Monte Carlo Tree Search for the Hide-and-Seek Game Scotland Yard

Article *in* IEEE Transactions on Computational Intelligence and AI in Games · December 2012

DOI: 10.1109/TCIAIG.2012.2210424

2 authors, including:

Mark H.M. Winands
Maastricht University
129 PUBLICATIONS **1,875** CITATIONS

Monte-Carlo Tree Search for the Hide-and-Seek Game Scotland Yard

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

***Abstract*—This article describes how Monte-Carlo Tree Search (MCTS) can be applied to play the hide-and-seek game Scotland Yard. This game is essentially a two-player game in which the players are moving on a graph-based map. First, we discuss how determinization is applied to handle the imperfect information in the game. We show how using determinization in a single tree performs better than using separate trees for each determinization. We also propose a new technique, called Location Categorization, that biases the possible locations of the hider. The experimental results reveal that Location Categorization is a robust technique, and significantly increases the performance of the seekers. Next, we describe how to handle the coalition of the seekers by using Coalition Reduction. This technique balances each seeker's participation in the coalition. Coalition Reduction improves the performance of the seekers significantly. Furthermore, we explain how domain knowledge is incorporated by applying ϵ-greedy playouts and move filtering. Finally, we compare the MCTS players to minimax-based players and we test the performance of our MCTS player against a commercial Scotland Yard program on the Nintendo DS. Based on the results we may conclude that the MCTS-based hider and seekers play at a strong level.**

***Index Terms*—Monte-Carlo Tree Search, Imperfect information, Cooperation in games, Scotland Yard**

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 [3]. MCTS is a best-first search technique which, in its basic form, does not rely on heuristic domain knowledge, contrary to more traditional search algorithms like $\alpha\beta$-search [4], max^n [5] and paranoid [6]. Instead of using a heuristic evaluation function, it applies Monte-Carlo simulations to guide the search. Bandit-based reinforcement learning algorithms [2], [7], [8] are applied to recursively build the search tree. MCTS has proven its strength in two-player games such as Go [9], Lines of Action [10], Hex [11] and Amazons [12], multi-player games such as Chinese Checkers [13] and one-player games such as SameGame [14]. MCTS is also widely used for General Game Playing [15].

A challenging domain is the class of games with imperfect information. In these games, information is hidden from the players. In the past, Monte-Carlo-based approaches have already been successfully applied to this class of games, such as Bridge [16], Poker [17], Scrabble [18], and Klondike Solitaire [19]. Recently, MCTS has been applied in imperfect information games as well. Ciancarini and Favini [20] 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 [21], [22]. Imperfect information games can also be played by using the expectimax framework [23]. Billings *et al.* [24] used variations of expectimax, MIXIMAX and its generalization MIXIMIX, to play poker. Schadd *et al.* [25] applied expectimax for playing the game Stratego. They enhanced the search with CHANCEPROBCUT, which allows forward pruning in chance nodes.

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. Most research in hide-and-seek games focuses on mathematical approaches for finding optimal strategies for the seekers (also called pursuers) that guarantees finding one or multiple hiders (also called evaders) in finite time [26], [27]. In this article, a computational intelligence approach in playing hide-and-seek games is investigated. The hide-and-seek game we focus on in this article 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. Scotland Yard has three properties that makes it a challenging and interesting domain for applying and improving tree search techniques. 1) It has imperfect information for some of the players. The seekers perform public moves, while the hider performs both public and private moves. 2) It contains a fixed coalition between 5 of the players, and 3) the game is asymmetric in its goals.

In this article we describe how MCTS can be applied to tackle these challenges. We first show how to handle imperfect information using different determinization techniques. 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

J. (Pim) A. M. Nijssen and Mark H. M. Winands are members of the Games and AI group at the Department of Knowledge Engineering, Faculty of Humanities and Sciences, Maastricht University, Maastricht, The Netherlands. E-mail: {pim.nijssen,m.winands}@maastrichtuniversity.nl

to incorporate domain knowledge in MCTS. We discuss how to use ϵ-greedy playouts for the seekers and the hider, and move filtering for the hider. Subsequently, a comparison is made between the MCTS players and minimax-based players. Finally, we evaluate the performance of our program against a commercial Scotland Yard program for the Nintendo DS.

This article is an extension of [28]. It contains additional experiments and the inclusion of minimax-based search techniques. Furthermore, an improved version of the MCTS program is used for the experiments. This improvement includes more accurate domain knowledge and optimizations to speed up the program.

The article is organized as follows. In Section II we briefly introduce the game Scotland Yard. In Section III we give an overview of the search techniques used in this article: MCTS, paranoid search and expectimax. In Section IV we explain how MCTS can be applied in Scotland Yard and which enhancements are proposed. In Section V we explain how paranoid search and expectimax are applied in Scotland Yard and how these techniques can be enhanced to improve the playing strength. The experiments and results are described in Section VI. Finally, in Section VII we present the conclusions based on the results, and some possible future research topics.

II. Scotland Yard

This section provides an introduction to the game of Scotland Yard. Subsection II-A gives a brief background on Scotland Yard and in Subsection II-B the rules are described.

A. Background

The game of Scotland Yard was introduced in 1983. It was developed by Manfred Burggraf, Dorothy Garrels, Wolf Hörmann, Fritz Ifland, Werner Scheerer and Werner Schlegel. The original version was published by Ravensburger, but the game was also distributed for the English-language market by Milton Bradley [29].

In 1998, Ravensburger Interactive Media GmbH developed a 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 developed in 2008. It was developed by Sproing Interactive Media GmbH 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 [30].

Only a limited amount of research has been performed in Scotland Yard so far. Doberkat *et al.* [31] applied prototype evaluation for cooperative planning and conflict resolution. Some of their proposed strategies are used in our MCTS program (see Subsection IV-A). Furthermore, Sevenster [32] 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 detectives, and 1 hider, called Mister X. Essentially, Scotland Yard is a two-player game, because the seekers work together in one team with one common goal. The game is played on a graph consisting of numbered vertices from 1 to 199. The vertices are connected by 4 different types of edges representing different transportation types: taxi, bus, underground and boat. A subgraph is displayed in Fig. 1. Each player occupies one vertex and a vertex can hold at most one player. The vertex currently occupied by a player is called the *location* of that player.

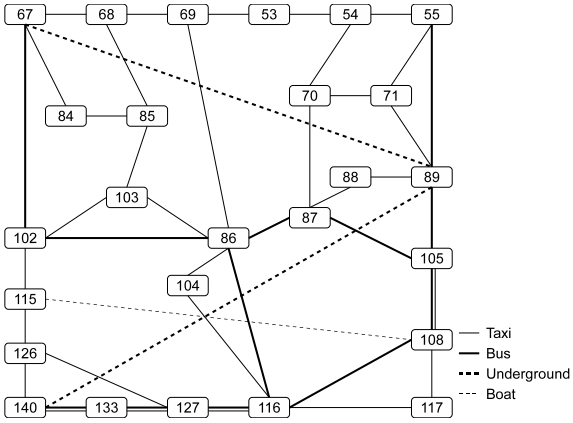

Fig. 1. A subgraph of the Scotland Yard map.

All players start at one of the 18 possible pre-defined starting vertices. Each player starts at a different vertex, which is chosen randomly. 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 alternately, starting with Mister X. A sequence of 6 moves by Mister X and the 5 detectives is called one *round*. When performing a move, a player moves along an edge to an unoccupied adjacent vertex, and plays the ticket corresponding to the chosen edge. Mister X receives the tickets paid by the detectives. When Mister X plays 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 him to move along any edge, including the boat. Along with a regular ticket, Mister X may also play one of his double-move tickets. All detectives then skip their turn for that round.

During the game, Mister X keeps his location secret. Only after moving 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 to the vertex 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. Search Techniques

This Section gives an overview of the search techniques used in this article. Subsection III-A describes MCTS and Subsection III-B briefly discusses the minimax variants.

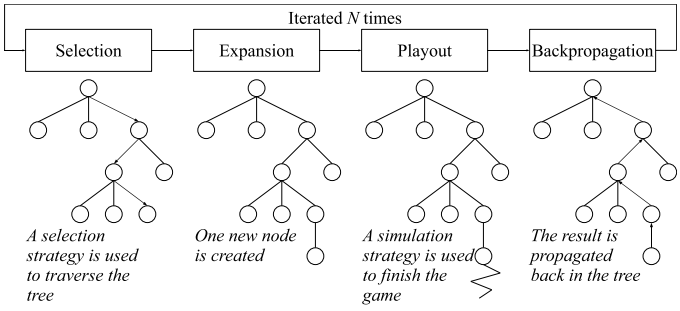

Fig. 2. MCTS scheme (slightly adapted from Chaslot *et al.* [33]).

A. Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) [1], [2] is a best-first 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 [4], 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 [33]: selection, expansion, playout and backpropagation (see Fig. 2). By repeating these four phases iteratively, the search tree is constructed gradually. These four phases are explained in more detail below.

Selection: In the selection phase, the search tree is traversed, starting from the root, using the *UCT selection strategy* [2]. In this article, UCT is enhanced with Progressive History [34]. This is a combination of Progressive Bias [33] and the history heuristic [35]. The child i with the highest score v_i in Formula 1 is selected.

$$v_i = \bar{x}_i + C\sqrt{\frac{\ln(n_p)}{n_i}} + W\frac{\bar{x}_a}{n_i(1-\bar{x}_i)+1} \qquad (1)$$

Here, $\bar{x}_i$ 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. $\bar{x}_a$ represents the average score of move a, i.e. the average score over all playouts in which move a was played. W is a positive constant that determines the influence of Progressive History. The larger the value of W, the longer Progressive History affects the selection of a node. 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. They may be random moves. However, game knowledge can be incorporated to make the playouts more realistic. This knowledge is incorporated in a *simulation strategy* [36], [37]. Though including knowledge in the playouts decreases the number of playouts per second, the more realistic playouts lead to more reliable results which improve the playing strength (see Subsection VI-B).

Another strategy to improve the quality of the playouts is by applying domain-independent techniques, such as the Last-Good-Reply policy [38]–[40], N-grams [40] and Pool-Rave [41]. These techniques are useful if domain-dependent knowledge is not available, computationally expensive, or ineffective.

In this article we only use domain-dependent knowledge (see Subsection IV-A). Combining domain-dependent and domain-independent techniques is beyond the scope of this article and is a possible direction of future research.

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 [4], 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].

B. Minimax-based techniques

Minimax search [42] is a classic depth-first search technique used to play many two-player games with perfect information, most notably Chess [43]. Minimax is usually enhanced with $\alpha\beta$-pruning [4], which reduces the size of the search tree from $O(b^d)$ to $O(b^{\frac{d}{2}})$ in the best case. The amount of pruning can be increased with move ordering, such as the killer move heuristic [44]. This heuristic uses the assumption that when a move is the best one in a certain position, it is also the best one in similar positions. For each ply, two killer moves are stored. A generalization of the killer move heuristic is the history heuristic [35]. The history heuristic is a move ordering technique based on the value of the moves in the past. These move ordering techniques work well when combined with iterative deepening [45]. The search starts with a tree with depth $d = 1$ and when a search is finished, a new search is performed with an increment of d as long as there is time left.

An extension of minimax search is paranoid search [6]. This technique is used in multi-player games. It assumes that all other players are cooperating against the root player. Using this assumption, the game can be reduced to a two-player game. A paranoid search tree contains MAX nodes for the root player and MIN nodes for all other players.

Expectimax [23] is an extension of minimax search to incorporate chance events in the search tree. This is done by adding chance nodes to the tree. The value of a chance node is computed by using Formula 2.

$$v_i = \sum_c P(c) \times v_c \qquad (2)$$

In this formula, v_i is the value of node i, $P(c)$ is the probability of child c being reached from i, and v_c is the value of child c.

In Section V we discuss how paranoid search and expectimax are applied for playing Scotland Yard.

IV. MCTS FOR SCOTLAND YARD

In this section we discuss how MCTS can incorporate the domain knowledge, imperfect information and cooperating players for Scotland Yard. This section is stuctured as follows. First, Subsection IV-A discusses how ϵ-greedy playouts are used to incorporate knowledge in the playout phase. Subsection IV-B explains how determinization can be applied to handle imperfect information. Next, Subsection IV-C discusses how to keep track of the possible locations of the hider. In Subsection IV-D we propose Location Categorization to bias the remaining possible locations. Subsection IV-E shows how to handle the fixed coalition of the seekers by using the backpropagation strategy called Coalition Reduction. Finally, Subsection IV-F describes move filtering that lets the hider use his tickets more efficiently.

A. ϵ-greedy playouts

In the MCTS algorithm, we apply ϵ-greedy playouts to incorporate domain knowledge [13], [46]. When selecting a move in the playouts, the move is chosen randomly with a probability of ϵ. Otherwise, a heuristic is used to determine the best move. Because Scotland Yard is an asymmetric game, different heuristics have to be defined for the hider and the seekers. Furthermore, separate values for ϵ can be determined for the hider and the seekers in the playout. For the hider, we have defined one heuristic to determine the best move. This heuristic, Maximize Closest Distance (MCD), maximizes the number of moves the closest seeker should make to arrive at the target vertex of the move. For the seekers, we have defined two different heuristics. The first, Minimize Total Distance (MTD), minimizes the sum of the number of moves the seeker should make to arrive at each possible location [31]. The second, Chase Actual Location or Chase Assumed Location (CAL), minimizes the number of moves the seeker should make from the target location to the location of the seeker. If this strategy is used by a seeker, the assumed location of the hider is used, because he does not know the actual location of the hider. This assumed location corresponds to the determinization selected at the start of the playout (see Subsection IV-B). Each player should use one heuristic for the hider and one for the seekers. This means that the hider's heuristic has to be combined with one of the two seeker's heuristics. These combinations are named MM (MCD and MTD) and MC (MCD and CAL).

B. Determinization

In order to deal with imperfect information, determinization can be used. This technique is also known as Perfect Information Monte Carlo [47]. The principle behind determinization is that the hidden information is filled in, while being consistent with the history of the game. Despite its theoretical shortcomings [48], [49], it has produced strong results in the past, for example in the trick-based card game bridge [16]. Ginsberg's bridge program, called GIB, uses determinization by dealing the cards that have not been played yet among the players while being consistent with the cards played in the previous tricks. Russell and Norvig [49] call this method 'averaging over clairvoyancy', however GIB can play bridge at an expert level.

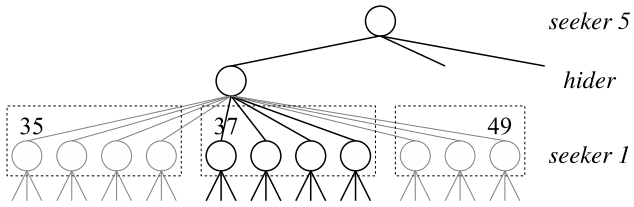

Fig. 3. Example of a determinization with a single tree. In this example, $L = \{35, 37, 49\}$ and the selected determinization is 37.

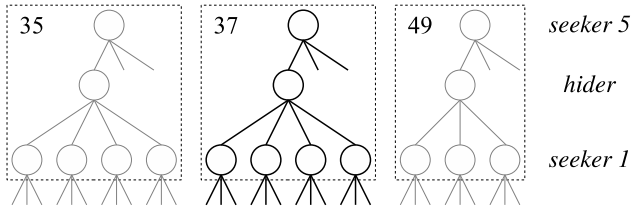

Fig. 4. Example of a determinization with separate trees. In this example, $L = \{35, 37, 49\}$ and the selected determinization is 37.

Other examples of games where determinization is applied to handle imperfect information include Phantom Go [50] and Kriegspiel [20]. Cazenave [50] applied determinization in Phantom Go, creating a Monte-Carlo program that was able to defeat strong human Go players. In Kriegspiel, the application of determinization did not work well; the MCTS-based player with determinization only played slightly better than a random player. Ciancarini and Favini [20] provided three reasons why this technique did not work: 1) the positions of the opponent's pieces were unrealistic, 2) the underestimation of the opponent's ability to coordinate an attack, and 3) in Kriegspiel there is no built-in notion of progress, contrary to games such as Go, Scrabble and Poker.

In Scotland Yard, the hidden information consists of the location of the hider. Based on the last surface location and the tickets the hider played since then, the seekers can deduce a list of possible locations of the hider, called L (see Subsection IV-C).

At the start of each iteration, an assumption is made about the location of the hider. This assumption is used throughout the whole iteration. There are two ways to build and traverse the search tree. The first approach is using single-tree determinization [28]. When generating the tree, at the hider's ply, all possible moves from all possible locations are generated. When traversing the tree, only the moves consistent with the assumption are considered. An example is given in Fig. 3.

The second approach is by generating a separate tree for each determinization [51]. In each tree, only the hider's moves that are consistent with the corresponding determinization are generated. An example is given in Fig. 4. After selecting a determinization at the root node, the corresponding tree is traversed. In the end, there are two approaches to select the best move. The first is majority voting [52]. Each candidate

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

$K \leftarrow L$
$L \leftarrow \emptyset$
if currentRound $\in S$ **then**
  $L \leftarrow$ location(hider)
else
  for all $p \in K$ **do**
    $T \leftarrow$ targets(p, t)
    $L \leftarrow L \cup (T \backslash \Delta)$
  end for
end if
return L

move receives one vote from each tree where it is the move that was played most often. The candidate move with the highest number of votes is selected as the best move. If more moves are tied, the move with the highest number of plays over all trees is selected. The second is averaging over all search trees [53]. The move with the highest average score over all trees is selected as the best move.

C. Limiting the possible locations

It is possible for the seekers to limit the list of possible locations by removing the vertices where the hider cannot be located. The list of possible locations, L, is updated every move. When the hider plays a ticket, the new list of possible locations is calculated, based on the old list of possible locations, the current locations of the seekers Δ, and the ticket t played by the hider, using Algorithm 1. S is the set of rounds when the hider surfaces. At the start of the game, L is initialized with the 18 possible starting locations, excluding 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 vertex he surfaced at. When a seeker makes a move, the target vertex of this move is excluded from L, provided this vertex was a possible location and the hider was not captured.

D. Location Categorization

Some of the possible locations calculated in Algorithm 1 are more probable than others. The performance of the seekers can be improved by biasing the possible locations of the hider. This technique is called Location Categorization [28]. The possible locations in L are divided into categories that are numbered from 1 to c, where c is the number of categories. The type of categorization is game dependent. For Scotland Yard, we investigate three different types of categorization:

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. For this categorization, we set c to 5. To accomodate for locations with a minimum distance larger than 5, all locations with a minimum distance of 5 or more are grouped into the same 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.

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

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

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

Category Combination	1	2	3	4	5
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

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. Similar to the minimum-distance categorization, we set c to 5. This means that all locations with an average distance of 5 or more are grouped into category 5.

Station: A categorization is made based on the transportation types connected to the possible location. We distinguish 4 different station types, which means that c = 4. Locations with only taxi edges belong to category 1, locations with taxi and bus edges belong to category 2, locations with taxi, bus and underground edges belong to category 3, and all locations with at least one boat edge belong to category 4.

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

For each category, a weight has to be determined to indicate the probability that a location of a certain category is chosen. This statistic may be obtained from game records of matches played by expert players. In this article the statistics are gathered by a large number of self-play matches. These statistics can later be used by the seekers to determine the weights of the categories. This approach is useful when the opponent is unknown and there are not sufficient games to gather a sufficient amount of information.

There are two different ways to store the statistics about the possible categories. 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 statistics is similar to the transition probabilities used in Realization Probability Search,

which was successful in Shogi [54], Lines of Action [55], and Amazons [56]. An example of the general table is given in Table I. In the *detailed* table, for each possible combination of categories, i.e. the union of all categories over L, 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 (i.e. rows) that occurred at least 100 times. For instance, category combination (2,3,4), where L contains locations in categories 2, 3 and 4, but not in 1 and 5, occured 152 times, where the hider was 23 times on a location of category 2, 39 times on a location of category 3, and 90 times on a location of category 4.

The seekers use a vector of length c to select a location for the hider at the start of each MCTS iteration. The values in this vector represent the weights of the categories. When using the general table, this vector consists of the values $[\frac{a_1}{n_1}, \frac{a_2}{n_2}, \cdots, \frac{a_c}{n_c}]$. When using the detailed table, the vector corresponding to the combination of categories is directly extracted from the table. If the total number of occurrences of this combination of categories is smaller than a certain threshold, in our MCTS program 100, the table is not used and the possible locations are randomly chosen. This only occurs on rare occasions.

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. The probability to choose location l is calculated by using Formula 3.

$$P(l) = \frac{w_{c_l}}{\sum_{m \in L} w_{c_m}} \tag{3}$$

In this formula, w_{c_l} and w_{c_m} represent the weights of the category to which locations l and m belong, respectively.

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. The probability of choosing location l using two-step selection is calculated by using Formula 4.

$$P(l) = \frac{w_{c_l}}{|c_l| \sum_{j=1}^{c} w_j} \tag{4}$$

In this formula, $|c_l|$ represents the number of possible locations that belong to the category of location l and w_j represent weight of category j.

We remark that Location Categorization uses a 'big-data' approach to set the weights. Such an approach has been successful in Shogi [54], Lines Of Action [55], Amazons [56] and Othello [57]. Of course, machine-learning techniques, though less trivial, could also be used to tune them.

E. Coalition Reduction

Scotland Yard is a cooperative multi-player game. Therefore, the seekers can be considered as one player, making the game essentially a two-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 during game play sometimes relied too much on the other seekers and did not make any efforts to capture the hider. For solving this problem, we propose Coalition Reduction [28]. 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 $r \in [0, 1]$. If the value of r is too small, seekers have the tendency to become less involved. 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 VI-F we experimentally fine-tune this parameter.

F. Move Filtering

The hider only has a limited number of black-fare and double-move 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, and double-move tickets are mostly used for escaping from dire positions.

We implemented some straightforward game-specific knowledge rules regarding the use of black-fare tickets to prevent the hider from squandering them [28]. 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 edges. 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. An exception is when the hider is located on a vertex with a boat connection. In this case, the hider may always use a black-fare ticket.

Double-move tickets are only used when the hider can be captued by one of the seekers. If all possible moves in the root node lead to a vertex which can be reached by one of the detectives in the next round, a double move ticket is added to the moves. If there is at least one 'safe' move, then double-move tickets are not added. If the search algorithm still selects a move which allows one of the seekers to capture the hider in the next round, a double-move ticket is added to the selected move. Of course, a double-move ticket can only be added if the hider has at least one of these tickets left.

In [28] we showed that move filtering is a considerable improvement for the hider, increasing the win rate of an MCTS hider from 19.4% $\pm$ 1.6 to 34.0% $\pm$ 1.9 against MCTS seekers.

V. Minimax-based Techniques for Scotland Yard

In this section, an overview is given how paranoid search and expectimax are used for playing Scotland Yard as the hider and the seekers respectively. In Subsections V-A and V-B we explain how paranoid search and expectimax are implemented for the hider and the seekers, respectively. In Subsection V-C

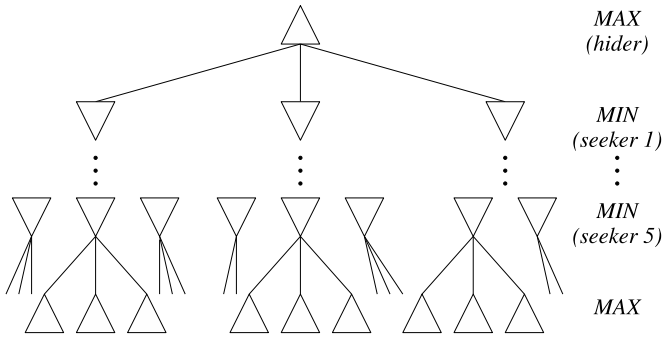

Fig. 5. Structure of the paranoid tree for the hider.

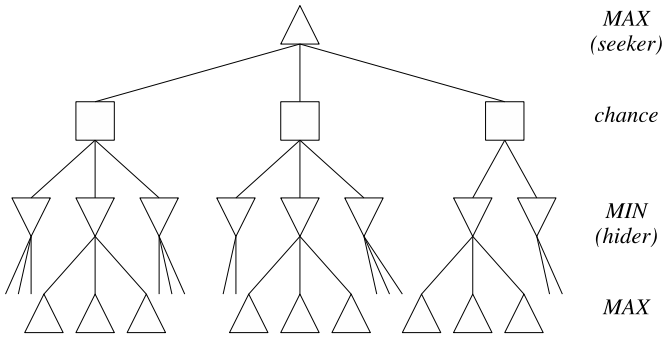

Fig. 6. Structure of the expectimax tree for the seekers.

a description of the evaluation functions is given. Finally, Subsection V-D explains how Location Categorization can be used with expectimax.

A. Paranoid search for the hider

Fig. 5 shows the general structure of the search tree built by the hider. The hider is the MAX player, while the seekers are the MIN players. The structure of the tree is similar to a paranoid search tree used in multi-player games [6]. However, in this case the paranoid assumption is correct because the opponents do have a coalition against the current player. With a paranoid search tree, pruning is possible. However, the tree can only be reduced to $O(b^{\frac{p-1}{p}d})$ in the best case [13], where p is the number of players. In two-player $\alpha\beta$-search, the tree can be reduced to $O(b^{\frac{d}{2}})$ [4], but for Scotland Yard the tree can only be reduced to $O(b^{\frac{5}{6}d})$ in the best case.

B. Expectimax for the seekers

Fig. 6 shows the structure of the expectimax search tree built by the seekers. At the second ply, the chance nodes are located. The edges leaving these nodes represent the possible locations of the hider. Each possible location is weighted equally, though it is possible to bias the weights to give more reliable results (see Subsection V-D). Another notable feature is that the seeker does not incorporate the other seekers in the search tree, i.e. the other seekers do not move. This has three advantages. 1) More pruning is possible. The size of the tree can be reduced to $O(|L| \times b^{\frac{d}{2}})$, where $|L|$ is the number of possible locations. 2) The seeker keeps participating in the game, instead of relying on other seekers. 3) The seeker achieves more long-term planning by investigating more MAX nodes. This is analogous to Best-Reply Search [58] for multi-player games, where the moves of all subsequent opponents are reduced to one ply. A disadvantage of this reduction is that the seekers do not consider the other seekers and thus there is no cooperation. Experiments with 1 second of thinking time per move revealed that reducing the tree produced a win rate of 40.6% $\pm$ 3.0 (see Table VII) against an MCTS hider, while without this reduction the win rate is 5.2% $\pm$ 1.4.

This tree reduction technique can also be applied to MCTS. However, experiments with 10 000 playouts per move showed that it decreases the win rate considerably, from 63.6% $\pm$ 3.0 (see Table V) to 34.3% $\pm$ 2.9 against an MCTS hider. This is because the advantages in expectimax do not apply in MCTS. 1) MCTS is a best-first search technique for which $\alpha\beta$-like pruning is not applicable. 2) Reliance on other detectives is already smaller due to the random moves in the ϵ-greedy playouts. Coalition Reduction reduces this reliance even more. 3) Playouts already cause that the seekers look further ahead.

C. Evaluation Function

For both the hider and the seekers, an evaluation function is necessary to evaluate the leaf nodes of the search tree. For the hider, we use an evaluation function that is based on the MCD playout strategy. First, the hider should stay as far away from the nearest detective as possible. Second, the hider should save black-fare tickets, unless he can increase the uncertainty about his location. The leaf nodes of the paranoid search tree for the hider receive a score by using Formula 5.

$$s_{hider} = 100 \times \min_{i \in D} (d_{hider,i}) + 10 \times t_{hider,BF} + |L| + \rho \quad (5)$$

Here, $d_{hider,i}$ is the distance from the location of the hider to the location of seeker i. $t_{hider,BF}$ represents the number of black fare tickets the hider has left. ρ is a small random value between 0 and 1.

For the seekers, we use an evaluation function that is similar to the MTD playout strategy used in MCTS. The seekers try to minimize the sum of the distances to all possible locations. The leaf nodes of the expectimax search tree of seeker i are evaluated by using Formula 6.

$$s_i = -\sum_{l \in L} d_{i,l} + \rho \quad (6)$$

Here, $d_{i,l}$ is the distance from the location of seeker i to possible location l. Again, ρ is a random value between 0 and 1.

D. Location Categorization

Similarly to MCTS, Location Categorization can be used in the expectimax framework to bias the algorithm towards more likely locations. Usually, in the chance level of the search tree for Scotland Yard, each location has an equal weight. By applying Location Categorization, more likely locations receive a larger weight than unlikely ones. The weight $P(i)$ of

the node representing location i is calculated by using Formula 7.

$$P(i) = w_{c_i} / \sum_{l \in L} w_{c_l} \quad (7)$$

In this formula, w_{c_i} and w_{c_l} represent the weights of the category to which locations i and l belong, respectively. This formula is similar to Formula 3, which is used for one-step selection.

VI. Experiments

In this section we first provide an overview of the experimental setup in Subsection VI-A. In Subsection VI-B we present the results of the experiments with ϵ-greedy playouts for the MCTS players. The determinization techniques for MCTS are compared in Subsection VI-C. In Subsection VI-D we give an overview of the performance of the MCTS seekers with Location Categorization. Next, in Subsection VI-E we show how Location Categorization influences the performance of the expectimax seekers. In Subsection VI-F we present how the MCTS seekers with Coalition Reduction perform. In Subsection VI-G, a comparison between the MCTS and minimax-based players is provided. Finally, in Subsection VI-H we give an overview of how MCTS 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 MCTS-based hider and seekers, C is set to 0.5. Progressive History [34] is used for both the hider and the seekers, with the value of W set to 5 for both player types. These values were achieved by systematic testing. For the hider we use move filtering and for the seekers we use determinization with a single tree. All MCTS players use 10 000 playouts for selecting the best move, except when stated otherwise. The expectimax and paranoid players receive 1 second of thinking time for each move. In all experiments, 1000 games are played to determine the win rate. The win rates are given with a 95% confidence interval. The experiments are run on a cluster consisting of AMD64 Opteron 2.4 GHz processors. Depending on the settings, one game takes approximately 2–4 minutes to finish.

B. ϵ-greedy playouts

In the first set of experiments we determine the influence of ϵ-greedy playouts on the playing strength of the hider and the seekers. Because the different playout strategies have different influences on the number of playouts per second, we limit the thinking time of the players on time instead of samples. Due to the asymmetric nature of Scotland Yard, we also can use different values of ϵ for the hider and the seekers in the playouts. Systematic testing showed that the best results are achieved with ϵ = 0.1 for the hider and ϵ = 0.2 for the seekers. Furthermore, we found that the MM strategy combination works best for the hider and the MC combination works best for the seekers.

TABLE III
Win rates of the MCTS seekers against the MCTS hider with and without ϵ-greedy playouts for different time settings.

Thinking time: 1 second

	hider	
seekers	MM	R
MC	73.5% ± 2.7	78.1% ± 2.6
R	39.7% ± 3.0	49.0% ± 3.1

Thinking time: 2.5 seconds

	hider	
seekers	MM	R
MC	74.9% ± 2.7	79.9% ± 2.5
R	55.8% ± 3.1	59.2% ± 3.0

Thinking time: 5 seconds

	hider	
seekers	MM	R
MC	74.0% ± 2.7	79.5% ± 2.5
R	41.8% ± 3.1	59.8% ± 3.0

In Table III, we present the win rates for the seekers with ϵ-greedy playouts and with random playouts (R) for different time settings (1 second, 2.5 seconds and 5 seconds). The results show that ϵ-greedy playouts are a major improvement for both the hider and the seekers. For example, with a thinking time of 5 seconds, the win rate of the seekers increases from 59.8% ± 3.0 to 79.5% ± 2.5 against the hider with random playouts. For the hider, the win rate increases from 40.2% ± 3.0 to 58.2% ± 3.1 against seekers with random playouts. Similar results are achieved with 2.5 and 5 seconds thinking time. The results also reveal that the seekers have a considerable advantage over the hider, which may be explained by the asymmetric nature of the game.

For the remainder of the experiments, we use the MC strategies for the seekers and the MM strategies for the hider.

C. Determinization

In the previous experiments we applied determinization with a single tree. In this set of experiments, we validate that this technique works better than using a separate tree for each determinization. We perform these experiments in such a way that both player types use either ϵ-greedy playouts or random playouts when competing against each other. Systematic testing showed that for separate trees the same values for C and ϵ are also optimal for the single tree. The results are summarized in Table IV. The upper part of the table shows that single-tree determinization gives the highest win rate with a fixed number of playouts. Especially when using ϵ-greedy playouts, this technique performs considerably better than separate trees. When using separate trees, majority voting performs significantly better than using the average score. We remark that using a single tree generates more overhead, because at the hider's ply, the moves have to be checked whether they are consistent with the selected determinization. This overhead, however, is relatively small. When taking this overhead into account, the difference between single-tree determinization and separate trees hardly changes. This may be concluded from the results presented in the lower part of the table, where the thinking time is limited to 1 second per move, instead of providing a fixed number of playouts.

TABLE IV
WIN RATES OF THE MCTS SEEKERS WITH DIFFERENT DETERMINIZATIONS AGAINST THE MCTS HIDER. BOTH PLAYER TYPES USE EITHER ϵ-GREEDY OR RANDOM PLAYOUTS.

10 000 playouts per move

	Playouts	
Determinization	ϵ-greedy	Random
Single tree	63.6% $\pm$ 3.0	51.8% $\pm$ 3.1
Separate trees + average score	31.3% $\pm$ 2.9	31.2% $\pm$ 2.9
Separate trees + majority voting	35.1% $\pm$ 3.0	37.5% $\pm$ 3.0

1 second per move

	Playouts	
Determinization	ϵ-greedy	Random
Single tree	73.5% $\pm$ 2.7	54.7% $\pm$ 3.1
Separate trees + average score	37.1% $\pm$ 3.0	38.5% $\pm$ 3.0
Separate trees + majority voting	39.9% $\pm$ 3.0	40.1% $\pm$ 3.0

TABLE V
WIN RATES OF THE MCTS SEEKERS WITH LOCATION CATEGORIZATION AGAINST THE MCTS HIDER.

Categorization	Table	Steps	Win rate
Minimum-distance	General	1	67.7% $\pm$ 2.9
Minimum-distance	General	2	66.3% $\pm$ 2.9
Minimum-distance	Detail	1	63.5% $\pm$ 3.0
Minimum-distance	Detail	2	65.6% $\pm$ 2.9
Average-distance	General	1	61.7% $\pm$ 3.0
Average-distance	General	2	59.6% $\pm$ 3.0
Average-distance	Detail	1	63.9% $\pm$ 3.0
Average-distance	Detail	2	63.6% $\pm$ 3.0
Station	General	1	58.6% $\pm$ 3.1
Station	General	2	58.0% $\pm$ 3.1
Station	Detail	1	57.9% $\pm$ 3.1
Station	Detail	2	58.5% $\pm$ 3.1

Default win rate: 63.6% $\pm$ 3.0

D. Location Categorization for MCTS

In the next set of experiments we check which combination of categorization, table type and number of selection steps works best when using Location Categorization. The statistics for the general and detailed table are gathered by letting MCTS seekers play 1000 games against an MCTS hider. The results are summarized in Table V. We let MCTS seekers with Location Categorization play against an MCTS hider. For reference, the seekers without Location Categorization win 63.6% $\pm$ 3.0 of the games against the hider. The win rate of the seekers without Location Categorization is denoted as the default win rate. The results in Table V show that the minimum-distance categorization works best. For this categorization, there is no large difference between the table types and the number of selection steps.

To test the robustness of this technique, we let the MCTS seekers with Location Categorization play against a different type of hider, namely the paranoid hider. We use the same weights that we used against the MCTS hider. Because in the previous set of experiments, it turned out that the minimum-distance categorization works best, we only use this categorization in this set of experiments. The results are given in Table VI. The results show that Location Categorization also significantly improves the performance of the seekers against a different type of opponent. With all settings a significantly better performance is achieved.

TABLE VI
WIN RATES OF THE MCTS SEEKERS WITH LOCATION CATEGORIZATION AGAINST THE PARANOID HIDER.

Categorization	Table	Steps	Win rate
Minimum-distance	General	1	86.4% $\pm$ 2.1
Minimum-distance	General	2	86.3% $\pm$ 2.1
Minimum-distance	Detail	1	87.5% $\pm$ 2.0
Minimum-distance	Detail	2	86.6% $\pm$ 2.1

Default win rate: 83.4% $\pm$ 2.3

TABLE VII
WIN RATES OF THE EXPECTIMAX SEEKERS WITH LOCATION CATEGORIZATION AGAINST THE MCTS HIDER.

Categorization	Table	Win rate
Minimum-distance	General	50.2% $\pm$ 3.1
Minimum-distance	Detail	44.2% $\pm$ 3.1
Average-distance	General	41.4% $\pm$ 3.1
Average-distance	Detail	40.3% $\pm$ 3.0
Station	General	38.2% $\pm$ 3.0
Station	Detail	39.9% $\pm$ 3.0

Default win rate: 40.6% $\pm$ 3.0

E. Location Categorization for expectimax

We also test how Location Categorization increases the performance of the detectives in the expectimax framework. We test this enhancement with the same three categorizations as in the MCTS experiments, including the same weights for the categories. We first let expectimax seekers play against an MCTS hider. The results are given in Table VII. Both player types received 1 second of thinking time. The results show that Location Categorization also works in the expectimax framework. Similar to the MCTS version, the minimum-distance categorization performs best, increasing the win rate against the MCTS hider from 40.6% $\pm$ 3.0 to 50.2% $\pm$ 3.1 when using the general table.

We also test the robustness of Location Categorization in the expectimax framework by testing against a paranoid hider. We remark that the weights of the categories are trained with a different type of seekers against a different type of hider. The results of this set of experiments are displayed in Table VIII. It appears that the detailed table gives better results than the general table. The minimum-distance categorization is still the best categorization.

F. Coalition Reduction

To test the performance of the MCTS seekers with Coalition Reduction, we let seekers play against different hiders with different values of r. For the seekers, we test the performance of Coalition Reduction with and without Location Categorization enabled. We also test against a paranoid hider to verify that this enhancement also works against another type of hider. Finally, we test the performance of the seekers with Coalition Reduction with different time settings. 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 presented in Table IX and Fig. 7. The seekers achieve the highest win rate with $r = 0.1$. The win rate increases from 63.6% $\pm$ 3.0 to 70.1% $\pm$ 2.8. With Location Categorization, the win rate even increases further, from 67.7% $\pm$ 2.9 to 76.2% $\pm$ 2.6. Also with $r = 0.2$ and $r = 0.3$ the seekers

TABLE VIII
WIN RATES OF THE EXPECTIMAX SEEKERS WITH LOCATION CATEGORIZATION AGAINST THE PARANOID HIDER.

Categorization	Table	Win rate
Minimum-distance	General	76.3% ± 2.6
Minimum-distance	Detail	79.1% ± 2.5
Average-distance	General	71.5% ± 2.8
Average-distance	Detail	77.3% ± 2.6
Station	General	69.3% ± 2.9
Station	Detail	65.1% ± 3.0

Default win rate: 74.1% ± 2.7

TABLE IX
WIN RATES OF MCTS SEEKERS WITH COALITION REDUCTION FOR DIFFERENT VALUES OF r AGAINST DIFFERENT HIDERS.

Seekers: / Hider:		MCTS + LC MCTS	MCTS MCTS	MCTS + LC Paranoid	MCTS Paranoid
	0	67.7% ± 2.9	63.6% ± 3.0	87.9% ± 2.0	85.1% ± 2.2
	0.1	76.2% ± 2.6	70.1% ± 2.8	92.9% ± 1.6	90.2% ± 1.8
	0.2	74.2% ± 2.7	65.3% ± 3.0	92.5% ± 1.6	88.0% ± 2.0
	0.3	72.1% ± 2.8	64.3% ± 3.0	91.0% ± 1.8	84.3% ± 2.3
	0.4	64.9% ± 3.0	54.9% ± 3.1	88.9% ± 1.9	82.1% ± 2.4
r	0.5	65.0% ± 3.0	51.0% ± 3.1	86.0% ± 2.2	79.1% ± 2.5
	0.6	52.9% ± 3.1	43.5% ± 3.1	83.0% ± 2.3	72.9% ± 2.8
	0.7	47.5% ± 3.1	42.6% ± 3.1	74.9% ± 2.7	70.4% ± 2.8
	0.8	39.7% ± 3.0	34.4% ± 2.9	69.4% ± 2.9	63.9% ± 3.0
	0.9	32.3% ± 2.9	30.7% ± 2.9	63.2% ± 3.0	57.2% ± 3.1
	1	23.1% ± 2.6	22.6% ± 2.6	50.7% ± 3.1	49.7% ± 3.1

play at least as strong with Coalition Reduction than without this enhancement. If r is increased further, the performance of the seekers drops significantly. With these settings, the detectives no longer cooperate well to strategically close in on the hider, allowing the hider to escape rather easily. If there is no cooperation, the win rate of the detectives drops to 22.6% ± 2.6. To validate these numbers, we also test Coalition Reduction against the paranoid hider. Again, the best results are achieved with r = 0.1, so it turns out that this is a good value that works well against different hiders.

Finally, we test Coalition Reduction with different time settings. Additionally, we give the MCTS seekers with Location Categorization and the MCTS hider 1000, 2500 and 100 000 samples per move. The results are given in Table X and Fig. 8. With 1000 samples per move for the hider and the seekers, better results are achieved with a higher value of r. With r = 0.5, a win rate of 55.6% ± 3.1 is achieved. When providing 2500 samples, the seekers achieve the highest win rate with r = 0.2 and r = 0.3. With these time settings they win 64.5% ± 3.0 and 64.1% ± 3.0 of the games, respectively. If 100 000 samples per move are provided for both player types, Coalition Reduction is still a significant improvement with r = 0.1. The results are similar to 10 000 samples per move. In conclusion, these numbers show that Coalition Reduction increases the performance of the seekers significantly. However, cooperation is still important as the performance decreases if r becomes larger than 0.3. If less time is provided, the value of r should be increased.

G. MCTS vs. minimax-based techniques

In this set of experiments, we compare the MCTS players to the minimax-based players and determine which technique performs best. For the MCTS hider, we use UCT with Progressive History, ϵ-greedy playouts and move filtering. For the

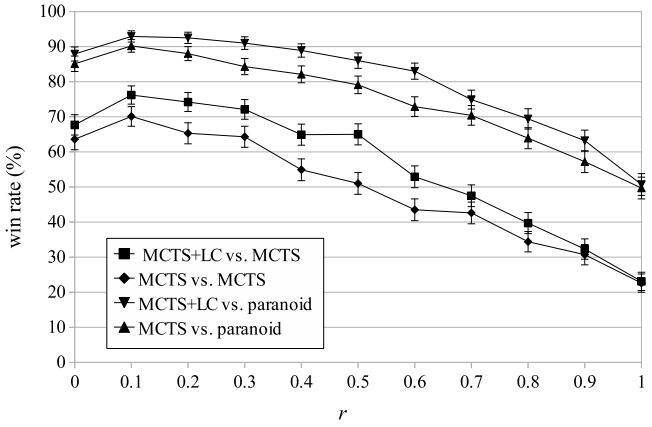

Fig. 7. Graphical representation of Table IX.

TABLE X
WIN RATES OF MCTS SEEKERS WITH COALITION REDUCTION FOR DIFFERENT VALUES OF r WITH DIFFERENT TIME SETTINGS.

Playouts:		1000	2500	10 000	100 000
	0	34.5% ± 3.0	48.3% ± 3.1	67.7% ± 2.9	77.4% ± 2.6
	0.1	43.0% ± 3.1	59.7% ± 3.0	76.2% ± 2.6	83.4% ± 2.3
	0.2	49.6% ± 3.1	64.5% ± 3.0	74.2% ± 2.7	75.3% ± 2.7
	0.3	49.9% ± 3.1	64.1% ± 3.0	72.1% ± 2.8	72.8% ± 2.8
	0.4	55.6% ± 3.1	58.7% ± 3.1	64.9% ± 3.0	68.6% ± 2.9
r	0.5	51.6% ± 3.1	58.5% ± 3.1	65.0% ± 3.0	60.1% ± 3.0
	0.6	48.7% ± 3.1	52.4% ± 3.1	52.9% ± 3.1	56.4% ± 3.1
	0.7	44.1% ± 3.1	50.1% ± 3.1	47.5% ± 3.1	45.8% ± 3.1
	0.8	40.2% ± 3.0	39.7% ± 3.0	39.7% ± 3.0	41.0% ± 3.1
	0.9	35.0% ± 3.0	37.0% ± 3.0	32.3% ± 2.9	32.7% ± 2.9
	1	26.0% ± 2.7	24.3% ± 2.7	23.1% ± 2.6	27.6% ± 2.8

MCTS seekers, we use UCT with Progressive History, single-tree determinization, ϵ-greedy playouts, Coalition Reduction with r = 0.1 and Location Categorization with the minimum-distance general table and 1-step selection. For the paranoid hider, we use killer moves, the history heuristic and move filtering. For the expectimax seekers, we use killer moves, the history heuristic and Location Categorization with the minimum-distance general table. All players receive 1 second of thinking time per move. Against the paranoid hider, the expectimax seekers won 76.2% ± 2.6 of the games and the MCTS seekers won 94.9% ± 1.4. Against the MCTS hider, the expectimax seekers managed to win 45.0% ± 3.1 and the MCTS seekers 81.2% ± 2.4 of the games. Consequently, the paranoid hider won 23.8% ± 2.6 of the games against the expectimax seekers, while the MCTS hider won 55.0% ± 3.1 of the games. Against the MCTS seekers, the paranoid hider won only 5.1% ± 1.4 of the games, while the MCTS hider won 18.8% ± 2.4. The results are summarized in Table XI. These results show that for both the hider and the seekers, MCTS works far better than the minimax-based players.

H. Performance against the Nintendo DS program

To test the strength of the MCTS-based program, it is matched against the Scotland Yard program on the Nintendo DS. The AI of this program is considered to be rather strong [30].

For the hider and the seekers, we use the same settings and enhancements as described in Subsection VI-G. It is not

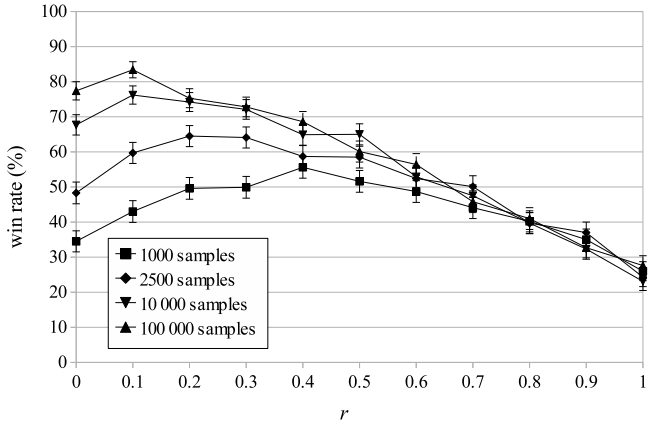

Fig. 8. Graphical representation of Table X.

TABLE XI
WIN RATES OF THE DIFFERENT SEEKERS AGAINST DIFFERENT HIDERS.

	hider	
seekers	MCTS	Paranoid
MCTS	81.2% ± 2.4	94.9% ± 1.4
Expectimax	45.0% ± 3.1	76.2% ± 2.6

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. To have a fair comparison, we set the thinking time of the MCTS program to 2 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, 34 games are won by our program. 23 of these games are won as the seekers and 11 as the hider. The Nintendo DS program wins 16 games, of which 14 as the seekers and 2 as the hider. These results show that the MCTS program plays stronger than the Nintendo DS program.

VII. CONCLUSIONS AND FUTURE RESEARCH

In this article we investigated how MCTS can be applied to play the hide-and-seek game Scotland Yard, how it can be enhanced to improve its performance, and how it compares to minimax-based search techniques.

Using ϵ-greedy playouts to incorporate some basic knowledge into the MCTS algorithm considerably improves the performance of both the seekers and the hider. We observed that using different ϵ values and different playout strategies for the different players in the playouts performs siginificantly better than random playouts.

For handling the imperfect information, we investigated two different determinization techniques, namely single-tree determinization and separate-tree determinization. When using separate trees, majority voting for selecting the best move produces a higher win rate than calculating the average over all trees. Single-tree determinization has a slight overhead, but even when taking this into account, it performs significantly better than using separate trees.

Furthermore, we proposed Location Categorization, a technique that can be used by both the MCTS and the expectimax 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. It significantly increases the playing strength of both the MCTS and the expectimax seekers. Location Categorization proved to be a robust technique, as the learned weights work for both players types against two different types of hider.

We also observed that the performance of the MCTS 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. It also became clear that cooperation is important, because the performance of the seekers drops significantly when the reduction becomes too large. Furthermore, if less playouts per move are provided, better results are achieved with a higher value of r.

In a direct comparison, it turned out that MCTS performs considerably better than paranoid search for the hider and expectimax for the seeker. A comparison between MCTS and minimax-based techniques is not easy because each technique can be enhanced in different ways and the efficiency of the implementations may differ. However, the results do give an idea of the playing strength of the different search techniques and we can also conclude that MCTS is easier to define and optimize than the minimax-based search techniques. 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.

We define three possible directions for future research. The first is to improve Location Categorization. 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 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. A similar technique may also be applied for Stratego to guess the ranks of the opponent's unknown pieces.

The second future research direction is to continue the recent work of Silver and Veness [59], who extended MCTS to Partially Observable Markov Decision Processes (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. With POMDPs, the theoretical shortcomings of determinization can be avoided. However, Scotland Yard may be too complex to model as a POMDP and use it for planning in a reasonable amount of time.

The third possible future research topic is modeling Scotland Yard as a Bounded Horizon Hidden Information Game (BHHIG) [60]. This technique does not have the theoretical

shortcomings of determinization, but it is also slower. A BHHIG can be used for modeling partially observable games in which information is regularly revealed. Teytaud and Flory showed that each BHHIG can be represented as a Game with Simultaneous Actions (GSA) and that the UCT algorithm can be adapted to such games. However, similar to POMDPs, Scotland Yard may be too complex to model as a BHHIG because the maximum number of moves between two fully observable nodes is 30, which may be too large.

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers of this article and the reviewers of our previous contributions at the CIG [28] and the BNAIC [61] for their useful suggestions and constructive criticism. This research is funded by the transnational University Limburg (tUL).

REFERENCES

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

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

[3] C. B. Browne, E. J. 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, 2012.

[4] D. E. Knuth and R. W. Moore, "An Analysis of Alpha-Beta Pruning," *Artificial Intelligence*, vol. 6, no. 4, pp. 293–326, 1975.

[5] 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, 1986, pp. 158–162.

[6] N. R. Sturtevant and R. E. Korf, "On Pruning Techniques for Multi-Player Games," in *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*. AAAI Press / The MIT Press, 2000, pp. 201–207.

[7] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, vol. 47, no. 2, pp. 235–256, 2002.

[8] S. Gelly and Y. Wang, "Exploration Exploitation in Go: UCT for Monte-Carlo Go," in *Neural Information Processing Systems Conference Online trading of Exploration and Exploitation Workshop*, Whistler, BC, Canada, 2006.

[9] 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: ACM, 2007, pp. 273–280.

[10] 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.

[11] 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.

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

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

[14] 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)*, ser. LNCS, H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands, Eds., vol. 5131. Berlin, Germany: Springer-Verlag, 2008, pp. 1–12.

[15] 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.

[16] 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)*, 1999, pp. 584–589.

[17] D. Billings, L. Pena, J. Schaeffer, and D. Szafron, "Using Probabilistic Knowledge and Simulation to Play Poker," in *Proceedings of the Sixteenth National Conference on Artificial Intelligence*. AAAI press, 1999, pp. 697–703.

[18] B. Sheppard, "Towards Perfect Play of Scrabble," Ph.D. dissertation, Department of Computer Science, Universiteit Maastricht, Maastricht, The Netherlands, 2002.

[19] 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*, A. Gerevini, A. Howe, A. Cesta, and I. Refanidis, Eds., 2009, pp. 26–33.

[20] 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: Morgan Kaufmann Publishers Inc., 2009, pp. 474–479.

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

[22] M. Ponsen, G. Gerritsen, and G. M. J.-B. Chaslot, "Integrating Opponent Models with Monte-Carlo Tree Search in Poker," in *Interactive Decision Theory and Game Theory Workshop at AAAI*, vol. 10, 2010, pp. 37–42.

[23] D. Michie, "Game-Playing and Game-Learning Automata," *Advances in Programming and Non-Numerical Computation*, pp. 183–200, 1966.

[24] D. Billings, A. Davidson, T. Schauenberg, N. Burch, M. Bowling, R. Holte, J. Schaeffer, and D. Szafron, "Game-Tree Search with Adaptation in Stochastic Imperfect-Information Games," in *Computers and Games (CG 2004)*, ser. LNCS, H. J. van den Herik, Y. Björnsson, and N. S. Netanyahu, Eds., vol. 3846. Berlin, Germany: Springer-Verlag, 2006, pp. 21–34.

[25] M. P. D. Schadd, M. H. M. Winands, and J. W. H. M. Uiterwijk, "CHANCEPROBCUT: Forward Pruning in Chance Nodes," in *IEEE Symposium on Computational Intelligence and Games (CIG 2009)*, P. L. Lanzi, Ed. IEEE, 2009, pp. 178–185.

[26] M. Adler, H. Räcke, N. Sivadasan, C. Sohler, and B. Vöcking, "Randomized Pursuit-Evasion in Graphs," *Combinatorics, Probability & Computing*, vol. 12, no. 03, pp. 225–244, 2003.

[27] N. Megiddo, S. Hakimi, M. Garey, D. Johnson, and C. Papadimitriou, "The Complexity of Searching a Graph," *Journal of the Association for Computing Machinery*, vol. 35, no. 1, pp. 18–44, 1988.

[28] J. A. M. Nijssen and M. H. M. Winands, "Monte-Carlo Tree Search for the Game of Scotland Yard," in *IEEE Conference on Computational Intelligence and Games*. IEEE, 2011, pp. 158–165.

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

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

[31] 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)*, ser. LNCS, P. Ciancarini and C. Hankin, Eds., vol. 1061. Berlin, Germany: Springer-Verlag, 1996, pp. 416–419.

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

[33] 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.

[34] J. A. M. Nijssen and M. H. M. Winands, "Enhancements for Multi-Player Monte-Carlo Tree Search," in *Computers and Games (CG 2010)*, ser. LNCS, H. J. van den Herik, H. Iida, and A. Plaat, Eds., vol. 6515. Berlin, Germany: Springer-Verlag, 2011, pp. 238–249.

[35] J. Schaeffer, "The history heuristic," *ICCA Journal*, vol. 6, no. 3, pp. 16–19, 1983.

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

[37] S. Gelly, Y. Wang, R. Munos, and O. Teytaud, "Modifications of UCT with Patterns in Monte-Carlo Go," INRIA, Paris, France, Tech. Rep., 2006.

[38] P. Drake, "The Last-Good-Reply Policy for Monte-Carlo Go," *International Computer Games Association Journal*, vol. 32, no. 4, pp. 221–227, 2009.

[39] H. Baier and P. D. Drake, "The Power of Forgetting: Improving the Last-Good-Reply Policy in Monte Carlo Go," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 2, no. 4, pp. 303–309, 2010.

[40] M. J. W. Tak, M. H. M. Winands, and Y. Bjornsson, "N-Grams and the Last-Good-Reply Policy Applied in General Game Playing," *Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 73–83, 2012.

[41] A. Rimmel, F. Teytaud, and O. Teytaud, "Biasing Monte-Carlo Simulations through RAVE Values," in *Computers and Games (CG 2010)*, ser. LNCS, H. J. van den Herik, H. Iida, and A. Plaat, Eds., vol. 6515. Berlin, Germany: Springer-Verlag, 2011, pp. 59–68.

[42] J. von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior*. Princeton, NJ, USA: Princeton University Press, 1944.

[43] F.-H. Hsu, *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton, NJ, USA: Princeton University Press, 2002.

[44] S. G. Akl and M. M. Newborn, "The Principal Continuation and the Killer Heuristic," in *1977 ACM Annual Conference Proceedings*. ACM Press, New York, NY, USA, 1977, pp. 466–473.

[45] D. Slate and L. Atkin, *Chess Skill in Man and Machine*. Berlin, Germany: Springer-Verlag, 1977, ch. 4. CHESS 4.5 – Northwestern University Chess Program, pp. 82–118.

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

[47] J. R. Long, N. R. Sturtevant, M. Buro, and T. Furtak, "Understanding the Success of Perfect Information Monte Carlo Sampling in Game Tree Search," in *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence*, M. Fox and D. Poole, Eds. AAAI press, 2010, pp. 134–140.

[48] I. Frank and D. Basin, "Search in Games with Incomplete Information: A Case Study using Bridge Card Play," *Artificial Intelligence*, vol. 100, no. 1-2, pp. 87–123, 1998.

[49] S. J. Russell and P. Norvig, *Artificial Intelligence, A Modern Approach*. Englewood Cliffs, NJ, USA: Prentice-Hall, 2002.

[50] T. Cazenave, "A phantom go program," in *Advances in Computer Games (ACG11)*, ser. LNCS, H. J. van den Herik, S.-C. Hsu, T.-S. Hsu, and H. H. L. M. Donkers, Eds., vol. 4250. Berlin, Germany: Springer-Verlag, 2006, pp. 120–125.

[51] D. Whitehouse, E. J. Powley, and P. I. Cowling, "Determinization and Information Set Monte Carlo Tree Search for the Card Game Dou Di Zhu," in *IEEE Conference on Computational Intelligence and Games*. IEEE, 2011, pp. 87–94.

[52] Y. Soejima, A. Kishimoto, and O. Watanabe, "Evaluating Root Parallelization in Go," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 278–287, 2010.

[53] T. Cazenave and N. Jouandeau, "On the Parallelization of UCT," in *Proceedings of the Computer Games Workshop*, H. J. van den Herik, J. W. H. M. Uiterwijk, M. H. M. Winands, and M. P. D. Schadd, Eds., 2007, pp. 93–101.

[54] 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.

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

[56] 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)*, ser. LNCS, H. J. van den Herik, S.-C. Hsu, T.-S. Hsu, and H. H. L. M. Donkers, Eds., vol. 4250. Berlin, Germany: Springer-Verlag, 2006, pp. 73–87.

[57] 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., Maastricht, The Netherlands, 2000, pp. 77–96.

[58] M. P. D. Schadd and M. H. M. Winands, "Best Reply Search for Multiplayer Games," *Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 1, pp. 57–66, 2011.

[59] 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. S. Zemel, and A. Culotta, Eds., 2010, pp. 2164–2172.

[60] O. Teytaud and S. Flory, "Upper Confidence Trees with Short Term Partial Information," in *Applications of Evolutionary Computation*, ser. LNCS, C. Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekrt, A. Esparcia-Alczar, J. Merelo, F. Neri, M. Preuss, H. Richter, J. Togelius, and G. Yannakakis, Eds. Berlin, Germany: Springer-Verlag, 2011, vol. 6624, pp. 153–162.

[61] J. A. M. Nijssen and M. H. M. Winands, "Monte-Carlo Tree Search for the Game of Scotland Yard," in *Proceedings of the 23rd Benelux Conference on Artificial Intelligence*, P. de Causmaecker, J. Maervoet, T. Messelis, K. Verbeeck, and T. Vermeulen, Eds., Ghent, Belgium, 2011, pp. 417–418, extended abstract.

J. (Pim) A. M. Nijssen received the M.Sc. degree in Artificial Intelligence from the Department of Knowledge Engineering, Maastricht University, Maastricht, The Netherlands, in 2009. Currently, he is working on his Ph.D. degree in Artificial Intelligence at the Department of Knowledge Engineering, Maastricht University, Maastricht, The Netherlands. His research covers the use of Monte-Carlo Tree Search in multi-player games and hide-and-seek games.

Mark H. M. Winands received the Ph.D. degree in Artificial Intelligence from the Department of Computer Science, Maastricht University, Maastricht, The Netherlands, in 2004. Currently, he is an Assistant Professor at the Department of Knowledge Engineering, Maastricht University. His research interests include heuristic search, machine learning and games. Dr. Winands serves as a section editor of the ICGA Journal and as an associate editor of IEEE Transactions on Computational Intelligence and AI in Games.