déterministe).

PNF Prenex Normal Form (Forme normale prenex).

QBF Quantified Boolean Formula (Formule Booléenne Quantifiée).

QCDCL Quantified Conflict Driven Clause Learning.

SAT Problème de satisfiabilité booléenne.

SMT Problème de satisfaisabilité modulo théories.

RÉSUMÉ

Les jeux combinatoires posent des défis complexes et nécessitent des raisonnements sophistiqués pour déterminer si un joueur a, ou non, une stratégie gagnante, et ce, indépendamment des choix de l'autre joueur. Les formules booléennes quantifiées (QBF), où l'alternance de quantificateurs universels et existentiels est de nature interactive, possèdent une sémantique naturelle en termes de jeux, ce qui en fait une représentation totalement adéquate et prometteuse pour la détermination de stratégies gagnantes. Toutefois, en pratique, la résolution de jeux à l'aide des solveurs QBF actuels reste difficile dans des délais raisonnables. Néanmoins, les QBF offrent un formalisme très souple et versatile qui permet d'exprimer de nombreuses conditions, de manières très différentes. L'objectif de cette thèse est d'explorer cette versatilité pour analyser les stratégies gagnantes du jeu de Tic-Tac-Toe d'Harary (HTTT), un *achievement game* bien connu qui a suscité l'intérêt autant des théoriciens que des membres de la communauté QBF.

À la différence des travaux existants sur les encodages QBF pour les jeux combinatoires, cette thèse innove en exploitant la dualité entre les joueurs et en introduisant des encodages tant pour l'existence d'une stratégie gagnante pour le premier joueur que pour l'existence d'une stratégie bloquante pour le second. De plus, ce travail montre qu'on peut formaliser avec profit l'existence de stratégies spécifiques, tout en ne réduisant pas la portée de la méthode en termes pratiques.

Tous les encodages introduits dans cette thèse sont évalués expérimentalement de façon détaillée sur un vaste ensemble d'instances HTTT. Cela permet non seulement d'établir la performance des méthodes développées, comparativement aux encodages existants, à l'aide des solveurs QBF les plus performants, mais aussi de tirer des conclusions générales sur le jeu de HTTT. Cette dernière avancée est d'importance puisque, au-delà de la performance de la résolution de HTTT, il y a un grand intérêt en combinatoire pour déterminer s'il y a, ou non, une stratégie gagnante pour les instances particulières de HTTT.

Cette thèse contribue donc à l'avancement des connaissances et de la performance de l'approche QBF, tout en appliquant ses méthodes à un jeu, en l'occurence HTTT, qui est d'un intérêt allant bien au-delà de la communauté QBF.

INTRODUCTION

Dans le jeu de Tic-Tac-Toe d'Harary (HTTT) (1), deux joueurs s'affrontent en déposant, tour à tour, une pierre d'une couleur différente sur un plateau. Par convention, dans les présentations récentes, le premier joueur est appelé Noir et le second, Blanc. Le premier joueur dépose donc des pierres noires et le second joueur, des pierres blanches. L'objectif du jeu consiste à réaliser, à une translation, rotation et réflexion près, une forme prédéterminée. Dans le jeu usuel, cette forme est toujours un *animal* ou *polyomino*, c'est-à-dire un ensemble de cases connexes par les arêtes. Il s'agit d'un *achievement game* (2), puisque une pierre déposée sur le plateau n'est jamais retirée ni déplacée. Le but du jeu est simplement de réaliser la forme.

On sait si le premier joueur a une stratégie gagnante dans le Tic-Tac-Toe d'Harary sur un plateau infini pour tous les polyominos à l'exception du *Snaky* (1) dont le statut exact est un des grands problèmes non résolus dans la communauté des jeux combinatoires. Lorsqu'il existe une stratégie gagnante pour une certaine forme, on dit que celle-ci est gagnante (*winner*) sinon elle est perdante (*loser*). Harary a fait la conjecture (1) que le Snaky est une forme gagnante, mais la question demeure ouverte après de nombreuses années. Il reste néanmoins que la détermination du statut (gagnant/perdant) d'une forme sur des plateaux de taille finie n'est pas aussi simple et réserve même quelques surprises, comme nous le verrons au chapitre 7. Ce jeu a suscité beaucoup d'intérêt en mathématiques combinatoires où il a été étendu à des plateaux hexagonaux (3), unidimensionnels (4) et multidimensionnels (5), et il a aussi attiré l'attention de la communauté QBF qui a introduit les plateaux en forme de tore (6). Dans cette thèse, nous allons analyser le jeu de HTTT sur des plateaux finis de forme normale (carrée) et des plateaux en forme de tore.

De façon générale, jouer à un jeu est une tâche cognitive qui a reçu une grande attention, tout particulièrement dans le domaine de l'intelligence artificielle (7). En effet, les jeux offrent un cadre structuré, doté de règles explicites, d'objectifs clairs et de résultats mesurables, ce qui en fait un environnement idéal pour expérimenter des algorithmes d'apprentissage automatique et de raisonnement. Dans ce contexte, un agent intelligent peut simuler des comportements proches de ceux observés chez les êtres humains, tels que la prise de décisions, la planification stratégique, l'adaptation à un environnement dynamique ou encore l'apprentissage par renforcement. Le jeu devient ainsi un laboratoire virtuel où l'on peut observer, tester et affiner les capacités cognitives d'un agent intelligent, tout en bénéficiant d'un contrôle précis sur les variables de l'environnement.

Parmi les différentes catégories de jeux utilisées en intelligence artificielle, les jeux combinatoires occupent une place de choix en raison de leur structure mathématique rigoureuse et de la richesse de leurs espaces de recherche (8) (9). Ces jeux, caractérisés par des règles déterministes, une information complète et l'absence de hasard, offrent un terrain idéal pour l'analyse algorithmique et la modélisation du raisonnement stratégique. Des exemples de tels jeux incluent le Gomoku (10), les échecs (11), le jeu de Hex (12), ainsi que les nombreuses variantes du Tic-Tac-Toe (13). Ce dernier, en particulier dans sa version développée par Frank Harary (1), a suscité un intérêt soutenu dans la littérature scientifique (6) (14) (15) en raison de sa simplicité apparente masquant une complexité combinatoire non triviale. Ces jeux servent non seulement de bancs d'essai pour les algorithmes de recherche, mais aussi de modèles abstraits pour explorer des concepts d'intérêt tels que la stratégie gagnante.

Le problème de la satisfaction d'une formule propositionnelle (SAT pour *satisfiability*) consiste à déterminer s'il y a pour une expression logique composée de variables booléennes et d'opérateurs logiques (tels que $\wedge$, $\vee$, $\neg$) une assignation des variables qui rend la formule vraie. Ce problème est central en informatique théorique, car il a été le premier à être démontré NP-complet (16). Cette classification signifie que SAT appartient à la classe des problèmes NP (*nondeterministic polynomial time*), c'est-à-dire des problèmes pour lesquels une solution peut être vérifiée rapidement (en temps polynomial), même si on ne sait pas toujours comment la trouver rapidement. Une autre façon de voir les problèmes NP-complet est d'utiliser l'analogie d'un puzzle qui peut être difficile à compléter, mais facile à vérifier une fois qu'on a la solution. Aussi, une autre propriété des problèmes NP-complet est que tous les problèmes NP peuvent s'y réduire. En revanche, la classe P regroupe les problèmes pour lesquels une solution peut être trouvée rapidement (en temps polynomial). La question ouverte P versus NP, autrement dit si tout problème dont la solution est vérifiable rapidement peut aussi être résolu rapidement, est l'un des plus grands défis non résolus en mathématiques et en informatique. Le problème SAT est donc au cœur de cette problématique, car une preuve que SAT peut être résolu en temps polynomial impliquerait que P = NP. D'un autre côté, en pratique, les formules booléennes et le problème SAT sont utilisés pour modéliser une grande variété de problèmes complexes, notamment en vérification de logiciels (17), en planification automatique (18), en conception de circuits (19) et plus généralement en intelligence artificielle puisqu'il s'agit d'un problème classique de résolution de contraintes.

Au cours des trente dernières années, les solveurs SAT ont connu une évolution spectaculaire, passant de méthodes de recherche naïves à des algorithmes hautement optimisés capables de résoudre des instances

industrielles complexes (20). Dans les années 1990, les solveurs reposaient principalement sur l'algorithme DPLL (21), un algorithme de recherche en profondeur, mais l'introduction de l'algorithme CDCL (22) au début des années 2000 a marqué un tournant majeur en permettant aux solveurs d'apprendre de leurs erreurs pour éviter les impasses. Cette avancée a été renforcée par l'intégration d'heuristiques sophistiquées, de littéraux surveillés, de techniques de redémarrage et de simplifications en prétraitement. Parallèlement, les compétitions internationales de solveurs SAT (23) ont favorisé l'innovation et la diffusion des meilleures pratiques. Aujourd'hui, grâce aux avancées des solveurs SAT, il est désormais possible de traiter efficacement des instances SAT de grande taille malgré la complexité théorique du problème (20). On peut même envisager de les utiliser pour s'attaquer à des problèmes combinatoires difficiles (24).

Le problème de la satisfiabilité d'une formule booléenne quantifiée (QBF pour *quantified Boolean formula*) est une extension du problème SAT dans laquelle les variables booléennes sont quantifiées de façon existentielle ($\exists$) et universelle ($\forall$). Le problème SAT correspond donc à QBF pour des formules contenant uniquement des variables existentielles. Une formule QBF permet ainsi d'exprimer des énoncés du type « il existe une affectation aux variables existentielles pour toutes les valeurs des variables universelles telles que la formule soit vraie ». Cette expressivité accrue permet de modéliser des problèmes plus complexes que ceux exprimables avec SAT. Contrairement à SAT, qui est NP-complet, QBF se situe dans une classe de complexité supérieure et est en fait PSPACE-complet (25). Cela signifie que tous les problèmes qui se résolvent à l'intérieur d'un espace mémoire polynomial peuvent s'y réduire. En pratique, les formules QBF sont utilisées pour modéliser des problèmes impliquant des interactions stratégiques (26), des jeux à information complète (27) et la vérification de systèmes (28), où la simple expressivité de SAT ne suffit plus.

De la même manière que les problèmes NP-complets peuvent être représentés en utilisant l'analogie des puzzles, les problèmes PSPACE-complets peuvent être vus en utilisant l'analogie de deux joueurs qui s'affrontent dans un jeu à information complète. C'est d'ailleurs une des raisons pour lesquelles plusieurs jeux combinatoires sont PSPACE-complets, tels que Gomoku (29), Amazons (30), Othello (31), Nim (32) et Hex (33).

Après les succès importants des solveurs SAT, il était tout naturel de développer des solveurs QBF dont l'utilité est de déterminer la satisfiabilité d'une formule QBF. Contrairement aux solveurs SAT, qui ne traitent que des variables existentielles, les solveurs QBF doivent gérer l'alternance des quantificateurs, ce qui rend ces solveurs un peu plus complexes. Vu la proximité des représentations SAT et QBF, il y a beaucoup de liens

en commun avec les méthodes développées pour les solveurs SAT. Nous allons aborder ces concepts plus en détail au chapitre 2.

Parallèlement aux avancées constantes dans le développement des solveurs QBF, la recherche sur des méthodes d'encodage QBF appliquées à des problèmes combinatoires joue un rôle tout aussi crucial. En effet, la manière dont un problème est formulé et traduit en une formule QBF a un impact direct sur la performance de la résolution. Un encodage efficace peut réduire considérablement la taille de la formule, limiter la profondeur de l'alternance des quantificateurs et mettre en évidence des structures logiques exploitables par les solveurs. À l'inverse, un encodage naïf ou mal adapté peut rendre la résolution inutilement complexe, voire intractable, même pour les solveurs les plus avancés.

L'amélioration de la qualité des encodages vise donc plusieurs objectifs : minimiser le nombre de variables et de clauses, préserver les propriétés structurelles du problème d'origine et faciliter l'application de techniques de simplification ou de prétraitement. Ainsi, l'efficacité globale d'une approche QBF repose sur une synergie entre la puissance du solveur et la qualité de l'encodage. Cette complémentarité justifie l'intérêt croissant pour les travaux qui explorent de nouveaux schémas d'encodage, adaptés à des classes particulières de problèmes, ou qui exploitent des connaissances *a priori* sur la structure du problème à résoudre. Les recherches suivantes (14) (15) sur la résolution de jeux combinatoires ont parfaitement démontré qu'il était possible d'obtenir des gains de performance importants grâce au développement de nouveaux encodages.

Les liens étroits entre les jeux combinatoires et la classe des problèmes PSPACE-complets font qu'il est tout naturel d'explorer l'application des formules QBF aux jeux combinatoires, et plus particulièrement au jeu de Tic-Tac-Toe d'Harary. Ce jeu constitue un terrain d'expérimentation particulièrement riche pour la communauté QBF, notamment en raison de la diversité des instances qu'il permet de générer (6). En effet, plusieurs configurations de formes cibles peuvent être définies, ce qui introduit une grande variabilité dans les objectifs à atteindre. De plus, l'espace des états croît de manière exponentielle avec la taille du plateau, entraînant une explosion combinatoire qui met à l'épreuve les capacités des solveurs et l'efficacité des encodages. Enfin, le jeu peut être joué sur différents types de plateaux, dont les deux variantes spécifiques qui seront analysées dans cette thèse. Ces caractéristiques font du Tic-Tac-Toe d'Harary un excellent candidat pour étudier les limites et les performances des encodages et solveurs QBF dans différents contextes de complexité.

La motivation de combiner QBF et le jeu de Tic-Tac-Toe d'Harary est aussi de répondre au défi que pose la résolution de formules QBF lorsque le niveau de quantification augmente puisqu'un nombre important d'alternance de quantificateurs est une caractéristique spécifique des jeux combinatoires (34).

Dans cette thèse, nous présenterons nos contributions aux encodages QBF du Tic-Tac-Toe d'Harary, qui permettent une amélioration de la performance de résolution QBF. Nous comparerons également nos résultats aux meilleurs encodages QBF actuels. Nous présenterons aussi nos contributions au niveau de l'analyse de formes spécifiques utilisées dans le Tic-Tac-Toe d'Harary.

La présentation de cette thèse est structurée de la façon suivante. Le chapitre 1 abordera une brève revue de la littérature sur SAT et QBF, tandis que le chapitre 2 présentera une revue de la littérature sur le Tic-Tac-Toe d'Harary. Le chapitre 3 portera sur l'historique des encodages QBF pour le jeu du Tic-Tac-Toe d'Harary. Le chapitre 4 présentera l'encodage PAIRING, qui est le premier encodage que nous avons réalisé et qui a mené à notre premier article (35). Le chapitre 5 portera sur l'encodage COVER, qui est le deuxième encodage que nous avons réalisé et qui a mené à notre deuxième article (36). Le chapitre 6 présentera une amélioration à un encodage existant nommé COR+ pour mettre à profit la symétrie des plateaux en forme de tore. Le chapitre 7 portera finalement sur une analyse détaillée sur certaines formes spécifiques du Tic-Tac-Toe d'Harary, ce qui est aussi une contribution de cette thèse.

CHAPITRE 1

DE SAT VERS QBF

Dans ce chapitre, nous présentons certaines notions préliminaires à la bonne compréhension de cette thèse.

1.1 Le problème SAT

Le problème SAT est le problème de satisfiabilité d'une formule de logique propositionnelle. Il est très important en théorie car, comme déjà mentionné, c'est le problème emblématique de la classe NP-complet (16), et tout problème de la classe NP peut s'y réduire. Un solveur SAT, c'est-à-dire un logiciel permettant de résoudre SAT, peut donc être appliqué à n'importe quel problème de la classe NP. Il est aussi important en pratique, car la performance des solveurs SAT modernes permet maintenant de les utiliser dans plusieurs domaines (37). Étant donné que la question P versus NP n'est toujours pas résolue, il est utile de rappeler qu'il n'existe actuellement aucun algorithme capable de résoudre toutes les instances SAT en temps polynomial.

Voici des exemples de formules SAT où $\vee$ représente la disjonction ("ou" logique), $\wedge$ la conjonction ("et" logique), et $\neg$ la négation.

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \tag{1.1}$$

$$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2) \tag{1.2}$$

La forme utilisée pour les équations 1.1 et 1.2 est la forme normale conjonctive (CNF). C'est la forme généralement utilisée pour exprimer une formule SAT. Elle est décrite de la façon suivante : un littéral est une variable ou sa négation, une clause est une disjonction (ou) de littéraux, et une formule CNF est une conjonction (et) de clauses. Il faut noter que toute formule propositionnelle peut être transformée en une formule équivalente en CNF.

Le problème SAT consiste donc à trouver une assignation aux variables booléennes de la formule qui la rendra vraie. Dans le cas de la formule 1.1, assigner $vrai$ à toutes les variables est une possibilité. Il peut

néanmoins y avoir plusieurs possibilités ; par exemple, dans ce cas-ci, on peut aussi assigner $faux$ à toutes les variables pour satisfaire la formule. Dans le cas de la formule 1.2, il n'existe aucune solution. On peut vérifier cela en essayant toutes les assignations possibles aux variables et, chaque fois, la formule sera insatisfaite.

Comme mentionné dans l'introduction, le problème SAT, comme tous les problèmes NP-complets, est analogue à un puzzle. Il peut être très difficile à résoudre, mais lorsque le casse-tête est complété, il est facile d'en confirmer la solution. Le problème SAT a plusieurs extensions telles que SMT, QBF et DQBF (20). Dans cette thèse, nous nous intéresserons uniquement au problème QBF.

1.2 Le problème QBF

Comme mentionné précédemment, le problème QBF (*Quantified Boolean Formula*) consiste à déterminer si une formule de logique propositionnelle avec quantificateurs existentiels et universels est satisfiable. C'est le porte-étendard des problèmes PSPACE-complets (25), et tous les problèmes PSPACE, qui incluent NP, peuvent s'y réduire. Les solveurs QBF peuvent ainsi être appliqués à n'importe quel problème PSPACE. De même que pour le problème SAT, et aussi parce que le problème QBF inclut tous les problèmes SAT, il n'existe actuellement aucun algorithme capable de résoudre toutes les instances QBF en temps polynomial.

Voici des exemples de formules QBF où $\exists x$ représente 'il existe un x' et $\forall x$ 'pour tous les x' :

$$\exists x_1 \forall x_2 \exists x_3 [(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)] \tag{1.3}$$

$$\forall x_1 \forall x_2 (x_1 \vee x_2) \tag{1.4}$$

La forme utilisée pour les équations 1.3 et 1.4 est la forme normale prenex (PNF). C'est la forme normalement utilisée pour exprimer une formule QBF. Elle est constituée de deux parties : le prenex et la matrice. Le prenex est une série de variables quantifiées placée au début de la formule et la matrice est une formule en forme normale conjonctive (CNF) qui suit cette série de quantificateurs. Toute formule QBF est équivalente à une formule PNF.

Le problème QBF consiste à déterminer si une formule QBF est satisfiable. Pour qu'elle le soit, un peu comme SAT, il faut trouver une assignation aux variables qui rendra la formule vraie. Cependant, à la différence de SAT, il faut prendre en considération l'ordre des variables dans le prenex et s'assurer qu'il existe une assignation aux variables existentielles qui satisfait la formule pour chaque valeur possible des variables universelles. Par exemple, pour la formule 1.3, si on assigne la variable x_1 à $faux$ et x_3 à $vrai$, la matrice est satisfaite peu importe la valeur de x_2. Pour la formule 1.4, il n'existe aucune solution, car lorsque x_1 et x_2 sont assignés à $faux$, la matrice n'est pas satisfaite. Il faut noter qu'une formule n'ayant que des variables existentielles est équivalente à une instance du problème SAT.

Le problème QBF, comme tous les problèmes PSPACE-complets, est analogue à deux adversaires qui s'affrontent dans un jeu à information complète : l'existentiel contre l'universel. Dans ce contexte, le joueur existentiel choisit les valeurs des variables existentielles et essaie de satisfaire la formule, tandis que le joueur universel choisit les valeurs des variables universelles et essaie de falsifier la formule. De plus, les deux joueurs jouent selon l'ordre des quantificateurs dans le prenex puisqu'une formule QBF doit se résoudre de cette façon. Alors, lorsqu'une variable existentielle est rencontrée, le joueur existentiel assigne une valeur à une variable ; similairement pour le joueur universel lorsqu'une variable universelle est rencontrée. Une formule QBF satisfiable est une formule où le joueur existentiel a une stratégie qui lui permet de gagner, peu importe ce que le joueur universel peut jouer. Donc, résoudre une formule QBF revient à trouver une stratégie gagnante pour le joueur existentiel.

Prenons l'exemple du prenex suivant :

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 (...) \tag{1.5}$$

Intuitivement, résoudre la formule 1.5 avec l'analogie de deux joueurs qui s'affrontent revient à se demander : est-ce qu'il existe une valeur de x_1 qui contrecarrera les valeurs de x_2 pour ensuite avoir une valeur de x_3 qui contrecarrera les valeurs de x_4 ?

Il existe plusieurs algorithmes pour résoudre les problèmes SAT et QBF. Nous les introduirons dans la section suivante.

1.3 Les solveurs SAT et QBF

Afin de résoudre une formule SAT ou QBF, on utilise un solveur. Les solveurs SAT ou QBF implémentent plusieurs algorithmes combinés dans le but de résoudre une formule. Un solveur QBF, qui doit implicitement résoudre plusieurs formules SAT de manière itérative, devra d'ailleurs inclure un solveur SAT. Nous ferons maintenant un survol de l'historique des principaux algorithmes utilisés pour résoudre les formules SAT et QBF. Ces algorithmes emploient plusieurs principes mathématiques, qui seront décrits plus loin.

1.3.1 La résolution SAT

L'histoire de la résolution SAT a véritablement commencé avec l'algorithme de Davis-Putman (DP) (21), qui utilisait le principe de résolution (38) produisant, à partir de deux clauses contenant une même variable de polarité opposée, une troisième clause combinant les deux premières, sans la variable utilisée pour la résolution. Nous verrons la résolution au point 1.4.4.

L'algorithme DP applique la résolution sur toutes les paires de clauses et peut potentiellement produire une explosion combinatoire du nombre de clauses. Les mêmes auteurs ont donc proposé l'algorithme de recherche de Davis–Putnam–Logemann–Loveland (DPLL) (39), qui a corrigé ce problème. Cependant, il s'est avéré que, lorsqu'aucune solution n'était trouvée sur une branche de la recherche, le retour à la dernière assignation effectuée ne permettait pas nécessairement d'apprendre et ainsi d'améliorer la recherche.

La plupart des implémentations modernes utilisent donc l'algorithme CDCL (*Conflict driven clause learning*) (40), qui permet souvent de revenir à une assignation antérieure à la dernière effectuée. CDCL est toujours un algorithme de recherche exhaustive ; cependant, lorsqu'une branche ne permet pas de trouver une solution, une analyse de ce conflit permet d'apprendre une nouvelle clause et d'éliminer d'autres branches sans solution, et par conséquent d'accélérer l'exploration de l'espace des solutions.

D'autres techniques ont aussi été intégrées aux solveurs SAT, telles que le redémarrage et les littéraux surveillés (22), qui ont grandement contribué à améliorer leurs performances.

Aujourd'hui, CDCL (40) est l'algorithme dominant le monde SAT et il est utilisé dans la plupart des solveurs. La recherche dans ce domaine a atteint une certaine maturité et les innovations que l'on peut voir sont de petits ajustements ou des techniques très spécifiques, utiles pour résoudre certaines classes d'instances.

1.3.2 La résolution QBF

Contrairement à SAT, la recherche dans QBF est encore jeune. Les solveurs QBF ne se sont pas stabilisés autour d'un seul algorithme. Dans tous les cas, il y a un solveur SAT intégré dans un solveur QBF puisqu'il y aura une formule SAT à résoudre ou plusieurs de manière itérative.

La difficulté de la résolution QBF vient du fait qu'il faut valider la satisfaction de la formule par une assignation des variables existentielles, peu importe ce que l'on assigne aux variables universelles. Autrement dit, si on a u variables universelles, il y a donc 2^u assignations possibles, et l'on doit s'assurer que l'on peut satisfaire la formule avec chacune de ces assignations.

Il y a trois catégories d'algorithmes de résolution QBF.

1.3.3 QCDCL

Un des algorithmes utilisés dans les solveurs QBF est QCDCL (41)(42). En résumé, on essaie de répliquer le succès de CDCL à la saveur QBF. C'est toujours un algorithme de recherche exhaustive, mais comme il faut une solution pour toutes les valeurs des variables universelles, la recherche ne s'arrête pas à la première solution trouvée. La recherche se termine plutôt quand il y a une solution aux variables existentielles pour chaque valeur des variables universelles ou lorsqu'il existe une assignation aux variables universelles où il n'y a aucune solution pour les variables existentielles. La résolution est toujours utilisée lors des conflits, mais on parle maintenant de Q-résolution, une technique qu'on verra plus loin dans la thèse. L'apprentissage de clauses de conflits se fait aussi de la même façon que pour CDCL, mais on doit tenir compte de l'ordre des variables dans le prenex. Il s'y ajoute l'apprentissage de cubes, que l'on expliquera plus loin.

1.3.4 Expansion

Une autre méthode utilisée dans les solveurs consiste à transformer une formule QBF en une formule SAT (43). En fait, on traite les quantificateurs universels de la gauche vers la droite, en remplaçant $\forall x \varphi$ par $\varphi(0) \wedge \varphi(1)$. À la fin, on obtient une formule SAT que l'on peut résoudre à l'aide d'un solveur SAT. Cette méthode peut paraître simple, mais elle crée une explosion exponentielle du nombre de clauses dans la formule SAT obtenue.

1.3.5 CEGAR

La méthode CEGAR (*Counterexample-guided Abstraction Refinement*) est une méthode qui vient du monde de la modélisation formelle (44). Appliquée à QBF, elle utilise l'analogie du jeu à deux adversaires que constitue une formule QBF (45). Autrement dit, on modélise l'interaction comme un affrontement entre les variables existentielles et universelles.

En fait, on résout la formule avec une approche récursive sur les variables dans le même ordre que le prenex. À chaque niveau de récursion, si la variable courante est existentielle, on essaie de satisfaire la formule, et si elle est universelle, on essaie de falsifier la formule. Par exemple, si la variable courante est existentielle et que l'on réussit à satisfaire la formule, on a donc une stratégie gagnante pour le joueur existentiel. Le joueur existentiel tentera donc une valeur pour la variable existentielle et le joueur universel contre-attaquera en trouvant une assignation aux variables universelles qui falsifiera la formule en tenant compte de l'assignation des variables existentielles actuelles. S'il réussit, alors le joueur existentiel construira une abstraction qui consiste à accumuler les stratégies universelles et à trouver une nouvelle assignation qui satisfera à la fois l'abstraction et la formule pour tenter de trouver une valeur adéquate pour la variable existentielle. On boucle de cette façon, jusqu'à ce que l'un des deux joueurs trouve une assignation que l'adversaire ne peut satisfaire ou falsifier.

1.4 Principes utilisés dans les solveurs QBF

Dans cette section, nous aborderons certains principes mathématiques, concepts ou techniques qui sont utilisés dans la résolution QBF. La majorité est tirée des mêmes techniques utilisées dans la résolution SAT, mais avec quelques nuances. Il est bien entendu impossible de tous les présenter, mais nous mentionnerons les plus importants.

1.4.1 Clauses et cubes

Dans le monde SAT, on travaille généralement en CNF (*Conjunctive Normal Form*) où chaque clause est une disjonction d'un ou de plusieurs littéraux, et les littéraux sont des variables ou leurs négations. Les clauses sont également apprises au fil de la résolution et chaque clause représente le complément d'une assignation partielle qui entraîne un conflit dans la formule.

Voici un exemple de clauses en CNF :

$$(l_1 \vee l_2 \vee l_3) \wedge (l_1 \vee \neg l_2 \vee \neg l_3) \tag{1.6}$$

Dans le monde QBF, on utilise également les CNF et on fait l'apprentissage de clauses de la même manière. Cependant, on travaille aussi en DNF (*Disjunctive Normal Form*) où chaque cube est une conjonction de littéraux, et un DNF est une disjonction de cubes. Les cubes sont pour DNF ce que les clauses sont pour CNF.

Voici un exemple de cubes en DNF :

$$(l_1) \vee (l_2 \wedge \neg l_3) \vee (\neg l_2 \wedge l_3) \tag{1.7}$$

Cette dualité entre CNF et DNF est nécessaire puisque l'on a à la fois des quantificateurs existentiels et universels. Par exemple, en QCDCL, lorsque les formules tombent en conflit, on fait l'apprentissage de nouvelles clauses. Lorsque l'on satisfait une formule, on fait l'apprentissage d'un cube, et les cubes représentent des assignations partielles qui satisferont la formule (41).

1.4.2 La dépendance des variables

Dans le monde SAT, aucune variable n'est dépendante d'une autre. Elles sont toutes au même niveau et on peut les assigner dans n'importe quel ordre. Il n'en va pas de même pour QBF. De façon naïve, une variable donnée est dépendante de toutes celles qui la précèdent dans le prenex. Cependant, lorsqu'on analyse le prenex et la formule plus attentivement, on peut voir qu'une variable ne sera pas nécessairement dépendante de tout ce qui la précède, mais seulement d'un sous-ensemble de variables (46)(47).

$$\exists \boldsymbol{a}, \boldsymbol{b} \forall \boldsymbol{x}, \boldsymbol{y} \exists \boldsymbol{c}, \boldsymbol{d}\ (a \vee b) \wedge (a \vee x \vee c) \wedge (b \vee c) \wedge (b \vee y \vee d) \tag{1.8}$$

$$\exists \boldsymbol{b}\ (\exists \boldsymbol{a} \forall \boldsymbol{x} \exists \boldsymbol{c}\ (a \vee b) \wedge (a \vee x \vee c) \wedge (b \vee c) \wedge \forall \boldsymbol{y} \exists \boldsymbol{d}\ (b \vee y \vee d)) \tag{1.9}$$

Par exemple, les équations 1.8 et 1.9 sont deux formules équivalentes. Plus en détail, on peut observer dans l'équation 1.8 que la variable d est seulement présente dans la dernière clause et donc on peut « pousser »son quantificateur plus loin, comme réalisé dans la formule 1.9. Avec ce même raisonnement, on peut « pousser »les autres quantificateurs dans la formule et obtenir l'équation 1.9. Au final, on peut voir dans la formule 1.9 que les ensembles de variables a, x, c et y, d sont indépendants les uns des autres et les solveurs QBF peuvent utiliser cette connaissance afin d'améliorer leur performance. Par exemple, deux variables indépendantes l'une de l'autre peuvent être assignées dans un ordre différent du prenex. Aussi, cela permet d'améliorer les performances des réductions universelles et existentielles, que nous verrons au point suivant.

1.4.3 Réduction universelle et existentielle

La réduction universelle et la réduction existentielle n'ont pas d'équivalent dans le monde SAT. Ce sont deux techniques utilisées uniquement dans QBF. Il s'agit de méthodes introduites avec la Q-résolution (48) pour les clauses que nous verrons à la section suivante et aussi dans (41) pour les cubes.

$$\frac{C \cup \{l\}}{C} \tag{1.10}$$

(1) C est une clause, $Q(l) = \forall$, $l' < l$ pour tout $l' \in C$ avec $Q(l') = \exists$
(2) C est un cube, $Q(l) = \exists$, $l' < l$ pour tout $l' \in C$ avec $Q(l') = \forall$

La relation d'ordre $l' < l$ signifie que l' apparaît avant l dans le prenex de la formule.

En d'autres mots, le principe de réduction universelle (la règle 1.10 (1)) affirme que si une variable universelle dans une clause n'a aucune variable existentielle qui lui succède dans le prenex de cette même clause, alors on peut retirer cette variable de la clause. La réduction existentielle (la règle 1.10 (2)) est la règle duale appliquée aux cubes. Si une variable existentielle dans un cube n'a aucune variable universelle qui la suit dans le prenex de ce même cube, alors on peut retirer cette variable du cube.

$$\exists e_1 \forall u \exists e_2 \, (e_1 \vee u) \wedge (u \vee e_2) \iff (e_1) \wedge (u \vee e_2) \tag{1.11}$$

Si on regarde la partie gauche de l'équation 1.11, on constate qu'il est possible d'appliquer la réduction universelle sur la première clause, mais pas sur la deuxième. La raison est que la première clause n'a pas de variable existentielle plus éloignée dans le prenex que la variable universelle u, mais ce n'est pas le cas de la deuxième clause. Donc on peut retirer u de la première clause, mais on doit la laisser en place dans la deuxième clause.

Sans entrer dans tous les détails de la justification des règles 1.10, que l'on peut retrouver dans (48), l'idée principale peut être comprise à partir de l'exemple 1.11. En fait, dans ce cas, la variable u peut être éliminée de la première clause car, du prenex, on voit que le choix de e_1 doit être adéquat pour toute valeur de u, incluant u assigné à $faux$. On doit donc nécessairement voir que la clause contenant (e_1) soit satisfaite.

1.4.4 Résolution et Q-résolution

Incontestablement, les principes les plus importants dans la résolution de problèmes SAT et QBF sont la résolution et la Q-résolution puisqu'elles permettent de simplifier, d'apprendre et de résoudre les formules. Le principe de résolution 1.12 est un principe mathématique qui a été défini à l'origine dans (38). Il est le cœur de l'algorithme de Davis-Putnam qui utilise cette technique à profusion. Il est aussi utilisé dans l'analyse de conflit de CDCL (40).

$$\frac{(\alpha \vee l) \wedge (\beta \vee \neg l)}{(\alpha \vee \beta)} \tag{1.12}$$

Le principe de résolution se justifie aisément puisque, dans toutes les assignations possibles de la variable l, les littéraux l et $\neg l$ sont des valeurs contraires et l'un est nécessairement $vrai$ et l'autre $faux$. Donc, dans 1.12, si ($\alpha \vee l$) et ($\beta \vee \neg l$) sont satisfaites, si l est $faux$ alors α doit être $vrai$ et si $\neg l$ est $faux$ alors β doit être $vrai$. On voit donc, dans les deux cas, que ($\alpha \vee \beta$) est satisfaite.

QBF reprend ce principe, mais on parle plutôt de Q-résolution (48). La différence est que la Q-résolution ne se fait uniquement que sur des variables existentielles. Ensuite, on applique la réduction universelle. Par exemple, dans la formule 1.13, la résolution est d'abord appliquée sur l, avant de réduire sur u_1 et u_2.

$$\frac{\exists e_1 e_2 \forall u_1 u_2 \exists l \, (u_1 \vee e_1 \vee l) \wedge (u_2 \vee e_2 \vee \neg l)}{(u_1 \vee e_1 \vee u_2 \vee e_2)} \implies (e_1 \vee e_2) \tag{1.13}$$

Étant donné que la Q-résolution est si importante pour la performance des solveurs QBF, la recherche s'est aussi penchée sur l'amélioration des systèmes de preuve (*proof systems*). Par exemple, la Q-résolution s'applique aussi aux cubes, comme démontré par (41). Également, on a la QU-résolution (49) et la LD-résolution (42) qui appliquent toutes les deux la Q-résolution sur les variables universelles, mais de façon différente. Finalement, la LQU-résolution et la LQU+-résolution (50) utilisent la QU-résolution et la LD-résolution. Toutes ces techniques sont utilisées complètement ou en partie par les solveurs QBF.

Nous n'approfondirons pas davantage l'algorithmie des solveurs puisque la contribution de cette thèse se situe au niveau de l'introduction de nouveaux encodages QBF permettant une résolution plus efficace à l'aide de solveurs QBF modernes, et non dans le développement de nouvelles méthodes de résolution pour le problème QBF. Nous aborderons toutefois les solveurs et les préprocesseurs dans la section suivante puisqu'ils jouent un rôle important dans cette thèse.

1.5 Les solveurs et préprocesseurs

Dans cette section, nous allons aborder brièvement les solveurs et préprocesseurs qui ont été utilisés au cours de cette thèse. Nous ne détaillerons pas sur chacun d'eux puisque cela dépasse le cadre de cette thèse, mais allons plutôt référer le lecteur aux articles originaux.

1.5.1 Les solveurs

Les solveurs QBF sont des outils faits pour déterminer la satisfiabilité des formules. Les solveurs utilisés dans cette thèse sont DepQBF (51), CAQE (52), Qute (53) et QESTO (54). Ils ont été choisis parce qu'ils ont été les gagnants du QBFEval'19 [1], une compétition de solveurs QBF.

Les solveurs vont résoudre les formules et indiquer si les formules sont satisfiables ou non. De plus, certains solveurs sont capables d'extraire les stratégies sous forme de fonctions de Skolem ou de Herbrand, qui sont essentielles non seulement pour la certification du résultat du solveur, mais aussi pour l'utilisation de QBF

1. http ://www.qbflib.org/eval19.html

dans des applications pratiques.

DepQBF (51) est un solveur QBF qui fonctionne principalement avec QCDCL. Il inclut une analyse des dépendances des variables, l'expansion de variables, les réductions et résolutions expliquées dans les sections précédentes, incluant également des axiomes qui ne sont pas expliqués dans cette thèse. La résolution se fait telle une recherche dans un arbre de la même façon que CDCL, mais en incluant les variables universelles.

Le solveur QBF (52) Caqe est basé sur CEGAR, mais il utilise aussi l'expansion de variables. Il inclut un solveur SAT qu'il utilise pour résoudre une « partie »où les variables existentielles affrontent les variables universelles. La résolution se fait en itération où l'un des deux camps essaie de satisfaire ou de falsifier la formule, et, chaque fois, la partie adverse a droit à une « contre-attaque »et raffine le modèle, jusqu'au moment où l'un des deux camps obtient la victoire et donne la réponse.

Le solveur Qute est basé sur QCDCL. Il utilise l'apprentissage des dépendances entre les variables, une approche distinctive qu'il implémente de manière spécifique. La résolution se fait par une recherche dans un arbre QCDCL.

Le solveur QBF Qesto (54) utilise aussi une approche où les variables existentielles et universelles s'affrontent de manière similaire à CEGAR. Il instancie un solveur SAT pour chaque paire de niveaux de quantifications existentielles et universelles de la formule QBF. Ensuite, pour chaque instanciation de solveur SAT, une formule SAT est construite, qui représente un sous-ensemble de la formule QBF, en sélectionnant les clauses qui incluent les variables du niveau voulu. La résolution se fait en itérant les appels au solveur SAT.

1.5.2 Les préprocesseurs

Les préprocesseurs sont des outils intéressants pour améliorer la performance des solveurs QBF. Ceux utilisés au cours de cette thèse sont Bloqqer (55), HQsPré (56) et QRatpre+ (57). Le rôle des préprocesseurs est de simplifier la formule QBF dans le but de la rendre plus facile à résoudre par un solveur. Chacun des préprocesseurs est en réalité un amalgame d'algorithmes et de techniques mathématiques permettant de simplifier la formule QBF. Leur fonction première est de simplifier une formule QBF avant que le solveur puisse le résoudre. L'objectif est de réduire la complexité de la formule, notamment en diminuant le nombre de variables et de clauses, la structure du prenex, etc. Cette étape de prétraitement, bien que cela consomme

du temps, vise à rendre le temps total (prétraitement et résolution) inférieur à celui de la résolution directe de la formule originale. En identifiant et en éliminant les parties redondantes ou en rendant la structure plus explicite, les préprocesseurs sont capables de réduire significativement la taille des formules.

1.6 Les contraintes de cardinalité

Comme cette thèse se concentrera sur l'encodage explicite en QBF, nous nous devons de faire usage des meilleures méthodes de représentation de problèmes usuels sous la forme de clauses. Nous aborderons dans cette section l'encodage de contraintes de cardinalité, qui jouent un rôle très important dans les encodages SAT et QBF. Il en existe plusieurs types (58) (59) (60) et, bien sûr, nous utilisons certaines contraintes de cardinalité dans les encodages que nous allons expliquer dans les chapitres suivants. Nous allons donc présenter les contraintes de cardinalité les plus importantes à la compréhension des encodages présentés dans cette thèse.

1.6.1 Les types de contraintes de cardinalité

Les contraintes de cardinalité dans le monde SAT et QBF sont utilisées afin de limiter le nombre de variables dans un ensemble de variables pouvant avoir une valeur donnée. Il existe plusieurs types de contraintes de cardinalité, notamment *At-Least-One*, *At-Least-N*, *At-Most-One*, *At-Most-N*, *Exactly-One*, *Greater-Than* et *Lower-Than*.

Une contrainte *Exactly-One* sur un ensemble de variables V signifie que l'on a une seule et unique variable $v \in V$ de cet ensemble à $vrai$. Toutes les autres variables $v' \in V$ sont donc assignées à $faux$. En contrepartie, si on appliquait une contrainte *At-Least-N*, on aurait $n > 1$ variables $v \in V$ qui devraient minimalement être assignées à $vrai$. Toutes les autres $v' \in V$ sont alors assignées à $faux$.

1.6.2 La contrainte *At-Least-One* (ALO)

Dans le monde SAT et QBF, la contrainte ALO est on ne peut plus simple. Il suffit de mettre une clause contenant l'ensemble des variables concernées. L'équation 1.14 est un exemple. Pour satisfaire la formule qui contient cette clause, on doit minimalement avoir l'une de ces variables assignées à $vrai$.

$$v_1 \vee v_2 \vee v_3 \vee v_4 \tag{1.14}$$

Malgré le fait qu'il s'agisse de la contrainte la plus simple, elle est importante, comme nous le verrons au cours de cette thèse.

1.6.3 La contrainte *At-Most-One* (AMO) et l'encodage Binaire

La contrainte AMO est un peu plus compliquée que ALO, car elle indique que s'il y a une variable d'un ensemble de variables assignée à $vrai$, alors toutes les autres variables doivent être assignées à $faux$. Il existe plusieurs façons d'y arriver. Celle que nous utilisons dans l'encodage COVER présenté au chapitre 5 est l'encodage Binaire (60).

L'intuition derrière l'encodage binaire consiste à introduire un ensemble de variables B supplémentaires pour représenter un nombre en binaire. Pour un ensemble de variables $V = v_1, v_2, v_3, \ldots$ auxquelles on veut appliquer la contrainte AMO, il suffit d'avoir suffisamment de variables dans B pour représenter les indices des éléments de V en binaire. Pour chaque variable $v_i \in V$, il suffit d'avoir des contraintes qui impliquent que si v_i est assignée à $vrai$ alors les variables de B représentent le nombre i en binaire. Forcément, il n'y aura qu'une seule variable $v_i \in V$ qui pourra être assignée à $vrai$. Remarquez aussi qu'il est possible qu'aucune variable ne soit assignée à $vrai$.

Prenons l'exemple de $V = v_1, v_2, v_3, v_4$, où nous avons quatre variables et une contrainte AMO sur ces quatre variables avec un encodage Binaire. Pour représenter les nombres 1,2,3 et 4, il suffit donc d'introduire deux nouvelles variables $B = b_1, b_2$ ainsi que les contraintes 1.15 à 1.18.

Les variables b_1 et b_2 représentent un nombre de binaire. On peut voir que, quelle que soit l'assignation des variables, au plus une variable v_i pourra être vraie puisque les conséquences des contraintes 1.15 à 1.18 sont incompatibles.

$$v_1 \implies (b_1 \wedge b_2) \tag{1.15}$$

$$v_2 \implies (b_1 \wedge \neg b_2) \tag{1.16}$$

$$v_3 \implies (\neg b_1 \wedge b_2) \tag{1.17}$$

$$v_4 \implies (\neg b_1 \wedge \neg b_2) \tag{1.18}$$

1.6.4 La contrainte *At-Least-One* (ALO) et l'encodage Binaire universel

On peut aussi utiliser la représentation binaire dans le but de faire une contrainte ALO. Cette contrainte peut aussi être utilisée en QBF dans le but d'analyser toutes les possibilités d'un ensemble donné. Si nous reprenons l'exemple vu à la section 1.6.3 avec les mêmes variables $V = \{v_1, v_2, v_3, v_4\}$ et supposons que les variables $B = \{b_1, b_2\}$ sont universelles avec l'ordre de quantification suivante : $\forall b_1 b_2 \exists v_1 v_2 v_3 v_4$.

$$(b_1 \wedge b_2) \implies v_1 \tag{1.19}$$

$$(b_1 \wedge \neg b_2) \implies v_2 \tag{1.20}$$

$$(\neg b_1 \wedge b_2) \implies v_3 \tag{1.21}$$

$$(\neg b_1 \wedge \neg b_2) \implies v_4 \tag{1.22}$$

Une assignation satisfaisant les contraintes 1.19 à 1.22 va nécessairement satisfaire l'une des variables de V. De plus, dans le cadre QBF, si les variables de l'ensemble B sont universelles, alors toutes les combinaisons de celles-ci seront explorées et nous aurons une situation où chacune des variables de l'ensemble V sera assignée à $vrai$.

1.6.5 La contrainte *Exactly-One* (EO) et le *Ladder encoding*

La contrainte EO est similaire à la contrainte AMO dans le sens où une variable d'un ensemble de variables est assignée à $vrai$ et toutes les autres variables doivent être assignées à $faux$. La seule différence entre AMO et EO est qu'une variable doit absolument être assignée à $vrai$. Ils existent plusieurs façons de faire ce type de contrainte. Essentiellement, il s'agit de la combinaison des contraintes AMO et ALO. Pour l'encodage PAIRING que nous allons présenter au chapitre 4, nous avons utilisé le *Ladder encoding* (61) qui fait la

contrainte *Exactly-One*.

L'encodage contient une série de clauses à deux variables (équations 1.23 à 1.26) dans le but de faire une chaîne d'implications (équation 1.27) qui contient toutes les équations (équations 1.23 à 1.26) et constitue une forme « d'échelle ».

$$ladder_1 \implies ladder_2 \tag{1.23}$$

$$ladder_2 \implies ladder_3 \tag{1.24}$$

$$ladder_3 \implies ladder_4 \tag{1.25}$$

$$ladder_4 \implies end \tag{1.26}$$

$$ladder_1 \implies ladder_2 \implies ladder_3 \implies ladder_4 \implies end \tag{1.27}$$

Dans la contrainte 1.27, si l'une des variables $ladder_i$ est assignée à $vrai$, alors toutes les suivantes dans la chaîne sont aussi assignées à $vrai$. À l'inverse, si l'une des variables est assignée à $faux$, alors toutes les précédentes sont assignées à $faux$. Il n'y a donc qu'au plus une variable assignée à $faux$ suivie d'une variable assignée à $vrai$ dans la chaîne, et c'est ce principe qui est utilisé pour avoir la contrainte *Exactly-One*.

Par exemple, pour avoir une contrainte *Exactly-One* sur les variables $var_1, var_2, var_3, var_4$ et var_5 en utilisant le *Ladder encoding*, on ajoute les clauses 1.28 à 1.32 qui forcent une seule variable à $vrai$ en utilisant le principe mentionné plus haut, soit le passage de $faux$ à $vrai$ pour deux variables $ladder$ consécutives.

$$ladder_1 \iff var_1 \tag{1.28}$$

$$\neg ladder_1 \wedge ladder_2 \iff var_2 \tag{1.29}$$

$$\neg ladder_2 \wedge ladder_3 \iff var_3 \tag{1.30}$$

$$\neg ladder_3 \wedge ladder_4 \iff var_4 \tag{1.31}$$

$$\neg ladder_4 \wedge end \iff var_5 \tag{1.32}$$

La variable end est toujours assignée à $vrai$ pour que le *Ladder encoding* fonctionne et qu'il y ait au moins un passage de $faux$ à $vrai$, quitte à ce que ce soit de la dernière variable $ladder$ à end. Mais, en l'assignant à $faux$, elle force toutes les variables var_i à $faux$. On peut voir cette variable comme un interrupteur « arrêt »ou « marche »sur les variables var_i. Ce principe sera important lorsque nous allons présenter notre encodage PAIRING.

1.6.6 Conclusion

Dans ce chapitre, nous avons présenté le problème SAT qui est le problème de satisfiabilité d'une formule propositionnelle ainsi que le problème QBF qui est le problème de satisfiabilité d'une formule propositionnelle quantifiée. Nous avons vu que SAT est NP-complet, que QBF est PSPACE-complet et que les problèmes PSPACE-complets peuvent être vus comme deux adversaires qui s'affrontent dans un jeu à information complète.

Nous avons rapidement abordé quelques concepts utilisés dans les solveurs SAT et QBF tels que la résolution, la Q-résolution, les différences entre les clauses et les cubes, la dépendance des variables, les réductions universelles et existentielles ainsi que les différents types d'algorithmes utilisés dans les solveurs QBF comme QCDCL, Expansion et CEGAR.

Pour finir, nous avons abordé les types de contraintes de cardinalité et, surtout, les plus importantes pour comprendre cette thèse, qui sont les contraintes *At-Least-One* (ALO), *At-Most-One* (AMO) et *Exactly-One* (EO). Nous avons vu aussi les encodages Binaire et *Ladder* qui représentent des contraintes AMO et EO, respectivement.

CHAPITRE 2
QBF ET LE JEU DE TIC-TAC-TOE D'HARARY

Dans ce chapitre, nous décrirons la problématique entourant les *achievement games* et le jeu de Tic-Tac-Toe d'Harary.

2.1 Le Tic-Tac-Toe d'Harary

Frank Harary, célèbre théoricien des graphes, a introduit les concepts de *achievement and avoidance games* sur les graphes (2). Dans un *achievement game*, deux joueurs choisissent à tour de rôle des éléments (par exemple des sommets ou des arêtes) d'un graphe dans le but de former une configuration cible - comme un sous-graphe particulier ou une propriété donnée. Le premier joueur à atteindre cette configuration gagne la partie. À l'inverse, dans un *avoidance game*, les joueurs cherchent à éviter de compléter une telle configuration, et celui qui est contraint de le faire perd.

Au cours de ses travaux, Harary a inventé une généralisation du jeu du Tic-Tac-Toe (1). Dans ce jeu, deux joueurs, Noir et Blanc, placent à tour de rôle une pierre sur un plateau de $N \times N$ cellules. Le premier à compléter une forme prédéterminée à une rotation, réflexion et translation près gagne la partie. La forme (un *animal* ou un *polyomino*) peut être n'importe laquelle, du moment que c'est un ensemble de cellules connectées par les arêtes des cellules du plateau, et elle est déterminée avant que la partie commence. On peut aussi choisir une forme constituée d'un ensemble de cellules quelconques, mais dans ce cas on parle de *créature* (4) et non plus d'*animal*.

Plusieurs travaux de recherche (1) (62) (63) (64) (65) ont été faits sur ce jeu dans le but de savoir pour quels animaux le premier joueur (Noir) a une stratégie gagnante, c'est-à-dire avec quelle forme il est assuré de gagner la partie s'il ne commet aucune faute. Un animal est dit gagnant s'il existe une stratégie gagnante pour le premier joueur (Noir). Si le deuxième joueur (Blanc) peut forcer une nulle, alors l'animal est dit perdant. En fait, comme nous allons bientôt le voir, il n'y a jamais de stratégie gagnante permettant à Blanc de compléter la forme. Donc, tout au plus, Blanc peut avoir une stratégie lui permettant d'empêcher Noir de compléter la forme. Il s'agit donc d'une stratégie de blocage pour Blanc.

Jusqu'à présent, tous les animaux ont été déterminés gagnants ou perdants sur un plateau normal infini à

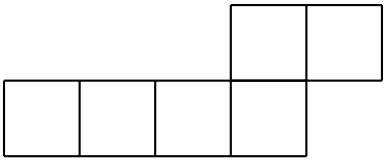


Figure 2.1 – Polyomino Snaky

l'exception d'un seul, le Snaky (figure 2.1). Dans ce cas, la question de l'existence d'une stratégie gagnante pour le premier joueur est toujours ouverte.

Sur les plateaux finis, la taille et le type de plateau sont importants, car ils influencent les stratégies gagnantes. Une stratégie gagnante sur un plateau fini d'une taille $N \times N$ peut être différente d'une stratégie gagnante sur un plateau $(N+1) \times (N+1)$. Bien que les formes aient été déterminées gagnantes ou perdantes sur des plateaux normaux, les tailles de plateaux pour lesquels il y a une stratégie gagnante n'ont pas toutes été découvertes. Aussi, aucune analyse exhaustive identifiant les formes et les tailles de plateaux en forme de tore pour lesquels Noir a une stratégie gagnante n'a jamais été faite. Les plateaux toriques sont les plateaux où les bordures sont connectées. Nous allons aborder ce sujet plus en profondeur au chapitre 7.

Avec les années, nos connaissances concernant le Snaky se sont approfondies. Nous savons qu'il est perdant lorsque la taille du plateau est de 8×8 ou moins (63). Nous savons aussi que, si le premier joueur se limite à jouer seulement sur les cellules adjacentes aux cellules déjà jouées, il est perdant (64). par contre, nous savons qu'il est un gagnant avec un handicap, c'est-à-dire qu'il est gagnant si le premier joueur peut jouer un coup supplémentaire au premier tour (62). Nous savons également que si le deuxième joueur utilise un *Beck-strategy*, une méthode statistique qui le restreint à jouer sur les cellules qui enlèvent le plus de formes possibles au premier joueur, alors le premier joueur est gagnant (63). Il a également été démontré que le Snaky est un *paving winner*, c'est-à-dire que le deuxième joueur ne peut faire une stratégie de pavage, une notion que nous aborderons à la sous-section 2.1.1, pour contrecarrer le premier joueur (65).

Il faut noter que, pour les *achievement games*, le deuxième joueur ne peut jamais avoir une stratégie gagnante lui permettant de compléter la forme. Cela a été démontré par John Nash et son *Strategy-stealing argument* (13). Le Tic-Tac-Toe d'Harary, où chaque joueur joue le même nombre de pierres à chaque tour, en est un bon exemple. Le *Strategy-stealing argument* suppose que si le deuxième joueur a une stratégie gagnante, alors le premier joueur peut se placer dans la position du deuxième joueur, jouer son premier

tour en fonction d'un tour précédent fictif et continuer la stratégie gagnante par la suite. Ainsi, il *vole* la stratégie gagnante du deuxième joueur. Étant donné qu'il gagnerait, il y aurait contradiction avec ce que l'on avait supposé à propos du deuxième joueur.

Donc, si le premier joueur n'a pas de stratégie gagnante, alors le deuxième joueur a une stratégie de blocage qui lui permet de contrecarrer son adversaire peu importe ce qu'il joue et de s'assurer d'un match nul. Pour la situation inverse, si le premier joueur a une stratégie gagnante, alors en l'appliquant, il est assuré de gagner, et le deuxième joueur n'a pas de stratégie de blocage. Par conséquent, l'existence d'une stratégie gagnante pour le premier joueur est équivalente à l'inexistence d'une stratégie de blocage pour le deuxième joueur.

2.1.1 Stratégies de pavage

Une stratégie de pavage est une stratégie de blocage particulière pour le deuxième joueur. Il suffit de partitionner le plateau en paires de cellules en s'assurant que toutes les formes cibles possibles sur le plateau contiennent au moins une de ces paires. La stratégie de blocage pour le deuxième joueur est alors simplement de jouer sur la deuxième cellule de la paire contenant le coup précédent de Noir. De cette façon, si toutes les formes contiennent une paire, cela garantit que le premier joueur ne pourra jamais compléter une forme et par conséquent qu'il ne peut être gagnant. Plusieurs travaux de recherche font mention de cette méthode (65)(4).

Prenons un exemple simple avec le polyomino Fatty, représenté à la figure 2.2, que l'on essaie de compléter sur un plateau de 5×5. À la figure 2.3, on a un pavage par des dominos, qui sont un type particulier de paires. D'ailleurs, la stratégie de blocage fonctionnerait tout aussi bien avec des paires qu'avec des dominos, ce qui nous sera utile au chapitre 4. On peut voir qu'il n'y a aucun endroit sur le plateau où l'on pourrait compléter la forme sans qu'elle contienne au moins une des paires de ce pavage. La stratégie de blocage pour Blanc est alors la suivante : si Noir joue sur une cellule d'un domino, alors Blanc joue sur l'autre cellule du même domino. On en conclut que Fatty n'est pas une forme gagnante sur un plateau de 5×5. Cette stratégie de pavage se fait au tour zéro (avant le début de la partie) et on peut donc déterminer le sort de Fatty avant même que la partie commence.

Bien que l'existence d'une stratégie de pavage par paires au tour zéro soit suffisante pour déterminer le sort de la partie, son inexistence laisse encore ouverte la possibilité qu'il puisse y avoir une stratégie de blocage

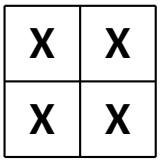


Figure 2.2 – Polyomino Fatty

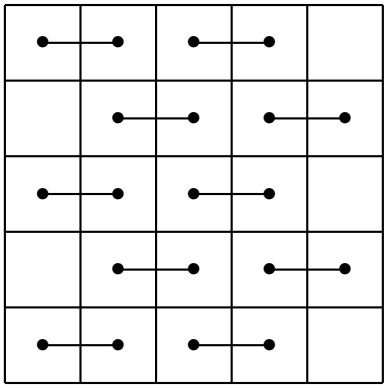


Figure 2.3 – Pavage pour Fatty

plus complexe pour le deuxième joueur. De manière similaire à (63), il est possible de chercher une stratégie de pavage par paires lorsque la partie est en cours. Dans ce cas, les formes cibles possibles du plateau qui contiennent une cellule blanche sont éliminées et on partitionne le reste du plateau en paires de cellules distinctes de sorte que toutes les formes cibles restantes contiennent une paire. Si on réussit, cela veut dire que le deuxième joueur a une stratégie de blocage à ce moment précis de la partie.

Prenons l'exemple du Snaky pour expliquer ce cas, représenté à la figure 2.4. Il est impossible d'établir une stratégie de pavage au tour zéro pour le Snaky, et il y a d'ailleurs un article complet sur le sujet (65). Cependant, il est possible d'avoir une stratégie de pavage en cours de partie à des moments bien choisis. Notre exemple de la figure 2.5 vient de (63) et nous montre une stratégie de pavage pour le Snaky. Lorsque Noir joue au premier tour sur **une** des multiples cellules représentées par **X**, alors Blanc joue sur la cellule **O**, et ensuite on obtient une stratégie de pavage. Dans cette stratégie, on peut aussi voir qu'il y a des paires de cellules disjointes représentées par les paires de lettres **a** à **f**. Cette stratégie de pavage n'est pas une preuve complète que le Snaky est perdant sur les plateaux de 8×8, mais démontre tout de même que le Snaky est perdant après que l'on a joué ces deux premiers coups. De façon générale, si l'on parvenait à établir une stratégie de pavage pour chaque choix initial de Noir, cela constituerait une stratégie de blocage complète pour le Snaky.

2.1.2 Bris de symétrie

La symétrie est très importante dans le jeu de Tic-Tac-Toe (13) et elle peut être vue de plusieurs angles différents. Dans le cas de nos encodages, nous utilisons deux angles qui concernent le premier tour uniquement. Nous reviendrons d'ailleurs sur ce sujet au chapitre 6 pour introduire un nouveau principe de

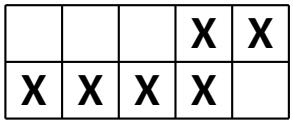


Figure 2.4 – Polyomino Snaky

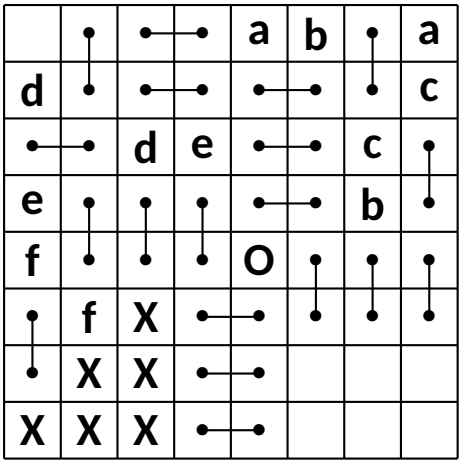


Figure 2.5 – Pavage pour Snaky

symétrie applicable aux plateaux en forme de tore.

Au premier tour, seulement un quart du plateau est important, comme vu à la figure 2.6 où seule la partition inférieure gauche est importante. La raison est que si Noir jouait sur une des autres cellules du plateau, nous pourrions effectuer la rotation du plateau entier pour que la cellule jouée soit une de la partition inférieure gauche.

Dans le même ordre d'idées, au premier tour, seulement une moitié du plateau séparé par une diagonale est importante, comme présenté à la figure 2.7. La raison est que si Noir jouait sur une cellule de la deuxième moitié du plateau, nous pourrions effectuer la réflexion du plateau pour obtenir une cellule de la première moitié du plateau.

Ces deux principes de symétrie sont combinés pour permettre à Noir de jouer seulement sur un huitième du plateau au premier tour, tel qu'il est démontré à la figure 2.8.

Pour les plateaux en forme de tore, la symétrie est encore plus simple : n'importe quelle cellule sur le plateau est équivalente à toutes les autres au premier tour. La raison est que les formes cibles possibles sur le plateau en forme de tore ne sont plus limitées par les bordures. Chaque cellule peut être utilisée pour compléter un nombre équivalent de formes cibles, contrairement à un plateau normal où il y a une différence entre les cellules en bordure et les cellules au centre. Puisque toutes les cellules sont équivalentes sur un plateau en forme de tore, on peut restreindre Noir à jouer sur une seule.

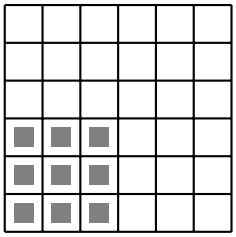


Figure 2.6 – Symétrie par rotation

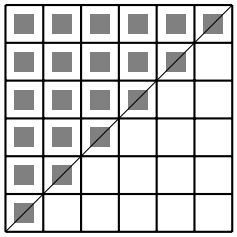


Figure 2.7 – Symétrie par réflexion

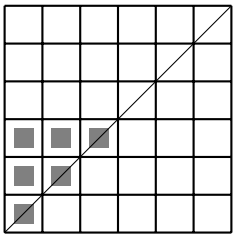


Figure 2.8 – Symétrie par rotation et réflexion

Comme la symétrie permet de réduire le nombre de possibilités pour le premier coup de Noir, cela réduit l'espace de recherche et en pratique le temps de calcul, comme le démontre l'expérimentation avec les solveurs QBF.

2.2 Conclusion

Dans ce chapitre, nous avons présenté le Tic-Tac-Toe d'Harary et les multiples recherches faites sur ce jeu. Nous avons aussi présenté le Snaky dont le statut de gagnant ou de perdant est, encore aujourd'hui, une question ouverte.

Nous avons exploré les stratégies de pavage, qui offrent au deuxième joueur une possibilité de mettre en œuvre une stratégie de blocage, et nous avons parlé du bris de symétrie, qui est utilisé pour filtrer l'espace de recherche du jeu au premier tour.

CHAPITRE 3
ENCODAGES DE JEU EN QBF

Dans ce chapitre, nous retracerons l'évolution des encodages QBF pour le jeu de Tic-Tac-Toe d'Harary par une revue approfondie de la littérature, mettant en lumière les contributions majeures qui ont façonné cet axe de recherche.

Nous poursuivrons en expliquant l'encodage de COR (*Corrective Encoding*) (14), une avancée significative tant sur le plan conceptuel que sur celui de la performance en réduisant considérablement le nombre de variables et de clauses par rapport à DYS (6). Nos travaux sur l'encodage PAIRING expliqué au chapitre 4 s'appuient sur l'encodage de COR.

Enfin, nous aborderons l'encodage de COR+ (*Polished Encoding*) (15), une version épurée et optimisée de COR. Cette simplification stratégique, tout en conservant la rigueur du modèle initial, a conduit à une amélioration substantielle des performances, démontrant l'intérêt d'une approche plus concise dans le contexte de la résolution QBF. Nos travaux sur l'encodage COVER expliqué au chapitre 5 s'appuient sur cet encodage.

Bien que les encodages du Tic-Tac-Toe d'Harary aient commencé avec DYS, nous nous abstiendrons de présenter cet encodage pour ne pas alourdir inutilement cette thèse. Nous allons quand même nous y référer afin de mettre en lumière certains aspects des encodages présentés. Nos travaux s'appuient sur COR et COR+.

3.1 Historique des encodages de jeu en QBF

Comme nous l'avons vu à la section 1.2, QBF a une sémantique en termes de jeux à information complète opposant deux joueurs, il est donc naturel d'encoder ce type de jeu en QBF. Toby Walsh, un chercheur bien connu de la communauté SAT et QBF, a lancé plusieurs défis à la communauté QBF lors de la conférence SAT 2003. Un de ces défis était justement de réaliser le jeu de Connect4 en QBF. Plus précisément, encoder un jeu à information complète opposant deux joueurs en QBF consiste à déterminer l'existence d'une stratégie gagnante pour l'un des deux joueurs. Ce défi a été relevé par (27) dans un article introduisant les bases de l'encodage de jeux en QBF. Il s'agit d'utiliser les niveaux de quantificateurs dans le prenex de la formule pour déterminer les tours des joueurs. On a dans le prenex un bloc (suite) de variables existentielles qui

décrivent le premier tour du premier joueur, puis un bloc de variables universelles qui décrivent le premier tour du deuxième joueur, etc. Ces auteurs ont aussi introduit les variables *gameover* et *cheat* permettant de déterminer quand une partie est terminée ou quand un joueur enfreint une règle du jeu.

L'idée de base des variables *gameover* est d'avoir une variable par tour pour déterminer si le jeu est terminé ou non à ce tour. Ensuite, on ajoute la variable *gameover* d'un tour donné à toutes les clauses qui décrivent le jeu à ce même tour. Aussi, lorsque la variable *gameover* d'un tour donné est assignée à $vrai$, toutes les variables *gameover* des tours qui suivent seront assignées à $vrai$. De cette façon, lorsque la partie est terminée et que la variable *gameover* d'un tour donné est assignée à $vrai$, toutes les clauses de ce tour et celles restantes de la formule sont automatiquement satisfaites. Il y a une variable *gameover* à l'intérieur de presque toutes les clauses.

Le principe des variables *cheat* est un peu différent. Chaque variable *cheat* est associée à une situation précise dans le jeu où un joueur donné a la possibilité de violer une règle du jeu. Pour avoir une stratégie gagnante valide, il faut que le premier joueur puisse gagner sans tricher. Alors l'encodage de (27) fait en sorte de falsifier la formule si le premier joueur triche. Cependant, pour le deuxième joueur, c'est une tout autre histoire, car lorsqu'il joue contre les règles du jeu, on ne doit pas falsifier la formule, mais plutôt la satisfaire puisque le premier joueur gagne dès que le deuxième viole les règles du jeu. C'est à quoi servent les variables *cheat*.

Par exemple, pour le cas du Tic-Tac-Toe d'Harary, on aura une variable *cheat* correspondant à la situation où Noir joue simultanément sur les cellules 1 et 2 au premier tour, alors que les règles du jeu stipulent qu'il ne doit jouer que sur une seule cellule à chaque tour. On introduira une autre variable pour les cellules 1 et 3, et ainsi de suite. Il y a donc une variable *cheat* pour chaque situation illégale du jeu, à chaque tour et pour les deux joueurs. Cela va de soi qu'il y a beaucoup de variables *cheat* dans la formule. En pratique, l'encodage présenté dans cet article (27) permet de déterminer qu'il n'y a pas de stratégie gagnante pour le premier joueur dans le jeu de Connect4 sur un plateau de 4×4 en 3 secondes et sur un plateau de 5×5 en 11 minutes.

Cet article a été suivi de (66) qui analyse un peu plus en détail les variables *cheat* renommées *indicator*. Les auteurs de (66) ont mis en lumière le fait que le nombre de variables *indicator* constitue une lourdeur pour les solveurs puisqu'il augmente considérablement l'espace de recherche. Ils ont pour cette raison proposé

une amélioration, soit regrouper les variables *indicator* en ajoutant des équivalences logiques sur des variables *indicator* de groupe, et ils ont testé cette méthode sur un jeu de *Evader/Pursuer*. Ils ont démontré qu'il y a une amélioration des performances en regroupant ces variables.

Plusieurs années plus tard, (6) a introduit DYS, le premier encodage QBF pour le Tic-Tac-Toe d'Harary. Cet encodage est basé sur celui de (27) pour le jeu de Connect4, tout en intégrant les améliorations proposées par (66). Comme pour (27), cet encodage utilise un nombre considérable de variables *gameover* et *cheat*. Les auteurs de l'article montrent que leur encodage est capable de résoudre les plateaux de 3×3 en quelques secondes, tandis que les plateaux de 4×4 nécessitent plus d'une journée de traitement. Bien que faisant avancer les connaissances scientifiques sur l'applicabilité pratique de QBF pour résoudre des jeux combinatoires, ces résultats restent toutefois limités. En fait, HTTT sur un plateau de 4×4 peut être résolu manuellement en énumérant les possibilités de jeu. L'application de QBF à des plateaux de tailles plus ambitieuses nécessite par conséquent des méthodes plus sophistiquées.

Une avancée majeure est l'introduction, en 2020, du *corrective encoding* (COR) (14). Il s'agit d'un encodage générique pour tous les *achievement* ou *positional games*. L'une des innovations proposées par les auteurs de cet article consiste à figer l'état du plateau à la fin de la partie et à déterminer le vainqueur uniquement lors du dernier tour, plutôt que de procéder à une validation à chaque tour comme le font les autres encodages. De cette façon, ils ont pu se débarrasser des variables *gameover* et ainsi avoir des clauses plus courtes. Ils se sont aussi débarrassés des variables *cheat* avec le *ladder encoding*, que nous avons vu à la section 1.6.5, alors que (66) croyait que ces variables étaient incontournables. Il s'agit d'une avancée majeure puisque les performances de COR ont considérablement surclassées celles de DYS avec plus d'une seconde pour les plateaux de 3×3 et environ cinq heures pour les plateaux de 4×4.

À la suite de l'introduction de COR, nous avons publié, conjointement avec mon directeur de thèse, l'article (35) et l'avons présenté à la conférence KI2021[1]. Cet article introduit l'encodage PAIRING qui innove par le fait qu'il résout le problème avec les perspectives du deuxième joueur. Nos performances surclassent celles de COR sur les plateaux de 4×4 avec une résolution en moins de deux minutes. Aussi, nous avons établi une méthode ingénieuse qui utilise les encodages COR et PAIRING de façon complémentaire. Nous allons aborder PAIRING ainsi que nos résultats au chapitre 4.

1. 44th German Conference on Artificial Intelligence - https ://ki2021.uni-luebeck.de/

Après l'arrivée de PAIRING, COR+ (15) introduit une amélioration significative de l'encodage de COR toujours avec la perspective du premier joueur. Cet encodage remplace en fait le *ladder encoding* par le *binary encoding* pour les contraintes AMO vues à la section 1.6.3 et réduit énormément le nombre de variables et de clauses. COR+ est beaucoup plus rapide que COR et résout les plateaux de 4×4 en moins d'une heure.

En 2023, (67) propose des améliorations dans la façon dont les états du jeu et les conditions gagnantes sont représentés sous forme de variables. Les auteurs de cet article ont testé ces améliorations sur des jeux positionnels tels que le jeu de Hex avec des plateaux de 19×19. Ils ont testé la variation de *maker-breaker* de HTTT et ils ont résolu le polyomino Z à une profondeur de 11 tours en huit minutes. Cette version de HTTT est toutefois différente de celle des autres travaux, incluant les nôtres, puisqu'on n'y tient pas compte du fait que le deuxième joueur puisse réaliser la forme visée. Dans cette version du jeu, le premier joueur gagne dès qu'il complète la forme visée, et ce indépendamment du fait que le deuxième joueur ait pu compléter la forme avant lui. Dans la version usuelle du jeu, le premier joueur ayant complété la forme gagne, ce qui est en général le cas dans les *achievement games*.

Finalement, nous avons publié l'article (36), toujours conjointement avec mon directeur de thèse, et l'avons présenté à la conférence Canadian AI 2025 [2]. Cet article introduit l'encodage COVER qui innove par le fait qu'on se restreint à une stratégie spécifique pour le premier joueur au lieu de chercher une stratégie quelconque, comme les autres encodages. Cet encodage est plus performant que COR+ dans plus de 61 % des cas. Nous allons aborder cet encodage ainsi que nos résultats au chapitre 5.

3.2 L'encodage de COR

Dans cette section, nous présenterons l'encodage de COR (14) en détail. La présentation ne suit pas totalement celle de (14), car nous l'avons légèrement simplifiée pour rendre l'encodage plus facile à suivre. Nous commencerons par présenter sommairement les variables et leurs utilités ainsi que la quantification (le prenex) utilisée dans l'encodage. Par la suite, nous expliquerons les clauses et terminerons par une analyse critique pour mettre en lumière les améliorations qui ont été faites par rapport à DYS.

2. https ://www.caiac.ca/en/conferences/canadianai-2025/home

3.2.1 Les variables

Les variables représentant le jeu, ainsi que leur signification, sont définies comme suit :

$$time_t \text{; la partie est en cours au tour } t \tag{3.1}$$

$$black_{t,x,y} \text{; il y a une pierre noire sur la cellule } (x,y) \text{ au tour } t \tag{3.2}$$

$$white_{t,x,y} \text{; il y a une pierre blanche sur la cellule } (x,y) \text{ au tour } t \tag{3.3}$$

$$occupied_{t,x,y} \text{; il y a une pierre noire ou blanche sur la cellule } (x,y) \text{ au tour } t \tag{3.4}$$

$$move_{t,x,y} \text{; Noir ou Blanc joue sur la cellule } (x,y) \text{ au tour } t \tag{3.5}$$

$$moveL_{t,j} \text{; } j\text{-ième chiffre de l'encodage binaire du choix de coup de Blanc au tour } t \tag{3.6}$$

$$ladder_{t,m} \text{; } \textit{ladder encoding} \text{ pour que les joueurs jouent un seul coup par tour} \tag{3.7}$$

$$win_e \text{; Noir a complété la forme cible } e \in E_B \tag{3.8}$$

3.2.2 Les quantifications

Les blocs de quantification apparaissent dans l'ordre des tours comme suit.

$$\exists time_t \tag{3.9}$$

$$\forall moveL_{t,j} \text{; au tour de Blanc seulement} \tag{3.10}$$

$$\exists move_{t,x,y} \tag{3.11}$$

$$\exists black_{t,x,y} \tag{3.12}$$

$$\exists white_{t,x,y} \tag{3.13}$$

$$\exists occupied_{t,x,y} \tag{3.14}$$

$$\exists ladder_{t,m} \tag{3.15}$$

et au dernier tour

$$\exists win_e \tag{3.16}$$

Le bloc de quantification du tour 1 signifie qu'il existe une valeur $time$ qui indique si la partie est toujours en cours, un choix d'un mouvement pour Noir, des statuts pour chaque cellule (indiquant si elle est occupée,

et, si oui, par une pierre de quel joueur), ainsi que des valeurs pour réaliser le *ladder encoding*, comme nous le verrons plus loin. Cela décrit totalement l'état du jeu au premier tour.

De son côté, le bloc de quantification du tour 2 signifie qu'il existe une valeur pour $time$ et que, pour tout choix d'un mouvement pour Blanc, il y a des statuts pour chaque cellule, ainsi que des valeurs pour réaliser le *ladder encoding*. Cela décrit cette fois l'état du jeu au deuxième tour et, de plus, le choix de Blanc est quantifié universellement car une stratégie gagnante pour Noir signifie que le choix de Noir est valide pour tous les choix possibles de Blanc.

Par la suite, les blocs de quantification alternent toujours entre un mouvement de Noir pour les tours impairs et un mouvement de Blanc pour les tours pairs. Finalement, le dernier bloc de quantification servira à affirmer l'existence de valeurs pour les variables win qui assureront que Noir a complété la forme et gagné la partie. Comme les choix de Noir sont quantifiés existentiellement et ceux de Blanc universellement, la formule au complet signifie l'existence d'une stratégie gagnante pour Noir.

3.2.3 Les clauses

Les clauses qui vont suivre expliquent la logique et la mécanique du jeu. Nous allons précéder chaque clause de sa signification pour en simplifier la lecture. De plus, bien qu'une clause soit une disjonction de littéraux comme présenté à la section 1.1, nous allons nous permettre d'utiliser aussi des implications de la forme $\neg x \wedge \neg y \implies z$ équivalentes à $(x \vee y \vee z)$. La lecture et la compréhension des clauses seront plus faciles. Nous utiliserons aussi des double-implications de la forme $(x \wedge y) \iff z$ qui sont équivalentes à $(\neg x \vee \neg y \vee z) \wedge (x \vee \neg z) \wedge (y \vee \neg z)$.

De façon générale, les indices des variables ont la signification suivante : t représente un tour, x, y représente une position sur le plateau et e représente une forme gagnante possible sur le plateau.

Si la partie est toujours en cours au tour t, alors elle l'était aussi au tour $t-1$.

$$time_t \implies time_{t-1} \tag{3.17}$$

Il n'y a pas de pierre sur le plateau au tour $t = 0$.

$$\neg black_{0,x,y} \wedge \neg white_{0,x,y} \wedge \neg occupied_{0,x,y} \tag{3.18}$$

Les deux joueurs ne peuvent pas jouer sur la même cellule.

$$\neg black_{t,x,y} \vee \neg white_{t,x,y} \tag{3.19}$$

Si une pierre était sur une cellule au tour précédent, alors elle reste là au tour courant.

$$black_{t-1,x,y} \implies black_{t,x,y} \tag{3.20}$$

$$white_{t-1,x,y} \implies white_{t,x,y} \tag{3.21}$$

Lorsque la partie est terminée, toutes les cellules inoccupées restent inoccupées.

$$(\neg time_t \wedge \neg black_{t-1,x,y}) \implies \neg black_{t,x,y} \tag{3.22}$$

$$(\neg time_t \wedge \neg white_{t-1,x,y}) \implies \neg white_{t,x,y} \tag{3.23}$$

Si une case est jouée, alors elle est occupée.

$$black_{t,x,y} \implies occupied_{t,x,y} \tag{3.24}$$

$$white_{t,x,y} \implies occupied_{t,x,y} \tag{3.25}$$

Si une case est occupée, alors elle est noire ou blanche.

$$occupied_{t,x,y} \implies (black_{t,x,y} \vee white_{t,x,y}) \tag{3.26}$$

Lorsque la partie est terminée, plus aucun autre coup n'est permis.

$$\neg time_t \implies \neg move_{t,x,y} \tag{3.27}$$

Un coup n'est pas permis si la case est déjà occupée.

$$occupied_{t-1,x,y} \implies \neg move_{t,x,y} \tag{3.28}$$

Jouer un coup place une pierre sur une cellule. Ici, Noir joue lorsque le tour t est impair et Blanc joue lorsque le tour t est pair. Dans les deux cas, on doit avoir $t > 0$ puisqu'au tour 0 personne ne joue, seul l'état du

plateau y est initialisé.

$$move_{t,x,y} \implies black_{t,x,y} \text{ pour } t \text{ impair} \tag{3.29}$$

$$move_{t,x,y} \implies white_{t,x,y} \text{ pour } t \text{ pair} \tag{3.30}$$

Si la partie n'est pas terminée, que la cellule n'était pas occupée au tour d'avant et que la combinaison des variables $moveL$ représente la position (x, y) en binaire, alors Blanc joue sur la cellule (x, y). Les variables $moveL$ font une contrainte ALO avec encodage binaire, tel que nous l'avons démontré à la section 1.6.4. Les positions (x, y) sont représentées en binaire d'une façon arbitraire, par exemple en utilisant l'ordre dans lequel on les rencontre dans un parcours. Les $L_0(v)$ et $L_1(v)$ représentent tous les 0 et 1 respectivement du chiffre binaire v représentant la position x, y.

$$\left(time(t) \wedge \neg occupied(t-1,x,y) \bigwedge_{j \in L_1(v)} moveL(t,j) \bigwedge_{j \in L_0(v)} \neg moveL(t,j) \right) \implies move_{t,x,y} \tag{3.31}$$

Si un joueur ne joue pas sur une cellule et que cette cellule n'était pas occupée au tour d'avant, elle reste inoccupée. Cela s'applique aux tours t impairs pour Noir et aux tours t pairs pour Blanc.

$$(\neg move_{t,x,y} \wedge \neg black_{t-1,x,y}) \implies \neg black_{t,x,y} \text{ pour } t \text{ impair} \tag{3.32}$$

$$(\neg move_{t,x,y} \wedge \neg white_{t-1,x,y}) \implies \neg white_{t,x,y} \text{ pour } t \text{ pair} \tag{3.33}$$

Les cellules inoccupées d'un joueur restent inoccupées lorsque ce n'est pas son tour à jouer. Cela s'applique aux tours t pairs pour Noir et aux tours t impairs pour Blanc.

$$\neg black_{t-1,x,y} \implies \neg black_{t,x,y} \text{ pour } t \text{ pair} \tag{3.34}$$

$$\neg white_{t-1,x,y} \implies \neg white_{t,x,y} \text{ pour } t \text{ impair} \tag{3.35}$$

Il doit y avoir au moins une configuration gagnante. Il s'agit ici d'une contrainte ALO, telle que nous l'avons vue à la section 1.6.2. Ici, E représente l'ensemble de toutes les formes gagnantes possibles sur le plateau et l'indice $e \in E$ représente une forme gagnante possible.

$$\bigvee_{e \in E} win_e \tag{3.36}$$

Une configuration gagnante est réussie lorsque toutes les cellules d'une forme cible sont noires. Pour chaque forme cible du plateaux il faut introduire les clauses suivantes où $(x_1, y_2), ...$ sont les coordonnées de chaque cellule de la forme cible $e \in E$ sur le plateau. Dans ce cas-ci, t est égal au dernier tour.

$$win_e \iff (black_{t,x_1,y_1} \wedge black_{t,x_2,y_2} \wedge black_{t,x_3,y_3} \wedge ...) \tag{3.37}$$

Blanc ne doit pas réussir une forme cible. Pour chaque forme cible formée des cellules, il faut introduire une clause similaire aux clauses 3.37. Donc, pour chaque forme cible du plateau, il faut introduire la clause suivante où $(x_1, y_1), (x_2, y_2), ...$ sont les coordonnées de chaque cellule de la forme cible $e \in E$ sur le plateau. Dans ce cas-ci aussi, t est égal au dernier tour.

$$(\neg white_{t,x_1,y_1} \vee \neg white_{t,x_2,y_2} \vee \neg white_{t,x_3,y_3} \vee ...) \tag{3.38}$$

Les prochaines clauses sont pour le *ladder encoding*, tel que nous l'avons vu à la section 1.6.5.

Le premier échelon de l'échelle :

$$ladder_{t,1} \iff move_{t,1,1} \tag{3.39}$$

Les échelons entre le début et la fin de l'échelle :

$$ladder_{t,i} \implies ladder_{t,i+1} \tag{3.40}$$

L'échelon de la fin. Si la partie est terminée, toutes les variables $move$ sont assignées à $faux$ par propagation.

$$ladder_{t,N} \implies time_t \tag{3.41}$$

À tous les échelons, on a une variable $ladder$ assignée à $vrai$ et une autre assignée à $faux$ qui assigne une variable $move$ à $vrai$.

$$(\neg ladder_{t,i-1} \wedge ladder_{t,i}) \iff move_{t,x,y} \tag{3.42}$$

Si la dernière variable $ladder$ est assignée à $faux$ et que la partie est en cours, c'est la dernière position du plateau.

$$(\neg ladder_{t,i} \wedge time_t) \iff move_{t,H,W} \tag{3.43}$$

3.2.4 Analyse et critique

L'encodage de COR (14) est très performant comparativement à DYS. Nous allons élaborer sur les améliorations qui nous semblent les plus importantes introduites par cet encodage.

Premièrement, le nombre de variables universelles a été grandement diminué. L'encodage de COR utilise la contrainte ALO avec l'encodage binaire universel, vu à la section 1.6.4, tandis que DYS utilise une variable universelle pour chaque position sur le plateau. Donc, pour un plateau de 4×4, nous avons 16 variables universelles par tour pour DYS, tandis que COR en aura seulement quatre. Comme nous explorerons toutes les combinaisons des assignations des variables universelles, cela va s'en dire que l'espace de recherche est beaucoup plus grand à explorer avec DYS qu'avec COR, ce qui pourrait expliquer pourquoi DYS nécessite davantage de temps de traitement.

Deuxièmement, la vérification des configurations gagnantes se fait au dernier tour plutôt qu'à chaque tour avec DYS. Avec COR, le choix des joueurs est propagé jusqu'au dernier tour puis la vérification des configurations gagnantes est faite. Cela réduit énormément le nombre de clauses de la formule.

Troisièmement, la majorité des clauses sont plus petites. Dans DYS, chaque clause contient une variable $gameover$ qui satisfait toutes les clauses lorsque la partie est terminée. La variable $gameover$ a été remplacée par $time$ dans COR, mais elle est limitée à un nombre réduit de clauses. Ainsi, les clauses sont en moyenne beaucoup plus petites dans COR que dans DYS.

Finalement, les variables $cheat$ ont toutes été enlevées dans COR, tandis que DYS les utilise abondamment. Ces variables déterminent les situations illégales du jeu pour les deux joueurs. Pour que les joueurs jouent selon les règles, COR utilise des contraintes de cardinalité et s'assure que les deux joueurs jouent un seul coup par tour.

3.3 Encodage de COR+

Dans cette section, nous allons présenter l'encodage de COR+ (15). De la même manière que COR, nous allons présenter une version différente de celle de (15) pour la rendre plus facile à suivre. L'encodage de COR+ est très similaire à celui de COR et, même si certains aspects sont identiques, nous préférons dupliquer l'information pour rendre la lecture plus facile. Nous commencerons par présenter sommairement les variables et leur utilité ainsi que la quantification utilisée dans l'encodage. Par la suite, nous expliquerons les clauses puis partagerons une analyse critique pour mettre en lumière les améliorations qui ont été faites par rapport à COR.

3.3.1 Les variables

Les variables représentant le jeu, ainsi que leur signification, sont définies comme suit :

$time_t$; la partie est en cours au tour t (3.44)

$black_{t,x,y}$; il y a une pierre noire sur la cellule (x, y) au tour t (3.45)

$white_{t,x,y}$; il y a une pierre blanche sur la cellule (x, y) au tour t (3.46)

$moveB_{t,j}$; j-ième chiffre de l'encodage binaire du choix de coup de Noir au tour t (3.47)

$moveW_{t,j}$; j-ième chiffre de l'encodage binaire du choix de coup de Blanc au tour t (3.48)

win_e; Noir a complété la forme cible $e \in E_B$ (3.49)

3.3.2 Les quantifications

Les blocs de quantification apparaissent dans l'ordre des tours comme suit :

$$\exists time_t \tag{3.50}$$

$\forall moveW_{t,j}$; au tour de Blanc seulement (3.51)

$$\exists moveB_{t,j} \tag{3.52}$$

$$\exists black_{t,x,y} \tag{3.53}$$

$$\exists white_{t,x,y} \tag{3.54}$$

et au dernier tour :

$$\exists win_e \tag{3.55}$$

3.3.3 Les clauses

Les clauses qui vont suivre expliquent la logique et la mécanique du jeu. À moins d'indication contraire, toutes les clauses suivantes sont les mêmes que celles de COR. Il faut toujours garder à l'esprit que COR+ est une version épurée de COR et donc son innovation est dans les clauses qui ont été supprimées, c'est-à-dire les clauses de 3.25 à 3.28 ainsi que les clauses du *ladder encoding* de 3.39 à 3.43. Nous aborderons plus en détail ce sujet dans la section 3.3.4.

Si la partie est toujours en cours au tour t, alors elle l'était déjà au tour $t-1$.

$$time_t \implies time_{t-1} \tag{3.56}$$

Il n'y a pas de pierre sur le plateau au tour $t = 0$. La différence avec COR est l'absence de la variable $occupied$ qui n'existe plus.

$$\neg black_{0,x,y} \wedge \neg white_{0,x,y} \tag{3.57}$$

Les deux joueurs ne peuvent pas jouer sur la même cellule.

$$\neg black_{t,x,y} \vee \neg white_{t,x,y} \tag{3.58}$$

Si une pierre était sur une cellule au tour précédent, alors elle reste là au tour courant.

$$black_{t-1,x,y} \implies black_{t,x,y} \tag{3.59}$$

$$white_{t-1,x,y} \implies white_{t,x,y} \tag{3.60}$$

Si un joueur ne joue pas sur une cellule et que cette cellule n'était pas occupée au tour d'avant, elle reste inoccupée. Cela s'applique aux tours t impairs pour Noir et aux tours t pairs pour Blanc.

$$\neg black_{t-1,x,y} \implies \neg black_{t,x,y} \tag{3.61}$$

$$\neg white_{t-1,x,y} \implies \neg white_{t,x,y} \tag{3.62}$$

Lorsque la partie est terminée, aucune nouvelle pierre n'apparaît sur le plateau.

$$(\neg time_t \wedge \neg black_{t-1,x,y}) \implies \neg black_{t,x,y} \tag{3.63}$$

$$(\neg time_t \wedge \neg white_{t-1,x,y}) \implies \neg white_{t,x,y} \tag{3.64}$$

Si la partie n'est pas terminée, que la cellule n'était pas occupée au tour d'avant et que la combinaison des variables $moveW$ est la bonne, alors la cellule x, y est blanche. Les variables $moveW$ font une contrainte ALO avec l'encodage Binaire universel, démontré à la section 1.6.4. Cette contrainte est différente de COR de deux façons : la variable $occupied$ a été remplacée par la variable $black$ afin de valider si Blanc peut jouer sur une cellule donnée et elle implique une variable $white$ au lieu d'une variable $move$.

$$(time_t \wedge \neg black_{t-1,x,y} \wedge \bigwedge_{j;[x,y](j)=0} moveW_{t,j} \quad \bigwedge_{j;[x,y](j)=1} moveW_{t,j}) \implies white_{t,x,y} \tag{3.65}$$

Si la cellule n'était pas noire au tour d'avant et qu'elle l'est au tour courant, cela implique la bonne combinaison des variables $MoveB$. Cela est une contrainte AMO avec encodage Binaire, expliqué à la section 1.6.3. Cette contrainte remplace le besoin du *ladder encoding* dans COR.

$$(\neg black_{t-1,x,y} \wedge black_{t,x,y}) \implies moveB_{t,j} \qquad ; [x,y](j) = 1 \tag{3.66}$$

$$(\neg black_{t-1,x,y} \wedge black_{t,x,y}) \implies \neg moveB_{t,j} \qquad ; [x,y](j) = 0 \tag{3.67}$$

Il doit y avoir au moins une configuration gagnante. Il s'agit ici d'une contrainte ALO, telle que nous l'avons vue à la section 1.6.2.

$$\bigvee_{e \in E_B} win_e \tag{3.68}$$

Une configuration gagnante est réussie lorsque toutes les cellules d'une forme cible sont noires.

$$win_e \iff (black_{t,x_1,y_1} \wedge black_{t,x_2,y_2} \wedge black_{t,x_3,y_3} \wedge ...) \tag{3.69}$$

Blanc ne doit pas réussir une forme cible.

$$(\neg white_{t,x_1,y_1} \vee \neg white_{t,x_2,y_2} \vee \neg white_{t,x_3,y_3} \vee ...) \tag{3.70}$$

3.3.4 Analyse et critique

L'encodage de COR+ (15) est très performant comparativement à COR. Nous allons élaborer sur les améliorations qui nous semblent les plus importantes et qui proviennent de cet encodage.

Premièrement, le *ladder encoding* a été enlevé. Il a été remplacé par la contrainte AMO, vue à la section 1.6.3, pour les coups de Noir. Pour Blanc, il y avait déjà la contrainte ALO avec l'encodage Binaire universel (section 1.6.4) et c'est suffisant.

Avec COR+, nous avons une contrainte AMO pour les coups de Noir et une contrainte ALO pour les coups de Blanc, au lieu d'une contrainte EO pour les coups des deux joueurs avec COR. L'idée derrière ce changement est subtile. Les solveurs font une recherche dans le but de satisfaire la formule. Dans ce sens, on doit s'assurer que Noir ne peut pas jouer deux fois ou plus dans un même tour, mais on n'a pas besoin de valider s'il n'a pas joué du tout, puisque ce n'est pas à son avantage de ne pas jouer. Dans le même ordre d'idées, on doit s'assurer que Blanc joue au moins une fois par tour, mais on n'a pas besoin de valider si Blanc joue plusieurs fois par tour, car ce n'est pas à l'avantage de Noir que Blanc joue plusieurs fois dans un même tour.

On remarque ici qu'il y a une nuance subtile entre les joueurs du jeu, Noir et Blanc, et ceux de la sémantique de la formule QBF, soient les joueurs existentiels et universels. On associe naturellement Noir au joueur existentiel puisqu'il doit trouver des coups formant une stratégie gagnante et Blanc au joueur universel puisque les variables universelles représentent les choix de Blanc. Quel sens faut-il alors donner aux choix des variables existentielles autre que celui des mouvements de Noir ? En fait, ces variables assurent la « gestion »du jeu, par exemple le positionnement des pierres en fonction du choix des joueurs. On considère donc qu'en plus de décider de ses propres choix Noir assure aussi cette gestion.

Donc, pour reprendre l'exemple précédent, Noir, comme joueur existentiel, pourrait théoriquement positionner plusieurs pierres blanches pendant le tour de Blanc, mais cela n'est pas dans son intérêt s'il veut satisfaire la formule qui affirme l'existence d'une stratégie gagnante pour lui. Néanmoins, il doit satisfaire les clauses, incluant celles qui imposent de déposer une pierre blanche sur la case choisie par le coup de Blanc. Le joueur existentiel doit réaliser cette opération, mais rien de plus. Il n'est plus nécessaire d'avoir des contraintes qui limitent l'apparition de pierres blanches sur le plateau. Cela est une propriété importante de cet encodage qui permet de réduire le nombre de contraintes et, par conséquent, de clauses.

Deuxièmement, les variables $move$ ont été retirées. Ainsi, le coup de Noir ou de Blanc n'assigne plus une variable $move$ qui propage ensuite une variable $black$ ou $white$, mais on assigne directement une variable $black$ ou $white$ sans utiliser une variable $move$. Il ne faut pas confondre les variables $move$ de l'encodage COR avec les variables $moveB$ et $moveW$ de l'encodage COR+, qui sont utilisées dans les encodages binaires présentés aux sections 1.6.3 et 1.6.4.

Finalement, la variable $occupied$ a aussi été retirée. Cela a réduit le nombre de clauses de la formule. Au lieu de valider si une cellule est occupée avec la variable $occupied$, on le valide avec les variables $black$ et $white$ directement.

3.4 Conclusion

Dans ce chapitre, nous avons fait une revue de l'historique des encodages QBF pour le jeu de Tic-Tac-Toe d'Harary et des contributions majeures qui ont fait progresser la connaissance dans ce domaine. Nous avons aussi présenté en profondeur les encodages de COR (14) et COR+ (15) ainsi que leurs innovations respectives par rapport aux encodages antérieurs.

CHAPITRE 4
L'ENCODAGE PAIRING

4.1 Introduction

Dans ce chapitre, nous présenterons l'encodage PAIRING qui encode le problème du jeu de HTTT avec la perspective du deuxième joueur (Blanc). Il s'agit d'établir une stratégie de blocage pour Blanc visant à empêcher Noir de gagner. Les instances QBF de ce chapitre sont satisfiables (SAT) si, et seulement si, l'instance correspondante dans l'un des encodages avec la perspective du premier joueur, soit DYS, COR et COR+, est insatisfiable (UNSAT).

4.1.1 Les stratégies de pavage

Un pavage du plateau est le partitionnement du plateau en paires de cellules distinctes, de telle sorte que toutes les formes cibles possibles sur le plateau contiennent au moins une de ces paires. Cela conduit à une stratégie dite de blocage par pavage de paires. Une forme cible possible peut contenir plusieurs paires et une paire peut être contenue dans plusieurs formes cibles possibles. Comme nous l'avons vu à la section 2.1.1, une stratégie de blocage par pavage pour le second joueur est de jouer sur la deuxième cellule de la paire contenant le coup précédent de Noir.

Notre encodage PAIRING généralise ce principe et représente une partie du jeu où, à tout moment, le deuxième joueur peut arrêter la partie et chercher un pavage par paires des cellules restantes. Formellement, l'encodage par paires représente un jeu de longueur k, où la condition gagnante vérifie l'existence d'un pavage des cellules non occupées par des paires de cellules, de telle sorte que toutes les formes cibles, à n'importe quelle position, seraient annulées, c'est-à-dire qu'elles contiennent soit une pierre du deuxième joueur, soit une paire du pavage.

Lorsque k est le nombre de tours maximum du jeu, c'est-à-dire $k = N \times N$ pour un plateau de $N \times N$, notre condition gagnante se réduit à vérifier que toutes les formes cibles possibles du plateau ne peuvent être complétées par le premier joueur et qu'il ne peut par conséquent pas gagner la partie. Ainsi, avec le nombre de tours maximum du jeu k, nos instances seront UNSAT si et seulement si le premier joueur a une stratégie gagnante. Cela résout donc le statut du jeu, mais de manière complémentaire aux encodages

habituels.

4.1.2 L'encodage PAIRING

Au niveau technique, l'encodage PAIRING suit une démarche similaire au *Corrective Encoding* (COR) de (14) mais avec une inversion des rôles. Dans notre cas, Noir est le joueur universel et Blanc est le joueur existentiel. De plus, contrairement à COR, nous encodons une stratégie de blocage et la condition gagnante est l'existence d'une stratégie de blocage par pavage.

4.1.2.1 Les variables

L'encodage est paramétré par la hauteur H et la largeur W du plateau, et par le nombre de tours k. Les tours sont numérotés $0, 1, \ldots, k$. Le plateau est initialisé au tour 0 pour ne contenir aucune pierre et le choix des cellules par les joueurs se produit aux tours $1, \ldots, k$. Ainsi, Noir joue aux tours $1, 3, \ldots,$ et Blanc joue aux tours $2, 4, \ldots$

Il y a $W \times H$ cellules sur le plateau, et donc $n = W \times H(W \times H - 1)/2$ paires distinctes de cellules. Nous indexons les paires de cellules par $id = 1, \ldots, n$. De même, selon la forme gagnante choisie pour une partie, il peut y avoir jusqu'à huit réflexions et rotations distinctes de cette forme. Nous indexons aussi les formes cibles. Le nombre exact de ces formes cibles variera, mais dans tous les cas nous les indexerons par $i = 1, 2, \ldots$

Les variables représentant le jeu, ainsi que leur signification, sont définies comme suit :

$time_t$; la partie est en cours au tour t (4.1)

$moveL_{t,j}$; j-ième chiffre de l'encodage binaire du choix de coup de Noir au tour t (4.2)

$move_{t,x,y}$; une pierre est posée sur la case (x, y) au tour t (4.3)

$ladder_{t,m}$; *ladder encoding* pour les coups de Noir (expliqué ci-dessous) (4.4)

$black_{t,x,y}$; il y a une pierre noire sur la case (x, y) au tour t (4.5)

$white_{t,x,y}$; il y a une pierre blanche sur la case (x, y) au tour t (4.6)

$occupied_{t,x,y}$; la case (x, y) est occupée au tour t (4.7)

$pair_{id}$; la paire id est dans le pavage (4.8)

$canceled_{i,x,y}$; la forme i à la position (x, y) est annulée (4.9)

4.1.2.2 Les quantificateurs

Les blocs de quantification apparaissent dans l'ordre des tours comme suit :

Au tour $t = 0$ (4.10)

$\exists time_0$ (4.11)

Ensuite on boucle $t = 1, \ldots, k$ (4.12)

$\exists time_t$ (4.13)

$\forall moveL_{t,j}$; au tour de Noir seulement (4.14)

$\exists move_{t,x,y}$ (4.15)

$\exists ladder_{t,m}$ (4.16)

$\exists black_{t,x,y}$ (4.17)

$\exists white_{t,x,y}$ (4.18)

$\exists occupied_{t,x,y}$ (4.19)

Et finalement, comme dernier bloc de quantification : (4.20)

$\exists pair_{id}$ (4.21)

$\exists canceled_{i,x,y}$ (4.22)

4.1.2.3 Les clauses

Les contraintes (clauses) QBF ainsi que leurs significations sont les suivantes, où $t = 1, \ldots, k, x = 1, \ldots, W$, et $y = 1, \ldots, H$.

Si la partie est toujours en cours au tour t, alors elle l'était aussi au tour $t - 1$.

$$time_t \implies time_{t-1} \tag{4.23}$$

Il n'y a pas de pierre sur le plateau au tour $t = 0$.

$$\neg black_{0,x,y} \wedge \neg white_{0,x,y} \wedge \neg occupied_{0,x,y} \tag{4.24}$$

Les deux joueurs ne peuvent pas jouer sur la même cellule.

$$\neg black_{t,x,y} \vee \neg white_{t,x,y} \tag{4.25}$$

Si une pierre était sur une cellule au tour précédent, alors elle reste là au tour courant.

$$black_{t-1,x,y} \implies black_{t,x,y} \tag{4.26}$$

$$white_{t-1,x,y} \implies white_{t,x,y} \tag{4.27}$$

Lorsque la partie est terminée, aucune nouvelle pierre n'apparaît sur le plateau.

$$(\neg time_t \wedge \neg black_{t-1,x,y}) \implies \neg black_{t,x,y} \tag{4.28}$$

$$(\neg time_t \wedge \neg white_{t-1,x,y}) \implies \neg white_{t,x,y} \tag{4.29}$$

Si une case est jouée, alors elle est occupée.

$$black_{t,x,y} \implies occupied_{t,x,y} \tag{4.30}$$

$$white_{t,x,y} \implies occupied_{t,x,y} \tag{4.31}$$

Si une case est occupée, alors elle est noire ou blanche.

$$occupied_{t,x,y} \implies (black_{t,x,y} \vee white_{t,x,y}) \tag{4.32}$$

Lorsque le jeu est terminé, aucun autre coup n'est permis.

$$\neg time_t \implies \neg move_{t,x,y} \tag{4.33}$$

Le coup n'est pas permis si la case est déjà occupée.

$$occupied_{t-1,x,y} \implies \neg move_{t,x,y} \tag{4.34}$$

Jouer un coup place une pierre sur une cellule. Ici, Noir joue lorsque le tour t est impair et Blanc joue lorsque le tour t est pair. Dans les deux cas, on doit avoir ($t > 0$).

$$move_{t,x,y} \implies black_{t,x,y} \tag{4.35}$$

$$move_{t,x,y} \implies white_{t,x,y} \tag{4.36}$$

Le joueur universel, Noir, choisit une cellule, spécifiée par l'assignation des bits $moveL_{t,j}$, et les variables $move_{t,x,y}$ sont configurées en conséquence. En d'autres mots, l'ensemble des variables $moveL_{t,j}$ d'un tour t constitue un nombre binaire et ce nombre implique une position de Noir. L'équation 4.37 en est un exemple vulgarisé.

$$(ensemble\ des\ moveL_t) \implies move_{t,x,y} \tag{4.37}$$

Plus précisément, chaque choix de cellule (x, y) est encodé par une chaîne de bits $[x, y]$ de longueur $\lceil \log_2(WH) \rceil$. Nous désignerons par $[(x, y)]_0$ l'ensemble des i pour lesquels le i-ème bit de cette chaîne est 0 et par $[(x, y)]_1$ l'ensemble des i pour lesquels le i-ème bit de cette chaîne est 1.

Nous avons donc, pour les tours de Noir, $t = 1, 3, \ldots$, seulement si la partie n'est pas terminée et si la cellule (x, y) n'est pas occupée au tour précédent, un choix de (x, y) (par les variables $moveL$) réalise un mouvement sur (x, y) (variables $move$).

$$time_t \wedge \neg occupied_{t-1,x,y} \bigwedge_{j \in [x,y]_0} \neg moveL_{t,j} \bigwedge_{j \in [x,y]_1} moveL_{t,j} \implies move_{t,x,y} \tag{4.38}$$

Comme nous l'avons vu à la section 2.1.2, sur un plateau carré, on peut briser la symétrie. Il suffit de considérer le cas où le premier joueur (Noir) joue son premier coup dans le triangle inférieur gauche du plateau. Lors de la génération des clauses (équation 4.38), pour le premier tour uniquement, seuls les coups possibles dans ce triangle sont considérés.

L'équation 4.39 décrit que lorsque Noir ne choisit pas de jouer sur une cellule donnée et que cette cellule n'était pas occupée par Noir au tour précédent, alors cette cellule reste inoccupée par Noir au tour courant. Cette équation s'applique seulement aux tours impairs. L'équation 4.40 décrit la même chose pour Blanc et s'applique seulement aux tours pairs.

$$(\neg move_{t,x,y} \wedge \neg black_{t-1,x,y}) \implies \neg black_{t,x,y} \tag{4.39}$$

$$(\neg move_{t,x,y} \wedge \neg white_{t-1,x,y}) \implies \neg white_{t,x,y} \tag{4.40}$$

Blanc ne peut pas jouer aux tours de Noir et vice-versa. Alors, on a la clause 4.41 qui décrit que les cellules inoccupées par Noir au tour précédent restent inoccupées par Noir au tour courant et s'applique seulement aux tours pairs. On a la même chose pour Blanc aux tours impairs avec la clause 4.42.

$$\neg black_{t-1,x,y} \implies \neg black_{t,x,y} \tag{4.41}$$

$$\neg white_{t-1,x,y} \implies \neg white_{t,x,y} \tag{4.42}$$

Afin de garantir qu'à chaque tour où la partie est en cours un seul et unique coup est effectué sur le plateau, on utilise le *ladder encoding* (61), expliqué à la section 1.6.5. Il y a donc dans les clauses 4.43 et 4.44 les

mêmes variables $ladder_{t,m}$ que l'on a vues précédemment, une pour chaque coup possible sur le plateau x, y, et chaque variable $ladder_{t,m}$ implique la suivante. La dernière variable de l'échelle est la variable $time_t$, qui est similaire à la variable end vue à la section 1.6.5, et elle indique si la partie est en cours.

$$ladder_{t,m} \implies ladder_{t,m+1} \tag{4.43}$$

$$ladder_{t,last} \implies time_t \tag{4.44}$$

Comme nous l'avons vu à la section 1.6.5, il ne peut y avoir qu'un seul endroit dans la chaîne d'implication des variables $ladder_i$ où une variable assignée à $faux$ est suivie d'une autre variable assignée à $vrai$ (sauf la première de l'échelle), alors chaque « échelon »de l'échelle est équivalent à une variable $move_{t,x,y}$ jusqu'à la variable $time_t$ qui doit être $vraie$ si la partie est en cours. Ainsi, si la partie est terminée au tour t, toutes les variables $move_{t,x,y}$ sont propagées à $faux$.

$$ladder_{t,first} \iff move_{t,x,y} \tag{4.45}$$

$$(\neg ladder_{t,m-1} \wedge ladder_{t,m}) \iff move_{t,x,y} \tag{4.46}$$

Nous ajoutons enfin notre condition gagnante par pavage de paires.

Nous devons d'abord nous assurer que si une paire de cellules est dans le pavage, alors aucune cellule de cette paire n'est occupée. Ainsi, pour chacune des cellules (x, y) d'une paire de cellules id, nous avons la contrainte suivante :

$$pair_{id} \implies \neg occupied_{k,x,y} \tag{4.47}$$

De plus, deux paires distinctes n'ont jamais de cellule commune. Par conséquent, pour des paires distinctes $id1, id2$ avec une cellule commune, nous avons la clause suivante :

$$pair_{id1} \implies \neg pair_{id2} \tag{4.48}$$

Aussi, si une paire est dans le pavage, elle annule toute forme la contenant. Nous avons donc, pour toutes les paires id contenues dans la forme i à la position (x, y), la clause suivante :

$$pair_{id} \implies canceled_{i,x,y} \tag{4.49}$$

Également, si la cellule contient une pierre blanche, alors une forme cible à une position donnée contenant cette cellule est annulée. Nous avons donc, pour toutes les cellules (x', y') contenues dans la forme i à la position (x, y), la clause suivante :

$$white_{k,x',y'} \implies canceled_{i,x,y} \tag{4.50}$$

Inversement, si une forme cible à une position donnée est annulée, alors elle contient soit une pierre blanche, soit une paire. Nous avons donc, pour la forme cible i à la position (x, y), $\mathcal{C}$ son ensemble de cellules et $\mathcal{P}$ l'ensemble des paires contenues dans cette forme à cette position, la clause suivante :

$$canceled_{i,x,y} \implies \Big(\bigvee_{(x',y') \in \mathcal{C}} white_{k,x',y'} \vee \bigvee_{id \in \mathcal{P}} pair_{id} \Big) \tag{4.51}$$

Enfin, toute forme i à n'importe quelle position (x, y) est annulée.

$$canceled_{i,x,y} \tag{4.52}$$

4.2 Résultats expérimentaux

Toutes les expériences sont effectuées sur un processeur Xeon CPU X5570, 2.93GHz, 64 Go de mémoire, avec un délai d'expiration (*timeout*) de 1000 secondes. Nous considérons les instances HTTT pour les polyominos formés d'au plus 6 cellules qui tiennent sur les plateaux et nous considérons à la fois les plateaux normaux (carrés) et ceux en forme de tore. Nous avons donc 48 instances sur un plateau de 3×3, 98 instances sur un plateau de 4×4, et 110 instances sur un plateau de 5×5. Nous présentons également quelques résultats sur le polyomino Snaky.

Les solveurs et préprocesseurs qui ont été choisis pour évaluer notre méthode sont les gagnants du QBFEval'19 [1], une compétition majeure de solveurs QBF. Ce sont aussi les mêmes solveurs et préprocesseurs

1. http ://www.qbflib.org/eval19.html

Tableau 4.1 – Temps de résolution en secondes avec et sans préprocesseur sur les plateaux de 3×3

Solveur	Préprocesseur	DYS	COR	PAIRING
DepQBF	Aucun	**12.85**	**1.12**	**0.77**
Caqe	Aucun	67.16	7.06	2.29
Qute	Aucun	5609.28	14.27	3.71
QESTO	Aucun	3171.66	9.21	3.00
DepQBF	Bloqqer	4.00	2.77	1.16
Caqe	Bloqqer	6.27	3.95	2.47
Qute	Bloqqer	7.91	13.50	2.95
QESTO	bloqqer	**3.47**	1.65	1.69
DepQBF	HQsPre	16.77	1.61	0.65
Caqe	HQsPre	645.47	44.42	38.74
Qute	HQsPre	20.37	14.41	3.79
QESTO	HQsPre	272.74	23.54	4.20
DepQBF	QRATPre+	7.97	**0.85**	**0.32**
Caqe	QRATPre+	109.33	2.77	0.51
Qute	QRATPre+	4120.26	19.05	2.90
QESTO	QRATPre+	1084.74	3.41	1.20

qui ont été choisis pour l'article de (14). Nous avons utilisé les versions les plus récentes des solveurs QBF suivants : DepQBF v6.03 (51), CAQE v4.0.1 (52), Qute v1.1 (53), QESTO v1.0 (54), et quatre préprocesseurs (incluant aucun) : QRATPre+ v2.0 (57), HQSPre v1.4 (56), et bloqqer v37 (55). Tous ces programmes ont été utilisés avec les paramètres de ligne de commande par défaut.

Pour les plateaux de 3×3, nous comparons l'encodage DYS de (6), le *Corrective Encoding* COR (14) et notre propre encodage PAIRING. Nous montrons dans le tableau 4.1 toutes les combinaisons de solveurs et préprocesseurs (y compris sans préprocesseur) sur les plateaux de 3×3. Les valeurs en gras sont pour chaque encodage les meilleures résultats sans préprocesseur et les meilleurs combinaisons (solveur/préprocesseur). Il n'y a eu aucun délai d'expiration dans ce tableau puisque les instances sont relativement petites et tous les solveurs ont produit des résultats corrects. Il y a, dans ces instances, exactement 8 gagnants et 40 perdants. Dans tous les cas, avec ou sans préprocesseur, on voit dans les données de ce tableau que l'encodage

Tableau 4.2 – Temps total de résolution en secondes pour les plateaux de 4×4, nombres d'Inconnus, de Gagnants et de Perdants

Solveur	Préprocesseur	COR				PAIRING			
		Temps	I	G	P	Temps	I	G	P
DepQBF	Aucun	**17159.72**	7	14	77	**114.63**	0	14	84
Caqe	Aucun	26194.98	12	14	72	5182.05	4	10	84
Qute	Aucun	66155.46	61	12	25	6922.28	6	8	84
Qesto	Aucun	61674.89	42	12	44	2862.34	1	13	84
DepQBF	bloqqer	**12949.31**	5	14	79	93.35	0	14	84
Caqe	bloqqer	31952.97	9	13	62	1655.59	0	14	84
Qute	bloqqer	60207.25	46	9	43	5593.09	3	12	83
Qesto	bloqqer	27345.76	14	14	70	2267.41	0	14	84
DepQBF	HQsPre	19429.22	11	13	74	551.95	0	14	84
Caqe	HQsPre	91380.83	91	7	0	9207.89	9	6	83
Qute	HQsPre	75547.53	19	10	69	7832.95	7	8	83
Qesto	HQsPre	89334.36	88	9	1	11455.34	11	4	83
DepQBF	QRATPre+	14176.01	5	14	79	**61.31**	0	14	84
Caqe	QRATPre+	24756.23	13	14	71	1869.08	0	14	84
Qute	QRATPre+	61449.46	31	11	56	5298.52	4	11	83
Qesto	QRATPre+	52742.54	24	12	62	2081.14	1	13	84

PAIRING permet le temps de résolution le plus court. De plus, confirmant les résultats de (14), l'encodage DYS est presque dix fois moins efficace que l'encodage de COR et encore moins efficace si on le compare à l'encodage PAIRING. C'est pour cette raison que pour les plateaux de taille 4×4 et plus nous nous sommes concentrés seulement sur les encodages COR et PAIRING.

Pour un plateau de 4×4, le tableau 4.2 montre encore en gras les meilleurs résultats sans préprocesseur et les meilleurs couples (solveur/préprocesseur). De plus, les colonnes I/G/P indiquent le nombre d'instances pour lesquelles le résultat est Inconnu/Gagnant/Perdant. Un résultat est inconnu lorsque le solveur dépasse le délai d'expiration (*timeout*), ce qui s'est produit pour les plateaux de 4×4 contrairement aux plateaux de 3×3. On voit d'ailleurs dans ce tableau que seul l'encodage PAIRING a réussi à obtenir des résultats complets,

Tableau 4.3 – Temps de prétraitement en secondes pour les plateaux de 4×4

	COR			PAIRING		
	bloqqer	HQSPre	QRATPre+	bloqqer	HQSPre	QRATPre+
Moyenne	1.78	27.98	0.10	1.87	8.99	0.38
Maximum	2.11	65.39	0.20	2.25	40.95	0.71
Total	174.03	2742.51	9.65	183.30	881.19	37.31

c'est-à-dire sans aucun inconnu, soit 14 gagnants et 84 perdants. De plus, une observation frappante des résultats du tableau 4.2, et cela dans tous les cas, est que notre encodage PAIRING est plus de deux ordres de grandeur plus rapide que l'encodage COR, le tout sans délai d'expiration.

Au niveau des préprocesseurs, on remarque que l'usage de HQsPre n'est jamais bénéfique, car le temps des solveurs est plus long dans tous les cas que sans préprocesseur. Le temps de prétraitement (tableau 4.3) est également beaucoup plus court pour Bloqqer et QRATPre+ que pour HQSPre. Nous n'avons donc pas utilisé HQsPre pour les plateaux de 5×5. Finalement, on remarque que l'usage des préprocesseurs est beaucoup plus avantageux pour COR que PAIRING, car si l'on ajoute le temps de prétraitement au temps de résolution, le gain est moindre chez PAIRING que COR.

Il est également intéressant de comparer COR et PAIRING en termes de taille des instances. Nous montrons dans le tableau 4.4 le nombre total de littéraux, de clauses et de quantificateurs pour toutes les formes de plateaux normaux et toriques. Alors que le nombre de quantificateurs est similaire pour les deux encodages, puisque COR et PAIRING ne diffèrent que sur la condition gagnante, PAIRING génère plus de littéraux et de clauses, en particulier pour les plateaux toriques.

Par conséquent, même si de façon générale on s'attend à ce qu'une instance de taille plus petite soit plus rapide à résoudre, l'encodage PAIRING, bien qu'ayant plus de littéraux et de clauses, est considérablement plus rapide à résoudre que les instances générées avec COR. Les littéraux et clauses supplémentaires de PAIRING permettent en pratique de simplifier la résolution du problème, du moins sur les solveurs récents avec lesquels nous avons expérimenté. On peut donc penser que le nombre supplémentaire de contraintes dans PAIRING réduit les possibilités et ultimement l'espace de recherche, ce qui devrait être favorable au temps de calcul.

Tableau 4.4 – Nombre total de littéraux, clauses et quantificateurs sur les plateaux de 4×4

	COR				PAIRING			
	Litt.	Clauses	Univ.	Exist.	Litt.	Clauses	Univ.	Exist.
Normal	554629	227944	1568	66850	675390	285428	1568	70682
Tore	637683	253794	1568	70241	950037	388083	1568	75329

Sur les plateaux de 5×5, puisque les temps de calcul augmentent considérablement, nous avons testé uniquement les paires de solveurs et de solveurs/préprocesseurs qui ont donné les meilleures performances sur les plateaux de 4×4 (comme présenté dans le tableau 4.2). Encore une fois, au tableau 4.5, les colonnes I/G/P indiquent le nombre d'instances pour lesquelles le résultat est Inconnu/Gagnant/Perdant. On y observe que COR ne résout que 7 des 110 instances, tandis que PAIRING en résout 72 (70 sans préprocesseur). En ce qui concerne le temps d'exécution, PAIRING nécessite moins de la moitié du temps requis par COR. PAIRING surpasse clairement, encore une fois, COR.

Jusqu'à maintenant, nous avons cherché une solution en considérant toujours une valeur de k maximale, c'est-à-dire une partie complète. Cela est correct puisqu'une partie sur un plateau de $H \times W$ ne peut dépasser $H \times W$ coups. Néanmoins, le temps d'exécution augmente rapidement avec la longueur de la partie k et résoudre la partie nécessite un $k = 25$ sur un plateau de 5×5, tandis que $k = 9$ et $k = 16$ suffisent pour les plateaux de 3×3 et 4×4, respectivement.

Il pourrait donc être profitable d'appliquer l'approfondissement itératif (*iterative deepening*) comme le propose (14) avec l'encodage COR. Le principe est que si l'on trouve une stratégie gagnante pour une valeur plus petite de k, alors cette stratégie reste valide pour k. L'approfondissement itératif consiste à tenter de trouver, de façon itérative, une stratégie pour des k croissants, arrêtant dès que l'on en a trouvé une. Comme le temps de calcul augmente très rapidement, en fonction de k, cette approche est plus efficace que de tenter directement le k maximal.

Nous allons néanmoins pousser ce principe encore plus loin en combinant l'encodage COR de (14) qui recherche une stratégie gagnante pour Noir à notre encodage qui recherche une stratégie de blocage pour Blanc. En alternant les deux encodages, de façon itérative, on peut trouver une solution beaucoup plus rapidement, comme nous allons le démontrer.

Tableau 4.5 – Temps de résolution en secondes pour les plateaux de 5×5, nombres d'Inconnus, de Gagnants et de Perdants

Solveur	Préprocesseur	COR				PAIRING			
		Temps	I	G	P	Temps	I	G	P
DepQBF	Aucun	**103083.14**	**103**	7	0	41365.22	40	10	60
DepQBF	bloqqer	103213.42	103	7	0	**38901.20**	**38**	10	62
DepQBF	QRATPre+	103114.56	103	7	0	39393.55	38	10	62

Notre approche débute au tour 0 avec notre encodage PAIRING pour déterminer l'existence d'une stratégie de blocage avant même de jouer le premier tour. On continue avec COR au premier tour puis avec PAIRING au deuxième tour. On alterne ainsi les encodages et le premier encodage qui retourne SAT nous donne le résultat. Notre approche nous permet d'établir, dès que l'un ou l'autre retourne une instance SAT, le statut (gagnant/perdant) du polyomino. Le fait d'y aller en itération nous permet de gagner en performance en limitant l'espace de recherche puisqu'on devrait trouver la solution, en général, bien avant d'avoir atteint le k maximal.

Le tableau 4.6 montre le temps cumulatif de résolution des instances pour des valeurs croissantes $k = 0, 1, 2, \ldots, 25$ en utilisant, comme nous venons de le décrire, l'encodage PAIRING pour les valeurs paires de k et l'encodage COR pour les valeurs impaires de k, en s'arrêtant à la première instance SAT.

En comparant les tableaux 4.5 et 4.6, on constate que l'approfondissement itératif combinant COR et PAIRING est effectivement efficace. Il réduit le nombre total de dépassements du délai d'attente (les inconnus) et le temps de résolution de près de moitié, par rapport à PAIRING seul qui donne les meilleurs résultats au tableau 4.5. Ici encore, les colonnes I/G/P sont pour Inconnu/Gagnant/Perdant dans les deux tableaux. Les 110 instances de plateau 5×5 sont utilisés pour comparer COR et PAIRING dans le tableau 4.5, tandis que le tableau 4.6 montre la différence entre les 55 instances de plateau 5×5 normaux contre les 55 instances de plateau 5×5 toriques.

Le tableau 4.6 montre également qu'il existe une grande différence entre les plateaux normaux et les plateaux toriques. Cela est naturel puisqu'il existe plus de façons de gagner pour Noir sur un plateau torique que sur un plateau normal et donc un espace de recherche plus grand. Cela se traduit aussi par plus de clauses, comme nous l'avons vu dans le tableau 4.4. Un autre fait remarquable sur les instances 5×5 est

Tableau 4.6 – Temps de résolution par approfondissement itératif en secondes pour les plateaux de 5×5, nombres d'Inconnus, de Gagnants et de Perdants

Solveur	Préprocesseur	Normal				Tore			
		Temps cumulé	I	G	P	Temps cumulé	I	G	P
DepQBF	Aucun	1023.71	1	7	47	21376.28	21	7	27
DepQBF	bloqqer	1016.29	1	7	47	21012.67	20	7	28
DepQBF	QRATPre+	1015.34	1	7	47	21310.61	20	7	28

que les préprocesseurs offrent très peu de gains de performance, tant en termes de temps de résolution que de nombre de formes résolues. Cela n'était pourtant pas le cas pour les plateaux de 4×4 et 3×3.

Il est aussi intéressant de regarder la distribution de la dernière valeur k considérée avec la méthode d'approfondissement itératif sur les plateaux de 5×5. Le tableau 4.7 montre, pour chaque $k = 0, \ldots, 11$, le nombre de polyominos pour lesquels la première instance SAT (ou délai d'attente) est atteinte pour cette valeur de k.

On note tout d'abord qu'il y a des polyominos résolus pour chaque valeur de k, à la fois paire et impaire. Il en résulte que ni les valeurs paires ni les valeurs impaires de k ne peuvent être ignorées et que les deux encodages COR et PAIRING sont essentiels pour rendre cette méthode efficace. Cela confirme notre intuition qu'il est préférable de combiner les deux méthodes que de se restreindre à une seule.

La grande question ouverte autour du jeu de HTTT reste évidemment le statut de Snaky sur un plateau de 9×9, et comme indiqué par (14), il s'agit d'un défi pour la communauté QBF. Il est donc intéressant de voir comment l'encodage PAIRING se comporte avec ce polyomino. Cependant, comme le montre le tableau 4.8, qui indique le temps de résolution en secondes pour DepQBF sur des plateaux normaux, le statut de perdant de Snaky sur un plateau de 8×8, un fait connu (63), est déjà hors de portée de notre codage PAIRING puisque 120 000 secondes représentent plus de 33 heures de calcul. Cependant, l'article (63) décrit une stratégie bloquante obtenue par un algorithme *branch-and-cut*, mais sans donner de temps de calcul qui permettrait de comparer l'efficacité de leur méthode avec la nôtre. De son côté, notre résultat pourra servir de base de comparaison pour de futures méthodes utilisant l'approche QBF pour déterminer le statut de Snaky ainsi que le nombre de tours nécessaires à cette détermination. Pour notre méthode, on voit très bien, au tableau 4.8, l'explosion exponentielle du temps de traitement nécessaire pour obtenir

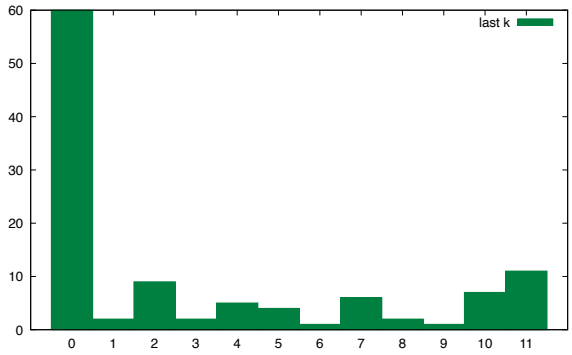


Tableau 4.7 – dernier k, DepQBF

Tableau 4.8 – PAIRING Snaky

Plateau	k	Résultat	Temps
6×6	0	UNSAT	1.53
	2	SAT	17.53
7×7	0	UNSAT	0.81
	2	UNSAT	345.64
	4	SAT	2277.39
8×8	0	UNSAT	0.79
	2	UNSAT	1857.64
	4	UNSAT	10596.75
	6	UNSAT	120004.47

une réponse. Comme déjà mentionné, on voit notamment que la résolution pour $k = 6$ sur un plateau de 8×8 correspond à plus de 33 heures de traitement. Résoudre Snaky sur un plateau de 9×9 nécessitera des avancées supplémentaires en matière de résolution et d'encodage QBF et reste donc un défi pour la communauté QBF.

4.3 Conclusion

Dans ce chapitre, nous avons présenté l'encodage PAIRING qui encode le problème du jeu HTTT dans la perspective du deuxième joueur (Blanc).

Nous avons introduit le concept de stratégie de pavage qui est essentiellement un partitionnement du plateau en paires de cellules. Si Noir joue sur une cellule, alors Blanc joue sur la cellule associée. Si toutes les formes cibles du plateau contiennent une paire, alors nous avons une stratégie de blocage par pavage de paires. Nous avons la preuve que Noir ne peut pas gagner.

Nous avons aussi présenté l'encodage PAIRING, les variables, l'ordre des quantifications et les clauses de l'encodage.

Ensuite, nous avons dévoilé nos résultats expérimentaux et démontré que l'encodage PAIRING QBF conduit à une résolution QBF très efficace pour le jeu de HTTT sur les plateaux de 3×3 et 4×4. De plus, l'appro-

fondissement itératif (*iterative deepening*) combinant les codages COR et PAIRING peut résoudre le jeu de HTTT sur des plateaux de 5×5 pour la plupart des formes cibles. Nous avons également révélé nos résultats sur la résolution du Snaky.

De façon plus générale, les résultats de ce chapitre montrent tout l'intérêt de l'exploration d'encodages alternatifs, comme PAIRING, pour résoudre le même problème, mais en l'exprimant d'une façon différente. De plus, nous avons montré que l'on peut combiner à profit des encodages complémentaires, comme COR et PAIRING, une méthodologie introduite par (68) sous le nom de primal/dual. Ce type d'approches est prometteur et il reste encore bien d'autres façons d'aborder le jeu de HTTT avec des expressions QBF.

Pour terminer, bien que nos résultats fassent avancer la résolution QBF des *achievement games*, ils ne permettent toujours pas de résoudre le statut de Snaky sur un plateau de 9×9. Cela nécessitera des avancées supplémentaires dans la résolution QBF, où les encodages duals (68) et la restriction du type de stratégie recherchée pourraient bien jouer un rôle important !

CHAPITRE 5
L'ENCODAGE COVER

5.1 Introduction

Dans ce chapitre, nous allons présenter l'encodage COVER. Contrairement à l'encodage PAIRING, COVER encode le problème du jeu de HTTT selon la perspective du premier joueur (Noir). Il s'agit d'établir une stratégie gagnante pour Noir. Les instances QBF de ce chapitre sont par conséquent satisfiables (SAT) si et seulement si Noir trouve une façon de gagner, quels que soient les coups de Blanc. Dans ce sens, il est similaire aux encodages de DYS, COR et COR+, car ils sont tous créés selon la perspective du premier joueur (Noir). Bien entendu, notre encodage comporte plusieurs différences que nous allons introduire dans ce chapitre. Nous présenterons d'abord les principes utilisés dans le *cover*, puis les variables, les quantificateurs et les clauses utilisés, et nous terminerons en faisant part de nos résultats expérimentaux.

5.1.1 Le cover

Les encodages QBF de DYS, COR et COR+ présentent tous un long préfixe de quantificateurs (prenex), avec un nombre de quantifications proportionnel au nombre de tours du jeu (k). De plus, chaque bloc de quantificateurs exprime le coup d'un joueur avec une longueur proportionnelle au nombre de coups possibles. L'objectif premier de l'encodage COVER est de réduire le nombre de choix possibles qu'un joueur peut effectuer dans le but de réduire le nombre de quantifications dans les blocs et ultimement l'espace d'état. C'est en effet un objectif naturel car, au fur et à mesure que la partie avance, certains choix sont évidemment plus judicieux que d'autres, ce qui devrait justement nous permettre de réduire le nombre de choix possibles tout en obtenant toujours une stratégie gagnante.

Notre méthode choisit un ensemble de cellules spécifiques, appelé le *cover*. Plus précisément, on définit cet ensemble pour qu'il contienne les cellules de toutes les formes qui contiennent le premier coup de Noir. Le but est de se concentrer sur des cellules offrant la possibilité de compléter une forme déjà partiellement occupée.

Restreindre les coups de Noir au *cover* est adéquat dans le sens où, si une stratégie gagnante limitant les coups de Noir est trouvée, c'est bel et bien une stratégie gagnante pour Noir. En revanche, cette approche

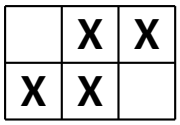


Figure 5.1 – Polyomino Tippy

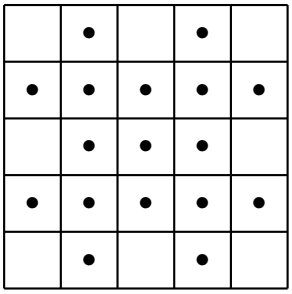


Figure 5.2 – Cover pour Tippy

n'est pas complète, en pratique, puisqu'il pourrait exister une stratégie gagnante pour Noir qui ne respecte pas cette limitation. Néanmoins, nous montrerons dans la section 5.2 que dans nos expériences cela ne se produit pas et qu'une stratégie gagnante pour Noir est trouvée avec COVER chaque fois qu'une stratégie générale est trouvée avec COR+. Donc, bien que cette méthode ne soit pas complète, en pratique elle est tout aussi effective que la meilleure méthode complète que l'on connaît.

Quant à restreindre les coups de Blanc au *cover*, cela est clairement incorrect. En effet, il faut s'assurer que Noir a une stratégie gagnante, quels que soient les coups choisis par Blanc. Pour rétablir la validité de l'approche, notre encodage COVER restreint les coups de Blanc au *cover*, mais autorise également un coup supplémentaire spécial « hors *cover* »(ooc) pour Blanc. COVER compte aussi le nombre de fois où Blanc joue ce coup spécial hors *cover* afin de vérifier la condition gagnante et de rétablir la validité de l'approche, comme nous le verrons bientôt.

Prenons l'exemple du polyomino Tippy de la figure 5.1. Si Noir joue son premier coup en plein centre sur un plateau de 5×5, en disposant tous les Tippy (à rotation et réflexion près) qui contiennent cette case du centre, on obtient le *cover* vu à la figure 5.2. Dans cet exemple, Noir a seulement 17 cellules à considérer plutôt que 25 pour ses prochains coups. De plus, notre encodage COVER tient compte du fait que Blanc peut jouer hors *cover*, mais ce sans devoir différencier les 8 cellules hors *cover* de notre exemple, comme nous le verrons dans un instant.

Dans DYS, COR et COR+, la condition gagnante est simplement que, à t_{end}, le dernier tour du jeu, Noir a réussi à compléter une forme cible mais pas Blanc. Comme Noir peut arrêter la partie à tout moment, empêchant que d'autres cellules soient revendiquées, cela lui garantit d'avoir une stratégie gagnante qui complète la forme avant Blanc.

La condition gagnante de COVER vérifie toujours que Noir a complété la forme cible, mais pour Blanc COVER considère plutôt, pour toutes les formes cibles F, la partie de cette forme cible F_c située dans le *cover* et celle F_o située en dehors du *cover* de la manière suivante : pour chaque forme cible F, pour laquelle la taille de F_o ne dépasse pas le nombre de coups hors *cover* de Blanc, COVER vérifie que Blanc n'a pas complété F_c. Cela garantit que Blanc n'aurait pas pu compléter F, quel que soit l'emplacement où il aurait pu jouer ses coups hors *cover* puisque même si Blanc avait joué sur toutes les cases de F_0 - ce qui pouvait se produire vu le nombre de coups hors *cover* - il n'aurait pas réalisé F. Cette approche est donc valide puisqu'elle vérifie que Blanc ne peut pas gagner.

Reprenons l'exemple du Tippy de la figure 5.1 et regardons les figures 5.3 et 5.4. Les c sont des cellules dans le *cover*. Blanc doit jouer sur chacune de ces cellules spécifiquement pour compléter la forme gagnante formée des cellules identifiées par c et h. Dans le cas de la figure 5.3, Blanc doit jouer sur les trois cellules c et une fois hors *cover*, puisque la cellule h est nécessaire, pour compléter la forme. D'un autre côté, à la figure 5.4, pour compléter la forme gagnante des cellules identifiées par les c, Blanc doit compléter la forme normalement puisque la forme est entièrement dans le *cover*.

Prenons le cas de la figure 5.5, qui est une situation hypothétique où le *cover* est composé des deux premières colonnes de la partie gauche du plateau. Blanc a joué sur les deux cellules c et deux fois hors *cover*. Il peut réussir à compléter le Tippy de deux manières différentes en combinant les c avec les h ou en combinant les c avec les H. Donc Blanc doit jouer sur les deux c et deux fois hors *cover* pour compléter le Tippy. À la figure 5.6, si Blanc joue quatre fois hors *cover*, il peut compléter tous les Tippy possibles que les cellules hors *cover* lui permettent, comme compléter la forme hors *cover* contenant les h.

Il est important de comprendre que Blanc ne joue pas les coups hors *cover* proprement dit, mais toujours un seul et même coup hors *cover*. Seul le nombre de coups effectués hors *cover* est calculé, ce qui est suffisant pour déterminer si Blanc pourrait compléter une forme en utilisant ce nombre de cellules hors *cover*.

En résumé, avec un *cover* de taille n, Noir est restreint à n coups possibles, tandis que Blanc est restreint à exactement $n + 1$ coups, comprenant, bien sûr, le coup hors *cover*. COVER limite donc le nombre de coups pour Noir comme pour Blanc.

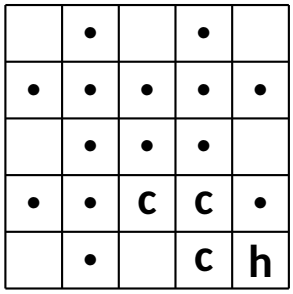


Figure 5.3 – Tippy avec une cellule hors *cover*

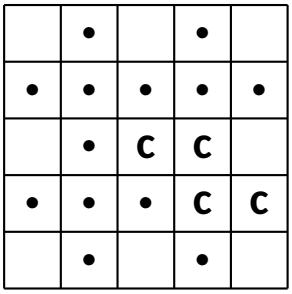


Figure 5.4 – Tippy dans le *cover*

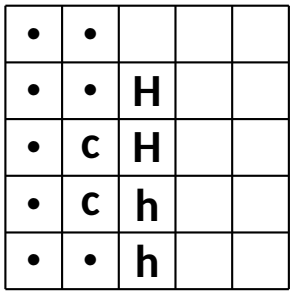


Figure 5.5 – Tippy avec deux cellules hors *cover*

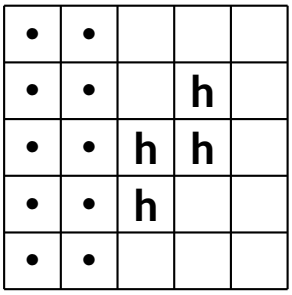


Figure 5.6 – Tippy entièrement hors *cover*

5.1.2 L'encodage COVER

Au niveau technique, l'encodage COVER suit une démarche similaire à celle de PAIRING, sauf que les joueurs sont inversés : Blanc est le joueur universel et Noir est le joueur existentiel. COVER intègre aussi les idées mises de l'avant par COR (14) et sa version améliorée COR+ (15).

5.1.2.1 Les variables

De la même façon que PAIRING, l'encodage COVER est paramétré par la hauteur H et la largeur W du plateau, et par le nombre de tours k. Les tours sont numérotés $0, 1, \ldots, k$. Le plateau est initialisé au tour 0 pour ne contenir aucune pierre et le choix des cellules par les joueurs se produit aux tours $1, \ldots, k$. Ainsi, Noir joue aux tours $1, 3, \ldots,$ et Blanc joue aux tours $2, 4, \ldots$

Voici les variables représentant le jeu et leur signification. L'étendue des paramètres x, y, e, t ci-dessous est la suivante, sauf indication contraire : $1 \leq x \leq W, 1 \leq y \leq H$ et $e \in E$ où E est l'ensemble des formes cibles sur le plateau et $t \in T = \{0, \ldots, t_{end}\}$ est l'ensemble des tours du jeu.

$time_t$: la partie est en cours au tour t (5.1)

$moveB_{t,j}$: j-ième chiffre de l'encodage binaire du choix de coup de Noir au tour t (5.2)

$moveW_{t,j}$: j-ième chiffre de l'encodage binaire du choix de coup de Blanc au tour t (5.3)

$incoverS_e$: la forme e est contenue dans le *cover* (5.4)

$incoverP_{x,y}$: la cellule x, y est dans le *cover* (5.5)

$out\_of\_cover\_inc_{t,i}$: Blanc a joué hors *cover* un nombre i de fois au tour t (5.6)

$black_{t,x,y}$: il y a une pierre noire sur la case x, y au tour t (5.7)

$white_{t,x,y}$: il y a une pierre blanche sur la case x, y au tour t (5.8)

win_e : toutes les cellules de la forme e sont noires au tour t_{end} (5.9)

5.1.2.2 Les quantificateurs

Voici les blocs de quantificateurs. Ils apparaissent dans l'ordre des tours en commençant par le tour $t = 0$ où le plateau est initialisé, comme suit :

$$\exists time_0 \tag{5.10}$$

$$\exists black_{0,x,y} \tag{5.11}$$

$$\exists white_{0,x,y} \tag{5.12}$$

À $t = 1$, Noir joue et le *cover* est défini.

$$\exists time_1 \tag{5.13}$$

$$\exists moveB_{1,j} \tag{5.14}$$

$$\exists incoverS_e \tag{5.15}$$

$$\exists incoverP_{x,y} \tag{5.16}$$

$$\exists black_{1,x,y} \tag{5.17}$$

$$\exists white_{1,x,y} \tag{5.18}$$

Pour $t = 2, ..., t_{end}$, Blanc joue pour t pair et Noir joue pour t impair. De plus, aux tours de Blanc, le compte des mouvements hors *cover* est enregistré avec les variables $out\_of\_cover\_inc_{t,i}$.

$$\exists time_t \tag{5.19}$$

$$\forall moveW_{t,j} \tag{5.20}$$

$$\exists moveB_{t,j} \tag{5.21}$$

$$\exists out\_of\_cover\_inc_{t,i} \tag{5.22}$$

$$\exists black_{t,x,y} \tag{5.23}$$

$$\exists white_{t,x,y} \tag{5.24}$$

Enfin, au dernier tour du jeu t_{end}, nous avons :

$$\exists win_e \tag{5.25}$$

5.1.2.3 Les clauses

Voici les clauses qui définissent l'encodage COVER. Comme pour PAIRING, $t = 1, \ldots, k$, $x = 1, \ldots, W$ et $y = 1, \ldots, H$.

Si la partie est toujours en cours au tour t, alors elle l'était déjà au tour $t-1$.

$$time_t \implies time_{t-1} \tag{5.26}$$

Il n'y a pas de pierre sur le plateau au tour $t = 0$.

$$\neg black_{0,x,y} \land \neg white_{0,x,y} \tag{5.27}$$

Les deux joueurs ne peuvent pas jouer sur la même cellule.

$$\neg black_{t,x,y} \lor \neg white_{t,x,y} \tag{5.28}$$

Si une pierre était sur une cellule au tour précédent, alors elle reste là au tour courant.

$$black_{t-1,x,y} \implies black_{t,x,y} \tag{5.29}$$

$$white_{t-1,x,y} \implies white_{t,x,y} \tag{5.30}$$

Lorsque la partie est terminée, aucune nouvelle pierre n'apparaît sur le plateau.

$$(\neg time_t \land \neg black_{t-1,x,y}) \implies \neg black_{t,x,y} \tag{5.31}$$

$$(\neg time_t \land \neg white_{t-1,x,y}) \implies \neg white_{t,x,y} \tag{5.32}$$

Lorsque ce n'est pas le tour d'un joueur, une pierre de sa couleur ne peut apparaître sur le plateau.

$$\neg black_{t-1,x,y} \implies \neg black_{t,x,y}; \text{ pour } t \text{ pair} \tag{5.33}$$

$$\neg white_{t-1,x,y} \implies \neg white_{t,x,y}; \text{ pour } t \text{ impair} \tag{5.34}$$

Par définition, le *cover* contient les cellules de toutes les formes cibles qui incluent le premier coup de Noir. Donc, (5.35) définit que $incoverS_e$ est $vrai$ lorsqu'une des cellules de la forme cible e a été jouée lors du premier coup de Noir.

$$incoverS_e \iff \bigvee_{(x,y)\in e} black_{1,x,y} \quad \text{pour } e \in E \tag{5.35}$$

Ensuite, (5.36) exprime que la cellule x, y est incluse dans le *cover* si elle est une cellule contenue dans une forme cible e du *cover*.

$$incoverP_{x,y} \iff \bigvee_{\{e \in E; (x,y) \in e\}} incoverS_e \tag{5.36}$$

Enfin, Noir ne peut pas jouer en dehors du *cover*, comme prévu.

$$\neg incoverP_{x,y} \implies \neg black_{t_{end},x,y} \tag{5.37}$$

Les clauses (5.38) et (5.39) représentent le choix de Noir. Un coup qui place une pierre sur la cellule x, y est encodé par le nombre binaire $[x, y]$. Ces clauses forment une contrainte *At-Most-One* de (14), telle que nous l'avons vue à la section 1.6.3. Cette contrainte empêche Noir de jouer plusieurs fois au même tour. Ces clauses expriment que la cellule x, y devient noire au tour t si elle n'a pas été jouée auparavant et que le j-ième chiffre du codage binaire représentant le choix de Noir est $moveB_{t,j}$.

$$(\neg black_{t-1,x,y} \wedge black_{t,x,y}) \implies moveB_{t,j} \qquad ; [x, y](j) = 1 \tag{5.38}$$

$$(\neg black_{t-1,x,y} \wedge black_{t,x,y}) \implies \neg moveB_{t,j} \qquad ; [x, y](j) = 0 \tag{5.39}$$

Comme nous l'avons vu à la section 2.1.2, on peut briser la symétrie. Sur un plateau normal, il suffit de considérer le cas où le premier joueur (Noir) joue dans le triangle inférieur gauche du plateau. Sur un plateau en forme de tore, le premier coup de Noir est sans importance puisque toutes les cellules sont équivalentes par symétrie. On peut donc choisir arbitrairement une cellule. Les équations (5.40) et (5.41) expriment ces deux situations.

$$\bigvee_{x=1}^{\lceil \frac{W}{2} \rceil} \bigvee_{y=1}^{\lceil \frac{H}{2} \rceil} black_{1,x,y} \text{ ; pour un plateau normal} \tag{5.40}$$

$black_{1,\lceil \frac{W}{2} \rceil, \lceil \frac{H}{2} \rceil}$; pour un plateau en forme de tore (5.41)

Pour les coups de Blanc, nous distinguons deux cas. Si Blanc joue à l'intérieur du *cover*, la clause (5.42) exprime que, si le jeu est toujours en cours, la cellule à x, y n'est pas noire, x, y est dans le *cover* et Blanc choisit la cellule x, y, alors la cellule à x, y devient blanche.

$$(time_t \wedge \neg black_{t-1,x,y} \wedge incoverP_{x,y} \wedge \bigwedge_{j;[x,y](j)=0} moveW_{t,j} \bigwedge_{j;[x,y](j)=1} moveW_{t,j}) \implies white_{t,x,y} \quad (5.42)$$

Toutefois, si Blanc choisit plutôt le coup hors *cover*, le compteur $out\_of\_cover\_inc$ est incrémenté. La clause (5.43) exprime que, si le jeu est toujours en cours, Blanc choisit le coup hors *cover*, et $out\_of\_cover\_inc_{t-2,i-1}$, alors $out\_of\_cover\_inc_{t,i}$. On remarque que la valeur la plus élevée atteinte pour i sera égale au nombre total de tours où Blanc a joué hors *cover*.

$$(time_t \bigwedge_{j;[ooc](j)=0} moveW_{t,j} \bigwedge_{j;[ooc](j)=1} moveW_{t,j} \wedge out\_of\_cover\_inc_{t-2,i-1}) \implies out\_of\_cover\_inc_{t,i} \quad (5.43)$$

Pour connaître la valeur binaire du choix de Blanc de jouer hors *cover* [ooc], il suffit d'utiliser une valeur quelconque qui n'est pas déjà utilisée pour représenter une position à l'intérieur du *cover* sur le plateau. Nous numérotons séquentiellement les cellules du plateau et utilisons la valeur binaire du nombre de cellule dans le *cover* + 1 pour [ooc].

Le compteur $out\_of\_cover\_inc_{t,0}$ doit être initialisé à 0 pour commencer.

$$out\_of\_cover\_inc_{t,0} \quad (5.44)$$

Ensuite, lorsque i est incrémenté par les clauses (5.43), alors cet incrément demeure pour le reste de la partie avec les clauses (5.45). Notez que, dans toutes ces clauses, i atteint la taille du nombre de cellules de

la forme cible. Cela est suffisant car il n'y a aucun intérêt à compter des coups pour plus de cellules que le nombre de cellules de la forme cible, comme nous le verrons dans la condition gagnante.

$$out\_of\_cover\_inc_{t-2,i} \implies out\_of\_cover\_inc_{t,i} \tag{5.45}$$

La clause (5.46) exprime que win_e est $vrai$ seulement si Noir a complété toutes les cellules d'une forme cible e. Ensuite, (5.47) garantit la première partie de la condition gagnante, à savoir que Noir doit compléter au moins une forme cible.

$$win_e \iff \bigwedge_{(x,y)\in e} black_{t_{end},x,y} \tag{5.46}$$

$$\bigvee_{e\in E} win_e \tag{5.47}$$

Nous devons néanmoins nous assurer que Blanc n'aurait pas pu gagner pendant le temps que Noir complétait une forme cible. Nous considérons, pour toute forme e et tout nombre i de coups de Blanc hors *cover*, tous les sous-ensembles p de la forme e contenant i éléments et nous exprimons à l'aide des clauses (5.48) que le fait d'avoir simultanément i coups de Blanc hors *cover*, toutes les cellules de $e \setminus p$ (toutes les cellules de e sauf celles de p) blanches, et toutes les cellules de p en dehors du *cover*, est impossible. Par conséquent, quels que soient les coups que Blanc aurait pu faire hors *cover*, cela n'aurait jamais pu compléter une forme. Cela garantit que Blanc n'aurait pas pu compléter une forme cible.

$$\neg out\_of\_cover\_inc_{t_{end},i} \bigvee_{(x,y)\in e\setminus p} \neg white_{t_{end},x,y} \bigvee_{(x,y)\in p} incoverP_{x,y} \tag{5.48}$$

5.2 Résultats expérimentaux

Nous comparons notre approche avec le meilleur encodage QBF actuel pour le jeu de HTTT, qui est l'encodage COR+ (15), une version améliorée de l'encodage COR de (14). C'est aussi pour cette raison que l'enco-

dage COR n'a pas été utilisé dans les évaluations de ce chapitre.

Nous évaluons notre encodage avec les solveurs QBF suivants, qui sont les mêmes qui ont servi à l'évaluation de PAIRING : DepQBF v6.03 (51), CAQE v4.0.1 (52), Qute v1.1 (53), QESTO v1.0 (54), et avec les préprocesseurs (y compris aucun) : QRATPre+ v2.0 (57), HQSPre v1.4 (56) et bloqqer v37 (55). Il s'agit d'ailleurs des versions les plus récentes de ces solveurs.

Toutes les expériences sont réalisées sur un ordinateur Dell OptiPlex 7050, Intel Core i7-7700 Quad-Core à 3,6 GHz, avec 16 Go de RAM DDR4 à 2400 MHz. Chaque solveur et préprocesseur a été utilisé avec les paramètres de ligne de commande par défaut.

Comme nous l'avons vu au chapitre 4 et aussi dans (35), PAIRING résout facilement toutes les instances sur un plateau de 4×4. De plus, COVER étant tout aussi efficace que PAIRING sur un plateau de cette taille, nos expériences sont effectuées sur les plateaux de 5×5, à la fois normal et en tore. Comme dans (6)(14), nous considérons tous les polyominos de tailles spécifiques. En particulier, nous expérimentons sur les 47 polyominos qui tiennent sur un plateau de 4×4 (afin de laisser un peu d'espace supplémentaire pour des formes non triviales sur nos plateaux de 5×5), à l'exception de l'Elam, qui est la forme à une seule cellule et qui est évidemment un gagnant au premier coup de Noir.

5.2.1 Profondeur itérative sur des plateaux de 5 x 5

Dans cette sous-section, nous comparons la méthode par profondeur itérative avec COR+/PAIRING à celle avec COVER/PAIRING.

Cette méthode produit 3177 instances QBF chacun pour COR+, COVER et PAIRING afin de résoudre les instances, comme expliqué à la section 4.2. Les résultats sont résumés dans le tableau 5.1.

Dans le tableau 5.1, les colonnes **I/G/P** indiquent Inconnu/Gagnant/Perdant, c'est-à-dire les cas restants inconnus où aucune stratégie gagnante ni de blocage n'est trouvée, le nombre de formes/plateaux pour lesquels une stratégie gagnante pour Noir est trouvée avec COR+ et COVER, et une stratégie de blocage pour Blanc est trouvée avec PAIRING.

On remarque qu'il n'y a qu'une seule valeur pour **B**, car le même nombre de stratégies gagnantes pour Noir a

Tableau 5.1 – Profondeur itérative pour COR+/PAIRING et COVER/PAIRING sur des plateaux de 5×5 avec un délai d'expiration de 2500 secondes

Solveur	Préprocesseur	I/G/P	COR+/PAIRING	COVER/PAIRING
CAQE	Aucun	19/13/62	48466.13	48397.52
	Bloqqer	18/**14**/62	45665.90	45308.02
	HQSPre	19/13/62	50256.66	50248.72
	QRATPre+	18/**14**/62	46024.13	45530.75
DepQBF	Aucun	18/**14**/62	46669.55	45903.58
	Bloqqer	18/**14**/62	47591.13	46078.10
	HQSPre	18/**14**/62	47491.20	46655.83
	QRATPre+	18/**14**/62	47175.92	46455.84
QESTO	Aucun	16/**14**/**64**	**44035.47**	**44911.54**
	Bloqqer	16/**14**/**64**	41270.15	41196.53
	HQSPre	18/12/**64**	48757.67	48756.93
	QRATPre+	16/**14**/**64**	**41868.17**	**41834.90**
Qute	Aucun	22/12/60	55356.24	55265.69
	Bloqqer	22/12/60	57134.60	57095.74
	HQSPre	21/12/61	53129.73	53122.35
	QRATPre+	22/12/60	55470.14	55385.08

été trouvé avec COR+ et avec COVER. De plus, nous avons vérifié que COR+ et COVER identifient tous les deux une stratégie gagnante dans les mêmes cas (forme/plateau) et au même tour de jeu (t_{end}). Ainsi, les deux stratégies gagnantes comportent le même nombre de coups pour les mêmes paires de formes/plateaux. Cela montre que, bien que COVER se limite à un type précis de stratégie gagnante, cela n'empêche pas qu'une stratégie gagnante soit trouvée avec COVER chaque fois qu'il y en a une trouvée avec COR+. COVER est donc tout aussi efficace à trouver une stratégie gagnante pour Noir qu'une méthode complète comme COR+ et la restriction à un type spécifique de stratégie n'a pas d'incidence en pratique, du moins sur notre jeu de données.

Concernant les performances des solveurs/préprocesseurs, on note dans le tableau 5.1 que QESTO est le solveur qui atteint le plus grand nombre de stratégies gagnantes/bloquantes, à savoir 14 et 64, respective-

ment. De plus, d'après les deux dernières colonnes du tableau, pour tous les préprocesseurs sauf HQSPre, QESTO obtient le meilleur temps d'exécution total, avec et sans préprocesseur. Enfin, les temps d'exécution pour COR+/PAIRING et COVER/PAIRING sont similaires. Globalement, le tableau 5.1 montre que COVER est tout aussi efficace que COR+ dans ce contexte de profondeur itérative avec PAIRING.

Cependant, comme la profondeur itérative s'exécute jusqu'au délai d'expiration lorsqu'aucune stratégie gagnante ou de blocage n'est trouvée, et puisque le tableau 5.1 montre le temps total incluant le temps nécessaire à l'exécution de PAIRING, ce tableau ne permet pas de comparer correctement les performances de COR+ et COVER. À cette fin, nous effectuerons une comparaison directe des temps d'exécution de COR+ et COVER.

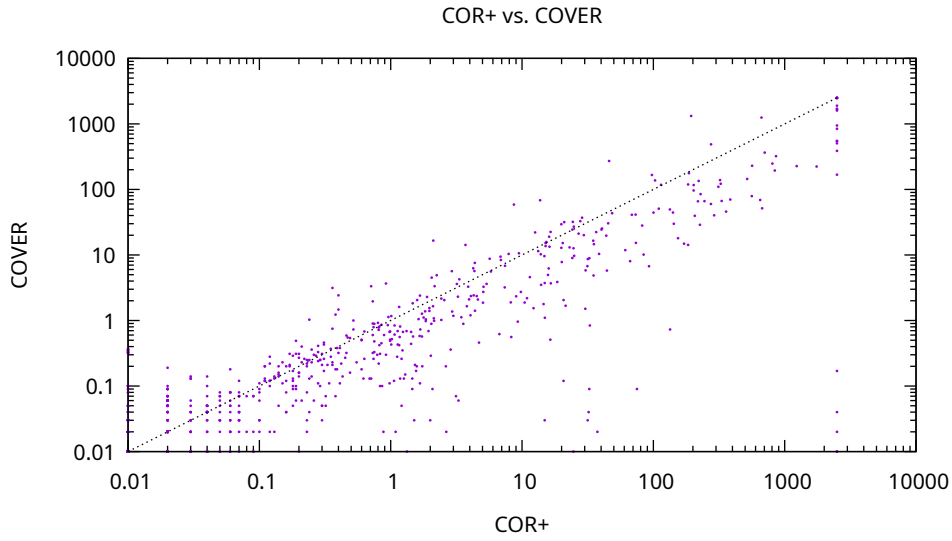


Figure 5.7 – Temps d'exécution pour COVER/COR+ sur des plateaux de 5×5 avec un délai d'expiration de 2500 secondes pour tous les préprocesseurs et solveurs

La figure 5.7 compare les temps d'exécution de COVER et COR+. Plus précisément, pour chaque type de plateau, forme, itération (valeur t_{end}) et paire préprocesseur/solveur des expériences présentées dans le tableau 5.1, la figure 5.7 montre les paires formées des temps d'exécution pour COVER et COR+. Cela permet ainsi de comparer COVER et COR+ sur la même tâche (trouver une stratégie gagnante pour Noir) pour une large gamme de configurations de jeu (type de plateau, forme, itération), et ce, avec de nombreuses paires préprocesseur/solveur. On compare donc COVER et COR+ sur exactement la même tâche, ce qui nous permet d'évaluer adéquatement leur performance relative.

Sur cette figure, on observe que les points apparaissent majoritairement sous la diagonale, indiquant que COVER surpasse COR+ dans la plupart des cas. De plus, une analyse détaillée des données montre que COVER est meilleur que COR+ dans les cas suivants :

- ✓ 61% $(336/547)$ des cas nécessitant plus de 0.01 seconde
- ✓ 70% $(248/355)$ des cas nécessitant plus de 0.1 seconde
- ✓ 74% $(136/183)$ des cas nécessitant plus de 1.0 seconde
- ✓ 76% $(74/98)$ des cas nécessitant plus de 10 secondes

Ainsi, COVER devient encore plus avantageux à mesure que la difficulté de l'instance augmente.

5.3 Conclusion

Dans ce chapitre, nous avons présenté l'encodage COVER qui encode le problème du jeu de HTTT avec la perspective du premier joueur (Noir).

Nous avons introduit le concept de *cover* qui est essentiellement une restriction du nombre de choix que Noir peut jouer dans le but de réduire l'espace de recherche. Le *cover* est l'ensemble des cellules de toutes les formes qui contiennent le premier coup de Noir. Ce même ensemble va aussi restreindre les coups de Blanc. Cependant, il faut comptabiliser le nombre de coups de Blanc hors *cover* pour s'assurer que la méthode reste valide.

Nous avons aussi présenté l'encodage COVER, les variables, l'ordre des quantifications et les clauses de l'encodage.

Nous avons fait part des résultats des expérimentations que nous avons effectuées et démontré que l'encodage QBF de COVER conduit à une résolution QBF efficace pour le jeu de HTTT sur les plateaux de 5×5 avec l'approfondissement itératif (*Iterative deepening*). En particulier, on a montré que COVER peut très bien se substituer à COR+ dans l'approfondissement itératif avec PAIRING. D'ailleurs, nos expérimentations montrent que COVER trouve une stratégie gagnante chaque fois que COR+ en trouve une, et qu'il n'y a donc pas d'inconvénients en pratique de remplacer la recherche d'une stratégie générale avec COR+ par la recherche d'une stratégie particulière avec COVER. En comparant ces encodages sur les mêmes tâches, on

démontre d'ailleurs qu'il y a un avantage à utiliser COVER comparativement à COR+ en termes de temps de calcul et qu'en fait COVER devient même plus avantageux à mesure que la difficulté de l'instance augmente.

Les travaux de ce chapitre innovent par le fait qu'il s'agit, à notre connaissance, du premier encodage QBF pour un jeu qui vise une stratégie d'un type particulier, plutôt qu'une stratégie générale. Comme nous avons démontré qu'il était efficace de procéder de cette manière et que cela ne se faisait pas au détriment du nombre de stratégie gagnantes trouvées, il s'agit tout probablement d'une voie d'avenir. On pourrait, par exemple, se demander s'il ne serait pas possible de trouver d'autres types de stratégies gagnantes qui pourraient mener à une résolution QBF encore plus efficace pour le jeu de HTTT. De façon encore plus générale, l'exploration de stratégies spécifiques à l'aide de QBF devrait faire avancer notre compréhension des jeux, ce qui est un enjeu important en informatique théorique. QBF pourrait donc servir à confirmer/infirmer l'existence de stratégies d'un type particulier, comme l'ont fait (62) (64) (65) avec des outils théoriques. Ce chapitre n'est donc qu'un premier pas dans cette direction et ce type de questions mériteraient de futurs développements.

CHAPITRE 6
SYMÉTRIE ET PLATEAUX TORIQUES

6.1 Introduction

Dans ce chapitre, nous allons présenter l'encodage COR++, une contribution de cette thèse qui est une légère modification de COR+, grâce à une innovation au niveau de la symétrie qui rend cet encodage plus performant sur les plateaux toriques. Nous n'allons pas présenter l'encodage au complet, mais seulement ce qui est différent de COR+. En fait, COR++ et COR+ ne diffèrent qu'au niveau du nombre de variables universelles, comme nous le verrons sous peu.

Le choix de COR+ au lieu de PAIRING ou de COVER a été fait en raison de la simplicité des changements puisque cette innovation est arrivée tard dans le développement des travaux de cette thèse et le temps manquait pour l'approfondir. Cependant, cette innovation pourrait tout aussi bien être intégrée à PAIRING et même à COVER si le *cover* était mis en place au troisième tour plutôt qu'au premier tour. Il reste qu'il faudrait faire une évaluation expérimentale dans les deux cas, ce qui est laissé à des travaux futurs.

Nous introduirons d'abord les changements au niveau de la symétrie, car c'est ce qui distingue COR+ de COR++, et puis nous présenterons nos résultats.

6.2 L'effet « tore »

À la section 2.1, nous avons introduit les plateaux en forme de tore et expliqué qu'ils ont les bordures connectées. Le haut du plateau est connecté avec le bas et la bordure gauche est connectée avec la droite. Ce que nous appelons l'effet « tore »est qu'une forme commencée sur le plateau à un endroit X peut se terminer de l'autre côté du plateau. Par exemple, une forme peut commencer au centre du plateau, puis évoluer vers la droite pour ensuite aboutir sur le côté gauche du plateau, car les bordures sont connectées.

Une autre façon de décrire cet effet est que les cellules en bordure de plateau ont plus de cellules voisines sur un plateau torique que sur un plateau normal. Ce concept sera important pour comprendre la symétrie sur des plateaux toriques de dimension paire, que nous verrons à la prochaine section. L'effet « tore »sera aussi important pour comprendre certains résultats du prochain chapitre.

6.3 La symétrie sur des plateaux toriques

Dans la sous-section 2.1.2, nous expliquons que la symétrie sur les plateaux normaux combine la symétrie par rotation et la symétrie par réflexion (figures 2.6, 2.7 et 2.8). Nous avons aussi vu qu'une seule cellule est équivalente à toutes les autres au premier tour de Noir pour les plateaux en forme de tore. Tous les encodages mentionnés jusqu'à présent (6) (14) (15), incluant les deux nôtres utilisent cette façon de faire.

Cependant, au niveau des plateaux en forme de tore, nous pouvons aller plus loin. Après la première cellule choisie par Noir au premier tour sur un plateau torique, nous pouvons pousser les mêmes concepts de symétrie par rotation et réflexion au deuxième tour pour Blanc, avec une nuance. La symétrie est désormais déterminée par la parité (paire ou impaire) des dimensions du plateau.

Puisque la symétrie sur un plateau de dimension impaire est plus facile à comprendre, nous débuterons par celle-ci. Comme toutes les cellules d'un plateau torique au premier tour sont équivalentes, on peut tenir pour acquis que le choix de Noir est la cellule centrale, telle que l'on peut le voir à la figure 6.1. La symétrie, par rotation et réflexion, peut s'appliquer au premier tour de Blanc comme dans le cas du premier tour de Noir sur un plateau normal. On peut se restreindre à un huitième du plateau, comme indiqué à la figure 6.1. Dans cette figure, chaque cellule vide sur le plateau torique au deuxième tour est équivalente à une des cellules avec un carré gris.

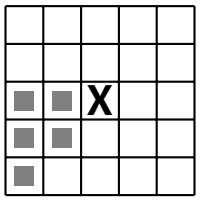


Figure 6.1 – Symétrie sur un plateau torique de dimension impaire

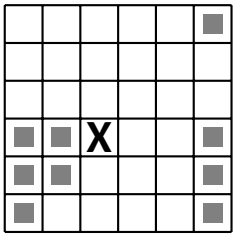


Figure 6.2 – Symétrie sur un plateau torique de dimension paire (1)

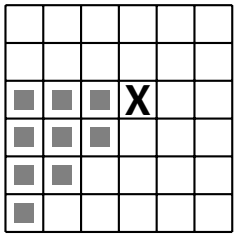


Figure 6.3 – Symétrie sur un plateau torique de dimension paire (2)

La symétrie sur un plateau de dimension paire est un peu plus difficile à conceptualiser et, pour bien la voir, il faut prendre en compte l'effet « tore ». Les plateaux des figures 6.2 et 6.3 sont équivalents et on voit que chaque cellule vide sur un des plateaux toriques au deuxième tour est équivalente à une des cellules avec un carré gris.

Pour mieux visualiser l'effet « tore »et la symétrie sur un plateau de dimension paire, on peut regarder les figures 6.4 et 6.5, où on voit que si l'on compte six cellules à partir de la cellule de Noir dans n'importe quelle direction, c'est-à-dire en haut, en bas, vers la droite, vers la gauche et aussi les deux diagonales, on arrive à la case départ. Il faut prendre note que la cellule **3** est la même pour les deux diagonales dans la figure 6.5.

Dans les figures 6.2 et 6.3, en comptant la distance en nombre de cellules des cellules vides par rapport à la cellule de Noir, on constate que toutes les cellules vides sont équivalentes à une des cellules avec un carré gris.

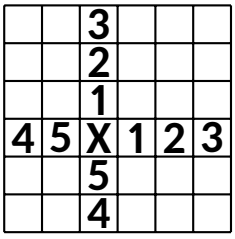


Figure 6.4 – La verticale et l'horizontale sur un plateau torique de dimension paire

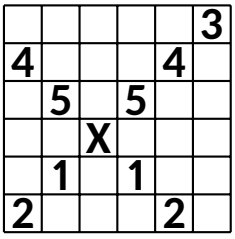


Figure 6.5 – Les deux diagonales sur un plateau torique de dimension paire

6.4 Encodage COR++

L'encodage COR++ reprend intégralement la structure de COR+ tel que présentée à la section 3.3. La seule distinction concerne les clauses 3.65, et ce uniquement lors du deuxième tour. Plutôt que de représenter l'ensemble des cellules x, y d'un plateau torique complet, nous ne considérons que les cellules pertinentes, telles que présentées aux figures 6.1 - 6.3.

6.5 Résultats expérimentaux

Étant donné que COR++ est une amélioration de l'encodage COR+ (15) sur les plateaux toriques seulement, nous comparons les deux approches sur de tels plateaux uniquement.

Nous évaluons notre encodage avec les solveurs QBF suivants, qui sont les mêmes qui ont servi à l'évaluation de PAIRING et COVER : DepQBF v6.03 (51), CAQE v4.0.1 (52), Qute v1.1 (53) et QESTO v1.0 (54).

Nous avons laissé tomber l'évaluation des préprocesseurs afin de gagner du temps, nous nous sommes concentrés sur les plateaux toriques de 3×3 et de 4×4 et nous avons utilisé les polyominos qui tiennent sur ces plateaux. Nous avons donc 24 instances sur un plateau torique de 3×3 et 49 instances sur un plateau torique de 4×4. Pour les plateaux de 5×5, nous avons utilisé 13 instances déjà connues pour être SAT afin de gagner du temps pour des raisons que nous expliquerons plus loin.

Comme précédemment, toutes les expériences sont réalisées sur un ordinateur Dell OptiPlex 7050, Intel Core i7-7700 Quad-Core à 3,6 GHz, avec 16 Go de RAM DDR4 à 2400 MHz. Chaque solveur a été utilisé avec les paramètres de ligne de commande par défaut.

Voici les résultats que nous avons obtenus.

Tableau 6.1 – Temps de résolution en secondes sur les plateaux tores de 3×3

Solveur	COR+	COR++
DepQBF	0.09	0.03
Caqe	0.78	0.46
Qute	2.5	1.16
Qesto	0.54	0.29

Nous montrons dans le tableau 6.1 les résultats en secondes pour chacun des encodages sur les plateaux toriques de 3×3. Il n'y a eu aucun délai d'expiration dans ce cas puisque les instances sont relativement simples et tous les solveurs ont donné les bons résultats, c'est-à-dire que les résultats SAT/UNSAT étaient corrects. Il y a exactement 4 gagnants et 20 perdants. Dans tous les cas, l'encodage COR++ a les meilleures performances, peu importe le solveur. Au niveau des solveurs, c'est DepQBF qui a les meilleures performances suivi de Qesto, Caqe et Qute, et ce, peu importe l'encodage.

Dans le tableau 6.2, nous avons séparé les temps d'exécution entre les instances ayant retourné SAT et celles ayant retourné UNSAT. Les solveurs ont été exécutés avec un délai d'expiration de 1000 secondes. La colonne TOTAL représente le total des colonnes SAT et UNSAT.

Tableau 6.2 – Temps de résolution en secondes sur les plateaux toriques de 4×4

	COR+			COR++		
Solveur	SAT	UNSAT	TOTAL	SAT	UNSAT	TOTAL
DepQBF	276	2534	2810	**121**	2524	2645
Caqe	405	2295	2700	**277**	3847	4124
Qesto	227	3631	3858	**64**	2963	3027

Nous avons séparé les résultats en deux catégories car, comme l'a remarqué (68) avec ses encodages *primal* et *dual*, il est souvent plus efficace d'avoir deux encodages, un visant les instances SAT et l'autre visant les instances UNSAT. L'usage des deux encodages simultanément, jusqu'à la première solution, est plus rapide que l'usage systématique d'un seul encodage. C'est d'ailleurs le principe que nous avons utilisé avec les encodages PAIRING et COVER aux chapitres 4 et 5.

On peut voir qu'avec COR++ les instances SAT, qui sont celles où Noir a une stratégie gagnante, sont résolues beaucoup plus rapidement avec les trois solveurs qu'avec COR+. On peut aussi voir que pour DepQBF et Qesto les instances UNSAT sont résolues légèrement plus rapidement avec COR++ qu'avec COR+. De son côté, le solveur Qute a rencontré un tel nombre d'erreurs et de délais d'expiration que nous l'avons exclu des résultats.

Tableau 6.3 – Temps de résolution en secondes sur les plateaux tores de 5×5

Solveur	COR+	COR++
DepQBF	1125	188
Caqe	1041	310
Qesto	862	126

Pour les plateaux de 5×5, nous avons utilisé les formes qui sont connues pour être SAT, c'est-à-dire qu'elles ont une stratégie gagnante pour Noir. Il s'agit de vérifier si, comme pour les plateaux de 4×4, COR++ est plus performant que COR+. On a utilisé les 8 formes connues pour être SAT sur un plateau torique de 5×5 et 5 formes qui sont des sous-créatures du Snaky, aussi connues pour être SAT sur un plateau torique de 5×5. Les sous-créatures du Snaky seront introduites au chapitre 7.

Le tableau 6.3, montre le gain de performance de COR++ comparativement à COR+. DepQBF et Qesto sont

toujours les plus performants. Qute n'a pas été utilisé vu les problèmes rencontrés avec les plateaux de 4×4.

Tableau 6.4 – Nombre de variables, clauses et quantificateurs sur les plateaux toriques pour le Domino

	COR+				COR++			
Plateaux	Var.	Univ.	Exist.	Clauses	Var.	Univ.	Exist.	Clauses
3×3	244	16	228	803	241	13	228	796
4×4	657	32	625	2354	656	31	625	2343
5×5	1501	60	1441	5952	1499	58	1441	5932

Le tableau 6.4 montre le nombre de variables, le nombre de variables universelles et existentielles et le nombre de clauses pour les deux encodages selon la taille du plateau. Dans ce cas-ci, les instances sont pour Domino, une forme simple à deux cellules connectées. Il est important de noter qu'il y a très peu de différences entre les deux encodages. Il n'y a que quelques variables universelles et quelques clauses en moins avec COR++ par rapport à COR+. Malgré tout, on peut quand même voir l'effet sur la performance.

6.6 Conclusion

Dans ce chapitre, nous avons introduit l'effet « tore »et expliqué ses particularités au niveau de la symétrie des plateaux toriques, où toutes les cellules sont équivalentes au premier tour pour Noir. Nous avons aussi démontré que la symétrie peut être étendue au deuxième tour pour Blanc et qu'elle dépend de la parité du plateau, c'est-à-dire si la taille du plateau est paire ou impaire. Il pourrait être intéressant d'envisager de futurs travaux pour approfondir cette notion et voir s'il n'y a pas d'autres situations où la symétrie pourrait être mise à bon escient.

Les résultats de ce chapitre montrent que la symétrie pour le deuxième coup est profitable pour les instances SAT. Il s'agit de résultats préliminaires qu'il serait intéressant de développer avec d'autres instances et des plateaux plus grands. Il serait aussi intéressant d'intégrer ce principe aux autres encodages comme COVER et PAIRING et de vérifier s'il y a, là aussi, un gain en performance.

CHAPITRE 7
ANALYSES DES FORMES

7.1 Introduction

Comme nous l'avons vu à la section 2.1, plusieurs recherches se sont intéressées à l'analyse de formes spécifiques. Harary avait énuméré plusieurs formes qu'il avait déterminées comme gagnantes, donnant, de plus, la taille du plateau et le nombre de coups nécessaires pour exécuter la stratégie gagnante. Mais d'autres recherches telles que (3) (13) (63) sont allées plus loin et décrivent en détail la stratégie gagnante pour réussir à compléter une forme spécifique ou même à en bloquer une en présentant une stratégie de pavage pour la contrer.

Le Snaky reste le grand mystère de ce jeu. Harary a fait la conjecture que le Snaky était gagnant sur un plateau normal de taille 15×15. Nous apprenons dans (13) que cette conjecture est basée sur le fait qu'un des collègues d'Harary battait tout le monde sur un plateau de cette taille lorsqu'il était le premier joueur. Il ne s'agit évidemment pas d'une justification suffisante, le problème reste entier et est toujours la grande question ouverte pour ce jeu.

De plus, les jeux combinatoires sont connus pour nécessiter des raisonnements complexes (9) (34) et en pratique le niveau de difficulté et le temps de résolution des solveurs QBF varient grandement d'une forme à l'autre.

Dans ce chapitre, nous allons présenter une analyse des données que nous avons recueillies sur les formes cibles du Tic-Tac-Toe d'Harary. Notre objectif est d'expliciter ce que nous savons au sujet des différentes formes, tant sur les plateaux normaux que toriques. Ces données ont été acquises tout au long de la thèse indépendamment des encodages ou des performances. Nous allons présenter les formes cibles, dont une différente catégorie de formes : les sous-créatures du Snaky, les résultats obtenus et les observations que nous en avons faites.

7.2 Les formes cibles

Dans cette section, nous aborderons les types de formes cibles et quelques-unes de leurs propriétés.

7.2.1 Les animaux

Au chapitre 2, nous avons vu que les formes utilisées dans le jeu du Tic-Tac-Toe d'Harary sont appelées « polyominos »et qu'elles sont formées d'un ensemble de cellules connectées par des arêtes. Une telle forme est aussi appelée « animal ». La raison est historique et brièvement expliquée dans (1). Dans ce même article (1), Harary a énuméré les formes connues comme étant gagnantes, les tailles minimales des plateaux pour les compléter et le nombre minimal de coups nécessaires pour réussir la stratégie gagnante. Nous avons corroboré certains de ces résultats et en avons invalidé d'autres, expliqués plus loin. Nous avons aussi analysé ces mêmes formes sur des plateaux toriques, ce qui constitue une contribution de cette thèse. Ces données sont présentées dans le tableau 7.1.

7.2.2 Les créatures

L'article (4) a introduit le concept de « créature »qui est, tout comme les animaux, un ensemble de cellules. Cependant, contrairement à celles des animaux, les cellules d'une créature ne sont pas toutes connectées. En fait, une créature peut être un ensemble quelconque de cellules. L'analyse des créatures en utilisant QBF constitue une autre contribution de cette thèse, car bien que les animaux aient été analysés dans la littérature, il n'y a que (4) qui ait traité des créatures, mais de façon théorique, sans considérer les encodages QBF.

7.2.3 Les sous-formes

Les sous-formes sont des formes (animaux ou créatures) qui peuvent être entièrement incluses à l'intérieur d'une autre forme. À ce moment, nous parlons de sous-animal lorsqu'il s'agit d'un animal ou de sous-créature lorsqu'il s'agit d'une créature entièrement contenue dans une autre forme. Les sous-formes partagent certaines propriétés avec les formes qui les incluent, comme nous le verrons plus loin.

7.2.4 Les variations des formes

Chaque forme a un nombre de variations possibles sur le plateau par rotation et réflexion de la forme. Certaines formes n'ont qu'une seule variation, telles que l'Élam et le Fatty. D'autres peuvent avoir jusqu'à huit variations, comme le Snaky et le FiveCellsZ. Le nombre de variations possibles d'une forme est important, car plus de variations donnent plus de chances à un joueur de pouvoir compléter une forme, autant pour

Noir que pour Blanc, puisqu'il y a une plus grande quantité de formes cibles possibles sur le plateau.

7.2.5 Les formes économiques

Une forme économique est une forme pour laquelle Noir a une stratégie gagnante qui se fait en nombre de coups égal au nombre de cellules contenues dans la forme. Par exemple, la forme Tic, qui contient trois cellules, est une forme économique puisque la stratégie gagnante pour la compléter se fait en trois coups de Noir. On peut aussi dire que la stratégie gagnante d'une forme économique est optimale, puisqu'elle s'effectue en un nombre minimal de coups pour réaliser la forme. Selon Harary, il n'y a que six formes économiques. Elles sont mentionnées dans le tableau 7.1. Nos résulats révèlent toutefois que ce n'est pas entièrement vrai.

7.2.6 L'effet « tore »

Nous avons présenté l'effet « tore »à la section 6.2. Nous reviendrons sur ce concept dans ce chapitre, car nous avons remarqué que cet effet est directement responsable de certaines stratégies gagnantes de certaines formes. Plus précisément, le fait que les cellules en bordure de plateau torique ont plus de cellules voisines contribue à la stratégie gagnante. Nous en discuterons plus loin dans les sections 7.3 et 7.4.

7.2.7 Propriétés des formes cibles

Toutes les formes, animaux ou créatures, partagent les mêmes propriétés suivantes :

Propriété 1 : Si une sous-forme $s \subset S$ est perdante par une stratégie de pavage sur un plateau de $n \times n$, alors S est perdante par la même stratégie de pavage sur un même plateau.

Le raisonnement est que si une sous-forme $s \subset S$ est perdante sur un plateau de $n \times n$ par une stratégie de pavage, la même stratégie de pavage peut être utilisée pour bloquer S puisque s est incluse dans S, alors S ne peut être gagnante.

La stratégie de pavage vue à la figure 2.3 en est un bon exemple. Toutes les formes cibles, animaux ou créatures, qui contiennent un Fatty peuvent être bloquées avec la même stratégie de blocage par pavage.

Propriété 2 : Si une forme S est gagnante sur un plateau de $n \times n$, $s \subset S$ n'est pas nécessairement gagnante aussi sur le même plateau.

La raison est qu'une stratégie gagnante G pour la forme S est adaptée pour S. Si l'on utilise la même stratégie pour une forme $s \subset S$, qui est moins complexe puisqu'elle est plus petite, il est possible que G crée une situation où Blanc peut passer à l'offensive et terminer la forme cible $s \subset S$ avant Noir, mais sans que Blanc ne réalise S. Nous verrons un exemple avec le Skinny et le FiveCellsY.

7.3 Observations sur les animaux

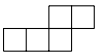
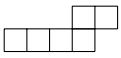
Dans le tableau 7.1, figurent les formes cibles connues pour être gagnantes selon Harary (1). Pour chaque forme, on donne le plateau et le nombre minimal de tours nécessaires à l'exécution de la stratégie gagnante. On utilise la nomenclature de NxN@k, qui signifie que sur un plateau de $N \times N$ il y a une stratégie gagnante pour Noir au tour k. Par exemple, le Domino a une stratégie gagnante avec 2x2@3, c'est-à-dire qu'il y a une stratégie gagnante pour Noir sur un plateau de 2×2 au tour 3. Harary a lui-même expliqué dans son article (1) qu'un jour ses conclusions devraient être revalidées par une recherche exhaustive. Il faut noter que les résultats d'Harary sont uniquement sur des plateaux normaux. Les colonnes Normal et Tore comprennent nos résultats accumulés tout au long de la thèse en utilisant les différents encodages, sauf pour la colonne Temps(N) et Temps(T) qui contiennent nos résultats faits avec COR++ à titre indicatif. Pour les temps de calcul, notre objectif est simplement de donner une mesure de la difficulté du problème et non le meilleur temps obtenu durant cette thèse. Les résultats sur les plateaux en forme de tore constituent une nouveauté, car personne ne les a analysés en détail auparavant.

Nous passerons en revue chacune des formes présentées dans le tableau 7.1 et partagerons nos observations.

Elam, Domino, Tic, El, Knobby, Elly

Les formes Elam, Domino, Tic, El, Knobby et Elly sont les plus simples puisqu'elles représentent toutes les animaux de trois cellules et moins et deux formes, le Knobby et Elly, de quatre cellules. Ce sont aussi les formes ayant été déterminées comme économiques par Harary. Nos expériences nous ont menés aux mêmes résultats que lui sur les plateaux normaux et nous avons établi les mêmes valeurs pour les plateaux toriques. Nous sommes aussi parvenus aux mêmes conclusions sur le fait qu'elles sont toutes économiques.

Tableau 7.1 – Les conclusions d'Harary et nos résultats

Nom	Forme	Harary	Économique	Normal	Temps(N)	Tore	Temps(T)
Elam		1x1@1	✓	1x1@1	< 1s	1x1@1	< 1s
Domino		2x2@3	✓	2x2@3	< 1s	2x2@3	< 1s
Tic		4x4@5	✓	4x4@5	< 1s	4x4@5	< 1s
El		3x3@5	✓	3x3@5	< 1s	3x3@5	< 1s
Knobby		5x5@7	✓	5x5@7	< 1s	4x4@7	< 1s
Elly		4x4@7	✓	4x4@7	< 2s	4x4@7	< 1s
Skinny		7x7@11	X	>7x7@11	> 12 j	>7x7@11	> 12j
Tippy		3x3@9 4x4@7	X	3x3@9 4x4@7	< 1s < 1s	3x3@7	< 1s
FiveCellsL		7x7@13	X	7x7@11	∼8 h	7x7@11	∼38 m
FiveCellsY		7x7@11	X	6x6@11	∼56m	6x6@11	∼32m
FiveCellsZ		6x6@11	X	6x6@11	∼1h	5x5@11	∼2m
Snaky		Inconnu	Inconnu	Inconnu	> 14 j	Inconnu	> 14 j

Skinny

Le Skinny est une forme extrêmement intéressante. Non seulement elle n'est pas gagnante au tour 11 sur un plateau de 7×7 normal tel que Harary l'avait prédit, mais elle n'est pas gagnante sur un plateau de 7×7 torique non plus. D'ailleurs, elle n'est pas gagnante avec 8x8@11 et 9x9@11, autant sur les plateaux normaux que toriques. Tous les encodages COR, COR+, COR++ et COVER ont donné le résultat UNSAT indiquant qu'il n'y a aucune stratégie pour Noir en seulement 11 tours.

L'intérêt de cette forme est qu'elle est une sous-forme du FiveCellsL et FiveCellsY et que ces deux formes sont gagnantes en 11 tours sur les deux types de plateaux, le 7×7 pour FiveCellsL et le 6×6 pour FiveCellsY. Cela démontre que les stratégies gagnantes pour FiveCellsL et FiveCellsY ne peuvent pas être appliquées au Skinny, justifiant ainsi la propriété 2 ci-dessus. Les raisons à ce sujet seront abordées dans la discussion à la section 7.5.

Nous savons qu'il existe une stratégie gagnante pour Skinny trouvée sur un forum internet (69). Néanmoins, elle ne semble confirmée par aucune autre source. Malheureusement, cette stratégie demande 15 tours et nos encodages, même modifiés pour être plus adaptés à cette stratégie et forcer Noir à jouer exactement selon la stratégie, n'ont pas réussi à la valider et confirmer qu'elle est belle et bien gagnante. Nous avons arrêté le solveur DepQBF après 12 jours sur cette question. On peut donc en conclure deux choses : soit cette stratégie est gagnante et nos encodages ne sont pas assez puissants pour la valider avec les solveurs actuels, soit il existe une erreur subtile dans cette stratégie et elle n'est pas gagnante. Dans tous les cas, des avancées algorithmiques permettront peut-être un jour de résoudre ce problème avec un encodage QBF.

Tippy

Le Tippy est une autre forme où nos résultats ont contredit Harary. Elle est effectivement gagnante avec 3x3@9 comme Harary l'avait prédit, mais sur un plateau de 4×4 la stratégie change et se fait en 7 tours. Tippy n'est pas économique sur un plateau de 3×3, mais elle le devient sur un plateau de 4×4. Alors ici nous apportons une nuance à la liste de formes économiques déterminée par Harary, car il n'a rien dit sur le fait que les formes peuvent changer de stratégie sur des plateaux de plus grande taille, et ainsi devenir économiques.

De plus, la forme est gagnante avec 3x3@7 sur un plateau torique. Ici, c'est clairement l'effet « tore »qui

entraîne la stratégie gagnante avec un coup de moins sur un plateau de 3×3 torique que sur un plateau de 3×3 normal. La raison est que les cellules en bordure de plateau ont plus de voisines ; c'est la seule différence entre les deux types de plateaux. La stratégie gagnante sur le plateau de 3×3 tore doit certainement utiliser ces cellules voisines.

FiveCellsL

Le FiveCellsL se fait en 7x7@11 tours au lieu de 7x7@13 comme Harary l'avait prédit. On a la même chose sur un plateau torique avec 7x7@11, on peut donc penser, sans toutefois l'affirmer avec certitude que la stratégie gagnante utilisée sur le plateau normal est également utilisée sur le plateau en forme de tore.

FiveCellsY

Le FiveCellsY se fait effectivement en 11 tours, mais sur un plateau de 6×6 au lieu d'un plateau de 7×7 comme Harary l'avait prédit. Il se fait aussi en 6x6@11 sur un plateau torique. Là encore, on peut penser qu'il s'agit de la même stratégie utilisée sur les deux types de plateaux.

FiveCellsZ

Pour le FiveCellsZ, Harary avait vu juste avec 6x6@11, mais on a découvert qu'il se faisait en 5x5@11 sur un plateau torique. L'effet « tore »contribue nettement à la stratégie gagnante sur un plateau de 5×5 torique puisque cette forme est perdante sur un plateau de 5×5 normal.

7.4 Les sous-créatures du Snaky

Pour faire avancer la question du Snaky, nous avons décidé de nous attaquer à ses sous-formes. La raison est que personne, y compris nous-mêmes, n'a réussi à déterminer si le Snaky est gagnant ou perdant. Comme tous les sous-animaux du Snaky ont été analysés par Harary (1) et d'autres chercheurs (3) (13) et qu'ils sont tous gagnants avec des stratégies gagnantes pour ces animaux déjà connues, nous avons décidé de nous attaquer aux sous-créatures du Snaky puisqu'elles n'ont jamais été analysées. Plus spécifiquement, puisque les deux plus grands sous-animaux du Snaky sont le FiveCellsL et le FiveCellsZ que nous savons que leurs sous-créatures sont gagnantes, nous avons choisi de nous attaquer aux sous-créatures du Snaky qui ne sont pas des sous-créatures de FiveCellsL ou de FiveCellsZ. Ces sous-créatures sont présentées dans le tableau

7.2.

Nous allons expliquer chacune de ces sous-créatures et faire part de nos observations.

2Cells

Le 2Cells est évidemment le plus simple et la stratégie est 5x5@3 sur les plateaux normaux et toriques. Il est aussi la sous-créature incluse dans toutes les autres sous-créatures du Snaky présentées dans le tableau 7.2. Le 2Cells représente les deux cellules du Snaky qui ne sont pas dans les sous-créatures de FiveCellsL et FiveCellsY. Aussi, comme le plateau est de 5×5 et que la longueur de la forme est de cinq cellules, il y a un fait intéressant selon lequel la stratégie gagnante doit absolument contenir des cellules sur deux bordures opposées du plateau.

3Cells1, 3Cells2, 3Cells3

Les 3Cells1, 3Cells2, 3Cells3 sont tous gagnants sur des 5x5@5. Ils sont tous économiques. Les stratégies gagnantes de ces formes ont, elles aussi, besoin d'avoir des cellules en deux bordures opposées du plateau.

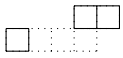
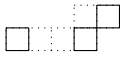
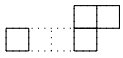
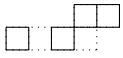
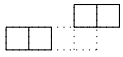
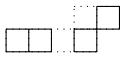
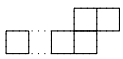
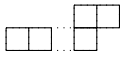
4Cells1

Pour le 4Cells1 et les autres sous-créatures de quatre cellules, la recherche des stratégies gagnantes commence à se complexifier, car les temps de calcul augmentent. Le 4Cells1 est un peu plus sophistiqué puisqu'il est gagnant en 5x5@11, 6x6@9 et 7x7@7. Autrement dit, plus il y a de l'espace et plus la forme devient économique. Cet effet est accentué sur les plateaux toriques puisqu'on obtient le même résultat que sur un plateau normal avec 5x5@11, mais la stratégie devient directement économique sur les 6x6@7 à cause de l'effet « tore ». On peut remarquer que le nombre de tours a une plus grande influence sur le temps que la taille du plateau, autant pour les plateaux normaux que les plateaux toriques.

4Cells2

Le 4Cells2 est gagnant en 6x6@11. Ce qui est intéressant avec cette forme, c'est qu'elle devient économique sur un plateau torique. La stratégie gagne quatre tours grâce à l'effet « tore ».

Tableau 7.2 – Les sous-créatures du Snaky

Nom	Sous-créature	Normal	Temps(N)	Tore	Temps(T)
2Cells		5x5@3	< 1s	5x5@3	< 1s
3Cells1		5x5@5	< 1s	5x5@5	< 1s
3Cells2		5x5@5	< 1s	5x5@5	< 1s
3Cells3		5x5@5	< 1s	5x5@5	< 1s
4Cells1		5x5@11 6x6@9 7x7@7	7m 10s 2s	5x5@11 6x6@7	16s 2s
4Cells2		6x6@11	36m	6x6@7	2s
4Cells3		****	> 3h	7x7@7 8x8@9	8s 224s
4Cells4		6x6@9 7x7@7	1m 14s	6x6@9 7x7@7	20s 6s
4Cells5		6x6@9 7x7@7	90s 11s	6x6@9 7x7@7	34s 5s
4Cells6		6x6@7	2s	6x6@7	3s
5Cells1		8x8@11 9x9@9	30h 31m	6x6@11 7x7@9	15m 159s
5Cells2		****	> 4h	8x8@11	30h
5Cells3		****	> 4h	****	> 4h
5Cells4		****	> 4h	****	> 4h

4Cells3

Le 4Cells3 n'est pas gagnant sur les plateaux de 6×6, 7×7, 8×8 et 9×9 avec k=11 sur un plateau normal. Il faut noter que PAIRING est UNSAT aussi à k=10 sur ces plateaux. Par conséquent, il y a toujours la possibilité que cette forme soit gagnante à un nombre de tours supérieur. Cette forme est spéciale puisqu'il n'y a que quatre variations sur le plateau au lieu de huit, comme toutes les autres sous-créatures du Snaky incluant le Snaky lui-même. Ce qui est intéressant, c'est qu'elle est gagnante sur un tore avec 7x7@7 et 8x8@9. Autrement dit, elle commence économique sur un plateau de 7×7 pour ne plus l'être sur un plateau de 8×8. Comme elle n'est pas gagnante sur un plateau normal au même tour que sur les plateaux toriques, cela veut dire que l'effet « tore »contribue directement à la stratégie gagnante et que sans lui, la stratégie gagnante n'a pas lieu.

4Cells4, 4Cells5

Les 4Cells4 et 4Cells5 sont gagnants avec 6x6@9 et 7x7@7, autant sur les plateaux normaux que toriques. Ici encore la forme devient économique avec un plateau plus grand.

4Cells6

Le 4Cells6 est la forme la plus simple de toutes les sous-créatures du Snaky à quatre cellules, car elle est économique avec 6x6@7. Elle a aussi un temps de calcul très court.

5Cells1

Le 5Cells1 est gagnant sur un plateau normal 8x8@11 et 9x9@9 et sur un plateau torique 6x6@11 et 7x7@9. Dans les deux cas, la stratégie gagne deux tours sur un plateau plus gros grâce à l'effet « tore ». Ici aussi, le temps de calcul diminue considérablement sur un plateau plus grand avec moins de tour, et ce, tant pour les plateaux normaux que toriques.

5Cells2

Le 5Cells2 est gagnant 8x8@11 torique et n'est pas gagnant avec 8x8@11 normal. Encore ici, la seule différence est l'effet « tore ».

5Cells3, 5Cells4

Les 5Cells3 et 5Cells4 ne sont pas gagnants sur les plateaux de 7×7, 8×8 et 9×9 avec k=11. Il faut noter que PAIRING est UNSAT avec k=10, laissant la possibilité que ces formes soient gagnantes à un nombre de tours supérieur.

7.5 Discussion

Harary avait vu juste sur les plus petites formes telles que Elam, Domino, Tic, Knobby et Elly, mais il s'était trompé sur les formes plus grosses comme Skinny, FiveCellsL et FiveCellsY. Bien entendu, les moyens technologiques en recherche exhaustive au début des années 1980 n'étaient pas ceux d'aujourd'hui. C'est pourquoi cela relève de l'exploit qu'il ait vu juste avec le FiveCellsZ, qui est une forme complexe.

La collecte des données présentées dans ce chapitre s'est faite tout au long de la thèse à l'aide des différents encodages que nous avons présentés. Néanmoins, les résultats des temps dans les tableaux 7.1 et 7.2 avec k > 11 proviennent tous de l'encodage COR++, introduit plus récemment. De plus, pour ces instances, seul DepQBF a été utilisé, puisque même s'il n'est pas le solveur le plus rapide, il est souvent le plus robuste. Par exemple, Qesto pour le Skinny avec k=11 termine abruptement avec un « out of memory error ». De son côté, DepQBF a tenté de résoudre le Skinny avec k=15 pendant 12 jours, sans succès.

Le contraste entre le Skinny et le FiveCellsL est intéressant. Il est particulièrement surprenant que le Skinny soit plus difficile à compléter que le FiveCellsL. Pourtant, le Skinny est une sous-forme du FiveCellsL et on s'attendrait à ce que sa résolution soit moins complexe. Aussi, les deux formes sont des sous-formes du Snaky, qui reste un mystère quant à l'existence d'une stratégie gagnante pour lui.

Nous avons identifié deux raisons qui pourraient expliquer pourquoi le Skinny est plus difficile à résoudre que le FiveCellsL. La première est que le Skinny n'a que deux variations, il y a donc moins de possibilités pour Noir de compléter cette forme et aussi plus de facilité pour Blanc de contrecarrer le Skinny sur le plateau que le FiveCellsL pour lequel il y a huit variations. Pour la deuxième raison, on peut considérer l'exemple de la figure 7.1 qui montre un début de partie pour le Skinny où Noir a déjà joué sur les cases marquées d'un X. Comme Noir possède ces deux cellules adjacentes, Blanc doit absolument réagir défensivement en bloquant l'un des deux côtés en jouant sur une des cases marquées d'un 1, sinon Noir gagne au tour suivant en jouant lui-même sur les cases 1. De façon similaire, si Blanc réussit à aligner deux cellules adjacentes,

c'est Noir qui se met sur la défensive. La stratégie gagnante pour le Skinny (69) proposée utilise aussi ce concept. D'un autre côté, pour le FiveCellsL, Blanc n'a pas à réagir aussi rapidement si Noir aligne deux cellules adjacentes. Nous pouvons penser que c'est la même chose pour le Snaky même si on ne connaît pas sa stratégie, s'il y en a une. Il s'agit bien sûr d'une analyse préliminaire qui gagnerait à être développée dans de futurs travaux. Nous en resterons néanmoins là dans le cadre de cette thèse.

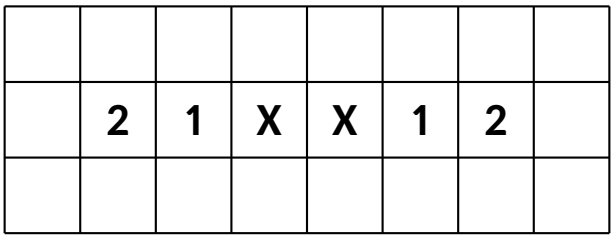


Figure 7.1 – Skinny partiel

Concernant les sous-créatures du Snaky du tableau 7.2, on a 4cells1, 4cells2, 4cells3, 5cells1 et 5cells2 qui sont plus faciles à compléter sur un tore que sur un plateau normal. On peut aussi inclure le Tippy dans cette liste, car lui aussi est plus facile à réaliser sur un tore que sur un plateau normal et il est également une sous-forme du Snaky. Cela fait donc six sous-formes du Snaky plus faciles à réaliser sur un tore que sur un plateau normal. Aussi, si l'on regarde le 5Cells1, la stratégie gagnante passe d'un plateau de 8×8 sur un plateau normal à un 6×6 sur un plateau torique pour le même nombre de tours. Nous avons découvert que le 5Cells1 était gagnant avec COVER et DepQBF, son temps de calcul était près de quatre heures pour le plateau de 8×8 normal et moins d'une heure pour le plateau de 6×6 torique. On peut donc en conclure que l'analyse du Snaky sur un plateau torique est d'un grand intérêt et devrait retenir l'attention de la communauté des jeux/QBF.

Dans le lot des sous-créatures à quatre cellules, la forme 4Cells3 mérite une attention spéciale, car elle est la seule à avoir quatre variations au lieu de huit comme les autres. On ne sait pas si elle est gagnante ou perdante sur un plateau normal jusqu'à 9x9 k=11. On peut se demander si le fait d'avoir moins de variations rend nécessairement la stratégie gagnante plus complexe, comme pour le Skinny. Il serait aussi d'intérêt de faire avancer la connaissance sur la forme 4Cells3.

De leur côté, les formes 5Cells3 et 5Cells4 mériteraient d'être analysées sur des plateaux toriques puisque

les deux autres sous-créatures du Snaky à cinq cellules, le 5Cells1 et le 5Cells2, sont plus faciles sur ce type de plateau. Il serait possible que ce soit également le cas pour 5Cells3 et 5Cells4. Cependant, nous n'avons trouvé aucune solution pour ces deux formes, quels que soient le solveur et l'encodage à notre disposition. Nous avons fait les tests avec 15 000 secondes ($\sim$4 heures) de délai d'expiration. C'est pourquoi le 5Cells3 et le 5Cells4 seraient deux autres formes intéressantes à analyser pour la communauté des jeux/QBF puisque des avancées sont nécessaires à leur résolution.

7.6 Conclusion

Dans ce chapitre, nous avons présenté quelques propriétés des formes, animaux, créatures et sous-formes. Nous avons aussi fait une comparaison des résultats d'Harary (1) avec les nôtres et corrigé quelques erreurs de son article. Nous avons partagé nos observations sur les formes gagnantes. Nous avons finalement abordé les sous-créatures du Snaky et dévoilé nos observations sur elles.

Nous avons terminé sur une discussion qui visait différentes formes, telles que le Skinny et le FiveCellsL, qui sont tous les deux des sous-formes du Snaky. Aussi, nous portons attention sur le fait que beaucoup de sous-créatures, comme les 4cells1, 4cells2, 4cells3, 5cells1 et 5cells2 ainsi que le Tippy, qui est aussi une sous-forme du Snaky, bénéficient de l'effet « tore »et que le Snaky aurait tout intérêt à être analysé sur un plateau torique par la communauté des jeux/QBF.

CONCLUSION

Dans cette thèse, nous avons démontré que les jeux combinatoires, tels que le Tic-Tac-Toe d'Harary, constituent un terrain d'expérimentation intéressant pour la recherche en résolution QBF. En développant de nouveaux encodages adaptés aux *achievement games* et en utilisant des solveurs QBF récents, nous avons contribué à l'amélioration de la performance de résolution de problèmes QBF relevant de la classe PSPACE-complet. Nos travaux répondent également à certaines critiques formulées dans la littérature (68) (66) (70), notamment en ce qui concerne la difficulté croissante liée à l'ajout de niveaux de quantification dans les formules QBF. Les résultats obtenus ouvrent la voie à de futures explorations, tant sur le plan des encodages que sur celui de l'analyse stratégique des jeux encore non résolus, comme dans le cas du Snaky.

Dans cette section, nous résumerons les principales contributions, présenterons les perspectives de recherche et conclurons par un mot de la fin.

8.1 Contributions

Voici les contributions que nous avons réalisées au cours de cette thèse.

L'encodage PAIRING a été notre première contribution. Cet encodage est le premier à utiliser la perspective du deuxième joueur dans le jeu du Tic-Tac-Toe d'Harary et des stratégies de pavage avec QBF dans le but de trouver des stratégies de blocage. Nous avons démontré que cet encodage permet une résolution efficace pour le jeu du Tic-Tac-Toe sur des plateaux de 3×3, de 4×4 ainsi que pour la plupart des instances 5×5.

L'utilisation de deux encodages, l'un avec la perspective du premier joueur et l'autre avec la perspective du deuxième joueur, en profondeur itérative (*iterative deepening*) constitue une méthode de recherche QBF qui emploie la force des deux encodages et représente également une avancée. Elle étend l'approche reposant sur la profondeur itérative (*iterative deepening*) avec un seul encodage. Avec notre méthode, les deux encodages sont essentiels pour que la méthode fonctionne et elle a permis la résolution des plateaux de 5×5.

Nous avons présenté ces deux contributions à la conférence de KI2021 - 44th German Conference on Artificial Intelligence et nous avons eu ainsi notre première publication (35).

La recherche de types de stratégies spécifiques avec l'encodage COVER est aussi une contribution de cette thèse. L'idée de restreindre le choix de Noir et de Blanc durant la partie et vérifier, en fin de partie, si Blanc aurait pu battre Noir en l'absence de restrictions constitue une nouveauté. De manière plus fondamentale, cette méthode permet de raisonner sur le jeu et d'en extraire de nouvelles connaissances, puisque nous montrons non seulement qu'il y a une stratégie gagnante, mais aussi qu'elle appartient à un type particulier, où les mouvements de Noir sont concentrés sur certaines cellules. Nous avons présenté l'encodage COVER à la conférence Canadian AI 2025, et cela a donné lieu à notre deuxième publication (36).

Une autre avancée a été la symétrie des plateaux toriques pour le premier tour de Blanc avec l'encodage de COR++. Cela était un ajustement de l'encodage de COR+, mais a permis un gain de performance significatif au niveau des plateaux tores.

Pour ce qui est de l'analyse des formes, les contributions sont multiples. La première est la confirmation ou la réfutation des résultats d'Harary dans (1) comme suit :

a) Le Skinny n'est pas gagnant au tour 11.

b) Le Tippy n'est effectivement pas économique sur un plateau de 3×3, mais il le devient sur un plateau de 4×4.

c) Le FiveCellsL est gagnant sur un 7x7@11 au lieu de 7x7@13.

d) Le FiveCellsY est gagnant sur un 6x6@11 au lieu de 7x7@11.

De plus, tous les résultats sur les plateaux toriques ainsi que toutes les analyses des sous-créatures du Snaky constituent une nouveauté.

Une dernière contribution, d'un point de vue plus fondamental, est la constatation que plusieurs formes ont des stratégies gagnantes qui changent sur des plateaux de tailles différentes. Le Tippy et certaines sous-créatures du Snaky sont de bons exemples. On peut voir que ces formes ont des stratégies gagnantes qui changent avec la taille du plateau et certaines d'entre elles deviendront même économiques.

8.1.1 Perspectives de recherche

Nos avancées ouvrent la porte à plusieurs avenues de recherche. Au niveau QBF, utiliser deux encodages, chacun avec la perspective d'un joueur, est selon nous une approche très prometteuse. Elle ouvre la voie à la spécialisation des encodages, qui rendrait les encodages très efficaces dans les cas où les instances sont SAT selon la perspective d'un joueur, mais pas nécessairement lorsqu'ils sont UNSAT. Alors, avec deux encodages opposés, chacun spécialisé selon la perspective d'un joueur, nous exploitons la force de chacun des encodages.

Pour PAIRING, cela voudrait dire de pousser un peu plus le concept de stratégie de blocage en incluant d'autres techniques utilisées dans l'état de l'art pour prouver qu'une forme est perdante. L'idée serait de se rendre compte que la forme est perdante le plus tôt possible dans la résolution d'une instance.

Pour COVER, cela pourrait être de rendre le *cover* plus dynamique, l'adapter au fur et à mesure que la partie avance et ainsi éliminer plus rapidement les cellules inutiles dans une configuration donnée. Ici aussi, l'idée serait de découvrir le plus tôt possible qu'une forme a un potentiel ou non dans une configuration donnée.

La symétrie est aussi une approche qui mériterait d'être analysée plus à fond. Avec COR++, nous avons vu que plusieurs cellules sont équivalentes entre elles au deuxième tour sur les plateaux tores. Est-ce possible d'avoir quelque chose de similaire au troisième tour ou même à des tours subséquents ? Il serait intéressant d'explorer de telles possibilités.

Au niveau de l'analyse des formes, l'analyse du Snaky sur les plateaux tores est, à notre avis, une piste intéressante à explorer. Aussi, dans les sous-formes du Snaky, le 4Cells3 ainsi que toutes les sous-créatures à cinq cellules représentent les cas les plus intéressants à investiguer, et les résoudre pourrait faire avancer la question du Snaky. Le 4Cells3 n'a que quatre cellules et quatre variations sur le plateau et, pourtant, nous n'avons aucune solution connue. Les 5Cells2, 5Cells3 et 5Cells4 n'ont aucune solution connue sur des plateaux normaux et représentent des cas difficiles.

8.1.2 Mot de la fin

Comme il est mentionné dans (13), les jeux combinatoires représentent un défi humiliant pour la communauté des jeux. La raison est tout simplement que, malgré la simplicité du jeu, personne n'a trouvé de

manière de contrôler l'explosion combinatoire qui lui est associée. Le Tic-Tac-Toe d'Harary ne fait pas exception et nous croyons que nous n'avons pas fini d'en découvrir à ce sujet. En espérant que nos contributions puissent aider la communauté scientifique à en découvrir plus.

BIBLIOGRAPHIE

(1) F. Harary, “Achieving the Skinny Animal,” Eureka, vol. 42, pp. 8–14, 1982.

(2) F. Harary, “Achievement and avoidance games designed from theorems,” Rendiconti del Seminario Matematico e Fisico di Milano, vol. 51, no. 1, pp. 163–172, 1981.

(3) J. P. Bode and H. Harborth, “Hexagonal polyomino achievement,” Discrete Mathematics, vol. 212, no. 1-2, pp. 5–18, 2000.

(4) K. Suetsugu, “Achievement games on a one-dimensional board,” Journal of Information Processing, vol. 25, pp. 678–681, 2017.

(5) I. Halupczok and J. C. Schlage-Puchta, “Some strategies for higher dimensional animal achievement games,” Discrete Mathematics, vol. 308, no. 16, pp. 3470–3478, 2008.

(6) Diptarama, R. Yoshinaka, and A. Shinohara, “QBF encoding of generalized tic-Tac-Toe,” in Quantified Boolean Formulas, QBF 2016. CEUR Workshop Proceedings, vol. 1719, pp. 14–26, 2016.

(7) P. McBurney and S. Parsons, “Games that agents play : A formal framework for dialogues between autonomous agents,” Journal of Logic, Language and Information, vol. 11, no. 3, pp. 315–334, 2002.

(8) A. S. Fraenkel, “Combinatorial games : Selected bibliography with a succinct gourmetintroduction,” Electronic Journal of Combinatorics, vol. 1, no. DynamicSurveys, pp. 1–109, 2018.

(9) E. D. Demaine, “Playing games with algorithms : Algorithmic combinatorial game theory,” in Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science, vol. 2136 LNCS, pp. 18–33, 2001.

(10) L. V. Allis, H. J. van den Herik, and M. P. Huntjens, “Go-Moku solved by new search techniques,” Computational Intelligence, vol. 12, no. 1, pp. 7–23, 1996.

(11) D. Tosi and M. Scalise, “Understanding Artificial Intelligence in Chess : The RubiChess Case Study,” in Communications in Computer and Information Science, vol. 2492 CCIS, pp. 22–34, 2025.

(12) S. S. Inampudi,
Enhancing Hex Strategy : AI Based Two-Distance Pruning Approach with Pattern-Enhanced Alpha-Beta Search, vol. 2053 CCIS. Springer Nature Switzerland, 2024.

(13) József Beck, “Combinatorial games : Tic-Tac-Toe theory,” Cambridge University Press, vol. 46, no. 01, pp. 46–0343–46–0343, 2008.

(14) V. Mayer-Eichberger and A. Saffidine, “Positional Games and QBF : The Corrective Encoding,” in Theory and Applications of Satisability Testing, SAT 2020., vol. 12178 LNCS, pp. 447–463, Springer International Publishing, 2020.

(15) V. Mayer-Eichberger and A. Saffidine, “Positional Games and QBF : A Polished Encoding,” arXiv 2005.05098 [cs.LO], 2023.

(16) S. A. Cook, “The complexity of theorem-proving procedures,” Proceedings of the 3rd IEEE Symposium on the Theory of Computation, pp. 151–158, 1971.

(17) K. L. McMillan, “Symbolic Model Checking,” Verification of Digital and Hybrid Systems, pp. 117–137, 2000.

(18) H. A. Kautz and B. Selman, “Planning as Satisfiability,” in 10th European Conference on Artificial Intelligence, ECAI 92, Vienna, Austria, August 3-7, 1992. Proceedings (B. Neumann, ed.), pp. 359–363, John Wiley and Sons, 1992.

(19) R. Bloem, U. Egly, P. Klampfl, R. Könighofer, and F. Lonsing, “SAT-based methods for circuit synthesis,” 2014 Formal Methods in Computer-Aided Design, FMCAD 2014, no. 317753, pp. 31–34, 2014.

(20) A. Biere, M. Heule, H. van Maaren, and T. Walsh, Handbook of satisfiability : Second edition, vol. 185 of Frontiers in Artificial Intelligence and Applications. IOS Press, 2021.

(21) M. Davis and H. Putnam, “A Computing Procedure for Quantification Theory,” Journal of the ACM, vol. 7, no. 3, pp. 201–215, 1960.

(22) M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, “Chaff : engineering an efficient SAT solver,” Proceedings of the 38th Design Automation Conference (DAC 2001), pp. 530–535, 2001.

(23) S. O. committee, “The international sat competition web page.” `https://satcompetition.github.io/`. Accessed : 2025-07-12.

(24) M. J. Heule and S. Szeider, “A SAT approach to clique-width,” International Conference on Theory and Applications of Satisfiability Testing, vol. 7962 LNCS, pp. 318–334, 2013.

(25) L. J. Stockmeyer and A. R. Meyer, “Word problems requiring exponential time : Preliminary report,” Proceedings of the Annual ACM Symposium on Theory of Computing, pp. 1–9, 1973.

(26) O. Arieli and M. W. Caminada, “A QBF-based formalization of abstract argumentation semantics,” Journal of Applied Logic, vol. 11, no. 2, pp. 229–252, 2013.

(27) I. P. Gent and A. G. D. Rowley, “Encoding Connect-4 using Quantified Boolean Formulae,” Modelling and Reformulating Constraint Satisfaction Problems, pp. 78–93, 2003.

(28) T. Jussila and A. Biere, “Compressing BMC Encodings with QBF,” Electronic Notes in Theoretical Computer Science, vol. 174, no. 3, pp. 45–56, 2007.

(29) S. Reisch, “Gobang ist PSPACE-vollständig,” Acta Informatica, vol. 13, no. 1, pp. 59–66, 1980.

(30) T. Furtak, M. Kiyomi, T. Uno, and M. Buro, “Generalized Amazons is PSPACE-complete,” in IJCAI International Joint Conference on Artificial Intelligence, pp. 132–137, 2005.

(31) S. Iwata and T. Kasai, “The Othello game on an n × n board is PSPACE-complete,” Theoretical Computer Science, vol. 123, no. 2, pp. 329–340, 1994.

(32) E. D. Demaine and Y. Diomidov, “Strings-And-Coins and Nimstring are PSPACE-complete,” De Gruyter Proceedings in Mathematics, pp. 108–120, 2022.

(33) S. Reisch, “Hex is PSPACE-complete,” Acta Informatica, vol. 15, no. 2, pp. 167–191, 1981.

(34) A. Shukla, A. Biere, L. Pulina, and M. Seidl, “A survey on applications of quantified boolean formulas,” Tools with Artificial Intelligence, ICTAI, vol. 2019-Novem, pp. 78–84, 2019.

(35) S. Boucher and R. Villemaire, “Quantified Boolean Solving for Achievement Games,” in German Conference on AI, KI 2021, vol. 12873 LNAI of Lecture Notes in Computer Science, pp. 30–43, Springer International Publishing, 2021.

(36) S. Boucher and R. Villemaire, “The QBF Cover encoding for Harary Tic-Tac-Toe’s,” Canadian Conference on Artificial Intelligence, may 19 2025. https ://caiac.pubpub.org/pub/km2g3t1j.

(37) J. Marques-Silva, “Practical applications of boolean satisfiability,” Proceedings - 9th International Workshop on Discrete Event Systems, WODES’ 08, pp. 74–80, 2008.

(38) A. Robinson, “A Machine-Oriented Logic Based on the Resolution Principle,” Journal of the Association for Computing Machinery, vol. 12, no. 1, pp. 23–41, 1965.

(39) M. Davis, G. Logemann, and D. Loveland, “A machine program for theorem-proving,” Communications of the ACM, vol. 5, no. 7, pp. 394–397, 1962.

(40) J. Marques Silva, K. Sakallah, J. P. Marques-Silva, and K. Sakallah, “GRASP - A New Search Algorithm for Satisfiability,” Computer Aided Design, pp. 220–227, 1996.

(41) E. Giunchiglia, M. Narizzano, and A. Tacchella, “Clause/term resolution and learning in the evaluation of quantified boolean formulas,” Journal of Artificial Intelligence Research, vol. 26, pp. 371–416, 2006.

(42) L. Zhang and S. Malik, “Conflict driven learning in a quantified Boolean satisfiability solver,” in IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, pp. 442–449, 2002.

(43) A. Biere, “Resolve and expand,” in Theory and Applications of Satisfiability Testing. SAT 2004, vol. 3542 LNCS, pp. 59–70, 2005.

(44) E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, “Counterexample-guided abstraction refinement for symbolic model checking,” Journal of the ACM, vol. 50, no. 5, pp. 752–794, 2003.

(45) M. Janota, W. Klieber, J. Marques-Silva, and E. Clarke, “Solving QBF with counterexample guided refinement,” Artificial Intelligence, vol. 234, pp. 1–25, 2016.

(46) M. Samer and S. Szeider, “Backdoor sets of quantified boolean formulas,” Journal of Automated Reasoning, vol. 42, no. 1, pp. 77–97, 2009.

(47) F. Lonsing and A. Biere, “Integrating dependency schemes in search-based QBF solvers,” in Theory and Applications of Satisfiability Testing – SAT 2010, vol. 6175 LNCS, pp. 158–171, 2010.

(48) H. K. Büning, M. Karpinski, and A. Flögel, “Resolution for quantified boolean formulas,” Information and Computation, vol. 117, no. 1, pp. 12–18, 1995.

(49) A. Van Gelder, “Contributions to the theory of practical quantified boolean formula solving,” in Principles and Practice of Constraint Programming. CP 2012, vol. 7514 LNCS, pp. 647–663, 2012.

(50) V. Balabanov, M. Widl, and J. H. R. Jiang, “QBF resolution systems and their proof complexities,” in Theory and Applications of Satisfiability Testing – SAT 2014, vol. 8561 LNCS, pp. 154–169, 2014.

(51) F. Lonsing and U. Egly, “DepQBF 6.0 : A search-based QBF solver beyond traditional QCDCL,” in Automated Deduction - CADE 26., vol. 10395 LNAI, pp. 371–384, 2017.

(52) L. Tentrup, “CAQE and QuAbS : Abstraction Based QBF Solvers,” Journal on Satisfiability, Boolean Modeling and Computation, vol. 11, no. 1, pp. 155–210, 2019.

(53) T. Peitl, F. Slivovsky, and S. Szeider, “Qute in the QBF Evaluation 2018,” Journal on Satisfiability, Boolean Modeling and Computation, vol. 11, no. 1, pp. 261–272, 2019.

(54) M. Janota and J. Marques-Silva, “Solving QBF by clause selection,” in IJCAI International Joint Conference on Artificial Intelligence, vol. 2015-Janua, pp. 325–331, 2015.

(55) A. Biere, F. Lonsing, and M. Seidl, “Blocked clause elimination for QBF,” Automated Deduction, vol. 6803 LNAI, pp. 101–115, 2011.

(56) R. Wimmer, S. Reimer, P. Marin, and B. Becker, “HQSpre – an effective preprocessor for QBF and DQBF,” in Tools and Algorithms for the Construction and Analysis of Systems, vol. 10205 LNCS, pp. 373–390, 2017.

(57) F. Lonsing and U. Egly, “QRATPre+ : Effective QBF Preprocessing via Strong Redundancy Properties,” in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 11628 LNCS, pp. 203–210, Springer International Publishing, 2019.

(58) E. Wynn, “A comparison of encodings for cardinality constraints in a SAT solver,” 2018.

(59) O. Bailleux and Y. Boufkhad, “Efficient CNF encoding of Boolean cardinality constraints,” Principles and Practice of Constraint Programming, vol. 2833 LNCS, pp. 108–122, 2003.

(60) C. Sinz, “Towards an optimal CNF encoding of boolean cardinality constraints,” Principles and Practice of Constraint Programming, vol. 3709 LNCS, pp. 827–831, 2005.

(61) I. P. Gent and P. Nightingale, “A New Encoding of AllDifferent into SAT,” in Third International Workshop on CP 2004 Workshop on Modelling and Reformulating CSPs - CP2004, pp. 95–110, 2004.

(62) H. Ito and H. Miyagawa, “Snaky is a winner with one handicap,” HERMIS-$\mu\pi$. Hellenic European Research on Mathematics and Informatics Science, no. 3, pp. 1–8, 2009.

(63) I. Halupczok and J.-C. Schlage-Puchta, “Achieving Snaky,” Integers, vol. 7, no. 1, pp. 1–28, 2007.

(64) H. Harborth and M. Seemann, “Snaky is an edge-to-edge loser,” Geombinatorics, no. 5, pp. 132–136, 1996.

(65) H. Harborth and M. Seemann, “Snaky is a paving winner,” Bull. Inst. Combin. Appl., no. 19, pp. 71–78, 1997.

(66) C. Ansotegui, C. P. Gomes, and B. Selman, “The Achilles’ Heel of QBF,” in 20th national conference on Artificial intelligence - AAAI’05, pp. 275–281, 2005.

(67) I. Shaik, V. Mayer-Eichberger, J. van de Pol, and A. Saffidine, “Implicit State and Goals in QBF Encodings for Positional Games (extended version),” 2023.

(68) A. Van Gelder, “Primal and dual encoding from applications into quantified boolean formulas,” Principles and Practice of Constraint Programming, vol. 8124 LNCS, pp. 694–707, 2013.

(69) D. Mathias, G. Martin, and M. Earnest, “How to find winning Strategy for 4 celled animals of Harary’s generalized tic tac toe.” `https://math.stackexchange.com/questions/4355825/how-to-find-winning-strategy-for-4-celled-animals-of-hararys-generalized-tic-ta`, 2022. Consulté le 2025-08-07.

(70) F. Lonsing and U. Egly, “Evaluating QBF solvers : Quantifier alternations matter,” in Principles and Practice of Constraint Programming, vol. 11008 LNCS, pp. 276–294, Springer International Publishing, 2018.