Application of UCT Search to the Connection Games of Hex, Y, *Star, and Renkula!

Tapani Raiko[⋆] and Jaakko Peltonen[⋆∗]

[⋆]Helsinki University of Technology,
Adaptive Informatics Research Centre,
P.O. Box 5400, FI-02015 TKK, Finland
`firstname.lastname@tkk.fi`

Abstract

Play-out analysis has proved a succesful approach for artificial intelligence (AI) in many board games. The idea is to play numerous times from the current state to the end, with randomness in each play-out; a good next move is then chosen by analyzing the set of play-outs and their outcomes. In this paper we apply play-out analysis to so-called 'connection games', abstract board games where connectivity of pieces is important. In this class of games, evaluating the game state is difficult and standard alpha-beta search based AI does not work well. Instead, we use UCT search, a play-out analysis method where the first moves in the lookahead tree are seen as multi-armed bandit problems and the rest of the play-out is played randomly using heuristics. We demonstrate the effectiveness of UCT in four different connection games, including a novel game called Renkula!.

1 Introduction

Many typical board game artificial intelligences are based on alpha-beta search, where it is crucial to evaluate the strength of a player's position at the leaves of a lookahead tree. Such approaches work reasonably well in games with small board sizes, especially if the worth of each game piece can be evaluated depending only on a few other pieces. Alpha-beta search works well enough as a starting point in for instance chess.

In several games, however, precisely evaluating the game state is difficult except for states very close to the end of the game. The idea in play-out analysis is that, instead of trying to evaluate a state on its own merits, the state is used as a starting point for (partly) random play-outs, each of which can finally be given a simple win-or-lose evaluation.

Play-out analysis is especially attractive for so-called *connection games*, because several such games have an interesting property: boards only get more full as the game progresses, and any completely filled board is a winning state for one of the players.

UCT search[1] (Kocsis and Szepesvari, 2006) is a recently introduced play-out method that has been applied successfully in the game of Go in Gelly et al. (2006). In this paper we apply UCT to create an AI for four different connection games.

2 Games

The term *connection games* (Browne, 2005) denotes a class of abstract board games where connectivity of game pieces is crucial. The connection games discussed in this paper share the basic rules and properties discussed below.

Starting from an empty board, two players alternately place pieces (also called stones) of their own color to empty points. There is only one kind of game piece, pieces never move, and pieces are never removed; that is, the only action is to place new pieces on the board. When the board has been completely filled up, the winner can be determined by checking which player has satisfied a winning criterion. In these games, the winning criterion has been designed so that in any filled up board, *one (and only one) player must have succeeded* (we discuss the details for each game later in the paper). As a result, players cannot succeed by pursuing their own goals in separate areas of the game board; to succeed and to stop the other player from succeeding are equivalent goals.

It is easy to show by the well-known *strategy stealing argument* that the first player to move has a winning strategy. Therefore often a so-called *swap rule* is used where after the first move has been played, the

[∗]J. Peltonen also belongs to Helsinki Institute for Information Technology.

[1]The acronym UCT was not written out in Kocsis and Szepesvari (2006); one possible way to write it out could be 'Upper Confidence bounds applied to Trees'.

second player may choose to switch colors. When the swap rule is used, it is not in the first player's interest to select an overly strong starting move, and the game becomes more balanced.

The games are differentiated by their goal (winning criterion), and by the shape of the game board. The goals of each game are described in Subsections 2.1, 2.2, 2.3, and 2.4. Each game is played on a board of a certain *shape*, but the *size* of the board can be varied (this would further hinder several typical AI approaches). There are two equivalent representations for boards: **(1)** the board is built from (mostly) triangles, stones are played at the end points, and points that share an edge are connected; or **(2)** the board is built from (mostly) hexagons, stones are played in the hexagons, and neighbouring polygons are connected.

We stress that even though the rules are simple at first sight (just place stones in empty places), actual gameplay in connection games can become very complex; typical concepts include *bamboo joints*, *ladders*, and maximizing the *reach* of game pieces.

2.1 Hex

The game of Hex (Figure 1) is one of the oldest connection games; it was invented by Piet Hein in 1942 and independently by John Nash. Hex is one of the games played in the Computer Games Olympiad.

The Hex board is diamond-shaped. The black player tries to connect the top and the bottom edges with an unbroken chain, while the white player tries to connect the left and the right edges. When the board is full of stones, either the black groups that reach the bottom edge reach also the top edge, or they are completely surrounded by white stones that connect the left and right edges. Therefore one player must win. Further information about the history, complexity, strategy, and AI in Hex can be found in Maarup (2005).

2.2 Y

The game of Y (Figure 2) was invented by Claude Shannon in the early 1950s and independently by Craige Schensted (now Ea Ea) and Charles Titus. It can be played on a regular triangle or a bent one (introduced by Schensted and Titus). The reason for the two boards is that on the regular board, the center is very important and the outcome of the game is often determined on this small part. The bent board from is more balanced in that sense; we use the bent board.

Both players try to connect all three edges of the board with a single unbroken chain of the player's

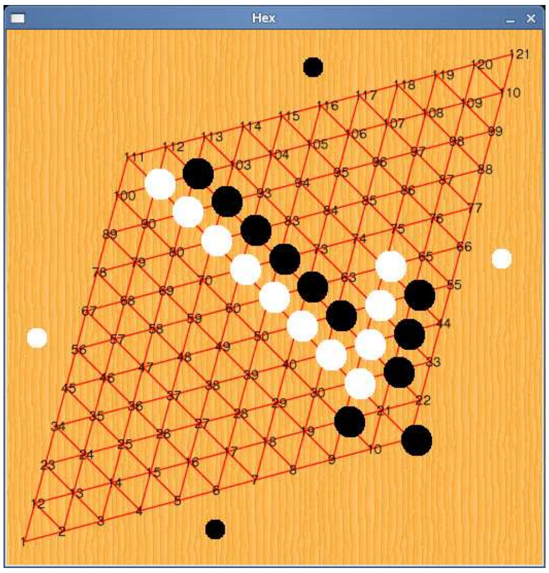

Figure 1: Game of Hex. The black player tries to connect the top and the bottom edges with an unbroken chain, while the white player tries to connect the left and the right edges. Note that any corner point between two edges belongs to both edges; the same applies also to Y and *Star. (Numbers on the board enumerate the allowed play positions, and circles outside the board clarify the target edges of each player.)

own color; this chain often has a 'Y' shape. The fact that one player must win follows from the so called Sperner's lemma or from micro reductions (van Rijswijck, 2002).

2.3 *Star

The game of *Star (Figure 3) was invented by Ea Ea. The intention behind the 'bent pentagon' shape of the board is again to balance the influence of the center and edges. *Star is closely related to the well-known game Go: in Go the goal is to gather more territory than the opponent, and survival of a group is often achieved by connecting it to another one.

In *Star the winner is evaluated by counting scores for the players. Each node on the perimeter of the board counts as one so-called *peri*. In the evalution process, connected groups of one color that contain fewer than two peries are not counted as groups of their own; instead, the possible peri goes to the surrounding group. Each remaining group is worth the number of peries it contains minus four. The player with more points wins. Draws are decided in favour of the player owning more corners. By construction,

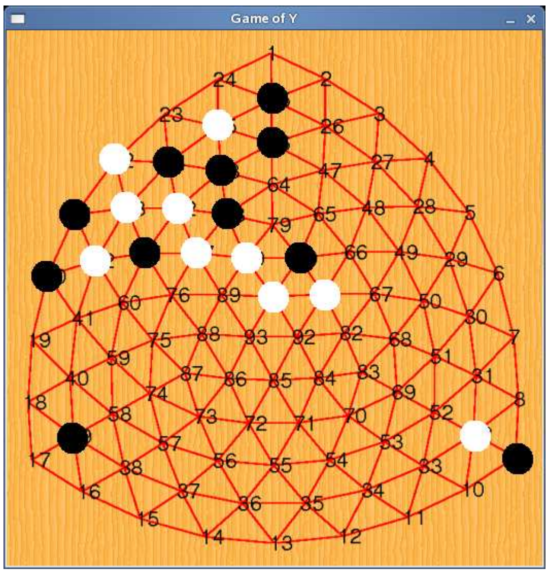

Figure 2: Game of Y. Both players try to connect all three edges of the board with a single unbroken chain of the player's own color.

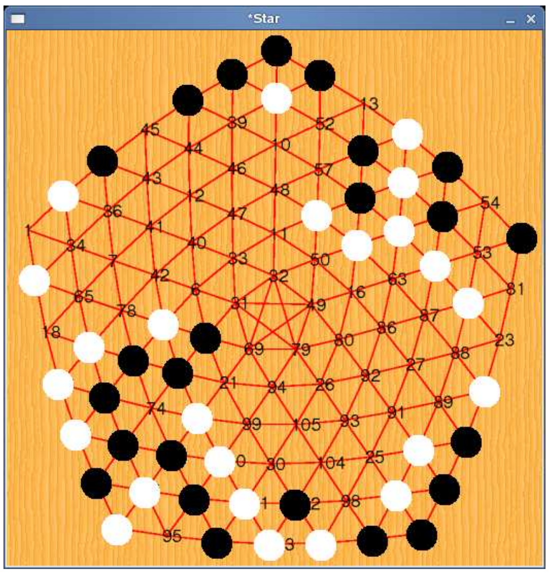

Figure 3: Game of *Star. Each node on the perimeter of the board counts as one 'peri'. Connected groups of one color that contain fewer than two peries are not counted as groups of their own; instead, the possible peri goes to the surrounding group. Each remaining group is worth the number of peries it contains minus four. The player with more points wins. Draws are decided in favour of the player owning more corners.

one of the players must win.

2.4 Renkula!

The game of Renkula! (Figure 4) was invented by Tapani Raiko in 2007 and is first published in this paper. It is played on a surface of a *geodesic sphere* formed from 12 pentagons and a varying number of hexagons. The dual representation with triangles can be made by taking an icosahedron and dividing each edge into n parts and each triangle to n^2 parts; the current software implementation provides four boards using $n = 2, 3, 4, 6$.

Red and blue players get turns alternately, starting with the red player. The player whose turn it is, selects an empty polygon to place a stone of his/her color. Another stone of the same color will be automatically placed in the polygon on the exact opposite side of the sphere. The player who manages to connect any such pair of opposite stones with an unbroken chain of stones of his/her color, wins (see Figure 5 for an example). Note that in contrast to the other games, Renkula! does not have any edges or pre-picked directions; connecting any pair of opposite stones suffices to win.

A winning chain always forms a loop around the sphere (typically the loop is very 'wavy' rather than straight). If you connect two poles of the sphere with a chain, the opposite stones of the chain complete the loop on the other side. The name Renkula!, coined by Jaakko Peltonen, refers to this property: 'renkula' is a Finnish word meaning a circular thing.

Like the previous three games, Renkula! also has the property that a filled-up board is a win for one and only one of the players. We briefly sketch the proof.

If one player has formed a winning chain, the other player could no longer form a winning chain even if the game was continued: the winning loop divides the rest of the sphere's surface into two separate areas. Each of the opponent's chains is restricted to one of those areas and can never reach the opposite area.

When the sphere is filled with stones, one of the players must have made a winning chain. Consider any red pair of opposite stones A and B on a sphere filled with stones. If they are connected to each other, red has won. Otherwise there are two separate red chains: C_A which includes at least A, and C_B which includes at least B. Because the chains are separate, there must be a loop of blue stones around the area that C_A reaches, and similarly for C_B. These blue loops are each other's opposites, so if they are connected, blue has won. If the blue loops are not connected, there must be red loops between the blue loops at the edge of what blue reaches. Because there is only a finite amount of polygons on the sphere, this recursion cannot continue indefinitely. Therefore one of the players must have won.

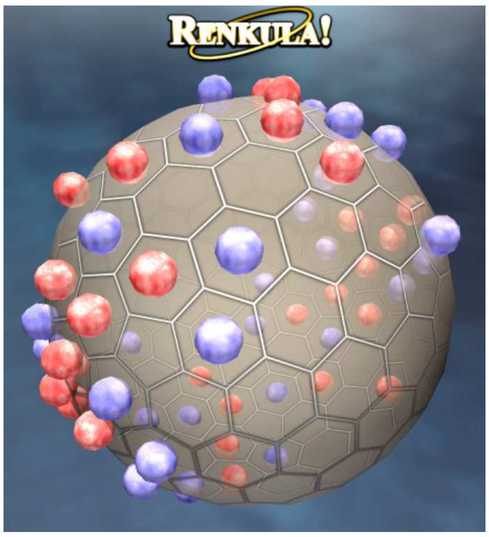

Figure 4: Game of Renkula!. Stones are placed as pairs at exact opposite sides of the sphere. The player whose stones connect any such pair with an unbroken chain, wins. Unlike the other game boards, the spherical Renkula! boards do not have edge points.

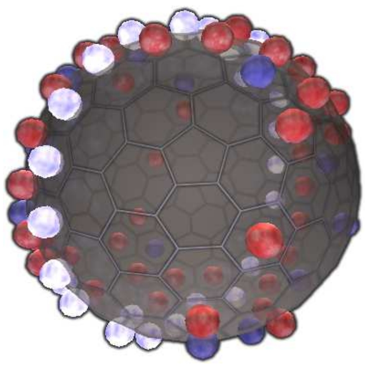

Figure 5: Blue has won a game of Renkula! with the highlighted chain.

3 AI based on UCT search

The UCT search (Kocsis and Szepesvari, 2006) is a tree search where only random samples are available as an evaluation of the states. The tree is kept in memory and grown little by little. The sample evaluations are done by playing the game to the end from the current state. In this paper a 'state' is a configuration of pieces on the board, and an 'action' is the placement of a new piece somewhere on the board.

At the start of the game, the tree contains only the root (initial game state), and leaves which are the new possible actions. To improve the tree, at each turn numerous play-outs are carried out from the current state to the end of the game. In each play-out, there are two ways to choose the move, as follows.

If the play-out is still in a known state (a state that already exists as a non-leaf node within the UCT tree), the actions are chosen using the highest *upper confidence bounds* on the *expected action value*. To compute the bounds, the following counts are collected: how many times $n(s)$ the state s has been visited in the search, how many times $n(s, a)$ action a was selected in state s, and what has been the average final reward $r(s, a)$ from each action. Assuming that the final rewards are binary (win/loss; this is the case in the four games of this paper), the upper confidence bound (Auer et al., 2002) becomes

$$u(s, a) = r(s, a) + c\sqrt{\frac{\log n(s)}{n(s, a)}}, \qquad (1)$$

where c is a constant that determines the balance between exploration and exploitation (see Auer et al., 2002, for discussion). We simply use $c = 1$. Note that if an action has never been chosen, the bound u becomes infinitely high and such actions are always tried out first.

When the play-out reaches a leaf node of the UCT tree, a new node is added to the tree. Thus the number of nodes in the tree equals the number of play-outs. The rest of the play-out is made using random moves either from a uniform random distribution or by some *heuristics*; we describe useful heuristics in the next section. Note that in this way the play-outs balance randomness and known information: the known confidence bounds determine the first steps of each play-out, and randomness is then used when known information no longer available. Each play-out refines the UCT tree by adding new nodes and by influencing the counts $n(s)$ and $n(s, a)$ and the values $r(s, a)$.

After the play-outs have been carried out, the move a having the highest play-out count $n(s, a)$ for the current state s is chosen. As the game goes on, the tree does not need to be reset; new play-outs could simply be carried out from the whatever state the game is currently at. (In the current implementation the tree is forgotten after each move.)

3.1 Heuristics for Connection Games

Here we describe novel heuristics and speed-ups for UCT suitable for connection games.

Speed-up 1: Suppose a play-out reaches a leaf node of the UCT tree; typically there will be numerous empty positions left on the board. In all of the presented games, assuming uniformly random play-outs, it is easy to show the empty positions end up filled with *random colored stones*, an equal number of each color. This 'fill-out' does not need to be done move by move: it is faster to simply go through the board once, filling all the empty points.

Speed-up 2: Suppose we are initializing $r(s, a)$ for the latest leaf node. It does not make any difference which of the above-described 'fill-out moves' is counted as the first one a. Therefore, assuming it is e.g. black's move, $r(s, a)$ for all filled in black stones a can be updated as if they were the next move.

Heuristic 1: As the random fill-out phase is fast, it can be useful to do more than one fill-out at once.

Heuristic 2: We consider so-called *bamboo connections*, also known as *bridges*, as a special case. Bamboo connections are a simple shape that reappears very often in any of the presented games. Figure 6 shows an example in the game of Renkula! but more generally they can also occur between a stone and the edge of the board. To break a bamboo connection, both empty positions in the connection must become filled up with stones of the other player. Using uniformly random playouts, there are four ways to fill the empty positions, and the connection is broken in one of these four cases. It is only rarely useful for a player to let his/her bamboo connection get broken, and it is rarely useful to fill both empty positions in a bamboo connection with your own stones; therefore, a useful heuristic is to recognise bamboo connections in the fill-out phase, and fill them with one stone of each color, thus avoiding the above-described two undesirable fill-out cases. The only exception is when different bamboo connections overlap. In those cases we acknowledge only the first one found.

Given the above-described improvements, the behavior of the resulting connection game AI can be adjusted by adjusting the number of play-outs carried out at each turn, the number of random fill-outs performed at the leaf nodes, and whether to use the bamboo connection heuristic. Larger numbers of play-outs and fill-outs obviously slow down the AI. A useful property of the AI is that it is possible to stop the search at any time, and simply select a move based on the current evaluations (this ability is in principle available for all the games; currently we have implemented it in Renkula! but not in the other games).

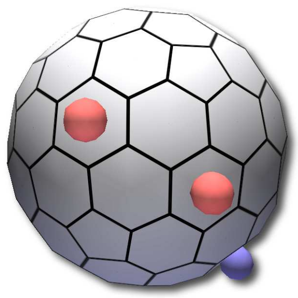
Figure 6: A bamboo connection, here shown on a Renkula! board. Blue player cannot prevent red from connecting its stones.

4 Conclusion

We have presented a UCT search based AI for four connection games, including a new game introduced here. Our implementations of the game AIs are freely available: the implementations of Hex, Y, and *Star are available at `www.cis.hut.fi/praiko/connectiongames/`, and the implementation of Renkula! is available at `www.nbl.fi/~nbl924/renkula/`. As a subjective evaluation, the algorithm seems to be quite strong at least on small boards.

References

P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multi-armed bandit problem. *Machine Learning*, (47):235–256, 2002.

Cameron Browne. *Connection Games: Variations on a Theme*. A K Peters, Ltd., 2005.

Sylvain Gelly, Yizao Wang, Rémi Munos, and Olivier Teytaud. Modification of UCT with patterns in Monte-Carlo Go. Technical Report RR-6062, 2006. `http://hal.inria.fr/inria-00117266`.

L. Kocsis and C. Szepesvari. Bandit based Monte-Carlo planning. In *Proc. of European Conference on Machine Learning*, pages 282–293, 2006.

Thomas Maarup. Hex: Everything you always wanted to know about hex but were too afraid to ask, 2005.

Jack van Rijswijck. Search and evaluation in Hex. Technical report, Department of Computing Science, University of Alberta, 2002.