# Monte-Carlo Tree Search for the Game of Scotland Yard

J. (Pim) A.M. Nijssen and Mark H.M. Winands

*Abstract*— **This paper describes how Monte-Carlo Tree Search (MCTS) can be applied to play the hide-and-seek game Scotland Yard. It is essentially a two-player game in which the players are moving on a graph-based map. We show how limiting the number of possible locations of the hider by using information about the hider's moves increases the performance of the seekers considerably. We also propose a new technique, called Location Categorization, that biases the possible locations of the hider. The experimental results show that Location Categorization is a robust technique which significantly increases the performance of the seekers in Scotland Yard. Next, we show how to handle the coalition of the seekers in Scotland Yard by using Coalition Reduction. This technique balances each seeker's participation in the coalition by letting them seek the hider more greedily. Coalition Reduction improves the performance of the seekers significantly. Furthermore, we explain how domain knowledge is incorporated by applying $\epsilon$-greedy playouts for the hider and the seekers and move filtering to improve the performance of the hider. Finally, we test the performance of our MCTS program against a commercial Scotland Yard program on the Nintendo DS. The results show that the MCTS-based program plays stronger than this program.**

## I. Introduction

Over the last few years, Monte-Carlo Tree Search (MCTS) [1], [2] has become increasingly popular for letting computers play a wide variety of games. MCTS is a best-first search technique which relies less on domain knowledge than more traditional search algorithms like $\alpha\beta$-search [3] and $\max^n$ [4]. Instead of using a heuristic evaluation function, it applies Monte-Carlo simulations to guide the search. Bandit-based reinforcement learning algorithms [2], [5] are applied to recursively build the search tree. MCTS has proven its strength in two-player games such as Go [6], Lines of Action [7], Hex [8] and Amazons [9], multi-player games such as Chinese Checkers [10] and one-player games such as SameGame [11]. MCTS is also widely used for General Game Playing [12].

A challenging domain for MCTS is the class of games with imperfect information. In these games, information is hidden from the players. In the past, MC-based approaches have already been successfully applied to this class of games, such as Bridge [13], Poker [14], Scrabble [15], and Klondike Solitaire [16]. Recently, MCTS has been applied in imperfect information games as well. Ciancarini and Favini [17] showed that MCTS is able to deal with imperfect information in the game of Kriegspiel, a variant of chess. However, their techniques are domain specific and cannot directly be applied to other types of games with imperfect information. MCTS variants have also been applied in Poker to handle the intricacies of the game [18], [19].

A subclass of games with imperfect information is the class of hide-and-seek games. Hide-and-seek games are played by two or more players in two teams: the hiders and the seekers. The goal of the seekers is to capture the hiders. The goal of the hiders is to avoid the seekers for as long as possible. The seekers should obtain information on the locations of the hiders throughout the game. The hide-and-seek game we focus on in this paper is Scotland Yard. Scotland Yard is a popular modern board game with a hide-and-seek mechanism, which won the prestigious *Spiel des Jahres* award in 1983. The seekers, called detectives, have to determine the location of a mobile hider, called Mister X, based on a limited amount of information. The seekers should cooperate in a coalition to capture the hider. Because there is only limited domain knowledge about Scotland Yard available, it is difficult to construct an evaluation function for an $\alpha\beta$- or expectimax-based algorithm. Therefore MCTS is a promising approach to let a computer play this game.

In this paper we describe how MCTS can be applied to handle imperfect information and fixed coalitions for Scotland Yard. We show how to use information about the hider's moves to prevent the seekers from investigating impossible game positions. We also propose a new technique, called Location Categorization. This technique allows the seekers to make a more reliable prediction for the current location of the hider. Next, we propose Coalition Reduction to handle the cooperation of the seekers, which can be used to let the seekers participate in the coalition more effectively. Furthermore, we explain how to incorporate some rudimentary Scotland Yard knowledge in MCTS. We discuss how to use $\epsilon$-greedy playouts for the seekers and the hider, and move filtering for the hider. Finally, we evaluate the performance of our MCTS-based program against a commercial Scotland Yard program, published by DTP Young Entertainment GmbH & Co. KG, for the Nintendo DS.

The paper is organized as follows. In Section II we give a brief introduction to the game Scotland Yard. In Section III we give an overview of MCTS. In Section IV we explain how MCTS can be applied in Scotland Yard and which enhancements are proposed. The experiments and results are described in Section V. Finally, in Section VI we present the conclusions based on the results, and some possible future research topics.

## II. Scotland Yard

In this section we give an introduction to the game of Scotland Yard. In Subsection II-A we give a brief background on Scotland Yard and in Subsection II-B we explain the rules.

### A. Background

The game of Scotland Yard was introduced in 1983. It was developed by Manfred Burggraf, Dorothy Garrels, Wolf

978-1-4577-0011-8/11/$26.00 ©2011 IEEE

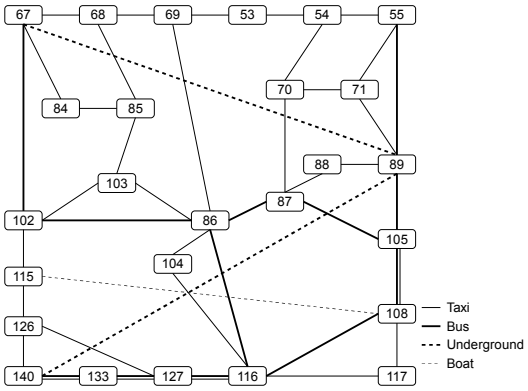


Fig. 1. A subgraph of the Scotland Yard map

Hörmann, Fritz Ifland, Werner Scheerer and Werner Schlegel. The original version was published by Ravensburger, but the game was also published for the English-language market by Milton Bradley [20].

In 1998, Ravensburger Interactive Media GmbH developed Scotland Yard program for Microsoft Windows, which was published by Cryo Interactive Entertainment. It did not only feature the original game, but also a computer enhanced version which introduced some role-playing game elements. Another Scotland Yard program was created in 2008. It was developed and published by DTP Young Entertainment GmbH & Co. KG and was released for the Nintendo DS. The AI of this program is regarded as quite strong [21].

Only a limited amount of research has been done in Scotland Yard so far. Doberkat *et al.* [22] applied prototype evaluation for cooperative planning and conflict resolution. Some of their proposed strategies are used in our MCTS program. Sevenster [23] performed a complexity analysis on Scotland Yard and proved that the generalized version of the game is PSPACE-complete.

### B. Rules

Scotland Yard is played by 6 players: 5 seekers, called seekers, and 1 hider, called Mister X. Essentially, Scotland Yard is a 2-player game, because the seekers work together in one team with one common goal. The game is played on a map consisting of numbered locations from 1 to 199. The locations are connected by 4 different transportation types: taxi, bus, underground and boat. A subgraph of the map is displayed in Figure 1.

All players start with their pawn randomly placed at one of the 18 possible pre-defined starting locations. Each player starts at a different location. Each detective receives 10 taxi, 8 bus, and 4 underground tickets. Mister X receives 4 taxi, 3 bus, and 3 underground tickets.

The players move their pawn alternately, starting with Mister X. A player moves his pawn along a connection to an unoccupied adjacent location, and pays the ticket corresponding to the chosen connection. Mister X receives the tickets paid by the detectives. When Mister X pays a ticket, it is removed from the game.

Additionally, Mister X receives 5 black-fare tickets and 2 double-move tickets. A black-fare ticket allows the use of any transportation type, including the boat. Along with a regular ticket, Mister X may also play one of his double-move tickets. He can then perform two moves in a row.

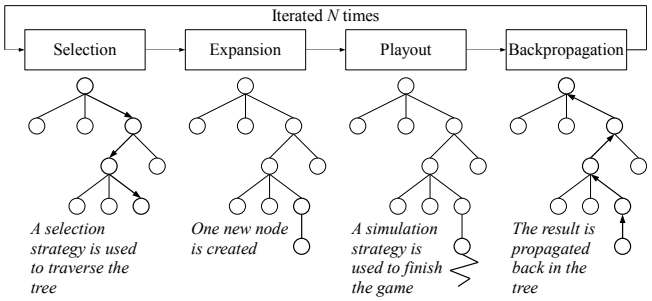


Fig. 2. MCTS scheme (slightly adapted from Chaslot *et al.* [24]).

During the game, Mister X keeps the location of his pawn secret. Only on rounds 3, 8, 13, 18 and 24 he has to announce his location. When Mister X moves, the detectives always get informed which ticket he used.

The goal for the detectives is to capture Mister X by moving one of their pawns to the location occupied by Mister X. The goal for Mister X is to avoid being captured until no detective can perform a move anymore. A detective cannot move if he does not own a ticket which allows him to leave his current location. The maximum number of rounds in Scotland Yard is 24.

## III. Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) [1], [2] is a search technique that gradually builds up a search tree, guided by Monte-Carlo simulations. In contrast to classic search techniques such as $\alpha\beta$-search [3], it does not require a heuristic evaluation function. In the MCTS tree, the nodes represent board positions and the edges represent possible moves. When the search process is started, the root node is created which represents the current game position. The MCTS algorithm consists of four phases [24]: selection, expansion, playout and backpropagation (see Figure 2). By repeating these four phases iteratively, the search tree is constructed gradually. We explain these four phases in more detail below.

*Selection:* In the selection phase, the search tree is traversed, starting from the root, using the *UCT selection strategy* [2]. The child $i$ with the highest score $v_i$ in Formula 1 is selected.

$$v_i = \bar{x}_i + C \times \sqrt{\frac{\ln(n_p)}{n_i}} \quad (1)$$

$\bar{x}_i$ denotes the average score of node $i$. $n_i$ and $n_p$ denote the total number of times child $i$ and parent $p$ have been visited, respectively. $C$ is a constant, which balances exploration and exploitation. This selection strategy is applied until a node is reached that is not fully expanded, i.e. not all of its children have been added to the tree yet.

*Expansion:* In the expansion phase, one node is added to the tree [1]. At the last selected node in the selection phase, this node is chosen randomly from the subset of children that are not added to the tree yet.

*Playout:* During the playout phase, moves are played, starting from the position represented by the newly added node, until the game is finished. These may be random moves. However, game knowledge can be incorporated to make the playouts more realistic. This knowledge is incorporated in a *simulation strategy* [25], [26].

*Backpropagation:* In the backpropagation phase, the result of the playout is propagated back along the previously traversed path up to the root node. The result is backpropagated in a negamax-like fashion, similar to MCTS implementations in Go [1].

These four phases are repeated either a fixed number of times or until the time runs out. After the search is finished, we select the robust max child, i.e. the child of the root with the highest number of visits, as the best move [1].

## IV. MCTS for Scotland Yard

The basic MCTS algorithm is designed for two-player games with perfect information. When using MCTS in a game with imperfect information, the algorithm has to be altered slightly. In Subsection IV-A we discuss how the number of possible locations of the hider can be limited. In Subsection IV-B we propose Location Categorization to bias the remaining possible locations. Next, in Subsection IV-C we explain how $\epsilon$-greedy playouts are used to incorporate knowledge in the playout phase. In Subsection IV-D we show how to handle the fixed coalition of the seekers by using the backpropagation strategy called Coalition Reduction. Finally, in Subsection IV-E we describe move filtering that lets the hider use his tickets more efficiently.

### A. Limiting the possible locations

When MCTS is applied in a hide-and-seek game, at each iteration of the algorithm a guess can be made concerning the location of the hider. For the seekers in Scotland Yard, this can be done by placing the hider on any of the empty locations on the board. However, it is possible for the seekers to limit the group of possible locations by removing the locations where the hider cannot be located, based on the information about the type of ticket he played.

The list of possible locations is updated every move. When the hider plays a ticket, the new list of possible locations $N$ is calculated, based on the old list of possible locations $M$, the current locations of the seekers $D$, and the ticket $t$ played by the hider, using Algorithm 1. Note that at the start of the game, $M$ is initialized with the 18 possible starting locations, minus the 5 starting locations of the seekers.

In this algorithm, the method targets$(p, t)$ returns the list of locations reachable from location $p$ using ticket $t$. When the hider surfaces, location(hider) is the location he surfaced at. When a seeker makes a move, the target location is excluded from the list of possible locations, provided this location was a possible location and the hider was not captured.

An example of this computation is given in Figure 3 using the subgraph of the map in Figure 1. Assume the hider surfaces at location 86 in round 8 and the seekers move to locations 85, 115, 89, 116 and 127, respectively. When

**Algorithm 1** Computation of the list of possible locations of the hider.

```
N ← ∅
if hider surfaces then
  N ← location(hider)
else
  for all p ∈ M do
    T ← targets(p, t)
    for all q ∈ T\(N ∪ D) do
      add q to N
    end for
  end for
end if
return N
```

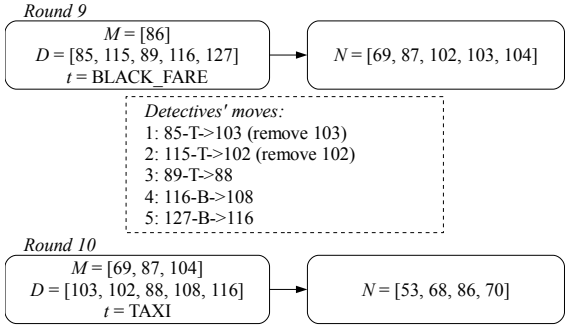


Fig. 3. Example of a computation of possible locations.

the hider plays a black-fare ticket in round 9, the new list of possible locations becomes $N$ = [69, 87, 102, 103, 104]. Location 116 is also reachable from 86 with a black-fare ticket, but because this location belongs to $D$, i.e. there is a seeker on that location, it is not added to $N$. After seeker 1 moves to 103 in round 9, this location is removed from the list of possible locations. After seeker 2 moves to 102, this location is removed as well. In round 10, the hider plays a taxi ticket. The locations reachable from location 69 are 53, 68 and 86 and are all added to $N$. The locations reachable by taxi from 87 are 70 and 88. Because 88 belongs to $D$, only 70 is added to $N$. The two locations reachable from 104 are 86 and 116. Because 86 already belongs to $N$ and 116 belongs to $D$, neither location is added to $N$.

### B. Location Categorization

Some of the possible locations calculated in Algorithm 1 are more probable than others. For example, when the hider played a black-fare ticket in round 9 in the example in Figure 3, he probably did not move to 102, 103, or 104, because he could be captured in this round by one of the seekers on these locations. It is more likely he moved to 69 or 87.

The performance of the seekers could be improved by biasing the possible locations of the hider. This is done by categorizing the possible locations. These categories are numbered from 1 to $L$, where $L$ is the number of categories. This technique is called Location Categorization. The type of categorization is game-dependent. For Scotland Yard, we use three different types of categorization:

| Category | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $a$ | 2454 | 9735 | 4047 | 1109 | 344 |
| $n$ | 12523 | 14502 | 7491 | 2890 | 1756 |

TABLE I
EXAMPLE OF A GENERAL TABLE WITH THE MINIMUM-DISTANCE CATEGORIZATION AFTER PLAYING 1000 GAMES.

*Minimum distance:* A categorization is made based on the distance of the possible location to the nearest seeker. The category number equals the number of moves this seeker has to perform to reach the possible location. We assume the seeker can use any transportation type except the boat. We make this assumption, because it allows us to use a lookup table to increase the speed of the program. All locations with a minimum distance of 5 or more are grouped into one category. The idea behind this categorization is that the possible locations near the seekers are investigated less often. The hider could try to exploit this behavior, though it is risky, offsetting a possible benefit.

*Average distance:* A categorization is made based on the average distance of all seekers to the possible location. This number is rounded down. The category number equals the average number of moves the seekers have to travel to reach the possible location. All locations with an average distance of 5 or more are grouped into one category.

*Station type:* A categorization is made based on the transportation types connected to the possible location. Locations with only taxi connections belong to category 1, locations with taxi- and bus connections belong to category 2, locations with taxi-, bus- and underground connections belong to category 3, and locations with a boat connection belong to category 4.

After the hider performs a move the possible locations are divided into the different categories, based on the selected categorization.

There are two different ways to store the information about the possible categories and the category of the location of the hider. In the *General* table, we store for each category both the number of times one or more possible locations belonged to the category, $n$, and the number of times the actual location of the hider belonged to the category, $a$. This way of storing and using information is similar to the transition probabilities used in Realization Probability Search, which was successful in Shogi [27], Lines Of Action [28], and Amazons [29]. An example of the General table is given in Table I. In the *Detailed* table, for each possible combination of categories, we store how many times the actual location of the hider belonged to each category. An example is given in Table II. This table only shows the category combinations which occurred at least 100 times.

There are two different ways to gather the information for these tables: offline and online. When using *offline* information gathering, first a large number of games is played and the information is stored in a file. This information can later be used by the seekers. This technique is useful when the opponent is unknown and there are not enough games to gather a sufficient amount of information. When using

| Category | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Combination | | | | | |
| 1 | 1542 | - | - | - | - |
| 2 | - | 2801 | - | - | - |
| 1,2 | 666 | 4776 | - | - | - |
| 3 | - | - | 977 | - | - |
| 1,3 | 14 | - | 252 | - | - |
| 2,3 | - | 67 | 208 | - | - |
| 1,2,3 | 210 | 1558 | 1642 | - | - |
| 4 | - | - | - | 262 | - |
| 2,3,4 | - | 23 | 39 | 90 | - |
| 1,2,3,4 | 18 | 224 | 263 | 179 | - |
| 2,3,4,5 | - | 57 | 191 | 183 | 88 |
| 1,2,3,4,5 | 2 | 210 | 448 | 307 | 164 |

TABLE II
EXAMPLE OF A DETAILED TABLE WITH THE MINIMUM-DISTANCE CATEGORIZATION AFTER PLAYING 1000 GAMES.

*online* information gathering, the seekers start without any information. At the end of each game, the seekers update the information with the statistics gathered from the last game. The advantage of online information gathering is that the player can adapt itself to its opponent when enough games are played.

The seekers use a vector with length $L$ to select a location for the hider at the start of each MCTS iteration. These values represent the weights of the categories. When using the General information, this vector consists of the values $[\frac{a_1}{n_1}, \frac{a_2}{n_2}, \cdots, \frac{a_L}{n_L}]$. When using the Detailed information, this vector is directly taken from the table, by extracting the vector corresponding to the combination of categories. If the total number of occurrences of this combination of categories is smaller than a certain threshold, in our MCTS program 100, the information is not used and the possible locations are randomly chosen.

There are two different ways the vector can be used to select a possible location. When using *one-step* selection, each possible location gets a probability to be selected. Roulette-wheel selection is used to select a possible location. The size of each possible location on the wheel is corresponding to the value of its category in the vector. When using *two-step* selection, each location category gets a probability to be selected. We use roulette-wheel selection to select a category. The size of each category on the wheel is corresponding to its value in the vector. After selecting a category, one of the possible locations from this category is randomly chosen.

We remark that we use a 'big-data' approach to set the weights. Such an approach has been successful in Shogi [27], Lines Of Action [28], Amazons [29] and Othello [30]. Of course, machine-learning techniques, though less trivial, could also be used to tune them.

### C. $\epsilon$-greedy playouts

In the MCTS program, we apply $\epsilon$-greedy playouts to incorporate domain knowledge [10], [31]. When selecting a move in the playouts, the move is chosen randomly with a probability of $\epsilon$. Otherwise, a heuristic is used to determine the best move. We use the following two heuristics for the hider and the seekers. When the hider has to move in the playout, he tries to maximize the distance to the closest

seeker. If there are several of such moves, he selects the one which maximizes the number of possible locations. If there are still multiple moves tied for best move, one is chosen randomly. When a seeker has to move in the playout, he selects the move which minimizes the sum of the distances to all possible locations of the hider [22]. If there are several of such moves, one is chosen randomly.

### D. Coalition Reduction

Scotland Yard is a cooperative multi-player game. Therefore, the seekers can be considered as one player, making the game essentially a 2-player game. If in a playout one seeker captures the hider, the playout is considered a win for all seekers and the result is backpropagated accordingly. However, when using this backpropagation rule we observed that seekers sometimes rely too much on the other seekers and do not make any efforts to capture the hider. For solving this problem, we propose Coalition Reduction. If the seeker who is the root player captures the hider, a score of 1 is returned. If another seeker captures the hider, a smaller score, $1 - r$, is returned, where $r$ = [0, 1]. If the value of $r$ is too small, seekers have the tendency to get lazy. If their own position is not good, i.e. they are far away from the possible locations of the hider, they tend to rely on the other seekers too much. If the value of $r$ is too large, the seekers become too selfish and do not cooperate anymore. In Subsection V-E we experimentally fine-tune this parameter.

### E. Move filtering

The hider only has a limited number of black-fare tickets, so he should use them wisely. Black-fare tickets should only be used by the hider to increase the uncertainty about his location or to travel by boat. We implemented some straightforward game-specific knowledge rules regarding the use of black-fare tickets to prevent the hider from squandering them. The hider is not allowed to use black-fare tickets in the following three situations: 1) during the first two rounds, 2) during a round when he has to surface, or 3) when all possible locations only have taxi connections. In the first situation, there is already a large uncertainty about the hider's location. In the second and third situation, using a black-fare ticket does not increase the uncertainty about the hider's location compared to using a 'normal' ticket.

## V. Experiments

In this section we first give an overview of the experimental setup in Subsection V-A. In Subsection V-B we show how limiting the possible locations of the hider influences the performance of the seekers. In Subsection V-C we give the results of the experiments with $\epsilon$-greedy playouts. In Subsection V-D we give an overview of the performance of the seekers with Location Categorization. In Subsection V-E we show how the seekers with Coalition Reduction perform. In Subsection V-F we show how the hider performs with move filtering. Finally, in Subsection V-G we give an overview of how our MCTS program performs against the Scotland Yard program on the Nintendo DS.

### A. Setup

The engines for Scotland Yard and the AI players are written in Java. For the hider, we set the exploration constant $C$ to 0.2. For the seekers, $C$ was set to 2.0. These values were achieved by systematic testing. All MCTS-based players used 10,000 playouts for selecting the best move. In all experiments, 2,500 games were played to determine the win rate. All results are given with a confidence level of 95%. The experiments were run on a cluster containing of AMD64 Opteron 2.4 GHz processors. Depending on the settings, one game takes approximately 2–4 minutes to finish.

In the experiments we use different types of players. The *Random* player selects moves in a uniform random way. The *Greedy* player applies a 1-ply search with the heuristics explained in Subsection IV-C. The *Basic-MCTS* player combines MCTS with UCT. It limits the possible locations using information about the hider. It uses uniform random playouts. This player acts as the basis of the Greedy-MCTS player. The *Greedy-MCTS* player uses $\epsilon$-greedy playouts with $\epsilon$ = 0.2. This value was obtained by systematic testing. It applies the heuristics explained in Subsection IV-C.

### B. Limiting the possible locations

In the first set of experiments, we test how limiting the number of possible locations increases the performance of MCTS. Random, Greedy, and Basic-MCTS seekers with and without limiting the possible moves played against a Random and a Greedy hider. The results are given in Table III.

These results show that limiting the possible locations of the hider using information about his moves is a substantial improvement. Without this limitation, MCTS performs worse than the Greedy seekers. If we limit the possible locations, MCTS performs considerably better than the 1-ply search based Greedy seekers.

### C. $\epsilon$-greedy playouts

To test the influence of $\epsilon$-greedy playouts on the performance of the seekers, we let Basic-MCTS and Greedy-MCTS seekers play against a Basic-MCTS and a Greedy-MCTS hider. The results are given in Table IV.

These results show that using $\epsilon$-greedy playout is a considerable improvement, not only for the seekers, but also for

| | hider | |
|---|---|---|
| seekers | Random | Greedy |
| Random | 51.4% ± 2.0 | 0.1% ± 0.1 |
| Basic-MCTS (no limiting) | 90.8% ± 1.1 | 0.1% ± 0.1 |
| Greedy | 100.0% ± 0.0 | 51.0% ± 2.0 |
| Basic-MCTS | 100.0% ± 0.0 | 82.4% ± 1.5 |

TABLE III

Win rates of four different players as seekers against a Random and a Greedy hider.

| | hider | |
|---|---|---|
| seekers | Basic-MCTS | Greedy-MCTS |
| Basic-MCTS | 47.7% ± 2.0 | 33.8% ± 1.9 |
| Greedy-MCTS | 84.5% ± 1.4 | 72.3% ± 1.8 |

TABLE IV

Win rates of different MCTS players as seekers against different MCTS players as the hider.

Thinking time: 1 second

| | hider | |
|---|---|---|
| seekers | Basic-MCTS | Greedy-MCTS |
| Basic-MCTS | 61.8% ± 1.6 | 50.6% ± 1.6 |
| Greedy-MCTS | 82.6% ± 1.3 | 77.4% ± 1.4 |

Thinking time: 2.5 seconds

| | hider | |
|---|---|---|
| seekers | Basic-MCTS | Greedy-MCTS |
| Basic-MCTS | 71.4% ± 1.5 | 61.0% ± 1.6 |
| Greedy-MCTS | 89.2% ± 1.0 | 83.6% ± 1.2 |

Thinking time: 5 seconds

| | hider | |
|---|---|---|
| seekers | Basic-MCTS | Greedy-MCTS |
| Basic-MCTS | 72.3% ± 1.5 | 65.6% ± 1.6 |
| Greedy-MCTS | 90.6% ± 1.0 | 87.4% ± 1.1 |

TABLE V

WIN RATES OF DIFFERENT MCTS PLAYERS AS SEEKERS AGAINST DIFFERENT MCTS PLAYERS AS THE HIDER WITH DIFFERENT TIME SETTINGS.

the hider. Against both different types of opponents, $\epsilon$-greedy playouts cause a large improvement of the playing strength.

If $\epsilon$-greedy playouts are used, the number of playouts per second is reduced by a factor of 2.5 compared to uniform random playouts. To test the influence of this reduction, we repeat the previous experiment, but instead of using a fixed number of playouts, a time limit per move is used. In Table V we show the results with time limits of 1, 2.5, and 5 seconds. For reference, the Basic-MCTS players can simulate 8,000–10,000 playouts per second at the start of the game. The Greedy-MCTS player can simulate 3,000–5,000 playouts per second. As the game progresses, the number of playouts per second increases, because the playouts become shorter.

The results reveal that $\epsilon$-greedy playouts also improve the performance of both the hider and the seekers significantly when a time limit is used. When the amount of time increases, the seekers perform relatively better than the hider. These results show that Scotland Yard is an uneven game. This is due to its asymmetric nature.

### D. Location Categorization

In the next set of experiments we check which combination of categorization, information type, selection steps and information gathering type works best when using Location categorization. We let Basic-MCTS seekers with Location Categorization play against a Basic-MCTS hider. When using offline information, the information is gathered by letting Basic-MCTS seekers play 2,500 games against a Basic-MCTS hider. The results are summarized in Table VI. For reference, Basic-MCTS seekers win 47.7% ± 2.0 of the games against a Basic-MCTS hider.

The results in Table VI show that the Minimum-distance categorization works best. For this categorization, there is no large difference between the information types, the number of selection steps, and the information gathering types.

To validate the performance of Location Categorization, we test its performance against two other players: a Greedy-MCTS player and a Greedy player. The results against

| Cat. | Info. | Steps | Online | Offline |
|---|---|---|---|---|
| Min. dist. | General | 1 | 52.5% ± 2.0 | 51.5% ± 2.0 |
| Min. dist. | General | 2 | 53.4% ± 2.0 | 52.5% ± 2.0 |
| Min. dist. | Detail | 1 | 53.6% ± 2.0 | 54.1% ± 2.0 |
| Min. dist. | Detail | 2 | 53.3% ± 2.0 | 53.6% ± 2.0 |
| Avg. dist. | General | 1 | 46.5% ± 2.0 | 48.3% ± 2.0 |
| Avg. dist. | General | 2 | 51.7% ± 2.0 | 49.4% ± 2.0 |
| Avg. dist. | Detail | 1 | 48.8% ± 2.0 | 46.0% ± 2.0 |
| Avg. dist. | Detail | 2 | 46.9% ± 2.0 | 49.9% ± 2.0 |
| Station | General | 1 | 44.7% ± 1.9 | 43.7% ± 1.9 |
| Station | General | 2 | 39.7% ± 1.9 | 39.6% ± 1.9 |
| Station | Detail | 1 | 42.4% ± 1.9 | 44.1% ± 1.9 |
| Station | Detail | 2 | 47.0% ± 2.0 | 47.3% ± 2.0 |
| Default win rate: 47.7% ± 2.0 | | | | |

TABLE VI

WIN RATES OF BASIC-MCTS SEEKERS WITH LOCATION CATEGORIZATION AGAINST A BASIC-MCTS HIDER.

| Cat. | Info. | Steps | Online | Offline |
|---|---|---|---|---|
| Min. dist. | General | 1 | 37.0% ± 1.9 | 38.0% ± 1.9 |
| Min. dist. | General | 2 | 37.4% ± 1.9 | 37.6% ± 1.9 |
| Min. dist. | Detail | 1 | 40.2% ± 1.9 | 37.6% ± 1.9 |
| Min. dist. | Detail | 2 | 37.9% ± 1.9 | 36.9% ± 1.9 |
| Default win rate: 33.8% ± 1.9 | | | | |

TABLE VII

WIN RATES OF BASIC-MCTS SEEKERS WITH LOCATION CATEGORIZATION AGAINST A GREEDY-MCTS HIDER.

the Greedy-MCTS hider are shown in Table VII and the results against the Greedy hider are given in Table VIII. For reference, the win rate of the Basic-MCTS seekers is 33.8% ± 1.9 against the Greedy-MCTS player and 82.4% ± 1.5 against the Greedy player. We use the same offline weights as in the previous experiments. Note that they were gathered by playing against a Basic-MCTS player, which behaves differently than these two players.

The results show that Location Categorization improves the performance against both players. Even when using the offline weights the seekers win more games than without Location Categorization.

Finally, we test whether adding Location Categorization to the Greedy-MCTS seekers increases their performance. We let the Greedy-MCTS seekers with Location Categorization play against a Greedy-MCTS hider with different settings. The offline weights are the same as the ones used in the previous experiments. The results are given in Table IX. The default win rate of Greedy-MCTS seekers against a Greedy-MCTS hider is 72.1% ± 1.8.

The results in Table IX show that Location Categorization also works combined with $\epsilon$-greedy playouts for the seekers.

| Cat. | Info. | Steps | Online | Offline |
|---|---|---|---|---|
| Min. dist. | General | 1 | 84.3% ± 1.4 | 84.9% ± 1.4 |
| Min. dist. | General | 2 | 82.2% ± 1.5 | 84.7% ± 1.4 |
| Min. dist. | Detail | 1 | 85.4% ± 1.4 | 85.0% ± 1.4 |
| Min. dist. | Detail | 2 | 84.3% ± 1.4 | 83.8% ± 1.4 |
| Default win rate: 82.4% ± 1.5 | | | | |

TABLE VIII

WIN RATES OF BASIC-MCTS SEEKERS WITH LOCATION CATEGORIZATION AGAINST A GREEDY HIDER.

| Cat. | Info. | Steps | Online | Offline |
|---|---|---|---|---|
| Min. dist. | General | 1 | 77.4% ± 1.6 | 75.3% ± 1.7 |
| Min. dist. | General | 2 | 74.6% ± 1.7 | 74.7% ± 1.7 |
| Min. dist. | Detail | 1 | 76.4% ± 1.7 | 74.3% ± 1.7 |
| Min. dist. | Detail | 2 | 76.2% ± 1.7 | 74.9% ± 1.7 |
| Default win rate: 72.1% ± 1.8 | | | | |

TABLE IX
WIN RATES OF GREEDY-MCTS SEEKERS WITH LOCATION CATEGORIZATION AGAINST A GREEDY-MCTS HIDER.

| $r$ | A | B | C |
|---|---|---|---|
| 0 | 47.7% ± 2.0 | 72.1% ± 1.8 | 75.3% ± 1.7 |
| 0.125 | 49.8% ± 2.0 | 73.6% ± 1.7 | 79.3% ± 1.6 |
| 0.25 | 50.7% ± 2.0 | 71.7% ± 1.8 | 80.6% ± 1.6 |
| 0.375 | 54.0% ± 2.0 | 72.0% ± 1.8 | 78.6% ± 1.6 |
| 0.5 | 53.4% ± 2.0 | 68.2% ± 1.8 | 74.6% ± 1.7 |
| 0.75 | 51.5% ± 2.0 | 60.0% ± 1.9 | 65.8% ± 1.9 |
| 1 | 42.8% ± 1.9 | 44.7% ± 2.0 | 50.5% ± 2.0 |

TABLE X
WIN RATES OF SEEKERS WITH COALITION REDUCTION WITH DIFFERENT VALUES OF $r$ AGAINST THE HIDER IN 3 CONFIGURATIONS.

Using the online weights seems to work only slightly better than using the offline weights gathered against the Basic-MCTS player.

The experiments show that overall there is no significant different between offline and online gathered information. From this we may conclude that Location Categorization is a robust technique when using offline gathered statistics, which can be used against any type of opponent.

### E. Coalition Reduction

To test the performance of seekers with Coalition Reduction, we let different types of seekers with Coalition Reduction play against different types of the hider with different values of $r$. We remark that for $r = 0$, Coalition Reduction is disabled. For $r = 1$, there is no coalition, and all seekers only work for themselves. The results are shown in Table X. We use three different configurations: A) Basic-MCTS seekers against a Basic-MCTS hider, B) Greedy-MCTS seekers against a Greedy-MCTS hider, and C) Greedy-MCTS seekers with Location Categorization using Minimum-distance categorization, the General table, offline gathered weights, and one-step selection against a Greedy-MCTS hider.

These results show that Coalition Reduction with $r = 0.375$ improves the performance of the seekers for configuration A. For configuration C, Coalition Reduction improves the performance as well. The best value of $r$ in this configuration is 0.25. For configuration B, Coalition Reduction does not improve the performance of the seekers significantly.

### F. Move filtering

For the hider, we test how using move filtering increases his performance. We use configuration C described in Subsection V-E, where the hider plays with and without move filtering.

Without move filtering, the hider wins 19.4% ± 1.6. With move filtering, the win rate of the hider is 34.0% ± 1.9. These results show that the performance of the hider improves considerably when he is prevented from squandering black-fare tickets. It is important for the hider to effectively use his limited stock of black-fare tickets.

### G. Performance against the Nintendo DS

To test the performance of the MCTS program, it is matched against the Scotland Yard program on the Nintendo DS. The AI of this program is considered to be rather strong [21]. We only test against this program, because other programs are too weak or impractical to play against.

For the seekers, we use $\epsilon$-greedy playouts, Location Categorization with a General table, offline gathered weights and one-step selection, and Coalition Reduction with $r = 0.25$. For the hider, $\epsilon$-greedy playouts and move filtering are used.

It is not possible to set the thinking time of the Nintendo DS player. It often plays immediately, but it sometimes takes 5–10 seconds to find a move. The thinking time of the MCTS program is on average 1.5 seconds.

Because these games have to be played manually, only 50 games are played, where each program plays 25 times as the seekers and 25 times as the hider. Out of these 50 games, 33 games are won by our program, i.e. 66% ± 13.1. 19 of these games are won as the seekers and 14 as the hider. The Nintendo DS program wins 17 games, of which 11 as the seekers and 6 as the hider. These results show that our program plays stronger than the Nintendo DS program.

## VI. CONCLUSIONS AND FUTURE RESEARCH

In this paper we investigated how MCTS can be applied to play the hide-and-seek game Scotland Yard and how MCTS can be enhanced to improve its performance.

We showed how information about the moves of the hider can be used to limit the number of possible locations. Limiting the possible locations considerably improved the performance of the seekers.

We proposed Location Categorization, a technique that can be used by the seekers in Scotland Yard to give a better prediction for the location of the hider. We introduced three types of categorization: Minimum distance, Average distance and Station. The experiments revealed that the Minimum-distance categorization performs best. By playing against different hider players, we showed that Location Categorization is a robust technique. By using weights that were gained by playing against a Basic-MCTS hider, the performance against the Greedy-MCTS hider and the Greedy hider was improved. Also, adding Location Categorization to Greedy-MCTS seekers increased the performance of the seekers. This enhancement made the seekers play considerably stronger than the strongest hider player.

Furthermore, using $\epsilon$-greedy playouts to incorporate some basic knowledge into the MCTS algorithm considerably improved the performance of both the seekers and the hider. Another way of incorporating game-specific knowledge we proposed is move filtering. Even with some straightforward rules, move filtering turned out to be a considerable improvement for the hider.

We also observed that the performance of the seekers can be improved by applying Coalition Reduction. This technique allows the seekers to cooperate more effectively in the coalition, by preventing them from becoming too lazy or too selfish. We remark that it still has to be determined whether this technique is generally applicable to games with fixed coalitions, or whether it only works for Scotland Yard.

Finally, we showed that MCTS, a technique which uses only basic domain knowledge, was able to play Scotland Yard on a higher level than the Nintendo DS program, which is generally considered to be a strong player.

There are several ways Location Categorization could be improved in the future. New types of categorization may be tested or different categorizations may be combined. This can de done by introducing three-step selection. The first two steps are used to select two categories using two different categorizations. In the third step, a possible location is selected which belongs to both selected categories. Another way of combining two categorizations is by taking the Cartesian product of the categories of both categorizations. It can also be interesting to test the performance of Location Categorization in other hide-and-seek games, for instance Battleship, a two-player game where both players act both as an immobile hider and seeker.

Another future research direction is to continue the recent work of Silver and Veness [32], who extended MCTS to partially observable environments (POMDPs). Their technique, Partially Observable Monte-Carlo Planning (POMCP), was successfully applied to Battleship and a partially observable variant of PacMan. Their technique could be applied to Scotland Yard as well.

## REFERENCES


[1] R. Coulom, "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search," in *Computers and Games (CG 2006)* (H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers, eds.), vol. 4630 of *LNCS*, (Berlin, Germany), pp. 72–83, Springer-Verlag, 2007.

[2] L. Kocsis and C. Szepesvári, "Bandit Based Monte-Carlo Planning," in *Machine Learning: ECML 2006* (J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, eds.), vol. 4212 of *LNCS*, (Berlin, Germany), pp. 282–293, Springer-Verlag, 2006.

[3] D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning," *Artificial Intelligence*, vol. 6, no. 4, pp. 293–326, 1975.

[4] C. Luckhart and K. Irani, "An algorithmic solution of n-person games," in *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI)*, vol. 1, pp. 158–162, 1986.

[5] S. Gelly and Y. Wang, "Exploration Exploitation in Go: UCT for Monte-Carlo Go," in *Neural Information Processing Systems Conference On-line trading of Exploration and Exploitation Workshop*, 2006.

[6] S. Gelly and D. Silver, "Combining Online and Offline Knowledge in UCT," in *ICML '07: Proceedings of the 24th International Conference on Machine Learning*, (New York, NY, USA), pp. 273–280, ACM, 2007.

[7] M. H. M. Winands, Y. Björnsson, and J.-T. Saito, "Monte Carlo Tree Search in Lines of Action," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 239–250, 2010.

[8] B. Arneson, R. B. Hayward, and P. Henderson, "Monte-Carlo Tree Search in Hex," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 251–258, 2010.

[9] R. J. Lorentz, "Amazons Discover Monte-Carlo," in *Computers and Games (CG 2008)* (H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands, eds.), vol. 5131 of *LNCS*, (Berlin, Germany), pp. 13–24, Springer-Verlag, 2008.

[10] N. R. Sturtevant, "An Analysis of UCT in Multi-player Games," *ICGA Journal*, vol. 31, no. 4, pp. 195–208, 2008.

[11] M. P. D. Schadd, M. H. M. Winands, H. J. van den Herik, G. M. J.-B. Chaslot, and J. W. H. M. Uiterwijk, "Single-Player Monte-Carlo Tree Search," in *Computers and Games (CG 2008)* (H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands, eds.), vol. 5131 of *LNCS*, (Berlin, Germany), pp. 1–12, Springer-Verlag, 2008.

[12] Y. Björnsson and H. Finnsson, "CADIAPLAYER: A Simulation-Based General Game Player," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, no. 1, pp. 4–15, 2009.

[13] 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)*, pp. 584–589, 1999.

[14] D. Billings, L. Pena, J. Schaeffer, and D. Szafron, "Using Probabilistic Knowledge and Simulation to Play Poker," in *AAAI*, pp. 697–703, AAAI press, 1999.

[15] B. Sheppard, *Towards Perfect Play of Scrabble*. PhD thesis, Universiteit Maastricht, The Netherlands, 2002.

[16] R. Bjarnason, A. Fern, and P. Tadepalli, "Lower Bounding Klondike Solitaire with Monte-Carlo Planning," in *International Conference on Automated Planning and Scheduling/Artificial Intelligence Planning Systems*, 2009.

[17] P. Ciancarini and G. P. Favini, "Monte Carlo Tree Search Techniques in the Game of Kriegspiel," in *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)* (C. Boutilier, ed.), (San Francisco, CA, USA), pp. 474–479, Morgan Kaufmann Publishers Inc., 2009.

[18] G. Van den Broeck, K. Driessens, and J. Ramon, "Monte-Carlo Tree Search in Poker using Expected Reward Distributions," in *Advances in Machine Learning* (Z.-H. Zhou and T. Washio, eds.), vol. 5828 of *LNAI*, (Berlin, Germany), pp. 72–83, Springer-Verlag, 2009.

[19] M. J. V. Ponsen, *Strategic Decision-making in Complex Games*. PhD thesis, Maastricht University, The Netherlands, 2011.

[20] "Scotland Yard — Board Game — BoardGameGeek." http://www.boardgamegeek.com/boardgame/438/scotland-yard, retrieved April 2011.

[21] A. Frackowski, "[NDS Review] Scotland Yard DS - Hold.Start.Select," 2011. http://holdstartselect.com/nds-review-scotland-yard-ds/.

[22] E.-E. Doberkat, W. Hasselbring, and C. Pahl, "Investigating Strategies for Cooperative Planning of Independent Agents through Prototype Evaluation," in *Coordination Models and Languages (COORDINATION '96)* (P. Ciancarini and C. Hankin, eds.), vol. 1061 of *LNCS*, (Berlin, Germany), pp. 416–419, Springer-Verlag, 1996.

[23] M. Sevenster, "The Complexity of Scotland Yard," in *Interactive Logic* (J. van Benthem, B. Löwe, and D. Gabbay, eds.), pp. 209–246, Amsterdam University Press, 2008.

[24] G. M. J.-B. Chaslot, M. H. M. Winands, J. W. H. M. Uiterwijk, H. J. van den Herik, and B. Bouzy, "Progressive Strategies for Monte-Carlo Tree Search," *New Mathematics and Natural Computation*, vol. 4, no. 3, pp. 343–357, 2008.

[25] B. Bouzy, "Associating Domain-Dependent Knowledge and Monte Carlo Approaches within a Go Program," *Information Sciences*, vol. 175, no. 4, pp. 247–257, 2005.

[26] S. Gelly, Y. Wang, R. Munos, and O. Teytaud, "Modifications of UCT with Patterns in Monte-Carlo Go," tech. rep., INRIA, 2006.

[27] Y. Tsuruoka, D. Yokoyama, and T. Chikayama, "Game-Tree Search Algorithm Based on Realization Probability," *ICGA Journal*, vol. 25, no. 3, pp. 132–144, 2002.

[28] M. H. M. Winands and Y. Björnsson, "Enhanced Realization Probability Search," *New Mathematics and Natural Computation*, vol. 4, no. 3, pp. 329–342, 2008.

[29] Y. Higashiuchi and R. Grimbergen, "Enhancing Search Efficiency by Using Move Categorization Based on Game Progress in Amazons," in *Advances in Computer Games (ACG 2005)* (H. J. van den Herik, S.-C. Hsu, T.-S. Hsu, and H. H. L. M. Donkers, eds.), vol. 4250 of *LNCS*, (Berlin, Germany), pp. 73–87, Springer-Verlag, 2006.

[30] M. Buro, "Experiments with Multi-ProbCut and a New High-Quality Evaluation Function for Othello," in *Games in AI Research* (H. J. van den Herik and H. Iida, eds.), pp. 77–96, 2000.

[31] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1998.

[32] D. Silver and J. Veness, "Monte-Carlo Planning in Large POMDPs," in *Advances in Neural Information Processing Systems 23* (J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, eds.), pp. 2164–2172, Curran Associates, Inc., 2010.