Universiteit Leiden
The Netherlands

Opleiding Informatica

Using Monte Carlo Tree Search to Play Cops & Robbers

Florian Varkevisser

Supervisors:
W.A. Kosters & M.J.H. van den Bergh

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

June 9, 2019

Contents

1. Introduction

Over the last few years, Monte Carlo Tree Search (MCTS) has been successfully used to create Artificial Intelligence (AI) agents for a variety of games. The most prominent example of MCTS being used is in Google's AlphaGo and AlphaZero, where it was used in conjunction with Deep Learning [1]. Go is a perfect information game, but MCTS has also successfully been used to play imperfect information games, for example Poker [2]. MCTS has also been used in some video games to create the AI agents [3].

The game of Cops & Robbers is a general hide-and-seek game, of which Scotland Yard is a specific variant. In the variant of Cops & Robbers that we will study the cops try to find a robber, who is hidden for a certain amount of turns. The specific rules will be explained in Chapter 2. In Chapter 3, we will discuss related research. In Chapter 4, we will discuss in-depth the strategies used for the various AI agents. In Chapter 5, we will present the results of our experiments. Chapter 6 contains the conclusion and suggestions for future research. Cops & Robbers is an asymmetric game, where the robber has perfect information and the cops have imperfect information. It is interesting to see if MCTS can be used to create AI agents for this game, both for the cops and for the robber(s).

The goal of this project is to create an agent for the robber and an agent for the cops using MCTS. The difficulty lies in some players having perfect information, the robber, and some having imperfect information, the cops. Based on experiments it seems that a rule based algorithm plays the game better than the MCTS agents. This project was done as a bachelor thesis at Leiden Institute of Advanced Computer Science (LIACS) at Leiden University and was supervised by W.A. Kosters and M.J.H. van den Bergh.

2. The game

In this chapter, we describe the game rules and other important information relating to the way the game is played.

2.1 Scotland Yard

Scotland Yard is played on a board with 199 stations in London. These stations are connected by one or more means of transport, where each has its own ticket type. These means of transport are the taxi, bus and underground. The robber, called "Mr. X", also has two additional types of tickets. One of those is a double move ticket, which allows Mr. X to move twice in one turn. The other type of ticket is the black ticket, which works as any other type of ticket and has the benefit of hiding the type of transport for the cops. The black ticket is also the only ticket that can be used to travel with the river boat.

Figure 2.1: Scotland Yard game board and travel log with tickets, Ravensburger [4]

To give some information to the cops, Mr. X places his/her[1] played tickets onto a travel log where everyone can see them. Additionally, on the 3rd, 8th, 13th and 18th move, Mr. X has to show himself on the board. The following turn, Mr. X removes himself from the board again. At the start of the game each cop gets 10 taxi tickets, 8 bus ticket and 4 underground tickets. Mr. X gets 4 taxi tickets, 3 bus tickets, 3 underground tickets, 2 double move tickets and 1 black ticket for each cop in the game. When a cop moves, she has to give her ticket to Mr. X. This causes Mr. X to have almost an infinite supply of cards. If a cop cannot use any tickets, she becomes stuck at her spot for the rest of the game. When taking turns Mr. X moves first and then the cops move in clockwise order. The cops win the game if one of them shares a station with Mr. X. Mr. X wins when no cop can move or when the last sport in his travel log is occupied by a ticket[5].

2.2 Abstract rules

In our version of Cops & Robbers, there is only one type of connection, which means that there are no transport cards in play and thus also no black cards. The robber does not have the ability to move more than once per turn. Furthermore the game lasts t rounds. We will use $t = 24$ for this project. The robber is visible every R_v moves and we will use $R_v = 4$ for this project. With $R_v = 4$ and $t = 24$ this results in the robber being visible in turn 4, 8, 12, 16, 20 and 24. The amount of cops playing is denoted by C and the amount of robbers by R; we will use $C = 3$ and $R = 1$ in this project.
The game board consists of an undirected connected graph $G = (V, E)$, where V is the set containing all *vertices* or *positions* and E is the set containing all *edges*. Note that, if the graph was not connected, only the part of the graph where the robber is present would be of interest. A move consists of a cop or robber moving along one edge connected to the current position. The set $V_c \subseteq V$ denotes the set containing all positions of the cops and $|V_c| = C = 3$ during the whole game, which means that the cops can never share a location. The set V_r is the set containing the position of the robber and $|V_r| = R = 1$. In this project $|V| \geq |V_c| + |V_r|$, which ensures that each cop and each robber can and will start on a separate node. At the start of the game, we have $V_c \cap V_r = \emptyset$, and the game ends when $V_c \cap V_r \neq \emptyset$ or the t-th move is made by the cops. At the start of the game, the starting position is randomly chosen. Every turn, each cop and each robber has to move to a neighbouring node. Each turn, the robber moves first and then the cops. All cops move at the same time, which means V_c except the current node of a cop is a valid target for each cop.

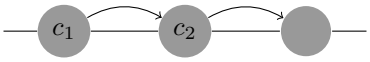

Figure 2.2: Two cops moving to a new position.

2.2.1 Special situations

Because all the cops move at the same time, there are some special situations regarding the possible moves. We will discuss these situations in the following subsections.

[1] In the rest of the thesis we will be using him for the robber and her for the cops to have a clear distinction between the two.

Because the cops all move at the same time, it is possible for them to target the old position of another cop. If two cops chose the position of each other, we say they swap places. Since all cops are identical to each other, this has the same effect as both cops staying on their place. An example of this is given in Figure 2.3. The nodes in the example can be connected to any amount of other nodes.

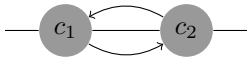

Figure 2.3: Swapping of two cops.

Even though all the cops move at the same time, there are still situations where cops can block each other from moving and thus halting the game. Figure 2.4 shows the simplest situation where cops block each other's movement. Since the cops cannot stay in their position, both cops only have the same node as a possible target. Because of this, the current game situation is unplayable. This can only happen in the starting situation, since otherwise there is a move possible, namely all the cops moving back to their previous positions.

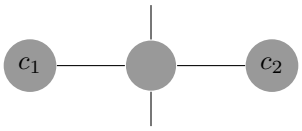

Figure 2.4: Simplest situation of cops blocking each other.

Because the middle node cannot be reached in a legal move, any type and size subgraph may be connected to that node. For subgraphs connecting to the nodes of the cops, some constraints are in place. The most important constraint is that the connecting node of the subgraph may not be a valid target for either cop. As soon as one of the cops in Figure 2.4 can move to another node, the situation becomes playable.

2.3 Graph types

As game board, we will use several different types of graphs. This way we can see if and how the graph affects the effectiveness of the different AI strategies. The various graphs will be discussed below.

2.3.1 Ladder graphs

One type of graphs that will be used is the *ladder graph*. This graph consists of two parallel lines of nodes that are connected at certain intervals. Each ladder graph we use has two properties: length ℓ and spacing s. The length denotes the amount of nodes that are present in each of the parallel lines. The spacing is the amount of nodes in between each connection, so, let say if $s = 0$ then every node of one line is connected to the corresponding node in the other line. With $s = 1$, every other node is connected (see Figure 2.5 for an example); with $s = 2$ there are two free nodes in between and so forth.

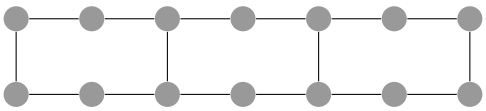

Figure 2.5: Ladder graph with $\ell = 7$ and $s = 1$.

Van den Bergh has proven [6] that it is possible for three cops to always catch a robber on these types of graphs given perfect information. It will be interesting to see how the different strategies and the imperfect information affect the chances of the cops catching the robber.

2.3.2 Circle graphs

When drawn, the *circle graph* contains a ring of c nodes, where each node is connected to two neighbours which form the ring when drawn. Each node is also connected with the nodes resulting from skipping every s nodes clockwise, with a given s such that $s + 1$ is a factor of c; There are s+1 inner rings formed. See Figure 2.6 and Figure 2.7 for some examples.

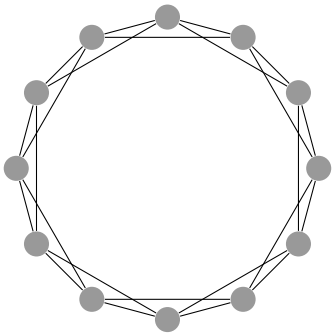

Figure 2.6: Circle graph with $c = 12$ and $s = 1$.

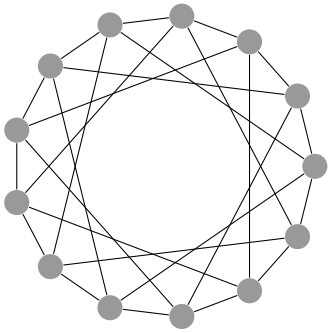

Figure 2.7: Circle graph with $c = 13$ and $s = 3$.

2.3.3 Modified Scotland Yard board

Another graph we will use is the graph representing a modified version of the game board of Scotland Yard. Since the graph G is an unweighted graph, we only have one type of connection. This means the graph contains all edges for each means of transport. It also might be interesting to vary the graph by excluding the edges of certain means of transport, creating different graphs. For this project, we will use the graph containing all means of transport except the ferry. The graph of this board can be seen in Figure 2.8.

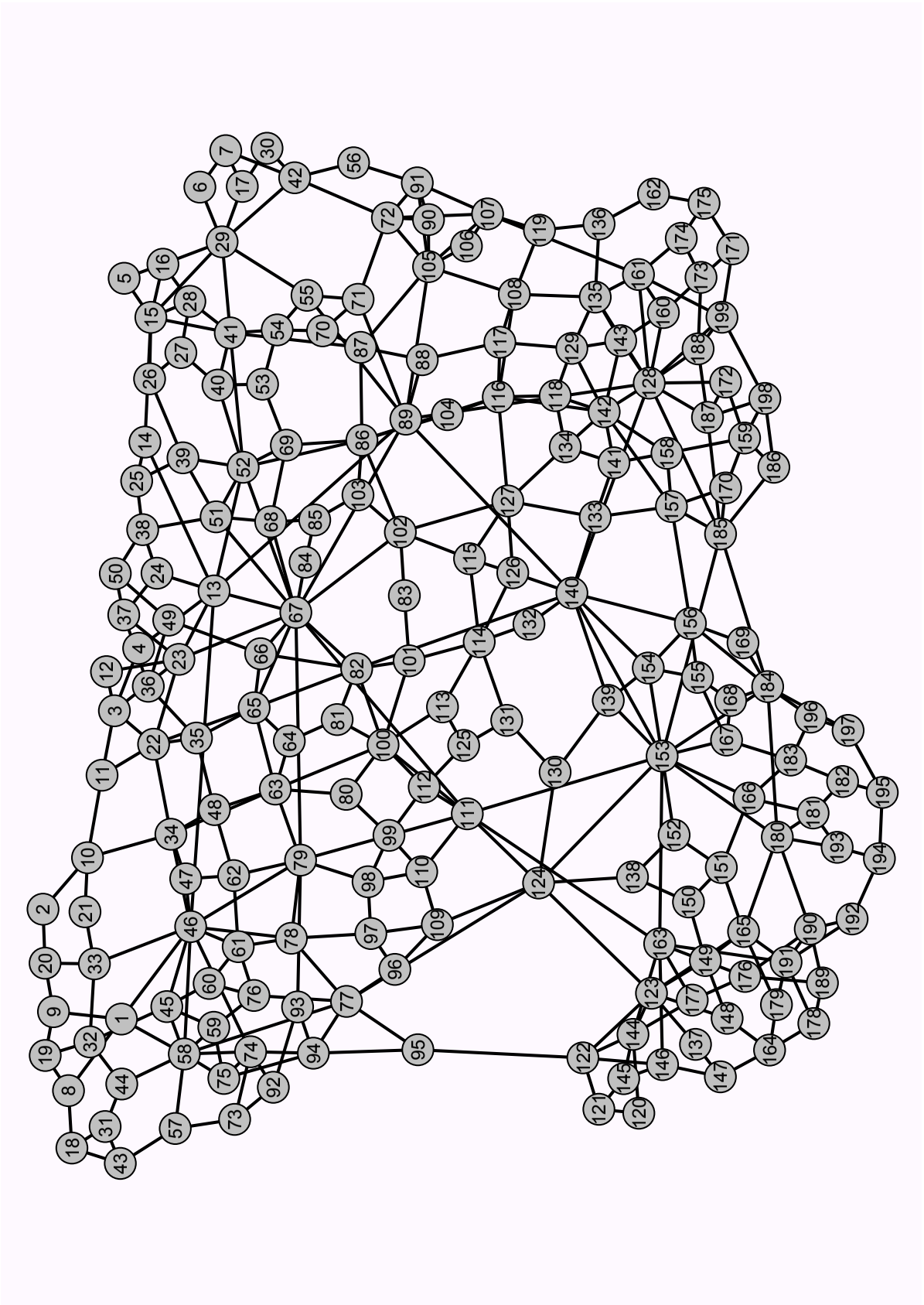

Figure 2.8: Graph representation of the modified Scotland Yard board, made using [7].

3. Related work

In this chapter, we discuss some work that is used to create AI agents for Cops & Robbers.

In 2012, Browne et al. present a survey of different Monte Carlo Tree Search (MCTS) methods [8], which shows the state of the research into MCTS methods at the time. The survey contains a brief summary of the background of MCTS research. Furthermore, the core algorithm of MCTS is explained together with multiple enhancements and variations of MCTS. The authors also show multiple applications of MCTS such as real-time games, non-deterministic games and various non-game applications. This survey will in the sequel function as the basis for creating the MCTS agents for the cops and the robber.

In 2012, Nijssen and Winands apply MCTS to the board game Scotland Yard[9]. They use Upper Confidence Bounds for Trees (UCT), enhanced with what they call progressive history as their selection policy. They use several techniques to implement domain-specific knowledge into the algorithm. One of these techniques is ε-greedy playouts. This means that, in the selection phase of the MCTS algorithm, a random move is chosen with a chance of ε. When there is not a random move chosen, a heuristic is applied for choosing the best move. For the robber, this means maximizing the number of moves the closest cop needs to do to catch the robber. For the cops, this could be one of two heuristics: they can minimize the sum of the distance to all possible locations, or the cops chase the location they assume the robber is on. To predict the location of the robber more accurately, *determinization* and *location categorization* is used. For the location categorization there are three different types explored: minimum-distance, average distance and station type. With minumum distance, the locations are categorized by the distance to the nearest cop. With average distance, the locations are categorized by the average distance to all the cops. With station type, the locations are categorized by station type. The last category is not applicable to our case, since we only have one station type. Both these strategies might also improve the quality of our MCTS algorithm.

In his master thesis [6], Van den Bergh provides a way to calculate the amount of cops needed to win a game of Cops & Robbers with perfect information. To calculate this number, Van den Bergh assumes the cops are using a guarding strategy, which is proved to be winning for certain types of graphs. It would be interesting to see if this concept of guarding can be applied to a version of Cops & Robbers with imperfect information.

4. Algorithms

In this chapter, we will discuss the specifics of the implementation. We will talk about the algorithms used and the implementation of the various AI strategies that are used. To get a better understanding of these strategies a few important concepts will also be explained.

4.1 Simulation software

The straightforward game engine we use for this project was written from scratch in C++. It is responsible for keeping track of the state of the game, checking for the win conditions and executing the various AI algorithms. At the start of the game, the engine is also responsible for determining the starting positions of the players. The program takes a space separated adjacency matrix (example in Figure 4.1), the size of the graph, the number of cops and the amount of rounds to be played as input. The adjacency matrix is stored in a two-dimensional array. When using MCTS based agents, the amount of play-outs that is to be used is also specified as an input. The software is also responsible for multiple play-outs and reporting back the results of the experiments. This will be explained in more detail in Chapter 5.

```
  1 2 3 4
1 0 1 0 1
2 1 0 1 1
3 0 1 0 0
4 1 1 0 0
```

Figure 4.1: Input file for graph G_e from Figure 4.2.

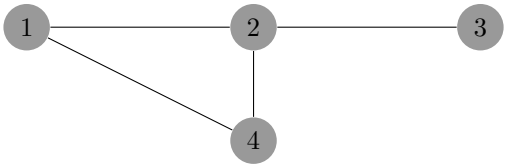

Figure 4.2: Example graph G_e.

4.2 Information set

We want to create an AI agent for the cops, that has to deal with imperfect information. It is important to understand what information the cops have and how their decisions are affected by having imperfect information. At any time, all players know the positions of all cops and the last position where the robber was visible. Since the robber always knows his own position, he has access to perfect information throughout the game. The cops only have access to perfect information on turns where the robber is visible, since the last known location of the robber is also the current location. When creating an AI agent, we have to make sure that the agent can deal with both situations. In any given turn, the possible locations for the robber are denoted by the set $V_p \subseteq V$. In turns where the robber is visible, V_p is equal to V_r and $|V_p| = 1$. In the turn following the one where the robber

was visible, the set V_p is equal to the neighbouring positions of V_p minus V_c and V_r. This is the case, because the game would be over if the robber is on a node in V_c, and cannot be on V_r since he has to move. Each turn, the neighbours of the previous V_p constitute the new set V_p and V_c is excluded from that for the same reason as before. The maximum size of V_p is dependent on the graph used and the amount of turns the robber is hidden. With each additional turn the robber is hidden, the set usually grows, because for each turn, all the neighbours of the set are added. The graph used influences the amount of positions that are added each turn to V_p. The size of V_p can never be larger than $|V| - C$, since the robber cannot be on a position with a cop, because otherwise the game would be over and the position of the robber would be known.

We provide an example of the growth of the information set. For this example, we will assume that the robber is only visible in the first turn. Given the graph in Figure 4.3 is the game board and given the current position of the robber, being node 5, $V_p = \{5\}$. In the next turn the robber is hidden and $V_p = \{3, 4, 6, 7\}$: V_p contains all nodes connected to node 5. The node 5 is excluded since the robber had to move away from that node. In the following turn $V_p = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. In this turn the nodes that were in the previous set V_p are in the set again, because they were reachable from another node in the set. For example, node 4 was reachable from both node 3 and node 6. As can be seen, depending on the type of graph, the size of set V_p can quickly increase.

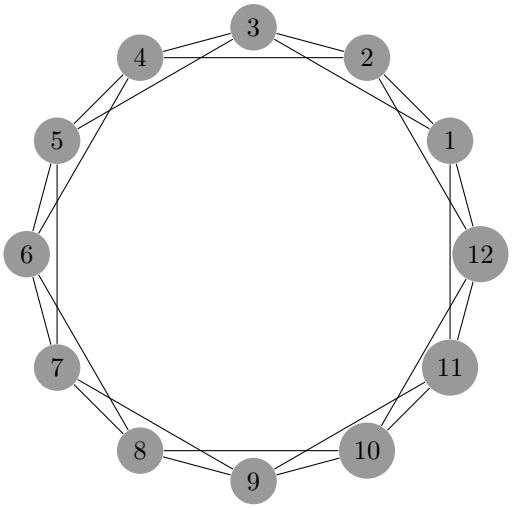

Figure 4.3: Circle graph with $c = 12$ and $s = 1$.

4.3 General movement of the cops

All cops move at the same time. To decide which node each cop will move to, we have to determine the new position of each cop using an AI agent. To reduce the time cops block each other, the cop with the fewest possible moves chooses a new node first using the current strategy. Then we determine for the other cops which one has the fewest moves and she then chooses a target node. We continue with this until all the cops have chosen a target node and then they all move at the same time. When a cop has chosen a target node, that node gets reserved and other cops are not allowed to choose that node as a target. If a cop cannot move because of a reservation of a previous cop, the cop that chose a target last, chooses a new node. If none of the other choices result in a possible move for the next cop, the cop before the current cop choose a new node.

4.4 Random player

The simplest agent we use is the random player. This agent is the same for both the cops and for the robber. The first step is determining which nodes are a valid target to move to; for the cops, the nodes reserved by other cops have to be excluded from the possible target nodes. For the robber, no nodes are excluded and the set of possible moves is identical to the set of connected nodes. We then randomly choose a node from that set as a move. Because the choice is completely random, the robber could end up moving onto a cop and thus losing the game.

4.5 Chasing algorithm for the cops

As an improvement to the random moves, we incorporate some information in the decision making. The cops will now move towards the last known location of the robber, until they are within n moves of that position. To determine the path, a cop will take breadth-first-search is used. While the cops are within n moves of the last known position, they will start to move randomly until the robber is visible again or they move further than n nodes away. We think that incorporating the last location of the robber will improve the chance of finding the robber with random moves. It will be important to find a value of n that directs the cops towards the robber, but also takes into account the movement of the robber. It might be interesting to see what the effect will be if n is determined by the distance to the last known position of the robber.

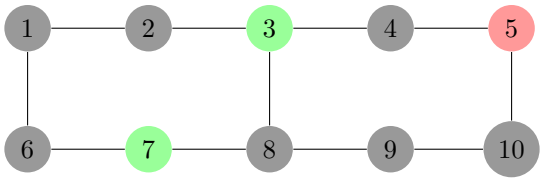

Figure 4.4: Ladder graph with $\ell = 5$ and $s = 1$ and the positions of the players.

-

To clarify, we work through an example situation with two cops on a ladder graph with $\ell = 5$ and $s = 1$. The graph and the positions of the cops are given in Figure 4.4 with the cops being the green nodes and the last visible position of the robber being red. For this example, we will use a range $n = 2$. First, we need to determine the move order of the cops. We will use the method described in Section 4.3. In this case, this results in the cop on node 7, which we will call C_7 for convenience, moving first, followed by the cop on node 3, called C_3. Cop C_7 only has two possible moves, node 6 and node 8, while C_3 has three moves, node 2, node 4 and node 8.
We then determine a, shortest path from node 7 to node 5, the last known position of the robber. In this case there are two shortest paths from node 7 to node 5 with length 4, namely $8 \to 3 \to 4 \to 5$ and $8 \to 9 \to 10 \to 5$. Both paths are longer than search range 2, so we randomly follow on of the found paths. Since both paths have node 8 as the first move, C_7 reserves node 8 for her move.
Now, we determine the move for C_3. Since C_7 reserved node 8 for her move, we exclude that node from the breadth-first-search. For C_3, there is only one shortest path with length 2, namely $4 \to 5$. Since the path length is within the search range, we choose a move for C_3 using the random agent, for example node 2. The state resulting from the moves of the cops is given in Figure 4.5.

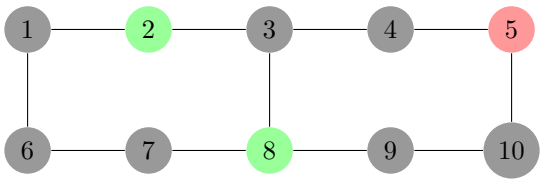

Figure 4.5: The situation resulting from the cops in Figure 4.4 moving.

4.6 Avoiding algorithm for the robber

The avoiding strategy for the robber uses the location of the cops to decide the best node to move to. For each possible move without a cop, we determine the total distance to all the cops. To determine the distances, breadth-first-search is used. A move with the highest total distance to the cops is chosen. If there is a tie, then we choose the node with the most neighbors. If there is still a tie, we randomly choose a move from these candidates. We go through an example with the same situation as in the example of Section 4.5, which is shown in Figure 4.4. The robber, denoted in red, has two possible moves, node 4 and node 10. We determine the total distance to all cops for each move. This results in a total distance of 4 for node 4, because the cop on node 3 is one move away and the cop on node 7 is three moves away. The move to node 10, has a distance of 6, since there are 3 three moves needed for either cop to reach the position of the robber. The robber will move to node 10 since that node has a higher total distance.

4.7 MCTS agent

MCTS is a tree search based algorithm [8], which means that the algorithm gradually builds a search tree with the root representing the current game state. For MCTS, this is done until some computational limit is reached, in our case this is a certain amount of play-outs. After the limit is reached, the algorithm returns the most promising child of the root. In our case, this is the child with the highest reward.

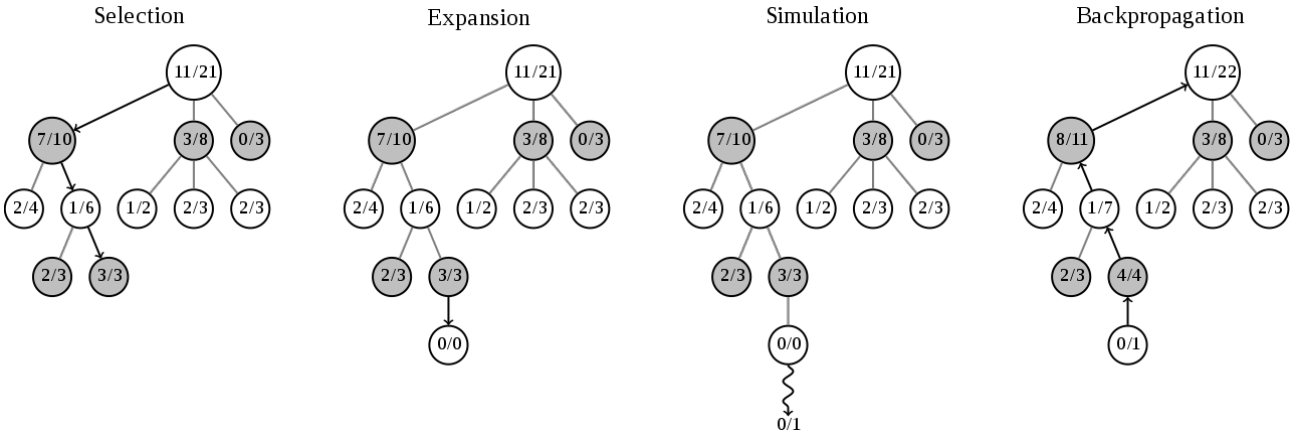

Figure 4.6: The different phases of MCTS [8].

The algorithm builds the search tree by repeating the four steps shown in Figure 4.6:

1. *Selection*: The algorithm recursively selects the best child node, until a non-terminal expandable node is found. A node is expandable if there are unexplored child nodes.

2. *Expansion*: One of the unexplored children is added to the search tree.
3. *Simulation*: A simulation is performed starting at the expanded node. This can be a simple algorithm, for example random play-outs, or an advanced algorithm to mimic real players more closely, for example a neural network [10].
4. *Backpropagation*: The result of the simulation is used to update the statistics of the selected nodes by passing the result of the simulation up the tree, usually those in the path from the root to the newly expanded node.

For the selection phase, we use the Upper Confidence Bounds for Trees (UCT) method [11]. The formula used is:

$$Q_i + C_p\sqrt{\frac{2\ln n}{n_i}}.$$

Here, Q_i is the expected reward from child node i, for which we will use the current score of that node. An unvisited node has a Q_i of ∞, which is to ensure that unvisited nodes will always be explored. Here, n is the amount of times the current nodes has been visited and n_i is the amount of time child node i has been visited. Furthermore, C_p is an exploration constant, which is used to balance the exploitation of high scoring nodes and the exploration of less visited nodes. For C_p we will use $\frac{1}{\sqrt{2}}$ [8].
In the expansion phase, we simply create a child node for every valid move. In the simulation step we use the random agent described in a earlier section to simulate the moves of the players. The simulation results in either a win for the cops or for the robber. This result is backed up in the tree in the last step.
The robber always uses the current state of the board as the initial state. Since the cops don't always know the current position of the robber, they use the last visible location of the robber as the initial position.

5. Experiments

In this chapter, we discuss the experiments we performed and the reasoning behind them. Furthermore, we will present the results of those experiments. The experiments will be categorized based on the graph type used.

5.1 Introduction to the experiments

As described in Section 4.1, the program is responsible for both the play-outs of the games as well as reporting back the moves taken by the agents and the amount of games won by both groups. Since this is a binary statistic, we only report the winrate of the cops and this winrate is $\frac{\text{won games}}{\text{games}}$. To determine the win rate, we play out 1000 games. The win rate of the robber is equivalent to $100\% -$ winrate of the cops. We will use the experiments with a random agent for both players as a baseline to compare the other agents to. For the MCTS players we have to decide on an amount of playouts. With some empirical testing on the Scotland Yard graph, we determined that we will use 100 playouts for the robber and 150 for the cops. These results will be presented in the section about the Scotland Yard graph. We also experimentally determined that using a search radius of 0 for the chasing agent, results in the best results for the cops. As described in Section 2.2 we will use one robber, who is visible every four moves, and three cops.
The software was run on a computer with an i7-4790k at 4 GHz with 16 GB of ram. To simulate both MCTS agents on the ladder graph with $s = 1$, took on average 15 ms for $\ell = 10$ and 246 ms for $\ell = 100$. For the chasing agent versus the evading agent the difference was smaller, with 15 ms on average for $\ell = 10$ and 20 ms for $\ell = 100$. On the circle graph the MCTS vs MCTS experiments needed on average 2 ms for $c = 10$ and 230 ms for $c = 100$. For the rule based experiments a game on $c = 10$ needed on average 2 ms and a game on $c = 100$ 14 ms.

5.2 Ladder graphs

The first type of graph we look at is the ladder graph. The smallest graph we look at has $\ell = 10$ and the biggest graph $\ell = 100$. For each length, we have three variants; $s = 1$, $s = 2$ and $s = 3$. For the ladder graph we also performed some experiments with double the amount of turns, so 48 turns in total. The robber still shows himself every 4 turns and all other rules also stay as described in Section 2.2
In Figure 5.1 the winrate of the random agent for the cops against the various robber agents is shown. The plot seems to imply that the spacing has little influence on the success of the cops. It looks there are three trends present, one for each robber strategy. From the graph, it becomes clear that a bigger graph favours the robber. What is interesting in this graph is that the evading agent for the robber performs better at smaller sizes of the graph, while the MCTS agent has a better performance overall. What is remarkable is that between $\ell = 10$ and $\ell = 25$ the winrate of the cops fluctuates quite a bit when playing against the evading robber.

For the other plots, we will only match the cops against the evading robber and the robber using MCTS. First, we will look how the cop agents perform against the evading robber agent. The results are shown in Figure 5.2 and for the results with 48 turns in Figure 5.3. Again the plots contain a trend for each strategy, two in this case, which seem to suggest that the spacing of the graph has a very small effect on the relative performance of either strategy. Furthermore, the differences in winrate between consecutive lengths of the graph is also noticeable. For both trends, the variance seems to be smaller at the lower and higher end of the plot. This is likely due to the size of the graph having a smaller impact on the end result of the game. What is interesting to see is that the chasing agent performs better than the MCTS agent across all distances. When we compare the graph of the normal game with the graph for the double rounds, we can see that having more round favours the cops significantly. The variance between beneficial sizes and non-beneficial sizes is also larger. For the MCTS cops there is about an 10% increase when the amount of rounds are doubled. For the chasing cops this increase is even larger at about 20%. Also the wintrate of the chasing cops decrease at a slower rate, than when playing with 24 turns.

Lastly we let the two cop agents play against the MCTS agent for the robber, see Figure 5.4 and Figure 5.5 for the version with 48 turns. Like before there is a trend corresponding to each strategy, with the chase algorithm again outperforming the MCTS agent. What is interesting to see is that the chasing agent stays close to a 100% winrate for bigger sizes than the MCTS agent. Again when doubline the maximum turns played, the winrate of both startegies increase. However the increase for the MCTS cops is smaller this time, at about 5%. Which seems to indicate that the MCTS agent benefits less form an increase in turns. On the other hand, the chasing cops benefit again a lot from the increase in rounds. The cops achieve winrates close to 100% for a much longer time. Also on average the cops achieve a 30% increase of the winrate compared to playing 24 rounds. Just as when playing against the evading robber, the rate in which the winrate decreases for the chasing cops is also much slower.

Something interesting we can see in all graphs and especially Figure 5.2, is that the line makes one big jump downward every few lengths. The amount of length values between each jump is dependent on the s of the graph. More precisely every $s + 1$-th length has a jump down for each graph. As explained in Section 2.3.1 we have s nodes in between each connection between the two lines. Only when the length of the graph is dividable by $s + 1$, there is also a connection on the end of the ladder. Without this connection, the robber can get stuck in the unconnected ends of the graph, allowing the cops to have a better chance of catching the robber. The graphs clearly show that the chasing agent for the cops outperforms the MCTS agent in all situations. Since the curvature of the trends is similar we can conclude that these differences are because of the difference in effectiveness of both strategies. Which is also indicated by the chase algorithm having a greater increase when the amount of turns are doubled.

S

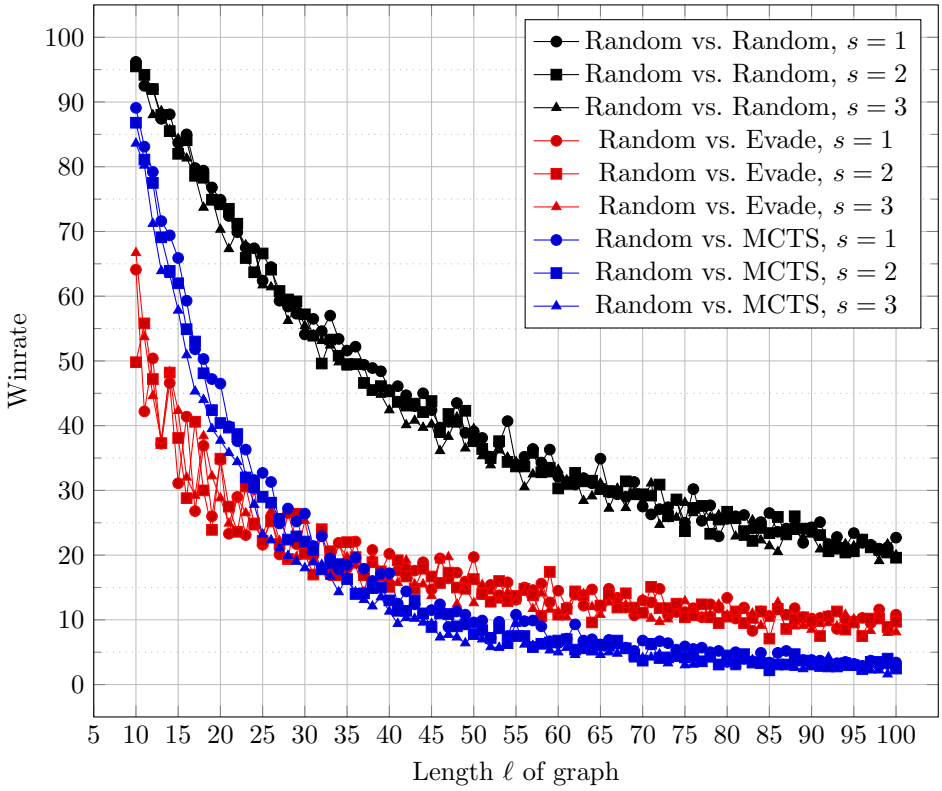

Figure 5.1: Winrate of the random agent for the cops on a ladder graph.

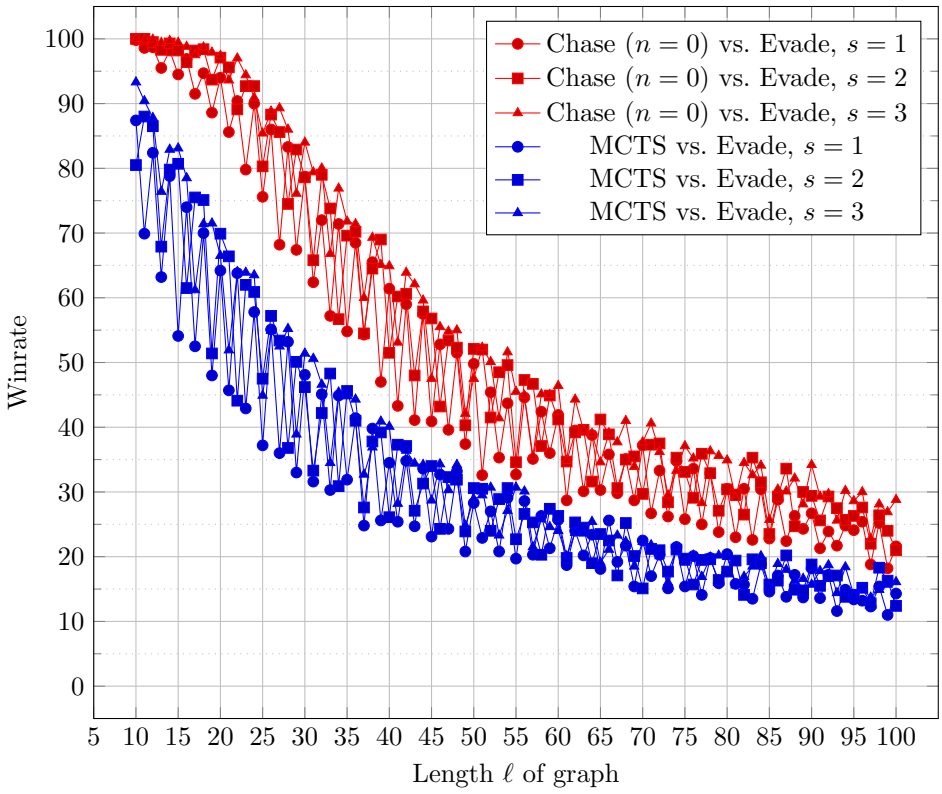

Figure 5.2: Winrate of the chasing and MCTS agent for the cops against the evading robber on a ladder graph.

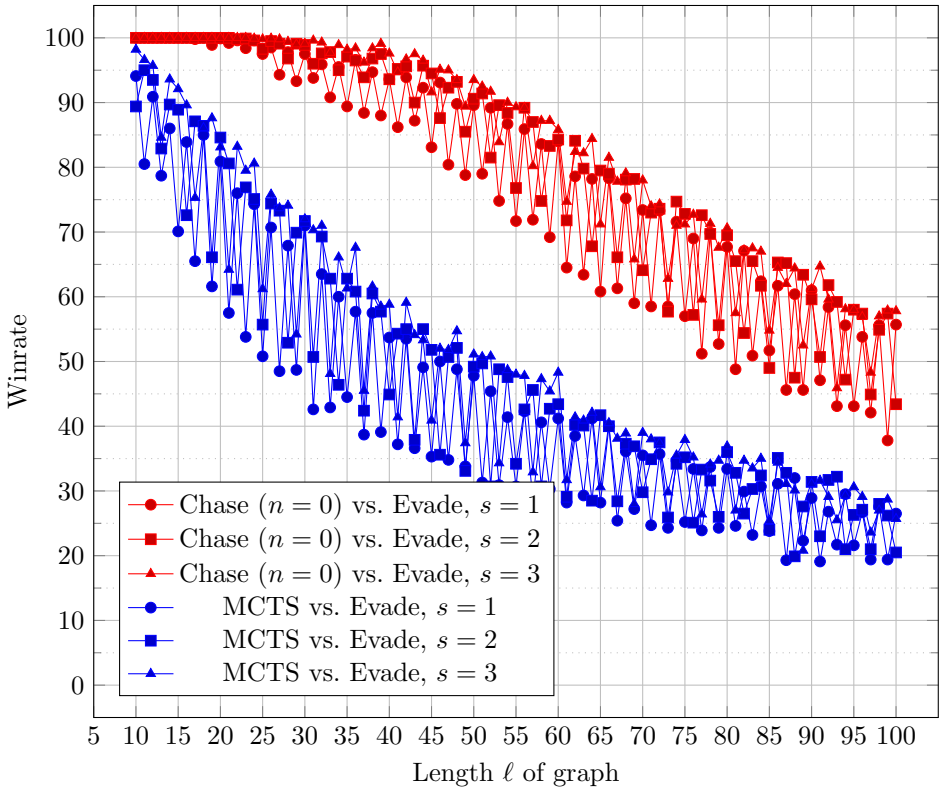

Figure 5.3: Winrate of the chasing and MCTS agent for the cops against the evading robber on a ladder graph with 48 turns.

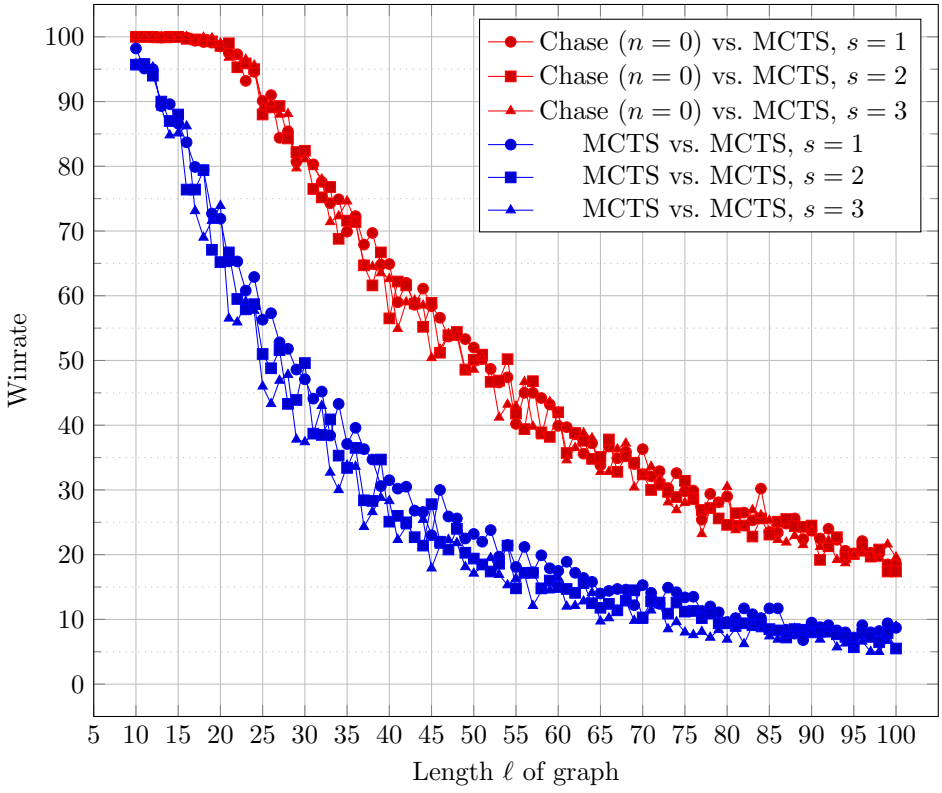

Figure 5.4: Winrate of the chase and MCTS agent for the cops against the MCTS agent of the robber on a ladder graph.

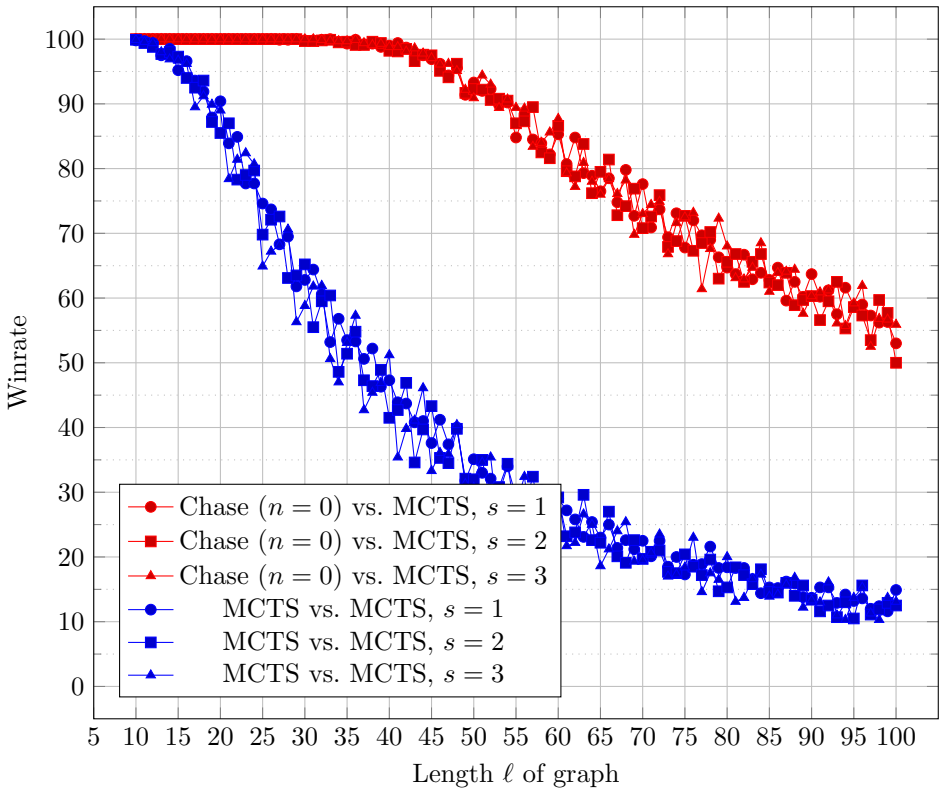

Figure 5.5: Winrate of the chase and MCTS agent for the cops against the MCTS agent of the robber on a ladder graph with 48 turns.

5.3 Circle graphs

The second graph type we will look at is the circle graph. Again we will use $s = 1$, $s = 2$ and $s = 3$ and go through all the sizes from $c = 10$ to $c = 100$. Figure 5.6 shows the results of the random cop against the different robber strategies. As with the ladder graph discussed in the previous section, each strategy forms a clear trend, but more separated. However, with the circle graph, it is clear that the value for s has a visible influence on the winrate of the cops. It is interesting to see that for both the random and the MCTS robber it is the case that a bigger s increases the winrate of the cops, while the opposite is true for the evading robber where a bigger s decreases the winrate.

When using a smart strategy for the cops against the evading robber, the winrates of the strategies are more separated. This can be seen in Figure 5.7. From the graph it it becomes apparent that when playing against the evading robber, the spacing of the graphs has a significant impact. Especially between $s = 1$ and the other spacing values, there is a clear difference. Until $c = 50$, the MCTS cops perform about the same as the chase algorithm with the same s, for $s = 1$ the MCTS algorithm even outperforms the chasing algorithm. For higher values, it starts to perform significantly worse than the chasing algorithm, especially when $s = 1$. This seems to indicate that 150 playouts are not sufficient for circle graphs with $c > 50$.

When we let the cops play against the MCTS robber each strategy for the cops form its own trend again, as can be seen in Figure 5.8. It is noteworthy that the cops using the chasing algorithm perform quite a bit better than the cops using MCTS.

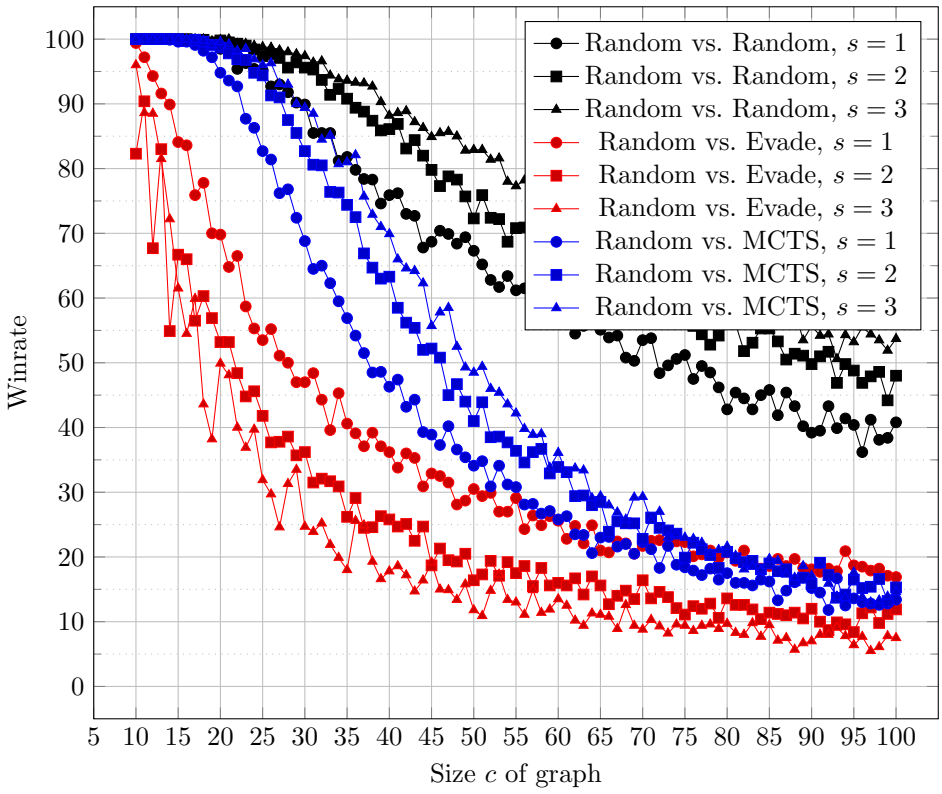

Figure 5.6: Winrate of the random agent for the cops on a circle graph.

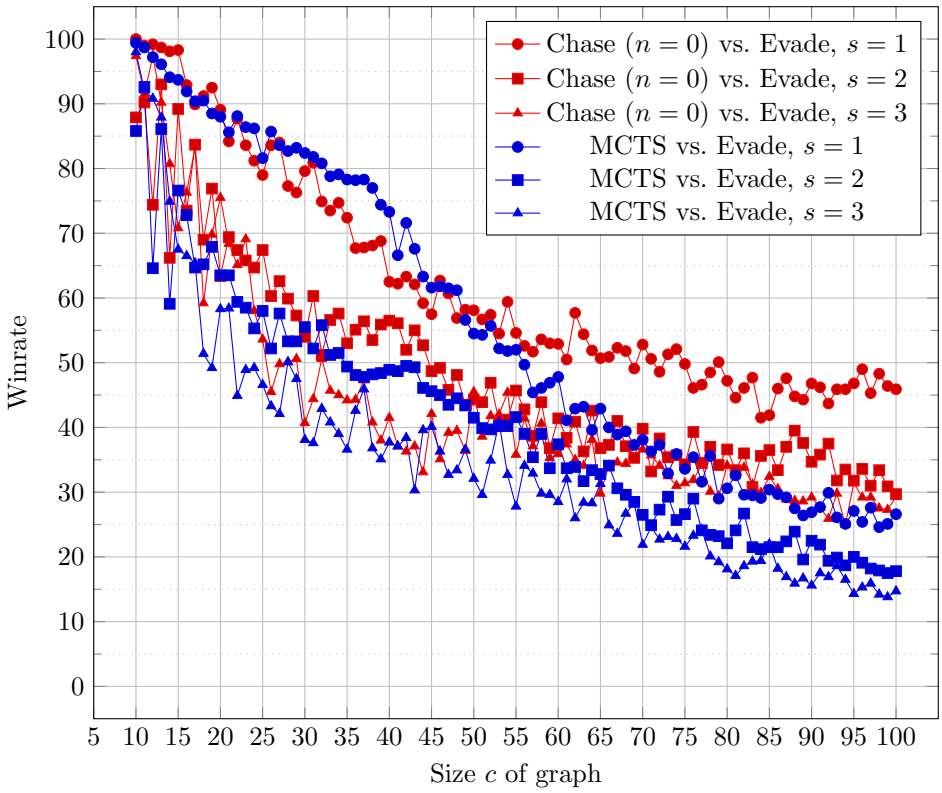

Figure 5.7: Winrate of the chasing and MCTS agent for the cops against the evading robber on a circle graph.

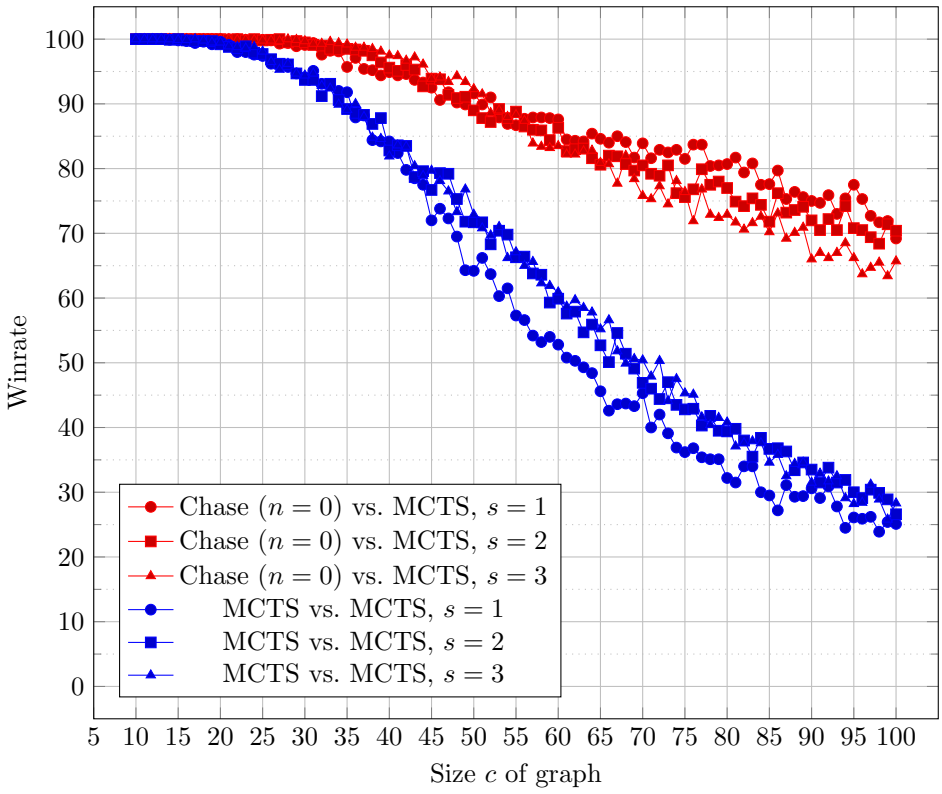

Figure 5.8: Winrate of the chase and MCTS agent for the cops against the MCTS agent of the robber on a circle graph.

5.4 Scotland Yard

The last graph we will look at is the modified Scotland Yard graph. Since this graphs is a modified version of a game board used in a real game, we used this graph to determine the search radius for the chasing cops and determine the amount of playouts used by each MCTS agent. First we let both MCTS agents play against a random player, while varying the playouts used. The results can be found in Figure 5.9. We can see that when playing against a random player, the amount of playouts used doesn't seem to have any effect on the winrate. The fluctuation in winrate is caused by the random start locations each game.
The winrates of the different radii can be seen in Figure 5.10. Here, we can see that the radius has a big influence on the winrate of the cops and that a higher radius leads to a lower winrate. It is interesting to see that the winrate doesn't change much after $r = 4$. This is the same as the number of rounds where the robber is hidden. The experimental data doesn't provide enough evidence to make any conclusions, so further research is required to see if there any connection. What the data does show is that the cops are most successful when moving to the last known position of the robber, so $n = 0$.
The results from the different agents playing against each other are shown in Table 5.1. The table clearly shows that the chase agent for the cops achieves significant better results than the MCTS agent. Interestingly, for the robber, it is a different story. There is not as big of a difference between the evade agent and the MCTS agent for the robber. When playing against the smart agents of the cops, the difference is less than 10%.

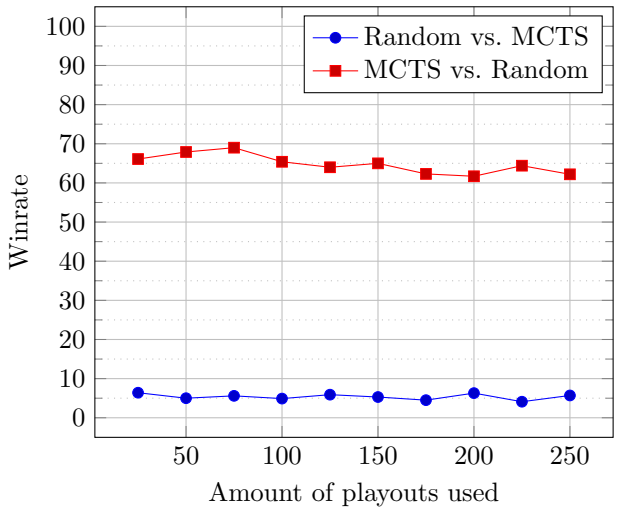

Figure 5.9: Winrate chasing cops against various robber strategies.

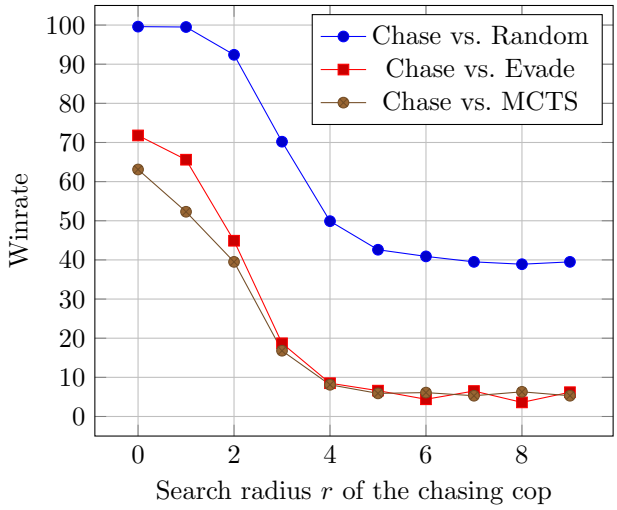

Figure 5.10: Winrate chasing cops against various robber strategies.

Cops \ Robber	Random	Evade	MCTS(100)
Random	42.2%	5.0%	4.9%
Chase($n = 0$)	99.6%	71.8%	75.5%
MCTS(150)	65.5%	22.0%	12.3%

Table 5.1: Winrates of the different agents playing against each other, with parameters in parenthesis.

5.5 Summary

To test the effectiveness of the different strategies, we had them play against each other on a multitude of graphs. For both the ladder and the circle graph, we varied the size and the spacing of the graph. With the ladder graph the spacing of the graph had a minimal effect on the winrate of either player. Here the size was the determining factor for the winrate. For the cops, the chasing agent performed better than the MCTS agent against both

smart strategies of the robber. For the robber, it is the case that the MCTS agent performs better than the evading agent.

On the circle graph, we got more interesting results. First, it becomes clear that the spacing of the graph has a much larger influence on the winrate of the cops, especially when the random cop agent is playing against the various robber agents. Also, when playing against the evading robber, the spacing has a big influence. For $s = 1$, the MCTS agent performs better than the chasing agent at certain sizes. Still, for a size greater than 50, the MCTS agent begins to perform worse than the chasing agent. When playing against the MCTS robber, the spacing has a much smaller effect. This indicates that the evading agent performs differently for each s chosen.

We used the modified Scotland Yard board as a baseline to test the effectiveness of different playout values for each MCTS agent. The number of playouts seemed to have little effect on the winrate, but clearly had an effect on the runtime of the strategy. On this graph too it was the case that the chasing agent performed better than the MCTS agent.

6. Conclusions and future research

We looked at the game of Cops & Robbers, a hide and seek game with imperfect information for one player, the cops, and perfect information for the other player, the robber. We created two player specific AI agents and one common agent. We looked at how these agents perform against each other on three different game board types. The main goal was to see if Monte Carlo Tree Search (MCTS) can successfully be used as an agent for the cops.

We have shown that using MCTS does increase the winrate of both players compared to the random agent. In general we can say that the chase strategy deals better with the imperfect information than the MCTS algorithm used. On both the ladder graphs and the modified Scotland Yard board, the chase agent for the cops and the evading agent for the robber performed better than the MCTS counterpart. We concluded that this likely was because of the value used to determine the score of a move in the MCTS algorithm. On the circle graph, the differences between the different strategies were more clear cut. The value of the graph parameters had a larger influence on the different strategies. Especially when playing against the evade strategy of the robber, the value of s had a big influence in the effectiveness of the strategies. It was also the case that the MCTS and chase strategies performed similarly for graph sizes smaller than 50.

Given the observation it is interesting to look more into the circle graphs, since they provided the most interesting results. We are curious to take a closer look as to why the spacing of the graph had such a big influence on the results. Furthermore it would also be interesting to use different patterns within the circle graph instead of simply skipping every s nodes.

With regards to the MCTS agent for the cops, it would be interesting to see if we can use the chase or the evade strategy to improve the simulations of the MCTS agent. Furthermore it should be researched how using a different score for the MCTS agent can improve its ability to deal with the imperfect information.

Bibliography

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017. DOI: 10.1038/nature24270.

[2] J. Heinrich and D. Silver, "Self-Play Monte-Carlo Tree Search in computer poker," *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[3] F. Frydenberg, K. R. Andersen, S. Risi, and J. Togelius, "Investigating MCTS modifications in general video game playing," in *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, Aug. 2015, pp. 107–113. DOI: 10.1109/CIG.2015.7317937.

[4] *Ravensburger website*. [Online]. Available: https://www.ravensburger.com/start/index.html.

[5] *Scotland Yard rules*. [Online]. Available: https://www.ravensburger.com/spielanleitungen/ecm/Spielanleitungen/Scotland_Yard_W_And_B_GB.pdf.

[6] Mark van den Bergh, "The game of Cops and Robbers on geometric graphs," Master Thesis, Universiteit Leiden, Leiden, 2017.

[7] M. Bastian, S. Heymann, and M. Jacomy, *Gephi: An open source software for exploring and manipulating networks*, 2009.

[8] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012. DOI: 10.1109/TCIAIG.2012.2186810.

[9] P. Nijssen and M. H. M. Winands, "Monte Carlo Tree Search for the hide-and-seek game Scotland Yard," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 4, pp. 282–294, Dec. 2012. DOI: 10.1109/TCIAIG.2012.2210424.

[10] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, Dec. 2018. DOI: 10.1126/science.aar6404.

[11] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Machine Learning: ECML 2006*, vol. 4212, Springer Berlin Heidelberg, 2006, pp. 282–293. DOI: 10.1007/11871842_29.