Endgame Detection in Tron

Lukas Kang

June 19, 2012

Abstract

This paper investigates whether a Monte-Carlo Tree Search (MCTS) agent playing the game of Tron can be improved by applying evaluation functions to the Play-Out phase of MCTS. Tron is a two-player board game and originated in the movie *TRON* in 1982. Two heuristics, namely the Voronoi heuristic and the "Tree of Chambers" heuristic are introduced and implemented in a MCTS agent with several enhancements including MCTS-Solver.

Experiments revealed that both heuristics perform well in certain scenarios, but fail in others. One drawback is the computational overhead caused by the heuristics. When tested with a fixed number of Play-Outs the agents using the new heuristics perform better and are able to beat the existing agents.

1 Introduction

Artificial Intelligence (AI) in games is a growing research field in past and in present times. The idea is to develop computer programs capable of playing games. More specifically, these computer programs should be able to play games as adaptive, efficient and also as victorious as possible. Researchers have developed a variety of game AI models, one of which is tree search. $\alpha\beta$-minimax search [7, 15] explores the game tree up to a certain level using iterative deepening search. Leaf nodes are evaluated according to an evaluation function. Such an evaluation function estimates the utility of a certain game state. $\alpha\beta$ is heavily dependent on the accuracy of the evaluation function, since a bad evaluation causes the algorithm to choose a weak move. Monte-Carlo Tree Search (MCTS) [3, 4, 8] does not necessarily require an evaluation function. It selects and expands nodes according to statistics and then randomly or semi-randomly simulates one complete game. The results of these simulated games are then propagated backwards through the whole tree up to the root. These simulations completely replace an evaluation function in MCTS. Each node stores a value depending on the simulation results, according to which the program chooses the next move. It is, however, possible to assess nodes with an evaluation function in MCTS as well [6].

MCTS has shown to outperform other search techniques in quite some games [2]. Go is the most prominent example of a game where MCTS is the only search technique which is able to compete with human grandmasters [9, 12], although still with handicap. Recently, research has been conducted on MCTS applied to the game of Tron [5]. It is a two-player simultaneous-move board game. Both players simultaneously move either vertically or horizontally on a chess-like board leaving walls on squares they visited. The player who lasts longest, i.e. who crashes into a wall after the opponent, wins the game. If both players move to the same square or both crash into a wall at the same time the game ends in a draw. In 2011 Den Teuling [5] tested several Play-Out strategies and other search enhancements in MCTS applied to Tron and concluded that their success is highly dependent on the board configuration. Especially an $\alpha\beta$-minimax program with a sophisticated evaluation function [14] outperformed all tested MCTS players. This paper aims at further enhancing the existing MCTS Tron player [5]. More specifically, a static evaluation function is implemented. The evaluation function is able to prematurely terminate a simulation [10, 17]. It is expected that nodes get a better estimate of the current game state value. An efficient evaluation function could allow the program to evaluate more nodes, because games are not simulated until the end anymore.

The following two research questions are investigated:

1. How can we construct evaluation functions that are able to accurately assess the game-theoretic score of Tron positions?
2. Do more accurate estimates of endgame positions increase the performance of an MCTS Tron player?

This paper is organized as follows. First, in Section 2 the rules and the origin of the game of Tron are discussed. Section 3 explains MCTS and its application to Tron. Next, Section 4 gives a detailed description of the new evaluation functions. Additionally, their advantages and disadvantages are discussed. This is followed by an explanation of how the new evaluation functions are applied to MCTS in Section 5. Experiments are given in

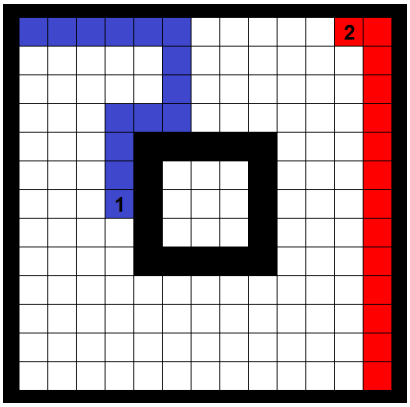

Figure 1: The game of Tron after 13 turns with a square in the middle as an initial obstacle (based on [5]).

Section 6. Finally, conclusions are drawn and future research suggestions are made in Section 7.

2 Tron

The game of Tron originated in the movie *TRON* by Walt Disney Productions from 1982. In the movie the game occurs as a multi-player game where the players ride on motorbike-like vehicles called "Light Cycles". It is played on an initially empty square field. The vehicles can change their direction of travel only in a 90 degree angle. Additionally they construct walls behind themselves. The path they have traveled becomes a wall and hence cannot be used again. The player's goal is to steer a vehicle making all opponents crash into a wall before her/himself. There are, though, differences between the game in the movie and the game of Tron, as referred to in this paper. The game of Tron, as referred to in this paper, is a two-player game where both players move at constant speed. Unlike the game in the movie, Tron can be played not only on square boards but on any rectangular board of $m \times n$ squares. Furthermore boards do not have to be empty at the beginning but may contain any kind of obstacles (see Figure 1). At each turn both players simultaneously move to an adjacent square. Note that at each turn a player has to decide upon a maximum of 3 squares to move onto (see Figure 1). The game ends as its original in the movie *TRON*. One player wins if she/he survives the other. When both players crash simultaneously, the game ends in a draw. The difficulty of Tron is to estimate the space which is left on the board for oneself.

3 Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) [3, 4, 8] is a search technique using a tree as data structure. As in other tree search techniques applied to computer games each level of the tree consists of nodes representing game states after a move of a certain player. In a two-player game having players i and j all even numbered layers correspond to game states where player i has to make a move and vice versa. The main difference between MCTS and most search techniques applied to games is that there is no requirement for an evaluation function. For instance $\alpha\beta$ search only works well with an adequate evaluation function. Leaf nodes are assigned a value by that function. The best move is selected according to values propagated back in the tree. In contrast MCTS uses randomized simulated games instead of an evaluation function. It can be split up in four distinct phases [3] (see Figure 2):

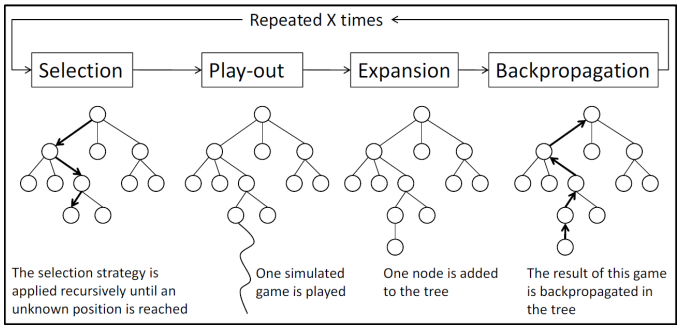

Figure 2: The four phases of Monte-Carlo Tree Search (based on [3]).

Selection. First, starting at the root, MCTS recursively selects a child from the current node. This selection can be made either according to a sophisticated strategy or random. Ideally the selection strategy should keep a balance between exploiting known and good move sequences and exploring new moves [3]. A selection strategy keeping this balance is Upper Confidence Bounds applied to Trees (UCT) [8]. UCT converges to the game-theoretic value of a position after a sufficient number of simulations. Each node saves both the number of times it was visited n_i and the current value v_i. Commonly v_i is the winning ratio of all simulated games which passed this node. Another possibility is v_i to be the total number of wins. In Tron a node's value is its winning ratio. UCT uses these values to select a child keeping a balance between exploitation and exploration of the tree. In fact, the following formula shows how UCT selects a child k from a node p:

$$k = argmax_{i \in I} \left(v_i + C \times \sqrt{\frac{\ln n_p}{n_i}} \right) \qquad (1)$$

In Tron UCT showed to perform best with the constant C set to 10 [5]. The Selection phase lasts until

there is no more child available to select, i.e. when a leaf node is selected.

Play-Out. Beginning with the game state represented by the selected node a game is simulated until an endgame position emerges. Such a Play-Out can be completely random. Mostly, however, sophisticated Play-Out strategies are applied. They specify which moves are preferred at which position. In Tron one Play-Out strategy is the wall hugging strategy [5]. It prefers moves that leave the most space afterwards to others. There are several maps available for Tron. It turned out that the best performing Play-Out strategies differ from one map to another [5]. None of the strategies beats all others in all tests. It was shown, however, that random Play-Outs are the most robust strategy throughout the testing. It won more than 50% of the test games against almost all other strategies. Therefore random Play-Outs are used in the game. Finally, the result of the game is stored in the node where the Play-Out started.

Expansion. After the Play-Out the Expansion phase starts. One or more children of the current leaf node are generated and added to the tree. Since in Tron there are at most three possible moves from each position, all children are added [5].

Backpropagation. Every time a simulated game is completed the n_i and v_i of all nodes which lie on the path from the leaf node back to the root are updated, i.e. the values are propagated backwards through the tree.

When the Backpropagation phase is completed selection starts again. This process is repeated until a certain number of iterations or a time threshold is reached. In that way every iteration can change the end result of MCTS. In the best case more iterations cause MCTS to ultimately select better moves and play better. At last, MCTS selects the move to be performed in the game. In our case, it selects the most 'secure' child of the root [3, 5]. The formula $v_i + \frac{A}{\sqrt{n_i}}$ specifies the secureness of a node i. Trial-and-error showed A=1 to be appropriate for experiments [5].

4 Determining the game-theoretic value

Evaluating positions in Tron involves estimating the space left to be filled on the board. An agent that is able to accurately assess the space left to fill would play optimally. It would always make a decision that maximizes the surplus of its own space left to be filled compared to the opponent's. The complexity of assessing the free space left to fill for one player is dependent on the current game position. In Tron there are positions possible, for which it is almost trivial to assess the space for both players. Even the future winner can be determined quickly in that case. Other positions, nevertheless, are much more difficult to evaluate. Subsection 4.1 deals with the space estimation of the existing Tron agent of Den Teuling [5]. Subsection 4.2 introduces the Voronoi heuristic, which is the basis concept of one of the two newly developed evaluation functions. The "Tree of Chambers" heuristic is elaborated in Subsection 4.3. It is more sophisticated than the Voronoi heuristic and the basis of the second new evaluation function.

4.1 Checker Board Heuristic

Den Teuling [5] uses a space estimation technique in order to enhance his UCT agent. A lower and an upper bound of the space left are calculated. For the lower bound, a Play-Out is performed with a Player using the so-called wall-following heuristic. This player always moves to an adjacent square which is adjacent to as many walls as possible and which leaves the most space for the following moves. The lower bound is the number of moves performed by that player. For calculating the upper bound the board is treated as a checker board containing gray and white squares. The idea is that every time the player moves, he steps on a square with a different color. The available space M is calculated as follows. $M = Z - |c_g - c_w|$, where Z is the total free space on the board and c_g and c_w are the number of gray and white squares, respectively [5]. The difference of the colored squares is subtracted from the free space of the board. Squares that cannot be reached are not counted. The "Checker Board Heuristic" assesses the space left to fill for a player only if he is separated from his opponent.

4.2 Voronoi Heuristic

A so-called Voronoi diagram (VD) [1, 16] indicates which regions of the board can be reached earlier by which player (see Figure 4). Squares that are at an equal distance for both players have a special role. They belong to the so-called 'battlefront' [14]. It plays an important role in the 'Tree of Chambers' heuristic (Subsection 4.3). A heuristic which works well as an evaluation of this kind of positions is the difference in size of the players region and the opponent's region [5, 14]. It is referred to as the Voronoi heuristic. Generally there are three distinguishable scenarios, two of which are well assessed by the Voronoi heuristic:

Separate components. Figure 3 shows a position of Tron after 20 moves. The players are in separate components of the board, i.e. the players cannot reach each other. In this case the task of assessing the available space is easy. Counting and comparing the sizes of both components will reveal

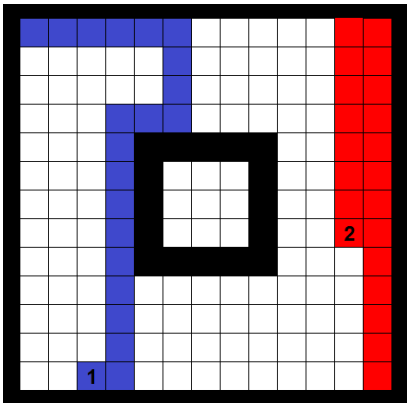

Figure 3: The game of Tron after 20 moves.

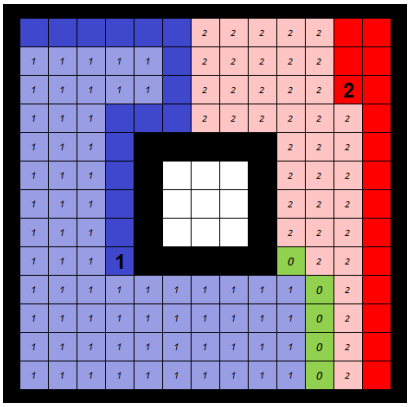

Figure 4: Voronoi diagram of the game of Tron after 15 moves. Squares containing a 1 belong to the region of Player 1 and vice versa. Squares containing a 0 belong to the battlefront.

who will ultimately win the game. In Figure 3 all empty squares on the right side of the board belong to the region of Player 2 (Red). Player 1 (Blue), on the other hand, has the smaller region on the left. Counting the squares on both sides shows that Player 2 has a surplus of 24 squares. If Player 2 efficiently fills his region, he is going to win the game. The Voronoi heuristic easily comes to the same conclusion, because all squares in a player's component of course also are of lower distance to him than to the opponent.

Connected component. It becomes more complex when the players are located in the same component of the board. The position shown in Figure 4 occurred after 15 moves. To assess the surplus of space for a player in this kind of position a Voronoi diagram (VD) [1, 16] can be computed. In the described scenario the Voronoi heuristic accurately assesses the space and is able to anticipate which

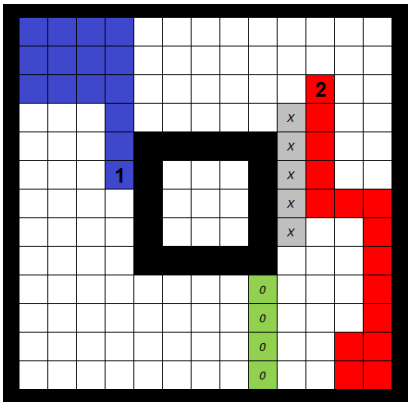

Figure 5: Tron game after 14 moves. Articulation points are indicated by an X. Squares which belong to the battlefront contain a 0.

player has better chances to win the game. The Voronoi diagram in Figure 4 evaluates a Tron position after 15 moves. Player 1's region (left) clearly is larger than Player 2's. Player 1 (Blue) can reach any square in his region before Player 2 (Red) does and vice versa. Player 1 simply has to move towards the battlefront (indicated by 0's) and then cut the connected component into two parts. The resulting position has two separate components with Player 1's region larger than Player 2's.

The third scenario includes positions containing more than one chamber. Subsection 4.3 explains what chambers are, how the "Tree of Chambers" heuristic is able to cope with them and why the Voronoi heuristic is not. The Voronoi heuristic explained in this section is implemented in an agent that is further referred to as the Voronoi Player.

4.3 Tree of Chambers

Positions with articulation points can cause the Voronoi heuristic to make mistakes. An articulation point is a square which, when passed by a player, cuts off parts of the board for that player. Figure 5 shows a Tron position after 14 moves. The Voronoi heuristic would determine Player 2 as the winner, since his region (57 squares) is larger than Player 1's region (53 squares). What makes this evaluation problematic is the fact that there are five articulation points on this board (indicated by an X). In this case more than just counting the free space is required to evaluate the position. If Player 2 (Red) chooses to fill the space in the upper right corner, Player 1 (Blue) can move to the lowest of the articulation points and cut his opponent off. When Player 2 decides to move down immediately the upper right region is cut off and cannot be filled anymore. In both cases there is no way for Player 2 to fill his entire Voronoi region.

Hence, Player 1 definitely wins the game. His region does not contain articulation points and therefore can be filled completely in both cases.

During the Google AI Challenge 2010, Andy Sloane developed an $\alpha\beta$-search agent [14]. He used a concept referred to as "Tree of Chambers" for his evaluation function. The idea is to restrict the Player's Voronoi region, if it contains articulation points. The position shown in Figure 5 has articulation points. As discussed, the Voronoi heuristic declares Player 2 as the winner. According to the "Tree of Chambers" heuristic, Player 2 loses the game. It assesses the Player's space as follows. Starting at the current Player's position the evaluation function recursively explores the squares adjacent to the Player's position, not iterating over articulation points. This explored space is the Player's chamber. It is bordered by walls and articulation points. Additionally, adjacent articulation points are saved in an array. Once the Player's chamber is defined, adjacent chambers are explored using the stored articulation points as initial squares. If such a chamber does not contain the battlefront it is counted as a neighbouring chamber. Otherwise, if the adjacent chamber contains the battlefront, it is discarded. Eventually, when all neighbouring chambers have been found and saved, the value is the size of the largest neighbouring chamber added to the player's chamber. In the case where the Player's Voronoi region does not contain articulation points, the value simply is the size of the player's Voronoi region (Player 1 in Figure 5). In any case the resulting value is the assessed space the Player is able to fill. As in the Voronoi heuristic (Subsection 4.2), this value is calculated for both players. The surplus of the player's value compared to the opponent's then is the "Tree of Chambers"-value of the evaluated position. Returning to Figure 5, Player 2's chamber is in the upper right corner. This chamber has only one adjacent chamber (bordered by 5 articulation points). This chamber is not added, because it contains the battlefront. Player 2's value, therefore, is the size of his chamber (35). Player 1's Voronoi region does not contain any articulation points. His value is the size of his Voronoi region (53). Comparing the two values gives a surplus of $53 - 35 = 18$ for Player 1. Thus, the "Tree of Chambers" heuristic assesses the position in Figure 5 as a win for Player 1, whereas his Voronoi region (53) clearly is smaller than his opponent's (55). The "Tree of Chambers" heuristic, as explained in this section, is also implemented and applied in an agent. It will further be referred to as the TC player. In fact, the TC Player combines the Voronoi- and the "Tree of Chambers" heuristic. In case there are no articulation points in the current position, the TC Player only uses the Voronoi heuristic, otherwise it looks for chambers. Note that the described "Tree of Chambers" heuristic fails when there are many chambers on a board. In Figure 6 there are 19 articulation points and 8 chambers. Every move might add or remove chambers and articulation points. The TC Player would only look for the chambers adjacent to his own. The calculated available space would vary from his real space and the assessment of the "Tree of Chambers" heuristic is not correct. Fortunately, this scenario does not occur often. Most of the time there are not more than three chambers and the "Tree of Chambers" heuristic accurately assesses the player's space. Sometimes, however, the heuristic fails.

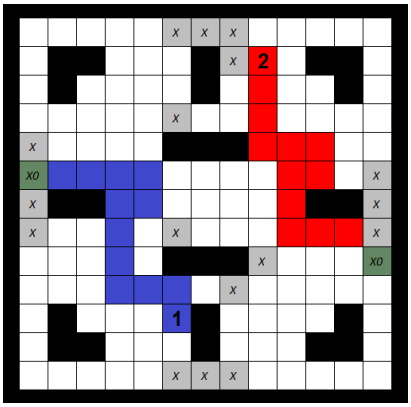

Figure 6: A Tron game after 11 moves. Articulation points are indicated by an X. Squares which belong to the battlefront contain a 0. "XO" indicates an articulation point which belongs to the battlefront.

5 Application to MCTS

Originally MCTS uses Play-Outs to assess the value of a game position. In Tron it is expected that the above mentioned evaluation functions give a good approximation of the real value of the explained types of game positions (see Subsections 4.2 and 4.3). In fact, in some cases these evaluation functions might be able to be more accurate than numerous Play-Outs. There are multiple possibilities implementing an MCTS agent with an evaluation function. One is to completely replace the Play-Out by an evaluation function [6]. Another possibility is to stop the Play-Out after a number of moves and assess the current node using the evaluation function [10, 17]. Both the Voronoi Player and the TC Player are extensions of the existing UCT agent developed by Den Teuling in 2011 [5]. His agent uses the MCTS-Solver enhancement, which is able to prove the game-theoretic value of a game position without excessive simulations [17]. Moreover it applies a so-called Predictive Expansion Strategy [5]. If the evaluator can certainly assess the future winner of a position during the Expansion phase the containing node is treated as a terminal node and no game is simulated. Additionally it stops the Play-Out

every 5 moves (Play-Out Cut-off) and tries to evaluate the current position. This, however, only works if the two players are separated from each other. The Play-Out keeps continuing for 5 moves until it encounters a position where the two players are separated.

For the new heuristics it has to be specified how high the surplus of space of a player needs to be in the current position in order to be assessed as a win for this player (Subsection 5.1). Additionally it has to be defined where the Play-Out is stopped (Subsection 5.2) in order to play Tron most optimally.

5.1 Determining a winner

A crucial part of the heuristic evaluation functions is the amount of space surplus the player must have in a position to be assessed as a win for this player. This value will be referred to as the Win-threshold. It is not clear that the player having only a slight surplus of space will ultimately win the game. In Figure 5 Player 2 has a larger Voronoi region than his opponent. However, the surplus of space is only 2 and his opponent is going to win the game. The Voronoi heuristic with Win-threshold 1 determines the wrong player to be winning. Tuning this value could improve the Voronoi heuristic. If that value would be 3 in the case of the scenario explained above, the Voronoi heuristic would not declare a winner and thereby avoid a mistake. In the case, where one of the new evaluation functions is not able to determine a winner, the Play-Out is continued for five more moves and another assessment takes place. One of the experiments described in Section 6 is to tune the surplus of space a player needs to have to be assessed as the winner, the Win-threshold.

5.2 Cut-off point

The point where to first stop a simulated game and evaluate the current position is crucial for MCTS. As described in Section 4, both new evaluators might give false assessments depending on the current position. The point where the simulated game is first stopped and a position is evaluated will be referred to as the Cut-off point. Once the Cut-off point is reached, the Play-Out is stopped every 5 moves and the current node is evaluated. This makes sure that the position which is evaluated is not at the very beginning of a game, but a few moves inside of it. The Cut-off point has to be tuned in such a way that the evaluation functions perform optimal. In Section 6 experiments which aim to find the most optimal Cut-off point are presented and Subsection 6.1 shows the results of all parameter tuning experiments.

6 Experiments

For the experiments the Java Tron program of Den Teuling [5] is used. In the implementation all rules mentioned

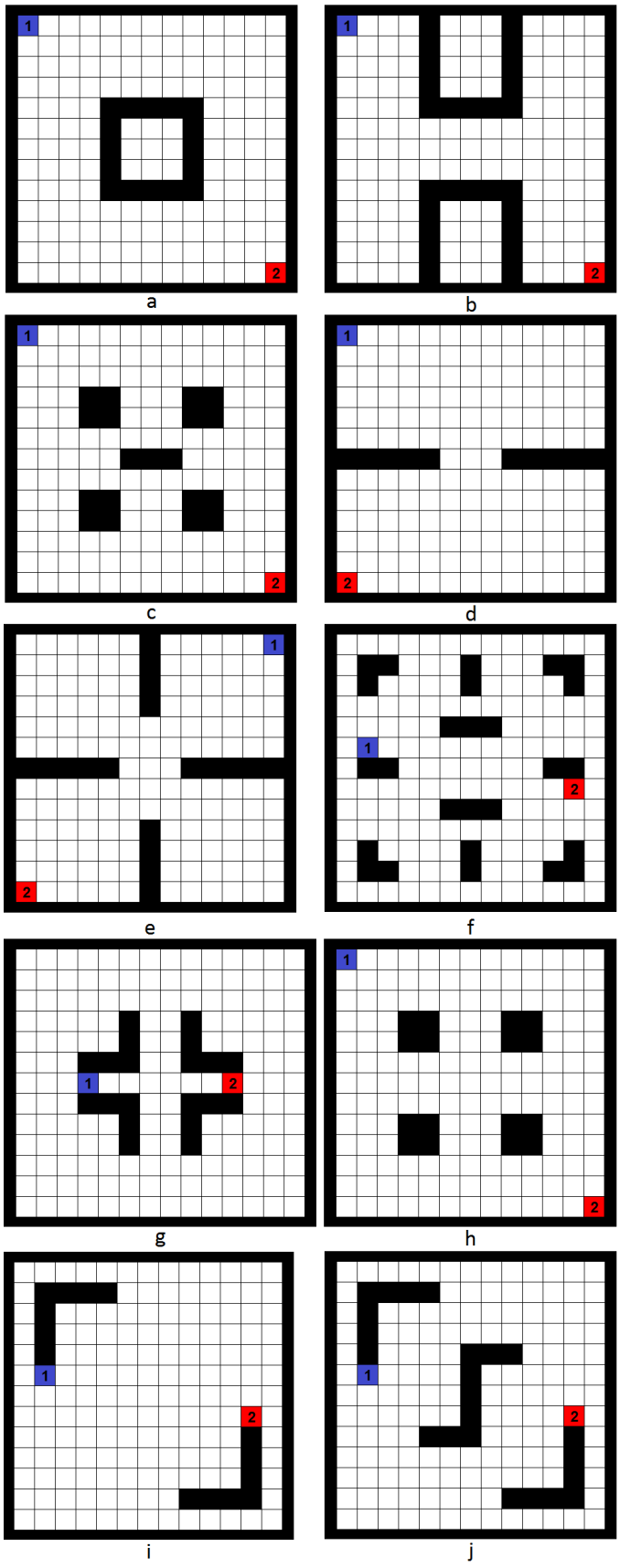

Figure 7: The 10 boards used in the experiments

in Section 2 are implemented. The used machine to run the experiments is an Opteron CPU with 2.4 GHz and 8 GB RAM. Two different types of experiments are conducted. The first is parameter tuning. Cut-off point, Win-threshold and the time setting are tuned. Parameter Tuning is performed with a UCT agent including the Play-Out Cut-off using the Voronoi evaluation against a simple UCT agent without enhancements. Second, both agents (Voronoi Player and TC Player; see Section 5) are tested to find their playing strength. In total 10 different symmetric boards are employed for the tests. They contain different kinds and numbers of obstacles and are chosen in order to find the overall playing strength of an agent. Figure 7 shows the boards used. To avoid a bias towards a specific starting position, the agents play with both colors. Both Players start once from both starting spots (Indicated by 1 and 2 in Figure 7). In that way both players have the same (dis)advantages.

The following experiments are conducted with a time setting of 5 seconds per move. The Cut-off point of the agent is tuned as follows. Four different types of a simple UCT agent are tested: Initially cutting off the Play-Out after 5, 10, 15 and 20 moves. Win-threshold is tested with values 1, 2, 3 and 4. Next, the time setting is tested with 5 values: 1, 2, 3, 4 and 5 seconds. Each parameter is tested against another simple UCT agent without Cut-off in 400 games spread over the 10 boards.

For both, the Predictive Expansion Strategy and the Play-Out Cut-off, the evaluator described in Subsection 4.1 is used in the UCT-Solver Player of Den Teuling [5]. The Voronoi-Solver Player and the TC-Solver player extend the UCT-Solver Player by additionally applying the heuristics explained in Subsections 4.2 and 4.3, respectively, for the Play-Out Cut-off. Both new players apply the Predictive Expansion Strategy in the same way as the UCT-Solver Player of Den Teuling [5]. The difference between the new players and the original one, hence, is only the heuristic applied in the Play-Out Cut-off. In fact, the new players use the Voronoi and the "Tree of Chambers" evaluation function, as long as the players are not separated from each other (see Subsection 4.2). When a position emerges where the players are separated the new players also use the evaluation function explained in Subsection 4.1. This is due to the fact that in case the two players are separated on the board, this evaluation function assesses positions more accurately than the two new ones do.

To find out about the playing strengths of the Voronoi Player and the TC Player, 80 games are played on each of the 10 maps (including the different starting positions). The opponent is the UCT-Solver player of Den Teuling [5]. In Subsection 6.2 the results are shown both for all boards and for each board separately.

Last, the Voronoi Player, the TC Player and the UCT-Solver Player are tested without a time limit but with a maximum of 10,000 Play-Outs per move against each other. This time the Voronoi Player and the TC Player play against each other as well. The results are presented in Subsection 6.2.

6.1 Parameter-tuning results

In this subsection the results of the parameter-tuning experiments are presented.

Win-threshold

The Win-threshold is the required surplus of space a player needs to have to be assessed as the future winner of the game. 400 games spread across the 10 maps (Figure 7) are played for each value. The value 15 is used for the Cut-off point and a time setting of 5 seconds per move is used. The result of the Win-threshold parameter (WT) tuning is shown in Table 1.

WT	Voronoi Cut	Confidence
1	62.88%	±4.73%
2	62.25%	±4.75%
3	54.50%	±4.88%
4	44.00%	±4.86%
	TC Cut	Confidence
1	60.63%	±4.79%
2	59.25%	±4.82%
3	57.90%	±4.84%
4	55.50%	±4.87%

Table 1: Win-threshold parameter tuning with a 0.95 confidence interval. Opponent is a simple UCT agent.

Both Players perform best with a Win-threshold of 1. This means that an assessed game position is considered as a win if a player has a surplus of 1 square in that position. The "Playing strength" experiments, hence, are conducted with a Win-threshold of 1 for both the Voronoi Solver Player and the TC Solver Player (results in Subsection 6.2).

Cut-off point

The Cut-off point is the point where a simulated game is allowed to be cut off for an evaluation. It is tuned by playing 400 games spread across the 10 boards for each tested value. The experiment is run with a Win-threshold of 1 and a time setting of 5 seconds per move. Table 2 shows the results of the Cut-off point (CP) parameter tuning.

The Voronoi Heuristic performs best when the game is played for 20 moves before first evaluating a game position. The TC Cut Players with a Cut-off point of 10, 15 and 20 score best. For the experiments conducted in Section 6.2 the Voronoi Solver Player uses a Cut-off point of 20 while the TC Solver Player uses one of 15. As

CP	Voronoi Cut	Confidence
5	58.50%	±4.83%
10	60.00%	±4.80%
15	59.50%	±4.81%
20	64.63%	±4.69%
	TC Cut	Confidence
5	55.50%	±4.87%
10	59.25%	±4.82%
15	60.25%	±4.80%
20	57.13%	±4.85%

Table 2: Cut-off point parameter tuning with a 0.95 confidence interval. Opponent is a simple UCT agent.

shown in Table 2 the TC Cut Player is approximately equally strong with Cut-off point values of 10, 15 and 20. A value of 15 is chosen to have a difference in the Cut-off point parameter between the two players.

Time setting
In this series of experiments the influence of the time setting is investigated by playing 400 games spread across all 10 boards. The time setting experiment used a Cut-off point of 15 and a Win-threshold of 1.

Time	Voronoi Cut	Confidence
1 sec	45.13%	±4.88%
2 sec	42.75%	±4.85%
3 sec	39.25%	±4.79%
4 sec	45.00%	±4.88%
5 sec	56.88%	±4.85%
	TC Cut	Confidence
1 sec	44.13%	±4.87%
2 sec	50.13%	±4.90%
3 sec	46.50%	±4.89%
4 sec	50.00%	±4.90%
5 sec	52.88%	±4.89%

Table 3: Time settings with a 0.95 confidence interval. Opponent is a simple UCT agent.

The results (Table 3) show that a time setting of 5 seconds outperforms the other tested values. Although, for both players, there is an outlier for 3 seconds, it is clear that 5 seconds to compute a move is best. All experiments that are conducted to assess the playing strength of the new players are using a time setting of 5 seconds per move. The more time per move the new players have, the better they perform. 10 or more seconds per move probably cause better performances, but also cause more time consuming experiments. A maximum of 5 seconds is chosen because of the time constraint of this paper. Note that the results (Table 3) are not in accordance with the other parameter-tuning experiments (Tables 1 and 2). This is because the machine used for the experiments was overloaded at the time and some experiments were thus conducted with different computing capabilities.

6.2 Playing Strength

This subsection presents the results of the conducted experiments to find out about the playing strength of the TC-Solver Player and the Voronoi-Solver Player. Both play 80 games on each of the 10 maps against the UCT-Solver Player using Play-Out Cut-off and Predictive Expansion Strategy [5].

TC-Solver vs. UCT-Solver
Table 4 shows the results of the 800 games played by the TC-Solver Player against the UCT-Solver player [5].

Board	TC-Solver	UCT-Solver	Confidence
a	60.00%	40.00%	±10.74%
b	20.00%	80.00%	±8.77%
c	24.38%	75.62%	±9.41%
d	3.75%	96.25%	±4.16%
e	33.75%	66.25%	±10.36%
f	58.12%	41.88%	±10.81%
g	73.75%	26.25%	±9.64%
h	23.75%	76.25%	±9.33%
i	89.38%	10.62%	±6.75%
j	55.00%	45.00%	±10.90%
Total	44.19%	55.81%	±3.44%

Table 4: Results of TC-Solver vs. UCT-Solver with a 0.95 confidence interval.

Although the TC-Solver Player achieves a better win-ratio than his opponent on 5 of the 10 boards, in total he only wins 44.19 % of the games. Note that the TC-Solver Player is only able to simulate 8,000 games on average per second, while the UCT-Solver Player simulates 36,000 on average. The 5 boards on which the TC-Solver Player is able to win more games than his opponent are entirely different (boards a, f, g, i and j in Figure 7). Therefore it is difficult to tell why he actually plays stronger on these and weaker on the others.

Voronoi-Solver vs. UCT-Solver
Table 5 shows the results of the 800 games played by the Voronoi-Solver Player against the UCT-Solver Player [5].

The Voronoi-Solver Player wins on 4 of the 10 boards. But on 3 of these 4 boards, he only achieves a win-ratio of below 60%. The Voronoi-Solver player simulates 15,000 games on average per second. This is a bit more than the TC Player is capable of, but still clearly lower than the UCT-Solver Player (36,000). In total the Voronoi Player wins 34.75% of the 800 games played. Similar as

Board	Voronoi-Solver	UCT-Solver	Confidence
a	56.88%	43.12%	±10.85%
b	13.12%	86.88%	±7.40%
c	25.00%	75.00%	±9.49%
d	3.75%	96.25%	±4.16%
e	21.88%	78.12%	±9.06%
f	56.25%	43.75%	±10.87%
g	53.75%	46.25%	±10.93%
h	30.00%	70.00%	±10.04%
i	89.38%	10.62%	±6.75%
j	25.00%	75.00%	±9.49%
Total	34.75%	65.25%	±3.3 %

Table 5: Results of Voronoi-Solver vs. UCT-Solver with a 0.95 confidence interval.

the TC-Player the Voronoi Player plays stronger than his opponent on 4 maps with major differences (boards a, f, g and i in Figure 7).

Play-Out limit

Table 6 shows the results for the games of the UCT-Solver Player, the Voronoi-Solver Player and the TC-Solver Player against each other. In all games the players applied a fixed number of 10,000 Play-Outs.

Play-Out limit	UCT-Solver	Voronoi-Solver	TC-Solver
UCT-Solver		$45.50 \pm 3.45\%$	$49.19 \pm 3.46\%$
Voronoi-Solver	$54.50 \pm 3.45\%$		$44.19 \pm 3.44\%$
TC-Solver	$50.81 \pm 3.46\%$	$55.81 \pm 3.44\%$	

Table 6: Result for the experiments with 10,000 Play-Outs.

Both new players, the Voronoi-Solver and the TC-Solver, beat the UCT-Solver. Moreover, the TC-Solver is able to win 55.81% of the games against the Voronoi-Solver. As the Cut-off point tuning experiments (Subsection 6.1) this series was conducted while the machine was under heavy load. In this case this does not influence the number of computations per move. It was observed that the experiments, especially including the TC-Solver, logged exceptions. Ususally an exception in a game causes the program to repeat this game. During this series of experiments, however, the program would occasionally count this game as a win for the player that finished his computations first. Hence, in some of the games, the slower player is disadvantaged. This explains why the TC-Solver Player clearly wins against the Voronoi-Solver Player, but does not achieve an as good win-ratio against the UCT-Solver Player.

7 Conclusion and Future Research

The experiments (Subsection 6.1 and 6.2) show, on the one hand, that unfortunately both new players are incapable of winning more than 50% of the games on average against the UCT-Solver Player of Den Teuling [5] with a time setting of 5 seconds per move. On the other hand, both new players achieved considerably good scores (Tables 4 and 5) on board i (Figure 7). Furthermore the TC-Solver Player is able to clearly beat the UCT-Solver Player on at least two more boards (boards a and g in Figure 7). Although, in the end the new Players lost (Tables 4 and 5), they are able to improve the performance of the existing Player in certain situations. One reason for the loss of the new players definitely is the computational overhead of the applied heuristics. As stated in Subsection 6.2 the UCT-Solver Player using the Predictive Expansion Strategy and the Play-Out Cut-off enhancement runs 36,000 Play-Outs per second on average. Both new players run substantially fewer Play-Outs. The Voronoi-Solver Player is able to make 15,000 Play-Outs per second. The TC-Solver Player only runs about 8,000 Play-Outs on average per second. Restricting not the time but the number of Play-Outs per move clearly increases the performance of both new players. The results presented in Table 6, however, do not represent the real playing strengths of the new Players (see Subsection 6.2). They are an indicator that the new heuristics accurately assess Tron positions.

Additional to the experiments from Subsection 6.2, the parameter-tuning experiments (Subsection 6.1) show that simple UCT agents with the new heuristics applied in the Play-Out Cut-off win against UCT agents without enhancements. This confirms the ability of both heuristics to accurately assess positions in Tron.

In the future it could be investigated in which exact situations the new agents outperform the existing agents. Then the players could be combined according to the obtained information. The TC heuristic seems to be performing quite well in certain situations. An agent which computes the "Tree of Chambers" heuristic only in such cases would most likely play stronger than all existing agents. Another part where there is room for improvement is diminishing the computational overhead of the "Tree of Chambers" heuristic. Improving upon that would increase the number of games the agent is able to simulate per second. Being able to simulate more games always means that the agent is capable of more thoroughly searching the state space. This, in the end, leads to a better playing Tron agent.

References

[1] Aurenhammer, F. (1991). Voronoi Diagrams - A survey of a fundamental geometric data structure. *ACM Computing Surveys*, Vol. 23, No. 3, pp. 345–405.

[2] Browne, C., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of Monte Carlo Tree Search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 4, No. 1, pp. 1–43.

[3] Chaslot, G.M.J-B., Winands, M.H.M., Uiterwijk, J.W.H.M., Herik, H.J. van den, and Bouzy, B. (2008). Progressive strategies for Monte-Carlo Tree Search. *New Mathematics and Natural Computation*, Vol. 4, No. 3, pp. 343–357.

[4] Coulom, R. (2007). Efficient selectivity and backup operators in Monte-Carlo Tree Search. *Computers and Games (CG 2006)* (eds. H.J. van den Herik, P. Ciancarini, and H.H.L.M. Donkers), Vol. 4630 of *Lecture Notes in Computer Science (LNCS)*, pp. 72–83, Springer-Verlag, Berlin Heidelberg, Germany.

[5] Den Teuling, N.G.P. (2011). Monte-Carlo Tree Search for the simultaneous move game Tron. *Maastricht University, The Netherlands, B.Sc thesis*.

[6] Kloetzer, J. (2010). *Monte-Carlo techniques: Applications to the game of the Amazons*. Ph.D. thesis, Japan Advanced Institute of Science and Technology, Nomi, Ishikawa, Japan.

[7] Knuth, D.E. and Moore, R.W. (1975). An analysis of Alpha-Beta pruning. *Artificial Intelligence*, Vol. 6, No. 4, pp. 293–326.

[8] Kocsis, L. and Szepesvári, C. (2006). Bandit based Monte-Carlo planning. *Machine Learning: ECML 2006* (eds. J. Fürnkranz, T. Scheffer, and M. Spiliopoulou), Vol. 4212 of *Lecture Notes in Artificial Intelligence*, pp. 282–293.

[9] Lee, C.-S., Müller, M., and Teytaud, O. (2010). Special issue on Monte-Carlo techniques and computer Go. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 2, No. 4, pp. 225–228.

[10] Lorentz, R.J. (2008). Amazons discover Monte-Carlo. *Computers and Games (CG 2008)* (eds. H.J. van den Herik, X. Xu, Z. Ma, and M.H.M. Winands), Vol. 5131 of *Lecture Notes in Computer Science (LNCS)*, pp. 13–24, Springer, Berlin Heidelberg, Germany.

[11] Miltzow, T. (2011). Tron, a combinatorial game on abstract graphs. *Arxiv preprint arXiv:1110.3211*.

[12] Rimmel, A., Teytaud, O., Lee, C.-S., Yen, S.-J., Wang, M.-H., and Tsai, S.-R. (2010). Current frontiers in computer Go. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 2, No. 4, pp. 229–238.

[13] Samothrakis, S., Robles, D., and Lucas, S.M. (2010). A UCT agent for Tron: Initial investigations. *Proceedings of IEEE Conference Computational Intelligence and Games*, pp. 365–371.

[14] Sloane, A. (2010). Google AI challenge post-mortem. `http://a1k0n.net/2010/03/04/google-ai-postmortem.html`. Accessed March 29, 2012.

[15] Neumann, J. von (1928). Zur Theorie der Gesellschaftspiele. *Mathematische Annalen*, Vol. 100, pp. 295–320. In German.

[16] Voronoï, G. (1907). Nouvelles applications des paramètres continus à la théorie des formes quadratiques. *Journal für die Reine und Angewandte Mathematik*, Vol. 133, pp. 97–178. In French.

[17] Winands, M.H.M., Björnsson, Y., and Saito, J-T. (2008). Monte-Carlo Tree Search Solver. *Computers and Games (CG 2008)* (eds. H.J. van den Herik, X. Xu, Z. Ma, and M.H.M. Winands), Vol. 5131 of *Lecture Notes in Computer Science (LNCS)*, pp. 25–36, Springer, Berlin Heidelberg, Germany.