TWIXT: THEORY, ANALYSIS AND IMPLEMENTATION

Kevin Moesker

Master Thesis DKE 09-07

Thesis submitted in partial fulfilment
of the requirements for the degree of
Master of Science of Artificial Intelligence
at the Faculty of Humanities and Sciences
of Maastricht University

Thesis committee:

Dr.ir. J.W.H.M. Uiterwijk
Dr. M.H.M. Winands
M.P.D. Schadd, M.Sc.
Dr. F. Thuijsman

Maastricht University
Department of Knowledge Engineering
Maastricht, The Netherlands
March 2009

Preface

This master thesis is the result of the final research project of the master programme "Artificial Intelligence" of Maastricht University. Dr.ir. Jos Uiterwijk introduced me to the abstract board game TwixT during the course Intelligent Search Techniques (IST). IST aims at making students knowledgeable on techniques and methods used in computer game playing. It is far from obvious which techniques and methods perform well in computer TwixT. I felt challenged to do my master thesis about computer TwixT, and the result, my master thesis, lies before you.

I gratefully acknowledge everybody who supported me during the effort of writing my master thesis. I am most indebted to dr.ir. Jos Uiterwijk, for the possibility to broaden my skills and for supporting me throughout the thesis with his patience and knowledge. I thank Mark Winands, for providing me with articles on TwixT tactics. David Bush, who wrote the articles, was kind to elaborately answer my questions on TwixT. This gave me an exceptional opportunity to take a peek into the mind of an expert player, for which I am very grateful.

I thank my family and friends, and I cannot end without special thanks to Mike who has helped me many times during the master programme.

Sincere thanks to all of you.

Kevin Moesker
Maastricht, March 2009

Abstract

We investigated how to combine game-specific knowledge and known game-tree search techniques effectively and efficiently in computer TwixT. TwixT belongs to the family of connection games and is a two-player zero-sum board game with perfect information. Challenging characteristics of TwixT include that it has a large branching factor and that the process of estimating a board's utility value is known to be complex. Human players use strategic and tactical heuristics that work under specific conditions, but it is unclear how these heuristics can be effectively combined and implemented in a TwixT playing program. The state-space (10^{140}) and game-tree complexity (10^{159}) make TwixT belong to the highest category in terms of complexity. This means that TwixT is unlikely to be solved in the near future. If the complexity would have been low, then we could have sufficed with a pure search-based approach for computer TwixT. It is evident that an approach to computer TwixT must add game-specific knowledge.

Our approach is to implement combinations of game-specific knowledge and known game-tree search techniques in AI players. We implemented two types of AI players. The first AI player is an $\alpha\beta$ player. We use network search algorithms to extract features from network board-representations. Extracted features from the network include: shortest-path weight, maximum flow, board dominance, and game termination. Implemented $\alpha\beta$ enhancements include iterative deepening, history-heuristic move ordering, board-dominance move ordering, and the use of a transposition table. The second AI player is a basic Monte-Carlo player.

A TwixT simulation environment implements TwixT computer rules and supports automated play between two AI agents. The effectivity of an AI player is measured by the win statistics from 100 simulated games versus other players. The effectivity of an AI player is made explicit by looking at the decision-making process. We tested the gameplay performance of both AI players on 8×8 boards. Experimental results show that the $\alpha\beta$ player with history-heuristic move ordering and with a transposition table is most effective and most efficient.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis reports the research on how to write a computer program that plays TwixT as effective and efficient as possible.

Chapter contents: Introduction — Game AI and Computer TwixT, Problem Statement and Research Questions, Thesis Outline.

1.1 Game AI and Computer TwixT

Traditionally, it is considered to be a major milestone towards computer intelligence if a computer can outsmart a person in a intellectually challenging game. Claude Shannon's famous article on computer chess [40] laid the foundations for research on automated game playing. Building high-performance game-playing programs became a major goal of artificial intelligence research. Chess-playing programs now play at world-champion level, but they do so with limited intellectual mechanisms compared to those used by a human, substituting large amounts of computation for understanding. For some games the best move cannot be obtained by large amounts of computation alone, and different techniques have to be tried [8, 9, 25].

One of these games is TwixT, which is a two-player connection game that is invented around 1960 by Alex Randolph. TwixT challenges computer-game researchers, because the search space is large and the process of evaluating a board-position's utility value is known to be complex. Little knowledge is available on how to make a computer play a strong game of TwixT. Currently no program for TwixT exists that cannot be easily beaten by an experienced TwixT player. Johannes Schwagereit's TwixT program, called T1J [39], is considered to be the strongest AI player for TwixT in a competitive field of a few weaker TwixT programs. Schwagereit wrote a challenge for the 7th Computer Olympiad [29], where he challenged programmers to participate in a TwixT tournament at a subsequent Computer Olympiad.

1.2 Problem Statement and Research Questions

We want to contribute to knowledge acquisition in the domain of computer TwixT. Our problem statement is formulated as follows:

"How can a computer program be written that plays the game of TwixT as effective and efficient as possible?"

The problem statement is decomposed into four research questions.

1. *What game-specific knowledge, used by human players, is applicable to computer TwixT?*
2. *What can we learn from research that is related to TwixT?*
3. *What is the complexity of TwixT?*
4. *How can we combine game-specific knowledge and known game-tree search techniques effectively and efficiently?*

The first research question contributes to the acquisition of game-specific knowledge that needs to be represented in a TwixT playing program. Game-specific knowledge acquisition is a process that includes:

- the collection of game-specific knowledge that is used by human TwixT players,
- the assessment what game-specific knowledge is feasible to be modelled in a suitable format for computer TwixT.

The second research question investigates insights from related research. To our knowledge, there is no previous research on computer TwixT. We explore possible approaches in computer TwixT by investigating research on similar games.

The third research question analyses the complexity of TwixT. The state-space and game-tree complexity express the complexity of TwixT and define the position of TwixT in the game-space. We need the position of TwixT in the game-space as an important indicator on the solvability of TwixT and on the potential of search-based and knowledge-based approaches [28].

The last research question is concerned with the effectivity and efficiency of combinations of game-specific knowledge and known game-tree search techniques. Our approach is to implement combinations of game-specific knowledge and known game-tree search techniques in AI players. The insights that are gained from answering the previous research questions are used to target what game-specific knowledge and known game-tree search techniques are implemented. A TwixT playing engine, which facilitates automated game-play between two AI players, is used for testing purposes. The effectivity of an AI player is expressed by the win statistics of many simulated games versus other AI players. The efficiency of an AI player can be made explicit by looking at the decision-making process. The experimental results show which of the AI players, and thus what combination of techniques, is most effective and most efficient.

1.3 Thesis Outline

Chapter 2 provides an answer to the first research question: “What game-specific knowledge, used by human players, is applicable to computer TwixT?”. Section 2.1 describes the official rules, the pen-and-paper version rules, and a rule set for computer TwixT. We collect strategic and tactical knowledge that is used by expert TwixT players (Section 2.2). In the concluding section, we discuss what game-specific knowledge is necessary and look at what game-specific knowledge is feasible to be modelled in a suitable format for computer TwixT (Section 2.3).

Chapter 3 provides an answer to the second research question: “What can we learn from research that is related to TwixT?”. We show what insights are gained from research on the connection games Hex, Bridg-It and the Shannon Switching Game (Section 3.1). We show that the Voronoi Game (Section 3.2) and Go (Section 3.3) share some interesting properties with TwixT. Section 3.4 explains how the strategy-stealing argument is used in TwixT. Section 3.5 shows a proof draft on the ‘Uncrossed Knight Path problem’, which gives an indication of the difficulties involved in finding connective paths in TwixT. In the concluding section, we summarize what insights from related research are adopted and explored further in relation to computer TwixT (Section 3.6).

Chapter 4 provides an answer to the third research question: “What is the complexity of TwixT?”. The state-space and game-tree complexity of TwixT are calculated in Section 4.1 and Section 4.2. Both complexities determine the position of TwixT in the game-space and we compare the position of TwixT to the position of other games (Section 4.3). In the concluding section, we discuss the implications of the complexity of TwixT on approaches for computer TwixT (Section 4.4).

Chapter 5 focuses on how game-specific knowledge and known game-tree search techniques can be combined in computer TwixT. We give an overview of well known game-tree search techniques (Section 5.1) and show how game-specific knowledge can be added (Section 5.2). In the concluding section, we give an overview on the techniques that are implemented for testing purposes (Section 5.3).

Chapter 6 provides an in-depth explanation of how networks are used in computer TwixT. A short introduction to network theory clarifies the terminology and concepts used throughout the rest of the chapter (Section 6.1). We explain four types of networks that we use in computer TwixT (Section 6.2). Section 6.3 explains how network topology update rules change the state of the board’s network representations. Section 6.4 explains how network features are extracted from network board-representations. We summarize the contents of Chapter 6 in the concluding section (Section 6.5).

Chapter 7 investigates the effectivity and efficiency of different combinations of game-specific knowledge and known game-tree search techniques. We show the details of the experimental setup (Section 7.1), discuss the experimental results (Sections 7.2-7.3), and summarize the experimental results in the concluding section (Section 7.4).

Chapter 8 is the concluding chapter. We summarize the answers on our four research questions (Section 8.1) and revisit the problem statement (Section 8.2). Finally, we show the opportunities for future research (Section 8.3).

Chapter 2

The Game of TwixT

In this chapter, we provide an answer to the first research question: "What game-specific knowledge, used by human players, is applicable to computer TwixT?".

Chapter contents: The Game of TwixT — The Rules of TwixT, TwixT Strategy and Tactics, Chapter Conclusions.

2.1 The Rules of TwixT

This section presents three rule sets of TwixT: the first rule set defines how a normal game of TwixT is played, the second rule set defines how the pencil-and-paper version of TwixT (TwixT PP) is played, and the third rule set defines how computer TwixT is played.

2.1.1 Normal TwixT Rules

We show the official rules as printed in an article by David Bush: *An Introduction to TwixT* [13].[1]

> The board is a 24×24 square grid of holes, minus the corner holes. For this article, one player will be referred to as 'White', and the other as 'Black.'(Many Twixt sets use different color schemes; in the USA, for example, most sets use red versus black.) The holes along the four edges are referred to as 'border rows'. The 'top' and 'bottom' rows are White's border rows, and the 'left' and 'right' border rows are Black's. These border rows are delineated from the rest of the board by borderlines, as shown in [Figure 2.1].
> Each player has a collection of pegs and links of his color. Approximately 50 pegs and 50 links for each side, a total of 200 pieces, is an ample supply. White moves first, then play alternates. Each move consists of the following steps:

[1]David Bush published two other articles on TwixT: *TwixT Strategy and Tactics Part 1* [14] and *TwixT Strategy and Tactics Part 2* [15].

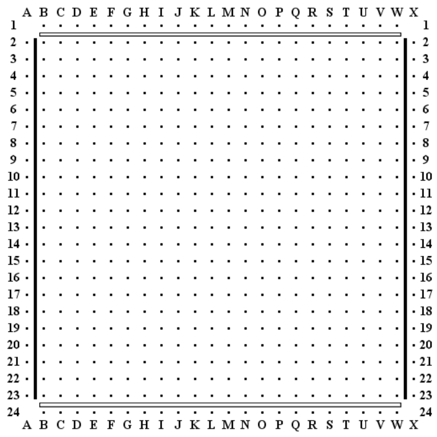


Figure 2.1: The initial 24 × 24 TwixT board position.

1. Place a peg of your color in any vacant hole except a hole in your opponent's border rows.
2. Place as many legal links as you wish between pairs of pegs of your color. You may place a link only between pegs which are at opposite corners of a 2 × 3 rectangle[2], like a knight's move in Chess. No link may ever cross another link, even one of the same color. You are allowed to remove as many of your own links as you wish prior to placing any links. If you do not have two pegs on the board a knight's move apart, you may not place any links on that move.

[...]

After White makes the first move, Black has the option of either responding normally, or swapping sides. If sides are swapped, the player who moved first as White is now Black and makes the next move. This rule[3] makes the game more balanced, as otherwise White would have a strong first-move advantage. The objective is to connect your border rows with a continuous chain of linked pegs. If neither side can complete such a chain, the game is a draw.

Figure 2.2 shows a board position that is won by White and a board position that is a draw.[4]

2.1.2 TwixT PP Rules

TwixT started in 1958 as a pencil-and-paper game (TwixT PP). The rules of TwixT PP are similar to the normal rules of TwixT. However, TwixT PP disallows link removal and allows crossing own links.

[2]This should be a 2 × 3 array of pegs, which corresponds to a 1 × 2 rectangle.

[3]This rule is called the pie rule.

[4]Some people say that the board position on the right side is not officially a draw, because a player can remove own links and give the opponent the opportunity to win. We will not consider such arguments.

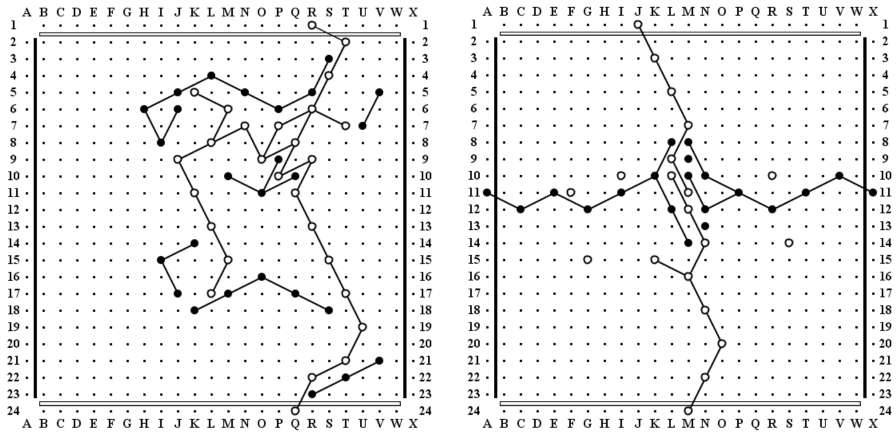


Figure 2.2: A TwixT position won by White (left), and a drawn TwixT position (right).

2.1.3 Computer TwixT Rules

There is no consensus on the rules for computer TwixT. Our rule set for computer TwixT follows the TwixT PP rules and adopts an auto-linking rule as proposed by Johannes Schwagereit [39]. The auto-linking rule imposes that a move only consists of a peg placement and that all links that can be added to a placed peg are automatically added. We exclude the pie rule from our rule set.

2.2 TwixT Strategy and Tactics

The exponential explosion of the number of possible board positions after each continuation of the game makes it difficult to have a deep understanding of the implications of a move. We explore what strategic and tactical knowledge is used be expert players.

2.2.1 Setups

Setups are the most common tactical peg patterns in TwixT. David Bush describes setups as follows [12]:

> A setup is a pattern of two pegs of the same color which can connect to each other in a single move in two different ways. The gap between these pegs is generally difficult for the opponent to attack, since if one connection is blocked then the other is usually still available. There are five setups, each characterized by a name and by two numbers which represent the horizontal and vertical distances between these pegs. The larger value is listed first. [Figure 2.3 shows the five basic setups. The '×' symbols indicate where a third peg of the same colour would form a double-link connection.]
>
> [...]

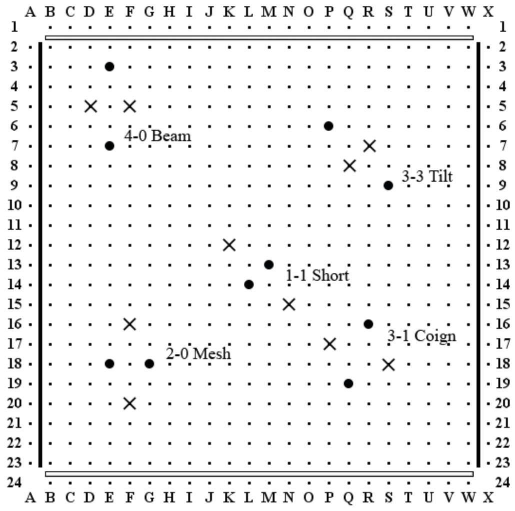


Figure 2.3: The five basic setups in TwixT.

There are plenty of other ways to place two pegs of the same color so that the gap between them is difficult to attack. The next diagram shows a few of them [see Figure 2.4].

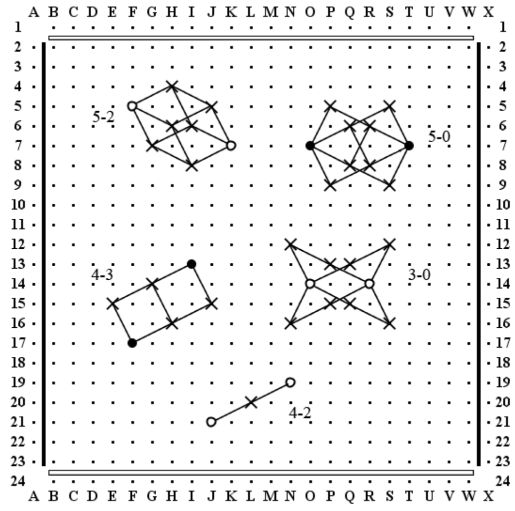


Figure 2.4: Several advanced setups in TwixT.

The 5-2 gap is particularly strong. These two pegs can connect in two moves in a variety of ways, usually too many for the opponent to block them all. The 5-0 gap is slightly more vulnerable. For example, if the O7 peg is unlinked, White might be able to attack at Q7. Then if Black plays R6 threatening P5 or Q8, White could play O8, threatening N6 or R5. The 3-0 gap involves some very tricky

> tactics. For example, if Black tries to attack with an unlinked peg at O15, White could respond with N16 which threatens to double link at P15. The 4-2 gap is technically not a setup, because there is only one way to connect these pegs in one move. But without a nearby peg, it may be difficult for Black to attack this pattern anyway. If Black plays L20, White could respond at K20 or at M20, forming a combination of a coign setup and a short setup. Since the short setup is so difficult to attack, White will probably manage to connect J21 to N19, albeit in three moves rather than one. White might also respond to L20 with either K21 or M19, which is much more complicated.

Knowledge of setup patterns enables players to play pegs that are likely to connect.

2.2.2 Connective Strength

The connective *strength* between two pegs is defined by the opportunity to connect the two pegs. In the smallest setting we discuss the connective strength of setups, where one move can connect two pegs. In a broader setting we discuss the connective strength between pegs in general, where an arbitrary number of moves is required to connect two pegs.

Connective Strength of a Setup

The following aspects are involved when analysing the strength of a setup:

- the width of the connection pattern,
- the support of own pegs and links,
- the threats of opponent pegs and links.

The first aspect considers the width of the *connection pattern*. A connection pattern consists of the set of holes and links that can be used to connect the pegs with one move. The ‘Coign’ setup has the widest setup connection pattern and is the strongest basic setup (see Figure 2.3). Setups with a wide connection pattern are harder to attack than setups with a skinny connection pattern.

The second aspect considers the support of own pegs and links. The interaction with other pegs is important, because it is of no use to connect a setup if the setup itself is blocked from reaching the border. Strong support from surrounding pegs reduces the opportunity of the opponent to block a setup. Setups can be unanchored, single anchored, or double anchored [20] as shown in Figure 2.5. Unanchored setups are much easier to break, single anchored setups are harder to break, while setups anchored at both ends are (usually) nearly impossible to break.

The third aspect considers threats of opponent pegs and links. Nearby opponent pegs and links strongly threat to cut off all possible linked paths between the two setup pegs.

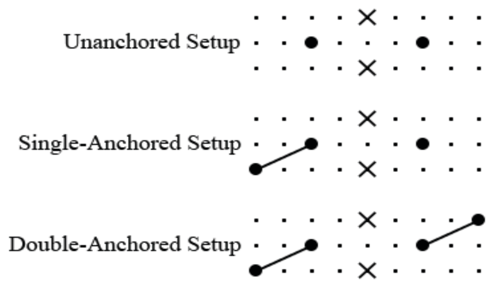


Figure 2.5: Unanchored, single anchored and double anchored setups.

Connective Strength between Pegs

There are many peg patterns where pegs are at an arbitrary distance. The first aspect that is used to analyse the strength of setups, the connection pattern width, has to be put in a broader perspective, because usually more than one move is required to connect two pegs. We use the term *manoeuvre space* for the total space on the board that can contain a formation of links and pegs to connect two pegs. In a strictly defined setting there would be an upper bound on the number of links in a carrier, but in the broadest setting the whole board can be used to connect two pegs. A skinny manoeuvre space between two pegs requires fewer moves of the opponent to drastically reduce the connective opportunity. Wide manoeuvre spaces prevent that the opponent can drastically reduce the connective opportunity in just a few moves. The shape of the manoeuvre space is an important indicator of strength. Not only the width, but also the distance between the pegs should be taken into account. Pegs that are nearby are usually connected stronger than pegs that are distant. Borders limit the manoeuvre space and sometimes cause peg patterns to fail in getting connected at the border while the same peg pattern would have worked in the centre of the board.

2.2.3 Strategic Balance

Experienced players play moves that offer a good balance between posing threats and blocking threats. A *block* is anything of your colour between an opponent peg and one of his sides. A block from your perspective is a *threat* for your opponent's perspective and vice versa.

Alan Hensel coined the term *potential paths* [27], which are connections to be built in a later stage of the game. Experienced TwixT players try to create multiple threats by creating multiple potential paths. Figure 2.6 shows a board position where White's potential paths are in a Y formation. White threatens to connect to the top in two ways.

A move is efficient if it serves multiple purposes. Schmittberger describes the importance of *efficiency* in connection games [41], and he remarks that keeping pieces spread out rather than bunched up can help to improve efficiency. Pieces that are close together tend to have a similar purpose and are not likely to be efficient. This also applies to TwixT. Experienced players consider the whole board when they try to determine the best move, and they try to avoid close battles in the early phase of the game.

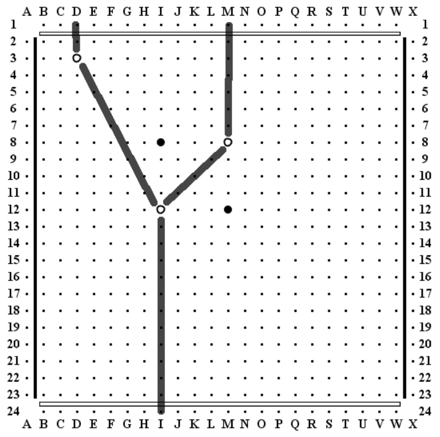


Figure 2.6: A TwixT board position with Y-structured potential paths for White.

2.2.4 Ladders

A *ladder* is a structure of pieces that is created by a forced sequence of moves towards a border. Not cutting off the opponent towards the border will cause a breakthrough in the defence. Therefore, the ladder continues until it hits a border or until an escape move is found. Ladders are intuitive for many people, because ladders also exist in Go (see Section 3.3).

Cardinal lines are guidelines that are used to predict who is going to win a ladder fight. Cardinal lines are straight lines on the board that start in one corner and continue towards the opposite side of the board along a series of holes that could be connected with links (see Figure 2.7).

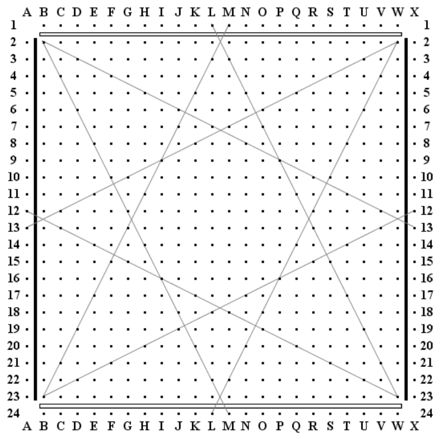


Figure 2.7: A TwixT board with cardinal lines.

Blocking on or in front of a cardinal line is favoured in most cases.[5] Figure 2.8 shows, on the left side, a ladder where Black blocks behind the B2-E8 cardinal

[5]Escape moves may exist to pose an exception.

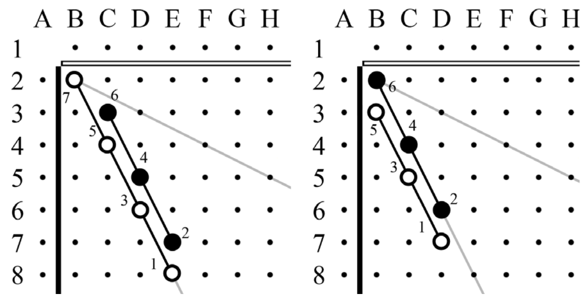


Figure 2.8: Black blocking behind the cardinal line (left), and Black blocking on the cardinal line (right).

line. Black loses the ladder fight, because White can connect with the top border. Figure 2.8 shows, on the right side, a ladder where Black blocks on the B2-E8 cardinal line. Black wins the ladder fight, because Black can connect with the left border.

A ladder is called 'solid' if the pegs in the ladder pattern are connected by links (see Figure 2.8). A ladder is called 'dotted' if the pegs in the ladder are not connected by links (see Figure 2.9). Dotted ladders are usually easier to break than solid ladders.

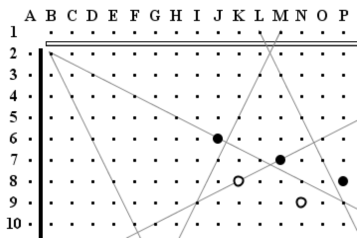


Figure 2.9: A dotted ladder.

2.3 Chapter Conclusions

This chapter contributed to the acquisition of game-specific knowledge that needs to be represented in a TwixT playing program. We described the official rules, the pen and paper version rules, and a rule set for computer TwixT. Our computer TwixT rule set adopts the TwixT PP rules with an added auto-linking rule. We examined strategic and tactical TwixT knowledge that is used by expert players. Human players estimate a board-position's utility value based on a complex interaction between many interrelated features that cannot be easily extracted by a computer. The strategic and tactical heuristics work under specific conditions, but it is unclear how these heuristics can be effectively combined and implemented in a TwixT playing program. Accurate predictive evaluation of a board-position's utility value requires a deep look-ahead capability. The look-ahead capability is limited in TwixT, because of the many possible continuations of a game for each board position.

Chapter 3

Related Research

In this chapter, we provide an answer to the second research question: "What can we learn from research that is related to TwixT?".

Chapter contents: Related Research — Connection Games, The Voronoi Game, Go, TwixT and the Strategy-Stealing Argument, Uncrossed Knight Path, Chapter Conclusions.

3.1 Connection Games

TwixT belongs to a family of games called *connection games*, where players have to build a specific type of connection with their pieces. Cameron Browne did a comprehensive study on connection games and wrote the book *Connection Games – Variations on a Theme*, studying over 200 key games and variants while exploring common themes, strategies and underlying mechanisms [10]. We discuss the connection games Hex, Bridg-It, and the Shannon Switching Game.

3.1.1 Hex

Hex is invented in 1942 by Danish mathematician and poet Piet Hein and is independently reinvented by John Nash in 1948. The game was presented to the general public in Martin Gardner's article in *Scientific American* [22]. Vadim Anshelevich, who built a strong Hex playing program called HEXY [3], describes the rules of Hex as follows [5]:

> Hex is a two-player game played on a rhombic board with hexagonal cells. The classic board is 11 × 11, but it can be any size. The 10×10, 14×14 and even 19×19 board sizes are also popular. The players, Red and Blue, or as some prefer, Black and White, take turns placing pieces of their color on non occupied cells of the board. Red's objective is to connect the two opposite red sides of the board with a chain of red pieces. Blue's objective is to connect the two opposite blue sides of the board with a chain of blue pieces. Red (Black) moves first. Hex can never end in a draw. If all cells of

the board are occupied, then a winning chain for Red or Blue must necessarily exist.

Hex is won by a player when the player has achieved an uninterrupted connection between the corresponding opposing borders. Figure 3.1 shows a Hex position that is won by White.

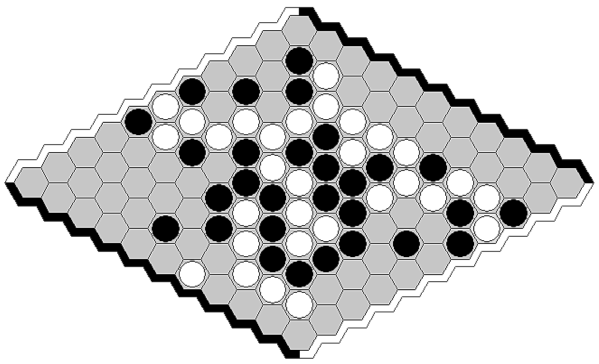


Figure 3.1: A Hex position won by White.

The game mechanics for building connections are different for Hex and TwixT. Hex is played on a hexagonal board, whereas TwixT is played on a square grid. For Hex, two neighbouring cells with a shared border occupied by the same colour are connected, whereas for TwixT, two cells at a knight's move distance occupied by the same colour are connected.

Hex has nice mathematical properties. Anatole Beck proved that Hex cannot end in a draw [6]. John Nash proved that the first player has a winning strategy when no swap is allowed, by using a strategy-stealing argument. His proof is non-constructive, which means that Hex is only *ultra weakly solved* [2]. TwixT can end in a draw, and it is unknown if a player has a winning strategy. Section 3.4 shows how the strategy-stealing argument can be used to prove that for TwixT there is no winning strategy for the second player when no swapping is allowed. In Hex and TwixT only one player can win, because the winning formation of linked pieces cannot be crossed by the opponent.

Highly ranked Hex programs use a theorem-proving approach as published by Vadim Anshelevich [4, 5]. Anshelevich introduces logical-deduction rules to decompose a certain class of subgames into sums of simpler subgames. Subgames are defined by a starting cell, an ending cell, and the carrier of the connection, which is a set of cells between. A certain class of subgames can be decomposed as virtual connections and virtual semi-connections. Virtual connections are strong connections for which a winning strategy is possible even if the opponent moves first. Virtual semi-connections are weaker connections for which a winning strategy is only possible when you have the first move. The logical-deduction rules are based on the rules of the game and use the hierarchical structure of virtual connections and virtual semi-connections within a subgame to predict who is winning the subgame.

Vadim Anshelevich also explains how to incorporate subgame win predictions into the evaluation function, but this is irrelevant for our purposes.

Anshelevich's evaluation of a board position is based on an *electrical-resistor-circuit* model, and he describes the basic idea as follows [5]:

> In this section we introduce a family of evaluation functions based on an electrical resistor circuit representation of Hex positions. [. . .] One can think of an electrical circuit as a graph. Edges of the graph play the role of electrical links (resistors). The resistance of each electrical link is equal to the length of the corresponding edge of the graph. Here, we see that the 'electrical circuit' language better suits our needs. With every Hex position, we associate two electrical circuits. The first one characterizes the position from Black's perspective (Black's circuit), and the second one characterizes the position from White's perspective (White's circuit). To every cell c of the board we assign a resistance r in the following way. For Black's circuit:
>
> $$r_B(c) = \begin{cases} 1 & \text{if } c \text{ is empty,} \\ 0 & \text{if } c \text{ is occupied by a black piece,} \\ +\infty & \text{if } c \text{ is occupied by a white piece.} \end{cases}$$
>
> For White's circuit:
>
> $$r_W(c) = \begin{cases} 1 & \text{if } c \text{ is empty,} \\ 0 & \text{if } c \text{ is occupied by a white piece,} \\ +\infty & \text{if } c \text{ is occupied by a black piece.} \end{cases}$$
>
> For each pair of neighbouring cells, $(c1,\ c2)$, we associate an electrical link with resistance:
>
> $$r_B(c1, c2) = r_B(c1) + r_B(c2), \text{ for Black's circuit,}$$
>
> $$r_W(c1, c2) = r_W(c1) + r_W(c2), \text{ for White's circuit.}$$
>
> Let R_B and R_W be distances between black boundaries in Black's circuit and between white boundaries in White's circuit, correspondingly. Now we define an evaluation function E as
>
> $$E = \log(R_B/R_W).$$
>
> A reasonable distance metric is the length of the shortest path on the graph connecting boundaries; however, distances can be measured in different ways. Following Shannon's idea, we applied an electrical voltage to the opposite boundaries of the board and measured the total resistance between them, R_B for Black's circuit and R_W for White's circuit [see Fig. 3.2].

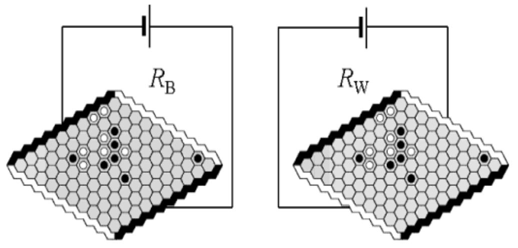


Figure 3.2: Black's and White's circuits.

> This idea was implemented by C.E. Shannon in a robot which played the game Bird Cage, also known as the game of Gale or Bridg-it [see [23]]. We prefer this method for measuring distances because according to the Kirchhoff electrical current laws, the total resistance takes into account not only the length of the shortest path, but also all other paths connecting the boundaries, their lengths, and their intersections.

Anshelevich's evaluation function takes into account how much closer Black is to building a winning black chain than White is to building a winning white chain. Player distances can be measured in different ways. Anshelevich's Hex program HEXY uses the total resistance in a electrical-resistor-circuit board representation to measure player distances. Jack van Rijswijk's Hex program QUEENBEE uses the two-distance in a network board-representation. The two-distance is the distance of the second shortest path between the borders.

Anshelevich's idea of having a hierarchical representation of subgames is less applicable in computer TwixT. Anshelevich's logical-deduction rules for Hex use the fact that a carrier-path's disconnection can only be the result of the placement of a stone on the carrier path. Hex is played on vertices, but TwixT is played on vertices and edges, and because of that, TwixT offers a greater diversity of actions that lead to a carrier-path's disconnection. Peg placement on a carrier path leads to disconnection, but peg placement outside a carrier path can also lead to disconnection when it is followed up by the placement of a connecting link that crosses and thus disconnects the carrier path. Anshelevich's logical-deduction rules for Hex have to be extended to be applicable in TwixT.

3.1.2 Bridg-It

Bridg-It (also known as Gale and Bird-Cage) is a connection game that is invented around 1960 by mathematician and economist David Gale. Bridg-It was first known as 'Gale', because of Martin Gardner's *Mathematical Games* article in Scientific American [23], where the game was called after its inventor. The game was marketed under the trade name of Bridg-It in the early 1960s. Claude Shannon built the first automated Bridg-It player in 1951, seven years before Gardner published about 'Gale', but Shannon called the game 'Bird-Cage' [23].

A Bridg-It board contains two overlapping grids of white and black dots (see Figure 3.3). All dots with a white colour belong to the first player, 'White', and all black dots belong to the second player, 'Black'. Figure 3.3 shows that the

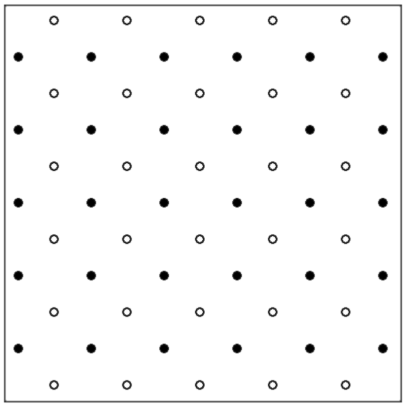


Figure 3.3: An initial Bridg-It board position.

'upper' and 'lower' border rows can only be played by White, and that the 'left' and 'right' border rows can only be played by Black. Players take alternate turns and place a horizontal or vertical bridge between dots of corresponding colour on the board. The rules disallow placement of a bridge that crosses an already placed bridge. The first player to succeed in building an uninterrupted path between the corresponding borders wins the game. Figure 3.4 shows a Bridg-It board position that is won by White.

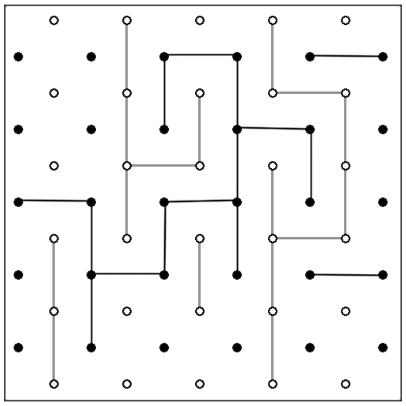


Figure 3.4: A Bridg-It position won by White.

Oliver Gross first discovered an explicit winning strategy (a pairing strategy) in the early 1960s [24]. Martin Gardner described the strategy of Claude Shannon's Bridg-It playing robot [23].

> A resistor network corresponds to the lines of play open to one of the players, say player A [see Fig. 3.5]. All resistors are of the same value. When A draws a line, the resistor corresponding to that line is short circuited. When B draws a line, the resistor, corresponding to A's line that is intersected by B's move, is open circuited. The entire network is thus shorted (i.e., resistance drops to zero) when A wins the game, and the current is cut off completely (i.e., resistance

becomes infinite) when B wins. The machine's strategy consists of shorting or opening the resistor across which the maximum voltage occurs. If two or more resistors show the same maximum voltage, one is picked at random.

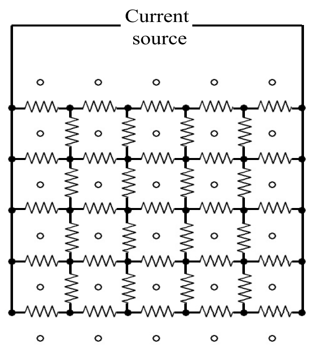


Figure 3.5: Resistor network for the robot Bridg-It player.

3.1.3 The Shannon Switching Game

The Shannon Switching Game is played on a graph with distinguished source and target vertices s and t. A graph with these properties is called a *network*, and an introduction to networks can be found in Chapter 6.

The edges of the network are initially all 'unmarked'. The first player, called *Short*, has the objective of creating a marked path between source and target. The second player, called *Cut*, has the objective of preventing a marked path between source and target. The players take alternate turns. A move for *Short* consists of marking one edge. Marked edges are permanent and cannot be removed. A move for *Cut* consists of removing one edge. A removed edge can no longer be marked. *Short* wins the game if a marked path is created from source to target, and *Cut* wins the game if it is impossible for *Short* to create such a path. Draws cannot occur in a Shannon Switching Game.

Alfred Lehman proved that the Shannon Switching Game has a winning strategy for *Short* if the network contains two non-overlapping spanning trees that both connect s and t [32]. A *Cut* move separates one of the spanning trees into two parts. *Short* will win if he makes response moves that mark the edge in the unseparated spanning tree that reconnects the two parts of the separated spanning tree.

Bridg-It is a special case of a *Shannon Edge Switching Game*, where the game is played by marking and removing edges. Figure 3.6 shows the initial Bridg-It board position with its overlaid network board-representation. White's source vertex is connected with the upper row vertices and the target vertex is connected to the lower row vertices. Source or target connecting edges are marked, which is indicated by black coloured edges in the figure. The grey coloured edges represent possible bridge locations for White. A move of *Short* can be seen as the placement of a white bridge between white dots. A move of *Cut* can be seen as the placement of a black bridge, which disables a white bridge placement.

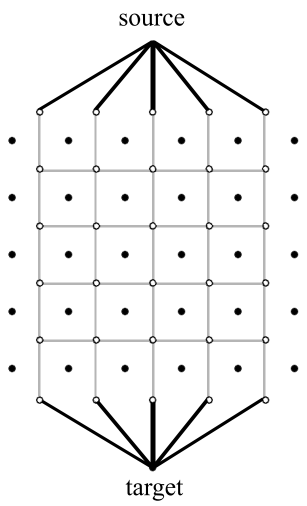


Figure 3.6: A graph representation of an initial Bridg-It board position.

Hex is a special case of a *Shannon Vertex Switching Game*, where the game is played by marking and removing vertices. We refer to an article of Jack van Rijswijk [36] for more information on how Hex can be played as a Shannon Vertex Switching Game. Both players have a network representation that suits their perspective of cell connectedness on the board. Placement of a stone on a cell will change the topology of both networks.

TwixT is neither a Shannon Edge Switching Game nor a Shannon Edge Switching Game, because moves are placed on the vertices (peg placement) and on the edges (link placement) of a network representation. Section 6.2 explains how network board-representations capture the game state, and Section 6.3 explains how network-topology update rules capture the rules of the game.

3.2 The Voronoi Game

The Voronoi Game is a two-player game that is used as a model for competitive facility location [1, 17]. Players are occupied with the following question: "Given a market space, how can I place my facilities such that I have more customers than my competitor?". The Voronoi Game models the total available market space as a continuous 2-dimensional rectangle. Two players alternately place a vertex in the market space to represent a player's facility location. The *market area* of a facility marks the space on the board in which the customers of the facility reside. Customers always buy goods at the facility that is nearest to their location. Thus, the market area of a facility corresponds with the facility's Voronoi region. The Voronoi region of a facility f is the set of points in the space for which f is the closest facility among all the facilities. The player with the biggest market area on the board after n moves wins the game. Figure 3.7 shows a Voronoi-Game board position that is won by Black.

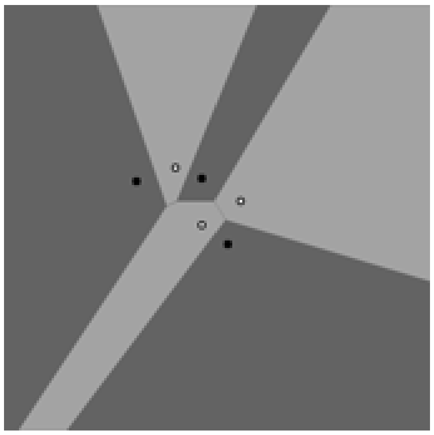


Figure 3.7: A Voronoi Game position won by Black.

The Voronoi Game would be closely related to TwixT if players would have been occupied with the following question: "Given a rectangular market space, how can I place my facilities such that my market area's connect from top to bottom or from side to side?".

For TwixT, a Voronoi region corresponds to the space that is directly influenced by a peg. Imagine a large TwixT board position (let's say a million by a million holes) and imagine that a prediction has to be made on who is winning. Figure 3.8 shows a large TwixT board position without and with a Voronoi tessellation of the space. Voronoi-region connectedness shows information on which gaps are dominant. On a large TwixT board, when two Voronoi regions of the same colour share a boundary it is more likely that the gap between the corresponding pegs will be connected than the alternative of the gap getting crossed by the opponent. The gap between those two pegs is dominant, because the distance between those pegs is smaller than the distance of any crossing opponent gaps. A player is dominant on the board if the player can travel between the corresponding opposing borders via neighbouring Voronoi regions of his colour. Only one player can be dominant on the board, because a dominant path by definition cannot be crossed by the opponent. A closer examination of the Voronoi tessellation in Figure 3.8 shows that Black can travel between the left and right border rows via black neighbouring Voronoi regions. Black is dominant on the board and is more likely to win than White.

Deducing board dominance based on Voronoi-region connectedness has drawbacks when we check for board dominance on TwixT boards with nearby pegs. We have to be aware that the set of vacant holes in a Voronoi region, the modelled influence area, is only an approximation of the true influence area of a peg. The true influence area is modelled with the knight's move distance metric, and the approximated influence area is modelled with the Euclidian distance metric. On large boards with large peg distances, there will be no significant differences between the shapes of the true influence area and the approximated influence area. On small boards with small peg distances, there will be bigger differences between the shape of the true and the approximated influence area. The assumption that nearby pegs are more likely to connect than distant pegs will also not always hold, in particular when distances between pegs get small.

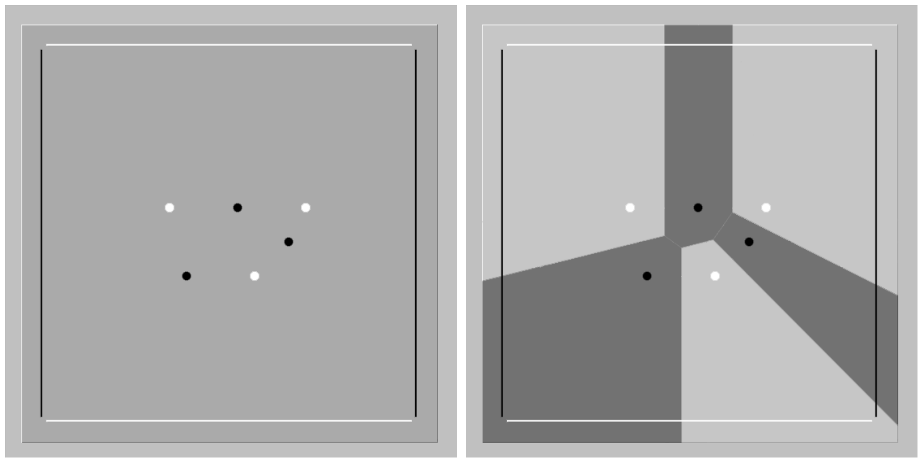


Figure 3.8: A board position on a large TwixT board (left), and a board position on a large TwixT board overlaid with the Voronoi tessellation (right).

3.3 Go

We mention Go as similar game to TwixT, because both games have a large branching factor (see Section 4.2) and because the process of estimating a board's utility value is known to be complex for both games. Bouzy and Helmstetter showed that using Monte Carlo simulations as an evaluation function works surprisingly well in computer Go [9], which raises the question how Monte-Carlo simulations would perform in TwixT (see Section 5.2.3). Many enhancements to basic Monte Carlo have been proposed and successfully applied [16, 19, 31], but we do not go into detail on enhancements and stick to the basics.

Another similar game property is the existence of ladder fights. A forced sequence of moves leads to a ladder pattern that continues until there is an escape move or a border is hit. Figure 3.9 shows an example of a ladder in Go. The rectangular symbol marks a forced response move for Black.

Knowledge on ladder development might narrow down the considered lines of play during board analysis. For TwixT, it is hard to narrow down the number of possible response moves. Certainly early in a game, many escape moves exist; therefore, we do not focus on ladders.

3.4 TwixT and the Strategy-Stealing Argument

We give Nash's sketch of the proof, which shows that the first player has a winning strategy for Hex when no swap is allowed. We show the sketch proof as printed in Martin Gardner's article about Hex [21].[1]

(i) Either the first or second player must win, therefore there must be a winning strategy for either the first or second player.

[1]The third argument is incomplete and omits a statement that says: whenever the first player, according to the winning strategy, has to place a piece on a cell which is already occupied, then the first player must play an arbitrary other (legal) move.

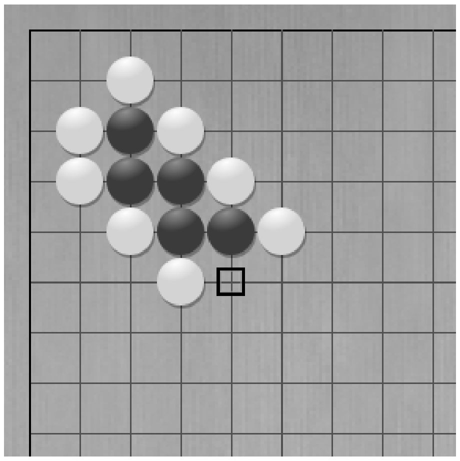


Figure 3.9: A Go ladder example.

(ii) Let us assume that the second player has a winning strategy.

(iii) The first player can now adopt the following defense. He first makes an arbitrary move. Thereafter he plays the winning second-player strategy assumed above. In short, he becomes the second player, but with an extra piece placed somewhere on the board. [This argument is usually referred to as the 'strategy-stealing' argument.]

(iv) This extra piece cannot interfere with the first player's imitation of the winning strategy, for an extra piece is always an asset and never a handicap. Therefore the first player can win.

(v) Since we have now contradicted our assumption that there is a winning strategy for the second player, we are forced to drop this assumption.

(vi) Consequently there must be a winning strategy for the first player.

We examine how the strategy-stealing argument can be used in TwixT. A close examination on Nash's proof shows that the strategy-stealing argument itself (iii) leads to the conclusion that the second player has no winning strategy. For TwixT, because the existence of a winning strategy for the second player can be assumed and because the winning strategy can be stolen similarly in TwixT by the first player, we can conclude by contradiction that the second player has no winning strategy when no swapping is allowed. The proof of a winning strategy for the first player requires a proof that TwixT is always won by either the first or the second player. TwixT does not have this property, because TwixT can be either a win for the first player, a win for the second player, or a draw. So, unlike Hex, nothing can be said about the existence of a winning strategy for the first player.

When swapping is allowed, the second player can prevent the first player from winning when he adopts the following strategy:

- Swap if the first player's first move leads to a win,
- Do not swap if the first player's first move does not lead to a win.

3.5 Uncrossed Knight Path Problem

Dominic Mazzoni and Kevin Watkins created a TwixT Proof draft [33], which shows that finding an uncrossed path between two pegs in two-colour TwixT is NP-complete. The proof of the NP-completeness of finding an uncrossed path between two pegs in two-colour TwixT is implied from a proof of the NP-completeness of single-colour TwixT in the following way:

> The single-color uncrossed knight path problem is as follows. Given a set of pegs belonging to a single color, and identifying two pegs as s and t, we would like to know if there exists an uncrossed path between them. Note that in the game of TwixT, there is the added complication of the pegs and bridges belonging to the other player, which limits the possible locations for bridges to be placed. Let us call the two-color uncrossed path problem the question of whether s and t are connected, using only pegs and bridges of the same color as s and t, in the presence of pegs and bridges of a different color. Note that if an algorithm was found for the two-color uncrossed path problem, it would immediately imply an algorithm for the single-color variation, though the reverse is not true. Therefore it will suffice to show that the single-color uncrossed path problem is NP-complete, and this implies the NP-completeness of the two-color problem as well.

The proof is non-constructive and shows the existence of an NP-complete algorithm without giving pointers on how an algorithm is constructed. The proof unfortunately only gives an indication of the difficulties involved in finding connection paths in TwixT, because crossing own links is allowed with our computer TwixT rule set.

3.6 Chapter Conclusions

This chapter investigated research in similar games. Research on the connection games Hex, Bridg-It and the Shannon Switching Game showed that network board-representations can evaluate the game state. Section 6.2 explains the network representations that we use in computer TwixT. We have also seen how network topology update rules can model the underlying game mechanics. Section 6.3 explains how network topology update rules capture TwixT linking rules. Anshelevich's Hex evaluation function considers how much closer a player is to building a winning chain than the opponent is to building a winning chain. Anshelevich's Hex program HEXY measures player distances by the total resistance in a player's electrical-resistor-circuit board-representation. Jack van Rijswijk's Hex playing program QUEENBEE uses the 'two-distance' to measure player distances in a player's network board-representation. Our TwixT evaluation function is inspired by their work (see Section 5.2.2).

Chapter 4

Complexity Analysis of TwixT

In this chapter, we analyse the state-space and game-tree complexity of TwixT. We compare the complexity of TwixT to other games and discuss the implications for an approach to computer TwixT.

4.1 State-Space Complexity

Victor Allis defines the state-space complexity as [2]: "the number of legal game positions reachable from the initial position of the game".

We calculated the number of possible peg configurations on the board[1], which is a lower bound on the state-space complexity of TwixT. Our calculation excludes link placement, treats mirrored board positions and rotated board positions as different positions, and includes the assumption that each player has no more than 50 pegs. Three areas exist on a TwixT board. The white border rows can only be occupied by White, the black border rows can only be occupied by Black, and the centre area can be played by both players. Each area has the corresponding capacity of 44, 44, and 484 holes. The naming convention for those areas are $Border_W$, $Border_B$, and *Common* as indicated in Figure 4.1.

Algorithm 4.1 shows the pseudo-code of the state-space complexity calculation and we explain the code below.

[1]Appendix A shows the matlab code

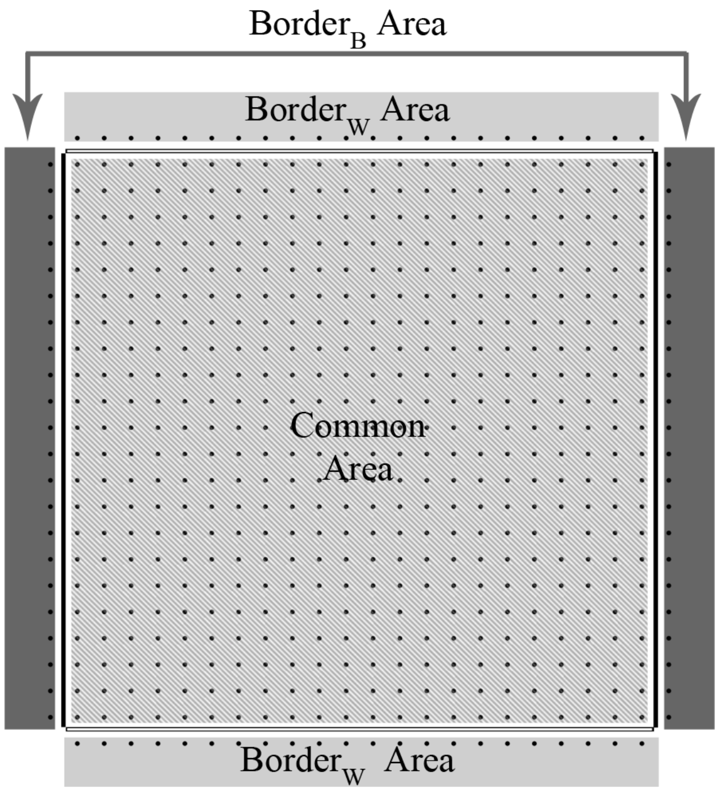


Figure 4.1: Three peg areas on a TwixT board.

Algorithm 4.1: TwixTStateSpace()

```
$stateSpace \leftarrow 0$
for $nrPegs \leftarrow 0$ to 100
do
    $nrPegs_W \leftarrow$ getNrWhitePieces($nrPegs$)
    $nrPegs_B \leftarrow$ getNrBlackPieces($nrPegs$)
    for $nrPegsBorder_W \leftarrow 0$ to $max(44, nrPegs_W)$
    do
        $nrCombiBorder_W \leftarrow \binom{44}{nrPegsBorder_W}$
        for $nrPegsBorder_B \leftarrow 0$ to $max(44, nrPegsB)$
        do
            $nrCombiBorder_B \leftarrow \binom{44}{nrPegsBorder_B}$
            $nrPegsCommon_W \leftarrow nrPegs_W - nrPegsBorder_W$
            $nrPegsCommon_B \leftarrow nrPegs_B - nrPegsBorder_B$
            $nrCombiCommon_W \leftarrow \binom{484}{nrPegsCommon_W}$
            $nrCombiCommon_B \leftarrow \binom{484-nrPegsCommon_W}{nrPegsCommon_B}$
            $nrCombiCommon \leftarrow nrCombiCommon_W \times nrCombiCommon_B$
            $totalPegCombi \leftarrow nrCombiCommon \times nrCombiBorder_W \times nrCombiBorder_B$
            $stateSpace \leftarrow stateSpace + totalPegCombi$
return ($stateSpace$)
```

The algorithm iteratively adds up the number of possible board positions for each total number of pegs. The number of total pegs ranges from 0 to 100 pegs.

At first, the algorithm determines the number of white pegs ($nrPegs_W$) and black pegs ($nrPegs_B$) for each total number of pegs, which is uniquely defined, because there is no piece capturing in TwixT.

Once the total number of white and black pegs is known, the algorithm iteratively considers all possible distributions of the white and black pegs over the three areas. The number of white pegs in the $Border_W$ area ($nrPegsBorder_W$) ranges from 0 to $\max(nrPegs_W, 44)$. This assures that there cannot be more white pegs in the $Border_W$ area than there are white pegs in total and that there cannot be more white pegs in the $Border_W$ area than the available capacity of 44. The number of black pegs in the $Border_B$ area ($nrPegsBorder_B$) ranges from 0 to $\max(nrPegs_B, 44)$. The number of white pegs in the *Common* area equals the difference between the total number of white pegs and the number of white pegs that are placed in the $Border_W$ area, and the number of black pegs in the *Common* area is calculated analoguely.

$$nrPegsCommon_W = nrPegs_W - nrPegsBorder_W \tag{4.1}$$

$$nrPegsCommon_B = nrPegs_B - nrPegsBorder_B \tag{4.2}$$

Once the number of white and black pegs is known for each area, then the number of possible configurations within each area is calculated. The number of possible peg configurations in the $Border_W$ area is computed by

$$nrCombiBorder_W = \binom{44}{nrPegsBorder_W}, \tag{4.3}$$

and the number of peg configurations in in the $Border_B$ area is calculated analoguely. We calculate how many configurations are possible in the *Common* area using white pegs.

$$nrCombiCommon_W = \binom{484}{nrPegsCommon_W} \tag{4.4}$$

Subsequently, we calculate the number of possible black peg configurations in the *Common* area, depending on the present white pegs.

$$nrCombiCommon_B = \binom{484 - nrPegsCommon_W}{nrPegsCommon_B} \tag{4.5}$$

Knowing the number of possible configurations of white and black pegs in the *Common* area, the total number of configurations in the *Common* area can be calculated by,

$$nrCombiCommon = nrCombiCommon_W \times nrCombiCommon_B. \tag{4.6}$$

Finally, we multiply the number of different board configurations for the three areas.

$$nrtotalPegCombi = nrCombiCommon \times nrCombiBorder_W \times nrCombiBorder_B \tag{4.7}$$

The summation of the calculated number of peg configurations for each number of pegs leads to the lower bound of approximately 10^{140} different board positions.

4.2 Game-Tree Complexity

A game tree is a directed graph where nodes represent game states and arcs represent legal moves, with the root node as the initial board position. The root node of a game tree is recursively expanded for all possible continuations from each game state until states are reached where the game comes to an end. Victor Allis [2] defines the game-tree complexity by using two auxiliary definitions:

Definition 4.1 *The solution depth of a node J is the minimal depth (in ply) of a full-width search sufficient to determine the game-theoretic value of J.*

Definition 4.2 *The solution search tree of a node J is the full-width search tree with a depth equal to the solution depth of J.*

Definition 4.3 *The game-tree complexity of a game is the number of leaf nodes in the solution search tree of the initial position(s) of the game.*

Victor Allis points out that calculating the exact game-tree complexity for a nontrivial game like chess is hardly feasible. He also mentions that $b_{\text{average}}^{d_{\text{average}}}$ gives a crude estimate of the game-tree complexity, where b_{average} is an estimate on the average branching factor and d_{average} is an estimate on the average depth of a game. We denote the initial branching factor as b_{initial}, the final branching factor as b_{final}, and the average branching factor as b_{average}. For convenience, we assume that all pegs are placed in the *Common* area during a game. This assumption implies that only a 22×22 area can be played, an area with a total capacity of 484. The following equations show the initial, final and average branching factor equations.

$$b_{\text{initial}} = 484 \tag{4.8}$$

$$b_{\text{final}} = 484 - d_{\text{average}} \tag{4.9}$$

$$b_{\text{average}} = \frac{b_{\text{initial}} + b_{\text{final}}}{2} \tag{4.10}$$

It is tricky to estimate the average game length of TwixT, because humans generally do not play out the whole game, and because human games end quickly when the playing strength is not equal. We estimate the average TwixT game length to be 60 when both players are highly skilled and if games are played out until the end. Using equation 4.10, the estimate of $d_{\text{average}} = 60$ yields an average branching factor of $b_{\text{average}} = 452$; therefore, TwixT has an estimated game-tree complexity of $O(b_{\text{average}}^{d_{\text{average}}}) \approx O(10^{159})$.

4.3 Comparison with other Games

Van den Herik *et al.* [28] investigated the state-space and game-tree complexity of several games as game characteristics for determining solvability. Table 4.1 shows the identification numbers of the games, the name of the games, and the matching state-space and game-tree complexities.

Id.	Game	State-space compl.	Game-tree compl.
1	Awari	10^{12}	10^{32}
2	Checkers	10^{21}	10^{31}
3	Chess	10^{46}	10^{123}
4	Chinese Chess	10^{48}	10^{150}
5	Connect-Four	10^{14}	10^{21}
6	Dakon-6	10^{15}	10^{33}
7	Domineering (8×8)	10^{15}	10^{27}
8	Draughts	10^{30}	10^{54}
9	Go(19×19)	10^{172}	10^{360}
10	Go-Moku (15×15)	10^{105}	10^{70}
11	Hex (11×11)	10^{57}	10^{98}
12	Kalah(6,4)	10^{13}	10^{18}
13	Nine Men's Morris	10^{10}	10^{50}
14	Othello	10^{28}	10^{58}
15	Pentominoes	10^{12}	10^{18}
16	Qubic	10^{30}	10^{34}
17	Renju (15×15)	10^{105}	10^{70}
18	Shogi	10^{71}	10^{226}

Table 4.1: State-space complexities and game-tree complexities of various games (van den Herik *et al*, 2002).

Figure 4.2 shows the estimated positions of the games in the game space. The numbers refer to the identification numbers. The grey ellipse in Figure 4.2 illustrates our estimated position of TwixT in the game space.

The estimated state-space complexity potentially exceeds all other games listed here.

The estimated game-tree complexity of TwixT is above average compared to other games, but it is lower than the game-tree complexity of Go. Go has an average branching factor of 250 and average game length of 150 moves, which results in a game-space complexity of approximately 10^{360} [2]. Twixt has an estimated average branching factor of 450 and an estimated average game length of 60 moves, which results in a game-space complexity of approximately 10^{159}. The most notable difference between both games is the average game length.

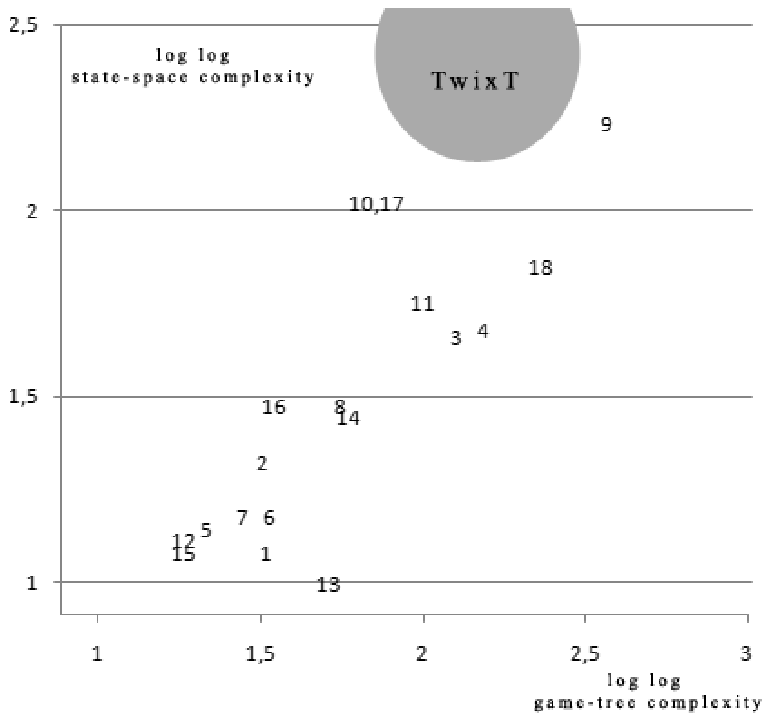


Figure 4.2: Approximate positions of games in the game space.

This is intuitive, because the shortest possible game in TwixT takes less moves than the shortest possible game in Go.

4.4 Chapter Conclusions

This chapter contributed to the determination of the position of TwixT in the game space. We calculated the number of possible peg configurations on the board of 10^{140}, which is a lower bound of the state-space complexity. The game-tree complexity is estimated to be 10^{159}. The state-space and game-tree complexity make TwixT belong to the highest category in terms of complexity. This means that TwixT is unlikely to be solved in the near future. If the complexity would have been low, then we could have sufficed with a pure search-based approach for computer TwixT. It is evident that an approach to computer TwixT must add game-specific knowledge. A comparison with other games shows that the state-space complexity of TwixT potentially exceeds all other shown games. The game-tree complexity of TwixT is above average compared to those games. The short average game length of TwixT indicates that probably many games can be played with Monte-Carlo simulations.

Chapter 5

Game-Tree Search in TwixT

In this chapter, we present well known game-independent game-tree search techniques. Furthermore, we describe how game-specific knowledge can be incorporated in computer TwixT.

5.1 Game-Tree Search Techniques

This section shortly describes the concept of a game tree and related game-independent search techniques.

5.1.1 Game Trees

A *game tree* is an acyclic directed graph where nodes represent game states and arcs represent legal moves. The root node of a game tree represents the initial board position. The root node is recursively expanded for all possible continuations of the game until game states are reached where the game ends (called a terminal position or leaf node). A *search tree* is a subtree in the game tree.

5.1.2 Minimax Search

The *minimax* search algorithm combined with a utility function is a traditional approach for creating an artificial-intelligent player in two-player, zero-sum games with perfect information. Claude Shannon described the algorithm in his article about computer chess [40].

The minimax algorithm determines the game-theoretic value of a game, assuming perfect play of both players. This assumption implies that both players use a minimax decision strategy. The player who currently has to make a move (max) tries to maximize the utility, and the opponent (min) tries to minimize the utility. John van Neumann proved that no player benefits from deviation of the minimax decision strategy [34]. The minimax algorithm performs a depth-first search in the game tree until a leaf node (terminal position) is reached. A utility function determines the *utility value* (win, loss or draw) at the leaf nodes, after which the utility value is propagated back to the root node. A max node's utility value becomes the best utility value of its successors and a min node's utility value the worst utility value of its successors.

The minimax value of the root node is the game-theoretic value of TwixT. A minimax AI player determines the minimax value of all successors of the search-tree's root and selects the move that leads to a successor position with the highest utility value.

A more practical implementation of minimax incorporates a cutoff test and replaces the utility function with a heuristic evaluation function. A cuttoff node is a terminal node or a node at a specified maximum depth in the search tree. A heuristic evaluation function estimates the utility of cuttoff nodes and all its successors are pruned.

5.1.3 $\alpha\beta$ Pruning

Allen Newell and Herbert Simon at Carnegie Mellon University and Cliff Shaw at the Rand Corporation were the first to use the $\alpha\beta$ pruning technique in their chess program [35].

The $\alpha\beta$ algorithm imposes a *search window* $[\alpha,\beta]$ at each min and max node. The lower bound α stores the best score found so far, reachable by the max player. The upper bound β stores the previously found worst score for max, reachable by the min player. During back-propagation, if the current node's value proves to be outside the search window, then a search through its siblings can be terminated. The evaluation values are propagated back to the search-tree's root, and an $\alpha\beta$ player plays the move that leads to the root's successor position with the highest expected utility.

5.1.4 Iterative Deepening

Iterative deepening is a search strategy that repeatedly invokes the $\alpha\beta$ algorithm with an increased maximum depth parameter. "This approach is the preferred search method when there is a large search space and the depth of the solution is not known" [37]. The overhead of multiple-visited nodes is small compared to the single-visited nodes at the maximum depth.

Iterative deepening is an anytime algorithm. When there is no time to complete a search at the current maximum depth, then, when no better move is found, the best move of the previous $\alpha\beta$ search is played. In a competitive setting, in each game a fixed amount of time is agreed on for each player. We use a simple time-management scheme where the granted processing time for a move is the remaining time for the player divided by a fixed constant. The relative differences in time costs between two sequential $\alpha\beta$ calls allow for a rough estimate on the finishing time of the next $\alpha\beta$ invocation. If the estimated finishing time exceeds the granted time, then there is no next search iteration, and the best move from the last search is played.

5.1.5 Transposition Table

A *transposition table*, first used in Greenblatt's chess program Mac Hack [26], is a hash table that stores previously found subtree-evaluation results. A subtree-evaluation result is reused when a position re-occurs in the search process. Such an identical re-occurred position is called a *transposition*. Storage of all board positions needs an infeasible amount of memory; therefore, transposition tables have a smaller capacity. Table entries are accessed by an entry index. The entry index of a board position is the hash key of the board position modulo the size of the transposition table.

Zobrist hashing is a commonly used hashing method for creating hash keys in board games [42]. A set of features uniquely represents the state of a board position. For Zobrist hashing, each feature is represented by a pseudo-random number. Zobrist hashing defines the hash key of a position by taking the set of features and subsequently performing an XOR operation on all representative pseudo-random numbers in an iterative fashion. When a feature is added or removed when a move is played, then the Zorbrist hash of the resulting position can be calculated by performing an XOR operation on the previous Zobrist hash and the pseudo-random number that represents the added or removed feature. The following set of features represent the state of a TwixT board position: the state of each hole on the board and the state of all links. All other information, including whose turn it is, can be derived from these features. A 24×24 TwixT board has 484 'Common Area' holes, which can be in three states (occupied by White, Black or unoccupied), and 88 border holes, which can be in 2 states. For both players there are 2008 linking possibilities between pegs, which can be in 2 states (unlinked or linked). The required number of generated pseudo-random numbers to 'Zobrist hash' a 24×24 TwixT board is $484 \times 2 + 88 + 2008$, which equals to 3064 numbers. For TwixT, we create a 32 bits hash key (of which not all bits are used) and a 32 bit lock key.

Transposition-table entries contain the following information:

- a 32 bit lock key,
- an entry type {lower/upper/exact},
- the utility value of the subtree,
- the best child move of the subtree,
- the depth of the subtree.

The hash and lock key give access to a transposition table entry. If a position is indexed to a specific entry in the table, then the lock key of the position and the stored position are compared to see if the positions match. The entry type indicates if the utility value was proven to fall inside or outside the search window at time of the search. We also store evaluation results of nodes at a maximum depth of the search tree, because evaluation in TwixT is known to be computationally expensive. An entry type's best move corresponds to the best successor move, and the depth corresponds to the depth of the subtree. For storing, we employ a 'deep' replacement strategy, which replaces a stored entry with a matching proposed entry if the proposed entry has a deeper or equal subtree than the stored entry.

We use subtree results of matching positions with equal subtree depth, because the evaluation function is known to be costly. When the entry type is 'exact', successor node search is skipped and the stored utility value is used. When the entry type is 'lower', then the subtree is explored with an alpha value set to the maximum of the current alpha and the stored utility value. The search window is updated similarly when the entry type is 'upper'.

5.1.6 History-Heuristic Move Ordering

A good move ordering improves the efficiency of the search process. Donald Knuth and Roger Moore analysed the importance of a good move ordering in the $\alpha\beta$ framework [30]. When moves are properly ordered, successful moves are tried first, which leads to many $\alpha\beta$ cutoffs.

The *history heuristic* is a simple heuristic which "maintains a history on every legal move seen in the search-tree. For each move, a record of the move's ability to cause a refutation is kept, regardless of the line of play" [38]. Whenever moves are generated, the moves are ordered in descending order based on the history-heuristic move value. Heuristic values for each legal move are stored in a table. Each table entry has an initial value of 0. Whenever during the search process a node causes an $\alpha\beta$ cut, or is found to be best, the corresponding history-heuristic table-entry is incremented by 2^d. The variable d indicates the depth of the subtree below the node. To reduce the influence of old good moves in move ordering, all history heuristic table entries are divided by 2 whenever a new search is started.

5.1.7 Monte-Carlo Sampling

The Monte-Carlo sampling technique considers the utility value of a randomly played out game as a sample. A search space S can be too large to be sampled entirely and a large sampled subset might be a good representative for S. Section 3.3 showed that Monte-Carlo techniques are used in computer Go. A simple Monte-Carlo approach in games uses win-lose-draw statistics of many randomly played games to give a good indication of the distribution of the leaf nodes in the search tree. A basic Monte-Carlo player performs a one ply search and selects the move that maximizes the expected utility based on many simulations.

5.2 Game-Specific Knowledge in computer TwixT

This section describes how game-specific knowledge can be used in computer TwixT.

5.2.1 TwixT Board-Dominance Move Ordering

Section 3.2 introduced the concept of board dominance for a player. Section 6.4.4 explains how board dominancy is deduced from a Delaunay Network board-representation. For move ordering, moves that lead to dominant board positions are tried before the moves that do not lead to a dominant board position.

Figure 5.1(a) shows a 24×24 TwixT board with one white peg in the middle. White is dominant on the board. Figure 5.1(b) shows the same board position where '$\times$' symbols indicate Black's response moves that lead to a dominant position for Black. Figure 5.1(c) again shows the same position, but now the '$\times$' symbols indicate David Bush's considered response moves [11].

We see that there is overlap between board dominant moves and the moves of an expert TwixT player, but David Bush also showed that our current model fails to look ahead. Moves that do not immediately lead to a board dominant position might have the potential to become dominant at a later stage of the game. David Bush illustrated this by showing a cluster of non-dominant moves that he would consider to play as Black. Figure 5.2(a) shows a 24×24 TwixT board with three placed pegs. White is dominant on the board. Figure 5.2(b) shows the TwixT position with '$\times$' symbols that indicate moves that lead to a dominant board position for Black. Figure 5.2(c) shows the same board position with the dominant moves for Black and with the considered non-dominant response moves of David Bush as '$\square$' symbols. The two directions in which those moves threaten to cut off White are indicated by two arrows.

David Bush's considered non-dominant response moves will likely lead to a dominant board after several moves have been played. Our board-dominance model does not look ahead and fails to see these moves.

5.2.2 TwixT Board Evaluation

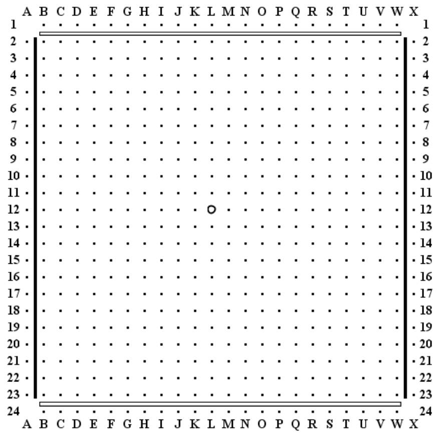
The $\alpha\beta$ algorithm uses a heuristic evaluation function to estimate the utility value of a position if the position is non-terminal. We explain in section 6.4.3 how terminal nodes are evaluated. We assign a utility value of $-\infty$ when the position is a loss, assign 0 utility when the position is a draw, and $+\infty$ when a position is a win. For non-terminal nodes we consider how much closer the evaluating player is to building a winning path of links compared to the opponent. Anshelevich models a player's distance with an electrical-resistance-circuit model in his evaluation function for Hex. The electrical-resistance-circuit model combines information on the distance yet to be travelled from side to side and the robustness of such a path. We model these concerns separately with two features, path distance and path robustness. We add a third feature to enhance discriminating power for those positions where path distance and robustness are equal. Our TwixT evaluation function is a weighted sum of three features.

$$Eval(p, player) = w_1 \cdot f_1(p, player) + w_2 \cdot f_2(p, player) + w_3 \cdot f_3(p, player) \quad (5.1)$$

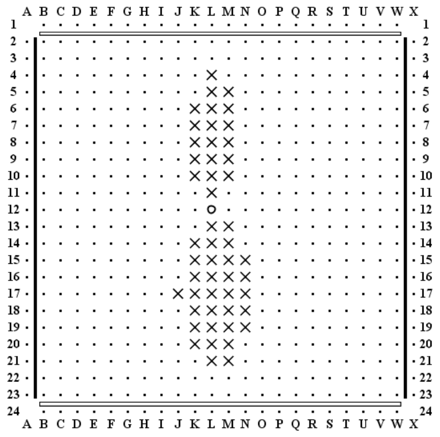
All features look at the difference between distance measurements of both players, which is similar to Anshelevich's approach.

Shortest-Path-Weight Difference Feature

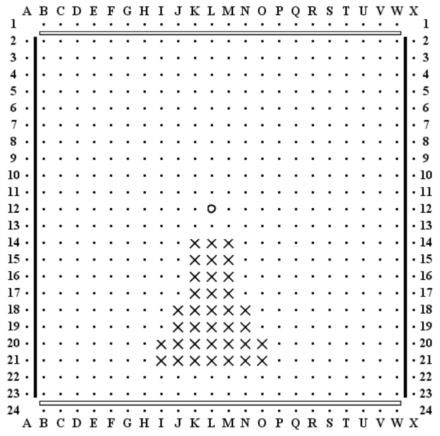
The first feature uses a shortest-path-weight distance metric to indicate a player's minimum number of links required to finalise a linked path from side to side. This feature is extracted from network board-representations by a shortest-path-weight algorithm (see Subsection 6.4.1). Let $SPW(p, Black)$ and $SPW(p, White)$ be the shortest-path-weight distance metric for Black and for White. Feature 1 is defined by Equation 5.2.



(a) A 24 × 24 TwixT board with one peg.

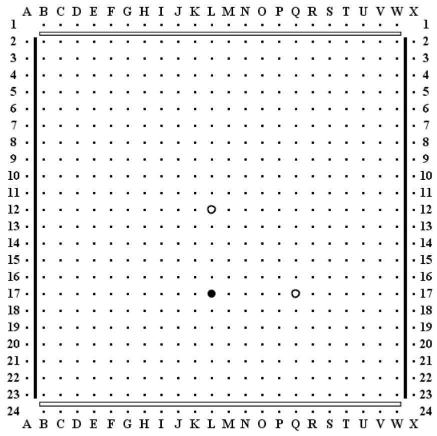


(b) Board dominant response moves.

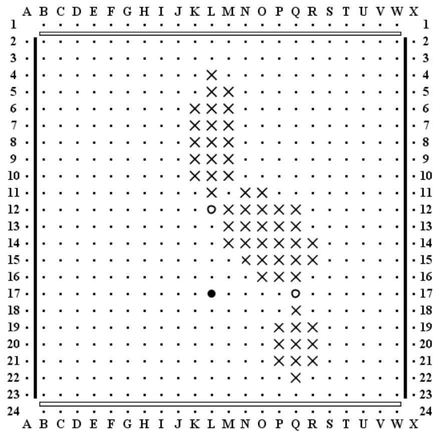


(c) David Bush's considered response moves.

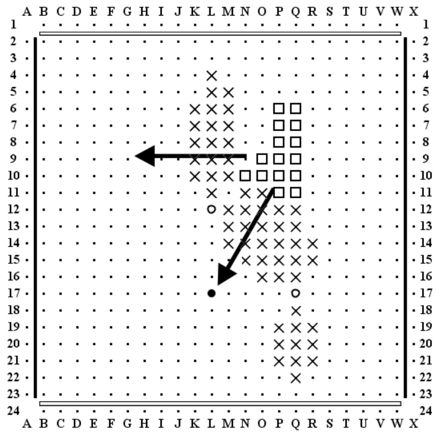
Figure 5.1: A 24 × 24 TwixT board position with one peg (a), board dominant response moves (b), and David Bush's response moves (c).



(a) A 24 × 24 TwixT board position with three pegs.



(b) Board dominant response moves.



(c) Board dominant response moves with David Bush's considered non-dominant response moves.

Figure 5.2: A 24 × 24 TwixT board position with three pegs (a), board dominant response moves (b), and board dominant response moves with David Bush's considered non-dominant response moves (c).

$$f_1(p, player) = \begin{cases} SPW(p, Black) - SPW(p, White) & \text{if player is White,} \\ SPW(p, White) - SPW(p, Black) & \text{if player is Black.} \end{cases} \tag{5.2}$$

Maximum-Flow Difference Feature

The second feature uses a maximum-flow distance metric to indicate a player's number of shortest paths available to finalize a linked path from side to side. This feature is extracted from network board representations by a maximum-flow algorithm (see Subsection 6.4.2). Let $MF(p, Black)$ and $MF(p, White)$ be the maximum-flow metric for Black and for White. Feature 2 is defined by Equation 5.3.

$$f_2(p, player) = \begin{cases} MF(p, Black) - MF(p, White) & \text{if player is White,} \\ MF(p, White) - MF(p, Black) & \text{if player is Black.} \end{cases} \tag{5.3}$$

Shortest-Path-Weight List Difference Feature

The third feature does not calculate the difference between a single measurement of both players. Instead it looks at a weighted difference between multiple found potential winning paths. The maximum flow algorithm can output more than only the maximum flow for a player. The algorithm determines multiple shortest paths of which the shortest-path weight is known; therefore, the algorithm can return an ordered list with shortest-path-weights for each player. This additional information does not cost extra processing time since we want to know the maximum flow anyway, and we explore how the information can be used for evaluation.

An intuitive approach is to look at the difference in shortest-path-weights between both lists. The number of elements in the list is variable, as for each player an arbitrary number of augmenting paths can be found. The measurements are ordered and the considered number of measurements for each player are made equal by discarding an ith shortest path measurement of a player when the opponent does not have an ith shortest path. Assume n measurements for both players and let $WM = [wm_o, wm_1, wm_2, ..., wm_n]$ be a measurement series for White and $BM = [bm_o, bm_1, bm_2, ..., bm_n]$ be a measurement series for Black. Furthermore, let w_i be a weight for the difference between the ith measurements. The third feature $f_3(p, player)$ is defined by Equation 5.4.

$$f_3(p, player) = \begin{cases} \sum_{i=0}^{n} w_i \cdot (bm_i - wm_i) & \text{if player is White,} \\ \sum_{i=0}^{n} w_i \cdot (wm_i - bm_i) & \text{if player is Black.} \end{cases} \tag{5.4}$$

The first found path is assumed to be more important than the later found ones. This can be modelled by a weighted sum of the differences where first path difference gets the highest weight and all subsequent path differences get a lower weight in a linear decreasing fashion. We want to be able to compare lists with an arbitrary number of considered measurements. We dynamically adjust the weights such that the total weight remains the same and the relative linear decreasing order of weights is also maintained. Assume that we want to distribute $10,000$ weight units over n elements with $n < 100$ in a linear decreasing order and where the elements are indexed from 0 to $n-1$. We calculate the weight of element with index i as follows:

$$\int 200 - 2x\,dx = -(x-200)x \tag{5.5}$$

$$\text{xPos}(x) = x \cdot \frac{100}{n} \tag{5.6}$$

$$w_i = (\text{xPos}(i) - 200)\text{xPos}(i) - (\text{xPos}(i_{+1}) - 200)\text{xPos}(i_{+1}) \tag{5.7}$$

Figure 5.3 illustrates how the surface below the $f(x) = 200 - 2x$ function defines the weights when there are four or eight measurements.

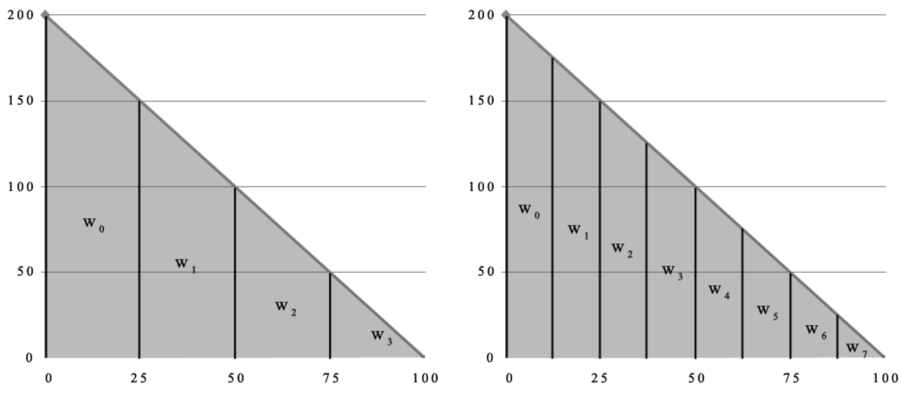


Figure 5.3: Weight distribution of four measurements (left), and the weight distribution of eight measurements (right).

Evaluation Feature Weights

The three features are weighted in the evaluation function. The first and second features are assumed to be equally important and they have a higher priority than the third feature. The third feature, has a low priority and is only added to enhance discriminating power when the first two features are unable to discriminate between positions. The shortest-path-weight list difference feature does not have a higher priority than the maximum-flow feature. The shortest-path-weight list difference evaluation is less stable because the result is based on the number of considered paths, which may lead to interpretation mistakes. The weights are empirically set to $w_1 = 10^4$, $w_2 = 10^4$, and $w_3 = 1$.

5.2.3 TwixT Monte-Carlo Enhancements

We enhance the Monte-Carlo player in TwixT by postponing game termination checks. A game termination check takes significantly more time than a simple peg placement. One can skip game termination checks in the beginning of the game, because extra peg placement after a game is ended does not change the outcome of the game. When it becomes likely that a game is finished, game termination can be repeatedly checked after n peg placements.

5.3 Chapter Conclusions

This chapter focused on how to combine known game-tree search techniques and game-specific knowledge in an approach for computer TwixT. Our approach on combining game-tree search techniques and game-specific knowledge is to implement combinations in AI players. We described game-independent game-tree search techniques that are commonly used in two-player zero-sum games with perfect information. We described two types of AI players. The $\alpha\beta$ player, uses a minimax decision strategy for move selection. Selected game-independent enhancements for $\alpha\beta$ include: iterative deepening, history-heuristic move ordering, and using a transposition table. The second type of AI player, the Monte-Carlo player, uses a one-ply search and selects the best move based on statistical win/draw/loss statistics of many randomly played games.

We also described how game-specific knowledge can be added. We described how to order moves such that moves that lead to dominant board positions are tried first. We explained how our evaluation function works. It uses three features that all express the difference between player win distances. The first feature looks at the difference between both players in the minimal required number of links yet to be placed in order to win the game. The second feature looks at the difference between both players in the number of shortest paths found from side to side. The third feature looks at the differences between both players in an ordered list of path weights. All features require that a TwixT board position is translated into a network representation. Network board-representations are used to extract player distances. The usage of network board-representations in computer TwixT is explained in Chapter 6. We also mentioned how we enhanced the Monte-Carlo player. Basically, we reduced the overhead of game-termination checks by postponing them.

Chapter 6

Network Search in TwixT

In this chapter, we explain how networks are used in computer TwixT.

Chapter contents: Network Search In TwixT — Introduction to Networks, Networks in the TwixT, Network-Topology Update Rules, Network Feature Extraction, Chapter Conclusions.

6.1 Introduction to Networks

A *graph* $G = (V, E)$ consists of a set of vertices (sometimes called nodes) V and a finite set of pairs of distinct vertices, called edges, E.

$$V = \{v_1, v_2, v_3, \ldots, v_n\} \tag{6.1}$$

$$E = \{e_1, e_2, e_3, \ldots, e_m\} \text{ with } e_k = \{v_i, v_j\} \wedge v_i, v_j \in V \tag{6.2}$$

A *directed graph* (digraph) $D = (V, A)$ consists of a set of vertices V and a finite set of ordered pairs, called arcs, A.

$$A = \{a_1, a_2, a_3, \ldots, a_m\} \text{ with } a_k = \{v_i, v_j\} \wedge v_i, v_j \in V \tag{6.3}$$

A *network* $N = (V, A, s, t, w)$ is a digraph (V, A) in which two vertices are distinguished as source and target vertex (s and t), and in which each arc has a non-negative weight. A weight function $w : A \to \mathbb{R}$ determines the weight of an arc.

Connected vertices 'shake hands' in our implementation of networks. If there is an arc from u to v, then there is also an arc from v to u. As a result, an undirected network is expressed by a directed network. For convenience, we define networks as if they were undirected $N = (V, E, s, t, w)$ with $w : E \to \mathbb{R}$.

6.2 Networks in TwixT

Our TwixT game engine stores the following four types of networks: link networks, allowed-link networks, combined networks, and Delaunay networks. We explain each network type from the perspective of White. Black's perspective can be easily derived and needs no further explanation.

6.2.1 Link Networks

We store White's link network and Black's link network, where each edge $e(v_1, v_2)$, $v_1, v_2 \notin \{s, t\}$ represents a placed link of the corresponding player. Figure 6.1 shows White's initial link network overlaid on a 12×12 TwixT board.

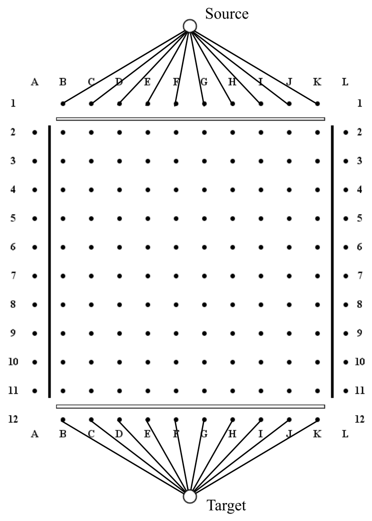


Figure 6.1: White's initial link network.

The set of vertices V of White's link network is initialized such that a representative vertex is included in V for each hole on the TwixT board. We denote the representative vertex of the hole with row r and column c as $v_{r,c}$. In White's link network, upper-row peg vertices connect with s and lower-row peg vertices connect with t. White's initial link network contains no other edges, because there are no links on the board. Link networks have a default weight function $w(u, v) = 1$.

6.2.2 Allowed-Link Networks

We store White's allowed-link network and Black's allowed-link network, where each edge $e(v_1, v_2)$, $v_1, v_2 \notin \{s, t\}$ represents a possible link placement of the corresponding player. Figure 6.2 shows White's initial allowed-link network overlaid on a 12×12 TwixT board.

The set of vertices V of White's allowed-link network is initialized such that a representative vertex is included in V for each hole on the TwixT board. In White's allowed-link network, upper-row peg vertices connect with s and

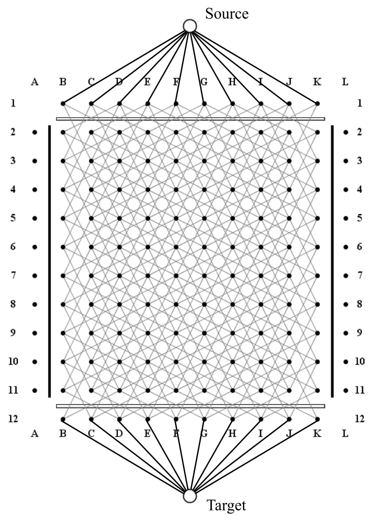


Figure 6.2: White's initial allowed-link network.

lower-row peg vertices connect with t. White's allowed-link network contains edges such that there is a representative edge in E for each possible link for White between holes on the board. Allowed links are allowed in the sense that they are not made impossible by the opponent. They are not instantly allowed according to the rules. Extra white peg's at a knight's move distance may be required to place the link. Allowed-link networks have a default weight function $w(u, v) = 1$.

6.2.3 Combined Networks

We store White's combined network and Black's combined network, where each edge $e(v_1, v_2)$, $v_1, v_2 \notin \{s, t\}$ represents placed link or allowed links for the corresponding players. White's combined network represents all holes on the board by a vertex $v_{r,c} \in V$. The set of edges of White's combined network is the union of edges in White's link network (E_{LN_W}) and edges in White's allowed-link network (E_{ALN_W}). An edge in White's combined network represents a placed link, an allowed link or a connection with the source or target vertex. We created two weight functions w_{travel} and w_{capacity}.

The first weight function models travel costs between vertices.

$$w_{\text{travel}}(u,v) \begin{cases} 0 & \text{if } e(u,v) \in E_{LN_W} \\ 1 & \text{if } e(u,v) \notin E_{LN_W} \wedge e(u,v) \in E_{ALN_W} \\ +\infty & \text{otherwise} \end{cases} \text{for } u, v \in V \quad (6.4)$$

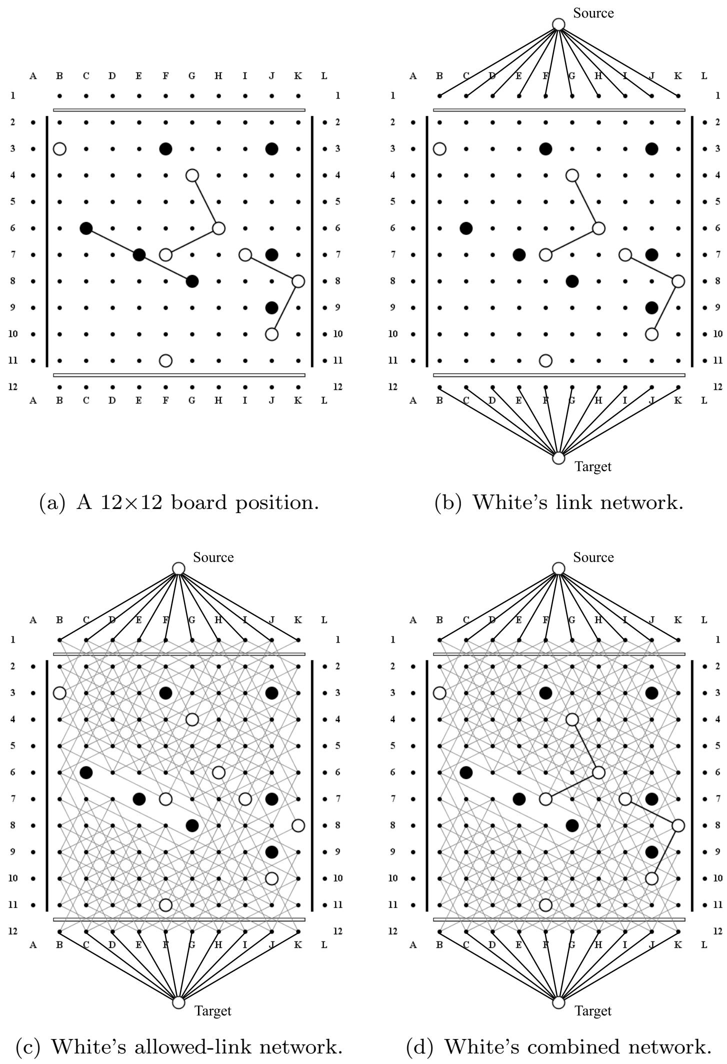
Edges that connect with source or target and edges that represent placed links get 0 travel costs. Edges that represent allowed link get a travel cost of 1.

The second weight function models flow capacity between vertices.

$$w_{\text{capacity}}(u,v)\begin{cases}+\infty & \text{if } e(u,v)\in E_{LN_W}\\ 1 & \text{if } e(u,v)\notin E_{LN_W}\wedge e(u,v)\in E_{ALN_W}\\ 0 & \text{otherwise}\end{cases} \text{ for } u,v\in V \quad (6.5)$$

Edges that connect with source or target and edges that represent placed links get $+\infty$ capacity. Edges that represent allowed link get a capacity of 1.

Figure 6.3 illustrates how White's perspective on a 12×12 board position is represented by White's link network, White's allowed-link network, and White's combined network. The network representations are overlaid on a 12×12 TwixT board.



(a) A 12×12 board position. (b) White's link network.

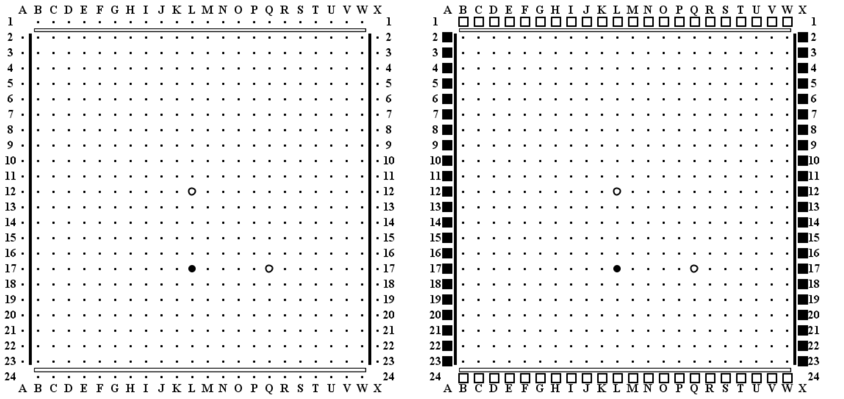
(c) White's allowed-link network. (d) White's combined network.

Figure 6.3: A 12×12 board position (a), White's link network (b), White's allowed-link network (c), and White's combined network (d).

6.2.4 Delaunay Networks

We store White's Delaunay network, where edges represent White's Voronoi-region connectedness. The process of creating a Delaunay network representation of a TwixT board position is explained below.

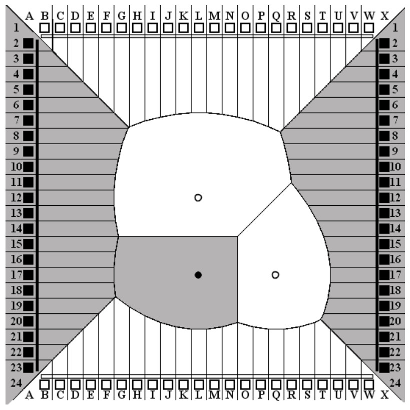
First, non existent pegs, which we call *virtual pegs*, are added to a TwixT board position with to correct strength-interpretation mistakes on the board. Black and white virtual pegs are placed at the corresponding border rows to indicate that the 'upper' and 'lower' border rows are under influence of White, and the 'left' and 'right' border rows are under influence of Black. Virtual pegs of corresponding colour are also added at the middle of a link to indicate the strength of the link. Figure 6.4 shows an example 24×24 TwixT board and the same board position with added virtual pegs. The white squares, at the 'upper' and 'lower' border rows, indicate White's virtual pegs and the black squares, at the 'left' and 'right' border rows, indicate Black's virtual pegs.

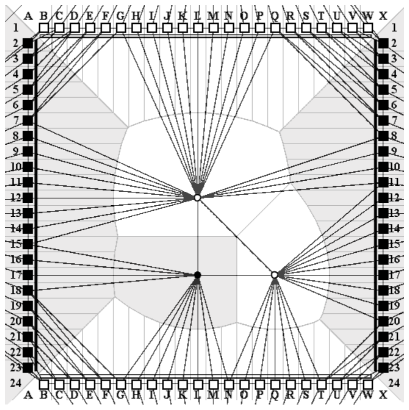


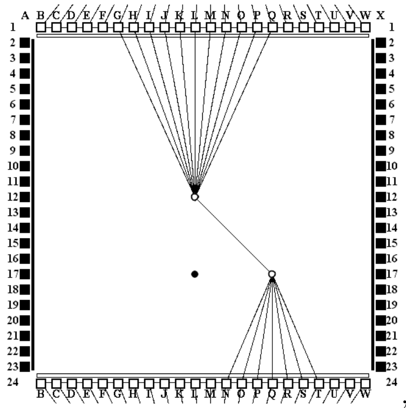
(a) A 24×24 TwixT board position. (b) A TwixT position with added virtual pegs.

Figure 6.4: A 24×24 TwixT board position (a), and a TwixT position with added virtual pegs (b).

Subsequently, we create a Delaunay triangulation, the dual representation of a Voronoi tessellation, to model Voronoi-region connectedness. The dual representation is all we need for dominance checking, because all pegs with a shared Voronoi region boundary are connected in the Delaunay triangulation. Many methods exist for drawing a Delaunay triangulation from a set of points on a plane. We used Fortune's sweep algorithm [7]. Ownership information is added to the Delaunay edges to indicate to which of the players an edge belongs. An edge is owned by White if it connects white pegs, owned by Black if it connects black pegs, and owned by no player otherwise. Figure 6.5(a) shows the Voronoi tessellation of the TwixT position of Figure 6.4(b). Figure 6.5(b) shows the Voronoi representation with the Delaunay triangulation. The Delaunay triangulation has edges between pegs that are owned by White, owned by Black, and owned by no player. A close examination shows that adjacent Voronoi regions are connected in the Delaunay triangulation. Figure 6.5(c) shows the Delaunay triangulation with White's owned edges, and Figure 6.5(d) shows the Delaunay triangulation with Black's owned edges.


(a) Voronoi representation of a TwixT position.


(b) Voronoi representation with Delaunay triangulation.


(c) Delaunay triangulation with only White's owned edges.

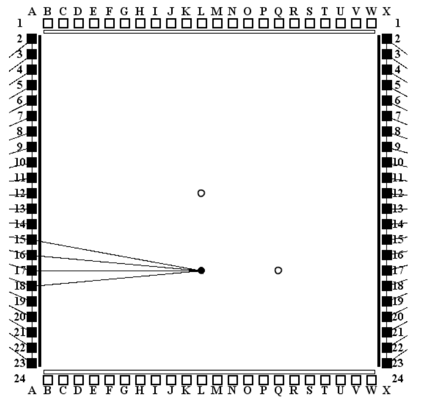

(d) Delaunay triangulation with only Black's owned edges.

Figure 6.5: Voronoi representation of a TwixT position (a), Voronoi representation with Delaunay triangulation (b), Delaunay triangulation with only White's owned edges (c), and Delaunay triangulation with only Black's owned edges (d).

Finally, a Delaunay network is created from the Delaunay triangulation to capture White's Voronoi-region connectedness as a network. All White's pegs and virtual pegs are represented as a vertex in the Delaunay network. White's Delaunay triangulation edges are represented by edges in the Delaunay network. The source vertex is connected with all upper-row peg vertices and a target vertex connects with all lower-row peg vertices. Delaunay networks have a default weight function $w(u, v) = 1$.

6.3 Network-Topology Update Rules

Network-topology update rules change the topology of the network representations according to TwixT linking rules. Only the link and allowed-link networks need to be updated after a move is performed on the board. Combined networks are derived from the link and allowed-link networks and we do not iteratively update the Delaunay network board-representation after moves are played.

We explain how the topology of link networks and allowed-link networks are updated after the placement of a white peg. Black is not allowed to link with $v_{r,c}$ for each white peg placement with row r and column c. Therefore, Black's allowed-link network is updated such that $v_{r,c}$ has no outgoing edges. Because of the auto-linking rule (see Section 2.1.3), one or more links are considered to be placed if there are white pegs at a knight-jump distance of one. White's allowed-link network indicates if a considered link is allowed. The placement of a link is represented by an edge between the corresponding vertices on White's link network. Placed links are removed from White's allowed-link network. Black is not allowed to cross the placed link. Therefore, all link-crossing allowed-links are removed from Black's allowed-link network. Figure 6.6(a) shows an example board position. White has one link in each corner and has one unlinked peg in the middle of the board. Figure 6.6(b) shows the connections that are going to be removed from Black's allowed-link network. Figure 6.6(c) shows Black's allowed-link network after all link-crossing allowed links are removed. We do not show the source and target vertices and do not show the source and target connecting edges for clarity reasons.

6.4 Network Feature Extraction

We use network search algorithms to extract features from the network representations of TwixT board positions. Extracted features include: shortest-path-weight, maximum flow, game termination, and board dominance.

6.4.1 Shortest-Path-Weight Feature

The shortest-path-weight feature expresses a player's minimal number of required link placements to win the game. We explain the shortest-path-weight metric from White's perspective. White's shortest-path weight is extracted from White's combined network with weight function w_{travel} (see Subsection 6.2.3).

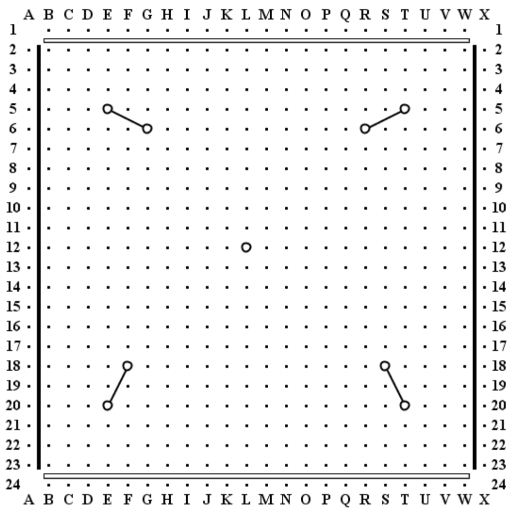
The weight of a path, $p = v_1 \to v_2 \to, \ldots, v_{k-1} \to v_k$, is defined by

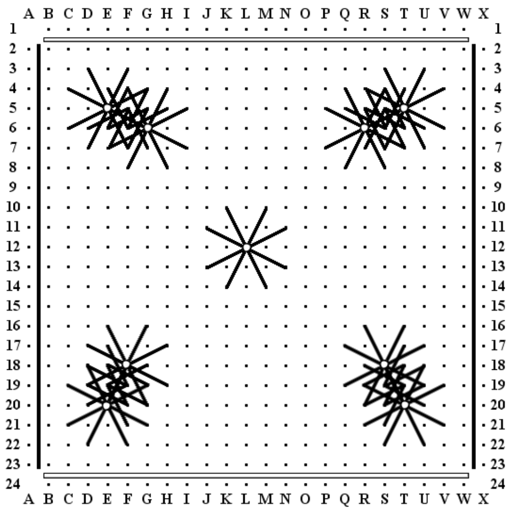
$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1}). \tag{6.6}$$

The *shortest-path weight* from s to t is defined by

$$\text{ShortestPathWeight}(s,t) = min\{w(p) : \text{p is a path from } s \text{ to } t\}. \tag{6.7}$$

Typically a breadth-first search (BFS) is used to find the shortest-path weight in a graph if the graph is a uniform-cost digraph. Dijkstra's shortest-path algorithm is typically used when the graph is a weighted digraph. White's combined network is not a uniform-cost digraph, but it is a special case of a weighted digraph, because all edges are binary weighted. This property has implications for the breadth-first-search algorithm and the Dijkstra's shortest-path algorithm. A basic implementation of BFS cannot deal with the fact that


(a) A 24×24 board board position with four links and an unlinked peg.


(b) Black's unallowed links.

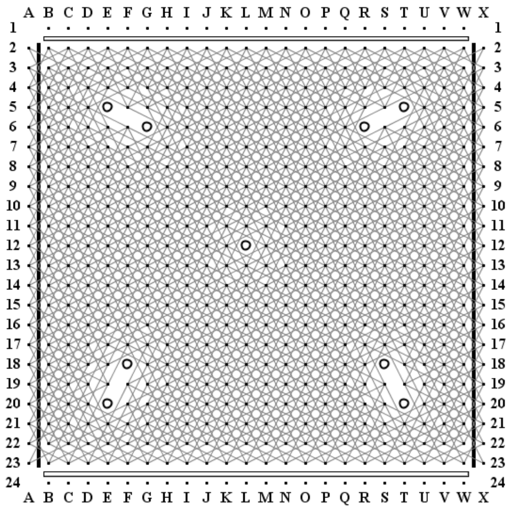

(c) Black's allowed-link network.

Figure 6.6: A 24×24 board position with four links and an unlinked peg (a), Black's unallowed links (b), and Black's allowed-link network (c).

there are edges with weight 1 and edges with weight 0. Dijkstra's shortest-path algorithm is an overdesigned solution for finding a shortest-path weight in a combined network, because it takes into account the possibility of different edges having various non-negative weights while edges can only be binary weighted. An extension of the standard BFS overcomes the limitations of not being able to deal with edges of weight 0 or 1. The extension is straightforward, but the basics of BFS have to be clear in order to understand the small change in the algorithm.

The basic BFS algorithm 6.1, as shown in Algorithm 6.1, uses a First-In-First-Out (FIFO) queue Q. All vertices $v \in V$ store the currently known distance to the source $d[v]$ and $d[v]$ is initially set to ∞. The source vertex is added to the empty FIFO queue and its known distance to the source is set to 0. As long as the queue is not empty, the BFS repeatedly dequeues the first vertex from Q to be visited and performs a *relaxation step* on all non-visited adjacent vertices. (The relaxation step is labelled as (i) in the pseudo code.) The relaxation step of an adjacent vertex includes: updating the known distance to the source; checking if the target is reached; and, if the target is not found, putting the vertex at the end of the queue to be visited later. The search terminates when the target is found or when the queue is empty. The BFS returns the target's known distance to the source.

```
**Algorithm 6.1:** BREADTH-FIRST SEARCH()

for each $v \in V$
do
    $d[v] \leftarrow \infty$
    $d[s] \leftarrow 0$
    $Enqueue(Q, s)$
    while $Q \neq \emptyset$
    do
        $u \leftarrow Dequeue(Q)$
        for each $v \in Adj[u]$                                (i)
        do
            if $d[v] = \infty$
            then
                $d[v] \leftarrow d[u] + 1$
                if $v = t$
                then return $(d[v])$
                else $\{Enqueue(Q, v)$
return $(\infty)$
```

Extended Breadth-First Search

We extended the BFS by changing the relaxation step of the algorithm to overcome the limitations of not being able to deal with edges of weight 0 or 1. We changed the position where a relaxed vertex is added in the FIFO queue when the target is not reached. An adjacent vertex is placed at the end of the queue when it is connected by an edge with weight 1. This corresponds to scheduling the vertex to be explored at a later stage of the search. An adjacent vertex is placed in front of the queue when it is connected by an edge of weight 0.

For the ordering of Q in a BFS the following holds: If vertex v comes after u in Q it implies that $d[v] = d[u]$ or $d[v] = d[u] + 1$. The placement of vertices reachable with edge weight 0 at the front of Q does not violate the ordering. The placement of vertices reachable with edge weight 1 to the end of Q also doesn't violate the ordering. This proves that Q remains properly ordered with the vertex placement as described above.

The distance updating happens as follows. When a neighbouring vertex is reachable by an edge with weight 1, then its new distance becomes the minimum of its old known distance and the known distance of the currently visited vertex plus 1. When a vertex is reachable by an edge with weight 0, then its new known distance becomes the minimum of its old known distance and the known distance of the currently visited vertex. Our implementation also keeps track of the parent vertex.

Dijkstra's shortest-path algorithm [18] is assumed to be known to the reader. The difference between the Dijkstra's shortest-path algorithm and extended BFS is that Dijkstra's shortest-path algorithm uses a priority queue instead of a FIFO queue, and that the relaxation step is slightly different. Initially all vertices are placed in the priority queue with a known distance to the source set to $+\infty$. During the relaxation step the known distance is updated and if the target is not found the priority queue typically has to be reordered based on the known distance to the source. As we have shown, there is no need for reordering; therefore, we have a small improvement using the extended BFS algorithm.

6.4.2 Maximum-Flow Feature

The Maximum flow feature expresses a player's freedom to travel from side to side. "The maximum flow problem is to find a feasible flow through a single-source, single-sink flow network that is maximum" [18]. White's maximum flow is extracted from White's combined network. Two weight functions are used, w_1 and w_2.

The maximum flow can be found by using the Edmonds-Karp algorithm [18]. The algorithm is very straightforward. A copy of the combined network is created to serve as a residue network. Repeatedly, a shortest path from s to t is determined (using the travel weighting function w_{travel}) and extracted from the residue network until there is no path between s and t. After an augmenting path is found, all path edges with a capacity of 1 are deleted from the residue network. The path edges with infinite capacity are not deleted. The number of shortest paths found determines the maximum flow. Special caution is required concerning path selection, because different path-selection strategies might lead to a different number of augmenting paths to be found. We remove paths according to a strategy where a shortest path with vertices closest to the target is selected.

6.4.3 Game-Termination Feature

The game termination feature indicates if a game is a draw, a win for White, a win for Black, or still in progress. The game is a draw if both players are unable to make an uninterrupted chain of links between the corresponding sides. In terms of networks: a board position is a draw if for both players no path exists from s to t in the corresponding combined network. The existence of a path between s and t is checked with an informed depth-first search. Our 'informed' version of a DFS inserts adjacent vertices into an ordered priority queue with an increasing Euclidean distance order of vertices in the queue. Figure 6.7(b) and Figure 6.7(c) show the combined networks for White and Black respectively for the drawn position (Figure 6.7(a)) that we have seen in Chapter 2 (see Figure 2.2). Both figures illustrate that a drawn position leads to a failed search for a path from s to t on White's combined network and Black's combined network.

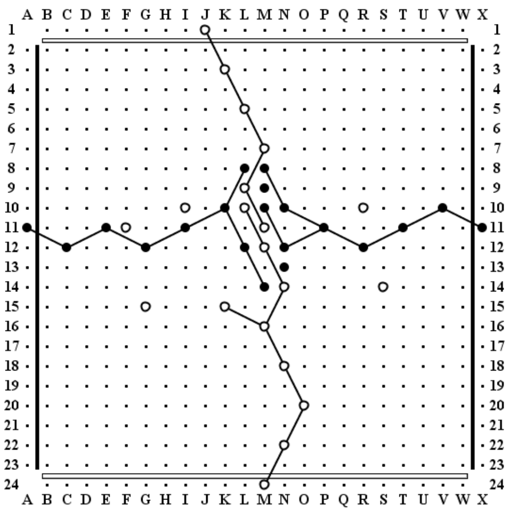
A win for a player can be checked strictly or softly. For White the game is won in a strict sense if there is a path from s to t in White's link network. There is a 'soft' win for White if there is no path from s to t in Black's combined link network. A soft win does not have to be a strict win because White, the soft winning player, might not have connected to the border rows. Connecting the border rows is normally a formality but it is not always possible. There is an exceptional situation where Black, the soft losing player, cannot move because there are no legal moves left. We declare such a situation as a win for White and use the soft win condition in our AI players. An informed depth-first search based on the Euclidian distance to the target checks for possible paths between sides for White and Black. If the game is not a draw, not a win for White, and not a win for Black, then the game is still in progress.

6.4.4 Board-Dominance Feature

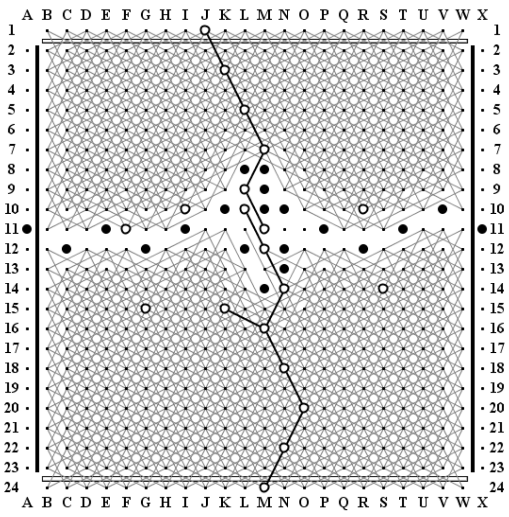
The board dominance feature expresses if a player can travel between the corresponding opposing sides via owned dominant gaps. Section 3.2 explains the concept of board dominance, and Delaunay Networks are explained in Subsection 6.2.4. In network terms: a board is dominant for White if White has a path from source to target in the Delaunay network. The existence of such a path is checked with a best-first search. If a board position is not dominant for White, then it is dominant for Black.

6.5 Chapter Conclusions

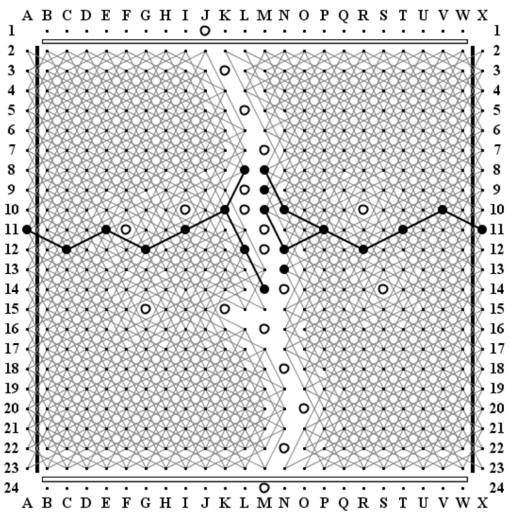
This chapter explained how networks are used in computer TwixT. A short introduction to network theory clarified the terminology and concepts used throughout the rest of the chapter. We have explained the following four types of networks: link networks, allowed-link networks, combined networks, and Delaunay networks. Two different edge-weight functions are expplained. The first weight function models travel costs between vertices and the second flow capacity between vertices. Network-topology update rules change the topology of the network board-representations according to TwixT linking rules. Search algorithms extract features from the network board-representations. We explained how we extract the minimal number of links for a player to win the game, extract the maximum flow of a player, and how to check for game termination and board dominance. Edges are binary weighted by the travel cost weighting function, which means that the weight of an edge is either zero or one. We extended the relaxation step of a normal breadth-first search to allow binary weighted edges.



(a) A drawn TwixT position.



(b) White's combined network.



(c) Black's combined network.

Figure 6.7: A drawn TwixT position (a), White's combined network (b), and Black's combined network (c).

Chapter 7

Experiments and Results

In this chapter, we measure the effectivity and efficiency of known search techniques and game-specific knowledge.

Chapter contents: Experiments — Experimental Design, $\alpha\beta$ Player Experiments, Monte-Carlo Player Experiment, Player Effectivity Experiment, Chapter Conclusions.

7.1 Experimental Design

Our approach is to implement combinations of game-specific knowledge and known game-tree search techniques in AI players. The AI players are tested on effectivity and efficiency. A TwixT simulation environment implements TwixT computer rules (Subsection 2.1.3) and supports automated play between two AI agents. Automated game playing takes place on an 8×8 TwixT board and both players conform to a game playing time of 10 minutes per player per game. The time scheduling per move for both players is based on the player's remaining time in the game divided by 4. The time scheduling allocates more time to the first moves. We implemented two types of AI players. The first AI player is an $\alpha\beta$ player. The implemented $\alpha\beta$ enhancements include: iterative deepening, history-heuristic move ordering, board-dominance move ordering, and the use of a transposition table. We use network search algorithms to extract features from network board-representations. Extracted features from the network include: shortest path weight, maximum flow, board dominance, and game termination (see Section 5.2). The board-dominance feature is used for move ordering, and the other features are used in the evaluation function. The second AI player is a Monte-Carlo player.

The effectivity of an AI player is measured by the win statistics from 100 simulated games versus other players. The effectivity of an AI player is made explicit by looking at the decision-making process. We test which combination of techniques leads to the best $\alpha\beta$ player, and subsequently we evaluate if the strongest $\alpha\beta$ player wins versus the basic Monte-Carlo player.

7.1.1 Board Size

Our choice of testing on an 8×8 board deserves some explanation. Testing on efficiency and effectivity of TwixT AI players on large boards has severe disadvantages. Large boards require more moves by both players to finish a game. The requirement of playing more moves within a constant time frame leads to less calculation time per move on average. Large boards reduce the look-ahead capability of a player, because on large boards there are on average more possible response moves for each game state. Having less time to search a larger space leads to bad playing strength. We want to allow playing strength comparable to an amateur player and game play on a 8×8 board seems to enable this. The drawback of playing on small board sizes is that they offer little variance in gameplay, because board positions allow just a few reasonable response moves.

7.2 $\alpha\beta$ Player Experiments

Our $\alpha\beta$ player uses the standard $\alpha\beta$ algorithm with iterative deepening and with the evaluation function as described in section 5.2.2. We add a randomized number between 0 and 1000 to the evaluation function to prevent that the games are deterministic. The randomized number is approximately 2% of the evaluation value range.

7.2.1 Evaluation Function

We use the throughput of the evaluation function as an indicator of the efficiency of the evaluation function. The throughput of the evaluation function is measured by how many times the initial TwixT board can be evaluated within one minute. The evaluation time of an initial TwixT board is the worst case evaluation time for our evaluation function. Table 7.1 shows the number of evaluated initial TwixT board positions for various board sizes.

Number of evaluated initial TwixT board positions per minute	*Board Size*
63432	6×6
22196	7×7
6966	8×8
3536	9×9
1351	10×10
728	11×11
353	12×12

Table 7.1: The number of evaluated initial TwixT board positions within one minute for various board sizes.

7.2.2 Move Ordering

We test the impact on efficiency and effectivity of history-heuristic (HH) move ordering (see Section 5.1.6), board-dominance (BD) move ordering (see Section 5.2.1), and board-dominance history-heuristic (BDHH) move ordering. BDHH is a combination of board-dominance move ordering and history-heuristic move ordering. BDHH has a board dominant list of moves in front of a non-dominant board list. Moves within each list are ordered based on their history-heuristic value in descending order.

For each test we play 100 games between the $\alpha\beta$ player with and without move ordering.

Move Ordering Effectivity Results

Tables 7.2 - 7.4 show the win results of using HH, BD, and BDHH move ordering.

	Winning player	
Starting player	$\alpha\beta$	$\alpha\beta$ **with HH ordering**
$\alpha\beta$	1	49
$\alpha\beta$ **with HH ordering**	6	44

Table 7.2: Win results of the $\alpha\beta$ player versus an $\alpha\beta$ player with history-heuristic (HH) move ordering.

	Winning player	
Starting player	$\alpha\beta$	$\alpha\beta$ **with BD ordering**
$\alpha\beta$	15	35
$\alpha\beta$ **with BD ordering**	15	35

Table 7.3: Win results of the $\alpha\beta$ player versus an $\alpha\beta$ player with board-dominance (BD) move ordering.

	Winning player	
Starting player	$\alpha\beta$	$\alpha\beta$ **with BDHH ordering**
$\alpha\beta$	0	50
$\alpha\beta$ **with BDHH ordering**	19	31

Table 7.4: Win results of the $\alpha\beta$ player versus an $\alpha\beta$ player with board-dominance history-heuristic (BDHH) move ordering.

We can conclude that the $\alpha\beta$ player is most effective when it uses history-heuristic move ordering. It is remarkable that the win/loss results of the best player show more wins when the opponent starts. It is also remarkable that the BDHH player loses 19 games to the $\alpha\beta$ player when it starts.

Move Ordering Efficiency Results

Tables 7.5 - 7.7 show the efficiency results of using HH, BD, and BDHH move ordering.

We can conclude that the $\alpha\beta$ player is most efficient when it uses history-heuristic move ordering.

	$\alpha\beta$		$\alpha\beta$ **with HH ordering**	
Depth	*Nodes*	*Time (ms)*	*Nodes*	*Time (ms)*
1	46	257	46	288
2	992	5936	298	1462
3	11885	65385	4000	21993

Table 7.5: $\alpha\beta$ player history-heuristic move ordering efficiency results.

	$\alpha\beta$		$\alpha\beta$ **with BD ordering**	
Depth	*Nodes*	*Time (ms)*	*Nodes*	*Time (ms)*
1	46	274	46	327
2	975	5930	622	5466
3	13211	72784	8884	70629

Table 7.6: $\alpha\beta$ player board-dominance (BD) move ordering efficiency results.

	$\alpha\beta$		$\alpha\beta$ **with BDHH ordering**	
Depth	*Nodes*	*Time (ms)*	*Nodes*	*Time (ms)*
1	46	269	46	334
2	981	6014	331	3608
3	13114	70784	4668	37637

Table 7.7: $\alpha\beta$ player board-dominance history-heuristic (BDHH) move ordering efficiency results.

7.2.3 Transposition Table

We test the efficiency of using a transposition table by looking at the differences between an $\alpha\beta$ player with and without transposition table while allowing a player to determine the best move within one hour. The tested position is the initial board position, and the evaluation function does not have an added random number. We use a transposition table with 131.072 (2^{17}) entries and both players use history-heuristic move ordering. Table 7.8 shows that less time is required to complete a search at a maximum depth when using a transposition table. Less nodes are visited, because previously stored evaluation results are used.

	$\alpha\beta$ **HH with transposition table**		$\alpha\beta$ **HH**	
Depth	*Nodes*	*Time*	*Nodes*	*Time*
1	49	469	49	453
2	189	1187	189	1172
3	2735	19219	3906	27453
4	36943	191844	50168	253844
5	215766	1021640	593984	3000891

Table 7.8: The efficiency results of the $\alpha\beta$ HH player with and without transposition table.

7.3 $\alpha\beta$ Player versus Monte-Carlo Player Experiment

A player's effectivity corresponds with the game playing strength versus other players. We tested the $\alpha\beta$ player with history-heuristic ordering and transposition table versus the basic Monte-Carlo player by playing 100 matches. Table 7.9 shows the win results of the $\alpha\beta$ player versus the basic Monte-Carlo player. The Monte-Carlo player has an average simulation performance of 77.654 simulations per minute, and on average 24 random moves are needed before the game terminates.

	Winning player	
Starting player	**$\alpha\beta$ with HH and TT**	**Basic Monte Carlo**
$\alpha\beta$ with HH and TT	46	4
Basic Monte Carlo	34	16

Table 7.9: Win results of the $\alpha\beta$ player with history heuristic (HH) and transposition table (TT) versus the basic Monte-Carlo Player.

We can conclude that the $\alpha\beta$ player with history heuristic and transposition table is much stronger than the basic Monte-Carlo player.

7.4 Chapter Conclusions

In this chapter we measured the effectivity and efficiency of known search techniques and game-specific knowledge. We explained the experimental design, have shown the results and interpreted the results. The experiments showed that with our experimental setup the $\alpha\beta$ player with history heuristic and transposition table is most efficient and effective for computer TwixT.

Chapter 8

Conclusions

The aim of this research was to investigate how a computer program can be written that plays the game of TwixT as efficient and effective as possible.

Chapter contents: Conclusion — Research Questions Revisited, Problem Statement Revisited, Future Research.

8.1 Research Questions Revisited

We revisit each of the research questions from section 1.2:

Research Question 1: *What game-specific knowledge used by human players is applicable to computer TwixT?*

Chapter 2 contributed to the acquisition of game-specific knowledge that needs to be represented in a TwixT playing program. We described the official rules, the pen and paper version rules, and a rule set for computer TwixT. Our computer TwixT rule set adopts the TwixT PP rules with an added auto-linking rule. We examined strategic and tactical TwixT knowledge that is used by expert players. Human players estimate a board-position's utility value based on a complex interaction between many interrelated features that cannot be easily extracted by a computer. The strategic and tactical heuristics work under specific conditions, but it is unclear how these heuristics can be effectively combined and implemented in a TwixT playing program. Accurate predictive evaluation of a board-position's utility value requires a deep look-ahead capability. The look-ahead capability is limited in TwixT, because of the many possible continuations of a game for each board position.

Research Question 2: *What can we learn from research that is related to TwixT?*

Chapter 3 investigated research in similar games. Research on the connection games Hex, Bridg-It and the Shannon Switching Game showed that network board-representations can evaluate the game state. Section 6.2 explains the network representations that we use in computer TwixT. We have also seen how network topology update rules can model the underlying game mechanics. Section 6.3 explains how network topology update rules capture TwixT linking rules. Anshelevich's Hex evaluation function considers how much closer a player is to building a winning chain than the opponent is to building a winning chain. Anshelevich's Hex program HEXY measures player distances by the total resistance in a player's electrical-resistor-circuit board-representation. Jack van Rijswijk's Hex playing program QUEENBEE uses the 'two-distance' to measure player distances in a player's network board-representation. Our TwixT evaluation function is inspired by their work (see Section 5.2.2).

Research Question 3: *What is the complexity of TwixT?*

Chapter 4 contributed to the determination of the position of TwixT in the game space. We calculated the number of possible peg configurations on the board to be 10^{140}, which is a lower bound of the state-space complexity. The game-tree complexity is estimated to be 10^{159}. The state-space and game-tree complexity make TwixT belong to the highest category in terms of complexity. This means that TwixT is unlikely to be solved in the near future. If the complexity would have been low, then we could have sufficed with a pure search-based approach for computer TwixT. It is evident that an approach to computer TwixT must add game-specific knowledge. A comparison with other games shows that the state-space complexity of TwixT potentially exceeds all other shown games. The game-tree complexity of TwixT is above average compared to those games. The short average game length of TwixT indicates that probably many games can be played with Monte-Carlo simulations.

Research Question 4: *How can we efficiently and effectively combine game-specific knowledge and known game-tree search techniques in a TwixT AI player?*

Chapter 5 focused on how to combine known game-tree search techniques and game-specific knowledge in an approach for computer TwixT. Our approach on combining game-tree search techniques and game-specific knowledge is to implement combinations in AI players. We described game-independent game-tree search techniques that are commonly used in two-player zero-sum games with perfect information. We described two types of AI players. The $\alpha\beta$ player, uses a minimax decision strategy for move selection. Selected game-independent enhancements for $\alpha\beta$ include: iterative deepening, history-heuristic move ordering, and using a transposition table. The second type of AI player, the Monte-Carlo player, uses a one-ply search and selects the best move based on statistical win/draw/loss statistics of many randomly played games.

We also described how game-specific knowledge can be added. We described how to order moves such that moves that lead to dominant board positions are tried first. We explained how our evaluation function works. It uses three features that all express the difference between player win distances. The first feature looks at the difference between both players in the minimal required number of links yet to be placed in order to win the game. The second feature looks at the difference between both players in the number of shortest paths found from side to side. The third feature looks at the differences between both players in an ordered list of path weights. All features require that a TwixT board position is translated into a network representation. Network board-representations are used to extract player distances. The usage of network board-representations in computer TwixT is explained in Chapter 6. We also mentioned how we enhanced the Monte-Carlo player. Basically, we reduced the overhead of game-termination checks by postponing them.

Chapter 6 explained how networks are used in computer TwixT. A short introduction to network theory clarified the terminology and concepts used throughout the rest of the chapter. We have explained the following four types of networks: link networks, allowed-link networks, combined networks, and Delaunay networks. Two different edge-weight functions are expplained. The first weight function models travel costs between vertices and the second flow capacity between vertices. Network-topology update rules change the topology of the network board-representations according to TwixT linking rules. Search algorithms extract features from the network board-representations. We explained how we extract the minimal number of links for a player to win the game, extract the maximum flow of a player, and how to check for game termination and board dominance. Edges are binary weighted by the travel cost weighting function, which means that the weight of an edge is either zero or one. We extended the relaxation step of a normal breadth-first search to allow binary weighted edge

Chapter 7 focussed on the last research question: "How can we efficiently and effectively combine game-specific knowledge and known techniques in a TwixT AI player?". The main focus was on measuring the efficiency and the effectivity of the search process.

We explained the experimental design. Our approach is to implement combinations of game-specific knowledge and known game-tree search techniques in AI players. The AI players are tested on effectivity and efficiency. A TwixT simulation environment implements TwixT computer rules (Subsection 2.1.3) and supports automated play between two AI agents. Automated game playing takes place on a 8×8 TwixT board and both players conform to a game playing time of 10 minutes per player per game. The time scheduling per move for both players is based on the player's remaining time in the game divided by 4. The time scheduling allocates more time to the first moves. We implemented two types of AI players. The first AI player is an $\alpha\beta$ player. The implemented $\alpha\beta$ enhancements include: iterative deepening, history-heuristic move ordering, board-dominance move ordering, and the use of a transposition table. We use network search algorithms to extract features from network board-representations. Extracted features from the network include: shortest-path weight, maximum flow, board dominance, and game termination (see Section 5.2). The board-dominance feature is used for move ordering, and the other features are used in the evaluation function. The second AI player is a basic Monte-Carlo player.

The effectivity of an AI player is measured by the win statistics from 100 simulated games versus other players. The effectivity of an AI player is made explicit by looking at the decision-making process. We tested which combination of techniques leads to the best β player and subsequently we evaluated if the strongest $\alpha\beta$ player wins versus the basic Monte-Carlo player.

Experiments showed that, with our experimental setup, the $\alpha\beta$ player with history heuristic and transposition table is most efficient and effective for computer TwixT.

8.2 Problem Statement Revisited

All answers to the research questions contributed to answer the problem statement of Section 1.2:

How can a computer program be written that plays the game of TwixT as effectively as possible?

Experiments show that, with our experimental setup, the $\alpha\beta$ player with history heuristic and transposition table is most efficient and effective for computer TwixT.

8.3 Challenges and Future Research

Our contribution to the domain of computer TwixT can serve as a basis for further research. The main challenge for future research will be finding more game-specific enhancements that lead to a higher effectivity and efficiency of AI players.

Little enhancements can be made by tuning parameters. It would be interesting to see how the $\alpha\beta$ player plays with different player distance metrics, edge weighting functions, augmenting path deletion strategies, evaluation feature weights, etc.

The real challenge is to find new methods that reduce the complexity or enhance predictive evaluation in a time efficient manner. Our methods can be taken as a basis, or as a source of inspiration, but they have to be extended to allow for better playing strength and better performance on bigger boards.

Our implementation of board-dominance checking was not time efficient, because the underlying board representation, the Delaunay triangulation, was created from scratch after a move is played. We expect an increase in performance of dominancy checking when the Delaunay triangulation is iteratively updated.

The randomly played games during Monte-Carlo simulations lead to situations where all response moves are equally likely to be played. The Monte-Carlo player's view on the utility of a move gets seriously distorted, because response moves in TwixT are not equally likely to be played. Assume that we create a rule base with manually coded local patterns, where each pattern we defines a probabilistic distribution of surrounding response moves. What would happen if a basic Monte-Carlo player's move selection is based on the local patterns? More advanced Monte-Carlo simulation techniques, such as UCT [31], can also be tested in the future.

Bibliography

[1] H.K. Ahn, S.W. Cheng, O. Cheong, M. Golin, and R.V. Oostrum. Competitive Facility Location along a Highway. In *9th Annual International Computing and Combinatorics Conference*, volume 2108, pages 237–246, 2001.

[2] V. Allis. *Searching for Solutions in Games and Artifcial Intelligence*. PhD thesis, University Maastricht, 1994.

[3] V.V. Anshelevich. Hexy Plays Hex. `http://home.earthlink.net/~vanshel/`, [Online; accessed 21/03/09].

[4] V.V. Anshelevich. The Game of Hex: An Automatic Theorem Proving Approach to Game Programming. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pages 189–194, Menlo Park, 2000. AAAI Press.

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

[6] A. Beck, M.N. Bleicher, and D.W. Crowe. *Excursions into Mathematics.* A K Peters, Ltd, 2000.

[7] B. van de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry.* Springer Verlag, 2nd revised edition ed edition, 2000.

[8] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron. The Challenge of Poker. *Artificial Intelligence Journal*, 134(1-2), 2002.

[9] B. Bouzy and B. Helmstetter. Monte-Carlo Go Developments. In H.J. van den Herik, H. Iida, and E.A. Heinz, editors, *Proceedings of the 10th Advances in Computer Games Conference (ACG-10)*, pages 159–174. Kluwer Academic, 2003.

[10] C. Browne. *Connection Games - Variations on a Theme.* K Peters, Ltd., 2005.

[11] D. Bush. Peronsal communication.

[12] D. Bush. TwixT. TwixT- Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Twixt`, [Online; accessed 21/03/09].

[13] D. Bush. An Introduction To TwixT. *Abstract Games Magazine Issue 2 Summer 2000*, pages 9–12, 2000.

[14] D. Bush. TwixT Tactics Part 1. *Abstract Games Magazine Issue 4 Winter 2000*, pages 6–8, 2000.

[15] D. Bush. TwixT Tactics Part 2. *Abstract Games Magazine Issue 8 Winter 2001*, pages 14–16, 2001.

[16] G.M.J.B Chaslot, J-T. Saito, B. Bouzy, J.W.H.M. Uiterwijk, and H.J. van den Herik. Monte-Carlo Strategies for Computer Go. In P.Y. Schobbens, W. Vanhoof, and G. Schwanen, editors, *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence*, pages 83–91. Namur, 2006.

[17] O. Cheong, N. Linal, S. Har-Peled, and J. Matousek. The One-Round Voronoi Game. In *18th Annual ACM Symposium on Computational Geometry*, pages 97–101, 2002.

[18] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and S. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, second edition edition, 2001.

[19] R. Coulom. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers, editors, *Proceedings of the 5th Computers and Games Conference (CG 2006)*, Berlin, 2007. Springer-Verlag.

[20] P. Eirich and S. Medcalf. Naming Board Elements and Peg Structures - TwixT. `http://twixt.wetpaint.com/page/Naming+Board+Elements+and+Peg+Structures?t=anon`, [Online; accessed 21/03/09].

[21] M. Gardner. Mathematical games - Concerning the game of Hex, which may be played on the tiles of the bathroom floor. *Scientific American*, 197:145–150, 1957.

[22] M. Gardner. *The Scientific American Book of Mathematical Puzzles and Diversions*. Simon and Schuster, 1959.

[23] M. Gardner. *The Second Scientific American Book of Mathematical Puzzles and Diversions*. Simon and Schuster, 1961.

[24] M. Gardner. Mathematical games. *Scientific American*, 205(1):148–168, July 1961.

[25] M.L. Ginsberg. GIB: Steps toward an expert-level bridge-playing program. In *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 584–589, 1999.

[26] R.D. Greenblatt, D.E. Eastlake, and S.D. Crocker. The Greenblatt Chess Program. In *Fall Joint Computer Conference*, volume 31, pages 801–810, 1967.

[27] A. Hensel. Blocks, balance, and potential paths. `http://twixt.wetpaint.com/page/Blocks,+balance,+and+potential+paths?t=anon`, [Online; accessed 21/03/09].

[28] H.J. van den Herik, J.W.H.M Uiterwijk, and J. van Rijswijck. Games Solved: Now and in the Future. *Artificial Intelligence*, 134:277–311, 2002.

[29] International Computer Games Association. 7th computer olympiad. http://www.cs.unimaas.nl/olympiad2002/, [Online; accessed 21/03/09].

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

[31] L Kocsis and C Szepesvári. Bandit based Monte-Carlo Planning. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, editors, *Proceedings of the EMCL 2006. Number 4212 in LNCS*, volume 4212 of *Lecture Notes in Computer Science (LNCS)*, pages 282–293, Berlin, 2006. Springer.

[32] A. Lehman. A Solution of the Shannon Switching Game. *Journal of the Society for Industrial and Applied Mathematics*, 12:687–725, 1964.

[33] D. Mazzoni and K. Watkins. Uncrossed Knight Paths is NP-complete, 1997. http://www.math.uni-bielefeld.de/~sillke/PROBLEMS/Twixt_Proof_Draft, [Online; accessed 21/03/09].

[34] J von Neumann. Zur Theorie der Gesellschaftsspiele. *j-MATH-ANN*, 100:295–320, 1928.

[35] A. Newell, J.C. Shaw, and H.A. Simon. Chess-playing programs and the problem of complexity. *IBM Journal of Research and Development*, 4:320–335, 1958. Reprinted (1963) in Computers and Thought (eds. E.A. Feigenbaum and J. Feldman), pp. 39-70. McGraw-Hill, New York, N.Y.

[36] J. van Rijswijck. Search and evaluation in Hex. Technical report, University of Alberta, 2002.

[37] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[38] J. Schaeffer. The history heuristic. *Journal of the International Computer Chess Association*, 6(3):16–19, 1983.

[39] J. Schwagereit. Programs to play TwixT. http://www.johannes-schwagereit.de/Twixt.html, [Online; accessed 21/03/09].

[40] C.E. Shannon. Programming a Computer for Playing Chess. *Philosophical Magazine*, 41:256–275, 1950.

[41] R. Wayne Schmittberger. *New Rules for Classic Games*. Wiley, 1992.

[42] A. L. Zobrist. A New Hashing Method with Application for Game Playing. *Technical report 88*, Computer Science Department, The University of Wisconsin,Madison, WI, USA, 1970. Reprinted in (1990) *ICCA Journal*,Vol. 13,No. 2, pp. 69-73.

Appendix A

Matlab Code State-Space Complexity

```
% Kevin Moesker
% MATLAB code for calculating the lower bound on the state-space
% complexity for TwixT.

% Note: I make an important assumption that there are only 50 pegs
% available for each player, and the placement of links is
% disregarded.

% common bookkeeping vars
n = 24;
borderBlack = (n-2) * 2;
borderRed = (n-2) * 2;
commonHoles = (n-2) * (n-2);

% init boardpositions to 0
totalBoardPositions = 0;

% the official rules state that there are 100 pegs total
% 50 black and 50 white

totalPegs = 100; % limit on the number of pegs 50 for each player

for j = 0: totalPegs
    j % j is the number of considered pegs
    % calculate the number of red and black pegs
    if(mod(j, 2) == 0)
     % we have even number of pieces
         nrRedPegs = j / 2;
         nrBlackPegs = j / 2;
     else
     % we have odd number of pieces
         nrRedPegs = (j-1) / 2;
```

```
        nrBlackPegs = j - nrRedPegs;
    end

    % the number of pegs for black and red are known
    % * determine the number of possible distributions over the
    % three areas.
    % (BlackOnly area - RedOnly area and Common Area)
    % * determine the bounds for iteration over the blackOnly
    % and redOnly area's.
    if(nrBlackPegs < borderBlack)
        maxBlackcounter = nrBlackPegs;
    else
        maxBlackcounter = borderBlack;
    end
    if(nrRedPegs < borderRed)
        maxRedcounter = nrRedPegs;
    else
        maxRedcounter = borderRed;
    end

    for i = 0: maxBlackcounter
        % i number of pegs in blackOnly area
        nrCombinationsBlackSide = nchoosek(borderBlack, i);
        for z = 0: maxRedcounter
            % z number of pegs in redOnly area
            nrCombinationsRedSide = nchoosek(borderRed, z);
            blackLeftForCombi = nrBlackPegs - i;
            redLeftForCombi = nrRedPegs - z;
            if(blackLeftForCombi + redLeftForCombi <= commonHoles)
                nrCombiBlack = nchoosek(commonHoles,
                 blackLeftForCombi);
                nrCombiRed = nchoosek(commonHoles -
                 blackLeftForCombi,
                redLeftForCombi);
                nrCombinationsCombiVlak = nrCombiBlack *
                 nrCombiRed;
                extraBoardPositions = nrCombinationsBlackSide *
                nrCombinationsRedSide *nrCombinationsCombiVlak;
                totalBoardPositions = totalBoardPositions +
                extraBoardPositions;
            end
        end
    end
end

% MATLAB OUTPUT:
% totalBoardPositions 1.5987e+139
% In statespacecomplexity at 60
% Warning: Result may not be exact. Coefficient is greater
% than 1.000000e+015  and is only accurate to 15 digits.
```