If we did all the things we are capable of, we would literally astound ourselves.

– Thomas A. Edison

University of Alberta

PLAYING AND SOLVING THE GAME OF HEX

by

Philip Thomas Henderson

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

Fall 2010
Edmonton, Alberta

Examining Committee

Ryan Hayward, Computing Science

Richard Nowakowski, Mathematics and Statistics, Dalhousie University

Mazi Shirvani, Mathematical and Statistical Sciences

Joseph Culberson, Computing Science

Martin Müller, Computing Science

To my family and friends,

Eccentricities notwithstanding.

Abstract

The game of Hex is of interest to the mathematics, algorithms, and artificial intelligence communities. It is a classical PSPACE-complete problem, and its invention is intrinsically tied to the Four Colour Theorem and the well-known strategy-stealing argument. Nash, Shannon, Tarjan, and Berge are among the mathematicians who have researched and published about this game.

In this thesis we expand on previous research, further developing the mathematical theory and algorithmic techniques relating to Hex. In particular, we identify new classes of moves that can be pruned from consideration, and devise new algorithms to identify connection strategies efficiently.

As a result of these theoretical improvements, we produce an automated solver capable of solving all 8×8 Hex openings and most 9×9 Hex openings; this marks the first time that computers have solved all Hex openings solved by humans. We also produce the two strongest automated Hex players in the world — Wolve and MoHex — and obtain both the gold and silver medals in the 2008 and 2009 International Computer Olympiads.

Acknowledgements

There are many people who assisted me throughout the creation of this research.

First and foremost is my supervisor, Ryan Hayward. His enthusiasm for Hex is quite contagious, and the questions he posed presented interesting challenges. His attention to written and visual presentation has improved my writing greatly, especially in terms of clarity and precision. Although we often had different ideas on how best to proceed, our discussions forced me to revise my preconceptions, and as a result I explored important techniques that I would have otherwise neglected.

I am also greatly indebted to Broderick Arneson. The Hex code was in disastrous shape when I first arrived, rendering the addition of any new functionality nearly impossible. Broderick's complete rewrite and ongoing optimizations of the code allowed me to concentrate my efforts on the research of new Hex theory and algorithms. Without a doubt, the efficiency of this new code enabled our results in terms of solving larger boards and producing stronger automated players. Lastly, the visualization and information tools he created helped greatly in terms of analyzing current performance and inspiring new developments.

Martin Müller was also of great help, being my principal teacher of combinatorial game theory, and assisting in my understanding of other key algorithms such as proof number search and Monte Carlo tree search.

Prior Hex researchers at the University of Alberta aided my understanding of existing Hex theory and algorithms, particularly Michael Johanson regarding H-search and Jack van Rijswijck regarding inferior cell analysis. I also had useful discussions with Morgan Kan, Markus Enzenberger, David Silver, Sylvain Gelly, Michael Buro, Darse Billings, and Zachary Friggstad.

My research was also assisted by several summer student researchers. The main contributors (in chronological order) are Geoff Ryan (connection template construction and player testing), Robert Budac (inferior cell patterns and tool creation), Laurie Charpentier (inferior cell identification algorithms), Andrea Buchfink (ladder template construction), Teri Drummond (ladder template construction), Matthew Delaney (handicap player construction), and Yuri Delanghe (evaluation function generation).

Lastly, I thank NSERC, Alberta Ingenuity, iCORE, and the University of Alberta for financial support of this research.

Thank you all.

Table of Contents

List of Tables

List of Figures

Nomenclature

For formal definitions, see the specified section.

Term	Section	Explanation
4-3-2	2.9.2	A common second player connection strategy.
AND rule	2.9.2	A deduction rule in the H-search algorithm that combines connection strategies in series.
backing up	2.9.2	Deducing connection strategies or inferior cells in previous states based on properties observed in successor states.
board size	2.2	The number of cells.
border	2.2	A coloured side of the Hex board.
border bridge	2.9.2	A bridge that has a border as one endpoint.
border template	2.9.2	A frequently occurring connection strategy that has a border as one endpoint.
braids	4.1	First player connection strategies of a particular form not found by H-search.
bridge	2.9.2	A common second player connection strategy with a carrier of size two.
bypass	2.5	Replacing a reversible move with all of the legal moves available in the state reached via the reversible-reverser move exchange.
capture-domination	2.9.1	Domination deduced using captured sets and monotonicity.
captured	2.9.1	A set of cells for which there exists a second player strategy to render all opponent moves dead.
captured-reversible	3.3	Reversible cell where an opponent move causes the cell to be player-captured.
captured-reversible graph	3.3	Graph modelling the interference relationship between captured-reversible cells.

carrier	2.9.2, 3, 5.3	Set of uncoloured cells required for a strategy (*e.g.*, connection strategy, fillin strategy).
cell	2.2	A hexagonal location which the players can colour during the course of a game.
chain	2.2	A maximal set of connected locations.
chain boundary	3.7.1	Set of chains adjacent to a region that are not internal.
chain component graph	3.7.1	Bipartite graph indicating adjacencies between uncoloured components and chains.
chain decomposition	3.7.1	Two-tuple of a region and its chain boundary.
chain deleted Hex graph	3.7.1	The graph obtained by deleting all vertices corresponding to coloured locations.
child limit	5.2.3	The number of live children considered in focused depth-first proof number search.
closed neighbourhood	3.4	Neighbourhood of a location unioned with the location itself.
completion	2.2	A continuation with no uncoloured cells.
completion Hex	3.5	Variant of Hex where the game is played until no uncoloured cells remain.
connected locations	2.2	Locations that are the endpoints of some coloured monochromatic path.
continuation	2.2	A Hex position where coloured cells have only been added.
dead	2.9.1	A cell that is not live.
dead-reversible	3.2	Synonym for vulnerable.
dimension	2.2	The length of a (regular) Hex board's side.
disproof set	5.2	A set of leaves in a proof number tree that are sufficient to prove that the player to move loses the root position.
dominated	2.5	A move which results in a (weakly) worse position than another available move.
double chain adjacent	3.7.2	The relationship between two uncoloured cells that are common neighbours of some Black chain and some White chain.
Elo	6	A rating system that indicates the relative strength and expected win percentage between players.

endpoint	2.2, 2.9.2	The first or last location in a path's sequence, or one of the two locations being connected via a connection strategy.
exchange tree	3.5	Derivation of a completion Hex strategy tree where two cells' roles are interchanged.
external	2.9.2	An opponent move outside a player's connection strategy carrier.
fillin	2.9.1	Colouring a set of cells in a position without altering its value.
fillin carrier	5.3.1	Set of cells required to maintain a fillin reduction.
fillin-domination	3.1	Domination deduced using fillin and monotonicity.
four-sided decomposition	3.8.4	Chain decomposition whose boundary forms a four cycle of touching chains.
generalized H-search	2.9.2	Complete version of H-search.
Generalized Hex	2.3	A game played on graphs, where players alternate turns performing vertex simplicialization and vertex deletion, and the players' goals are to connect/disconnect two marked vertices.
graph neighbour domination	3.4	Domination deduced by graph neighbourhood sets.
graph neighbourhood	3.4	The neighbourhood of an uncoloured cell in a graph corresponding to a Hex position.
H-search	2.9.2	An algorithm to identify connection strategies in a Hex position.
handicap cells	B.1	The initial moves in a handicap strategy.
Hex graph	2.3	A graph modelling a Hex position as in Generalized Hex.
hot game	2.5	A combinatorial game where it is desirable to be the player to move.
independent captured-reversible set	3.3	Set of captured-reversible cells corresponding to an independent set in a captured-reversible graph.
induced path domination	3.5	Domination deduced by membership in minimal winning sets.
interboundary connection	3.7.1	A monochromatic path connecting two boundary chains of a region.

interboundary equivalence	3.7.1	Relationship between two completions of a region that possess all the same interboundary connection properties.
interfere	3.3	Relationship between two captured-reversible cells when one's reverser is in the other's carrier.
internal chain	3.7.1	A chain that does not contain a border, and only neighbours a region's uncoloured components.
irregular board	2.2	A board whose sides are not all equal length.
key	2.9.2	The first move of a virtual semi connection.
killer	2.9.1	A move that renders a vulnerable cell dead.
knowledge computation	5	The process of computing all inferior cell analysis and connection strategy information for a Hex state.
knowledge threshold	6.3.2	Parameter in Monte Carlo tree search. Used to determine when a node warrants time-costly knowledge computations.
ladder	2.9.2	Border template using a series of threats, typically forming chains parallel to the border.
live cell	2.9.1	An uncoloured cell for which there exists some completion in which the cell's colour determines the winner.
live child	5.2.2	A child node in a proof number tree whose disproof number is not infinity.
location	2.2	A cell or a border.
loopy game	2.5	A combinatorial game where a series of legal moves can result in a repeat position.
maintain	2.9.2	The process of following a connection strategy.
maintenance assumption	A.2	The assumption that a player will maintain a particular connection strategy.
maximum winning carrier	5.3.2	A set of uncoloured cells that corresponds to a winning carrier if a particular player wins.
miai	4.3	Pair of uncoloured cells that serve the same purpose; if opponent plays one, then player immediately responds with the other.
miai list	4.3.1	List of miai connection substrategies tracked by each connection strategy in an augmented version of H-search.

midpoint	2.9.2	The common endpoint in an AND rule deduction.
misère game	2.5	In combinatorial game theory, a game that is won by the first player to not have a legal move available.
monotonicity	2.1	The property that additional coloured cells cannot be disadvantageous for the player using that colour.
most proving node	5.2	A leaf node in a proof number tree that intersects a minimum proof set and a minimum disproof set.
mustplay	2.9.2	The intersection of all opponent winning virtual semi connection carriers.
neighbour domination	3.4	Domination deduced via neighbourhood sets on the Hex board.
no-draw property	2.1	The property that a Hex position with no uncoloured cells must contain a winning path for at least one player.
normal game	2.5	In combinatorial game theory, a game that is won by the last player to have a legal move available.
OR-all	2.9.2	Applying the OR rule to all known first player connection strategies, to determine if any OR rule deductions are possible.
OR-k rule	2.9.2	Restriction of the OR rule to consider at most k first player connection strategies.
OR rule	2.9.2	A deduction rule in the H-search algorithm that combines connection strategies in parallel.
outcome classes	2.5	Combinatorial game theory synonym for position values.
partition chain	4.1.1	A chain that can be used to partition a connection strategy into two independent connection strategies.
pass move	2.5	A move where the position is unaltered; only the player to move changes.
path	2.2	A sequence of locations, where consecutive pairs of locations are adjacent.
PC algorithm	4.1.1	Algorithm to compute partition chains in parallel with H-search deductions.
permanently inferior	3.6	Type of fillin where the strategy extends beyond the set of coloured cells.
planarity	2.1	The property that at most one player can form a winning chain on a Hex board.

position	2.2	Defined by the board dimension and each cell's colour.
position value	2.2	Either a Black win, White win, or first player win, depending on the value of its two corresponding states.
primary cells	B.1	The first row cells adjacent to handicap cells.
probe	2.9.2	An opponent move within a player's connection strategy carrier.
proof set	5.2	A set of leaves in a proof number tree that are sufficient to prove that the player to move wins the root position.
prune	2.5	Eliminating a move from the set of legal moves being considered.
random game simulation	6.3.1	The phase of Monte Carlo tree search used to evaluate a leaf node's position.
reduced position	3.1	A Hex position derived from another Hex position via fillin.
regular board	2.2	A board whose sides are all equal length.
reverser	2.5	The negating response to a reversible move.
reversible	2.5	A move whose benefit can be negated by an opponent response.
Shannon vertex-switching game	2.3	Synonym for Generalized Hex.
split decomposition	3.7.2	Chain decomposition whose boundary is composed of three borders and one other coloured chain.
star decomposition	3.7.4	Chain decomposition where both players have a move available that captures the entire region.
star game	2.5	The simplest combinatorial game that is a first player win; both players only have moves to the zero game.
state	2.2	Defined by its position and the player to move.
state value	2.2	Either a Black win or a White win; the minimax value of a Hex state.
strategy carrier	5.3.1	Carrier of a winning connection strategy on a fillin-reduced state.

strategy-stealing argument	2.1	A proof by contradiction argument where one player adopts the winning strategy of their opponent, thereby resulting in both players having winning strategies.
surreal numbers	2.5	The number system developed by combinatorial game theory.
swap rule	1.1	Rule that can be added to Hex, where the first player selects Black's first move and then the second player chooses to play as Black or White.
touch	3.8.4	The relationship between two opposite-coloured chains that are neighbours or form an opposite-coloured bridge.
tree traversal	6.3.1	The phase of Monte Carlo tree search that traverses from the tree's root to the next leaf to evaluate.
tree update	6.3.1	The phase of Monte Carlo tree search that updates tree node data using the results of a random simulated game.
uncoloured component	3.7.1	Set of uncoloured cells corresponding to a component in the chain deleted Hex graph.
uncoloured region	3.7.1	The union of one or more uncoloured components.
union-connection	2.9.2	Connection strategy with one fixed endpoint, and a choice for the other endpoint.
unique probe deduction	5.4.5	Deducing a state value from a solved state using a pairing strategy on a dead-reversible cell and its killer.
vertex implosion	2.3	The compound process of vertex simplicialization followed by vertex deletion.
vertex simplicialization	2.3	Adding edges between a vertex's neighbours such that its neighbourhood becomes a clique.
virtual connection	2.9.2	A second player connection strategy.
virtual semi connection	2.9.2	A first player connection strategy.
vulnerable	2.9.1	A move that can be rendered dead by an opponent move.
vulnerable-by-capture	2.9.1	A move that can be rendered dead by the combination of a killer move and its captured set.
winning carrier	5.3	Carrier of a winning connection strategy.
winning carrier transposition	5.4.1	State whose value is deduced using the winning carrier of a solved state.

winning chain	2.2	A chain that contains two opposing borders.
winning connection strategy	2.9.2	A connection strategy whose endpoints are opposing borders.
winning path	2.2	A path whose endpoints are opposing borders, and whose locations are each uncoloured or the same colour as the endpoints.
zero game	2.5	A combinatorial game that is a second player win; neither player has a legal move available.

Chapter 1

Introduction

The game of Hex is of interest to the mathematics, algorithms, and artificial intelligence communities.

The invention of this game is intrinsically tied to the Four Colour Theorem [85] and the well-known strategy-stealing argument [128]. Hex, and its natural generalization the Shannon vertex-switching game, are classical PSPACE-complete problems [16, 52, 145]. Proving the no-draw property of Hex is equivalent to proving the Brouwer Fixed Point Theorem in two-dimensions [59], and Hex is also one of the first games for which an artificial intelligence player was created [158]. Nash, Shannon, Tarjan, and Berge are among the mathematicians who have researched and published about this game [21, 22, 52, 128, 158].

Despite its simple rules, Hex presents a significant challenge to artificial intelligence. Due to its large branching factor, humans have consistently outperformed computers both in terms of playing and solving Hex on all but the smallest board sizes [114, 181]. Although a reasonably strong evaluation function exists [9, 121, 158], humans' ability to intuitively decompose strategies and prune irrelevant regions have helped them maintain their advantage.

In this thesis we expand on previous research, further developing the mathematical theory, algorithms, and artificial intelligence techniques relating to this fascinating game.

1.1 Rules of Hex

Hex is a two-player perfect information game played on an $n \times n$ array of hexagonal cells. The two players are Black and White, and each player is assigned a distinct pair of opposing borders. With Black moving first, players alternate turns. On their turn, a player colours an uncoloured cell with their colour. The winner is the player who completes a path of their colour connecting their two opposing borders. See Figure 1.1.

In practice the first player advantage is significant, so Hex is typically played with the *swap rule*, which states that the first player selects the placement of Black's first move, and the second player then chooses whether to play as Black or White. Whoever is White makes the next move, and the

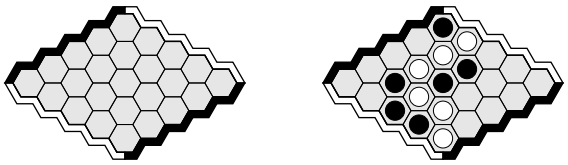


Figure 1.1: 5×5 Hex boards: empty and a completed game won by White.

players alternate turns thereafter.

1.2 Objectives

Solving and playing games via computers has been of interest to the artificial intelligence community since its earliest beginnings. The game of Hex is a classical PSPACE-complete problem, so it is unlikely that a polynomial-time algorithm exists to solve arbitrary Hex positions. Given this, it seems more beneficial to develop and improve techniques that prune the search space.

In particular, Hex positions possess important graph-theoretic properties, and combinatorial game theory is applicable in terms of pruning inferior moves and analyzing combinatorial decompositions. Hex algorithms exist to identify connection strategies, resulting in early termination of the search space.

In summary, the objectives of this doctoral research are to:

- expand on the mathematical and algorithmic knowledge for the game of Hex, and
- apply and adapt artificial intelligence techniques to make use of such knowledge.

1.3 Overview

This thesis is structured as follows:

- In Chapter 2 we review all previous work related to Hex, including the basic properties, concepts, and notation that will be used throughout this thesis.
- In Chapter 3 we apply combinatorial game theory to reformulate previous inferior move analysis in the game of Hex. We then identify several new types of inferior cell. Graph-theoretic properties of board decompositions are explored, and efficient algorithms applying this knowledge are produced.
- In Chapter 4 we discuss enhanced algorithms for identifying Hex connection strategies, and compare several variations in terms of efficiency and completeness.
- In Chapter 5 we discuss the automated solving of Hex states, including improvements to previous search algorithms and the application of our new techniques. We review our solver's performance, including the surpassing of all previous benchmarks.

- In Chapter 6 we examine the performance of three artificial intelligence Hex players, each with a different foundational search algorithm and evaluation methodology. We also analyze the benefits of applying our new theory to heuristic players.
- In Appendix A we discuss the application of our new inferior cell analysis to probes of a common connection strategy.
- In Appendix B we discuss the application of our new inferior cell analysis to produce an efficient and explicit handicap strategy for Hex.
- In Appendix C we analyze all of the Hex games from the 2008 and 2009 International Computer Olympiads.
- In Appendix D we list open questions relating to the game of Hex.

1.4 Contributions

The main results of this thesis can be summarized as follows:

- Further developing Hex inferior cell analysis, including:
  - Identifying captured-reversible moves.
  - Identifying neighbourhood domination and induced path domination.
  - Identifying permanently inferior cells.
  - Identifying decompositions using opposite-colour bridges.
  - Identifying cyclic decompositions, and their relation to captured sets.
  - Identifying star decompositions, and their relation to move domination.
  - Applying the above to prune connection strategy probes and deduce further domination implications.
  - Applying the above to construct an efficient and explicit handicap strategy for Hex.
- Developing several efficient modifications of the H-search algorithm that identify more connection strategies, including:
  - Producing a new orthogonal deduction rule for identifying new connection strategies from existing ones.
  - Applying inferior cell analysis to allow for partial intersection of connection strategies in deduction rules.
  - Applying common substrategies to allow for partial intersection of connection strategies in deduction rules.

- Developing an extremely strong automated Hex solver, including:
  - Applying inferior cell analysis to deduce many state values from each solved state.
  - Using strategy-stealing arguments to prune states during search.
  - More than a 100-fold speedup over other state-of-the-art solvers.
  - Being the first to produce an automated solver capable of solving any and all 8×8 openings.
  - Being the only ones to produce an automated solver capable of solving any 9×9 openings. This marks the first time automated solvers have surpassed humans in terms of solved Hex openings.
- Developing strong automated Hex players, including:
  - Using alpha-beta search, Monte Carlo tree search, and proof number search to produce three distinct Hex players.
  - Applying our inferior cell analysis and Hex solver to significantly improve our automated players.
  - Winning both the gold and silver medals for Hex in the 2008 and 2009 International Computer Olympiads.

1.5 Publications

The research described in this thesis includes results appearing in the following publications (listed in chronological order by submission date):

- Philip Henderson and Ryan B. Hayward. Probing the 4-3-2 edge template in Hex. In van den Herik et al. [164], pages 229–240.
- Broderick Arneson, Ryan B. Hayward, and Philip Henderson. Wolve 2008 wins Hex tournament. *ICGA Journal*, 32(1):49–53, March 2009.
- Philip Henderson, Broderick Arneson, and Ryan B. Hayward. Solving 8x8 Hex. In Boutilier [26], pages 505–510.
- Philip Henderson, Broderick Arneson, and Ryan Hayward. Hex, braids, the crossing rule, and XH-search. In van den Herik and Spronck [165], pages 88–98.
- Broderick Arneson, Ryan B. Hayward, and Philip Henderson. MoHex wins Hex tournament. *ICGA Journal*, 32(2):114–116, June 2009.
- Philip Henderson and Ryan B. Hayward. A handicap strategy for Hex. In Richard J. Nowakowski, editor, Games of No Chance IV. Cambridge University Press, 2010 (in press).

- Broderick Arneson, Ryan B. Hayward, and Philip Henderson. Solving Hex: Beyond humans. Accepted to Computers and Games, 2010.
- Broderick Arneson, Ryan B. Hayward, and Philip Henderson. Monte Carlo Tree Search in Hex. Accepted to Transactions on Computational Intelligence and AI in Games, Special Issue on Monte Carlo Techniques and Computer Go, 2010.
- Philip Henderson and Ryan B. Hayward. Captured-reversible moves and star decomposition domination in Hex. Submitted to Integers, 2010.

Chapter 2

Related Work

In this chapter we summarize previous research on Hex and related topics. We also introduce much of the notation and terminology that will be used throughout this thesis.

2.1 Fundamental Hex Properties

Hex was invented independently by Piet Hein in 1942 and Nobel laureate John Nash in 1948, and in both cases its invention was closely related to mathematical properties. Hein was contemplating the (then unsolved) Four Colour Conjecture, attempting to disprove it [85, 115]. He noted that with a tesselation of hexagons, unlike a tesselation of triangles or squares, any two-colouring would always avoid deadlock and hence guarantee a monochromatic path for one of the colours. By contrast, Nash was looking for a game whose value (assuming optimal play) could be deduced, yet where the method for attaining this outcome was completely unknown. Nash came to realize that if no draw was possible, and if having an extra move was never disadvantageous, then the existence of a first player winning strategy was guaranteed. This was the inspiration for the now well-known strategy-stealing argument [128].

The key properties of Hex are:

1. If all cells are coloured, then at most one player has a winning path. This is due to *planarity*.
2. If all cells are coloured, then at least one player has a winning path. This is the *no-draw property*.
3. Colouring additional cells for one player can never be to their disadvantage. That is, Hex is *monotonic*.
4. The two players have *isomorphic* roles on the empty $n \times n$ board position.
5. The first player must have a winning strategy by the *strategy-stealing argument*.

Of these properties, the second is the most difficult to prove. In fact, proving the no-draw property of Hex is equivalent to proving the Brouwer Fixed Point Theorem in two dimensions [59];

proofs (and sketches of proofs) of this property abound [19, 173].

As mentioned in Chapter 1, Hex is often played with the *swap rule*. Since every Hex state is either a Black win or a White win by the no-draw property, and since the second player can select whether to play as Black or White following the first player's selection of a Hex state, it follows that the swap-rule variant of Hex must be a second player win.

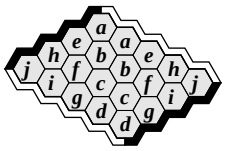


Figure 2.1: A winning pairing strategy on the 5×4 board.

Another proposed handicap method is to play Hex on rhomboids (*i.e.*, on $m \times n$ boards where $m \neq n$). However, Claude Shannon observed that this game is a trivial win for the player whose opposing borders are closer together, regardless of who plays first, using a simple pairing strategy [60]. See Figure 2.1.

2.2 Basic Terminology and Notation

The *size* of a Hex board is its number of cells. Unless stated otherwise, throughout this thesis we will be assuming play on *regular* $n \times n$ Hex boards, not *irregular* $m \times n$, $m \neq n$ Hex boards. The *dimension* of a (regular) Hex board is the length of one board side. That is, an $n \times n$ board has dimension n and size n^2.

Cells are the hexagonal locations in which either player can play. *Borders* are the four coloured sides of the Hex board; these can be referred to by direction: *North, South, East, West*. *Locations* includes both cells and borders. The *colour of a location* l, denoted $\chi(l)$, is one of *Black, White,* or *Uncoloured*, and we use the notational shorthand B, W, U respectively. *Coloured cells/locations* refers to cells/locations whose colour is Black or White, while *uncoloured cells/locations* refers to cells/locations whose colour is not Black nor White. For instance, the colour of the North and South borders is always Black, and borders are always coloured.

Unless stated otherwise, throughout this thesis we will be assuming that Hex is played without the swap rule. Thus a Hex *player* P is either Black or White, and $\overline{P}$ denotes the opponent of P. A *Hex position* is defined by the board dimension and each cell's colour. A *Hex state* is defined by a Hex position and the player to move.

Hex is a perfect information game with no draws, so a Hex state has one of two values: a *Black win* or a *White win*. Hex is monotonic, so no position is a *second player win*, so a Hex position has one of three values: a *Black win* regardless of who moves first, a *White win* regardless of who moves first, or a *first player win*. See Figure 2.2. In this thesis (in)equality among states and positions relates only with respect to these values.

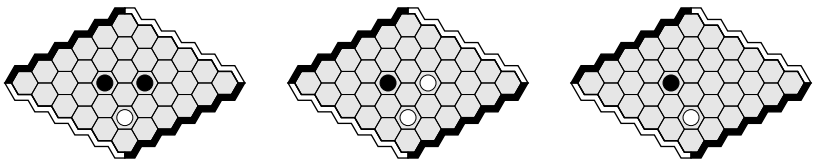


Figure 2.2: Black win, White win, and first player win Hex positions.

For Hex states S_1, S_2, we write $S_1 \geq_P S_2$ if the value of state S_1 is at least as good for player P as state S_2, namely if P has a winning strategy in S_1 whenever P has a winning strategy in S_2. Clearly $S_1 \geq_P S_2$ if and only if $S_2 \geq_{\overline{P}} S_1$. Given a Hex position H, H^P represents the state whose position is H with player P to move. For Hex positions H_1, H_2, we write $H_1 \geq_P H_2$ if $H_1^P \geq_P H_2^P$ and $H_1^{\overline{P}} \geq_P H_2^{\overline{P}}$. That is, $H_1 \geq_P H_2$ implies that player P prefers position H_1 to position H_2 regardless of who moves next. We write $X \equiv Y$ if two states/positions have the same value, namely $X \geq_P Y$ and $X \geq_{\overline{P}} Y$. We write $X = Y$ if two states/positions are identical.

A P move is a move by player P, and a $P(c)$ move is a move by player P to uncoloured cell c. For a position H, a player P, an uncoloured cell c, a set of uncoloured cells C, and a set of coloured cells D:

- $H + P(c)$ is the position obtained from H by P-colouring c,
- $H + P(C)$ is the position obtained from H by P-colouring all cells in C, and
- $H - D$ is the position obtained from H by uncolouring all cells in D.

For a Hex position H and a colour or set of colours C, we denote by $H \to C$ the set of locations in H whose colour is C or in C. If we wish to restrict our attention to a set of locations L in H, we use L_H, or simply L if the position is implicit.

For positions H_1 and H_2, we say that H_2 is a *continuation* of H_1 if $(H_1 \to B) \subseteq (H_2 \to B)$ and $(H_1 \to W) \subseteq (H_2 \to W)$. A continuation with no uncoloured cells is called a *completion*.

Given a cell, its *neighbours* are the locations directly adjacent to it. The neighbours of a border are all cells in the adjacent row/column. We use $N(l)$ to denote the neighbour set of location l. For instance, a cell has at most six neighbours (it has fewer than six if it is adjacent to one or more borders), and the cardinality of each border's neighbour set is equal to the board's dimension.

A *path* is a sequence of locations $l_1, l_2, \ldots, l_k$ such that l_i and l_{i+1} are neighbours for $1 \leq i < k$. Such a path is an (l_1, l_k)-path, and l_1, l_k are called the *endpoints* of the path. A *winning path* is a path whose endpoints are opposing borders, and whose locations are each uncoloured or the same colour as the endpoints.

Two coloured locations x, y are *connected* if there exists a monochromatic (x, y)-path. A *chain* is a maximal set of connected locations. A *winning chain* is a chain that includes two opposing borders. Note that the *colour of a chain* is equal to the colour of every location in the chain. Given a chain, its neighbours are those locations that neighbour at least one of its elements, but that are not

contained within the chain. That is, for chain $C = \{l_1, \ldots, l_k\}$, $\chi(C) = \chi(l_1) = \cdots = \chi(l_k)$ and $N(C) = \cup_{i=1}^{k} N(l_i) \setminus C$.

2.3 Hex Graphs

We assume the reader is familiar with basic graph theory, including paths, connected components, cliques, independent sets, cutsets, and list colouring. This thesis uses the notation and terminology of [27].

The game of Hex can be thought of as a game on a graph, where initially each uncoloured cell and Black border is represented by a distinct vertex, with edges connecting neighbouring locations [173]. A Black move to a cell makes all pairs of neighbours adjacent and then deletes the vertex; we call these two stages *vertex simplicialization* and *vertex deletion* respectively, or simply *vertex implosion* for the combined process. A White move to a cell deletes the corresponding vertex. Black wins if the two vertices corresponding to Black borders become direct neighbours, while White wins if they disconnect the graph such that these two vertices are in different connected components.

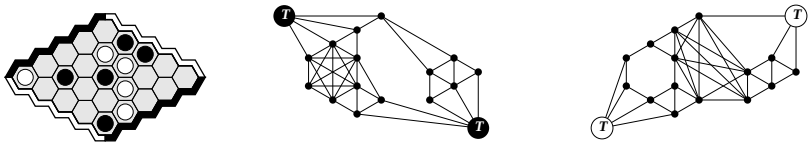


Figure 2.3: A Hex position and its Black and White graphs. This figure is taken directly from van Rijswijck's thesis [173].

The graph of a Hex position obtained by this process is called its *Black graph*, and in this formulation we call Black the *Short* player, and White the *Cut* player. The *White graph* of a Hex position is defined similarly, with the roles of Black and White interchanged. See Figure 2.3.

This concept can also be generalized to any graph: two vertices are marked (*i.e.*, the borders to be connected), and Short and Cut alternate turns performing vertex implosion and vertex deletion respectively (on unmarked vertices only), until either the two marked vertices are direct neighbours or in different components. This generalized version is known as the *Shannon vertex-switching game*, or simply *Generalized Hex* [16, 52, 95].

2.4 Computational Complexity

We assume the reader is familiar with the basics of computational complexity, including O-notation and the complexity classes P, NP, and PSPACE. Please refer to [45, 69] for details.

Determining the winner of a Hex (or Generalized Hex) position is a PSPACE-complete problem [52, 145]. Thus, developing an *efficient* (*i.e.*, polynomial-time) algorithm to solve arbitrary Hex positions is equivalent to proving that P equals PSPACE and, as a consequence, proving that P equals NP.

A logical characterization of the class of PSPACE-complete problems can be constructed using first-order logic, a built-in successor relation, and an operator corresponding to Generalized Hex [16].

2.5 Combinatorial Game Theory

Combinatorial game theory (CGT) is the study of two-player perfect information games, and thus is directly applicable to Hex; see [23] for details. However, we will not use CGT notation nor CGT values (*i.e.*, the *surreal numbers*) to describe Hex positions, but rather our own simpler value definitions from §2.2, which correspond to *outcome classes* in CGT.

In the CGT framework:

- Hex is a *hot* game, since both players always prefer to have the next move.
- Hex is not *loopy*, since a valid move sequence can never result in a repeat position.
- Hex is *normal*, not *misère*, since the winner is the last player to make a legal move (*i.e.*, the game terminates when a winning chain is formed).

Only two surreal numbers are of interest in this thesis. The first is the *zero game* $0 = \{|\}$, which is a second player win since neither player has a legal move available. The second is the *star game* $* = \{0|0\}$, which is a first player win since both players only have legal moves to a zero game, a second player win. The star game is the simplest first player win, since only one legal move remains in the game, regardless of who moves first.

We will be applying CGT's theory regarding inferior moves, namely dominated and reversible moves. When determining the value of a position, dominated moves can be pruned and reversible moves can be bypassed without altering the position's value.

Let H be a Hex position with legal P moves m_1, m_2 such that $H + P(m_1) \geq_P H + P(m_2)$. Then we say that m_1 *P dominates* m_2 in H. In the CGT framework, it has been proven that P dominated moves can be *pruned* from consideration by player P, so long as P considers at least one dominating move.

Let H be a Hex position with a legal P move m_1, and suppose that $H + P(m_1)$ has a legal $\overline{P}$ move m_2 such that $H \geq_P H + P(m_1) + \overline{P}(m_2)$. Then we say that m_1 is a *P reversible move* with *$\overline{P}$ reverser* m_2 in H. Intuitively, the opponent response negates any benefit gained by the previous move, reversing the latter's effect on the position's value. In the CGT framework, it has been proven that reversible moves can be *bypassed*, meaning that it can be assumed that P move m_1 in H will automatically be replied to with $\overline{P}$ move m_2. Thus, P move m_1 in H can be deleted and replaced by all legal P moves in $H + P(m_1) + \overline{P}(m_2)$.

We note that players need not alternate turns in a subgame, and sometimes we will want to allow this in Hex: a *pass move* is a move in which a player does not colour any cell. By monotonicity, the

pass move is always dominated by all other legal moves, and thus adding the pass move as an option does not change the value of any Hex states or positions.

2.6 Other Game Theory

Besides CGT, there are several other types of game theory that are applicable to Hex. We will not be applying any of these in this thesis, but we summarize their contributions to Hex below.

Van Rijswijck introduced set colouring and binary combinatorial games as an attempt to integrate combinatorial decompositions in Hex with combinatorial values and CGT results [172, 173]. In order to satisfy CGT's winning conditions (*i.e.*, determined when a player no longer has any legal moves), van Rijswijck introduced atomic values True and False, which add a fixed number of moves to the corresponding player. Conjunctions and disjunctions are modelled by adding the appropriate number of atomic values, thereby emulating the decision formula to determine the winner while ensuring that CGT will agree on the outcome.

The research by van Rijswijck is related to Game-SAT [185] — a game where players alternate turns assigning values to boolean variables, in an attempt to make a formula evaluate to true or false — when restricted to two colours. The monotonicity of Hex ensures that a player would never want to use their opponent's colour, so the minor rule differences between these two theoretical games do not affect the optimal play or outcome when applied to Hex.

Yamasaki studied the theory of division games [180], of which Hex is a particular example. A division game is a two-player game played on a set, where each player claims any single unclaimed element from the set on their turn. Unlike in Hex, turns do not necessarily alternate in division games. A division game is played until all elements are claimed, and a specified function maps any final partition to the winning player (no draws are allowed). For instance, in Hex the elements of the set are all uncoloured cells, and the function expresses which player has a winning chain in a given partition. Most of Yamasaki's results demonstrate value (in)equalities given some modification of a division game, such as adding new elements to the set or altering the move ordering in some fashion. Yamasaki also focused on games where the sets are regular (*i.e.*, never disadvantageous), misère (*i.e.*, never advantageous), or negligible (*i.e.*, never affect the outcome). In this manner, Yamasaki was able to re-derive many previously-known Hex properties. The difference between Yamasaki's framework and that of van Rijswijck is that the former does not enforce alternating turns, while the latter allows for more general (*i.e.*, not necessarily binary) partitioning.

Jensen and Toft studied Hex in relation to positional games on hypergraphs [95]. Players alternate turns colouring vertices in a hypergraph, and a player wins if they manage to claim all elements of any hyperedge. They note that the no-draw property of Hex can be modelled as a complete bipartite graph, where each vertex in one half of the bipartition corresponds to a distinct winning path for that player: list-colouring such a graph, where the colours available to a vertex are the Hex cells in its corresponding winning path, must be impossible since this would correspond to a draw in

Hex. Some results for positional hypergraph games are known, but unfortunately these statements are trivial when restricted to Hex, or else simply inapplicable.

2.7 Hex Variants

As mentioned earlier, the swap rule and irregular board sizes are attempts to balance the game of Hex; Beck's Hex is another attempt to balance the game of Hex. In Beck's Hex, the White player chooses Black's first move; play then proceeds as normal. Thus, Black wins Beck's Hex if and only if every first move for Black is winning. However, as Beck proved, an acute corner cell is guaranteed to be a losing opening move in Hex for all boards of dimension at least two [18]; this proof uses a strategy-stealing argument and properties of winning paths. A similar proof by Beck shows that a cell neighbouring an acute corner cell and a Black border is also a losing opening move on all boards of dimension at least three [19].

Another natural variation of Hex is misère Hex, where a player wins if their opponent forms a winning chain. Using a strategy-stealing argument, it can be shown that the losing player can delay their loss until the last uncoloured cell is played, implying that the parity of the board size determines the winner [109]. In other words, misère Hex is a win for Black if and only if the board size is even. When the board size is even, the acute corner cell is a winning opening move [50, 180].

Alpern and Beck studied Hex played on the annulus, with one player trying to connect the inside to the outside (*i.e.*, forming a path connecting ends of a cylinder) and the other trying to form a closed ring [5]. They proved that this game retains the no-draw property, and that if the cylinder has an even rotational dimension, then a simple pairing strategy guarantees that the player connecting the ends of the cylinder will always win, even as the second player and regardless of the board's dimensions. The question of who wins on the annulus with odd rotational dimension is still open.

Kriegspiel Hex — Hex where neither player can see the board — is a first player win on boards of dimension at most three. Furthermore, no guaranteed winning strategy can exist for boards of dimension four or greater [120]. Many other Hex variants have been defined, such as random-turn Hex (a coin toss determines who gets the next move), Vex (connecting an obtuse corner cell to either of the two opposing borders), Vertical Vex (connecting a cell to a border), Tex (played on an infinite Hex board, where one player tries to enclose their opponent's opening move, while the other tries to perpetually escape), their misère variants, and so on [51, 137]. However, with the exception of random-turn Hex, very few results exist for these games [115].

2.8 Related Games

Aside from direct variants of Hex, there are also many connection games that are closely related. One example is Y, a game played on a triangular grid of hexagonal cells in which players try to form a chain connecting all three borders [156]. Y is a generalization of Hex in that any Hex position can

be modelled on a Y board. Y also possesses the properties of no draws and monotonicity, and so is also a first player win.

Another example is Havannah, a game played on a hexagonal grid of hexagonal cells in which players try to form either a chain connecting two corner cells, a chain connecting three border cells, or a closed ring. Havannah is monotonic, but draws are possible.

There are also two important variants of Generalized Hex. The multi-Shannon game is played on general graphs, but the Connect player is trying to connect more than two marked vertices [24]. The Shannon edge-switching game is similar to Generalized Hex, except that players Cut and Short delete and contract edges respectively. The game of Bridg-it, invented by Gale, is the Shannon edge-switching game played on a rectangular grid. Unlike the Shannon vertex-switching game, the edge-switching game has been efficiently solved using matroids, both in its normal and misère forms [76, 110]. The proof was later simplified to a graph-theoretic argument, demonstrating that the invariant necessary for Connect to win is the maintenance of two edge-disjoint spanning trees [38].

2.9 Hex-Specific Research

2.9.1 Inferior Cell Analysis

By analyzing the graphs of a given Hex position, it is possible to efficiently prune many moves from consideration. Previously known pruning techniques involved dead, vulnerable, captured, and capture-dominated cells, which we summarize here.

Following observations by Beck *et al.* [18] and Schensted and Titus [156], Hayward and van Rijswijck defined a class of provably useless Hex cells, called dead cells: with respect to a particular Hex position, an uncoloured cell c is *live* if there exists some completion of the position in which changing c's colour changes the winner [84]; an uncoloured cell is *dead* if it is not live. By definition, a cell is live/dead for Black if and only if it is live/dead for White.

Equivalently, a cell is live if and only if it is contained in some minimal set of uncoloured cells that can yield a winning chain. Thus determining whether a cell is live reduces to determining in a graph whether a given vertex is on a minimal path joining two other given vertices; this problem is NP-complete for general graphs [24]. If a vertex is separated from the two endpoints by a clique cutset, then it cannot be on a minimal connecting path, and therefore must be dead.

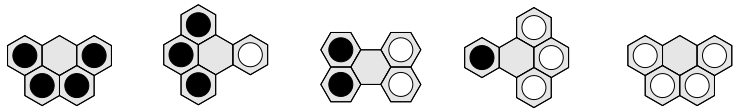


Figure 2.4: Dead patterns. In each case colouring the empty cell Black or White does not alter a position's value.

Some dead cells can be recognized by matching patterns of neighbouring cells. For example, for each pattern in Figure 2.4 the uncoloured cell is dead [83]. Since there is no completion in which a dead cell's colour matters, it follows that:

- a dead cell in a position remains dead in all continuations, and
- a dead cell can be arbitrarily coloured without changing a position's value.

If a player has a winning strategy in a Hex state, then they have a winning strategy with no move to a dead cell [84].

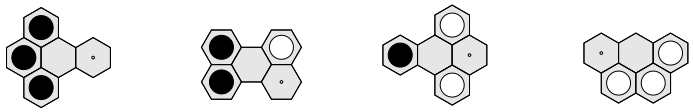


Figure 2.5: Black vulnerable patterns. In each case a White move to the dotted cell kills the empty cell.

An uncoloured cell c is P *vulnerable* if $\overline{P}$ has a move that makes c dead; this $\overline{P}$ move is c's $\overline{P}$ *killer*. See Figure 2.5. If a player P has a winning strategy in a Hex state, then they have a winning strategy with no move to a dead or P vulnerable cell [84].

While dead cells remain dead in all continuations, P vulnerable cells need not remain P vulnerable. For instance, if c is P vulnerable with $\overline{P}$ killer k, and later P plays at cell k, then $\overline{P}$ may no longer have a move that makes c dead. However, for a position H_1 with a P vulnerable cell c, and a continuation H_2 of H_1 where c is still uncoloured and only $\overline{P}$-coloured cells have been added, then c is dead or P vulnerable in H_2.

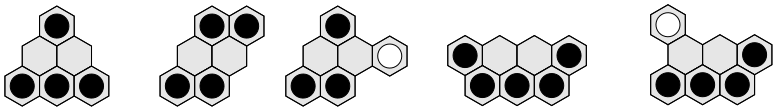


Figure 2.6: Black captured patterns. In each case colouring the empty cells Black does not alter a position's value.

A set C of uncoloured cells in position H is P *captured* if P has a second player strategy on C such that for each terminal position L produced by the strategy (*i.e.,* L is a continuation of H where only cells in C have been coloured, and no cell in C remains uncoloured), each cell in $C_L \to \overline{P}$ is dead in position $L - (C_L \to \overline{P})$. Since a dead cell can be assigned any colour without altering a position's value, it follows that for each such terminal position L,

$$L \equiv L - (C_L \to \overline{P}) \equiv (L - (C_L \to \overline{P})) + P(C_L \to \overline{P}),$$

and so $L \equiv L - C + P(C)$. In other words, if $\overline{P}$ ever plays in a P captured set, then P has a replying strategy that guarantees no net benefit to $\overline{P}$. See Figure 2.6.

For a Hex position H and pairwise non-intersecting uncoloured cell sets X, Y, Z, if X is P captured in H, then X is P captured in the continuation $H + P(Y) + \overline{P}(Z)$, as the cells in Y and Z neither affect the capturing strategy nor revive any cell that is dead in a terminal position. Thus if H is a Hex position with a P captured set X, then $H \equiv H + P(X)$. Moreover, combining a captured set strategy with a winning strategy on the reduced board yields a winning strategy for P in the original position [79].

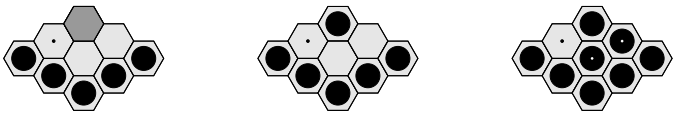


Figure 2.7: A cell that is White vulnerable-by-capture. If Black plays the shaded cell, Black captures cells which in turn kill the dotted cell.

The definition of vulnerable cells can be expanded using captured cells: an uncoloured cell c is P *vulnerable-by-capture* in position H if $\overline{P}$ has a move k such that $H + \overline{P}(k)$ has $\overline{P}$ captured set X and c is dead in $H + \overline{P}(X \cup \{k\})$. See Figure 2.7. As before, k is the $\overline{P}$ killer of the P vulnerable cell.

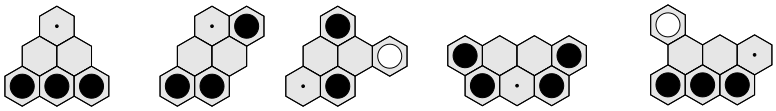


Figure 2.8: Black domination patterns. In each case a Black move to the dotted cell capture-dominates a Black move to any of the empty cells.

If position $H + P(c)$ has a P captured set X, then by monotonicity it follows that in position H the cell c P dominates all cells in X [77]. That is, for all cells x in X,

$$H + P(c) \equiv H + P(c) + P(X) \geq_P H + P(x).$$

See Figure 2.8. To distinguish this type of domination from the more general CGT meaning, we call this *capture-domination*.

Fillin refers to colouring a set of cells in a given position without altering its value. For instance, colouring dead cells or P-colouring a P captured set are examples of fillin.

Van Rijswijck used the five local dead patterns to identify vulnerable, captured, and capture-dominated patterns [173]. See Figure 2.9.

2.9.2 Identifying Connection Strategies

In Hex, a common tactical question is whether a player has a strategy to create a chain that connects two locations, an obvious example being a chain connecting two opposing borders. We may also want to connect locations that are not already coloured, so we define a *connection strategy* for player

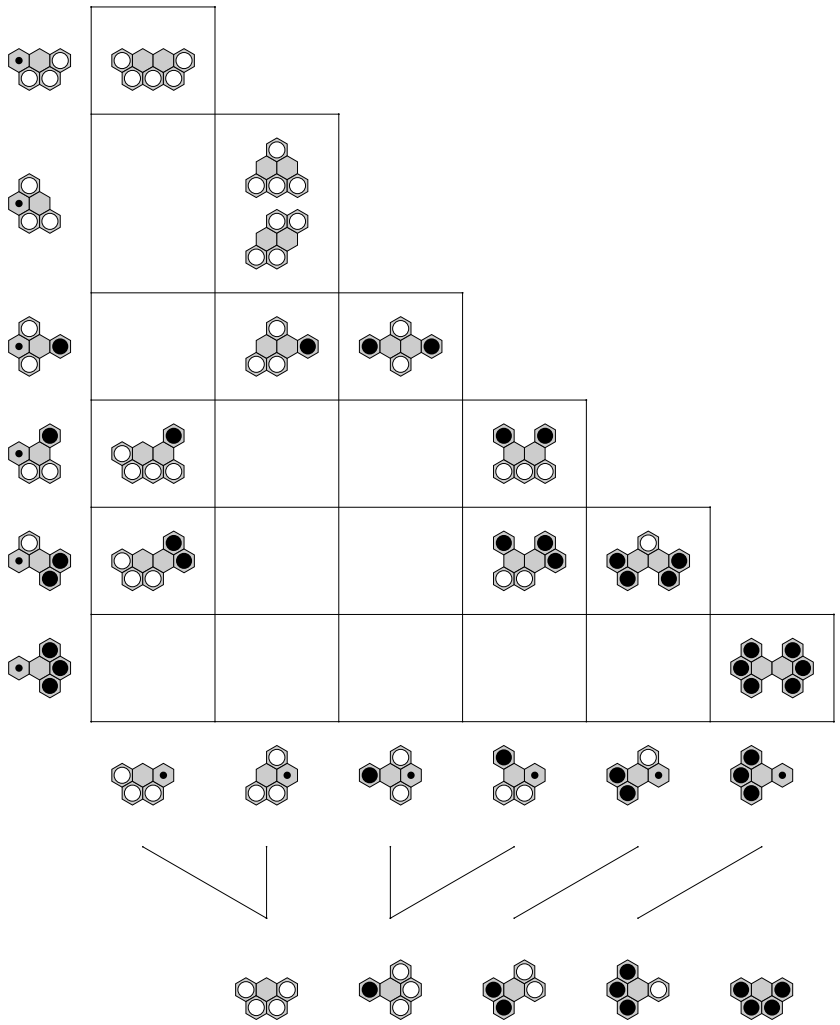


Figure 2.9: Deducing inferior cell patterns. This figure is taken directly from van Rijswijck's thesis [173].

P in a given Hex position to be an (alternating-turn) strategy that guarantees the construction of a P chain that either neighbours or contains each of the two specified locations. A second player connection strategy (respectively first player connection strategy) is a *virtual connection* or VC (respectively *virtual semi connection* or SC). For a VC/SC, the two locations being connected are its *endpoints*, and the set of uncoloured cells used in the connection strategy is its *carrier*. The initial move of an SC strategy is its *key*. See Figure 2.10.

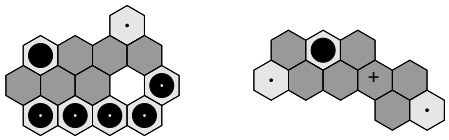


Figure 2.10: Diagrams of a Black VC and a Black SC. Carriers are shaded, endpoints are dotted, and the SC key is $+$.

With respect to a player's connection strategy, a *probe* is a move by their opponent to a carrier cell; all other opponent moves are *external*. A player *maintains* a virtual connection if they always respond to opponent probes with the corresponding connection strategy response. A connection strategy is *winning* if its endpoints are opposing borders. A VC is a *border template* if one of its endpoints is a border and it is commonly occuring (*e.g.*, it matches on positions with very few coloured cells). Informally, a *ladder* is a border template that uses a series of forced threats, often resulting in both players producing parallel coloured rows/columns. David King has produced a thorough list of border templates [100]. Two common VCs are the *bridge* and the *4-3-2*. See Figure 2.11. Since a P border is equivalent to a row of P-coloured cells, then the *border bridge* matches the leftmost captured pattern in Figure 2.6.

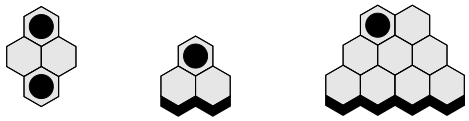


Figure 2.11: A bridge, a border bridge, and a border 4-3-2.

Anshelevich developed *H-search* [7, 10, 11], a hierarchical VC/SC composition algorithm. H-search uses one base case and two deduction rules, iterating until no further connections can be deduced:

1. *Base case*: A player has an empty set carrier VC between each pair of neighbouring locations such that neither is opponent-coloured.

2. *AND rule*: If a player has VCs α_1, α_2 with respective endpoints $\{p_1, p_2\}, \{p_2, p_3\}$ and carriers C_1, C_2 such that $C_1 \cup \{p_1\}$ and $C_2 \cup \{p_3\}$ do not intersect, then

- if p_2 is uncoloured, then combining α_1, α_2 forms an SC with endpoints $\{p_1, p_3\}$, carrier $C_1 \cup C_2 \cup \{p_2\}$, and key p_2,
- if p_2 is a coloured cell for the player, then combining α_1, α_2 forms a VC with endpoints $\{p_1, p_3\}$ and carrier $C_1 \cup C_2$.

In each case the location p_2 is called the *midpoint* of the AND rule.

3. *OR rule*: If a player has SCs $\alpha_1, \ldots, \alpha_k$ each with endpoints $\{p_1, p_2\}$ but with respective carriers $C_1, \ldots, C_k$ such that $C_1 \cap \ldots \cap C_k$ is the empty set ($k \geq 2$), then combining $\alpha_1, \ldots, \alpha_k$ forms a VC with endpoints $\{p_1, p_2\}$ and carrier $C_1 \cup \ldots \cup C_k$.

There are several ways in which H-search can be altered:

- The base case can be enlarged to any list of precomputed connection strategies (*e.g.*, border templates).
- Each chain selects a single element location as its representative, and the representative's neighbours are all uncoloured cells in the chain's neighbourhood; recall the graphs representing Hex positions. All elements of a chain have empty set carrier VCs to one another, so H-search will perform many redundant computations without this alteration.
- Superset carrier connection strategies can be deleted [121]. Since all deductions are restricted by carrier intersection, it is wasteful to compute deductions using a VC/SC with the same endpoints but a carrier that is a strict superset of another; this alteration reduces such redundant work.
- Allowing borders to be midpoints of the AND rule. This variant of H-search typically finds far more connections [121, 122].
- If ORing all known SCs does not produce a VC, then ORing a subset of them cannot produce a VC. This initial *OR-all* check improves H-search's efficiency.
- In a recursive implementation of the OR rule, backtrack immediately if the most recent SC did not shrink the cumulative carrier intersection. This improves the algorithm efficiency without changing its output, due to the redundancy of superset carrier connection strategies [140].
- The OR rule can be restricted to a bounded number of SCs, with the exception of the OR-all check; with a bound of k we call this the *OR-k rule*. However, this alteration can reduce the number of connections identified by H-search.
- Heuristic limits can be used to restrict the VCs/SCs for each pair of endpoints that are considered by H-search's deduction rules. The use of such limits is accompanied by an ordered sort of VCs/SCs by carrier size, as smaller carrier connections are more likely to be helpful in

H-search deductions [121]. This alteration can decrease the number of connections identified by H-search.

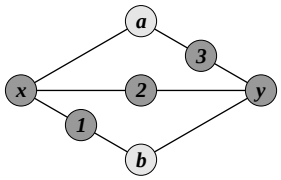


Figure 2.12: An SC not found by H-search. The SC has endpoints $\{a, b\}$ and key x (or key y).

As Anshelevich observed, H-search does not identify all connection strategies [7, 10], including the SC shown in Figure 2.12. Anshelevich developed a *generalized H-search* algorithm which is complete, and allows arbitrary size carrier intersections in its deductions [8]. This algorithm essentially computes k*-connections* for arbitrary k: connection strategies where one player gets k unanswered moves before players alternate turns (*e.g.*, a VC is a 0-connection and a SC is a 1-connection). However, generalized H-search is far too slow in practice as it uses deduction rules whose computational complexity grows exponentially in k, and thus exponentially in terms of board size. This computational complexity is to be expected since identifying all connection strategies necessarily implies identifying any connection strategies between opposing borders, and thus determining the value of the Hex position (a PSPACE-complete problem, see §2.4).

Van Rijswijck noted that during depth-first search, a VC deduced by H-search in one state implies the existence of a SC in its predecessor. By *backing up* such connections, one partially alleviates H-search's incompleteness. Rasmussen *et al.* expanded on this work by storing discovered connections in a generalized form, so that they can be applied to a wider class of Hex positions.

In yet another attempt to address H-search's incompleteness, Rasmussen *et al.* use an independent VC search whenever the OR rule finds a set of SCs with small carrier intersection [142]. Naturally this finds more connections, but unfortunately the search time can be exponential in the number of uncoloured cells.

This focus on identifying connection strategies is due to their ability to prune the search space:

- If P has a winning VC, then P wins the current state regardless of who moves next.
- If P has a winning SC and P moves next, then P wins the current state.
- If P has a winning SC and $\overline{P}$ moves next, then any $\overline{P}$ move external to this SC is provably losing.

The *mustplay* for the player to move is the intersection of their opponent's winning SC carriers. It follows that all moves outside of the mustplay are losing. See Figure 2.13.

Despite the benefits of H-search — pruning losing moves and producing perfect endgame play — it is time-costly; efficient implementations can compute the connection strategies for about 25

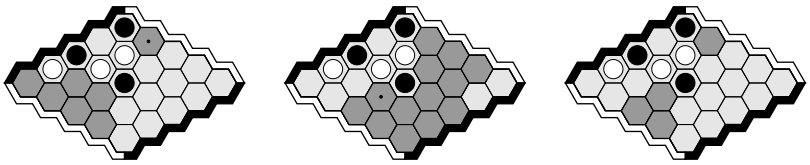


Figure 2.13: Two White winning SC carriers and the corresponding mustplay for Black.

positions per second on tournament-size 11 $\times$ 11 boards [121]. Furthermore, this performance is worse if borders can be midpoints of the AND rule.

Aside from VCs/SCs, other types of connection strategies have been defined. For instance, Noshita defined *union-connections*, which are connections of the form "location x connects to location y or location z" [131].

2.9.3 Solving Small Boards

Using Allis' terminology [3], the empty $n \times n$ Hex board is ultra-weakly solved (*i.e.*, we know the value of the initial board state). One goal of current Hex research is to achieve the following on successively larger board sizes:

- To weakly solve the initial board state (*i.e.*, to find a winning first player strategy),
- To ultra-weakly solve all opening moves (*i.e.*, to determine the second player's correct choice of colour when playing with the swap rule),
- To weakly solve all opening moves (*i.e.*, to find a winning strategy for the second player when playing Hex with the swap rule), and
- To strongly solve the board size (*i.e.*, to develop an algorithm capable of solving any position on the given board size in a reasonable amount of time).

Strongly solving Hex positions on board sizes up to 5 $\times$ 5 is easy, so few comments have been made about such positions [60, 85]. In 1995 Enderton developed an algorithm capable of weakly solving all 6 $\times$ 6 openings [48]. In 2000 van Rijswijck's automated solver could strongly solve the 6 $\times$ 6 board [169].

In 2001 Yang weakly solved 7 $\times$ 7 by hand, and in 2002 Yang *et al.* weakly solved 17 of the 49 opening moves on 7 $\times$ 7 [182, 184]. Yang's main tool is the decomposition method: build larger connection strategies from basic ones, so that a common substrategy can be used to respond to large sets of moves, thus dampening the combinatorial explosion. Yang's solution uses over 40 templates, and its correctness proof has 12 pages of case analysis. In 2004 Noshita weakly solved 7 $\times$ 7 with a similar but simpler proof; this was attained by applying union-connections [131].

In 2003 Hayward *et al.* weakly solved all 7 $\times$ 7 openings [83]; this was the first automated 7 $\times$ 7 solution. Two tools were fundamental to their success: inferior cells and H-search. Move

ordering was also of key importance, since the solver used depth-first search: when H-search could not find any opponent winning SCs, moves were ordered using Shannon's electric circuit resistance evaluation function (see §2.9.4); otherwise moves were ordered by mustplay size, breaking ties with the circuit evaluation function. In 2007 Rasmussen *et al.* produced a faster automated solution by having their algorithm keep track of connections that cannot be found by H-search [144].

However, before these automated results, in 2002 and 2003 Jing Yang had already weakly solved the centre openings for 8×8 and 9×9 by hand [181]. In 2005 and 2006 Noshita and Mishima *et al.* presented further manual 8×8 opening solutions [125, 132]. Figure 2.14 summarizes these results, with the omission of the single opening solved on 9×9.

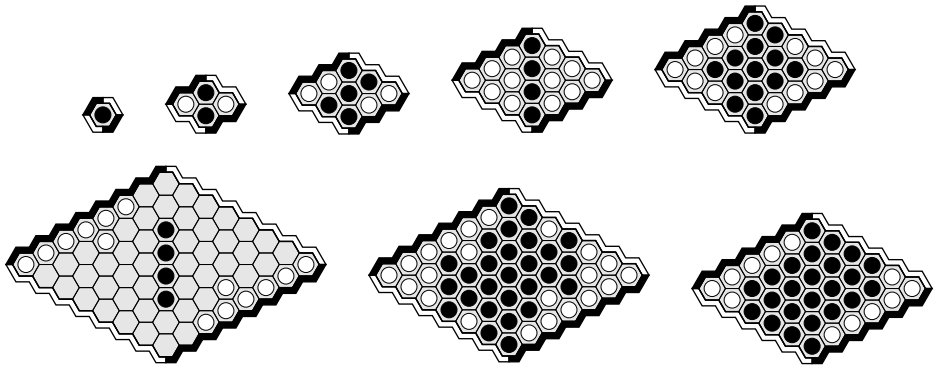


Figure 2.14: Previously solved opening moves. Colour of cell indicates winner if Black opens there. 8×8 openings were only solved by hand.

2.9.4 Automated Players

Another goal of Hex research is to develop strong automated players. Claude Shannon and E.F. Moore developed the first automated Hex player in the 1950s, an electronic circuit network which set positive charges for one player's coloured cells, negative charges for the other player's coloured cells, and then played at a certain saddle point. The computer played strong opening moves but sometimes erred in tactical situations [158]. Shannon also developed a computer to play Bird Cage, now known as Bridg-it (see §2.8). This circuit network set the resistance of one player's coloured cells to zero, the resistance of the other player's coloured cells to infinity, and then played at a cell with greatest voltage drop [64].

In 2000 Anshelevich's Hexy won the first Computer Olympiad Hex competition [9]. Hexy's evaluation function uses an augmentation of Shannon's Bird Cage circuit, in which extra wires are added which correspond to VCs found by H-search [7]. Hexy uses this evaluation function in a straightforward alpha-beta search.

Hexy's strongest competitor was van Rijswijck's Queenbee. Queenbee uses alpha-beta search with selective extensions to search deeper on important lines, as well as some basic inferior cell analysis. Its evaluation function is based on two-distance, an approximation of the shortest winning

path after opponent resistance [169, 171].

Melis' Six won the next three Computer Olympiad Hex competitions in 2003, 2004, and 2006 [80, 122, 177]. Six significantly refined Hexy's framework by improving H-search efficiency via heuristic limits, using borders as AND rule midpoints, restricting the alpha-beta branching factor, tuning the evaluation function, and pruning some dead cells [121]. Although Six only uses a truncated 2-ply alpha-beta search, it is generally considered to be a strong player on boards up to 11×11 [114].

Six's strongest competitor was Hayward *et al.*'s Mongoose. Mongoose is another refinement of the Hexy alpha-beta framework, and is superior to Six in terms of inferior cell analysis. However, Mongoose's H-search is not as strong as Six's, and Mongoose searches to odd-ply depths, resulting in worse performance than shallower even-ply depths [122]. Another competitor was Rasmussen *et al.*'s HexKriger, a learning program whose performance improved as the tournament progressed [80].

Chapter 3

Inferior Cell Analysis

When playing or solving the game of Hex, many cells can be pruned from consideration. As stated in §2.9, this pruning can be performed by checking for local patterns and is quite effective. For instance, simply colouring the captured cells of border bridge virtual connections resulted in a tenfold speedup when solving all openings on the 7×7 Hex board [83].

In this chapter we begin by generalizing previous inferior cell classes, and reformulating them in terms of CGT. We then develop several new classes of inferior cell, including captured-reversible, neighbourhood domination, induced path domination, permanently inferior, and investigate combinatorial decompositions. Lastly, we end with some notes on our implementation.

3.1 Generalized Definitions

We begin with two minor generalizations of existing inferior cell analysis.

Recall the definitions of P vulnerable and P vulnerable-by-capture from §2.9. To unify these two definitions, we define the $\overline{P}$ *carrier* of a P vulnerable cell c to be the set of $\overline{P}$ captured cells that help killer k render c dead. If no $\overline{P}$ captured set is required to kill the P vulnerable cell (*i.e.*, the original definition of P vulnerable), then the $\overline{P}$ carrier is simply the empty set.

Secondly, in all previous implementations of Hex inferior cell analysis, the fillin cells were only computed once on the original position. However, by colouring fillin and repeatedly applying the inferior cell patterns to the *reduced position*, much more fillin can be identified.

That is, for a position H, a cell set F is fillin if F partitions into cell sets $F_1, \ldots, F_t$ such that each F_j is dead or P_j captured in position $H + P_1(F_1) + \ldots + P_{j-1}(F_{j-1})$, where each P_j is either P or $\overline{P}$; this follows by simple induction. We define P *fillin* to be fillin where only P's colour is used to reduce the position.

In addition to iteration finding more fillin, it also allows us to generalize capture-domination to fillin-domination: If position $H + P(c)$ has P fillin F, then by monotonicity it follows that in position H the cell c P *fillin-dominates* all cells in F. See Figure 3.1.

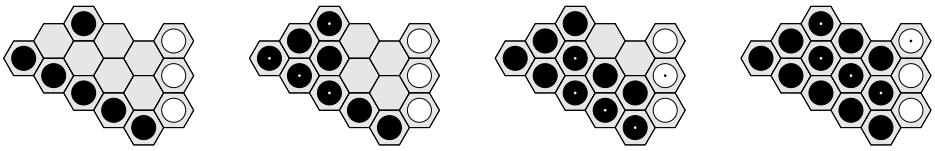


Figure 3.1: Iteratively-computed fillin. Black-colouring Black captured sets can lead to new Black captured sets being identified.

3.2 CGT Reformulation

We now examine previous Hex inferior cell results through the lens of combinatorial game theory. This will both motivate and simplify the formulation of some of our new classes of inferior cells. Our first result is that a P vulnerable cell is P reversible:

Lemma 1 *Let H be a Hex position with a cell c that is P vulnerable to $\overline{P}$ killer k with (possibly empty) $\overline{P}$ carrier X. Then c is P reversible in H, with $\overline{P}$ reverser k.*

Proof: $H \geq_P H + \overline{P}(\{k\} \cup X)$ by monotonicity. Also, $H + P(c) + \overline{P}(k) \equiv H + P(c) + \overline{P}(\{k\} \cup X) \equiv H + \overline{P}(\{k\} \cup X)$, since X is $\overline{P}$ captured and c is dead. Thus we have $H \geq_P H + P(c) + \overline{P}(k)$, satisfying the definition of P reversible. □

From now on we refer to vulnerable cells as *dead-reversible*. Note that the pruning of dead-reversible moves in Hex is stronger than the bypassing of reversible moves guaranteed by combinatorial game theory.

Our second result shows that playing in one's own captured set is equivalent to playing a pass move:

Lemma 2 *Let H^P be a Hex state where player P has a winning move to a cell c that is P captured. Then position H is a P win.*

Proof: Let F be a set of P captured cells in H, with c in F. Then $H \equiv H + P(F)$, and $H \geq_{\overline{P}} H + P(c) \geq_{\overline{P}} H + P(F)$ by monotonicity, so $H \equiv H + P(c)$. But c is a winning P move in H^P, so $(H + P(c))^{\overline{P}}$ is a P win, and $H + P(c) \equiv H$ is a P win. □

3.3 Captured-Reversible Moves

Just as knowledge of dead cells allows us to define dead-reversible cells, so knowledge of captured cells allows us to define captured-reversible cells. The latter concept is somewhat counter-intuitive since it is the opponent's move that yields the player's captured set.

Definition 1 *A cell c in position H is P* captured-reversible *if there is a cell r such that $H + \overline{P}(r)$ has P captured set F containing c.*

Lemma 3 *Let cell c be P captured-reversible in Hex position H. Then cell c is P reversible in H.*

Proof: By Definition 1, there is a $\overline{P}$ move r in H that yields P captured set F containing c. By monotonicity, $H+\overline{P}(r) \geq_{\overline{P}} H+P(c)+\overline{P}(r) \geq_{\overline{P}} H+P(F)+\overline{P}(r)$. By the definition of captured, $H+\overline{P}(r) \equiv H+P(F)+\overline{P}(r)$, so $H+\overline{P}(r) \equiv H+P(c)+\overline{P}(r)$. Thus $H \geq_P H+\overline{P}(r) \equiv H+P(c)+\overline{P}(r)$, satisfying the definition of P reversible. □

Captured-reversible cells are reversible, so they can be bypassed. However, we would like to prune them from consideration completely, as is done with dead-reversible cells. Whether pruning all captured-reversible cells always preserves position value is still an open question; however, we now show sufficient conditions which allow some pruning.

For a P captured-reversible move m with $\overline{P}$ reverser r and resulting P captured set F, we call F a P *captured-reversible carrier* of m. With respect to such selected reversers r_1, r_2 and carriers F_1, F_2 of P captured-reversible moves m_1, m_2, we say that m_1 and m_2 *interfere* if r_1 is in F_2 and/or r_2 is in F_1. The *captured-reversible graph* $G_{\gamma(H,P)}$ of position H for player P is defined as follows:

- select a set of P captured-reversible moves m_j in H,
- for each m_j, select a $\overline{P}$ reverser r_j and corresponding carrier F_j,
- vertices of $G_{\gamma(H,P)}$ correspond to the moves m_j,
- vertices are adjacent if and only if their corresponding moves interfere.

An independent vertex set in $G_{\gamma(H,P)}$ is called an *independent captured-reversible set* for player P in position H.

Lemma 4 *Let H_1 be a Hex position with an uncoloured cell c and an independent P captured-reversible set $I_1 = \{m_1, \ldots, m_n\}$, where each captured-reversible cell m_j has selected $\overline{P}$ reverser r_j and carrier F_j. Then in position $H_2 = H_1 + \overline{P}(c)$ the set $I_2 = \{m_j \in I_1 : c \notin \{r_j\} \cup F_j\}$ is an independent P captured-reversible set.*

Proof: Each m_j in I_2 is P captured-reversible in H_2 since r_j remains a legal move for $\overline{P}$, and F_j remains captured for any continuation of $H_1 + \overline{P}(r_j)$ in which all cells in F_j are uncoloured. In defining the captured-reversible graph $G_{\gamma(H_2,P)}$, we can select the same reversers and carriers for all cells in I_2 to guarantee independence. □

Theorem 1 *Let H be a Hex position with a set of dead cells D, a set of P dead-reversible cells V, and an independent P captured-reversible set I. If H^P is a P win, then either P has a winning move not in D or V or I, or $H^{\overline{P}}$ is a P win.*

Proof: Proof by contradiction. Let position H be a counterexample with the smallest number of uncoloured cells. Thus, P has some winning move in H^P, but each such move is in D or V or I,

and $H^{\overline{P}}$ is a $\overline{P}$ win. For H^P, let $W = \{m_1, \ldots, m_n\}$ be the set of winning P moves that are not in D or V. Pruning dead and dead-reversible moves cannot eliminate all winning moves, so W is a nonempty subset of I.

For some $m_j \in W$, let $T_j = H + \overline{P}(r_j)$, where r_j is the $\overline{P}$ reverser of m_j. By the definition of captured-reversible, it follows that $T_j \equiv H + P(m_j) + \overline{P}(r_j)$; see the proof of Lemma 3. Since $(H + P(m_j))^{\overline{P}}$ is a P win, then $(H + P(m_j) + \overline{P}(r_j))^P \equiv T_j^P$ is a P win. By monotonicity $T_j \geq_{\overline{P}} H$, so $T_j^{\overline{P}}$ is a $\overline{P}$ win.

T_j has fewer uncoloured cells than H, so T_j is not a counterexample to this Theorem. Thus in position T_j, for any dead cell set D_j, P dead-reversible cell set V_j, and independent P captured-reversible set I_j, P has a winning move not in D_j or V_j or I_j.

Recall that dead cells are dead in all continuations, and P dead-reversible cells are dead or P dead-reversible in all continuations where only $\overline{P}$-coloured cells are added. Thus we can select D_j, V_j such that $D_j \cup V_j \supseteq (D \cup V) \setminus \{r_j\}$. Also, we can select I_j to be the set of cells in I whose $\overline{P}$ reverser is not r_j. I_j is an independent P captured-reversible set in T_j by Lemma 4. Thus there exists a winning P move m in T_j^P that is not in D_j or V_j or I_j.

Since $T_j = H + \overline{P}(r_j)$, P move m is also winning in H^P. Thus by our assumption m is in $(D \cup V \cup I) \setminus (D_j \cup V_j \cup I_j \cup \{r_j\}) \subseteq I \setminus I_j$. Thus m is P captured-reversible with $\overline{P}$ reverser r_j in H, meaning that it is P captured in T_j. By Lemma 2, position T_j is a P win, a contradiction. □

If state $H^{\overline{P}}$ is a P win, then any legal move in state H^P is winning for P by monotonicity. Thus we can apply Theorem 1 as follows: given a Hex position and a player P for whom we are trying to find a winning move, we can identify dead cells, P dead-reversible cells and an independent P captured-reversible set, and prune all these inferior cells from consideration with the caveat that we consider at least one legal move.

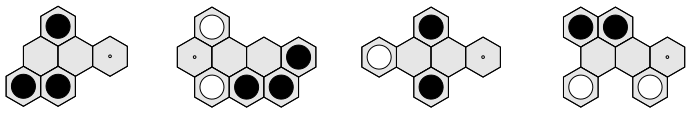


Figure 3.2: Black captured-reversible patterns. In each case a White move at the dotted cell results in the empty cells being Black captured.

Using known captured patterns, we can identify captured-reversible patterns. See Figure 3.2.

Note that all of the above captured-reversible results can be generalized to fillin-reversible results, by replacing P captured sets with P fillin. Indeed, by definition fillin-reversible contains both dead-reversible and captured-reversible. However, since we know of no local patterns that are fillin-reversible but neither dead-reversible nor captured-reversible, we will only use these two subclasses.

3.4 Neighbourhood Domination

We define the *closed neighbourhood* of a location l to be the union of its neighbourhood set $N(l)$ with itself, and this is denoted by $N[l]$.

We define the *Black graph neighbourhood* of an uncoloured cell c in a Hex position H to be the set of uncoloured cells in H whose corresponding vertices are adjacent to c's vertex in the Black graph of H; recall Figure 2.3. We define the *Black graph closed neighbourhood* of c in H to be the union of its Black graph neighbourhood and $\{c\}$. We denote these as $N_B(c)$ and $N_B[c]$ respectively. Similarly, we define the *White graph neighbourhood* and *White graph closed neighbourhood* with respect to the White graph of a position, and denote them as $N_W(c)$ and $N_W[c]$ respectively.

Definition 2 *Let c_1, c_2 be uncoloured cells in Hex position H. Then we say that c_1 P* neighbour dominates *c_2 in position H if $(N[c_1] \to \{U, P\}) \supseteq (N[c_2] \to \{U, P\})$.*

Definition 3 *Let c_1, c_2 be uncoloured cells in Hex position H. Then we say that c_1 P* graph neighbour dominates *c_2 if $N_P[c_1] \supseteq N_P[c_2]$.*

We begin by observing that neighbour domination implies graph neighbour domination:

Lemma 5 *Let c_1, c_2 be uncoloured cells in Hex position H such that c_1 P neighbour dominates c_2. Then c_1 P graph neighbour dominates c_2.*

Proof: Follows from the definition of the P Hex graph. □

Next, we prove that graph neighbour domination is a form of domination:

Theorem 2 *Let c_1, c_2 be uncoloured cells in Hex position H such that c_1 P graph neighbour dominates c_2. Then c_1 P dominates c_2.*

Proof: In the P graph G for position $H + P(c_1)$, the vertices $(N_P(c_1))_H$ form a clique. Thus c_2 must be dead in *G*, as its neighbours form a clique. Thus $H + P(c_1) \equiv H + P(c_1) + P(c_2) \geq_P H + P(c_2)$. □

Corollary 1 *Let c_1, c_2 be uncoloured cells in Hex position H such that c_1 P neighbour dominates c_2. Then c_1 P dominates c_2.*

Proof: Follows from Lemma 5 and Theorem 2. □

By the proof of Theorem 2, neighbour domination and graph neighbour domination are really just forms of fillin-domination. However, a benefit of these definitions is that, in addition to local pattern matching, simple graph-theoretic algorithms can be used to identify domination, whereas no such algorithm is known for capture-domination.

Figure 3.3: Labelling of the 4-3-2 virtual connection carrier.

We also note that domination can be used not only to prune legal moves from consideration, but also to deduce equivalences and inequalities among states. As an example, we assume we have a Hex position with a 4-3-2 as labelled in Figure 3.3, and consider domination among various probe-maintenance exchanges.

Corollary 2 *Let H be a Hex position with a Black 4-3-2 labelled as in Figure 3.3. Then $H + W(4) + B(2) \geq_W H + W(3) + B(2)$.*

Proof: In position $H + B(2)$, $\{5, 6\}$ is Black captured, and so cell 4 White neighbour dominates cell 3. □

Corollary 3 *Let H be a Hex position with a Black 4-3-2 labelled as in Figure 3.3. Then $H + W(2) + B(4) \geq_W H + W(6) + B(4)$.*

Proof: In position $H + B(4)$, $\{7, 8\}$ is Black captured, and so cell 2 White neighbour dominates cell 6. □

3.5 Induced Path Domination

Definition 4 *Let H be a Hex position with uncoloured cells c_1, c_2. Then we say that c_1 P* induced path dominates *c_2 if, for each minimal set W of uncoloured cells in H that contains c_2 and creates a winning chain if P-coloured, then W also contains c_1.*

We wish to show that induced path domination implies domination. To do so we will use *completion Hex*, which is identical to Hex except that the game terminates when there are no more uncoloured cells. Due to the planarity of Hex, the winner of completion Hex is unique, and is identical to the winner of normal Hex. A benefit of considering completion Hex is that we can easily modify strategy trees:

Definition 5 *Let H be a Hex position with uncoloured cells c_1, c_2, and suppose T_1 is a strategy tree for completion Hex whose root position is $H + P(c_1)$. Then the (c_1, c_2)* exchange tree *of T_1 is the tree obtained by exchanging c_1 and c_2 throughout T_1 (i.e., the root position becomes $H + P(c_2)$, and every move to c_2 in T_1 becomes a move to c_1).*

Lemma 6 *Let H be a Hex position with uncoloured cells c_1, c_2, and suppose T_1 is a strategy tree for completion Hex whose root position is $H + P(c_1)$. Then the (c_1, c_2) exchange tree T_2 of T_1 is a valid completion Hex strategy tree for position $H + P(c_2)$.*

Proof: By definition of a strategy tree, in T_1 each node with the losing player to move considers all legal moves, and each node with the winning player to move considers a single move. Then in T_2 the board will again be played until completion, as the roles of cells c_1, c_2 have simply been exchanged everywhere in the tree. Likewise, in T_2 each node with the losing player to move considers all legal moves, and each node with the winning player to move considers a single move. □

Now we have the tools necessary to discuss induced path domination:

Theorem 3 *Let H be a Hex position with uncoloured cells c_1, c_2, and suppose that c_1 P induced path dominates c_2. Then c_1 P dominates c_2 in H.*

Proof: We wish to prove that $H + P(c_1) \geq_P H + P(c_2)$. Assume for contradiction that $H + P(c_2)$ is a P win and $H + P(c_1)$ is a $\overline{P}$ win. The same must be true in completion Hex, so let T_2 be the P-winning strategy tree for position $H + P(c_2)$, and let T_1 be the (c_2, c_1) exchange tree of T_2. Select a $\overline{P}$-winning leaf node l_1 in T_1, and let l_2 be the corresponding leaf in T_2. Since l_2 must be P-winning, and since l_1, l_2 can only differ on the colouring of c_1 and c_2, then it must be the case that l_2's P winning path(s) require cell c_2. However, in all continuations of H, all P winning paths that use c_2 must require c_1. This implies that both c_1 and c_2 are P-coloured in l_2, and so both c_1 and c_2 are P-coloured in l_1. Thus l_1 and l_2 are identical, a contradiction. □

Unfortunately induced path domination does not produce new pruning, because of the following observation:

Lemma 7 *Let H be a Hex position with uncoloured cells c_1, c_2. If c_1 P induced path dominates c_2, then c_2 is P dead-reversible with $\overline{P}$ killer c_1.*

Proof: By the definition of induced path domination, c_2 is dead in position $H + \overline{P}(c_1)$ since it can no longer be on any minimal winning paths. □

Note that the converse of Lemma 7 holds for dead-reversible cells with empty set carriers, but does not necessarily hold for dead-reversible cells with non-empty carriers. Despite its pruning redundancy, induced path domination is useful for deducing position equivalences and inequalities. As with neighbourhood domination, we illustrate this fact using probe-maintenance exchanges of a 4-3-2 VC:

Corollary 4 *Let H be a Hex position with a Black 4-3-2 labelled as in Figure 3.3. Then $H + W(2) + B(3) \equiv H + W(5) + B(3)$.*

Proof: In position $H + B(3)$, $\{6, 7\}$ is Black captured, and so cells 2 and 5 White induced path dominate each other. □

Corollary 5 *Let H be a Hex position with a Black 4-3-2 labelled as in Figure 3.3. Then $H + W(1) + B(4) \geq_W H + W(3) + B(4)$.*

Proof: In position $H + B(4)$, $\{7, 8\}$ is Black captured, and so cell 1 White induced path dominates cell 3. □

Appendix A applies these neighbourhood domination and induced path domination 4-3-2 results to prune 4-3-2 probes.

3.6 Permanently Inferior Cells

We now introduce a new kind of fillin: permanently inferior cells. Unlike with captured sets, the strategy for maintaining permanently inferior cells extends beyond the set of cells that are coloured.

Definition 6 *Let H be a Hex position with a set $C \supseteq \{c_1, c_2\}$ of uncoloured cells such that c_1 is P dead-reversible to c_2, and each cell in $C \setminus \{c_2\}$ is $\overline{P}$ dead-reversible to a killer and carrier both contained within C. Then we say that c_1 is P* permanently inferior*, and that $C \setminus \{c_1\}$ is its corresponding* carrier.

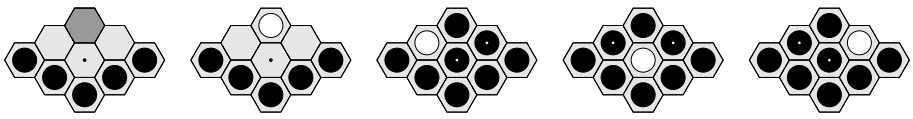


Figure 3.4: A Black permanently inferior pattern. The dotted cell is Black dead-reversible to the shaded cell. The three unshaded cells are each White dead-reversible to the shaded cell, with the other two unshaded cells being the killer's carrier. Thus Definition 6 is satisfied, and so by Theorem 4 the dotted cell can be Black-coloured without changing the position's value.

Figure 3.4 illustrates a Black permanently inferior pattern and its maintenance strategy against all possible White probes. Figure 3.5 shows the other two known Black permanently inferior patterns.

Theorem 4 *Let H be a Hex position with a P permanently inferior cell c_1. Then $H \equiv H + P(c_1)$.*

Proof: By monotonicity, $H \geq_{\overline{P}} H + P(c_1)$, so we need only prove that $H + P(c_1) \geq_{\overline{P}} H$. So suppose that $\overline{P}$ has a winning strategy on H. Then $\overline{P}$ has a winning strategy for H in which $\overline{P}$ never plays a dead or $\overline{P}$ dead-reversible cell. Let c_2 and C be as in Definition 6. Notice that the uncoloured cells of $C \setminus \{c_2\}$ remain $\overline{P}$ inferior in any continuation from H in which $\overline{P}$ has not coloured any cell of $C \setminus \{c_2\}$; that is, they will each remain $\overline{P}$ dead-reversible or become dead. It follows that if $\overline{P}$ ever plays in C, then their first such move is to c_2, at which point c_1 is dead. Thus

$\overline{P}$'s winning strategy for H never requires them to play at c_1, and so their winning strategy also applies to position $H + P(c_1)$. □

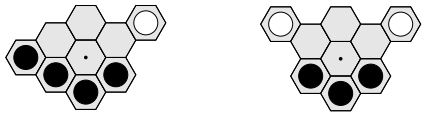


Figure 3.5: Two more Black permanently inferior patterns. In each case, colouring the dotted cell Black does not alter a position's value.

Thus permanently inferior cells are a new type of fillin, and so we can generalize the definitions of fillin, P fillin, fillin-domination, and so on in the expected manner. The permanently inferior pattern shown in Figure 3.4 is quite common, as colouring a border bridge's captured carrier often creates this pattern.

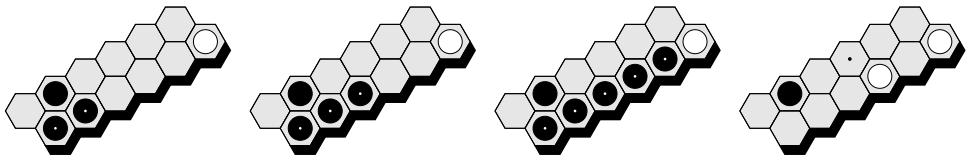


Figure 3.6: Fillin strategy conflict. Black has a border bridge with captured carrier. From this, Black deduces permanently inferior fillin, and then two more Black captured cells. White can play a cell intersecting both the permanently inferior carrier and a captured set. Since the permanently inferior cell was deduced first, its corresponding strategy must be followed (*i.e.*, the dotted cell should be played).

Because the strategy for permanently inferior cells extends beyond the coloured cell, the winning strategy on the original board cannot be a straightforward (*i.e.*, disjoint) combination of the fillin-reduced board strategy with the permanently inferior strategy. Indeed, even among the fillin patterns themselves there could be a conflict regarding the strategy on certain cells. Such conflicts can be resolved so long as the fillin deduction order is recalled, and the strategy of the earliest applicable fillin is applied. See Figure 3.6.

Theorem 5 *Let H be a Hex position with P fillin F that partitions into cell sets $F_1, \ldots, F_t$ such that each F_j is dead, P captured, or P permanently inferior in position $H + P_1(F_1) + \ldots + P_{j-1}(F_{j-1})$. Then P can maintain F on its carrier (i.e., the union of the F_j carriers) by responding to each $\overline{P}$ probe of F's carrier with the move recommended by the earliest-deduced F_j that demands a response.*

Proof: Any P fillin F_j — dead, captured, or permanently inferior — that demands a response, always produces a move that kills the probing $\overline{P}$-coloured cell in position $H + P_1(F_1) + \ldots + P_{j-1}(F_{j-1})$. Thus the result of any such exchange is equivalent to recolouring the $\overline{P}$-coloured cell with P's colour, assuming all previously deduced P fillin is maintained. Further, extra P-coloured

cells can never obstruct a P strategy, so any P fillin successor of F_j must still have a valid strategy. □

In Appendix B we apply this new fillin to produce an efficient and explicit handicap strategy for Hex. Permanently inferior patterns can also help prove the equivalence of two moves in the acute corner:

Corollary 6 *Let H be a Hex position with an acute corner as in Figure 3.7. Then $H + B(a) \equiv H + B(b)$.*

Proof: In position $H + B(a)$, cell b is dead, so $H + B(a) \equiv H + B(\{a, b\}) \geq_B H + B(b)$. In position $H + B(b)$, cell a is Black permanently inferior, so $H + B(b) \equiv H + B(\{a, b\}) \geq_B H + B(a)$. □

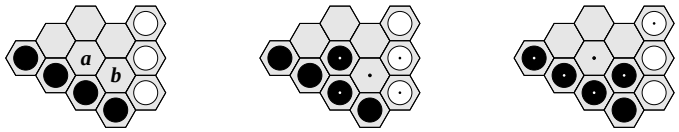


Figure 3.7: Acute corner cell equivalence. If Black claims cell a, then cell b is dead. If Black claims cell b, then cell a is Black permanently inferior.

3.7 Combinatorial Decompositions

3.7.1 Chain Decompositions

Thus far we have discussed inferior cells that can be pruned. Now we examine combinatorial decompositions of Hex positions. In particular, we focus our attention on combinatorial decompositions created by chains partitioning the uncoloured regions of the board:

Definition 7 *Let H be a Hex position, and let G be the (Black or White) Hex graph of the initial Hex board. Delete all vertices in G corresponding to elements of chains in H. Then the resulting graph is called the* chain deleted Hex graph *of H.*

Note that the set of deleted vertices includes all borders, and thus the resulting graph is identical regardless of whether G is the Black or White Hex graph. Thus the chain deleted Hex graph is well-defined.

Definition 8 *Let H be a Hex position and let G be its chain deleted Hex graph. Suppose G has components $U_1, \ldots, U_k$ ($k \geq 0$). Then these components correspond to a partitioning of the uncoloured cells in H, and so we call each U_i an* uncoloured component *of H.*

Given a Hex position H, we define its *chain component graph* as follows:

- The chain component graph is bipartite, with vertices in one part corresponding to chains of H, and vertices in the other part corresponding to uncoloured components of H.
- The vertex corresponding to chain C_i is adjacent to the vertex corresponding to uncoloured component U_j if and only if $N(C_i)$ intersects U_j.

Definition 9 *Let H be a Hex position. We call a set of uncoloured cells an* uncoloured region *of H if it is the union of one or more of H's uncoloured components.*

Definition 10 *Let H be a Hex position with chain C and region R. We say that C and R are* adjacent *in H if C is adjacent to one or more of R's uncoloured components in the chain component graph of H.*

Definition 11 *Let H be a Hex position with chain C and region R. Chain C is* internal *to R if it does not contain a border, and it is adjacent only to uncoloured components contained within R.*

Definition 12 *Let H be a Hex position with region R. Then the* chain boundary *of R is the set of chains in H that are adjacent to R and not internal to R.*

These concepts are illustrated in Figure 3.8. For instance, the region defined by uncoloured component 6 has one internal chain and seven boundary chains, while the region defined by uncoloured components 5–7 has eight internal chains and eight boundary chains.

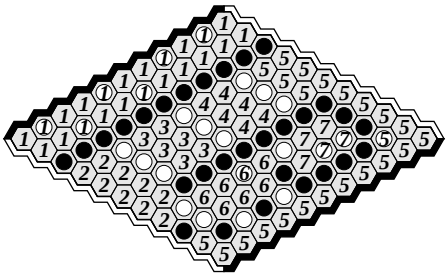


Figure 3.8: A Hex position with seven uncoloured components. The label of an uncoloured cell indicates its membership among the uncoloured components. The label of a coloured chain indicates that it is an internal chain of the region defined by the corresponding uncoloured component.

Informally, the chain boundary of a region is the set of chains through which the region's uncoloured cells can interact with the rest of the board. In other words, the rest of the board only perceives what occurs in this region by how it affects connections between its boundary chains. We now express this formally:

Lemma 8 *Let H be a Hex position with region R, and let $W = \{l_1, \ldots, l_j\}$ be a winning path in H such that $l_x \in R$ for some $1 \leq x \leq j$. Let l_y be the first successor of l_x in W that is neither uncoloured, nor an element of an internal chain of R. Then l_y is an element of a boundary chain of R.*

Proof: First we note that l_y is well-defined since l_j is a boundary, and so $x < j$ since l_x is uncoloured, and also l_j is not uncoloured nor an internal chain of any region by Definition 11. Thus $x < y \leq j$, and the subpath $W_x = (l_x, \ldots, l_{y-1})$ can only be composed of uncoloured cells and elements of internal chains of R.

By the definition of uncoloured components, consective uncoloured cells of W_x must remain in the same uncoloured component. By the definition of chains, any consecutive same-coloured cells must be part of the same chain. Since W is a winning path, transitions between opposite-coloured cells are impossible. Finally, since all internal chains of R are only adjacent to uncoloured components in R, then any transitions from uncoloured cells to coloured cells or vice-versa remain confined to elements of R and internal chains of R. Thus W_x is restricted to elements of uncoloured components and internal chains of R.

If l_{y-1} is coloured, then l_y must be the same colour, since W is a winning path. This is impossible, as l_y would then be an element of the same chain as l_{y-1}, contradicting the selection of l_y. Thus l_{y-1} is uncoloured and an element of R, while l_y is coloured, and not an element of an internal chain of R. So by definition the chain containing l_y is a boundary chain of R. □

By symmetry, the same argument holds for the last predecessor of l_x in W that is neither uncoloured, nor an element of an internal chain of R.

Definition 13 *Let H be a Hex position with region R and let C_x, C_y be distinct same-colour chains in R's chain boundary. Then an* interboundary connection *in R with endpoints C_x, C_y is a monochromatic path connecting C_x, C_y whose non-endpoint locations are each contained within R or internal chains of R.*

Definition 14 *Let H be a Hex position with region R. We say that completions R_1, R_2 of R are* interboundary equivalent *if for all pairs of same-colour R bounding chains C_x, C_y, an interboundary connection in R with endpoints C_x, C_y exists in R_1 if and only if it exists in R_2.*

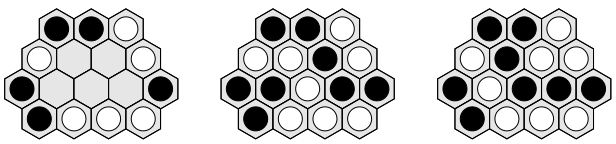


Figure 3.9: A region and two interboundary equivalent completions.

Lemma 9 *Let H_1 be a Hex position with region R, and let H_2 and H_3 be completions of H_1 that are identical on $(H \to U) \setminus R$ and interboundary equivalent on R. Then the winner of H_2 is the winner of H_3.*

Proof: Assume for contradiction that this is not the case. So there exists a P winning path W that exists in H_2, but no P winning path exists in H_3. Since H_2 is identical to H_3 on all cells outside of R, it must be the case that W intersects R. By Lemma 8 we know that any element of R in W is bounded by two elements of boundary chains of R. Since H_3 is interboundary equivalent to H_2, we can substitute its own interboundary connections for each of these bounded segments, thereby producing a P winning path on H_3. Contradiction. □

In other words, such regions allow us to combinatorially decompose strategies. Thus we define a *chain decomposition* in a Hex position to be a two-tuple of a region and its corresponding chain boundary.

3.7.2 Generalized Chain Decompositions

We now show that we can further decompose some uncoloured components while still maintaining the desired decomposition properties.

Definition 15 *Let c_1, c_2 be uncoloured neighbours in Hex position H. If cells c_1, c_2 are also neighbours of some Black chain C_B and some White chain C_W, then we say that they are* double chain adjacent.

Figure 3.10: An opposite-colour bridge.

For instance, the carrier cells of an opposite-colour bridge — that is, a bridge VC whose endpoints are opposite-coloured cells — are double chain adjacent. See Figure 3.10.

Lemma 10 *Let H_1 be a Hex position with double chain adjacent cells c_1, c_2, and let H_2 be a completion of H_1 with a P winning path. Then there exists a P winning path in H_2 where c_1 and c_2 are not consecutive locations.*

Proof: If a P winning path in H_2 has c_1, c_2 as consecutive locations, then a new P winning path can be constructed by rerouting the path through the elements of a P-coloured chain that neighbours both c_1 and c_2 in H_1. □

Theorem 6 *Let us redefine the chain deleted Hex graph of a position H so that edges between double chain adjacent cells are also deleted. Then Lemma 9 still holds.*

Proof: All definitions except Definition 7 remain the same. Lemma 8's proof is no longer valid, since there can exist winning paths that avoid boundary chains by containing consecutive double

chain adjacent cells. However, we can apply Lemma 10 to produce a winning path without consecutive double chain adjacent cells, and Lemma 9's interboundary connection replacement proof is still valid on this new winning path. □

The new chain deleted Hex graph has the same vertex set as the original, but a subset of the edges. Thus the number of uncoloured components can only increase, and thus the number of chain decompositions can only increase. We generalize the definition of chain decompositions accordingly.

The simplest chain decomposition is one where the region is the set of all uncoloured cells. In this case the chain boundary is the set of four chains containing borders; all other chains are internal by definition. This particular chain decomposition cannot help us decompose our strategy, but it does imply the PSPACE-completeness of determining the existence of a first player or second player interboundary connection strategy.

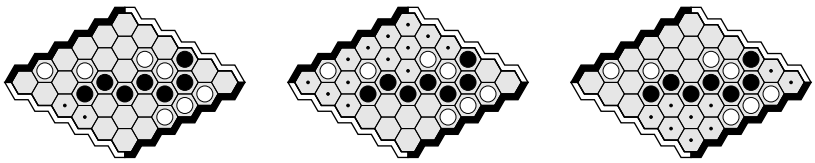


Figure 3.11: A Black split decomposition and two corresponding regions.

Another simple chain decomposition is the split decomposition, in which three boundary chains contain borders and one boundary chain does not contain a border.

Definition 16 *A* P split decomposition *is a chain decomposition whose chain boundary is composed of two* $\overline{P}$ *chains, each containing a distinct* $\overline{P}$ *border, and two* P *chains, exactly one containing a single* P *border.*

See Figure 3.11. Note that in this example, the split decomposition uses the generalized results obtained via double chain adjacent cells. Since only the interboundary connections of this region matter, then either $\overline{P}$ has a strategy to form a winning chain in this region, or else $\overline{P}$ can simply let P claim all of its uncoloured cells. We formalize and generalize this concept in the next section.

3.7.3 Dead and Captured Regions

Lemma 11 *Let* H *be a Hex position with region* R*, such that* R*'s chain boundary contains at most one Black chain and at most one White chain. Then all cells in* R *are dead.*

Proof: Since interboundary connections require two distinct chain endpoints, then no interboundary connections are possible in R. Hence, all completions of R are interboundary equivalent, and thus by Lemma 9, changing the colour of any cell in R does not change the winner of a completion of H. Thus all cells in R are dead. □

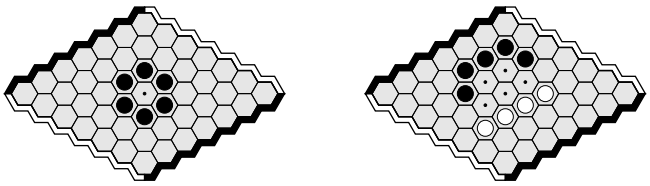


Figure 3.12: Two dead chain decomposition regions.

See Figure 3.12. Lemma 11 is subsumed by the previous knowledge that cells isolated by clique cutsets are dead, since the uncoloured neighbours of a chain form a clique. However, this result leads naturally to the concept of captured chain decomposition regions:

Theorem 7 *Let H be a Hex position with region R, and assume player P has a second player strategy on R that prevents $\overline{P}$ from making any interboundary connections. Then R is P captured.*

Proof: If P follows this second player strategy on R, then any $\overline{P}$-coloured cells in R are not on any interboundary connection, and thus cannot be on any $\overline{P}$ winning path. It follows that any $\overline{P}$-coloured cells must be dead in the leaf of this strategy tree on R, which satisfies the definition of a P captured set. □

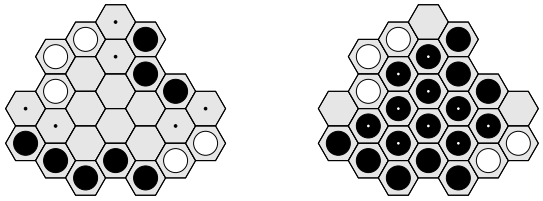


Figure 3.13: A Black captured chain decomposition region.

See Figure 3.13. Since a P captured set can be P-coloured, then it follows that a second player strategy on region R to prevent any $\overline{P}$ interboundary connections is equivalent to a second player strategy on region R to maximize the P interboundary connections. Also, any P move that creates a P captured region R capture-dominates all uncoloured cells in R.

3.7.4 Star Decomposition Domination

Of course, not all chain decomposition regions are settled. We consider the simplest unsettled case, where both players have a first player strategy to capture the region. The first move of such a strategy captures the region, and so capture-dominates all other moves in the region. It follows that each player need consider only one move in the region, and the first to play their corresponding move captures the entire region. Due to the strategic resemblance to the surreal number $* = \{0|0\}$, we call such a chain decomposition a *star decomposition*.

Definition 17 *A chain decomposition (R, C) is a* star decomposition *if each player has a move in R that captures the region for themself.*

Unlike with captured regions, the move that creates a star decomposition need not dominate all cells inside the region. However, some domination can be deduced by determining when additional coloured cells in the region do not affect either player's strategy.

Theorem 8 *Let H be a Hex position such that a P move m yields a star decomposition with corresponding region R. Let $Q \subseteq R$ be a set of cells on which $\overline{P}$ has a first player strategy to prevent all P interboundary connections in R. Then in H, m P dominates every cell in $R \setminus Q$.*

Proof: If we can prove that $H + P(m) \equiv H + P(m) + P(R \setminus Q)$, then the result follows by monotonicity. Since the cells in $R \setminus Q$ are within the star decomposition, they can only affect interboundary connection strategies within R.

If P is the first to play inside the star decomposition region, then R becomes P captured, so the P-colouring of cells $R \setminus Q$ did not alter the position's value as they would be assigned to P in either case.

If $\overline{P}$ is the first to play inside R, then Q becomes $\overline{P}$ captured since P's additional cells do not obstruct $\overline{P}$'s strategy, and this strategy still prevents all P interboundary connections by definition. Thus the P-coloured set $R \setminus Q$ is dead, or equivalently, by Lemma 9 the resulting completion of R is interboundary equivalent to $\overline{P}$-colouring all of R. Thus once again the P-colouring of $R \setminus Q$ did not alter the position's value. □

In other words, $\overline{P}$'s star decomposition strategy is not adversely affected by P-colouring $R \setminus Q$.

Corollary 7 *Let H be a Hex position such that a P move m creates a star decomposition with region R. Let $I \subseteq R$ be the set of cells intersecting all of $\overline{P}$'s first player strategies to prevent all P interboundary connections on R. Then in H, m P dominates every cell in $R \setminus I$.*

Proof: By repeated application of Theorem 8 to every such first player strategy for $\overline{P}$. □

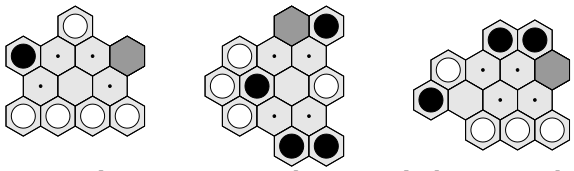


Figure 3.14: Star decomposition domination. In each case a Black move to the shaded cell forms a star decomposition and dominates a Black move to any of the dotted cells.

Star decompositions arise frequently, allowing us to prune several moves that cannot otherwise be pruned. See Figure 3.14.

3.8 Algorithms

Thus far we have established the theory for several new types of inferior cell, as well as chain decomposition properties. We now discuss how these ideas can be efficiently incorporated into our Hex program.

3.8.1 Local Patterns

By far the simplest extension is simply adding new local inferior cell patterns to the existing library. This method was adopted for captured-reversible and permanently inferior cells, as well as for the common 4-3-2 VC star decomposition domination shown in Figure 3.14.

Since the acute corner often leads to much fillin and other deductions — see for instance Figures 3.1 and 3.7, as well as Appendix A — several such patterns were added as well.

Some inferior cell patterns in the original set generated by van Rijswijck are not minimal, in that they specify the colour of a cell that is irrelevant to the conclusion of inferiority. The pattern library was revised (by hand) to reduce such inefficiencies.

In an attempt to fix these deficiencies, an automated algorithm was developed with Laurie Charpentier and Broderick Arneson to compute all minimal dead patterns of a bounded size. The current implementation is somewhat slow: it requires roughly one month to identify all dead patterns of radius at most two. However, new minimal dead cell patterns were identified, so further work in this area could be beneficial.

3.8.2 Graph-Theoretic Inferior Cell Analysis

As mentioned earlier, neighbourhood domination and induced path domination are subsumed by existing inferior cell analysis, but they provide the benefit that they can be identified using simple graph-theoretic properties. Likewise, dead and dead-reversible cells can be identified via graph-theoretic algorithms, and we added such algorithms to supplement the local patterns.

For instance, any cell isolated by a clique cutset is dead, and the uncoloured neighbours of a chain form a clique in the same-coloured Hex graph. Thus a basic check for dead cells is whether some subset of the neighbours of a chain forms a cutset. A simple linear-time search on the graph can check this for each chain, and since there are at most a linear number of chains in the size of the board, then in quadratic time we can find all such dead cells.

Likewise, if all but one of the neighbours of a cell are neighbours of some P chain, then it follows that the cell is P dead-reversible, since a $\overline{P}$ move to the exception neighbour results in a clique cutset separating that cell. Such reasoning need not be restricted to the board neighbourhood, but can also be applied to the cell's neighbourhood in the Black and White Hex graphs.

Since the dead, dead-reversible, and neighbourhood domination algorithms all rely on knowledge of chains' uncoloured neighbours, this information need only be computed once. For instance, a cell is dead-reversible if the set subtraction of a chain's neighbourhood and a cell's neighbourhood

has size one, and one cell dominates another if the set subtraction of their neighbourhoods is the empty set.

Lastly, two dead-reversible cells that are each other's reverser form a captured set so long as their respective carriers do not conflict. Thus combining the graph-theoretically identified dead-reversible cells with existing dead-reversible local patterns results in more captured cells being identified as well.

3.8.3 Backing Up Domination

As stated earlier, we generalize capture-domination to fillin-domination, recursively identify and colour fillin, and obtain captured regions from chain decompositions. To identify all these different types of fillin-domination, we simply compute the fillin for successor states, and return the resulting domination deductions to the immediate predecessor. This is particularly natural in various forms of search, such as depth-first search, alpha-beta search, and so on; it can be more costly in other settings.

We use any returned domination information to produce a directed domination graph, and select a set of dominating representatives, pruning all other cells from consideration. The set of dominating representatives is currently selected heuristically, and so not necessarily a minimum set.

3.8.4 Decomposition Algorithms

The identification of dead regions is already performed by the aforementioned chain clique cutset algorithm, so we are predominantly concerned with captured regions and star decomposition domination. However, both of these require knowledge of strategies that prevent opponent interboundary connections, for which there is no known methodology. Furthermore, identifying such strategies is PSPACE-complete.

However, blocking all opponent interboundary connections is equivalent to maximizing one's own interboundary connections, and connection strategy information is available via H-search. Thus we restrict our identification of decomposition regions as follows:

Definition 18 *We say that two opposite-coloured chains in a Hex position* touch *if they are neighbours, or if they contain the endpoints of an opposite-colour bridge.*

Definition 19 *A* four-sided decomposition *is an uncoloured region whose chain boundary is a* 4*-cycle of touching chains.*

In a four-sided decomposition with region R, a VC to connect two of the same-coloured bounding chains is a second player strategy to prevent the opponent from producing any interboundary connections in R. In other words, the existence of a VC for player P whose carrier is a subset of R and whose endpoints are the bounding P chains implies that R is P captured. Likewise, if both

players have interboundary connection SCs in a four-sided decomposition, then it is a star decomposition.

Touching chains can be identified using chain neighbourhoods and a simple linear-time scan for opposite-colour bridges. Given this bipartite graph of Black and White chains, with edges connecting touching chains, 4-cycles can be found efficiently, especially when confined to pairs of chains with connecting VCs/SCs. Since the carrier of the VC/SC is known, we need only check that it is a subset of the region defined by the 4-cycle.

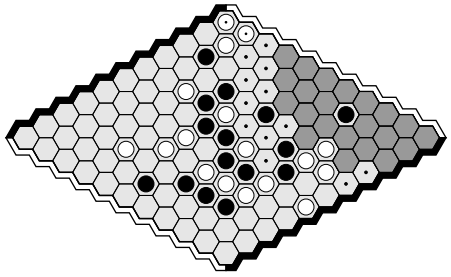


Figure 3.15: Captured non-chain decomposition: White maintaining the shaded VC (and the border bridge) creates a clique cutset in the White Hex graph, and so captures the shaded and dotted cells.

More generally, if two P-coloured chains are VC-connected, and if the union of their neighbours forms a cutset that isolates the VC carrier, then the VC carrier is P captured. This follows since any probes of the VC must be dead once the connection is complete, and in fact this simple algorithm can also identify captured combinatorial decompositions that are not chain decompositions. See Figure 3.15.

Star decomposition domination is identified similarly, except SCs internal to the region must be identified for both chain boundary pairs. The identification of such SC carriers allows for fillin by the proof of Theorem 8, and the corresponding domination information can be passed back to the predecessor state.

H-search is costly, but if it is being performed anyhow, the resulting connection strategies might as well be used to identify more captured sets and dominated moves. This marks the first time that VC/SC information is used for inferior cell analysis.

Chapter 4

Connection Strategy Algorithms

As mentioned in §2.9.2, H-search provides many benefits to Hex programs, such as mustplay pruning and early search termination. Recall that H-search is incomplete, while generalized H-search is far too slow. Thus a natural question is whether there exist extensions of H-search that identify more connections without increasing the time cost by too great a factor.

Analyzing the performance of H-search is rather difficult, since the number of iterations depends on the number of deduced connections, and thus varies greatly according to both position connectivity and algorithm completeness. Thus we focus our attention on the computational complexity of each deduction rule, rather than the algorithmic framework as a whole.

Deduction rule	**Computational complexity**
AND rule	$O(n^3 l_V^2)$
OR-3 rule	$O(n^2 l_S^3)$
OR-4 rule	$O(n^2 l_S^4)$

Table 4.1: Computational complexity of H-search deduction rules.

In our analysis we assume that H-search is implemented as in Six (Melis' automated Hex player; see §2.9.4), with heuristic limits on the number of VCs/SCs between each pair of locations that are used by the deduction rules, and with the VCs/SCs sorted by carrier size so that preference is given to connection strategies with smaller carriers. We also assume that the OR rule has a limit on the number of SCs it may consider at one time (excepting the OR-all rule). We denote the number of locations by n, and the heuristic limits for VCs and SCs by l_V and l_S respectively. The computational complexity of existing deduction rules is shown in Table 4.1. We note that for Six, the parameters l_V and l_S are ten and twenty-five respectively [121].

4.1 Partition Chains and the Crossing Rule

Anshelevich proved the incompleteness of H-search using the SC shown in Figure 2.12. We call SCs of this form *braids*, as the substrategies are intertwined together; that is, a braid is an SC that

decomposes into VCs between each of $\{a, x\}$, $\{b, y\}$, and SCs between each of $\{x, y\}$, $\{x, b\}$, $\{a, y\}$. Thus a naive algorithm for identifying braids is the following:

1. Select four non-opponent coloured locations, denoted as a, b, x, and y.
2. For each such a, b, x, y, find the following connection strategies, ensuring their carriers are pairwise disjoint:
   - A VC with endpoints $\{a, x\}$ and carrier C_1.
   - A VC with endpoints $\{b, y\}$ and carrier C_2.
   - A SC with endpoints $\{x, y\}$ and carrier C_3.
   - A SC with endpoints $\{x, b\}$ and carrier C_4.
   - A SC with endpoints $\{a, y\}$ and carrier C_5.
3. Whenever five such connection strategies are found, conclude that there exists an SC with endpoints $\{a, b\}$, carrier $C_1 \cup \cdots \cup C_5 \cup \{x, y\}$, and key x (or key y).

This naive braid deduction rule is $O(n^4 l_V^2 l_S^3)$, and thus far slower than all previous deduction rules. In the remainder of this section we develop the crossing rule, a deduction rule to identify some braids far more efficiently.

4.1.1 Partition Chains

To describe the crossing rule, we first need to describe partition chains of connection strategies.

Definition 20 *Let V be a connection strategy with endpoints $\{p_1, p_2\}$ and carrier C. Then a* partition chain *of V is a chain Z for which there exists a partition C_1, C_2 of carrier C such that the following holds, with possible relabelling of V's endpoints:*

- *C_1 is the carrier of a VC from Z to p_1.*
- *If V is a VC, then C_2 is the carrier of a VC from Z to p_2.*
- *If V is a SC, then C_2 is the carrier of a SC from Z to p_2.*

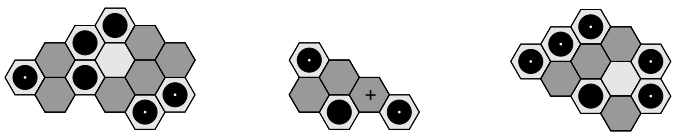


Figure 4.1: A VC and SC with partition chains and a VC with none.

Thus a partition chain decomposes a connection strategy into two disjoint connection strategies. See Figure 4.1. We now describe the *PC algorithm*, which computes partition chains for each connection strategy C, denoted as $PC(C)$, and works in parallel with H-search:

1. Let V be a base case connection strategy. Then $PC(V) = \emptyset$.

2. Let V be a VC computed via the OR rule from SCs $S_1, \ldots, S_k$. Then $PC(V) = \emptyset$.

3. Let S be an SC with key k computed via the AND rule from VCs V_1, V_2 with uncoloured midpoint k. Then $PC(S) = PC(V_1) \cup PC(V_2)$.

4. Let V be a VC computed via the AND rule from VCs V_1, V_2 with chain midpoint Z. Then $PC(V) = PC(V_1) \cup PC(V_2) \cup \{Z\}$.

We now prove the correctness of the PC algorithm:

Lemma 12 *Let V be a connection strategy with endpoints $\{p_1, p_2\}$ and carrier C. Then any chain Z in $PC(V)$ is a partition chain of V.*

Proof: Proof by induction: The statement holds vacuously for any connection strategy V for which $PC(V)$ is the empty set. Thus by the definition of the PC algorithm, the statement holds for all base case connection strategies and VCs computed via the OR rule. Thus let V be a connection strategy deduced via the AND rule from VCs V_1, V_2.

If V is a VC, then its midpoint in the AND rule deduction is a chain Y, and so $PC(V) = PC(V_1) \cup PC(V_2) \cup \{Y\}$. For partition chain $Z = Y$, the statement clearly holds for a partition of C into the carriers of V_1 and V_2.

If $Z \in PC(V_1)$, then V_1 must be computed via the AND rule since it is a VC with a non-empty partition chain set. By the induction hypothesis, the carrier of V_1 can be partitioned by Z into VCs A and B, where A is a VC with endpoints $\{p_1, Z\}$ and B is a VC with endpoints $\{Z, Y\}$. By the AND rule, combining B with V_2 over midpoint Y forms a VC V_3 with endpoints $\{Z, p_2\}$, and thus the statement holds for Z with a partition into the carrier of A and the carrier of V_3. By symmetry, the statement holds for any $Z \in PC(V_2)$. The proof is similar if V is an SC deduced by the AND rule, so the result follows from mathematical induction. □

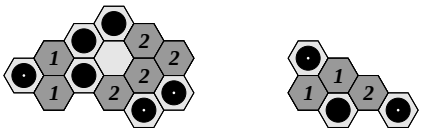


Figure 4.2: Illustrating Lemma 12. The carrier of a connection strategy can be partitioned into two connection strategies using a partition chain as an intermediate endpoint.

It is an open question whether the PC algorithm can be generalized to identify more partition chains. We now apply Lemma 12 to produce the crossing rule.

4.1.2 The Crossing Rule

Observe in Figure 4.3 that if the endpoints $\{a, b\}$ of a braid are same-coloured chains, then the braid 'untangles' into three pairwise carrier-disjoint SCs with uncoloured endpoints $\{x, y\}$ such that two of these SCs have at least one distinct partition chain. We now formally prove that finding two uncoloured cells with three such SCs is sufficient to conclude the existence of an SC between pairs of distinct SC partition chains.

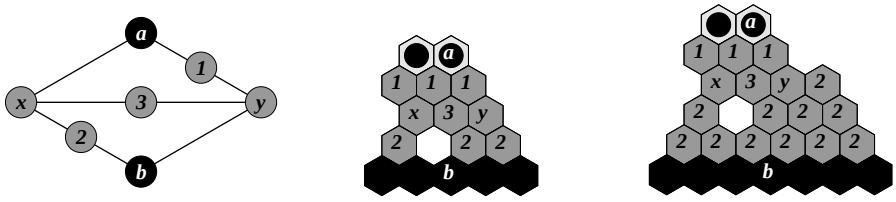


Figure 4.3: Crossing Rule SCs. For $j = 1, 2, 3$, cells labelled j form carrier C_j of SC S_j with endpoints $\{x, y\}$, where S_1 and S_2 have distinct partition chains. By Theorem 9 we conclude the existence of an SC with endpoints $\{a, b\}$.

Theorem 9 *Consider a Hex position with SCs S_1, S_2, S_3 with pairwise disjoint carriers C_1, C_2, C_3 each with uncoloured endpoints $\{x, y\}$. Further, assume that both $Z_1 = PC(S_1) \setminus PC(S_2)$ and $Z_2 = PC(S_2) \setminus PC(S_1)$ are nonempty. Then for any chains z_1, z_2 in Z_1, Z_2, there exists an SC with endpoints $\{z_1, z_2\}$ whose carrier is $\{x, y\} \cup C_1 \cup C_2 \cup C_3$ and with key x or key y.*

Proof: By Lemma 12, we can partition S_1's carrier using partition chain z_1 into VC V_1 and SC W_1. Likewise, we can partition S_2's carrier using partition chain z_2 into VC V_2 and SC W_2.

If VCs V_1, V_2 have a common endpoint, without loss of generality say x, then by the AND rule there exists an SC with endpoints $\{z_1, z_2\}$, key x, and whose carrier is the union of $\{x\}$ with the carriers of V_1, V_2. This SC satisfies the claim with a smaller carrier, so clearly the statement holds for any larger carrier.

If VCs V_1, V_2 have no common endpoint, then without loss of generality we can assume that V_1 connects z_1 to x and V_2 connects z_2 to y. It follows that W_1 connects z_1 to y and W_2 connects z_2 to x. By construction the carriers of V_1, W_1, V_2, W_2 and S_3 are all pairwise carrier-disjoint, so the stated SC strategy is as follows:

1. Play key x as the first move.
2. Thereafter maintain VCs V_1, V_2 against all probes.
3. Consider the opponent's first probe of this SC that is external to V_1 and V_2:
    - If the probe is also external to W_2, then playing the key of W_2 completes the connection from z_1 to x to z_2 via V_1 and W_2.

- Otherwise, respond to the probe of W_2 with y. Note that y has two disjoint SCs to z_1 — W_1, and the AND rule deduction over x of S_3 and V_1 — and thus y has a VC to z_1. y also has a disjoint VC to z_2 via V_2, so by the AND rule this maintenance results in a VC between z_1 and z_2.

Thus in both cases there exists an SC with endpoints $\{z_1, z_2\}$, key x or key y, and carrier $\{x, y\} \cup C_1 \cup C_2 \cup C_3$. □

Note that in the first case of Theorem 9's proof, an SC with a smaller carrier is deduced via the AND rule. Since it is common for H-search implementations to discard duplicate connection strategies with superset carriers, this case is not relevant in practice as the crossing rule's output SC will be discarded.

Note that in the second case of Theorem 9's proof, the partition is symmetric with respect to uncoloured cells x and y (simply interchange the z_1, z_2 labels), and thus either x or y can be the key in this case.

Note also that Theorem 9 applies to any such z_1, z_2, and the deduced SC's carrier and key are identical for all such chain endpoint pairs. Thus only one carrier and key need to be computed, and the 'same' SC can be added to various endpoints.

Lastly, note that the crossing rule is a unique deduction rule in that its input connection strategies share no endpoints with the output deduced connection strategy. Thus in some sense the AND rule and OR rule make parallel deductions, while the crossing rule makes orthogonal deductions. This is an interesting property of partition chains, and it remains to be seen whether additional deduction rules can be produced using this information.

4.1.3 Incorporating the Crossing Rule into H-search

In order to incorporate the crossing rule into H-search, we need to incorporate the PC algorithm into H-search. Thus we need to consider both the time and space effects of incorporating the PC algorithm, as well as the computational complexity of the crossing deduction rule.

Firstly, each connection strategy currently needs to track its endpoints, carrier, and possibly key. When adding partition chains, we can choose either to track all of them, or to track only a bounded number. The former requires storage of a set of locations (similar to how the carrier is stored) despite the fact that most connection strategies will have no partition chains. The latter requires only a modest increase in our memory requirements, but forces a choice of which partition chain(s) to retain. Since memory is not a bottleneck for our Hex program, we opted for the (simpler and more complete) first solution.

Secondly, the PC algorithm will not worsen the computational complexity of H-search's base case or deduction rules:

Lemma 13 *H-search's computational complexity is not altered by incorporating the PC algorithm.*

Proof: For each base case connection strategy C, $PC(C) = \emptyset$, which is no more computationally complex than carrier initialization. For each OR rule deduced connection strategy C, $PC(C) = \emptyset$, which is no more computationally complex than its carrier union and intersection operations. For each AND rule deduced connection strategy C, $PC(C)$ is the union of input partition chain lists, which is no more computationally complex than its carrier union and intersection operations. □

By Theorem 9 the crossing rule requires the identification of two uncoloured cells with three pairwise carrier-disjoint SCs connecting them, two with distinct partition chains. In terms of board size and the heuristic limits for SCs and VCs, it follows that the crossing rule computational complexity is $O(n^2 l_S^3)$. Recall from Table 4.1 that this is comparable to existing H-search deduction rules, and roughly the square root of the naive braid deduction rule discussed in §4.1.

We also need to extend the PC algorithm to connection strategies deduced via the crossing rule; we do so in the simplest way possible:

- Let S be a SC computed via the crossing rule from SCs S_1, S_2, S_3. Then $PC(S) = \emptyset$.

Note that Lemma 12 and Theorem 9 can easily be extended to H-search with the crossing rule, as the former holds vacuously for any connection strategy with no partition chains.

Lastly, since the crossing rule applies only when connection strategies have partition chains, in practice almost all crossing rule deductions involve a border. For this reason, the crossing rule is only really useful when H-search uses border chains as midpoints of the AND rule.

4.2 Carrier Intersection on Captured Sets

Note that the connection strategy deduction rules discussed thus far — AND rule, OR rule, and crossing rule — all require the carriers of input connection strategies to be disjoint, either pairwise or collection-wise. If this restriction is relaxed and some intersections are allowed, this could increase the number of connection strategies deduced by these rules. Using this idea, we now improve H-search by using inferior cell analysis, namely by considering captured sets.

4.2.1 Key Captured Sets

As mentioned in §3.1, fillin is computed iteratively, so all (identifiable) captured cells are already coloured. Since the carriers of connection strategies are sets of uncoloured cells, carrier intersection cannot involve cells that are currently captured.

However, for SCs the player with the connection strategy is assumed to move first, and thus we can consider the captured set of its key. If this key captures a set that eliminates carrier intersection, then the connection strategy is valid:

Theorem 10 *Let V_1, V_2 be VCs for player P with carriers C_1, C_2 and endpoints $\{l_1, l_2\}$ and $\{l_2, l_3\}$ respectively. Further, assume that l_2 is uncoloured and that P-colouring l_2 P captures set $C \supseteq$*

$C_1 \cap C_2$. *Then there exists an SC for player* P *with endpoints* $\{l_1, l_3\}$, *key* l_2, *and carrier* $C_1 \cup C_2 \cup C \cup \{l_2\}$.

Proof: Let P play at uncoloured cell l_2, P-capturing set C. Then a P VC with endpoints $\{l_1, l_2\}$ exists on carrier $C_1 \setminus C$, since the P-colouring of $C \cup \{l_2\}$ cannot hinder P's V_1 strategy. Likewise, P has a VC with endpoints $\{l_2, l_3\}$ on carrier $C_2 \setminus C$. Thus by the AND rule, a VC exists following P's move at l_2, and the carrier of this VC is at most $(C_1 \cup C_2) \setminus (C \cup \{l_2\})$. □

Since the smallest possible carrier is desirable, we perform two checks when computing the AND rule:

1. If the two VC carriers do not intersect, compute the resulting SC as before.
2. If the two VC carriers intersect, check if their intersection is a subset of the key's captured set. If so, apply Theorem 10 to produce an SC.

Note that Theorem 10 can easily be extended to use P fillin instead of P captured sets.

4.2.2 Endpoint Captured Sets

This observation regarding captured sets of keys can also be made regarding captured sets of uncoloured endpoints, since a connection strategy is only concerned with forming a chain that neighbours such endpoints. In other words, assuming the endpoints are coloured does not affect the connection strategy, and may create sets of cells that act as though they are captured within the connection strategy's carrier:

Lemma 14 *Let* V *be a VC for player* P *with carrier* C *and endpoints* $\{l_1, l_2\}$. *Assume* l_1 *is uncoloured, that* P*-colouring* l_1 P *captures set* S, *and that* $C \cap S \neq \emptyset$. *Then* P *has a VC on* $C \cup S$ *with endpoints* $\{l_1, l_2\}$ *where* P *plays the corresponding capturing strategy on* S.

Proof: First assume that l_1 is P-coloured instead of uncoloured. Then set S is P captured, and V's connection strategy is not hindered by the P-colouring of S, so in this fillin-reduced position player P has a VC with endpoints $\{l_1, l_2\}$ on carrier $C \setminus S$. Thus if only l_1 is P-coloured, then P has a VC on $C \cup S$ where P plays the corresponding capturing strategy on S. But by the definition of a connection strategy, the colour of its endpoints is irrelevant, and thus the same strategy is valid when l_1 is uncoloured. □

Lemma 15 *Let* V *be a SC for player* P *with carrier* C *and endpoints* $\{l_1, l_2\}$. *Assume* l_1 *is uncoloured, that* P*-colouring* l_1 P *captures set* S, *and that* $C \cap S \neq \emptyset$. *Then* P *has a SC on* $C \cup S$ *with endpoints* $\{l_1, l_2\}$ *where* P *plays the corresponding capturing strategy on* S.

Proof: Similar to the proof of Lemma 14. □

Although Lemmas 14 and 15 may increase the carrier size, they compensate by ensuring a certain strategy on the captured set, which allows connection strategies to intersect on these sets without conflict:

Theorem 11 *Let $V_1, \ldots, V_k$ be a set of P connection strategies with a common uncoloured endpoint l, and that P-colouring l results in a P captured set S. Furthermore, assume that a connection strategy deduction rule (AND/OR/crossing) applies to $V_1, \ldots, V_k$ except that (some of) their carriers intersect on subsets of S. Then the corresponding connection strategy can be deduced, so long as the carrier is enlarged to include all of S.*

Proof: Applying lemmas 14 and 15 to each of $V_1, \ldots, V_k$ that intersects S produces a set of new connection strategies $V_1^*, \ldots, V_k^*$ whose carriers either contain S or are disjoint from S, and whose strategies play the P capturing strategy on set S in the former case. Then any carrier intersection on S is not problematic, as all the intersecting connection strategies agree on the corresponding strategy. □

As with Theorem 10, Theorem 11 can easily be extended to use P fillin instead of P captured sets. Likewise, since smaller carriers are preferable, applying Theorem 11 requires checking whether the normal deduction rule applies and, if not, checking whether the intersecting carriers can be resolved via the captured sets of one or both uncoloured endpoints.

4.2.3 Incorporating Captured Set Carrier Intersection into H-search

In order to efficiently incorporate captured set carrier intersection into H-search, we first perform a preprocessing step where we identify the Black and White captured sets for each uncoloured cell. This simply requires checking the captured-domination inferior cell patterns, and so is linear in the size of the board since the number and size of the patterns is constant.

Note that if Theorems 10 and 11 are extended to allow fillin, then this preprocessing step requires more time since fillin is computed iteratively. Thus for simplicity and efficiency, in practice we restrict ourselves to the captured set versions of these results.

Note that the added checks do not alter the computational complexity of the deduction rules, since carrier intersections and unions are already being computed; this adjustment simply increases the number of checks by a constant factor, since the intersection is now checked against the captured sets of the key and uncoloured endpoints.

Note that multiple captured sets can be applied simultaneously (*e.g.*, for instance, those of both endpoints, or an endpoint and a key) so long as the captured sets do not intersect one another, since the capturing strategies must remain disjoint and independent.

Lastly, captured set carrier intersection can be used with or without the crossing rule. Neither of these augmentations dominates the other with respect to connection strategies identified, and their combination can find even more connection strategies. For instance, an SC found by combining the

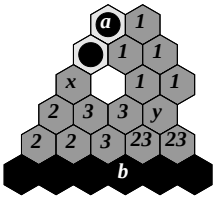


Figure 4.4: An SC found by the crossing rule combined with captured set carrier intersection. For $j = 1, 2, 3$, cells labelled j form carrier C_j of SC S_j between x, y. Cells $B = C_2 \cap C_3$ are captured if Black plays y. $PC(C_1)$ contains a and not b, and $PC(C_2)$ contains b and not a. Combining these SCs yields an SC with endpoints $\{a, b\}$, key y, and carrier $\{x, y\} \cup B \cup C_1 \cup C_2 \cup C_3$.

crossing rule with captured set carrier intersection is shown in Figure 4.4. Recall that SCs deduced from the crossing rule have two potential keys, so before deciding on a key both of their captured sets can be checked with respect to the carrier intersection. Two more SCs (and the resulting VCs) that are found via this algorithmic combination are shown in Figure 4.5.

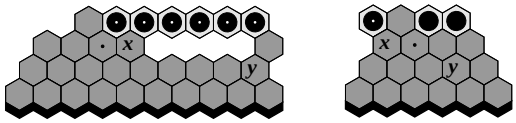


Figure 4.5: Border VCs found by combining captured set carrier intersection with the crossing rule. The newly identified SCs avoid the marked cell, and so allow the resulting VCs to be deduced via the OR rule.

4.3 Common Miai Substrategy

In the previous section we observed that connection strategy carriers can intersect on captured sets without problem, as a common strategy on the captured set can be followed. Likewise, if knowledge of the intersection set strategy were known and identical in all of the intersecting connection strategies, then once again there would be no problem with such intersection.

Although this knowledge would relax the restriction of intersecting on captured sets, the challenge of this goal is to be able to track the substrategies of each deduced connection strategy. This could be done by storing the tree of substrategy deductions for each connection strategy, but this method would require far more storage space and time. For simplicity, we restrict our attention to simple substrategies, known in Go as *miai* or twin points (*i.e.,* two moves that serve the same purpose). Common instances of miai in Hex are the uncoloured cells in a bridge VC's carrier, and the uncoloured cells of a captured set of size two.

4.3.1 Incorporating Common Miai Substrategy into H-search

In order to incorporate common miai substrategies into H-search, each connection strategy will keep track of its own list of miai. The *miai list* of a connection strategy is a list of pairs of cells in its carrier such that, when one cell in a pair is probed, the connection strategy always demands an immediate response at the other cell in the pair. In order to store miai efficiently, every possible pair of cells is assigned a unique number, and we store a list of the corresponding numbers. For algorithmic efficiency, miai lists store these numbers in increasing order.

Assuming this change in the VC/SC data structure, we can then use the algorithm in Figure 4.6 to compute the intersection of connection strategy carriers. This algorithm begins with two connection strategies, V_{in} and V_{new}, with I_{in} initialized to V_{in}.carrier(). The output two-tuple (I_{out}, V_{out}) can then be passed as input (I_{in}, V_{in}) in the next iteration (*e.g.*, in the OR rule, where the intersection is computed collection-wise, not pairwise). The output V_{out} is a valid connection strategy if I_{out} is a subset of the miai of V_{out}.

```
Algorithm: Connection Strategy Carrier Intersection
Input:          (I_{in}, V_{in}), V_{new}
Output:         (I_{out}, V_{out})
Preconditions:  V_{in} and V_{new} are connection strategies with
                I_{in} equal to V_{in}.carrier(), or
                V_{new} is a connection strategy and
                (I_{in}, V_{in}) is the output of a previous iteration
Postconditions: if I_{out} is a subset of V_{out}'s miai,
                then V_{out} is a valid connection strategy

    I_{out} = I_{in} intersect V_{new}.carrier();
    V_{out}.carrier() = V_{in}.carrier() union V_{new}.carrier();

    miaiCheckSet = V_{in}.carrier() intersect V_{new}.carrier();
    iterating through V_{in}.miai() and V_{new}.miai() lists in order
        if V_{in}.miai().current() = V_{new}.miai().current()
            V_{out}.miai().add(V_{in}.miai().current());
            V_{in}.miai().next();
            V_{new}.miai().next();
        else if V_{in}.miai().current() < V_{new}.miai().current()
            if V_{in}.miai().current() does not intersect miaiCheckSet
                V_{out}.miai().add(V_{in}.miai().current());
            V_{in}.miai().next();
        else
            if V_{new}.miai().current() does not intersect miaiCheckSet
                V_{out}.miai().add(V_{new}.miai().current());
            V_{new}.miai().next();
        end if
    end iteration
end Algorithm
```

Figure 4.6: Connection strategy carrier intersection with miai lists.

We now outline the basic invariants of this algorithm, each of which is easily verified by induction:

1. I_{out} is the intersection of all connection strategy carriers thus far.

2. V_{out}.carrier() is the union of all connection strategy carriers thus far.

3. V_{out}.miai() is an ordered subset of the miai from the input connection strategies.

4. V_{out}.miai() contains exactly the subset of miai from the input connection strategies that are common to all of its intersecting connection strategies.

5. If $I_{out} \subseteq V_{out}$.miai().union(), then there is no strategy conflict between any pairwise intersecting connection strategies (*i.e.*, any intersection is on common miai).

Thus we replace connection strategy carrier intersection with the above algorithm, obtaining a two-tuple where the output may be a valid connection strategy depending on the relationship of the cumulative carrier intersection to the cumulative miai list.

In order to not worsen the computational complexity of connection strategy deduction rules, as well as to guarantee a fixed size for the connection strategy data structure, we limit each VC/SC to a constant number of distinct miai pairs. Given this limit on miai list length, the algorithm in Figure 4.6 performs a constant number of operations that are no more computationally complex than computing carrier intersection. Thus we maintain the computational complexity of the connection strategy deduction rules.

Note that the above algorithm only retains a subset of existing miai, and never identifies new miai. The base case rule for adding miai is when the OR rule combines two SCs to create a VC, and each of the SCs has a disjoint carrier of size one. This base case automatically identifies all bridges and captured sets of size two. For this reason, we believe that allowing common miai substrategies to intersect will strictly dominate the captured set carrier intersection technique. However, the common miai substrategy has not been implemented at this time, and thus its potential remains to be explored.

4.4 Implementation Details

After much experimentation by Broderick Arneson, our H-search implementation came to strongly resemble that of Six. However, we also incorporated some other minor augmentations, such as Rasmussen's OR rule carrier intersection optimization (see §2.9.2).

Another addition, implemented with the help of Broderick Arneson, Andrea Buchfink, and Teri Drummond, was to incorporate David King's border and ladder templates. This algorithm computes all possible combinations and translations of such templates for the given board size, and uses this information as an extended base case for H-search.

The pseudocode in Figure 4.7 sketches the overall framework of our H-search implementation. This algorithm performs a static computation of connection strategies, but H-search can also be implemented in an incremental fashion, where the connection strategies of a previous state are known and only connections affected by the most recent move need to be recomputed and/or updated. This

```
Algorithm: Augmented H-search
    initialize VC/SC carrier lists:
        for each pair E of endpoints
            E.VCList.makeEmpty();
            E.SCList.makeEmpty();
        end for

    initialize queue Q with base VCs:
        Q.makeEmpty();
        for each pair E of adjacent endpoints
            E.VCList.add(baseVC(E));
            Q.add(E);
        end for
        for each pair E of template-connected endpoints
            E.VCList.add(templateVC(E));
            Q.add(E);
        end for

    while (not Q.isEmpty())
        E <- Q.removeFront();
        compute crossing rule on E's SCs:
            for each new SC Z with endpoint pair F found,
                Q.add(F);
                F.SCList.add(Z);
            end for
        compute OR rule on E's SCs:
            for each new VC Z found
                E.VCList.add(Z);
            end for
        compute AND rule on E's VCs with both of E's endpoints:
            for each new VC/SC Z with endpoint pair G found,
                Q.add(G);
                G.(VC/SC)List.add(Z);
            end for
    end while
end Algorithm
```

Figure 4.7: Augmented H-search.

incremental version is most useful within search algorithms where neighbouring positions are often investigated, such as depth-first search and alpha-beta search.

The incremental version of H-search begins by deleting any existing connection strategies that intersect opponent-coloured cells, and shrinking the carrier and/or promoting (from SC to VC) any connection strategies that intersect only player-coloured cells. The remaining connection strategies form the base case, and the processing queue is initialized to contain all pairs of endpoints for which one or more of the previous connection strategies was altered or destroyed. Because our Hex engine deduces and colours iterated fillin for every move, the number of newly-coloured cells can be significant, destroying many of the previously-valid connection strategies. On average we find the incremental version of H-search to be roughly twice as fast as the static version of H-search.

Regarding time, using borders as AND rule midpoints is a costly option, slowing down H-search by a factor of about seven on average, although this is highly dependant on the position's border connectivity and can easily exceed a factor of ten. Adding both the crossing rule and carrier set intersection when using this option results in only a 15% time increase, which suggests that few new connection strategies are found via these methods.

If borders are excluded from being AND rule midpoints, then the crossing rule is essentially useless, as partition chains are practically non-existent. However, carrier set intersection on captured sets finds many important connections, and in practice roughly doubles the computation time. Likewise, an enlarged base case roughly doubles the computation time, as suggested by Table 5.4.

Finally, we note that using all of our augmentations does not make H-search complete; there are many connections that it cannot find.

Chapter 5

Solving Hex

Thus far we have described new inferior cell analysis and H-search augmentations. We now describe how these tools can be applied when solving a Hex state.

Given a non-terminal Hex state, compute its inferior cell and connection strategy knowledge as follows:

1. Compute all fillin (*i.e.,* dead, captured, permanently inferior) for both players, colouring the corresponding cells and iterating until no more fillin can be deduced.
2. Perform all inferior cell analysis pruning (*i.e.,* dead-reversible, captured-reversible, and various forms of domination) for the player to move.
3. Run (augmented) H-search on the fillin-reduced position, computing connection strategies for both players.
4. Apply deduced connection strategies to identify captured decomposition regions and star decomposition domination. If this produces new fillin, colour the corresponding cells and return to step 1. with this fillin-reduced board.

Once this *knowledge computation* process has terminated, then the state's value can be determined if any of the following conditions hold:

- One player has a winning chain (*i.e.,* due to fillin).
- The player to move has a winning SC.
- The player to move has an empty mustplay.

Otherwise, constrain the moves to consider as follows:

- All coloured cells — including fillin — are excluded from consideration.
- All cells outside of the mustplay are excluded from consideration.
- All dead-reversible cells are excluded from consideration.

- Captured-reversible and dominated cells are excluded under the previously-defined constraints (*e.g.*, independent set and consideration of some dominating move).

This algorithmic framework is analagous to the framework used by Hayward *et al.* when solving 7×7 [83], except that decomposition computations have been added, which can cause iteration of the entire knowledge computation process. The other major differences are the iteratively-computed fillin, stronger inferior cell analysis tools, and H-search augmentations.

In the remainder of this chapter we discuss solver's underlying search algorithm, as well as additional solver-specific deduction tools that can be built from our theory. Lastly, we review our solver's performance on major benchmarks.

5.1 Depth-First Search

Hayward *et al.* solved 7×7 using a depth-first search (DFS) algorithm. Their move ordering algorithm orders cells primarily by mustplay size (when the opponent has any winning SCs) and breaks ties using an evaluation function based on Shannon's electrical circuit model (*e.g.*, similar to Hexy, Six, and Mongoose).

Although this move ordering works well on boards up to 7×7, its performance worsens on larger board sizes. Rasmussen noted that this mustplay move ordering is vital to the algorithm's success [140]. There are two main reasons for this:

- moves that confine the opponent to a small mustplay tend to be stronger moves, and
- whenever such moves are losing, they can usually be disproven quickly since the mustplay is more constrained in the subtree.

The former suggests that proof number search (PNS) is a good candidate for Hex, as PNS naturally prefers moves with smaller branching factors, and improves on depth-first search in that it can more easily correct previous erroneous decisions (*i.e.*, since it does not commit itself based on initial impressions). We now discuss PNS, and the obstacles to applying it to Hex.

5.2 Proof Number Search

Allis *et al.* introduced proof number search, an algorithm for solving two-player perfect information games in which exploration is guided by the search tree's branching factor rather than domain-specific knowledge [3, 4]. The principal idea is to use the branching factor to guide the search so that it produces a proof tree of smallest size.

Given node n in search tree T, the proof number $\phi(n)$ of n is the minimum number of leaves in T that must be solved in order to prove that node n is a win for the player to move[1]. Similarly, the

[1]This definition differs from Allis' original work in that (dis)proof numbers are not for a particular player. However, this negamax formulation produces an equivalent search and simplifies the transition to depth-first proof number search.

disproof number $\delta(n)$ of n is the minimum number of leaves in T that must be solved in order to prove that node n is a loss for the player to move. The (dis)proof number of a node n depends only on the (dis)proof numbers of its children $n_{c_1}, \ldots, n_{c_k}$, and so can be computed recursively:

$$\phi(n) = \min_{i=1,\ldots,k} \delta(n_{c_i}), \delta(n) = \sum_{i=1,\ldots,k} \phi(n_{c_i}) \tag{5.1}$$

By definition, a terminal node n has $\phi(n) = 0$ and $\delta(n) = \infty$ (respectively $\phi(n) = \infty$ and $\delta(n) = 0$) if it is a win (respectively loss) for the player to move. By convention a leaf node n has $\phi(n) = \delta(n) = 1$.

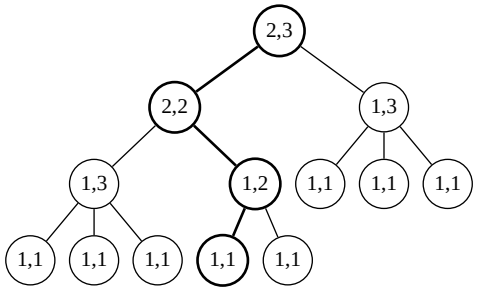


Figure 5.1: A search tree with (negamax) proof and disproof numbers. Dark lines show a path to a most proving leaf node.

For a node n with player P to move, a set of leaves in tree T is a *proof set* (respectively *disproof set*) for n if determining them to be wins (respectively losses) for P is sufficient to prove that n is a win (respectively loss) for P. A leaf node is a *most proving node* if it intersects a minimum proof set and a minimum disproof set; intuitively, this is a leaf node that contributes most towards resolving the value of a state. Allis proved that a most proving node can be found by repeatedly selecting a child with minimum disproof number among its siblings. See Figure 5.1.

PNS has been applied with success to a variety of games, including connect four, go-moku, and checkers [3, 151]. However, PNS stores the entire search tree in memory, which prevents it from solving games of high complexity. Several PN variants have been introduced to address this issue, most notably depth-first proof number (DFPN) search by Nagai [126] and two-level proof number search (PN^2) by Breuker [31].

PNS does not require domain-specific knowledge, but such knowledge can be applied to provide heuristic initialization of leaf node proof and disproof numbers, indicating which leaf nodes are deemed more likely to be wins/losses, as well as their respective solving difficulty. Kishimoto used this technique to halve the time required to solve certain types of Go problems [101]. We are not aware of any other successful technique for applying heuristic information to PNS and its variants.

5.2.1 Applying PNS to Hex

There are three impediments to applying PNS to Hex: incremental H-search, weak moves that produce fillin, and initially-uniform branching factors.

The first impediment is that our incremental version of H-search, which is faster than the static version of H-search, does not integrate easily with PNS. While DFS repeatedly transitions between states that differ by one move, the most proving nodes expanded by PNS can be in different parts of the search tree, and so multiple incremental updates are usually required between successive knowledge computations. As mentioned in §4.4, H-search is costly, and the static version is roughly twice as slow as the incremental version. At this time we have no solution to this problem other than to use static H-search (and other knowledge) computations. It follows that PNS can only surpass DFS if the former explores fewer than half as many Hex positions in its search tree. Also, because we do not want to recompute this costly static information at each node along the search tree path from one most proving node to the next, each node must store its knowledge-based list of children.

The second impediment is that many weak moves, particularly those near the borders and acute corners, typically produce significant fillin. The DFS mustplay move ordering only considers branching factor relating to mustplay, but PNS always considers branching factor, and thus initially shows preference to these weak fillin-generating moves.

The third impediment is that Hex begins with near-uniform branching factors when neither player has a (detectable) winning SC, so that initially PNS performs an inefficient breadth-first search. Since the initial branching factor is roughly 50–100 for the larger boards we wish to solve, this initial combinatorial explosion creates an excessively large search tree.

To alleviate the second and third problems, we designed our own variant of PNS that temporarily constrains the branching factor using a move ordering heuristic.

5.2.2 DFPN Search

Since we are concerned with memory usage in addition to search tree size, our PNS variant builds on Nagai's DFPN search algorithm rather than the original PNS algorithm.

The Multiple Iterative Deepening (MID) function is the driving method of DFPN search; it performs iterative deepening with local thresholds for (dis)proof numbers. Rather than store the entire search tree, DFPN search uses a transposition table (TT) that stores the nodes in the tree, the key distinction being that nodes may be overwritten before being revisited. Thus the tradeoff is the loss of some previously-computed information — possibly requiring repeat work — versus the ability to solve larger search spaces.

The helper methods of DFPN search update the (dis)proof numbers using the recursive formulas in (5.1), and select the most proving child as the next node to expand. If a node's child c has $\delta(c) \neq \infty$ (*i.e.*, it is not known to be a losing move), then it is said to be *live*. The SelectChild method also tracks the second smallest disproof value (whenever there is more than one live child), as this value is required to update the iterative deepening (dis)proof number bounds. See Figure 5.2.

```
// Setup for the root node
bool DFPN(node r) {
  r.φ ← ∞; r.δ ← ∞;
  MID(r);
  if (r.δ = ∞)
    return true;
  else
    return false;
}


// Iterative deepening at each node
void MID(node n) {
  TTlookup(n, φ, δ);
  // Exceed thresholds
  if (n.φ ≤ φ || n.δ ≤ δ) {
    n.φ ← φ; n.δ ← δ;
    return;
  }
  // Terminal node
  if (IsTerminal(n)) {
    Evaluate(n);
    // Store (dis)proven node
    TTstore(n, n.φ, n.δ);
    return;
  }

  GenerateMoves(n);
  // Iterative deepening
  while (n.φ > ΔMin(n) &&
          n.δ > ΦSum(n)) {
    n_c = SelectChild(n, φ_c, δ_c, δ_2);
    // Update thresholds
    n_c.φ ← n.δ + φ_c − ΦSum(n);
    n_c.δ ← min(n.φ, δ_2 + 1);
    MID(n_c);
  }
  // Store search results
  n.φ ← ΔMin(n);
  n.δ ← ΦSum(n);
  TTstore(n, n.φ, n.δ);
}
```

```
// Select the most promising child
node SelectChild(node n,
                 int &φ_c, int &δ_c, int &δ_2) {
  node n_best;
  δ_c ← φ_c ← ∞;
  for (each child n_child) {
    TTlookup(n_child, φ, δ);
    // Store the smallest and second
    // smallest δ in δ_c and δ_2
    if (δ < δ_c) {
      n_best ← n_child;
      δ_2 ← δ_c; φ_c ← φ; δ_c ← δ;
    }
    else if (δ < δ_2)
      δ_2 ← δ;
    if (φ = ∞)
      return n_best;
  }
  return n_best;
}


// Compute smallest δ of n's children
int ΔMin(node n) {
  int min ← ∞;
  for (each child n_child) {
    TTlookup(n_child, φ, δ);
    min ← min(min, δ);
  }
  return min;
}


// Compute sum of φ of n's children
int ΦSum(node n) {
  int sum ← 0;
  for (each child n_child) {
    TTlookup(n_child, φ, δ);
    sum ← sum + φ;
  }
  return sum;
}
```

Figure 5.2: DFPN pseudocode.

5.2.3 Focused DFPN Search

We now introduce our variant of the DFPN search algorithm, called Focused DFPN (FDFPN).

In order to identify one winning move with certain (dis)proof number bounds, PNS and its variants must first show that all sibling moves — including weak moves — cannot be solved with smaller (dis)proof number bounds. Our goal is to modify DFPN so that it focuses its effort on the strongest moves, thereby eliminating work on the weakest moves and dampening the breadth-first search behaviour in the opening.

To accomplish this, FDFPN requires a domain-specific move ordering function. For each node, the children are heuristically ordered according to this function, and initially only a fixed proportion — the *child limit* — are put in the search tree. As losing moves are identified, live children later in the ordering are added into the search tree. For games with uniform branching factor, a breadth-first search still occurs, but with a smaller branching factor, and so an exponentially smaller tree size. Similarly, search tree nodes that are winning for the player to move can be solved without expending any effort on children beyond the limit, thereby eliminating wasted work on weaker moves.

The differences between the FDFPN and DFPN search algorithms are small, as shown in Figure 5.3. In FDFPN, the MID method generates the ordered moves for node n (according to some domain-specific function), and computes a child limit l. Whenever a recursive MID call identifies a losing child, it prunes the corresponding move and recomputes the child limit. MID also passes the child limit l to the helper functions SelectChild, ΔMin, and ΦSum, which iterate over the first l live children instead of all children.

As mentioned in §5.2.2, SelectChild stores the two smallest δ values among n's children, so the child limit l must be at least two for any node n with more than one live child. Also, it is desirable that the child limit for a node reflects its branching factor, since otherwise the (dis)proof number foundation of PNS would be misguided; for instance, a constant child limit l would prevent FDFPN search from distinguishing among branching factors greater or equal to l. Keeping these desired properties in mind, our child limit formula is as follows, where $1 \leq \text{base}$ and $0 < \text{fraction} \leq 1$:

$$\text{child limit} = \text{base} + \lceil \text{fraction} \times \text{live children} \rceil \tag{5.2}$$

Each time a child of node n is solved, this formula either maintains the current child limit, thereby introducing a new child into the search tree, or else reduces the child limit by one, thereby maintaining the current set of search tree children for node n. In each case the (dis)proof numbers for n decrease, indicating that the node has become easier to solve, and ensuring further exploration in the immediate future. Figure 5.4 illustrates this process.

5.2.4 FDFPN Algorithm Analysis

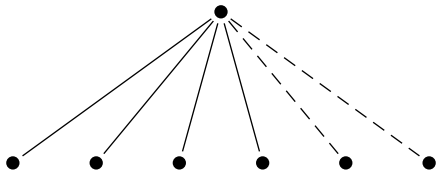
Solving a losing node requires solving all of its children. FDFPN search guarantees that each child will eventually be considered: losing children are pruned from the ordered list, and the child limit

```
    // Setup for the root node
    bool DFPN(node r) {
      r.φ ← ∞; r.δ ← ∞;
      MID(r);
      if (r.δ = ∞)
        return true;
      else
        return false;
    }


    // Iterative deepening at each node
    void MID(node n) {
      TTlookup(n, φ, δ);
      // Exceed thresholds
      if (n.φ ≤ φ || n.δ ≤ δ) {
        n.φ ← φ; n.δ ← δ;
        return;
      }
      // Terminal node
      if (IsTerminal(n)) {
        Evaluate(n);
        // Store (dis)proven node
        TTstore(n, n.φ, n.δ);
        return;
      }

(*)   GenerateOrderedMoves(n);
(+)   ComputeChildLimit(l);
      // Iterative deepening
(*)   while (n.φ > ΔMin(n, l) &&
(*)          n.δ > ΦSum(n, l)) {
(*)     n_c = SelectChild(n, l, φ_c, δ_c, δ_2);
        // Update thresholds
        n_c.φ ← n.δ + φ_c − ΦSum(n);
        n_c.δ ← min(n.φ, δ_2 + 1);
        MID(n_c);
(+)     // Identified a move as losing
(+)     if (n_c.φ = ∞) {
(+)       PruneLosingMove(n_c);
(+)       ComputeChildLimit(l);
(+)     }
      }
      // Store search results
      n.φ ← ΔMin(n);
      n.δ ← ΦSum(n);
      TTstore(n, n.φ, n.δ);
    }
```

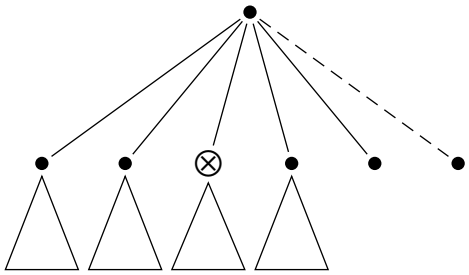
```
    // Select the most promising child
(*) node SelectChild(node n, int l,
                     int &φ_c, int &δ_c, int &δ_2) {
      node n_best;
      δ_c ← φ_c ← ∞;
(*)   for (each child n_child, among first l) {
        TTlookup(n_child, φ, δ);
        // Store the smallest and second
        // smallest δ in δ_c and δ_2
        if (δ < δ_c) {
          n_best ← n_child;
          δ_2 ← δ_c; φ_c ← φ; δ_c ← δ;
        }
        else if (δ < δ_2)
          δ_2 ← δ;
        if (φ = ∞)
          return n_best;
      }
      return n_best;
    }


    // Compute smallest δ of n's children
(*) int ΔMin(node n, int l) {
      int min ← ∞;
(*)   for (each child n_child, among first l) {
        TTlookup(n_child, φ, δ);
        min ← min(min, δ);
      }
      return min;
    }


    // Compute sum of φ of n's children
(*) int ΦSum(node n, int l) {
      int sum ← 0;
(*)   for (each child n_child, among first l) {
        TTlookup(n_child, φ, δ);
        sum ← sum + φ;
      }
      return sum;
    }
```

Figure 5.3: FDFPN pseudocode. (*) indicates modified DFPN code and (+) indicates new code.

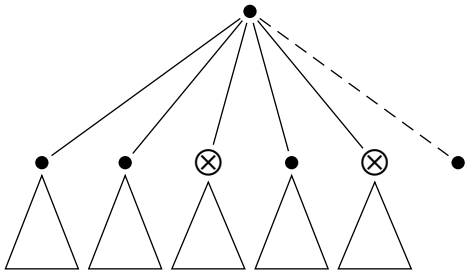
Initial node expansion. Consider only the first $b + \lceil 6f \rceil = 4$ of 6 live children:



Discover 3rd move loses. Explore new child, as now consider first $b + \lceil 5f \rceil = 4$ of 5 live children:



Discover 5th move loses. No new child, as now consider first $b + \lceil 4f \rceil = 3$ of 4 live children:



Discover 2nd move wins. Node is solved without exloring the 6th move:

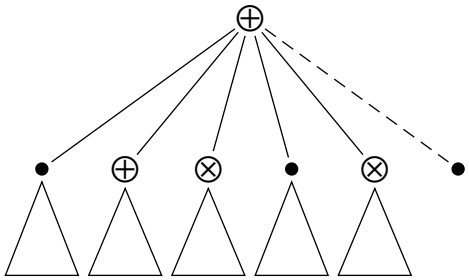


Figure 5.4: FDFPN child limit updates with base $b = 1$ and fraction $f = 0.5$.

l always retains access to some live children. Thus the only difference between DFPN and FDFPN at losing nodes is that the dis(proof) number bounds are different when some children are revealed and/or solved. Experimental results suggest that this does not cause any significant inefficiencies.

Solving a winning node requires solving only one child, although several children may be explored before a winner is found. For winning nodes, it is desirable that at least one winning child — preferably the easiest to solve — should appear within the initial child limit. In this preferable case, then FDFPN search cannot exceed the (dis)proof number bounds that DFPN search would attain; furthermore, it examines a subset of the children examined by DFPN search:

Observation 1 *Consider a node N with child node c in an FDFPN search tree whose child limit is computed via Equation (5.2), and assume that c is within N's child limit at time t. Then for all times $T > t$, c is either within N's child limit or else c is no longer live.*

Observation 2 *Consider a node N with n children in an FDFPN search tree whose child limit is computed via Equation (5.2) with base b and fraction f, and assume that $x < n$ children of N are proven losing prior to finding a winning child. Then max$(0, n - x - b - \lceil f(n - x) \rceil)$ children of node N never became accessible in the search tree.*

Observation 2 is of course the intended strength of FDFPN search. However, the dual observation is what happens when the move ordering is poor, and no winning moves are within the initial child limit:

Observation 3 *Consider a node N with n children in an FDFPN search tree whose child limit is computed via Equation (5.2) with base b and fraction f, and assume that the first x children of N are losing, where $b + \lceil fn \rceil \leq x < n$. Then at least $\lceil \frac{x-b-fn}{1-f} + \epsilon \rceil$ children of N must be proven losing before N can be solved.*

Thus, a child limit that is too restrictive for the quality of the move ordering can force the solving of nodes that would normally remain unsolved in DFPN search, potentially imposing large inefficiencies in the search tree.

5.2.5 FDFPN Experimental Results

We tested DFPN search against both FDFPN search and DFPN search with heuristic leaf initialization.

Using a set of puzzles on boards of dimension 8–10, FDFPN (with base 1, fraction 0.2) takes less than 60% of the time required by DFPN (fraction 1.0). See Figure 5.5. Further exploration of the parameter space improves this marginally to 55.6% of DFPN's time (with base 1, fraction 0.21). By comparison, our attempts at heuristic leaf (dis)proof number initialization never attains less than 80% of DFPN's time.

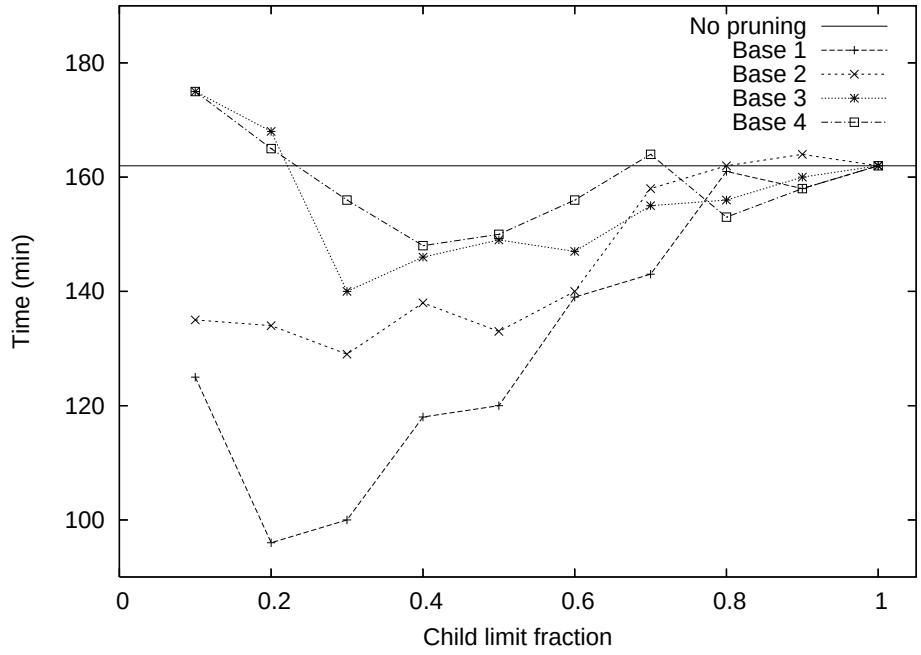


Figure 5.5: FDFPN search parameters vs. solving times.

The data suggests a base of one is best, and smaller base values generally dominate larger ones. As expected in light of Observation 3, excessive pruning worsened solving times, even beyond the time required by DFPN search with no pruning. One phenomenon for which we have no explanation is that the optimal fraction parameter for a given base seems to increase as the base parameter increases. This seems counter-intuitive, as one might expect the optimal fraction to decrease in order to counteract the base's forced increase in child exploration.

We also tested FDFPN's dependence on good move ordering, with respect to the base and fraction parameters of the child limit formula. This was done by running FDFPN search with a random move ordering on the set of all 7×7 Hex openings. See Figure 5.6.

As expected (given Observation 3), the data shows that a good move ordering is vital to the effectiveness of FDFPN search, as greater pruning only worsens the performance here. Although two data points show FDFPN search with slightly improved times despite random ordering, these cases have minimal pruning and might simply reflect the stochastic nature of this experiment and/or the improbability of all strongest winning moves being excluded by such minimal pruning.

5.2.6 FDFPN Future Improvements

Although FDFPN improves on the performance of DFPN search, both with and without heuristic leaf initialization, its fragility with respect to the heuristic move ordering is a concern. It would be preferable if a child's inclusion/exclusion could be revised during search, or if there was a gradual scaling of effort rather than a hard limit. Likewise, the reliance on an external heuristic move

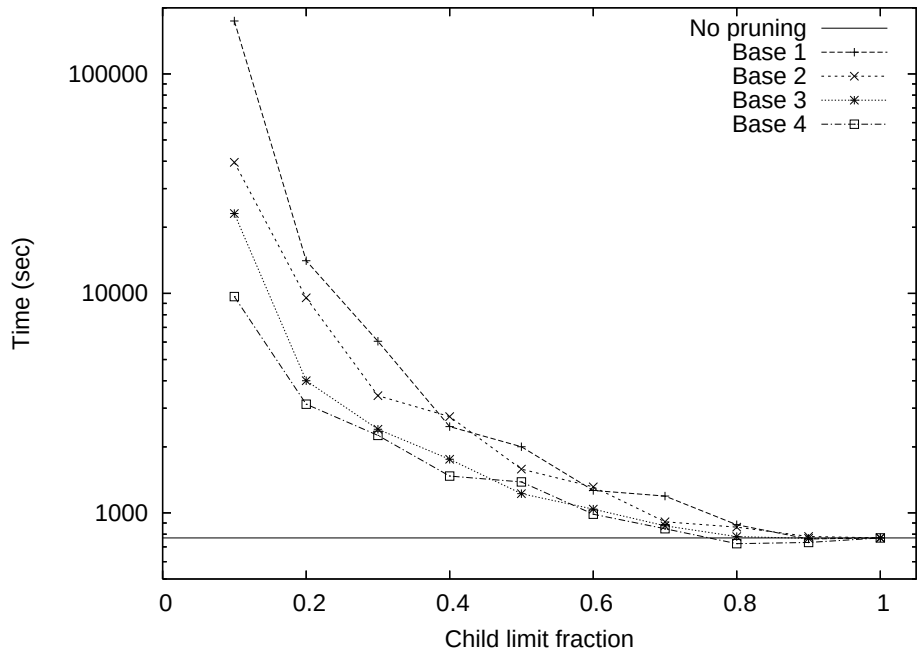


Figure 5.6: FDFPN search times with random move ordering.

ordering makes FDFPN less applicable than regular PNS and its variants.

Modifications of the child limit function could also be helpful. For instance, perhaps the child limit should depend not only on the number of live children, but also on the magnitude of differences in their heuristic evaluation.

5.3 Winning Carriers

A solver search algorithm essentially identifies winning connection strategies that cannot be deduced by H-search. Just as the carrier of a winning VC/SC can be used to identify losing moves (*e.g.*, moves outside of the mustplay), van Rijswijck noted that we can use the carrier of solver-found connection strategies to identify more losing moves, as discussed in §2.9.2. Van Rijswick's algorithm for computing these *winning carriers*[2] is recursive, and follows immediately from the definition of a winning connection strategy:

1. Base case: H-search finds a winning VC/SC, the winning carrier is the carrier of this winning VC/SC.

2. Inductive case when the player to move wins: the winning (SC) carrier is the union of the winning move's cell and the winning (VC) carrier of the winning child.

[2] Van Rijswijck calls these proof sets. We rename them to avoid confusion with the (dis)proof sets of PNS.

3. Inductive case when the player to move loses: the winning (VC) carrier is the union of all children's winning (SC) carriers.

Case 3. includes the SC carriers that define the mustplay, since these carrier cells are required to refute moves outside of the mustplay. Likewise, for any pruned dead-reversible moves, their killers and carriers must be included in the winning carrier, since the latter are required to refute the former. Note that this recursive algorithm can be easily incorporated into the recursive DFS framework.

In order to apply this idea to our new solver algorithm, we must take into account two new factors:

- connection strategies are computed on (iteratively) fillin-reduced boards, and
- our solver's underlying search engine has changed from DFS to PNS.

We present our adjustments for these factors below.

5.3.1 Fillin and Winning Carriers

Since we iteratively compute and colour fillin cells, there are two factors relevant to computing a winning carrier for the original position:

1. The *fillin carrier*: the set of cells required to maintain the fillin reduction.
2. The *strategy carrier*: the set of cells required for the winning strategy on the fillin-reduced board.

The strategy carrier for the reduced board can be computed in the same way as the winning carrier for non-reduced boards. Thus the new complication of our algorithm is the computation of a fillin carrier. In order to maintain the validity of the winning player's fillin, the set of cells required to maintain this fillin must be included in the fillin carrier. For a dead cell the carrier is simply the dead cell, and for a captured set the carrier is simply the captured set. However, in the case of permanently inferior cells the carrier extends beyond the filled in cell, and thus cells in permanently inferior carriers may be any colour — including the opponent's colour — in the fillin-reduced board. The validity of this fillin carrier follows from the discussion in §3.6. The winning carrier of the original board is the union of the fillin carrier and the strategy carrier on the fillin-reduced board, and so can still be computed in DFS for fillin-reduced boards.

5.3.2 PNS and Winning Carriers

Computing winning carriers in PNS is more complicated than in DFS for the following reasons:

1. Hex states are not encountered in a simple recursive order, so the data required to compute winning carriers needs to be stored within the TT so that it can be used if and when the state is finally solved.

2. In (F)DFPN, TT entries may be overwritten, causing this recursively-computed winning carrier information to be lost. Note that unlike lost H-search and inferior cell analysis information, this data cannot be recomputed statically.

3. Since inferior cell analysis is computed statically for each Hex state, then unlike DFS the fillin of reduced states need not agree with the fillin of their predecessors. In other words, fillin-reduced states may not be continuations of their fillin-reduced predecessors.

Given these obstacles, we adopted a restricted form of winning carrier computation in PNS and its variants. The following algorithm computes the *maximum winning carrier* for the player to move, assuming they have a winning strategy:

1. Given Hex state H_1^P and its fillin-reduced state H_2^P,

   - compute the fillin carrier of H_2^P for P as described in §5.3.1, and
   - set the strategy carrier to be all uncoloured cells in H_2^P.

2. Return P's maximum winning carrier: the union of this fillin carrier and strategy carrier.

Since any strategy carrier must be a subset of the uncoloured cells of the reduced board, and since the fillin carrier is computed as before, then it follows that either P has a winning strategy in H_1^P on the maximum winning carrier, or else H_1^P is a $\overline{P}$ win.

The advantage of this method is that it eliminates the recursive portion of the definition, allowing this carrier to be computed solely from static information. The disadvantage is the assumption of a strategy carrier that requires all uncoloured cells, which may result in a larger winning carrier than could otherwise be computed, and hence less pruning.

We also apply this winning carrier differently in PNS. Given a child that is being expanded (*i.e.*, all its static knowledge is currently available), we can compute the maximum winning carrier for the player to move. Upon returning to the parent of this child, we give it the computed maximum winning carrier and prune all other children that are external to this carrier. We now prove that this child pruning does not alter the parent state's value:

Theorem 12 *Let n be a node in a search tree, and let n_c be one of its children. Furthermore, let C be the maximum winning carrier of n_c for its player to move P. Then all $\overline{P}$ moves at node n that are not in $C \setminus \{n_c\}$ are dominated by the $\overline{P}$ move to n_c.*

Proof: If n_c is a $\overline{P}$ winning move, then the result holds by definition. So assume n_c is a losing $\overline{P}$ move, implying that P wins n_c, and thus P has a winning strategy for n_c confined to the maximum winning carrier C. It follows that P has a winning (first player) strategy on C in any continuation of node n's Hex position where $\overline{P}$ has only played cells outside of the set C. Thus all $\overline{P}$ moves at n that are not in C are losing. □

Note that this pruning is distinct from previous winning carrier pruning in that we prune siblings before determining who wins the child state (*i.e.,* the maximum winning carrier may not correspond to a winning strategy for the player to move).

5.3.3 Winning Carrier Reduction

Since smaller winning carriers yield more pruning, we also developed an efficient algorithm that, given a winning carrier, tries to deduce the existence of a smaller winning carrier:

1. Let S_1 be a Hex state with winning carrier C_1 for player P. Assign all uncoloured cells outside of C_1 to $\overline{P}$, and call this new state S_2.

2. Compute $\overline{P}$ fillin on S_2, iterating until no more cells can be $\overline{P}$-coloured. Call the resulting state S_3.

3. Define $C_2 \subseteq C_1$ to be the set of cells that are uncoloured in S_3. Then C_2 is a winning carrier for P in state S_1.

This process is illustrated in Figure 5.7.

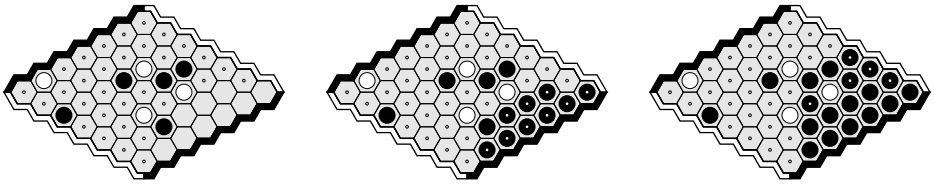


Figure 5.7: A White winning carrier for state S_1. Assigning all cells outside of the carrier to Black results in state S_2. Computing Black fillin on S_2 results in state S_3, whose uncoloured cells define a reduced winning carrier.

Theorem 13 *Given a valid winning carrier C_1 for player P in Hex state S, the output C_2 of the winning carrier reduction algorithm is a valid winning carrier for player P in Hex state S.*

Proof: By the definition of a winning carrier, assigning all uncoloured cells outside of C_1 to $\overline{P}$ cannot affect P's winning strategy. Thus C_1 is a winning strategy in state S_2. By the definition of $\overline{P}$ fillin, the value of state S_2 is equal to the value of state S_3. Thus P has a winning strategy on S_3, and of course this winning strategy must be confined to the uncoloured cells of S_3, namely C_2. Since S_3 is a continuation of S_1 (and S_2) where only $\overline{P}$-coloured cells have been added, P's winning strategy on C_2 in S_3 is also valid in S_1 (and S_2). □

We note that because of the way in which PNS winning carriers are computed — that is, on fillin-reduced boards rather than recursively — winning carrier reduction can never shrink the carrier found by a PNS-based solver. Thus winning carrier reduction is only useful for a DFS-based Hex solver.

5.4 Deducing Solved State Values

Winning carrier pruning is a method of applying one winning strategy to multiple sibling Hex positions in the search tree. We now show that given the value of one Hex state, we can apply inferior cell analysis and other properties of Hex to deduce the value of many other Hex states.

Each of the value deduction methods we illustrate can of course be applied to other deduced states, and so there is the natural concept of the closure of these solved state deductions; that is, the set of all states whose value can be deduced via the application of these operations. We have not implemented this complete version; rather, we simply apply each deduction independently to the original solved state.

These value deductions can be applied to the original state and/or its fillin-reduced state. We restrict ourselves to value deductions on the original state for the following reasons:

1. Computing fillin requires significant time, especially when using captured decomposition regions which require connection strategy information.

2. Fillin can result in an imbalanced number of coloured cells (*e.g.*, more White-coloured cells than Black-coloured cells). Deduced states of this form will likely be unreachable during search, making such output useless.

5.4.1 Winning Carrier Deductions

We now generalize the application of winning carriers to any Hex state where the winning player's cells and carrier are unaffected, and the player to move unchanged.

Theorem 14 *Let S be a Hex state where player P wins on carrier C. Let S' be a state where $(S \to P) \subseteq (S' \to P)$, $C \subseteq (S' \to \{U, P\})$, and the player to move in S' is either P or identical to the player to move in S. Then state S' is a P win.*

Proof: By definition, P's winning strategy depends only on uncoloured carrier C and the P-coloured cells in state S. Thus in the continuation of S where all uncoloured cells external to C are $\overline{P}$-coloured, P still has a winning strategy on C (with the same player to move); let us call this continuation state T.

By definition, S' has all P-coloured cells that are in T, and S' has no more $\overline{P}$-coloured than are in T. If S' and T have the same player to move, then by monotonicity $S' \geq_P T$, and so it follows that state S' is a P win. If S' and T have different players to move, then S' has P to move and T has $\overline{P}$ to move. This cannot be disadvantageous for player P, and so again it follows that state S' is a P win. □

We call the states deduced via Theorem 14 *winning carrier transpositions*. Theorem 14 can result in many state deductions (*i.e.*, all possible subsets external to C and the P-coloured cells can

be assigned to $\overline{P}$). In practice we are only interested in states that can be reached during search. In other words, we restrict the deduced states S' to those states that have the same number of $\overline{P}$-coloured cells as S, or possibly one more $\overline{P}$-coloured cell if the player to move is altered from $\overline{P}$ to P. Even with this restriction, a single solved state can produce thousands of winning carrier transpositions, so we typically restrict this method to states of limited search depth. Note that winning carrier reduction increases the number of winning carrier transpositions.

To apply this technique to (F)DFPN, we store the deduced terminal states in the TT. For DFS, we keep a database of shallow solved states for verification purposes and a playable version of the search tree strategy. Thus to apply this technique to DFS, we store the (shallow) deduced states in the database.

5.4.2 Strategy-Stealing Argument Deductions

The strategy-stealing argument applies not only to the empty Hex board, but also to any Hex position where the roles of Black and White are equivalent. For instance, any Hex position where a mirroring of the board defines a bijection from Black-coloured cells to White-coloured cells is a first player win. Thus any move to such a position is a losing move, and can be pruned from consideration. Note that there are two distinct diagonals in which to mirror the Black and White cells. See Figure 5.8.

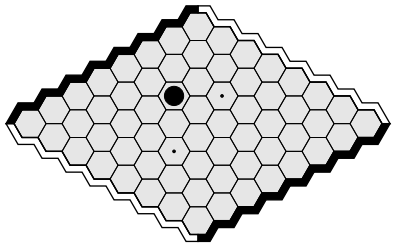


Figure 5.8: Strategy-stealing deductions: White can prune each dotted cell from consideration, since each resulting state is a first player win.

Application of this technique means that the solver only ultra-weakly solves the root state, since the explicit strategy is not known for these pruned states. However, the proof tree our algorithm finds can be extended to a complete strategy tree by later computing explicit winning strategies for all states pruned by the strategy-stealing argument.

5.4.3 Player Exchange Deductions

Player exchange deductions involve transforming a state by exchanging the roles of players P and $\overline{P}$. We do this as follows:

1. Mirror the position's coloured cells with respect to either diagonal.

2. Make all P-coloured cells $\overline{P}$-coloured and vice-versa.

3. Switch the player to move.

If the original position H_1 is a P win with player $Q \in \{P, \overline{P}\}$ to move, then the deduced position H_2 is a $\overline{P}$ win with player $\overline{Q}$ to move; this follows immediately since we simply exchanged the roles of the two players.

However, the deduction method as currently described is useless: it makes deductions about unreachable states, as the total number of coloured cells remains the same, but the player to move is altered. In order to produce transposition deductions for reachable states, the number of coloured cells must change parity. In order to change coloured cell parity without altering the outcome, we apply the following properties:

1. By monotonicity, winner-coloured cells can be added without changing a state's value.
2. By monotonicity, loser-coloured cells can be removed (*i.e.,* uncoloured) without changing a state's value.
3. By Theorem 14's proof, loser-coloured cells can be added outside of the winning player's carrier without changing a state's value (note that the carrier had to be mirrored as well).

Original state properties	**Adjustment of unreachable player exchange state**
Black to move, Black wins	Colour Black one cell outside carrier
Black to move, White wins	Colour Black one cell, or uncolour one White cell
White to move, Black wins	Colour Black one cell outside carrier
White to move, White wins	Colour Black one cell, or uncolour one White cell

Table 5.1: Player exchange deductions. Given a state with the specified winner and player to move, compute the player exchange state, and then use the listed alterations to attain reachable states whose value can be deduced.

Assuming alternating turns, the number of Black-coloured cells is always equal to or one greater than the number of White-coloured cells, so these three possible adjustments are constrained by the combination of the winning player and the player to move. Table 5.1 summarizes the possibilities. See Figure 5.9 for a sample player exchange deduction.

5.4.4 Domination Deductions

Domination can be applied to deduce solved states in the following manner, assuming player P is the winner:

1. Any P-coloured cell c_1 can be uncoloured, and a cell c_2 that P-dominates c_1 is P-coloured instead.
2. Any $\overline{P}$-coloured cell c_1 can be uncoloured, and a cell c_2 that c_1 $\overline{P}$-dominates is $\overline{P}$-coloured instead.

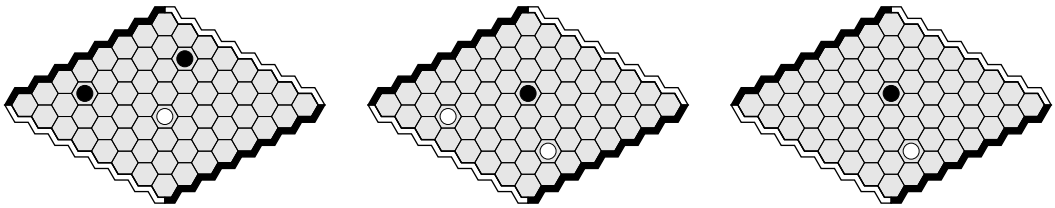


Figure 5.9: The original state is a White win with White to move. If we mirror the coloured cells and switch their colour, we obtain a state that is a Black win with Black to move. This state is unreachable, but we can uncolour a White-coloured cell to deduce a reachable state that is a Black win with Black to move.

The correctness of these domination deductions follows immediately from the definition of domination (*i.e.,* consider the state with c_1 uncoloured).

Although all states reached via domination deductions are reachable in the game, they may not be reachable by our solver's search since it prunes dominated moves. However, a move that is not currently dominated (and hence can be played during solver's search) may become dominated in some continuation (and hence can be shifted by domination deductions). Furthermore, all forms of domination — fillin, neighbourhood, induced path, and star decomposition — can exhibit this temporary behaviour. We know of no efficient algorithm to determine whether a state found via domination deduction can be reached by the solver algorithm, so we simply compute and store all domination deductions.

5.4.5 Unique Probe Deductions

Our final solved state deduction method is based on dead-reversible cells whose carrier is the empty set.

Theorem 15 *Let H^Q be a Hex state, $Q \in \{P, \overline{P}\}$, such that P has a dead-reversible cell c with killer k and an empty set carrier. If $(H + P(k) + \overline{P}(c))^Q$ is a $\overline{P}$ win, then H^Q is a $\overline{P}$ win.*

Proof: Let S_1 denote $\overline{P}$'s winning strategy for $(H + P(k) + \overline{P}(c))^Q$ in completion Hex, and define S_2 to be the combination of S_1 with a pairing strategy on $\{c, k\}$. We shall prove that S_2 is a $\overline{P}$-winning strategy for H^Q in completion Hex.

If P does not play at c nor k, then both these cells will be $\overline{P}$-coloured, and by monotonicity this outcome is at least as good for $\overline{P}$ as the corresponding terminal state in S_1, implying that $\overline{P}$ wins. If P ever plays c, then $\overline{P}$ responds with k and by definition c is dead. Thus again it is as if $\overline{P}$ claimed both of these cells, and so this state is at least as good for $\overline{P}$ as the corresponding state in S_1, and hence a $\overline{P}$ win. Lastly, if P ever plays k, then $\overline{P}$ responds with c and the state is identical to a $\overline{P}$-winning state in S_1. □

Intuitively, one player obtained their best possible local exchange and still lost. Thus we can uncolour the two cells involved in the exchange, and conclude that they would lose again. We

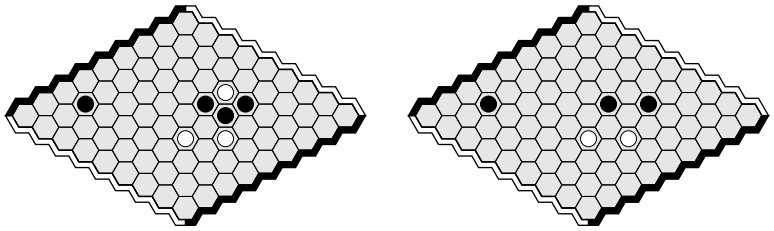


Figure 5.10: By Theorem 15, if the left state is a Black win, then it follows that the right state is a Black win.

call these *unique probe deductions*, since the pruning of dead-reversible cells ensures that the losing player has a unique line of local play available. Such deductions are identified using patterns derived from dead-reversible patterns with an empty carrier set. See Figure 5.10.

5.5 Experimental Verification

Given the complexity of our solver's tools (*e.g.*, inferior cell analysis, connection strategy deductions, deduced state values), the question of how to verify our solver's correctness is an important one. Our experimental verification consists of the following:

- For all opening moves (and some principal variations) solved by others, we verify that our algorithm obtains the same result.
- For irregular Hex boards, we verify that our algorithm obtains the proven outcome for all opening moves.
- We use a library of over 100 problems — including those generated and solved by others, but predominantly our own — to test our algorithm.
- We apply our solver during tournaments and for post-game analysis (as discussed in Chapter 6 and Appendix C), and verify that we get consistent results.

Algorithms such as winning carrier reduction and solved state deductions are extremely useful for finding any errors, since they magnify any existing problems. Also, we include many assertions in the code to verify the maintenance of theoretical properties.

5.6 Experimental Results

In this section we provide experimental results indicating the contributions of our many new algorithmic tools, both solver-specific as well as the more general inferior cell analysis and H-search augmentations. Not all of the above ideas have been implemented, and in several cases an algorithm is implemented for our DFS-based solver but not our FDFPN-based solver, or vice-versa. We show how our current solver is far superior to all of its predecessors and competitors, solving all 8×8

openings and more than half of the 9×9 openings. All tests were run on an Intel Core 2 Quad Q9559 LGA775 (2.66GHz/1333FSB/12MB) desktop with 2GB RAM.

5.6.1 Feature Contributions

Initially our solver was DFS-based, so we provide the feature contribution data for this version first. See Table 5.2.

Feature f off	**% time**	**% nodes**
Mustplay move ordering	656	2550
Decompositions	129	151
Transposition table	128	140
Acute 4-3-2 pruning	115	112
H-search border templates	111	126
Solved state deductions	108	111
Winning carrier reduction	98	101

Table 5.2: DFS solver feature contributions for 7×7 Hex.

This version of the DFS solver took roughly 10 minutes to solve all 7×7 openings, with a search tree containing 8.3e4 internal nodes. At that time, solved state deductions only encompassed winning carrier deductions and player exchange deductions, and decompositions only included split decompositions and four-sided captured decompositions. The transposition table (with 2^{20} entries) stores recently solved states to avoid resolving transpositions. Computing H-search using borders as AND rule midpoints is too slow for solvers; the improvement in pruning does not compensate for the increase in computing time. As a result, solvers never use the crossing rule.

As Rasmussen observed, mustplay move ordering is a major factor in the solver's performance. Disappointingly, winning carrier reduction only results in minimal search space reduction, and actually worsens time performance. Note the significant gap between time and search space factors for both decompositions and border templates; although these additions are worthwhile, their algorithmic cost per node is significant.

Now our solver is FDFPN-based. The TT size is fixed and entries can be overwritten, so we approximate the size of the search tree using two metrics: the number of MID calls and the number of static knowledge computations. For the FDFPN child limit, we use a default base of one and factor of 0.25, rather than optimizing the parameter choice for different board sizes and openings; see Figure 5.5. Star decomposition domination is not yet fully implemented, and so currently involves only one hard-coded inferior cell pattern at this time, shown as the leftmost diagram in Figure 3.14. Also, the following features have not yet been implemented for the FDFPN solver:

1. split decompositions,
2. common miai substrategy,
3. winning carrier deductions,

4. player exchange deductions,

5. domination deductions, and

6. unique probe deductions.

Feature f off	**% time**	**% MID**	**% knowledge**
Focused DFPN	141.1	128.2	133.3
Permanently inferior cells	100.5	100.2	100.1
Captured-reversible cells	95.7	99.8	100.1
Acute 4-3-2 pruning	99.2	102.2	102.2
Star decomposition domination	103.0	102.3	102.5
Captured decompositions	110.8	117.7	115.5
H-search border templates	156.1	215.9	209.8
H-search captured intersection	104.3	218.7	212.6
Winning carrier pruning	103.3	103.9	104.2
Strategy-stealing deduction	99.2	102.8	103.1

Table 5.3: FDFPN solver feature contributions for 7×7 Hex.

This FDFPN-based solver solves all 7×7 openings in roughly 6.5 minutes, with 88,236 MID calls and 53,954 knowledge computations. Its feature contribution data is summarized in Table 5.3. Disappointingly, the only features that seem to contribute significantly are focused DFPN, captured decompositions, and H-search base case enhancement via border templates. All other features have a negligible, or even negative, effect on time. Search space is not a major concern as memory use is essentially fixed.

Feature f off	**% time**	**% MID**	**% knowledge**
Focused DFPN	214.4	165.2	217.2
Permanently inferior cells	117.0	115.5	115.0
Captured-reversible cells	106.7	105.9	105.8
Acute 4-3-2 pruning	105.3	103.4	103.6
Star decomposition domination	89.8	94.1	94.6
Captured decompositions	209.3	237.1	229.3
H-search border templates	186.1	444.2	458.1
H-search captured intersection	141.8	302.7	315.0
Winning carrier pruning	107.2	106.0	106.2
Strategy-stealing deduction	93.9	96.6	94.9

Table 5.4: FDFPN solver feature contributions for one 9×9 Hex opening.

The data thus far suggests that most of our theoretical enhancements are not of much practical use when solving 7×7. We decided to see if this was also true for more difficult problems, and so remeasured the feature contributions with respect to solving a single 9×9 Hex opening. The results appear in Table 5.4.

In almost all cases, we see that feature contributions improved with board size. We believe this is partly because the computational complexity of most of our algorithmic improvements is polynomial

in the board size, while the corresponding increase in search space pruning grows exponentially. Furthermore, as the average game length increases, more weak moves are no longer immediately losing nor easily detectable via previous methods, and so these features become more likely to save significant search time.

It is somewhat surprising that pruning moves via the strategy-stealing argument or via star decomposition domination is a net loss. The computing cost of these checks is quite small, although the change in branching factor that they induce could cause solver to pursue weaker moves. In the future we will try modifications of these features, such as restricting the strategy-stealing argument to shallower search depths, and implementing the full version of star decomposition domination.

5.6.2 Benchmarks

Ultimately the main test of Hex solvers has been solving the opening positions of successively larger Hex boards. As of 2007, the best known automated solver (by Rasmussen *et al.*) could solve all 7 $\times$ 7 openings in roughly 61 hours [140]. By contrast, our current solver performs the task in about 6.5 minutes.

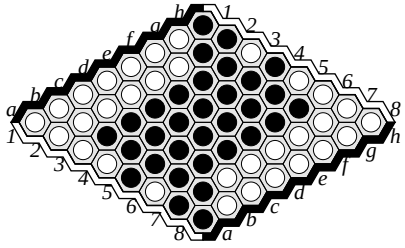


Figure 5.11: Solved 8 $\times$ 8 opening moves.

We are the first to solve any, and all, 8 $\times$ 8 openings via an automated solver. Our DFS-based solver first accomplished this task in 301 hours in 2008. Our FDFPN-based solver currently takes 31 hours. See Figure 5.11.

Another person/group has since independently solved all 8 $\times$ 8 openings, although they have only published this informally, announcing their results as they were produced on Little Golem's Hex forum [114]. We have requested details of their algorithm, computing resources, and search space statistics, but have received no reply. However, from their postings it is known that it took them over one month of computation time, and that they were using "about 10 computers" [114]. Thus a conservative estimate is that our solver outperforms theirs by a factor of 100.

Our solver is also the first and only automated algorithm to have solved any 9 $\times$ 9 Hex openings. Thus far we have solved 55 of the 81 openings, each opening taking between 1 and 25 days, with the hardest openings requiring roughly 120 million MID calls and 70 million static knowledge computations. This marks the first time that automated solvers have solved all human-solved Hex openings. See Figure 5.12.

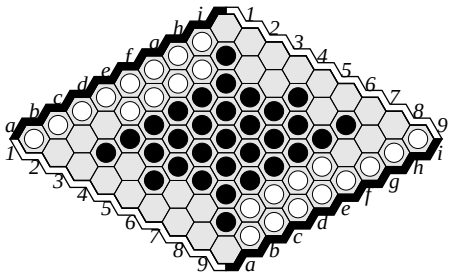


Figure 5.12: Solved 9 × 9 opening moves.

Despite the number of solved 9 × 9 openings, the work to date likely represents only a small fraction of the time needed to solve all 9 × 9 openings. On smaller $n \times n$ boards, solving the openings adjacent to a White border takes more than half of the total time. Based on the pattern of previous board sizes, we estimate that most of the remaining openings will take on the order of 100 days each, with the exception of a6, a8, a9, which we estimate will take on the order of 1000 days each. Thus a rough estimate is that solving all 9 × 9 openings will take our current solver approximately 10 years.

Board size	**Fastest opening**	**All openings**
7 × 7	0.5s	384s
8 × 8	155s	112,121s
9 × 9	96,168s	unknown

Table 5.5: Current solving opening times by board size.

Given the data presented in Table 5.5, it appears that solving all $n \times n$ openings requires about the same time as solving the easiest $(n + 1) \times (n + 1)$ opening. Thus with current hardware and software we expect that a single 10 × 10 opening can also be solved in a decade, but that solving all 10 × 10 openings would take more than 1,000 years.

Chapter 6

Automated Hex Players

Like Go, Hex has a large branching factor that limits automated players to shallow and/or selective search. Unlike Go, Hex has a reasonably strong evaluation function, and so straightforward alpha-beta techniques have been successful [9, 121]. Thus, our initial aim was to improve on the existing alpha-beta framework, and produce a stronger Hex AI via improved inferior cell analysis, stronger connection strategy deduction algorithms, and adjustments of the evaluation function; the resulting player is called Wolve. However, recently Monte Carlo Tree Search (MCTS) has been attaining great success in a variety of games, most notably Go [46, 49, 56, 70, 105]. As a result, we also pursued a MCTS-based Hex player called MoHex (in honour of Sylvain Gelly *et al.*'s Go program, MoGo). Lastly, our success with PNS and its variants in solving Hex positions suggested that perhaps a PNS-based Hex player was possible. This recent experimental player is called PNS-Hex.

In this chapter we describe the common tools used by these automated players, and then describe each of their distinct aspects. Experimental results against the open source 2003–2006 gold-medallist Six are also provided, and Appendix C provides a thorough analysis of all 2008 and 2009 olympiad games.

Unless otherwise stated, all experiments are on boards of dimension 11, the board dimension used in Hex olympiads. Since olympiad games are played with the swap rule, an automated Hex player needs to respond competently to every opening move the opponent might select. Thus, in our testing one round iterates over all openings with the swap rule off, with each program playing each opening once as Black and once as White. To reduce the standard error, we typically run experiments for several rounds.

While this testing format is helpful in identifying weaknesses (*e.g.*, openings where we perform poorly as both Black and White), it significantly dampens any strength differences, as polarized openings are played twice, which essentially guarantees some wins for the weaker player. Thus the strength gains reported in our experiments underestimate the expected tournament performance; in practice we have found that a 75% win rate in our tournaments corresponds to a nearly insurmountable difference in olympiad play. For instance, the 2006 Wolve program wins 30% of its games against Six in iterated opening tournaments, but lost all four of its games against Six in the olympiad

[80].

Lastly, in addition to percentage wins, we also refer to Elo differences. The Elo rating system uses the following formula to convert between Elo differences and expected win percentage:

$$E_P = \frac{1}{1 + 10^{(R_{\overline{P}} - R_P)/400}} \tag{6.1}$$

For instance, surpassing an opponent's Elo rating by 100, 200, 300, and 400 points corresponds to expected win rates of approximately 64, 76, 85, and 91 percent respectively.

6.1 Tools

All previously-described inferior cell analysis and connection strategy deduction algorithms are accessible to both players. In addition, players can run the solver in a parallel thread, so that positions solvable within tournament time settings are played perfectly by the solver: in winning positions, any known winning move is played, and in losing positions, the losing move requiring the most MID calls to solve is played (*i.e.*, the move that provides the most resistance). Aside from perfect endgame play, the solver also results in faster endgame play, allowing the allocation of more computing time to earlier stages of the game. At this time the parallel solver only informs the automated player if it solves the root position. In the future we hope to have the solver inform the player of any solved positions within its search tree.

Another common tool for the players that we developed is an opening book. This opening book is automatically constructed using the player's evaluation function, and is based on the algorithm developed by Lincke [113], which makes a tradeoff between book depth and evaluation score; that is, the opponent must sacrifice more of their evaluation score in order to exit the book at a shallower depth. However, we made a few modifications to Lincke's algorithm to improve performance:

1. Rather than using a tree structure, positions explored by the algorithm are stored in a database, thereby avoiding repeat work for transpositions.

2. Rather than considering all legal moves in the opening book, a branching factor limit is enforced. When a position is explored often enough, the branching factor limit is increased.

3. After the book expansion process, we iterate over all book positions with a polarized evaluation, and try to solve these positions. Any solved positions are stored, and the book values are updated accordingly.

Because different automated players prefer different types of positions, in practice we found that such opening books only improve performance when built by the player using it. For instance, a book built using Wolve's evaluation function does not improve MoHex's performance.

6.2 Wolve

Wolve is the successor to the University of Alberta's 2003 and 2006 silver-medallist alpha-beta based programs, Mongoose and Wolve; see previous olympiad reports [80, 122, 177]. Although Wolve is the common name of the 2006 and current alpha-beta player by the University of Alberta Hex research group, the Hex code base was entirely rewritten by Broderick Arneson in 2007, so the only commonality of these two automated players is their name; thus, from now on we refer to the older program as Wolve2006.

As with Hexy, Six, Mongoose, and Wolve2006, Wolve is based on computing connection strategies with H-search, and using the resulting VCs to augment Shannon's circuit-based evaluation function. However, Wolve benefits from the following improvements:

1. connection strategies are computed on fillin-reduced boards, resulting in improved connection strategy knowledge and possibly smaller mustplay, and

2. fillin and inferior cell pruning reduces the set of moves to be considered.

Wolve is also improved via iterative deepening, searching to 1-ply, 2-ply, and then 4-ply if enough time remains. Wolve uses a narrower 1-ply alpha-beta branching factor than Six (15-15-15-15 versus 20-15) to make this deeper search viable in tournament time conditions, but since Six often considers (and plays) inferior moves, especially dead-reversible moves, this difference is less significant in terms of the non-inferior moves considered.

To increase the chance of alpha-beta cutoffs, the best move from the previous iteration is moved to the front of the ordered move list for the next iteration. Iterative deepening allows Wolve to identify more fillin-domination from previous iterations, and it uses this information to prune more moves from consideration.

The circuit evaluation function is somewhat pathological with respect to fillin-domination, as it often prefers fillin-dominated positions due to their greater quantity of connection strategies (*i.e.*, fillin can exponentially decrease the number of connection strategies if their carriers only differ within the captured sets). The circuit evaluation function also exhibits a pronounced odd/even-ply behaviour, and so 3-ply Wolve is actually slightly weaker (and of course slower) than 2-ply Wolve. As a result, we skip 3-ply during iterative deepening, although 1-ply is retained for the purposes of move ordering adjustments and fillin-domination pruning.

Since Wolve's evaluation function is based on a VC-augmented version of Shannon's electric circuit evaluation function, Wolve's playing strength greatly depends on the connection strategies that are deduced. Recall that using borders as AND rule midpoints greatly slows down H-search computations. For this reason, whenever H-search uses borders as midpoints, the H-search VC/SC heuristic limits (l_V, l_S) are decreased from (25, 50) to (10, 25) to help manage the time increase.

Table 6.1 summarizes the relative strengths and computation times of several Wolve variants against Six. Since both programs are nearly deterministic, only one round was played. Wolve did

Wolve variant	Win %	Time per game (avg, max)
2-ply 20-15 border midpoints	65.7 ± 3.1	167.0, 519.4
2-ply 20-15 border midpoints, augmented	61.6 ± 3.1	169.7, 551.2
2-ply 20-15 border midpoints, augmented, crossing	63.6 ± 3.1	194.5, 608.5
4-ply 15-15-15-15	52.1 ± 3.2	183.6, 624.4
4-ply 15-15-15-15 augmented	81.8 ± 2.5	397.9, 1183.8
4-ply 15-15-15-15 border midpoints	82.2 ± 2.5	2238.5, 6900.8

Table 6.1: Wolve variants: performance against Six.

not use a parallel solver nor an opening book. Since Six uses a 2-ply 20-15 alpha-beta search, the top Wolve entry is essentially identical to Six except for our much stronger inferior cell analysis, including iterative deepening fillin-domination pruning; that is, inferior cell analysis alone accounts for a 113 Elo gain. However, application of our H-search augmentations does not improve the performance of 2-ply Wolve.

4-ply Wolve with borders as midpoints is 153 Elo stronger than 2-ply Wolve with borders as midpoints. However, even with decreased VC/SC heuristic limits, this 4-ply variant requires 37 minutes per game on average and, due to high variance in H-search computation time, can require nearly 2 hours for one game. Since olympiad time constraints are 30 minutes per player, this Wolve variant will often be constrained to a weaker 2-ply search later in the game, even when using an opening book and parallel solver. By contrast, the 4-ply Wolve variant that uses regular H-search is far too weak, and is actually outperformed by all of the above 2-ply variants. However, by augmenting H-search with border templates and captured set carrier intersection, we obtain a 4-ply Wolve that matches the strength of our previous best 4-ply Wolve variant, and is also five to six times faster.

Wolve has been successful in the International Hex Olympiads, winning the gold and silver medals in 2008 and 2009 respectively. Wolve has also dominated Six, winning 5 of their 6 olympiad games. In the future we hope to improve Wolve via the parallelization of its alpha-beta search engine, and the incorporation of killer/history heuristics [2, 179].

6.3 MoHex

Although historically the best automated Hex players have been based on an alpha-beta framework, the success of Monte Carlo Go encouraged us to apply Monte Carlo Tree Search to Hex. Below we describe MoHex, our MCTS Hex player that won silver and gold in the 2008 and 2009 olympiads respectively.

6.3.1 MoHex Framework

Monte Carlo Tree Search

Monte Carlo Tree Search is a best-first search algorithm that is guided by the outcome of random game simulations. One iteration of the algorithm is composed of three basic phases:

1. *tree traversal* from the root to some leaf node,

2. *random game simulation* from the leaf node's corresponding game position, and

3. *tree update* using information from the simulated game.

The algorithm is anytime, in that it repeats these steps until no more time remains. After the tree traversal phase, the search tree is expanded by adding the children of the selected leaf node. When MCTS terminates, the child with the largest subtree (*i.e.*, the child whose subtree produced the most simulations) is selected as the best move.

MoHex's MCTS is built on the codebase of Fuego, the Go program developed by Müller *et al.* at the University of Alberta [49].

Tree Traversal and Update

MoHex uses the upper confidence bounds applied to trees (UCT) framework combined with the all-moves-as-first (AMAF) heuristic to select the best child during tree traversal [71, 105].

The UCT framework tracks a value for each node, based on the sum of its average win/loss performance (based on all random simulated games that started in its subtree) plus an exploration term (that increases the value for less explored nodes). The tree traversal starts at the root, recursively proceeding to the child of highest value until it reaches a leaf node. This formula is designed to balance the complementary concerns of exploitation (*i.e.*, applying the best performing move) versus exploration (*i.e.*, trying moves that have largely been ignored).

The AMAF heuristic uses each random simulated game to update win/loss statistics for all moves in the simulated game, rather than for only the first move in the simulated game. Each move played by the winner is assigned a win, and each move played by the loser is assigned a loss. Thus the AMAF heuristic accelerates the rate at which MCTS accumulates data, although the resulting data may be less accurate. Tree updates occur at each node along the path from the leaf to the root, thereby influencing leaf node selection in future tree traversals.

Like Fuego, MoHex plays strongest when it uses an exploration constant of zero, effectively turning off UCT exploration and relying solely on the AMAF heuristic to find strong candidate moves.

Random Game Simulation

Recall that completion Hex (see §3.5) has the same outcome as Hex, so a random game simulation can be played until the board is completely filled, rather than checking for game termination after each random move. Hence Hex game simulations can be efficiently implemented: add all uncoloured cells to an array, shuffle them randomly, and play the remaining moves in order. A consequence of this implementation is that each legal move's AMAF statistics are updated after each game simulation.

As with Go, we have found that a MCTS player is improved by adding some knowledge to the game simulations, rather than playing completely randomly. MoHex uses a single pattern during random game simulation: if a player probes an opponent's bridge, then the opponent always replies so as to maintain the connection. If multiple bridges are probed simultaneously, then one such bridge is randomly selected and then maintained.

Yopt, a competing MCTS Hex program produced by Cazenave and Saffidine, uses an additional game simulation pattern based on the 4-3-2 VC [146]. However, Cazenave and Saffidine report that this pattern only seems to be beneficial in self-play, and in our tests MoHex shows no strength gain from this pattern.

6.3.2 Applying Hex Knowledge

Like many other MCTS players, MoHex uses knowledge-intensive algorithms in important parts of the tree, as well as flags to indicate solved states [178]. Using a fixed *knowledge threshold* parameter, if any node is visited often enough during tree traversal, then both inferior cell analysis and the H-search algorithm are run on that position. There are two possible outcomes:

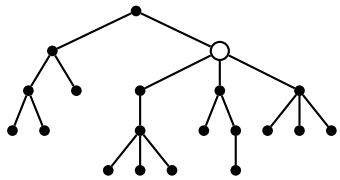
1. fillin or H-search solves the position, or
2. the position value is still unknown.

In the first case, all child subtrees are deleted, and the tree node is marked such that any tree traversal that encounters this node omits the random game simulation, and simply updates its ancestor nodes using the correct outcome.

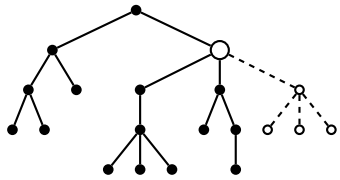
In the second case, subtrees corresponding to moves that can be pruned via inferior cell analysis or mustplay results are deleted from the tree. Furthermore, the fillin of this position is stored permanently at the tree node, and applied to every tree traversal. Since fillin is computed statically at each such tree node, there can be disagreement between the fillin of a node and the fillin of its child (*i.e.*, a child's fillin-reduced position is not necessarily a continuation of the parent's fillin-reduced position), so the descendant node's fillin takes precedence, and any prior fillin knowledge is ignored.

Fillin produces two benefits. Firstly, the random game simulations are shorter (since the number of uncoloured cells has decreased), which allows more game simulations per second. Secondly, the accuracy of the game simulations improves, since by definition fillin computes the correct local outcome.

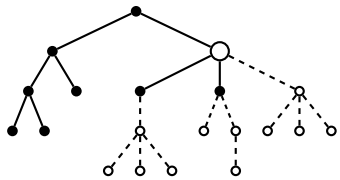
Although each child node corresponding to a fillin move is deleted, a fillin move might still be available in some child's subtree, possibly yielding an illegal game sequence in which a fillin move is played. To avoid this problem, when fillin is computed, each unpruned child's subtree is deleted excepting their roots and relevant statistics (*e.g.*, UCT and AMAF data). Note that any subsequent tree expansions below the parent node will not conflict with its fillin. This process is illustrated in Figure 6.1.



1) Node reaches knowledge threshold; perform inferior cell analysis and H-search computations.



2) Prune children that are inferior or outside of mustplay.



3) Remove subtrees of all remaining children.

Figure 6.1: Applying Hex knowledge to the Monte Carlo tree.

The knowledge threshold is typically fairly small (*e.g.,* the 2009 olympiad version of MoHex had a knowledge threshold of 50 node visits), so the size of any truncated subtree is small and, as we shall show, the subsequent loss of information is in practice more than compensated by the gain in performance.

Lock-Free Parallelization

MoHex uses the Fuego codebase, and so benefits from Fuego's lock-free parallel MCTS [49]. MoHex's knowledge computations are handled within this lock-free framework. It is possible for different threads to perform duplicate knowledge computations concurrently, but this is extremely rare in practice, and causes no theoretical problems (*i.e.,* only a marginal waste of computing resources).

Time Management

Unlike Wolve, MoHex is an anytime algorithm, so a more refined time management system is possible. In olympiad conditions, each player has 30 minutes to generate all of their moves. Since MoHex uses the parallel solver in these conditions, then MoHex needs to generate on average about 20 moves. Thus MoHex can easily allot one minute per move.

As noted earlier, MoHex selects the move whose subtree generates the most simulations. It is possible to abort search early if the gap in number of simulations between the top two candidates cannot be overcome in the time remaining; this greatly reduces computation time without affecting

performance. Because of this, in the 2009 olympiad MoHex allotted 96 seconds per move, and never experienced any time difficulties.

6.3.3 Experimental Results

We now discuss how MoHex's playing strength is affected by these various factors.

Scaling

MCTS is a parallelizable anytime algorithm, so the scaling of its performance with respect to time and number of threads is important. As with many other MCTS programs, MoHex's strength seems to scale logarithmically versus time, with each doubling of the game simulations producing roughly an additional 36 Elo of strength: 8s/move MoHex defeats 1s/move MoHex 65.1% of the time. See Figure 6.2.

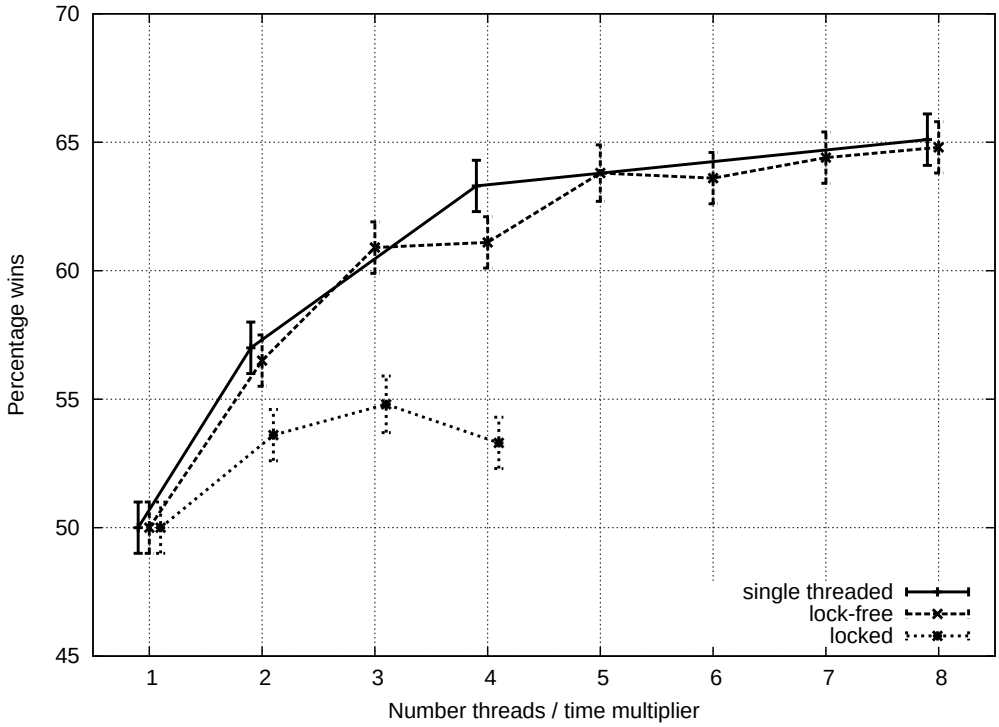


Figure 6.2: Performance of locked, lock-free, and time-scaled single threaded MoHex against single threaded 1s/move MoHex.

As with Fuego, the lock-free version of multithreaded MoHex scales far better than the locked alternative. Indeed, the effect here is even more dramatic than that in Go — scaling of the locked version is worse with two threads, and performance actually degrades with only four threads — presumably because the game simulations in Hex are so much faster than in Go, and the threads spend most of their time in the tree.

Heuristic Techniques

Both the bridge pattern and AMAF heuristic give major strength gains for MoHex. The bridge pattern produces a 105 Elo strength gain against a naive UCT implementation; this improved version is surpassed by another 181 Elo by adding the AMAF heuristic. Based on the scaling information above, this total strength gain is roughly equivalent to a 250-fold increase in computing time. See Table 6.2.

Incrementally Added Feature	Win %	Elo gain
Bridge pattern	64.7% ± 1.4%	105
AMAF heuristic	73.9% ± 1.3%	181

Table 6.2: The bridge pattern and AMAF heuristic improve playing strength by 286 Elo.

We tested many inferior cell analysis patterns as game simulation patterns. In all cases these patterns gave MoHex no strength gain, or even worsened performance. This provides evidence that provably correct information in game simulations can weaken MCTS players.

Tree Knowledge

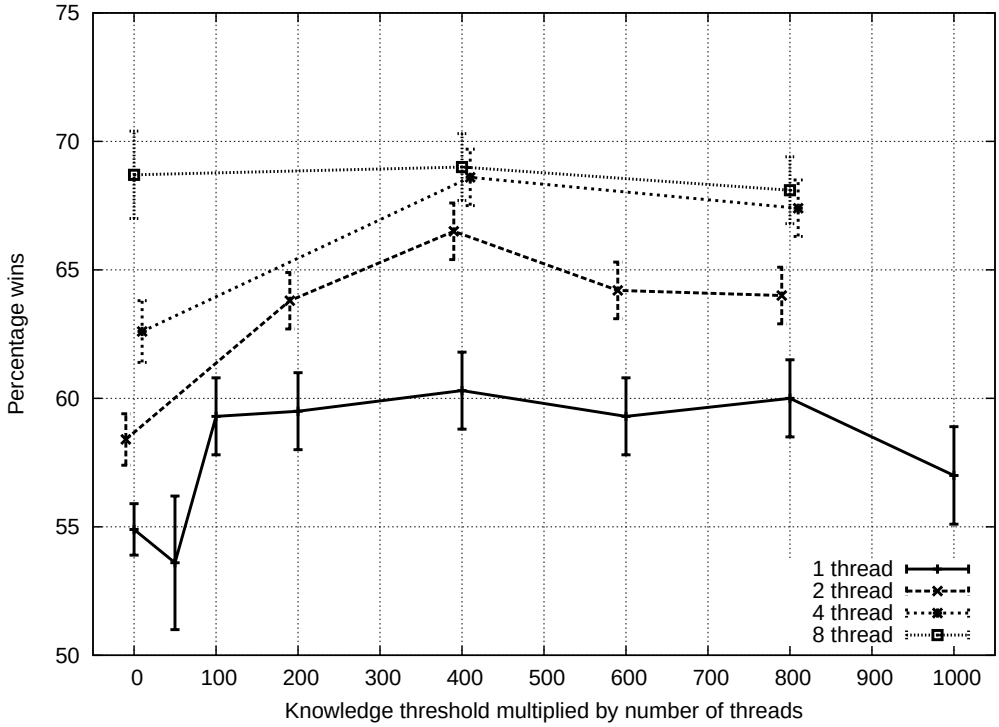


Figure 6.3: Threaded 8s/move MoHex with knowledge against 2-ply Wolve. A knowledge threshold of zero means that no knowledge is computed.

Adding connection strategy and inferior cell knowledge within MoHex's Monte Carlo tree is roughly equivalent to doubling the number of game simulations. The optimal knowledge threshold

seems to be about 400 for the single threaded version, and to decrease proportionally with the number of threads. We note that a low knowledge threshold can worsen performance, as not enough time is spent on random game simulations. See Figure 6.3.

Opening Book

The opening play of MoHex can be inconsistent (see Appendix C), perhaps because there is so little existing structure to guide the random game simulations. Our initial opening book results are promising: an opening book for 9×9 Hex that was constructed in one day produces gains of 85 Elo, which is worth more than a doubling of simulations on that board size. As book size increases, playing strength gains grow logarithmically. We have found that it is important for the opening book to be computed using a MoHex evaluation that is at least as strong as the MoHex player using the book; otherwise the book's strength gains are diminished. See Figure 6.4.

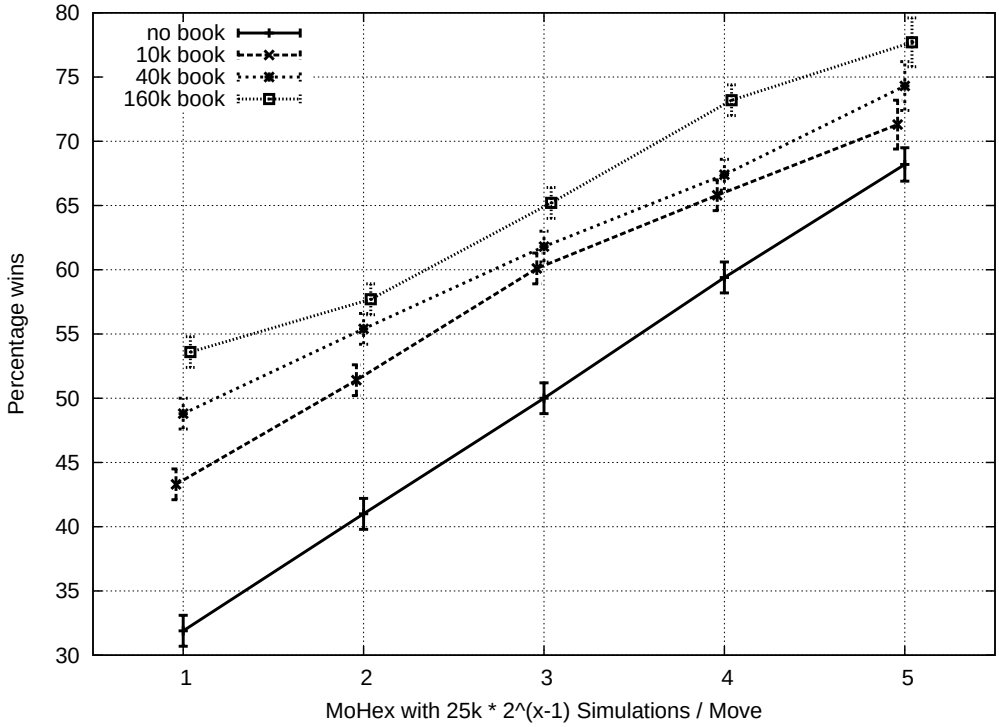


Figure 6.4: MoHex with books of increasing size against 100k / move MoHex with no book.

H-search Augmentations

MoHex uses connection strategies only to determine captured decomposition fillin, to define the mustplay, and to identify solved node positions. In general we have found more complete versions of H-search to be not as useful in MoHex, as the slight improvement in pruning does not compensate for H-search's time increase (which decreases the number of game simulations that can be performed). Thus our best version of MoHex does not use borders as AND rule midpoints, even though this

variant of H-search is preferred by Wolve. However, MoHex does use border template base case augmentations.

We have tried to incorporate connection strategies into random game simulations. Two such attempts were to heuristically select connections to maintain during the simulation, and to add connection strategy maintenance choices as moves in the Monte Carlo tree. Unfortunately, all such techniques greatly worsened performance, with the best variation to date using common responses in the Monte Carlo tree to guide responses in the simulated game. If connection strategies can be used more effectively by MoHex, it is possible that H-search augmentations could prove worthwhile.

Tournament Strength

As stated earlier, MoHex won the silver and gold medals in the 2008 and 2009 Hex Olympiads. In these two tournaments, MoHex won 4 of its 6 games against Six and 5 of its 6 games against the competing MCTS Hex player Yopt. Furthermore, all of MoHex's losses were in 2008, prior to many algorithmic improvements. We summarize MoHex's playing strength (without the use of an opening book or parallel solver) against Six and Wolve in Table 6.3. Basically, MoHex dominates Six and is evenly matched with Wolve.

Opponent	**MoHex Win %**
Six	76.6 ± 3.6
Wolve 4-ply 20-20-20-20 augmented	49.2 ± 3.2

Table 6.3: MoHex: performance against Six and Wolve.

6.4 PNS-Hex

While applying our FDFPN-based Hex solver to olympiad positions and puzzles, we observed that the solver typically focused its efforts on the best move long before it proved the position's value. Thus, the solver's early efforts served as a good predictor of what would end up being the best move (*i.e.*, either winning, or losing but giving the greatest resistance).

This is not surprising, since an effective solver must spend more of its time exploring stronger moves. Based on this observation we speculated that the solver could work as an anytime Hex player, running within the given time constraints until it either solved the position, or else selected a move heuristically using its relative efforts on each child's subtree thus far. Two possible metrics were considered, the latter suggested by Broderick Arneson:

1. select the child with the most MID calls, or
2. select the child whose ϕ is largest.

Only the first metric has been implemented so far. We tested this basic player against MoHex

on both 9×9 and 11×11 boards, with MoHex generating 10,000 random game simulations per move. The results are shown in Figure 6.5.

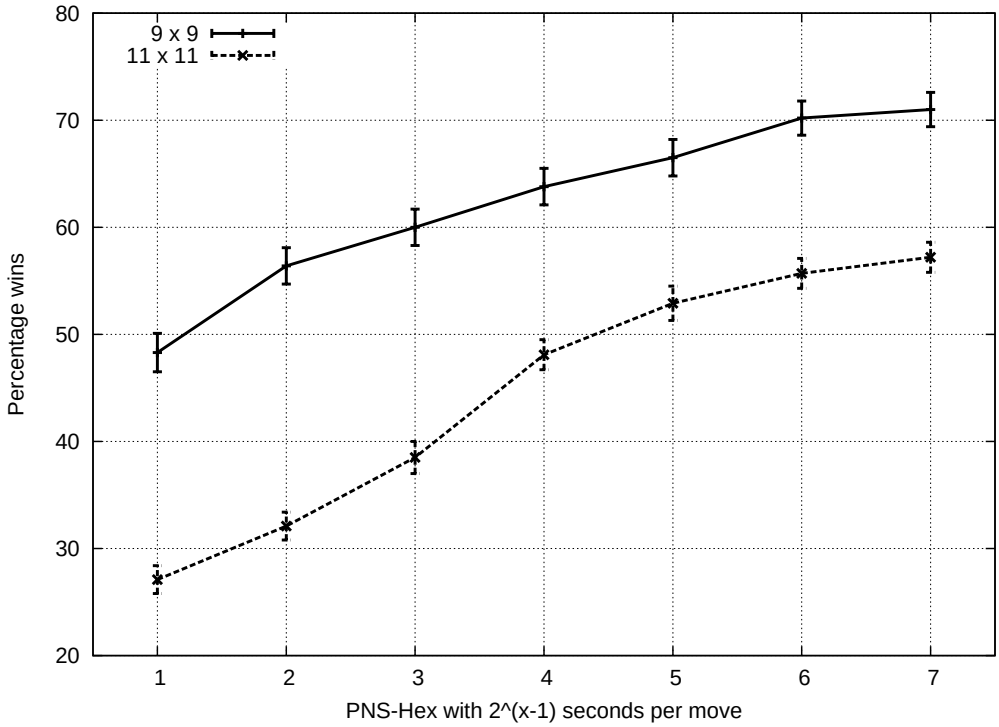


Figure 6.5: PNS-Hex: performance against MoHex-10k.

On a per-second basis, this basic automated player is roughly equal with MoHex on the 9×9 board: MoHex-10k's total time per game (10s) is about halfway between the total time required by 1s and 2s PNS-Hex (7.7s and 14s respectively). However, on the olympiad-sized 11×11 board, PNS-Hex requires 3.5 times as much computation time as MoHex to produce equal playing strength (42s total time for MoHex versus total times of 94s and 191s for 8s and 16s PNS-Hex respectively). This is largely due to PNS-Hex's very weak opening play, as the lack of any mustplay for early positions results in nearly-uniform branching factors, where PNS-Hex is essentially unguided in its search.

Given the initial success of this basic player, we plan to investigate the following adjustments of PNS-Hex:

1. Test the ϕ metric (versus MID call metric) for move selection.
2. Test various focused child limit parameters for FDFPN.
3. Test the use of a database that accumulates (dis)proof number values for opening positions over time (*i.e.*, as PNS-Hex plays many games).
4. Test the benefits of using an opening book.

If this concept proves useful, it can easily be generalized to other games, as well as general game playing.

Chapter 7

Conclusion

The research presented in this thesis represents a significant contribution to the mathematical and algorithmic knowledge of Hex. Inferior cell pruning, fillin, and decomposition analysis have all been improved, and several efficient augmentations of connection strategy deduction algorithms have been developed.

Applying this new theory has resulted in the world's best Hex solver, an algorithm that has surpassed all previous benchmarks, provides the first instance of an automated solver dominating humans, and outperforms all competing automated solvers by at least two orders of magnitude.

Applying this new theory has also helped produce the world's two strongest automated Hex players — Wolve and MoHex — who have dominated the International Hex Computer Olympiads since their introduction, and far surpass the previous gold medallist Six.

Many important open questions and challenges remain, and this research raises many new questions and challenges (see Appendix D).

Hex is a game that has interested mathematicians and computer scientists since its invention. The graph-theoretic, combinatorial game theoretic, and artificial intelligence aspects of this game ensure that it will continue to do so in the forseeable future.

Bibliography

[1] *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA*, Menlo Park, 2000. AAAI Press / The MIT Press.

[2] Selim G. Akl and Monroe M. Newborn. The principal continuation and the killer heuristic. In *ACM '77: Proceedings of the 1977 annual conference*, pages 466–473, New York, NY, USA, 1977. ACM.

[3] L. Victor Allis. *Searching for Solutions in Games and Artificial Intelligence*. PhD thesis, University of Limburg, Maastricht, Netherlands, 1994.

[4] L. Victor Allis, Maarten van der Meulen, and H. Jaap van den Herik. Proof-number search. *Artificial Intelligence*, 66(1):91–124, 1994.

[5] Steve Alpern and Anatole Beck. Hex games and twist maps on the annulus. *Amer. Math. Monthly*, 98(9):803–811, 1991.

[6] Vadim V. Anshelevich. Hexy's home page. home.earthlink.net/˜vanshel/.

[7] Vadim V. Anshelevich. The game of Hex: An automatic theorem proving approach to game programming. In AAAI2000 [1], pages 189–194.

[8] Vadim V. Anshelevich. The game of Hex: The hierarchical approach. Combinatorial Game Theory Workshop. MSRI, Berkeley. http://www.msri.org/publications/ln/msri/2000/gametheory/anshelevich/1/index.html, July 2000.

[9] Vadim V. Anshelevich. Hexy wins Hex tournament. *ICGA Journal*, 23(3):181–184, 2000.

[10] Vadim V. Anshelevich. The game of Hex: The hierarchical approach. In Richard J. Nowakowski, editor, *More Games of No Chance*, volume 42 of *MSRI Publications*, pages 151–165. Cambridge University Press, Cambridge, 2002.

[11] Vadim V. Anshelevich. A hierarchical approach to computer Hex. *Artificial Intelligence*, 134(1–2):101–120, 2002.

[12] Broderick Arneson, Ryan B. Hayward, and Philip Henderson. MoHex wins Hex tournament. *ICGA Journal*, 32(2):114–116, 2009.

[13] Broderick Arneson, Ryan B. Hayward, and Philip Henderson. Wolve 2008 wins Hex tournament. *ICGA Journal*, 32(1):49–53, 2009.

[14] Broderick Arneson, Ryan B. Hayward, and Philip Henderson. Monte Carlo Tree Search in Hex. Accepted to Transactions on Computational Intelligence and AI in Games, Special Issue on Monte Carlo Techniques and Computer Go, 2010.

[15] Broderick Arneson, Ryan B. Hayward, and Philip Henderson. Solving Hex: Beyond humans. Accepted to Computers and Games, 2010.

[16] A.A. Arratia-Quesada and Iain A. Stewart. Generalized Hex and logical characterizations of polynomial space. *Information Processing Letters*, 63(3):147–152, 1997.

[17] Argimiro Arratia-Quesada. On the descriptive complexity of a simplified game of Hex. *Logic Journal of the IGPL*, 10(2):105–122, 2002.

[18] Anatole Beck, Michael N. Bleicher, and Donald W. Crowe. *Excursions into Mathematics*, chapter 5, pages 327–339. Worth, New York, 1969.

[19] Anatole Beck, Michael N. Bleicher, and Donald W. Crowe. *Excursions into Mathematics: the Millennium Edition*, chapter Appendix 2000. A.K. Peters, Natick, Massachusetts, 2000.

[20] C. Berge. Vers une théorie générale des jeux positionnels. In R. Henn and O. Moeschlin, editors, *Mathematical Economics and Game Theory, Essays in Honor of Oskar Morgenstern*, Lecture Notes in Economics, volume 141, pages 13–24. Springer Verlag, Berlin, 1977.

[21] Claude Berge. L'art subtil du Hex. Manuscript, 1977.

[22] Claude Berge. Some remarks about a Hex problem. In D. Klarner, editor, *The Mathematical Gardner*, pages 25–27. Wadsworth International, Belmont, 1981.

[23] Elwyn Berlekamp, John H. Conway, and Richard K. Guy. *Winning Ways for Your Mathematical Plays*, volume 1–4. A.K. Peters, 2nd edition, 2000.

[24] Yngvi Björnsson, Ryan Hayward, Michael Johanson, and Jack van Rijswijck. Dead cell analysis in Hex and the Shannon game. In Adrian Bondy, Jean Fonlupt, Jean-Luc Fouquet, Jean-Claude Fournier, and Jorge L. Ramirez Alfonsin, editors, *Graph Theory in Paris: Proceedings of a Conference in Memory of Claude Berge*, pages 45–60. Birkhäuser, 2007.

[25] BoardSpace. www.boardspace.net/english/index.shtml, 2009.

[26] Béla Bollobás and Imre Leader. The angel and the devil in three dimensions. *J. Comb. Theory Ser. A*, 113(1):176–184, 2006.

[27] J.A. Bondy and U.S.R. Murty. *Graph Theory*, volume 244 of *Graduate Texts in Mathematics*. Springer, 2nd edition, 2008.

[28] D.L. Book. What the Hex. *The Washington Post*, page H02, 1998.

[29] Craig Boutilier, editor. *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, 2009.

[30] Brian H. Bowditch. The angel game in the plane. *Comb. Probab. Comput.*, 16(3):345–362, 2007.

[31] Dennis M. Breuker. *Memory versus Search in Games*. PhD thesis, Maastricht University, Maastricht, Netherlands, 1998.

[32] Cameron Browne. *Hex Strategy: Making the Right Connections*. A.K. Peters, Natick, Massachusetts, 2000.

[33] Cameron Browne. Hex strategy part 1: Introduction and basic strategy. *Abstract Games Magazine*, 2, 2000.

[34] Cameron Browne. Hex strategy part 2: Board analysis. *Abstract Games Magazine*, 3, 2000.

[35] Cameron Browne. Hex strategy part 3: Ladders. *Abstract Games Magazine*, 4, 2000.

[36] Cameron Browne. Hex strategy part 4: Computer Hex. *Abstract Games Magazine*, 8:17–21, 2001.

[37] Cameron Browne. Hex strategy part 6: Opening theory. *Abstract Games Magazine*, 10:25–28, 2002.

[38] J. Bruno and L. Weinberg. A constructive graph-theoretic solution of the Shannon switching game. *Circuit Theory, IEEE Transactions on*, 17(1):74–81, Feb 1970.

[39] Michael Buro. Toward opening book learning. *ICCA Journal*, 22(2):98–102, 1999.

[40] Michael Buro. Simple amazons endgames and their connection to hamilton circuits in cubic subgrid graphs. In Marsland and Frank [117], pages 250–261.

[41] Michael Buro. Improving heuristic mini-max search by supervised learning. *Artif. Intell.*, 134(1–2):85–99, 2002.

[42] Garikai Campbell. On optimal play in the game of Hex. *INTEGERS: Electronic Journal of Combinatorial Number Theory*, 4:1–23, 2004.

[43] Guillaume Chaslot, Mark Winands, H. Jaap van den Herik, Jos Uiterwijk, and Bruno Bouzy. Progressive strategies for Monte-Carlo tree search. In *Joint Conference on Information Sciences, Salt Lake City 2007, Heuristic Search and Computer Game Playing Session*, 2007.

[44] J.H. Conway. The angel problem. In Richard J. Nowakowski, editor, *Games of No Chance*, volume 29 of *MSRI Publications*, pages 3–12. Cambridge University Press, Cambridge, 1996.

[45] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Cliff Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[46] Rémi Coulom. Computing Elo ratings of move patterns in the game of Go. Draft, submitted to ICGA Computer Games Workshop 2007, 2007.

[47] Chrilly Donninger and Ulf Lorenz. Innovative opening-book handling. In van den Herik et al. [164], pages 1–10.

[48] Bert Enderton. Answers to infrequently asked questions about the game of Hex. www.cs.cmu.edu/˜hde/hex/hexfaq/, 1995.

[49] M. Enzenberger and M. Müller. Fuego homepage, 2008. Date of publication: May 27, 2008. Date retrieved: March 14, 2010.

[50] Ronald J. Evans. A winning opening in reverse Hex. *Journal of Recreational Mathematics*, 7(3):189–192, 1974.

[51] Ronald J. Evans. Some variants of Hex. *Journal of Recreational Mathematics*, 8:120–122, 1975.

[52] Shimon Even and R. Endre Tarjan. A combinatorial problem which is complete in polynomial space. *Journal of the Association for Computing Machinery*, 23(4):710–719, 1976.

[53] M.R. Fellows. The Robertson-Seymour theorems: A survey of applications. *Contemporary Mathematics*, 89:1–18, 1989.

[54] Hilmar Finnsson and Yngvi Björnsson. Simulation-based approach to general game playing. In Fox and Gomes [57], pages 259–264.

[55] *19th Annual Symposium on Foundations of Computer Science, 16-18 October, 1978, Ann Arbor, Michigan, USA*. IEEE, 1978.

[56] D. Fotland. The Many Faces of Go, version 12, 2009. Date retrieved: April 1, 2009.

[57] Dieter Fox and Carla P. Gomes, editors. *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*. AAAI Press, 2008.

[58] Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors. *Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings*, volume 4212 of *Lecture Notes in Computer Science*. Springer, 2006.

[59] David Gale. The game of Hex and the Brouwer fixed point theorem. *American Mathematical Monthly*, 86(10):818–827, 1979.

[60] Martin Gardner. Mathematical games. *Scientific American*, 197(1):145–150, June 1957.

[61] Martin Gardner. Mathematical games. *Scientific American*, 197(2):120–127, August 1957.

[62] Martin Gardner. Mathematical games. *Scientific American*, 197(4):130–138, October 1957.

[63] Martin Gardner. *The Scientific American Book of Mathematical Puzzles and Diversions*, chapter 8, pages 73–83. Simon and Schuster, New York, 1959.

[64] Martin Gardner. *The 2nd Scientific American Book of Mathematical Puzzles and Diversions*, chapter 7, pages 78–88. Simon and Schuster, New York, 1961.

[65] Martin Gardner. Mathematical games. *Scientific American*, 205(1):148–161, July 1961.

[66] Martin Gardner. Mathematical games. *Scientific American*, 232(6):106–111, June 1975.

[67] Martin Gardner. Mathematical games. *Scientific American*, 233(6):116–119, December 1975.

[68] Martin Gardner. *Hexaflexagons and Other Mathematical Diversions: The First Scientific American Book of Puzzles and Games*, chapter 8, pages 73–83. University of Chicago Press, Chicago, USA, 1988.

[69] Michael R. Garey and David S. Johnson. *Computers and Intractability*. W.H. Freeman and Co., San Francisco, 1979.

[70] S. Gelly, Y. Wang, R. Munos, and O. Teytaud. Modification of UCT with patterns in Monte-Carlo Go, 2006. Technical Report RR-6062.

[71] Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In Ghahramani [74], pages 273–280.

[72] Sylvain Gelly and Yizao Wang. Exploration exploitation in Go: UCT for Monte-Carlo Go, December 2006.

[73] Sylvain Gelly, Yizao Wang, Rémi Munos, and Olivier Teytaud. Modification of UCT with patterns in Monte-Carlo Go. Technical Report 6062, INRIA, France, November 2006.

[74] Zoubin Ghahramani, editor. *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvalis, Oregon, USA, June 20-24, 2007*, volume 227 of *ACM International Conference Proceeding Series*. ACM, 2007.

[75] Robert Haas and Michael Hoffman. Chordless paths through three vertices. *Theoretical Computer Science*, 351(3):360–371, 2006.

[76] Yahya Ould Hamidoune and Michel Las Vergnas. A solution to the misère Shannon switching game. *Discrete Mathematics*, 72(1-3):163–166, 1988.

[77] Ryan Hayward, Yngvi Björnsson, Michael Johanson, Morgan Kan, Nathan Po, and Jack van Rijswijck. Solving 7×7 Hex with domination, fill-in, and virtual connections. *Theoretical Computer Science*, 349(2):123–139, 2005.

[78] Ryan B. Hayward. Berge and the art of Hex. To appear in A. Bondy, V. Chvátal, eds., A biography of Claude Berge, Princeton University Press, 2003.

[79] Ryan B. Hayward. A note on domination in Hex. Technical report, University of Alberta, 2003.

[80] Ryan B. Hayward. Six wins Hex tournament. *ICGA Journal*, 29(3):163–165, 2006.

[81] Ryan B. Hayward. A puzzling Hex primer. In Richard J. Nowakowski, editor, *Games of No Chance III*. Cambridge University Press, 2008 (in press).

[82] Ryan B. Hayward, Broderick Arneson, and Philip Henderson. Automatic strategy verification for Hex. In van den Herik et al. [163], pages 112–121.

[83] Ryan B. Hayward, Yngvi Björnsson, Michael Johanson, Morgan Kan, Nathan Po, and Jack van Rijswijck. Solving 7×7 Hex: Virtual connections and game-state reduction. In H. Jaap van den Herik, Hiroyuki Iida, and Ernst A. Heinz, editors, *Advances in Computer Games*, volume 263 of *International Federation for Information Processing*, pages 261–278. Kluwer Academic Publishers, Boston, 2003.

[84] Ryan B. Hayward and Jack van Rijswijck. Hex and combinatorics. *Discrete Mathematics*, 306(19–20):2515–2528, 2006.

[85] Piet Hein. Vil de laere Polygon? *Politiken*, December 1942.

[86] Philip Henderson, Broderick Arneson, and Ryan Hayward. Hex, braids, the crossing rule, and XH-search. In van den Herik and Spronck [168], pages 88–98.

[87] Philip Henderson, Broderick Arneson, and Ryan B. Hayward. Solving 8x8 Hex. In Boutilier [29], pages 505–510.

[88] Philip Henderson and Ryan B. Hayward. Probing the 4-3-2 edge template in Hex. In van den Herik et al. [167], pages 229–240.

[89] Philip Henderson and Ryan B. Hayward. Captured-reversible moves and star decomposition domination in Hex. Submitted to Integers, 2010.

[90] Philip Henderson and Ryan B. Hayward. A handicap strategy for Hex. In Richard J. Nowakowski, editor, *Games of No Chance IV*. Cambridge University Press, 2010 (in press).

[91] Joachim Hertzberg, Michael Beetz, and Roman Englert, editors. *KI 2007: Advances in Artificial Intelligence, 30th Annual German Conference on AI, KI 2007, Osnabrück, Germany, September 10-13, 2007, Proceedings*, volume 4667 of *Lecture Notes in Computer Science*. Springer, 2007.

[92] HexWiki. www.hexwiki.org/index.php?title=Main_Page, 2008.

[93] R.M. Hyatt. Book learning - a methodology to tune an opening book automatically. *ICCA Journal*, 22(1):3–12, 1999.

[94] ig Game Center. www.iggamecenter.com/, 2009.

[95] Tommy R. Jensen and Bjarne Toft. *Graph Coloring Problems*, chapter 17, pages 271–275. Discrete Mathematics and Optimization. Wiley Interscience, New York, 1995. Problem 17.14: Winning Hex.

[96] JHex. jhex.sourceforge.net/, 2003.

[97] Kenneth Kahl, Stefan Edelkamp, and Lars Hildebrand. Learning how to play Hex. In Hertzberg et al. [91], pages 382–396.

[98] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[99] Stefan Kiefer. Die Menge der Virtuellen Verbindungen im Spiel Hex ist PSPACE-völlstandig. Technical report, Studienarbeit Nr. 1887, Facultät für Informatik, Electrotechnik, und Informationstechnik, Universität Stuttgart, 2003.

[100] David King. Hall of hexagons - the game of Hex. www.drking.plus.com/hexagons/hex/index.html, 2007.

[101] Akihiro Kishimoto. *Correct and Efficient Search Algorithms in the Presence of Repetitions*. PhD thesis, Dept. of Computing Science, University of Alberta, Edmonton, Canada, 2005.

[102] Akihiro Kishimoto and Martin Müller. A solution to the GHI problem for depth-first proof-number search. *Inf. Sci.*, 175(4):296–314, 2005.

[103] Oddvar Kloster. A solution to the angel problem. *Theor. Comput. Sci.*, 389(1-2):152–161, 2007.

[104] Donald E. Knuth and Ronald W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975.

[105] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In Fürnkranz et al. [58], pages 282–293.

[106] Harold William Kuhn and Sylvia Nasar, editors. *The Essential John Nash*. Princeton University Press, 2002.

[107] C. Kuratowski. Sur le problème des courbes gauches en topologie. *Fund. Math.*, 15:217–283, 1930.

[108] Martin Kutz. Conway's angel in three dimensions. *Theor. Comput. Sci.*, 349(3):443–451, 2005.

[109] Jeffrey Lagarias and Daniel Sleator. *The Mathemagician and Pied Puzzler: A Collection in Tribute to Martin Gardner, editors Elwyn Berlekamp and Tom Rodgers*, chapter 3, pages 237–240. A.K. Peters, 1999.

[110] Alfred Lehman. A solution of the Shannon switching game. *Journal of the Society of Industrial and Applied Mathematics*, 12:687–725, 1964.

[111] Mark Levene and Judit Bar-Ilan. Comparing typical opening move choices made by humans and chess engines. *The Computer Journal*, 50(5):567–573, 2007.

[112] David Lichtenstein and Michael Sipser. Go is PSPACE hard. In FOCS19 [55], pages 48–54.

[113] Thomas R. Lincke. Strategies for the automatic construction of opening books. In Marsland and Frank [117], pages 74–86.

[114] Little Golem. www.littlegolem.net/jsp/, 2009.

[115] Thomas Maarup. Everything you always wanted to know about Hex but were afraid to ask. Master's thesis, University of Southern Denmark, 2005.

[116] Thomas Maarup. Hex. maarup.net/thomas/hex/, 2005.

[117] T. Anthony Marsland and Ian Frank, editors. *Computers and Games, Second International Conference, CG 2000, Hamamatsu, Japan, October 26-28, 2000, Revised Papers*, volume 2063 of *Lecture Notes in Computer Science*. Springer, 2002.

[118] András Máthé. The angel of power 2 wins. *Comb. Probab. Comput.*, 16(3):363–374, 2007.

[119] David A. McAllester. Conspiracy numbers for min-max search. *Artif. Intell.*, 35(3):287–310, 1988.

[120] J.W.A. McWorter. Kriegspiel Hex. *Mathematics Magazine*, 54:85–86, 1981.

[121] Gábor Melis. Six. six.retes.hu/, 2006.

[122] Gábor Melis and Ryan Hayward. Six wins Hex tournament. *ICGA Journal*, 26(4):277–280, 2003.

[123] Machine Intelligence for Hex. www.cs.newcastle.edu.au/ mihex/index.html, 2003.

[124] John Milnor. A Nobel prize for John Nash. *Mathematical Intelligencer*, 17(3):11–17, 1995.

[125] Ken Mishima, Hidetoshi Sakurai, and Kohei Noshita. New proof techniques and their applications to winning strategies in Hex. *Proceedings of 11th Game Programming Workshop in Japan*, pages 136–142, 2006.

[126] A. Nagai. *Df-pn Algorithm for Searching AND/OR Trees and Its Applications*. PhD thesis, Dept. of Information Science, University of Tokyo, Tokyo, Japan, 2002.

[127] Sylvia Nasar. *A Beautiful Mind: A Biography of John Forbes Nash, Jr.* Simon and Schuster, 1998.

[128] John Nash. Some games and machines for playing them. Technical Report D-1164, RAND, February 1952.

[129] John Nash, December 1999. Telephone conversation with Ryan Hayward and Jack van Rijswijck.

[130] Cyril Tse Ning, editor. *1st International Conference on Application and Development of Computer Games in the 21st Century (ADCOG 2003) held on 22-23 November 2001 in City University of Hong Kong, HKSAR, China*. City University of Hong Kong, 2001.

[131] Kohei Noshita. Union-connections and a simple readable winning way in 7×7 Hex. *Proceedings of 9th Game Programming Workshop in Japan*, pages 72–79, 2004.

[132] Kohei Noshita. Union-connections and straightforward winning strategies in Hex. *ICGA Journal*, 28(1):3–12, 2005.

[133] Kohei Noshita. Union-connections and proof of the winning strategy in 8×8 Hex. chess.cs.uec.ac.jp/˜noshita/hex88proof.pdf, 2006.

[134] Kevin O'Gorman. Hex theory and proofs project. hex.kosmanor.com/hex/theory.html, 2005.

[135] Kevin O'Gorman. The OHex board. hex.kosmanor.com/hex-bin/board/, 2008.

[136] Mehmet A. Orgun and John Thornton, editors. *AI 2007: Advances in Artificial Intelligence, 20th Australian Joint Conference on Artificial Intelligence, Gold Coast, Australia, December 2-6, 2007, Proceedings*, volume 4830 of *Lecture Notes in Computer Science*. Springer, 2007.

[137] Yuval Peres, Oded Schramm, Scott Sheffield, and David B. Wilson. Random-turn Hex and other selection games. *American Mathematical Monthly*, 114:373–387, May 2007.

[138] Klutz Press, editor. *The Book of Classic Board Games*. Klutz Press, Palo Alto, California, 1991.

[139] Rune Rasmussen. A multiple rule framework that improves strategies for automatic Hex players. Master's thesis, Smart Devices Laboratory School of SEDC, Queensland University of Technology, Brisbane, Queensland, Australia, 2004.

[140] Rune Rasmussen. *Algorithmic Approaches for Playing and Solving Shannon Games*. PhD thesis, Queensland University of Technology, Brisbane, Queensland, Australia, 2007.

[141] Rune Rasmussen. Rune's home page. www.members.optusnet.com.au/xyzrune/, 2008.

[142] Rune Rasmussen and Frédéric Maire. An extension of the H-search algorithm for artificial Hex players. In Webb and Yu [175], pages 646–657.

[143] Rune Rasmussen, Frédéric Maire, and Ross Hayward. A move generating algorithm for Hex solvers. In Sattar and Kang [148], pages 637–646.

[144] Rune K. Rasmussen, Frédéric D. Maire, and Ross F. Hayward. A template matching table for speeding-up game-tree searches for Hex. In Orgun and Thornton [136], pages 283–292.

[145] Stefan Reisch. Hex ist PSPACE-vollständig. *Acta Informatica*, 15:167–191, 1981.

[146] Abdallah Saffidine. Utilization d'UCT au Hex. Technical report, Ecole Normale Superieure de Lyon, 2008.

[147] *SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings*, 2004.

[148] Abdul Sattar and Byeong Ho Kang, editors. *AI 2006: Advances in Artificial Intelligence, 19th Australian Joint Conference on Artificial Intelligence, Hobart, Australia, December 4-8, 2006, Proceedings*, volume 4304 of *Lecture Notes in Computer Science*. Springer, 2006.

[149] Jonathan Schaeffer. Conspiracy numbers. *Artificial Intelligence*, 43(1):67–84, 1990.

[150] Jonathan Schaeffer, Yngvi Björnsson, Neil Burch, Robert Lake, Paul Lu, and Steve Sutphen. Building the checkers 10-piece endgame databases. In van den Herik et al. [165], pages 193–210.

[151] Jonathan Schaeffer, Neil Burch, Yngvi Bjornsson, Akihiro Kishimoto, Martin Muller, Rob Lake, Paul Lu, and Steve Sutphen. Checkers is solved. *Science*, 317:1518–1522, 2007.

[152] Jonathan Schaeffer, Joseph Culberson, Norman Treloar, Brent Knight, Paul Lu, and Duane Szafron. Reviving the game of checkers. In *Games of No Chance*, pages 119–136. Cambridge University Press, 1991.

[153] Jonathan Schaeffer, Joseph C. Culberson, Norman Treloar, Brent Knight, Paul Lu, and Duane Szafron. A world championship caliber checkers program. *Artif. Intell.*, 53(2–3):273–289, 1992.

[154] Jonathan Schaeffer and Robert Lake. Solving the game of Checkers. In Richard J. Nowakowski, editor, *Games of No Chance*, volume 29 of *MSRI Publications*, pages 119–133. Cambridge University Press, Cambridge, 1996.

[155] Jonathan Schaeffer, Martin Müller, and Yngvi Björnsson, editors. *Computers and Games, Third International Conference, CG 2002, Edmonton, Canada, July 25-27, 2002, Revised Papers*, volume 2883 of *Lecture Notes in Computer Science*. Springer, 2003.

[156] Craige Schensted and Charles Titus. *Mudcrack Y and Poly-Y*. Neo Press, Peaks Island, Maine, 1975.

[157] Sensei's library: Hex. senseis.xmp.net/?Hex, 2009.

[158] Claude E. Shannon. Computers and automata. *Proceedings of the Institute of Radio Engineers*, 41:1234–1241, 1953.

[159] Ian Stewart. Mathematical recreations: Hex marks the spot. *Scientific American*, pages 100–103, September 2000.

[160] Walter Stromquist. Winning paths in n-by-infinity Hex. *INTEGERS: Electronic Journal of Combinatorial Number Theory*, 7:1–3, 2007.

[161] Robert Endre Tarjan. Decomposition by clique separators. *Discrete Mathematics*, 55(2):221–232, 1985.

[162] Ken Thompson. Retrograde analysis of certain endgames. *International Computer Chess Association Journal*, 9(3):131–139, 1986.

[163] H. Jaap van den Herik, Paolo Ciancarini, and H.H.L.M. Donkers, editors. *Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers*, volume 4630 of *Lecture Notes in Computer Science*. Springer, 2007.

[164] H. Jaap van den Herik, Shun-Chin Hsu, Tsan-Sheng Hsu, and H.H.L.M. Donkers, editors. *Advances in Computer Games, 11th International Conference, ACG 2005, Taipei, Taiwan, September 6-9, 2005. Revised Papers*, volume 4250 of *Lecture Notes in Computer Science*. Springer, 2006.

[165] H. Jaap van den Herik, Hiroyuki Iida, and Ernst A. Heinz, editors. *Advances in Computer Games, Many Games, Many Challenges, 10th International Conference, ACG 2003, Graz, Austria, November 24-27, 2003, Revised Papers*, volume 263 of *IFIP*. Kluwer, 2004.

[166] H. Jaap van den Herik, Jos W.H.M. Uiterwijk, and Jack van Rijswijck. Games solved: Now and in the future. *Artificial Intelligence*, 134(1–2):277–311, 2002.

[167] H. Jaap van den Herik, Xinhe Xu, Zongmin Ma, and Mark H.M. Winands, editors. *Computers and Games, 6th International Conference, CG 2008, Beijing, China, September 29 - October 1, 2008. Proceedings*, volume 5131 of *Lecture Notes in Computer Science*. Springer, 2008.

[168] J. van den Herik and P. Spronck, editors. *Advances in Computer Games, 12th International Conference, ACG 2009, Pamplona, Spain, 2009. Revised Papers*, volume 6048 of *Lecture Notes in Computer Science*. Springer, 2010.

[169] Jack van Rijswijck. Computer Hex: Are bees better than fruitflies? Master's thesis, University of Alberta, Edmonton, Canada, 2000.

[170] Jack van Rijswijck. Queenbee's home page. www.cs.ualberta.ca/˜queenbee/, 2000.

[171] Jack van Rijswijck. Search and evaluation in Hex. Technical report, University of Alberta, Edmonton, Canada, 2002. sites.google.com/site/javhar1/javharpublications.

[172] Jack van Rijswijck. Binary combinatorial games. Combinatorial Game Theory conference, Banff. sites.google.com/site/javhar1/javharpublications, 2005.

[173] Jack van Rijswijck. *Set Colouring Games*. PhD thesis, University of Alberta, Edmonton, Alberta, Canada, 2006.

[174] Steven Walczak. Improving opening book performance through modeling of chess opponents. In *CSC '96: Proceedings of the 1996 ACM 24th annual conference on Computer science*, pages 53–57, New York, NY, USA, 1996. ACM.

[175] Geoffrey I. Webb and Xinghuo Yu, editors. *AI 2004: Advances in Artificial Intelligence, 17th Australian Joint Conference on Artificial Intelligence, Cairns, Australia, December 4-6, 2004, Proceedings*, volume 3339 of *Lecture Notes in Computer Science*. Springer, 2004.

[176] Sue Whitesides. An algorithm for finding clique cut-sets. *Inf. Process. Lett.*, 12(1):31–32, 1981.

[177] Jan Willemson and Yngvi Björnsson. Six wins Hex tournament. *ICGA Journal*, 27(3):180, 2004.

[178] Mark H.M. Winands, Yngvi Björnsson, and Jahn-Takeshi Saito. Monte-Carlo tree search solver. In van den Herik et al. [167], pages 25–36.

[179] M.H.M. Winands. *Informed Search in Complex Games*. PhD thesis, Universiteit Maastricht, Maastricht, Netherlands, 2004.

[180] Yōhei Yamasaki. Theory of division games. *Publications of the Research Institute for Mathematical Sciences*, 14(2):337–358, 1978.

[181] Jing Yang. Jing Yang's web site. www.ee.umanitoba.ca/˜jingyang/, 2003–2008.

[182] Jing Yang, Simon Liao, and Mirek Pawlak. A decomposition method for finding solution in game Hex 7x7. In Ning [130], pages 96–111.

[183] Jing Yang, Simon Liao, and Mirek Pawlak. Another solution for Hex 7x7. Technical report, University of Manitoba, Winnipeg, Manitoba, Canada, 2002.

[184] Jing Yang, Simon Liao, and Mirek Pawlak. New winning and losing positions for 7×7 Hex. In Schaeffer et al. [155], pages 230–248.

[185] Ling Zhao and Martin Müller. Game-SAT: A preliminary report. In SAT2004 [147].

Appendix A

Probing the 4-3-2 Virtual Connection

Border bridge carrier fillin resulted in roughly a tenfold speedup when Hayward *et al.* first solved the 7×7 Hex openings [83]. Given these benefits and the fact that the 4-3-2 VC appears frequently, Hayward posed the question: "When are probes of a Black 4-3-2 inferior?". This appendix summarizes what is known thus far.

A.1 Winning Probes

Probes of an opponent's border bridge are dead-reversible moves, so such probes cannot be unique winning moves in any Hex position. However, probes of an opponent's 4-3-2 are not in general inferior, since probes 1, 2, 4 of an opponent's 4-3-2 can be unique winning moves. See Figure A.1, and recall the 4-3-2 carrier labelling from Figure 3.3.

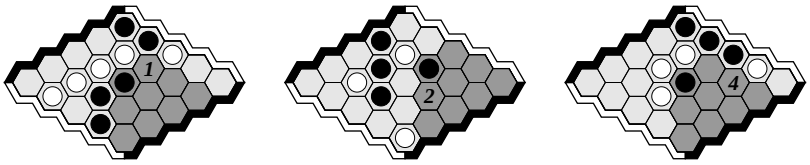


Figure A.1: Probes 1, 2, 4 of a Black 4-3-2 VC can each be a unique winning move for White.

It follows that one cannot unconditionally prune these three probes. It might be speculated that these three probes dominate all other 4-3-2 probes, however this also turns out to be false as probes 3 and 5 can be winning moves when probes 1, 2, 4 are all losing moves. See Figure A.2.

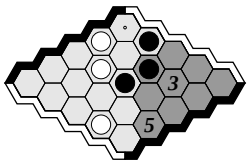


Figure A.2: White's only winning moves are the dotted cell and 4-3-2 probes 3 and 5.

In this last example, probes 3 and 5 are not the only winning moves for White, as an external

winning move exists. Furthermore, probes 3 and 5 merely delay an eventual external winning move, and are not necessary for White's win. More formally, probes 3 and 5 are not part of any shortest White winning strategy.

We know of no Hex state in which one of the 4-3-2 probes 3, 5, 6, 7, 8 is a unique winning move, nor of a Hex state in which one of the 4-3-2 probes 6, 7, 8 wins but probes 1, 2, 4 all lose. Probes 1, 2, 4 seem to be stronger than the others, so we conjectured [88] the following:

Conjecture 1 *Let H be a Hex position, and let C be the carrier of a Black 4-3-2 in H. If White has a winning move in H^W, then White has a winning move that is not probe 3, 5, 6, 7, or 8 of C.*

This conjecture is still open. In the rest of this appendix we find supporting evidence and conditions under which it holds.

A.2 Maintained 4-3-2 Virtual Connections

The first set of conditions under which Conjecture 1 holds, is the assumption that the 4-3-2 VC will be maintained against all opponent probes; we call this the *maintenance assumption*. This is often the case in practice, although admittedly it is rare to know this with absolute certainty. Given the maintenance assumption, our result is actually slightly stronger in that we prove the domination of probes $\{3, 5, 6, 7, 8\}$ by probes $\{1, 4\}$.

Theorem 16 *Let H be a Hex position, and let C be the carrier of a Black 4-3-2 in H for which the maintenance assumption holds. Then the White probes 3, 5, 6, 7, 8 of C are each dominated by at least one of the White probes 1, 4 of C.*

To prove Theorem 16, we must consider Black's possible maintenance responses to these probes. We start by considering the dominating probes 1 and 4.

Lemma 16 *Let H be a Hex position, and let C be the carrier of a Black 4-3-2 in H for which the maintenance assumption holds. If White probes at cell 1 of C, then Black will respond at cell 2 or cell 4 of C.*

Proof: First note that cells 2 and 4 do indeed maintain the Black 4-3-2 against White probe 1. In $H + W(1) + B(2)$, set {3,5,6,7} is Black captured. In $H + W(1) + B(4)$, set {7,8} is Black captured. By capture-domination the result follows. □

Lemma 17 *Let H be a Hex position, and let C be the carrier of a Black 4-3-2 in H for which the maintenance assumption holds. If White probes at cell 4 of C, then Black will respond at cell 2 of C.*

Proof: First note that cells 2, 3, 5, 6, 7 are the only moves that maintain the Black 4-3-2 against White probe 4. In $H + W(4) + B(2)$, set $\{3, 5, 6, 7\}$ is Black captured. By capture-domination the result follows. □

We shall prove domination by showing that if White probes 1 and 4 are losing, then probes 3, 5, 6, 7, 8 are also losing. If White probes 1 and 4 are losing, then the maintenance assumption implies the following by Lemmas 16 and 17:

- At least one of $H + W(1) + B(2)$ and $H + W(1) + B(4)$ is a Black win.
- $H + W(4) + B(2)$ is a Black win.

We can relate these three positions to those obtained by other probes, as in the following lemma:

Lemma 18 *Let H be a Hex position with a Black 4-3-2. Then $H + W(1) + B(2) \geq_W H + W(6) + B(4)$.*

Proof: In position $H + W(6) + B(4)$, set $\{7, 8\}$ is Black captured, and Black can adopt a pairing strategy on $\{1, 3\}$ and $\{2, 5\}$ that maintains the 4-3-2. Thus we need only show that this pairing strategy always results in a completion that is White dominated by $H + W(1) + B(2)$.

Since cell 3 is White dead-reversible to cell 1 and they are paired, then without loss of generality we can assume White probes at 1 and Black maintains at 3. Given that cells 3 and 7 are Black, then the White cell 6 is actually dead, and so can be recoloured Black. But then cells 2 and 5 are Black captured, and so Black's pairing strategy there will ensure that any White move to 2 or 5 becomes dead. Thus any completion of the 4-3-2 in $H+W(6)+B(4)$ with Black adopting the given pairing strategy is equivalent to $H + W(1) + B(\{2, 3, 4, 5, 6, 7, 8\})$. Since $H + W(1) + B(2) \geq_W H + W(1) + B(\{2, 3, 4, 5, 6, 7, 8\})$, this concludes the proof. □

Applying Lemma 18 to the assumed conditions of our Theorem, we can conclude that either White probe 6 is losing (to Black maintaining at 4) or White probe 1 wins against Black maintaining at 2, the latter implying that White probe 1 loses against Black maintaining at 4.

Many 4-3-2 probe-maintenance position domination results can be deduced using neighbourhood domination and induced path domination, as illustrated in §3.4 and §3.5; some of these results are summarized in Figure A.3. We now have all the tools required to prove Theorem 16:

Proof: Assume White probes 1, 4 of C lose to Black maintaining the 4-3-2. Then we need only show that White probes 3, 5, 6, 7, 8 all lose to Black maintaining the 4-3-2.

By neighbourhood domination, $H + W(4) + B(2) \geq_W H + W(3) + B(2)$ since set { 5, 6 } is Black captured. Similarly, $H + W(4) + B(2) \geq_W H + W(7) + B(2)$ since set { 5, 6 } is Black captured. By induced path domination, $H + W(4) + B(2) \geq_W H + W(8) + B(2)$ since set { 3, 5, 6, 7 } is Black captured. Since by assumption White probe 4 is losing to Black maintaining the 4-3-2, then by Lemma 17 White probes 3, 7, 8 all lose to Black maintaining at cell 2.

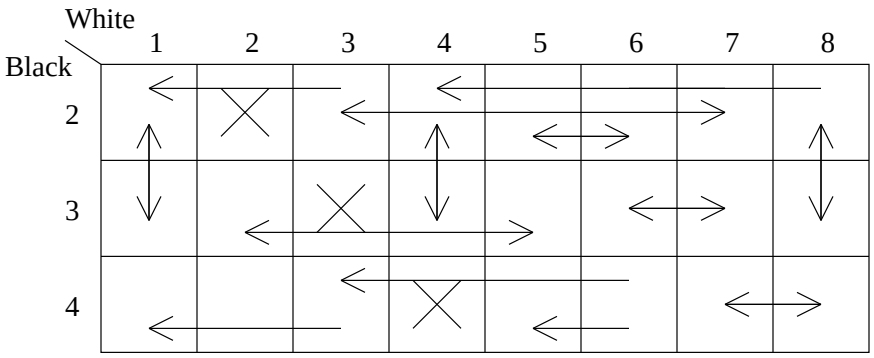


Figure A.3: Some White domination relations among 4-3-2 White probe, Black maintenance positions. Each unidirectional arc points from a position to a White dominating position, while bidirectional arcs indicate equivalent positions. X indicates an impossible position. Arcs which can be deduced by domination transitivity are omitted for clarity.

By Lemma 18, $H + W(1) + B(2) \geq_W H + W(6) + B(4)$. By induced path domination, $H + W(1) + B(4) \geq_W H + W(6) + B(4)$ since set { 7, 8 } is Black captured. Since by assumption White probe 1 is losing to Black maintaining the 4-3-2, then by Lemma 16 White probe 6 loses to Black maintaining at cell 4.

If White probes C at 5, then consider a Black response at cell 2. In the position $H+W(5)+B(2)$, cell 5 cannot be on any minimal White winning paths without further moves in C. If White follows up with a probe at 1, Black can respond at cell 4. In $H + W(\{1, 5\}) + B(\{2, 4\})$, set $\{3, 6, 7, 8\}$ is Black captured and cell 5 is dead, so this position Black dominates both $H + W(1) + B(2)$ and $H + W(1) + B(4)$, and thus must be a Black win by our assumption.

If White follows up with probe 6, then Black can respond at cell 4. In position $H + W(\{5, 6\}) + B(\{2, 4\})$, set $\{7, 8\}$ is Black captured, and so cell 6 is White induced path dominated by cell 1. Thus this position Black dominates $H + W(\{1, 5\}) + B(\{2, 4\})$, which as we have just shown is a Black win.

If White instead follows up with one of the probes 3, 4, 7, or 8 then Black can respond at cell 6. In $H + W(\{x, 5\}) + B(\{2, 6\})$ with $x \in \{3, 4, 7, 8\}$, cell 5 is dead, so each of these four positions is equivalent to or Black (neighbourhood or induced path) dominates $H + W(4) + B(2)$. Thus these four positions are all Black wins by our assumption.

Thus White has no winning follow up probes in C, and thus position $H + W(5) + B(2)$ is a Black win. □

A.3 Acute Corner 4-3-2 Virtual Connections

Next we consider restrictions of Conjecture 1 where the 4-3-2 VC carrier is in an acute corner. The 4-3-2 can be oriented in two ways, depending on the location of the non-border endpoint. See Figure A.4. Due to the coordinate system commonly used in Hex, we call them the a3 4-3-2 and b3

4-3-2 respectively.

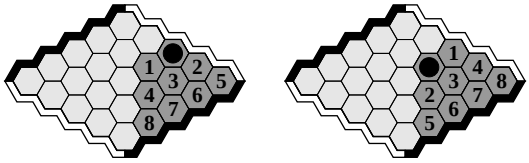


Figure A.4: Acute corner a3 4-3-2 and b3 4-3-2 virtual connections.

When probing such 4-3-2s, the nearby White border makes capturing easier, yielding the following probe results:

Lemma 19 *Let H be a Hex position, and let C be the carrier of a Black a3 4-3-2 in H. Then White probes { 2, 3, 5, 7 } of C are inferior.*

Proof: White probe 4 creates a star decomposition in the acute corner, so $H + W(4) \equiv H + W(\{2, 4, 5\}) \equiv H + W(\{3, 4, 7\})$, and thus probe 4 capture-dominates probes 2, 3, 5, 7. □

Note that White probe 6 also capture-dominates probes 2 and 5 due to the border bridge it creates. Thus probes 1, 4, 6, 8 dominate probes 2, 3, 5, 7 for the a3 4-3-2, which is both stronger and weaker than our original conjecture: stronger because it prunes out probe 2, weaker because probes 6 and 8 remain.

Lemma 20 *Let H be a Hex position, and let C be the carrier of a Black b3 4-3-2 in H. Then White probes { 1, 3, 4, 5, 6, 7, 8 } of C are inferior.*

Proof: All White probes in { 1, 3, 4, 5, 6, 7, 8 } are dead-reversible with Black reverser cell 2. □

Aside from pruning acute 4-3-2 probes from consideration, one can also prune moves when the acute corner is still uncoloured:

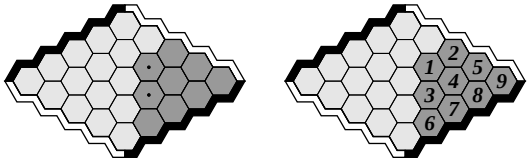


Figure A.5: The two dotted cells Black dominate all other shaded cells in the uncoloured acute corner.

Theorem 17 *Let H be a Hex position, and assume the nine cells forming a 4-3-2 shape in an acute corner are all uncoloured, labelled as in Figure A.5. Then Black can prune cells { 2, 4, 5, 6, 7, 8, 9 } from consideration.*

Proof: As shown in Figure 3.1, cell 3 capture-dominates cells { 4, 5, 6, 7, 8, 9 }, thus all that remains is to prune cell 2 from consideration. We shall do so by proving that $H + B(1) \geq_B H + B(2)$.

If Black is the first to play in the acute corner, we note that $H + B(\{1,3\})$ captures all other cells in the acute corner. Thus by monotonicity $H + B(\{1,3\}) \geq_B H + B(\{2,x\})$ for all $x \in \{1,3,4,5,6,7,8,9\}$.

If White is the first to play in the acute corner, then by Lemma 20 White must probe $H + B(1)$ at cell 3. This creates a star decomposition, so $H + B(1) + W(3) \equiv H + B(1) + W(\{3,5,9\})$. By neighbourhood domination, $H + B(1) + W(\{3,5,9\}) \geq_B H + B(2) + W(\{3,5,9\})$. Once again, due to a star decomposition $H + B(2) + W(\{3,5,9\}) \equiv H + B(2) + W(3)$. Combining these results, we conclude that $H + B(1) + W(3) \geq_B H + B(2) + W(3)$.

Since Black prefers $H + B(1)$ to $H + B(2)$ regardless of who moves first, then by definition it follows that $H + B(1) \geq_B H + B(2)$. □

Since in practice the acute corner is often uncoloured or with only one coloured cell, these results are often applicable when solving or playing positions.

Appendix B

Handicap Strategy

We give an explicit $\lceil \frac{n+1}{6} \rceil$ handicap strategy for Hex on the $n \times n$ board: the first player is guaranteed victory if they are allowed to colour $\lceil \frac{n+1}{6} \rceil$ cells on their first move.

Our handicap strategy colours handicap cells in the second row — so that all cells in the first row are Black fillin via dead, captured, and permanently inferior cells — and applies Shannon's pairing strategy for irregular boards to the fillin-reduced board. The resulting handicap strategy is both explicit and efficient.

B.1 Handicap Locations and Fillin

We begin by describing the location of Black's initial $\lceil \frac{n+1}{6} \rceil$ cell colourings on the $n \times n$ Hex board. Since trivial first player strategies are known for $n \times n$ boards with n at most five, we focus exclusively on $n \times n$ boards with n at least six.

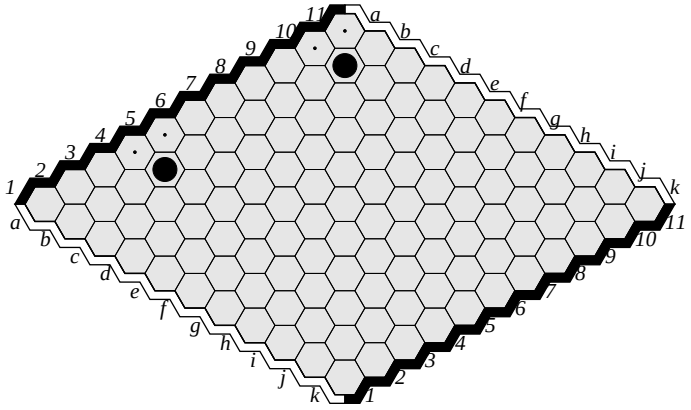


Figure B.1: Handicap cell colouring: handicap cells are Black-coloured and primary cells are dotted.

The *handicap cells* are always in the second row from the Black border, and located in column 4 and in columns $n - 1 - 6j$ for each j in $\{0, \ldots, \lfloor \frac{n}{6} \rfloor - 1\}$. The *primary cells* are the first row cells adjacent to a handicap cell; that is, the cells in the carrier of a handicap cell border bridge. See Figure B.1.

Lemma 21 *For any $n \times n$ Hex board with $n \geq 6$, the above handicap initialization specifies the colouring of $\lceil \frac{n+1}{6} \rceil$ handicap cells.*

Proof: Since $j \leq \lfloor \frac{n}{6} \rfloor - 1$, it follows that $n - 1 - 6j \geq 5$, so the handicap cell in column four is distinct. Since j iterates from zero to $\lfloor \frac{n}{6} \rfloor - 1$ inclusive, there are exactly $\lfloor \frac{n}{6} \rfloor + 1$ handicap cells including the one in column four. Finally, $\lfloor \frac{n}{6} \rfloor + 1 = \lceil \frac{n+1}{6} \rceil$. □

We now show that colouring the handicap cells results in the entire first row being Black fillin.

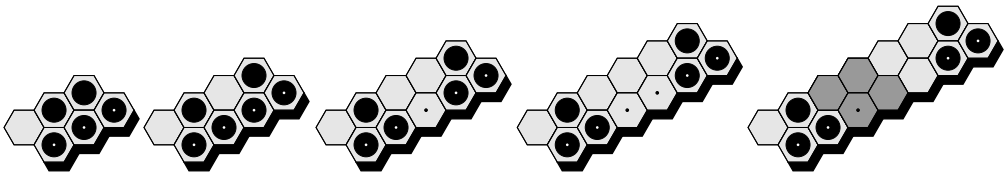


Figure B.2: Gaps between consecutive handicap cells.

Lemma 22 *Let H_1 be the Hex position obtained from the handicap cell colouring of a $n \times n$ Hex board, and let H_2 be the position obtained from H_1 by Black-colouring the entire first row. Then $H_1 \equiv H_2$.*

Proof: All handicap cells are on the second row, so all primary cells are in border bridge carriers, and so are Black captured. See the two leftmost cases in Figure B.2.

The handicap cells are never separated by more than five uncoloured cells in the second row, so their respective primary cells are never separated by more than four uncoloured cells in the first row. If a primary cell gap is of size one, the uncoloured cell is dead, and can be coloured Black without changing the value of H_1. If a primary cell gap is of size two, the uncoloured cells are Black captured. See the next two cases in Figure B.2.

If the primary cell gap is of size three or four, then the first row uncoloured cells neighbouring primary cells are Black permanently inferior. See the rightmost case in Figure B.2. Colouring all such permanently inferior cells and iterating on this reduced board, any remaining first row gap is of size one or two, so once again the uncoloured first row cells are dead or Black captured.

For the gap between the column four handicap cell and the White border, the first row uncoloured cell neighbouring a primary cell is permanently inferior, and the fillin-reduced first row gap of size two is Black captured.

Since H_1 has been transformed to H_2 via Black fillin, it follows that these two positions have the same value. □

B.2 Existence Proof and Explicit Strategy

Theorem 18 *On a Hex board of dimension $n \geq 6$, Black has a winning $\lceil \frac{n+1}{6} \rceil$ handicap strategy.*

Proof: By Lemma 21, $\lceil \frac{n+1}{6} \rceil$ handicap cells are coloured on a Hex board of dimension $n \geq 6$. By Lemma 22, this initial handicap position produces Black fillin that colours the entire first row, and so is equivalent to an $(n-1) \times n$ Hex board in Black's favour (*i.e.,* with Black traversing the shorter distance) with $\lceil \frac{n+1}{6} \rceil$ Black-coloured cells on the first row of this irregular board. By monotonicity this position Black dominates an initial $(n-1) \times n$ Hex board in Black's favour, and the latter is a Black win by Shannon's pairing strategy. □

Theorem 18 claims the existence of a handicap strategy. However, by the proofs of Lemma 22 and Theorem 18, simply maintaining the fillin with the corresponding inferior cell strategies (in the order deduced, when there are conflicts; see Theorem 5) and combining this with Shannon's pairing strategy yields an explicit handicap strategy:

- Colour the $\lceil \frac{n+1}{6} \rceil$ handicap cells, as specified above.
- In response to each White move, follow the earliest rule that is both applicable and legal:
    1. If White colours a primary cell, then colour a neighbouring primary cell (*i.e.,* its killer in the primary cell captured set).
    2. If White colours a dead-reversible cell in the carrier of a first row permanently inferior cell, then colour its killer within the permanently inferior carrier.
    3. If White colours a dead-reversible cell in a first row Black captured set, then colour its killer.
    4. If White colours a first row dead cell, then colour any uncoloured cell.
    5. If White colours a cell outside of the first row, then colour its partner in the $(n-1) \times n$ Shannon pairing strategy.
    6. Colour any uncoloured cell.

This is the most efficient handicap strategy known for Hex on all unsolved board sizes (*i.e.,* boards of dimension at least ten). In particular, we note that the late Claude Berge, who was a Hex enthusiast [21, 22], would often give beginners three handicap cells on 11×11 Hex boards, suggesting that he did not expect them to find a winning strategy requiring fewer than four handicap cells. We would like to think that our two-cell handicap strategy for 11×11 Hex would have surprised him.

Appendix C

Olympiad Games

This chapter summarizes the performance of Wolve and MoHex in the 2008 and 2009 International Computer Olympiads. In both of these tournaments the participants were Wolve, MoHex, Six, and Yopt.

For each game we display the entire game sequence, the first and second player (with the winner in bold), any endgame states our solver can identify as Black or White winning, and any additional observations. Similar commentary appears in our tournament reports [12, 13]. However, because our solver has improved significantly since these tournaments took place, the commentary below is more thorough.

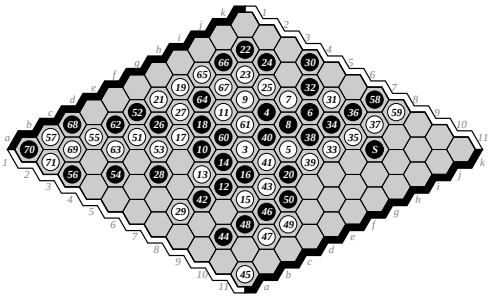
C.1 2008 Olympiad

	Wolve	MoHex	Six	Yopt	total	result
Wolve		1-3	4-0	4-0	9-3	gold
MoHex	3-1		2-2	3-1	8-4	silver
Six	0-4	2-2		2-2	4-8	bronze
Yopt	0-4	1-3	2-2		3-9	4th

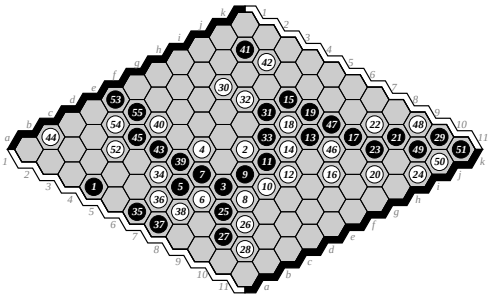
Table C.1: 2008 Hex Computer Olympiad results.

As mentioned in Chapter 6, MoHex and Wolve share many features, such as their algorithms for computing connection strategies and identifying inferior cells. However, sometimes features which aided one program were detrimental to the other. For this reason, during this competition MoHex had stronger inferior cell and connection strategy computations, and only Wolve used an opening book.

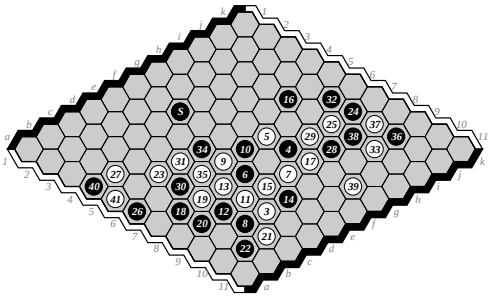
Gábor Melis did not attend this tournament, so Six's opening moves were selected by Nathan Sturtevant. The tournament had two rounds, played on distinct days. In each round, each player opened once against each opponent.



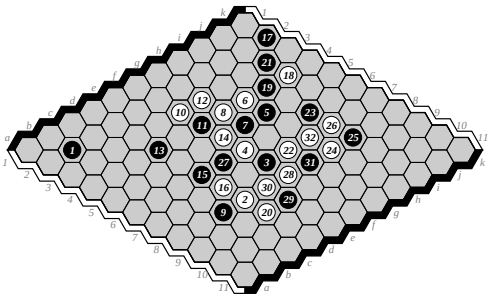
Game 1: **Six** - MoHex
Black winning: 17–19
White winning: 20+
Commentary: 20.B[f8] is a blunder as 20.B[f3] is winning.



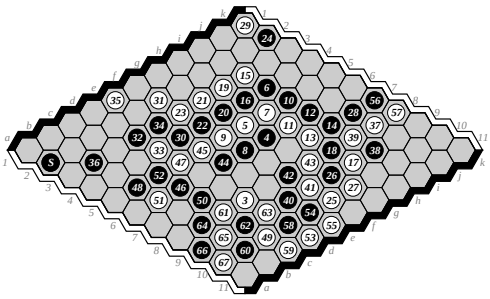
Game 2: **MoHex** - Six
Black winning: 20+
White winning:
Commentary: Throughout the game MoHex seems to have the advantage, as Six is constantly defending and is never given the chance to form a reasonable counterattack.



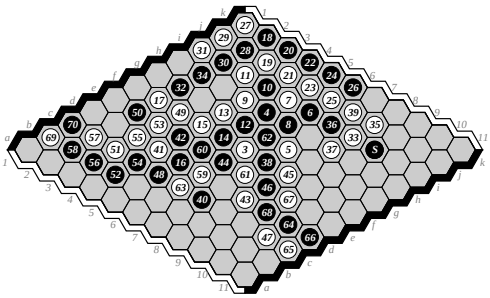
Game 3: **Wolve** - Yopt
Black winning:
White winning: 16+
Commentary: The opening move of f3 seems imbalanced, giving Yopt an initial advantage. However, Wolve plays an interesting opening, confusing the situation well (*e.g.*, move 9.W[e6]). 18.B[b7] by Yopt seems a bit weak, and is far easier for our solver to refute than 18.B[c7].



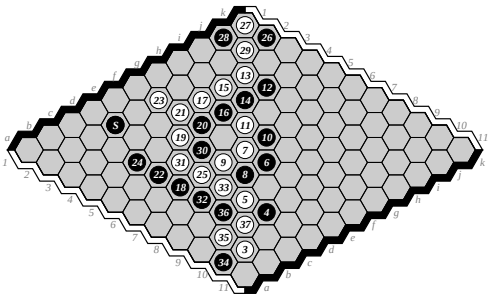
Game 4: Yopt - **Wolve**
Black winning:
White winning: 13+
Commentary: Wolve's first four moves were generated by its opening book. The opening is fairly standard and seems reasonably even until 9.B[c8]. Yopt played outside of the mustplay with 15.B[d6], resulting in an easy win for Wolve (15.B[c6] offered much stronger resistance).



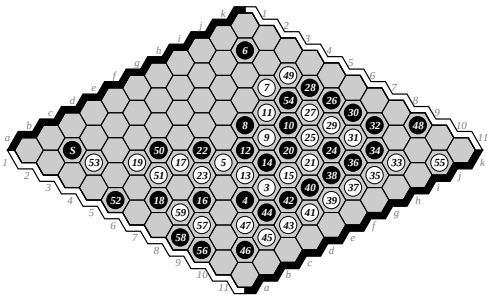
Game 5: **Wolve** - Six
Black winning: 27–39
White winning: 40+
Commentary: Six blunders with 40.B[e9], as 40.B[e8] is a (unique) winning move. Six's play seems strong until then.



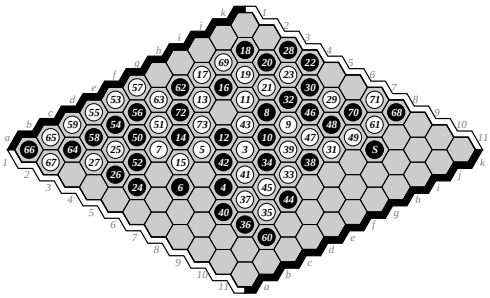
Game 6: Six - **Wolve**
Black winning: 33–39, 41+
White winning: 32, 40
Commentary: Six blunders with 33.W[i8] as 33.W[i7] is winning. Later Wolve blunders with 40.B[c7] as 40.B[e3] is winning. Six immediately returns the favour, making a major blunder with 41.W[d4] when 41.W[e3] is a simple win for White.



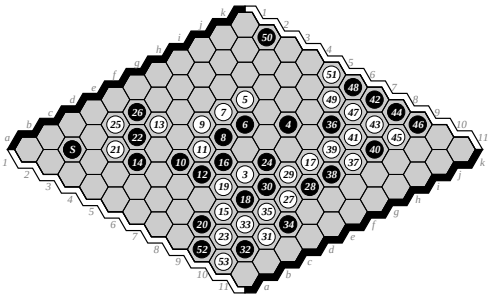
Game 7: **MoHex** - Yopt
Black winning:
White winning: 20+
Commentary: Yopt's 22.B[c5] is outside of the mustplay; this weak move could have been avoided with a basic implementation of H-search. 22.B[k2] provides much more resistance according to our solver. Move 20.B[f4] by Yopt is captured-reversible, and MoHex responds with its reverser.



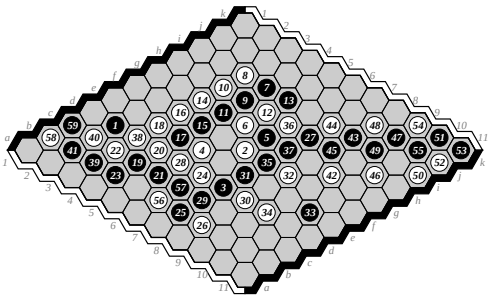
Game 8: **Yopt** - MoHex
Black winning:
White winning: 16+
Commentary: Move 6.B[j2] by MoHex seems very weak, giving Yopt a free move to strengthen the centre and/or create important border connections. Unsurprisingly, Yopt is winning in all solved continuations.



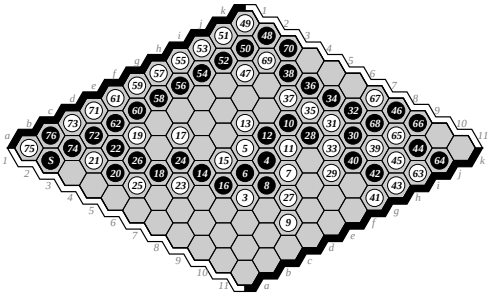
Game 9: **Six** - Yopt
Black winning:
White winning: 24+
Commentary: Opening play seems reasonable for both players. Move 26.B[b4] by Yopt is both dead-reversible and outside of the mustplay, greatly simplifying Six's advantage.



Game 10: **Yopt** - Six
Black winning: 25–35
White winning: 22–24, 36+
Commentary: 25.W[d2] is a dead-reversible move by Yopt, and a blunder as 25.W[e2] is winning. 36.B[i7] is a major blunder by Six, as 36.B[h8] is a trivial win.



Game 11: **MoHex** - Wolve
Black winning: 20+
White winning:
Commentary: Wolve's 22.W[c3] is a dead-reversible move, and it is killed by MoHex's response at 23.B[b4]. Such poorly-conceived probes occur frequently with the circuit evaluation function used by Six and Wolve.



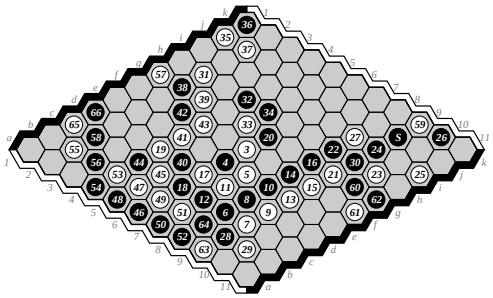
Game 12: Wolve - **MoHex**
Black winning: 27+
White winning: 26
Commentary: Wolve's 27.W[e9] is a blunder as 27.W[h9] is winning. 21.W[b3] is also a bit weak since it is star decomposition dominated by 21.W[d2].

C.1.1 Round 1

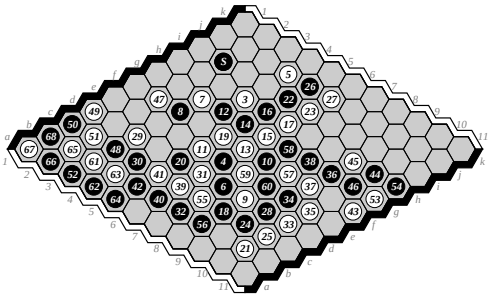
In the first round of the tournament, all games either had a fairly biased opening (*i.e.*, game states within the first 20 moves can be solved in a reasonable amount of time) or else there were blunders in the endgame stages. Games 2–4, 7–8, 11 are in the first category, games 5–6, 9–10, 12 are in the second category, and game 1 is in both.

Given this fact, stronger endgame play would have helped significantly (*i.e.*, whenever the opening was not overly biased), and so for the second round we added a simple endgame solver to Wolve and MoHex. This solver ran for 15 seconds prior to the player's normal search, but only after move 15 due to tournament time conditions. The success of this last-minute addition eventually led to our parallel solver.

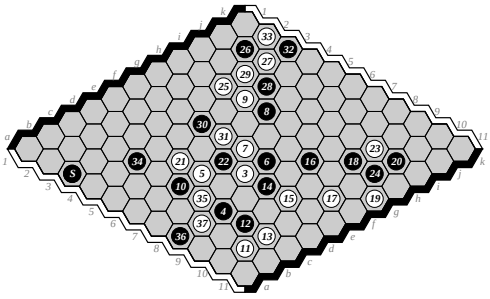
C.1.2 Round 2



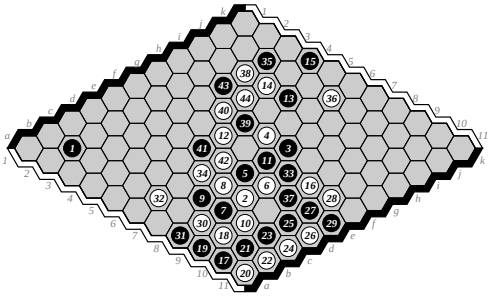
Game 13: Six - **MoHex**
Black winning: 21+
White winning:
Commentary: Six's play along the 9th row is quite weak, as it gives MoHex a strong wall of influence. Move 25.W[i11] by Six is dead-reversible, and MoHex kills it with 26.B[k10].



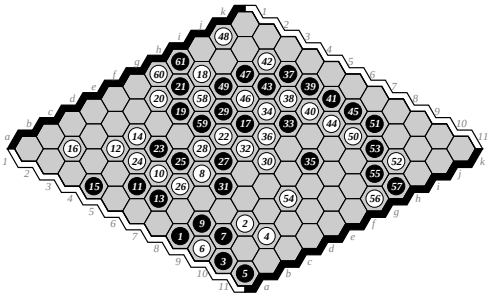
Game 14: MoHex - **Six**
Black winning: 27+
White winning: 20
Commentary: 21.W[b10] is likely a blunder by MoHex, as 21.W[b7] is winning and MoHex is losing shortly thereafter; our solver is not strong enough to solve the intermediate positions.



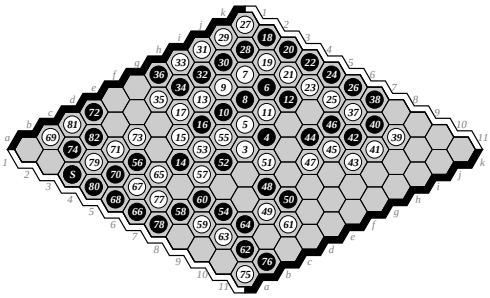
Game 15: **Wolve** - Yopt
Black winning:
White winning: 14+
Commentary: Wolve dominates this game throughout. Move 22.B[e6] by Yopt is particularly weak, being both dead-reversible and outside of the mustplay.



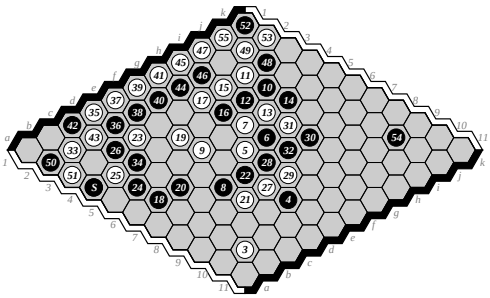
Game 16: Yopt - **Wolve**
Black winning:
White winning: 25+
Commentary: Again Wolve seems to have a strong advantage throughout. Move 33.B[f8] by Yopt is dead-reversible.



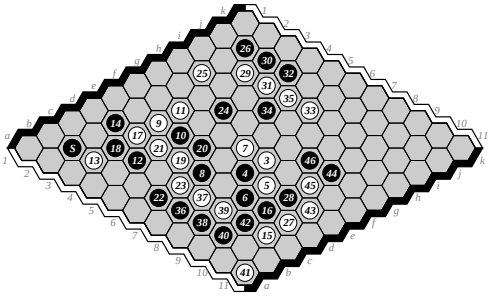
Game 17: **Wolve** - Six
Black winning: 28+
White winning:
Commentary: After 16.W[b2] it seems as though Wolve has fallen into a bad opening trap, with Six gaining all the influence. However, Wolve's probes of Six's VCs give ample compensation and allow Wolve to win.



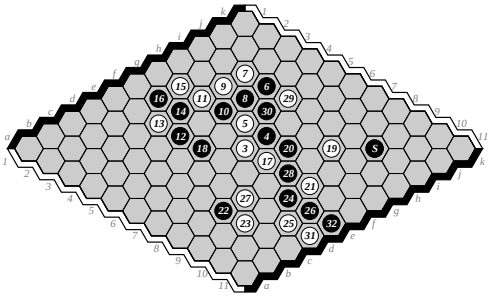
Game 18: Six - **Wolve**
Black winning: 45–47, 49+
White winning: 48
Commentary: This close game between Six and Wolve is the toughest to solve in the entire olympiad (in the number of moves played before it can be solved). 48.B[e8] is a blunder by Wolve, as 48.B[d8] is winning. 49.W[d9] is a blunder by Six, as 49.W[h6] and 49.W[f8] are both winning.



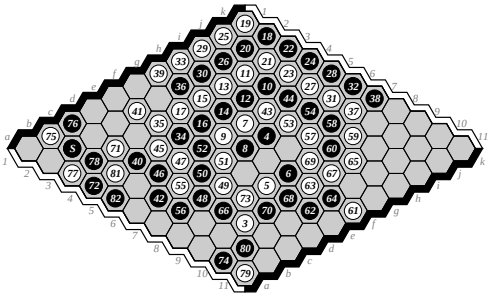
Game 19: **MoHex** - Yopt
Black winning:
White winning: 30+
Commentary: 30.B[h7] by Yopt seems weak, as 30.B[h6] gives far greater resistance to our solver.



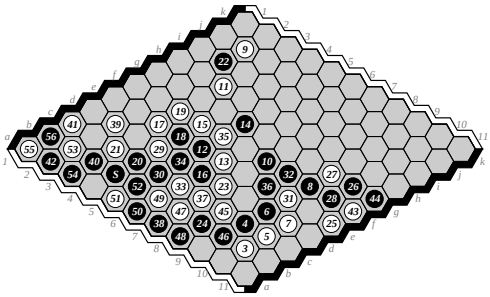
Game 20: Yopt - **MoHex**
Black winning: 19+
White winning:
Commentary: 17.W[d3] by Yopt is dead-reversible, and MoHex responds by killing it. Yopt's mustplay remains small from move 20 onwards.



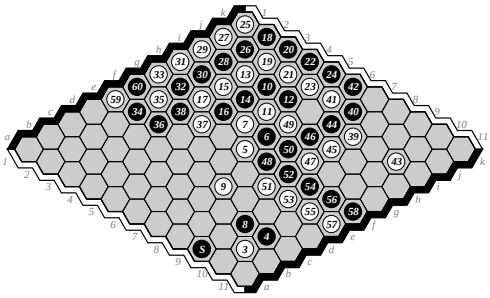
Game 21: Six - **Yopt**
Black winning: 17+
White winning:
Commentary: Yopt is winning this game very early on, presumably because of Six's imbalanced opening move. 15.W[g2] by Six also looks weak.



Game 22: Yopt - **Six**
Black winning: 41, 55+
White winning: 42–54
Commentary: 42.B[b6] is a blunder by Six as 42.B[b5] is winning, and 55.W[c6] is a blunder by Yopt as 55.W[b7] is winning. The latter blunder is detectable both by inferior cell analysis (it is dead-reversible) and by mustplay. This game ensures that Six gets the bronze medal and that Yopt finishes in fourth place.



Game 23: MoHex - **Wolve**
Black winning: 19+
White winning:
Commentary: MoHex's opening play seems quite weak, especially 9.W[j2]. A common flaw of MoHex is to favour b10 and j2, even when these moves are seemingly irrelevant to the current threats. This game clinches the gold medal for Wolve.



Game 24:	Wolve - **MoHex**
Black winning:	43+
White winning:	34–42
Commentary:	Even with the 15-second solver, Wolve played an endgame blunder with 43.W[i10] as 43.W[h9] is a winning move. Our current solver requires 20 minutes to identify the winning move, but Wolve's move is found to be losing in well under 1 minute.

Unlike in the first round, some games in the second round were neither overly biased in the opening, nor marred by endgame blunders. Examples include games 16, 17, and 19. It is unclear what contribution the solver made, as the number of (detectable) endgame mistakes by Wolve and MoHex was unchanged; the game positions were not rerun without solver to determine this difference. However, given situations like game 24, partial results from the solver could prove more useful.

C.1.3 Summary

Given that Six easily won the gold medal in the 2003-2006 competitions, its relative performance in the 2008 competition suggests that the level of automated Hex players improved greatly.

Overall Yopt played reasonably well in the opening and midgame, but its lack of mustplay pruning and inferior cell analysis resulted in several obvious blunders. MoHex's greatest weakness seemed to be inconsistent opening play. For instance, games 8 and 23 illustrate its capacity for catastrophic openings. On the other hand, MoHex was the only program to defeat gold medallist Wolve. Six and Wolve usually play well, but make a surprising number of endgame blunders as illustrated by games 5, 6, 10, 12, 18, 22 and 24.

C.2 2009 Olympiad

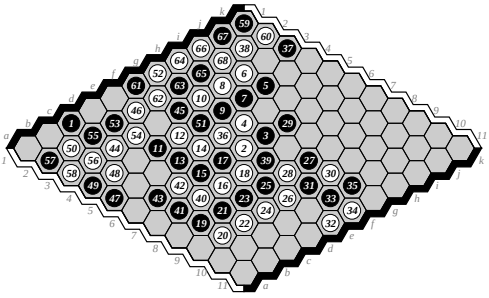
	MoHex	Wolve	Six	Yopt	total	result
MoHex		2-0	2-0	2-0	6-0	gold
Wolve	0-2		1-1	2-0	3-3	silver
Six	0-2	1-1		1-1	2-4	bronze
Yopt	0-2	0-2	1-1		1-5	4th

Table C.2: 2009 Hex Computer Olympiad results.

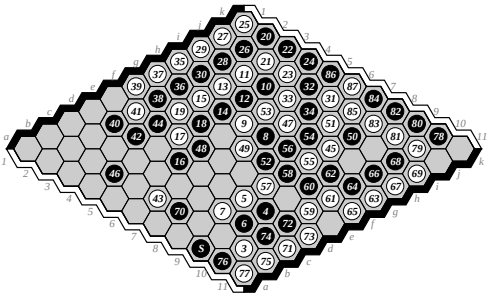
A fifth program, Bit2, registered but withdrew before the competition. Due to time constraints, the four remaining programs only opened once against each opponent. The tournament was played entirely in one day, with no program alterations between games.

Gábor Melis did not attend the tournament, so Six's opening moves were selected by Yngvi Björnsson and Jakub Pawlewicz. Wolve used three threads: two for parallelized board evaluation and one for a parallel solver. MoHex used eight threads: seven for parallelized Monte Carlo tree search, and one for a parallel solver. At this time our solver was still based on depth-first search.

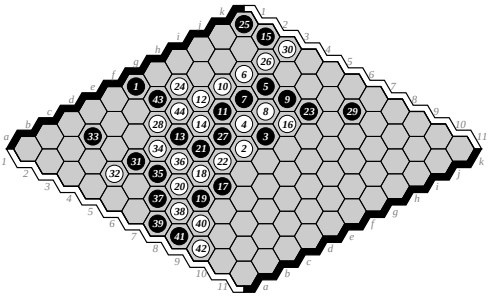
C.2.1 Round 1



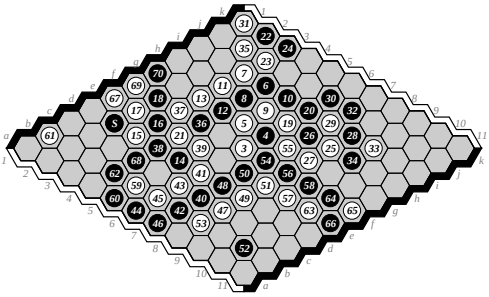
Game 1: Wolve - **MoHex**
Black winning:
White winning: 29+
Commentary: Neither program made an obvious error, and MoHex's evaluation scores suggest that this was a close game throughout.



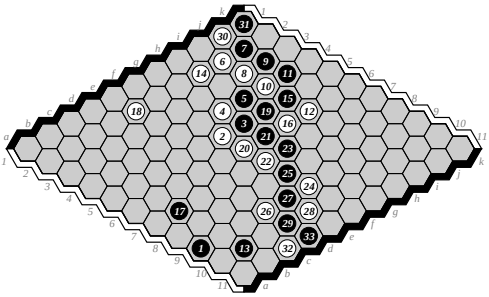
Game 2: **MoHex** - Wolve
Black winning:
White winning: 28+
Commentary: B[c4] provides far greater resistance to our solver then Wolve's moves 27, 29, 31, etc. By 17.W[e4] MoHex likes its position.



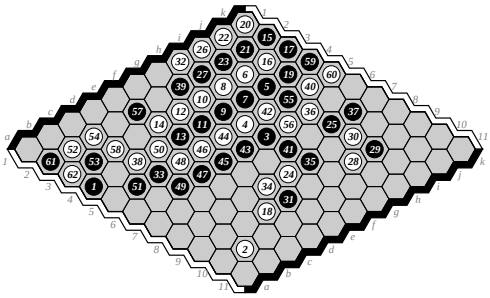
Game 3: Wolve - **Six**
Black winning:
White winning: 21+
Commentary: The opening seems balanced, but Wolve's situation seems to deteriorate badly around 17.B[d7]. Six's 24.W[g2] is a strong move.



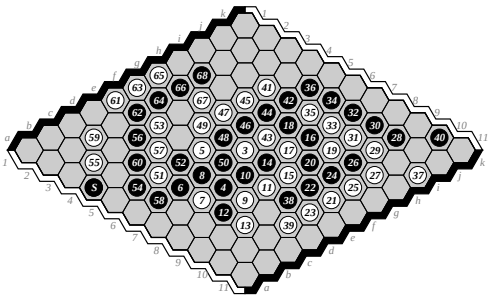
Game 4: Six - **Wolve**
Black winning: 27+
White winning:
Commentary: Six plays dead-reversible moves 17.W[e2] and 31.W[k1]. The former is killed by Wolve with 18.B[f2], but for the latter Wolve ignores killer j2 since it evaluates 32.B[j7] to be more important.



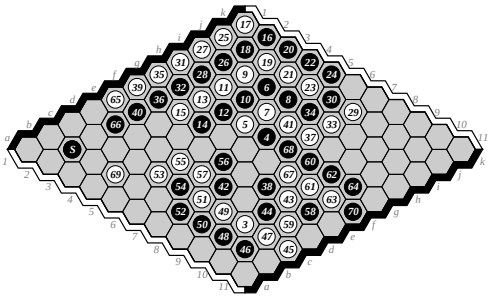
Game 5: **MoHex** - Six
Black winning: 14+
White winning:
Commentary: MoHex thinks 14.W[h2] is weak, as its evaluation score jumps to 0.75 when generating 15.B[i5].



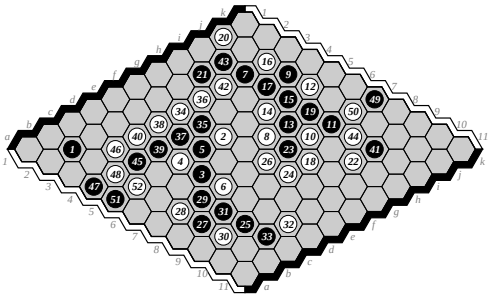
Game 6: Six - **MoHex**
Black winning: 33–36, 38
White winning: 37, 39+
Commentary: MoHex's opening play is weak, with Six having a strong wall of influence by move 19. MoHex manages to make the game close, and both players repeatedly blunder near the end until the parallel solver takes over for MoHex. Some winning alternatives are 37.B[c10], 38.W[j5], and 39.B[d4].



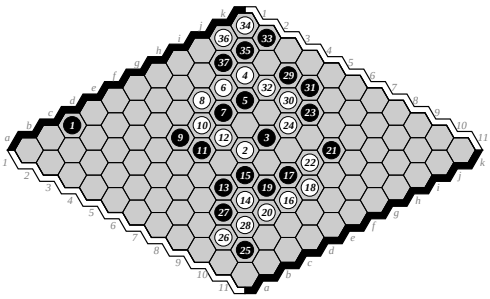
Game 7: Six - **Yopt**
Black winning: 33+
White winning:
Commentary: 37.W[i7] is a dead-reversible move by Six, but Yopt chooses not to kill it; our solver suggests that killer 38.B[k2] would have resulted in a simpler win. Nevertheless, Yopt seems to have the advantage throughout.



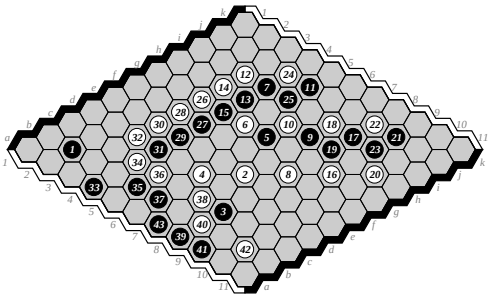
Game 8: Yopt - **Six**
Black winning: 37+
White winning:
Commentary: This is a close game, with Six pulling ahead. Yopt's play along the first row only helps Six, 36.B[f2] kills White's f3 chain, and 56.B[e6] is an elegant move by Six. Our solver suggests that Yopt's endgame resistance was a little weak.



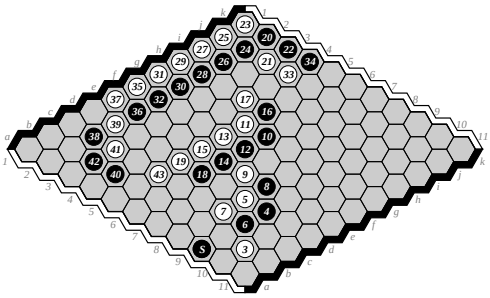
Game 9: Yopt - **Wolve**
Black winning:
White winning: 21+
Commentary: The opening of this game is rather unusual, particularly moves 9.B[j4] and 20.W[j1]. Wolve realizes it is winning by 28.W[b7], and so plays seemingly unusual moves from here on. Yopt has no endgame solver and does not see the win; during this endgame its evaluation score climbs above 0.9 before eventually identifying its loss.



Game 10: **Wolve** - Yopt
Black winning: 22+
White winning:
Commentary: Solver finds the winning 25.B[b10] for Wolve during the game. This result ensures gold for MoHex.



Game 11: Yopt - **MoHex**
Black winning: 20–30
White winning: 31+
Commentary: The game's opening is somewhat unusual, with parallel bridges forming and MoHex letting Yopt connect to one border easily. Yopt plays a dead-reversible move with 25.B[i5], but MoHex does not kill it. Yopt blunders with 31.B[d4], as both 31.B[j2] and 31.B[c4] are winning. This result ensures silver for Wolve.



Game 12:	**MoHex** - Yopt
Black winning:	35
White winning:	36+
Commentary:	This game is complicated, and very close until near the end. 36.B[b2] is a blunder by Yopt, as 36.B[b3] is winning. Identifying this winning move requires roughly two weeks of solving time, but Yopt's move is proven to be losing within ten seconds. This result ensures bronze for Six.

C.2.2 Summary

The quality of play in the 2009 tournament seems to have improved somewhat on the 2008 tournament performance. For instance, the number of solver-detectable blunders decreased from an average of 0.583 to 0.416 per game. Similarly, the average number of coloured cells in a game's shallowest solvable position increased from 22.79 to 26.25, suggesting that the 2009 opening positions are less polarized.

MoHex performed exceptionally well, not losing a single game despite its occasional weak opening play (*e.g.*, game 6). Wolve unfortunately did quite poorly compared to its normal measured performance, losing all of its games against MoHex and surprisingly even one against Six. Six was unchanged from the previous year and Yopt, while showing improvement from its previous version, again suffered from a lack of connection strategy deduction algorithms and inferior cell pruning.

Appendix D

Open Questions

Many important questions about Hex remain unsolved. We list them here in the hope that they will inspire and guide future research in this area. Questions are organized by category, and within each category are ordered by perceived difficulty level.

D.1 Winning Opening Moves and Strategies

- Solve all remaining 9×9 openings.
- Solve one or more 10×10 openings.
- Determine whether every first player win Hex position contains a cell that is a winning move for both players (posed by Ryan Hayward).
- Identify a winning opening move for all $n \times n$ Hex boards.
- Determine whether the centre is a winning opening move for all $n \times n$ boards.
- Determine whether all cells on the main diagonal are winning opening moves for all $n \times n$ Hex boards.

D.2 Graph Theory and Computational Complexity

- Identify graph classes where Generalized Hex is solvable via pairing strategies.
- Identify graph classes where Generalized Hex is solvable in polynomial-time. Paths, trees, and cycles are trivial, but investigate interval graphs, chordal graphs, bounded degree, bounded treewidth, etc.
- Determine mathematical properties and invariants of vertex implosion.
- Determine mathematical properties of graphs derived from planar graphs via vertex implosion.
- Determine the computational complexity of identifying live/dead cells on Hex graphs.

- Determine whether knowing the sequence of vertex implosions and vertex deletions from a given planar graph can reduce the computational complexity of problems on the resulting graph.
- Determine whether P equals PSPACE.
- Determine whether P equals NP.

D.3 Combinatorial Game Theory

- Find sufficient properties for reversible moves to be pruned, rather than bypassed, from combinatorial games.
- Find other combinatorial games where decomposition domination exists.
- Resolve van Rijswijck's open problems regarding the expression of Hex positions using combinatorial game theory and surreal numbers.

D.4 Hex Variants

- Develop results for the Hex variant Vex.
- Develop results for the Hex variant Tex.
- Determine the computational complexity of random-turn Hex.
- Find an explicit handicap strategy that requires fewer than $\lceil \frac{n+1}{6} \rceil$ handicap cells on the $n \times n$ Hex board.
- Find an existence proof for a handicap strategy that requires $o(n)$ handicap cells on the $n \times n$ Hex board.
- Solve Hex on the annulus when the ring dimension is odd.

D.5 Inferior Cell Analysis

- Find all dead cell patterns of radius at most two.
- Find all captured set patterns of radius at most two.
- Find all dead-reversible, capture-dominated, and captured-reversible patterns of radius at most two.
- Find an efficient algorithm that computes all possible deduced state values from a solved Hex state.

- Improve efficiency and generality of algorithms identifying chain combinatorial decompositions.
- Find an algorithm to recognize permanently inferior cell patterns, and find all such patterns of radius at most two.
- Determine whether captured-reversible cells can be unconditionally pruned.
- Develop an algorithm to automate domination deduction using existing base case domination techniques, and use this algorithm to produce more domination patterns.
- Resolve the 4-3-2 probe conjecture.

D.6 Connection Strategies

- Parallelize the H-search algorithm.
- Implement Anshelevich's generalized H-search with a bound on handicap set size, and test its performance.
- Determine if the partition chain algorithm can be generalized to compute larger sets of partition chains.
- Develop a deduction framework for incorporating union-connections into H-search.
- Generalize the common miai substrategy algorithm to allow for larger common substrategies.
- Determine how to efficiently recognize and store important learned connections during search.
- Determine whether there is a seventh row border template in Hex that requires no coloured cells.
- Determine whether there is a bound on border template distance in Hex when there are no coloured cells.

D.7 Solver

- Improve FDFPN's performance.
- Eliminate FDFPN's reliance on an (external) heuristic move ordering.
- Resolve the problem of PNS and its variants preferring moves that produce fillin when no mustplay exists.
- Parallelize PNS and its variants.
- Improve PNS and its variants for games with initially uniform branching factors.

D.8 Players

- Improve alpha-beta framework for Wolve, including parallelization and use of the killer or history heuristics.
- Build a strong 11×11 opening book.
- Further develop the PNS-based Hex player, attaining strong play on 11×11 boards.
- Beat top humans on 11×11.
- Find an evaluation function that outperforms the electric circuit model.
- Incorporate connection strategies into Monte Carlo simulations.
- Dynamically identify strategy decompositions in MCTS, and use this to improve its performance.
- Beat top humans on 14×14.