Search and evaluation in Hex

Jack van Rijswijck

Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada T6G 2H1
javhar@cs.ualberta.ca

Abstract. The state of the art in Hex playing programs around 2002 is that computers can play perfectly on positions up to board size 6×6 and are on par with the best human players on board sizes up to about 9×9. This report describes currently used and proposed approaches to game tree search and heuristic evaluation functions.

1 Introduction

For an excellent introduction to the game of Hex and its strategy, see Browne's book [Bro00]. Introductory articles on Hex appeared in Scientific American by Gardner [Gar59] and Stewart [Ste00]. Proofs of the no-draw property of Hex are included in articles by Gale [Gal86] and Beck [BBC69]. The PSPACE-completeness of a generalized version of Hex was proved by Even and Tarjan [ET76]; the proof for Hex itself was supplied by Reisch [Rei81].

The algorithmic approaches to playing Hex described in this report can be summarized as follows:

Virtual connections are used in Vadim Anshelevich's program *Hexy* [Ans00];
Decomposition patterns are used by Jing Yang to prove some 7×7 and 8×8 opening values [YLP01,YLP02a,YLP02b];
Pattern search is based on Yang's method and was tested but not yet used in Jack van Rijswijck's program *Queenbee* [Rij00];
Network flow models have been proposed in several forms, notably in Hexy where an electric network is simulated;
Graph distance is used in Queenbee;
Y-Reduction was suggested by Steven Meyers based on observations by Craig Schensted [Mey02].

The first three methods are explained in Section 2, while Section 3 describes the network flow and two-distance methods. The Y reduction approach is presented in Section 4. Appendix A includes some background on the graph representation of Hex.

2 Search

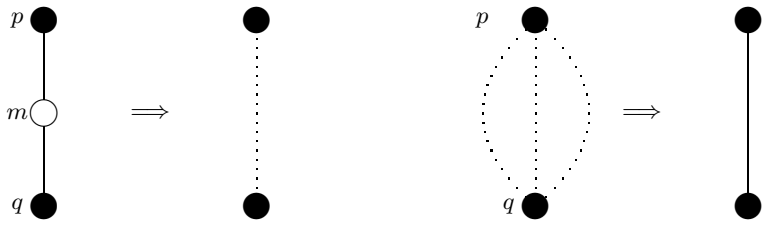
Neither virtual connections nor decomposition patterns are themselves game tree search methods. Both methods use local patterns of proven connections, building up new patterns from smaller ones. A variant of decomposition patterns using global patterns is used as a safe game tree search enhancement in pattern search.

2.1 Virtual Connections

Virtual connections are local patterns that guarantee a connection. There are two types of virtual connections: weak and strong. A *weak connection*, built by the *And rule*, is a guaranteed win providing the winning player plays first. A *strong connection*, built with the *Or rule*, is a win no matter who plays first. Each connection has a *carrier*, which is the set of cells required to be empty for the connection to work.

The And rule is depicted in Figure 1-I. It establishes a weak connection between p and q provided that the intermediate node m is empty and there are strong connections between p and m and between m and q. The connection can be made by playing at m. The And rule requires the two carriers not to overlap; the carrier of the resulting weak connection is the union of these two carriers plus the cell m.

Figure 1-II shows the application of the Or rule. A strong connection between p and q exists if there are two or more weak connections between p and q, provided that the carriers of these connections have an empty total intersection. This ensures that the opponent cannot block all weak connections at once; the connection can then be secured by strengthening one of the unaffected weak connections. The carrier of the resulting strong connection is the union of the carriers of the weak connections.



I: And rule
two strong connections form a weak one

II: Or rule
several weak connections form a strong one

Fig. 1. The *And rule* and the *Or rule*

One drawback of virtual connections is that they are *incomplete*, in the sense that there are examples of positions which cannot be proved with the And-Or

rules. Figure 2 gives an example, based on one given in [Ans00]. The position is a weak virtual connection between p and q, where m is a winning move. The virtual connection method would aim to prove the connection by finding strong virtual connections between p and m and between m and q, which fails in this case because there is no strong virtual connection betweem p and m.[1] The same story goes for the only other winning move, the symmetrically equivalent n.

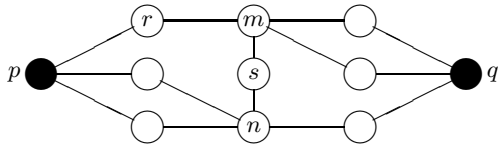


Fig. 2. A weak virtual connection that cannot be proved with the And-Or rules

The reason that the virtual connection method cannot prove this position is that the And rule contains the implicit assumption that the winning connection is going to use the intermediate point m. In this case, play proceeds with the opponnent being forced to occupy r. The unique winning reply is n. If the opponent then plays s, the connection can be established but it does not include node m. This violates the hidden assumption of the And rule, which is the reason why it cannot be discovered by virtual connection search.

2.2 Decomposition Patterns

Decomposition patters are quite similar to virtual connections; they, too, are built up from smaller patterns and ensure a connection between two groups of pieces with a pattern of cells that need to be empty for this connection to work. Figure 3 shows an example, where groups b_1 and b_2 have a guaranteed connection. The connection uses the smaller pattern A which connects a_1 to a_2. The recipe for the connection between a and b is as follows:

- if the opponent plays at one of [1, 3, 4, 7, 8], then reply at 2, and use pattern A between 2 and b_2;
- if the opponent plays at one of [2, 5, 6], then reply at 4, and use pattern A between b_1 and 4 and pattern A between 4 and b_2.

One important difference between virtual connections and decomposition patterns is that the latter can contain non-empty cells. An example is shown in Figure 4. The black cell c_1 in the middle is connected to the group c_2 on the right using the group of two cells on the left.

[1] Strictly speaking there *is* a strong connection between p and m, but it necessarily uses all of the empty nodes, and thus it will always overlap with any strong connection between m and q.

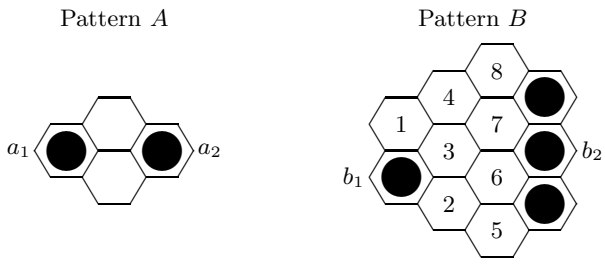


Fig. 3. Pattern B decomposes into several instances of smaller pattern A

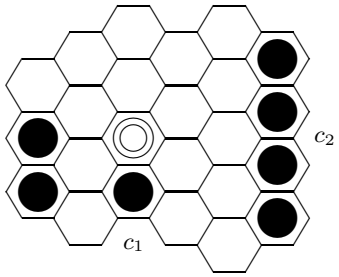


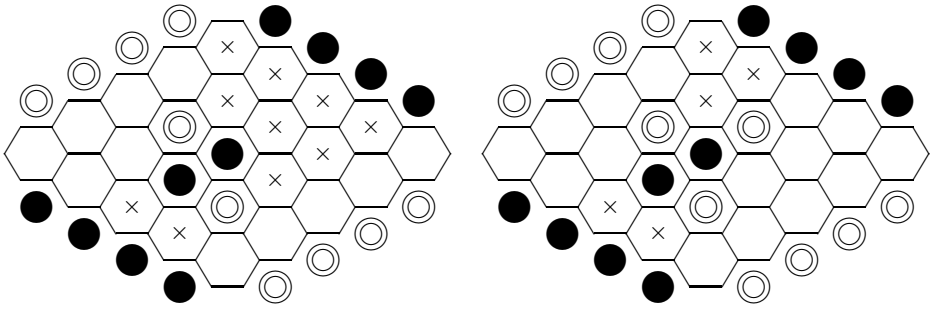
Fig. 4. Pattern C contains nonempty cells

Decomposition patterns can any Hex position. The decomposition patterns are remarkably efficient, enabling Jing Yang to prove some 7×7 opening moves using just a few hundred patterns where a game tree search would need trillions of nodes. Unfortunately no automated algorithm for building up an efficient library of decomposition patterns has been devised yet; Yang's proofs are built by hand and subsequently checked by computer.

2.3 Pattern Search

Pattern search is inspired by decomposition patterns, but it uses global patterns instead of local ones. It is an enhancement of game tree search. Figure 5 shows a board position where White to move is lost. The loss only depends on the cells marked '×': even if White were to occupy all the unmarked cells on the board, Black would still win. The collection of ×-cells is the *threat pattern* that proves the loss.

If White plays as in Figure 5-II, the resulting position is a win for Black. The threat pattern indicated in the diagram is all that Black needs for the win, and therefore it proves that all unmarked cells in 5-II were also losing moves for White in the previous position. It thus refutes 15 additional moves at once, improving the standard game tree search in which each of those moves would have to be refuted independently.



I: White to move loses II: Black to move wins

Fig. 5. Only the cells marked '×' are relevant

Formally, a collection Ψ of empty cells in a Hex position P is a threat pattern if the result of the game is unaltered even when the losing side occupies all the empty cells not in Ψ. A threat pattern is not unique to a position, since adding an empty cell to a threat pattern always creates another valid threat pattern. There exists a threat pattern in every position, since the pattern that consist of all the empty cells is always trivially valid.

In a position where the game is over, the threat pattern is the empty set. In any other position P a pattern can be calculated recursively:

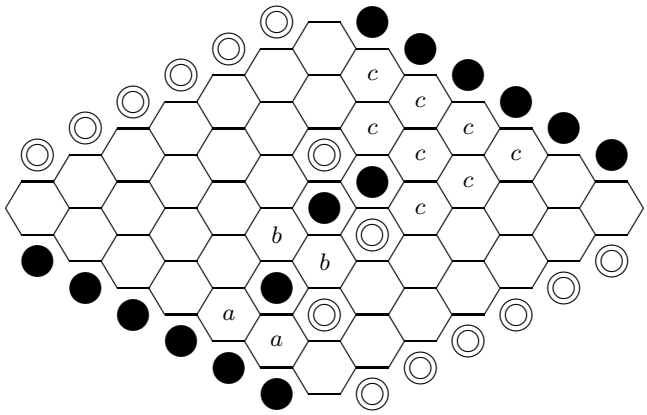
- if there is a winning move m, then P is a win and the threat pattern consists of m along with the losing threat pattern of the resulting position;
- if there is no winning move, then there is a collection of winning threat patterns for the opponent that have an empty total intersection, and the union of these patterns forms the losing threat pattern for P.

These two pattern rules are analogous to the And and Or rules of virtual connections.

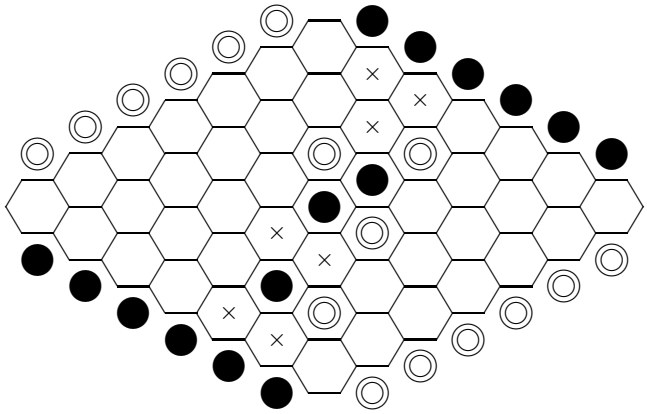
2.4 Decomposition pattern search

Threat patterns combine the advantages of being automated and complete, but they are less efficient because the patterns are not local. This is not a problem in a winning position, since only one winning move needs to be tried. But it is a problem in a losing position such as the one in Figure 6-I. The losing pattern contains three independent local connections. Whenever White plays in one of the three local connections, Black replies in the same one. Any move by White outside of the threat pattern is irrelevant, and Black can afford to skip a move in reply. In effect, all the cells outside of the threat pattern are "dead cells", and the winning player need never to play in any of them.

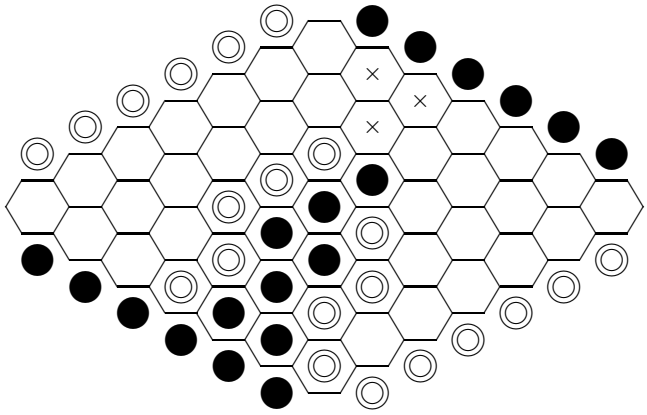
Each of the local connections could be proved independently. Pattern search does not recognize this; for each move that White tries in region c, the subsequent search proves the connections at a and b again from scratch.



I: White to move loses because of three independent local patterns



II: White tries one reply



III: Conjecturing away the seemingly unrelated parts of the pattern

Fig. 6. Checking the conjectured local connections

There may be a way to make pattern search treat the local patterns independently. When White tries the move in Figure 6-II, the search returns a win for Black with the indicated pattern. White now notices that this pattern consists of three groups of cells. Two of those groups were not touched by the latest white move played. So White may now conjecture the following: *If* the position is indeed a loss, *and* the losing threat pattern consists of independent sub-patterns, *then* the latest move played only interfered with one of those sub-patterns.

Let the *interfered* groups be those groups of the pattern in 6-II that are adjacent to White's last move, and call the other groups *untouched*. If the conjecture is true, then the untouched groups are strong local connections, and the interfered group is a weak local connection. This weak local connection would be part of a strong local connection that is not yet fully discovered.

To prove the conjecture, White can alter the position as shown in 6-III. The untouched sub-patterns are replaced by black pieces, to solidify their connections, and they are surrounded by white pieces. The latter is necessary, because the conjecture is that the remaining parts of the losing pattern in 6-I are independent from the part that is yet to be discovered. It must therefore be enforced that Black wins without using any of the cells adjacent to the untouched groups, which is achieved by adding the surrounding white pieces.

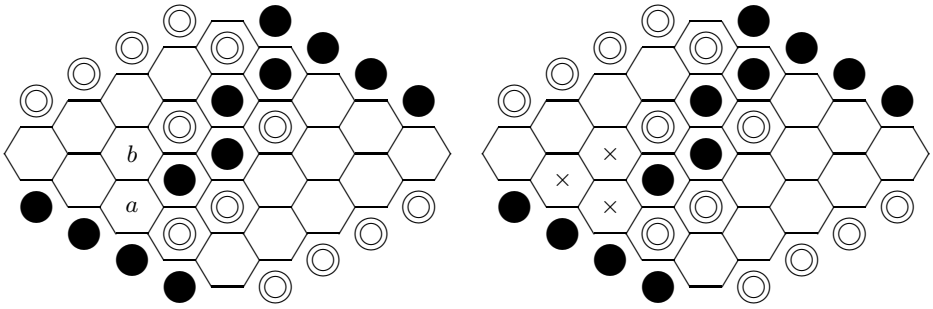
In the position thus created, White needs only to check the cells in the interfered group of 6-II. These are the three cells indicated in 6-III. *If* this yields a loss for White, *then* position 6-I is also a loss for White, and the threat pattern for 6-I is the one in 6-III plus the untouched groups of 6-II.

2.5 Heuristic pattern search

While very powerful in theory, threat patterns are very sensitive to good move ordering in practice. Figure 7 shows an example. If the move at a is tried first, the algorithm will conclude that the position is a win with a threat pattern consisting of only cell a. If on the other hand the move at b is tried first, the win is still proved, but the resulting threat pattern contains three cells. The pattern is still valid and correct, but it is bigger than it needs to be.

When backing up the patterns, they quickly grow too large if special care is not taken to keep them as small as possible. A pattern that covers the entire board is trivially valid, but it is also useless since the search then becomes a standard alpha-beta search.

Another aspect of pattern search is that it can only prove wins and losses. It does not find heuristic values when no proof can be found. Both these problems can be alleviated by using pattern search as an enhancement of standard heuristic alpha-beta search. If iterative deepening is used, the fastest win will be spotted first, and the threat pattern will tend to be as small as possible. Pseudocode for a pattern-enhanced alpha-beta algorithm is included in Appendix B.



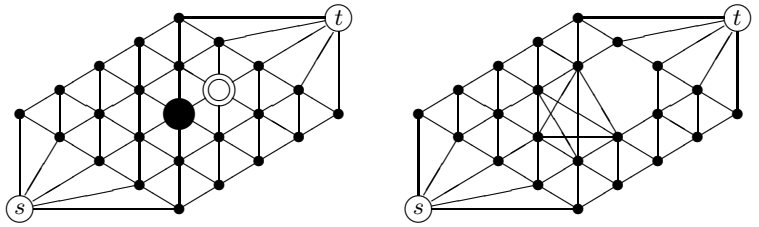
I: Black to move wins II: threat pattern after trying move b

Fig. 7. The result of an unfortunate choice of move

3 Evaluation

Methods to arrive at heuristic evaluations of Hex positions are difficult to find. Concepts such as material balance and mobility, useful in many other board games, are meaningless in Hex. Evaluation methods are commonly based on measuring properties of the game graph, such as network flow in Hexy and graph distance in Queenbee. The only known method that does not use such a model is Y-reduction, to be described in Section 4.

Graph models such as network flow and distance use a graph representation of Hex as shown in Figure 8, in which Black tries to connect s and t. Each position has two such graphs, one from Black's point of view and one from White's. See Appendix A for an explanation of these graphs.



I: graph colouring representation II: graph reduction representation

Fig. 8. Graph representations from Black's point of view

3.1 Network flow

The graph reduction representation of a Hex position can be viewed as a flow network, where fluid flows from s to t. Each link has unit capacity, except for

the links emanating from s and t which have infinite capacity. The maximum flow capacity from s to t is taken as a heuristic evaluation of the position.

Closely related to this approach is the electrical circuit model, in which it is not fluid but electricity that flows through the network. The network links in this case do not have maximum capacity, but they all have unit resistance. The links at s and t have zero resistance. The overall resistance between s and t is used as a heuristic evaluation of the position. Shannon and Moore built a physical Hex playing machine which used this approach [Sha53]. The same method is used in Hexy.

These models can also be used to obtain heuristic values for the available moves, based on the amount of current flowing through the links of the network. Hexy uses the energy dissipation of all links adjacent to a node, which in the graph reduction representation amounts to summing the squares of all the currents in those links, as the links all have unit resistance.

Network flow models are related to the concept of graph connectivity, which refers to the number of distinct paths that connect two vertices in a graph. A high degree of connectivity leads to a high flow capacity or electrical capacity, and in terms of Hex, it leads to a favourable position as there are many ways to connect.

3.2 Graph distance

The graph distance between s and t can be used as a rough heuristic estimate of the position, as it starts at $n+1$ and reaches 1 if and only if the game is won. This estimate turns out to be rather poor, as can be seen by recognizing that playing a move in the center of an empty board does not decrease the graph distance for the opponent.

A better method is the "two-distance" measure used in Queenbee. In the regular graph distance metric, the distance of a vertex v to the goal node t is one more than the minimum distance of v's neighbours to t. Measuring the distance this way amounts to computing the number of "free moves" needed to complete the connection, ignoring the opponent's endeavours to block these connections. The two-distance looks instead at the *second shortest* distance of v's neighbours to t, the intuition being that the opponent can block the shortest one.

Queenbee sums up the two-distance to the two black borders in the black reduction graph. The smallest of these numbers is taken as the overall black distance; the number of occurrences of this distance on the board is black's potential. When two positions have equal distance between the borders, then the position with the highest potential is preferable as there are more ways to realize this distance. The full board evaluation is obtained by calculating the same numbers for White in the white graph, and then considering the difference between these numbers for Black and for White. Moves are evaluated by summing the black and white numbers in each cell, recognizing that important cells are those that are close to establishing a winning connction for both players. players.

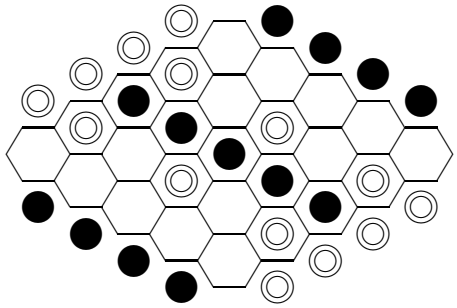


Fig. 9. A counterexample of two-distance

Unlike network flow and connectivity models, two-distance captures the adversarial nature of a two-player game. However, it suffers from one important drawback: It can sometimes label a position as lost when it is in fact a win. An example is shown in Figure 9. The black two-distance between the black borders is infinite, while the white two-distance between the white borders is not. The two-distance metric concludes that Black is lost. In reality, the position is a win for Black even when White plays first. Such pathological positions are rare, but they do occur in game tree searches and may occasionally alter the search result.

4 Y Reduction

The Game of Y, discovered by Craig Schensted, is closely related to Hex. Indeed, Hex is a special case of it. This makes the game PSPACE-complete, and any method for playing Y immediately yields a method for playing Hex. Such a method is *Y reduction*, based on observations by Craig Schensted and Steven Meyers.

4.1 The game of Y

The game of Y is played on a triangular board tesselated with hexagons. The goal is to establish a chain that connects all three sides of the board, such as in Figure 10. Figure 11 demonstrates that Hex is a special case of Y. Playing Y in this diagram is equivalent to playing Hex in the empty region. Thus, a size $n \times n$ Hex board can be turned into a Y board of size $2n - 1$ by adding two triangular regions. Each region is filled with pieces of the colour of the Hex border that it is appended to.

By arguments analagous to those used in Hex, it can be seen that Y cannot end in a draw and that it must be a theoretical win for the first player. One way of arriving at a surprising proof of the no-draw property is based on *micro reduction*, which is described in the next section.

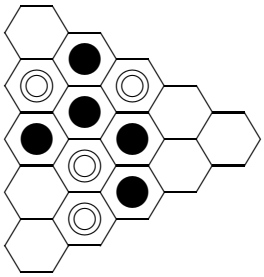


Fig. 10. A size-5 Y board with a winning chain for Black

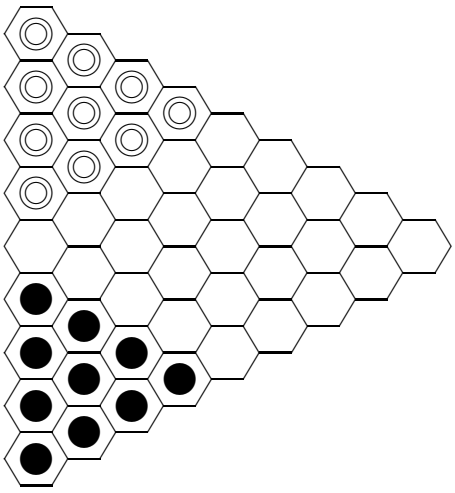


Fig. 11. Hex is a special case of Y

4.2 Micro reduction

Consider a size-n Y board completely filled with black and white pieces, as in Figure 12-I. This board is then reduced to a board of size $n-1$, where each cell on the $n-1$ board corresponds to a group of three cells on the n board. These three cells must be neighbours, and form a little triangle oriented the same way as the whole board. The $n-1$ board then gets filled with black and white pieces, where the colour of a cell is determined by the majority of the colours of the three corresponding cells on the bigger board.

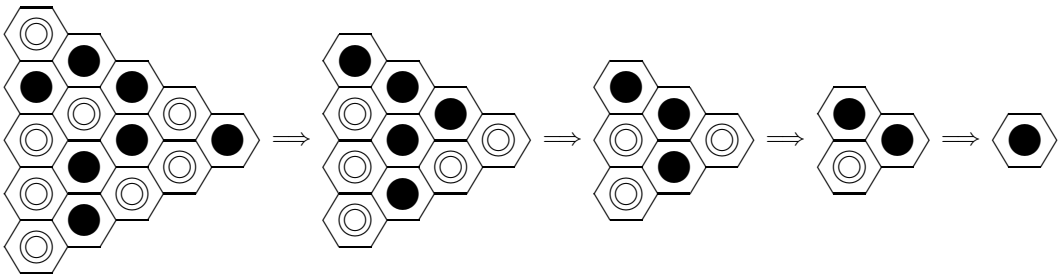


Fig. 12. Micro reduction steps

As it turns out, the colour of the lone piece on the size-1 board indicates who the winner is on each of the preceding boards. This is because every chain of one colour is preserved by the reduction step, as each link in the chain corresponds to a small triangle with at least two pieces of that colour. If a chain touches one side of the board, then so does the corresponding chain in the reduced board. A winning chain will therefore produce corresponding winning chains in all of the smaller boards, including the final trivial size-1 board on which a winning chain is simply the one piece on the board.

This micro reduction method immediately proves that Y cannot end in a draw. By extension, it gives a no-draw proof for Hex that is quite different from the ones given by Beck [BBC69] and Gale [Gal86].

4.3 Macro reduction

A Y board can also be reduced to a board of one size smaller by omitting one of the three border rows. This is called macro reduction. Each position has three macro-reduced subpositions. The observation is that a position contains a winning chain if and only if at least two of the three macro-reduced subpositions do. This can be seen by realizing that the winners of the three subpositions are readily displayed in the size-2 board of the penultimate step in the micro reduction chain.

Macro reduction does not immediately suggest a plausible method for playing Y, since it decomposes the board into 3^n subpositions, which is a number

far greater than the $O(n^3)$ pieces that micro reduction produces, even when taking into account that there are many duplicates in the tree of macro reduced subpositions.

4.4 Reduction of partially empty boards

If micro reduction can be extended to positions with empty cells, a heuristic evaluation method for Y and for Hex is generated. One way to do this is to assign a *probability of ownership* to each cell on the board. Cells that already contain pieces have probability 1 or 0 according to who owns them. Empty cells have probability $\frac{1}{2}$. When reducing a triangle of cells with probabilities p_1, p_2, and p_3, the probability q of the reduced cell is the probability of owning at least two of the p_i cells. According to probability theory, this is

$$q = p_1p_2 + p_1p_3 + p_2p_3 - 2p_1p_2p_3. \tag{1}$$

In reality, Y is not a game of chance. Moreover, the probabilities p_i are not independent, since playing a piece on the board alters a growing number of probabilities down the chain of reduced diagrams. Nevertheless this method may provide a good quick-and-dirty way to give a heuristic evaluation of a Y position.

To simplify matters, the interval [-1,+1] can be used instead of [0,1]. In that case, already played pieces have values -1 and +1 and an empty cell has value 0. The equation then becomes

$$q = \frac{1}{2}(p_1 + p_2 + p_3 - p_1p_2p_3). \tag{2}$$

This reduction method generates a pyramid of values, starting with the $\frac{1}{2}n(n+1)$ cells of the game board and going down to the single value of the size-1 board that represents the final evaluation. The number of calculations carried out in the entire reduction chain is $\frac{1}{6}n(n+1)(n+2)$. The calculations can be done incrementally, since they are all local. This saves quite a bit of work as playing a move leaves most of the values unchanged on the bigger boards in the reduction chain.

4.5 Move evaluation

The reduction heuristic can be used to rate the available moves. The most straightforward way to do this is to try each move and see how much the evaluation changes. This would amount to a $O(n^3) \times O(n^2)$ computation. A good estimate can be obtained much faster by calculating the partial derivative of the final evaluation with respect to each of the values in the reduction pyramid. These can be calculated easily; if v is the final evaluation then

$$\frac{\partial v}{\partial p_1} = \frac{\partial v}{\partial q} \cdot \frac{\partial q}{\partial p_1} = \frac{\partial v}{\partial q} \cdot \frac{1}{2}(1 - p_2p_3). \tag{3}$$

This is the contribution of $\triangle(p_1p_2p_3)$. Since p_1 is usually part of three reduced triangles, the contributions of the other triangles need to be added.

This computation builds a second pyramid of $O(n^3)$ values in $O(n^3)$ steps, using the values contained in the reduction pyramid. The move evaluation pyramid is built in the other direction, starting with the size-1 diagram.

The partial derivatives are not exact predictions for the change in v, but this need not matter as v was itself a rough estimate.

References

[Ans00] V. Anshelevich. The game of Hex: An automatic theorem proving approach to game programming. In *AAAI proceedings*, 2000.

[BBC69] A. Beck, M. D. Bleicher, and D. W. Crowe. *Excursions into Mathematics*. Worth Publishers Inc., New York, 1969.

[Bro00] C. Browne. *Hex Strategy: Making the Right Connections*. A. K. Peters, Natick, Massachusetts, 2000.

[ET76] S. Even and R. E. Tarjan. A combinatorial problem which is complete in polynomial space. *Journal of the Association for Computing Machinery*, 23:710–719, 1976.

[Gal86] D. Gale. The game of Hex and the Brouwer fixed point theorem. *American Mathematical Monthly*, pages 818–827, 1986.

[Gar59] M. Gardner. *The Scientific American Book of Mathematical Puzzles and Diversions*, chapter The game of Hex. Simon and Schuster, New York, 1959.

[Mey02] S. Meyers. Personal communication, 2002.

[Rei81] S. Reisch. Hex ist pspace-vollständig. *Acta Informatica*, 15:167–191, 1981.

[Rij00] J. van Rijswijck. Computer Hex: Are Bees better than Fruitflies? Master's thesis, University of Alberta, 2000.

[Sha53] C.E. Shannon. Computers and Automata. *Proceedings of the Institute of Radio Engineers*, 41:1234–1241, 1953.

[Ste00] I. Stewart. Hex marks the spot. *Scientific American*, pages 100–103, September 2000.

[YLP01] J. Yang, S. Liao, and M. Pawlak. A decomposition method for finding solution in game Hex 7x7. In *International Conference on Application and Development of Computer Games in the 21st Century*, pages 96–111, November 2001.

[YLP02a] J. Yang, S. Liao, and M. Pawlak. Another solution for Hex 7x7. Technical report, University of Manitoba, 2002. Online: `http://www.ee.umanitoba.ca/ jingyang/TR.pdf`.

[YLP02b] J. Yang, S. Liao, and M. Pawlak. New Winning and Losing Positions for 7x7 Hex. *Computers and Games*, 2002.

A Hex Graphs

Hex is a special case of the *Shannon Switching Game*, a graph vertex colouring game. The Shannon Switching Game can be played on any graph by choosing two particular vertices s and t, and giving player *Black* the task to connect these two vertices while player *White* tries to prevent this. Figure 13 shows a game graph that is equivalent to a 5×5 Hex board.

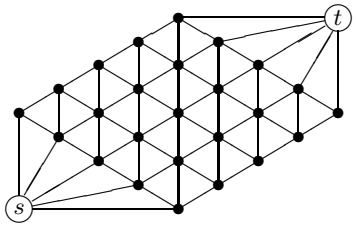
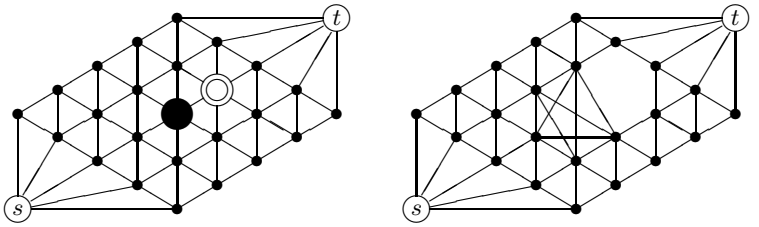


Fig. 13. A Shannon Switching Game graph equivalent to 5×5 Hex

When White colours a vertex v, the resulting position is equivalent to the one in which v was simply deleted from the graph. Similarly, a move by Black is equivalent to *contracting* the vertex, which means removing it from the graph and introducing new edges between each pair of v's neighbours. Thus, the graph is reduced by one vertex every move, and the game ends when s and t become direct neighbours or become disconnected. Figure 14 shows a position in the graph colouring game on the left, and the equivalent position in the graph reduction game on the right.



I: graph colouring representation II: graph reduction representation

Fig. 14. Two representations of the same Hex position

The graphs in Figure 14 represent the game from Black's point of view. The same position can also be represented from White's point of view as in Figure 15.

B Algorithms

Below follows the pseudo code for the pattern search algorithm and the Y reduction algorithm. The basic pattern search in Table 1 only returns wins and losses, no heuristic values. The auxiliary function `won`(P) detects a winning chain *for the side to move* in position P. The values returned consist of a win/loss value along with a threat pattern that proves this result. As per the definition of a threat pattern, anything outside of the pattern does not affect the win or loss.

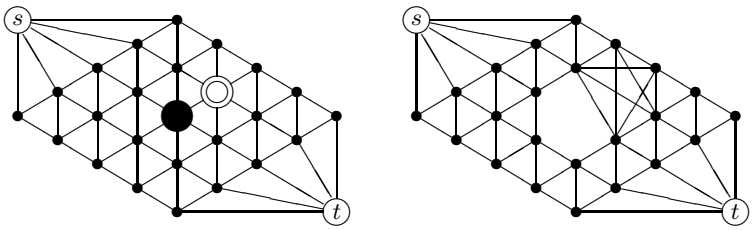


Fig. 15. The same position as in Figure 14, but from White's point of view

Table 2 shows how to enhance basic alpha-beta search with pattern search. The search behaves like alpha-beta, with the addition of threat patterns that are returned whenever the value is a proven win or loss. If the returned value is a heuristic value, the returned threat pattern is ignored. The auxiliary function `evaluate`(P) provides a heuristic evaluation of position P, which must be strictly more than `loss` and less than `win`.

Table 3 contains the Y reduction algorithm for calculating the heuristic board evaluation as well as the individual move evaluations. The algorithm first seeds `eval`[...][...][`boardsize` $-$ `1`] with the contents of the board, and then calculates the successive Y reductions using

$$e_{i,j}^{(k)} = f(e_{i,j}^{(k+1)}, e_{i+1,j}^{(k+1)}, e_{i,j+1}^{(k+1)})$$

where $f(p,q,r) = \frac{1}{2}(p+q+r-2pqr)$, and $e^{(k)}$ contains the values in the triangle of size $k+1$. Eventually `eval`[`0`][`0`][`0`] is reached, containing the final heuristic value. Next, the move values are calculated, starting with `move`[`0`][`0`][`0`] $= 1$ and using

$$\begin{aligned}
m_{i,j}^{(k)} &= \frac{\partial e_{0,0}^{(0)}}{\partial e_{i,j}^{(k)}} \\
&= \sum_{p,q} \left(\frac{\partial e_{0,0}^{(0)}}{\partial e_{p,q}^{(k-1)}} \cdot \frac{\partial e_{p,q}^{(k-1)}}{\partial e_{i,j}^{(k)}}\right) \\
&= \frac{\partial e_{0,0}^{(0)}}{\partial e_{i,j}^{(k-1)}} \cdot \frac{\partial e_{i,j}^{(k-1)}}{\partial e_{i,j}^{(k)}} + \frac{\partial e_{0,0}^{(0)}}{\partial e_{i-1,j}^{(k-1)}} \cdot \frac{\partial e_{i-1,j}^{(k-1)}}{\partial e_{i,j}^{(k)}} + \frac{\partial e_{0,0}^{(0)}}{\partial e_{i,j-1}^{(k-1)}} \cdot \frac{\partial e_{i,j-1}^{(k-1)}}{\partial e_{i,j}^{(k)}} \\
&= m_{i,j}^{(k-1)} \cdot g(e_{i+1,j}^{(k)}, e_{i,j+1}^{(k)}) + \\
&\quad m_{i-1,j}^{(k-1)} \cdot g(e_{i,j}^{(k)}, e_{i-1,j+1}^{(k)}) + \\
&\quad m_{i,j-1}^{(k-1)} \cdot g(e_{i+1,j-1}^{(k)}, e_{i,j}^{(k)})
\end{aligned}$$

where $g(q,r) = \frac{\partial}{\partial p} f(p,q,r) = \frac{1}{2}(1-qr)$.

The search algorithm can be enhanced with any desirable alpha-beta enhancement, just as long as care is taken that `win` and `loss` values are only

returned when they are proven, and that the patterns are used only when the value is proven. The Y reduction algorithm can be improved by doing the calculations incrementally as moves are made and unmade on the board. The algorithm as shown in Table 3 computes values for the game of Y; it is easily modified for $n \times n$ Hex by converting the position into a Y position with `boardsize` $= 2n-1$ and seeding two triangular regions appropriately as described in Section 4.1 and Figure 11.

```
type pattern_t {
     int value ∈ {win,loss};
     boolean pattern[boardsize][boardsize];
}
```

```
pattern_t Patternsearch(position pos) {
     pattern_t Ψ;

     if (Won(pos)) {
        Ψ.pattern := ∅;
        Ψ.value := win;
        return Ψ;
     }

     movelist := Ω;                                  /* initiate move list with all empty cells */
     Ψ.pattern := ∅;                                 /* initiate pattern with empty set */

     foreach (move ∈ movelist) {
        ψ = Patternsearch(pos ⊕ move);
        if (ψ.value ≡ loss) {                        /* move was a winning move; return immediately */
           Ψ.value := win;
           Ψ.pattern := move ∪ ψ.pattern;
           return Ψ;
        }
        else {                                       /* move was a losing move */
           Ψ.pattern := Ψ.pattern ∪ ψ.pattern;
           movelist := movelist ∩ ψ.pattern;         /* discard all moves not in ψ.pattern */
        }
     }

     Ψ.value := loss;                                /* no winning moves found; position is a loss */
     return Ψ;
}
```

Table 1. The pattern search algorithm

```
type pattern_t {
     float value;
     boolean pattern[boardsize][boardsize];
}
```

```
pattern_t Pattern-ab-search(position pos, float α, float β, int depth) {
     pattern_t Ψ;
     float best;

     if (Won(pos)) {
        Ψ.pattern := ∅;
        Ψ.value := win;
        return Ψ;
     }

     if (depth ≡ 0) {                        /* return heuristic evaluation, pattern is irrelevant */
        Ψ.value := evaluate(pos);
        return Ψ;
     }

     movelist := Ω;                                  /* initiate move list with all empty cells */
     Ψ.pattern := ∅;                                      /* initiate pattern with empty set */
     Ψ.value := loss;

     foreach (move ∈ movelist) {
        ψ = Pattern-ab-search(pos ⊕ move, -β, -Max(α,best), depth-1);
        if (ψ.value ≡ loss) {                 /* move was a winning move; return immediately */
           Ψ.value := win;
           Ψ.pattern := move ∪ ψ.pattern;
           return Ψ;
        }
        else if (ψ.value ≡ win) {                                  /* move was a losing move */
           Ψ.pattern := Ψ.pattern ∪ ψ.pattern;
           movelist := movelist ∩ ψ.pattern;              /* discard all moves not in ψ.pattern */
        }
        Ψ.value := Max(Ψ.value, -ψ.value);
        if (Ψ.value ≥ β) {                                                  /* alpha-beta cutoff */
           return Ψ;
        }
     }

     return Ψ;
}
```

Table 2. Pattern-enhanced alpha-beta search

```
type pyramid_t {
    float eval[boardsize][boardsize][boardsize];
    float move[boardsize][boardsize][boardsize];
}
```

```
float f(float p, float q, float r) {
    return ½(p + q + r − pqr);
}


float g(float q, float r) {
    return ½(1 − qr);
}


pyramid_t Y-eval(position pos) {
    pyramid_t Δ;

    /* seed the eval[...][...][boardsize-1] values */
    forall(i ≥ 0, j ≥ 0, i + j < boardsize) {
        Δ.eval[i][j][boardsize-1] := pos[i][j]; /* +1 = me, -1 = opponent, 0 = empty */
    }

    /* calculate all other eval values */
    for (k := boardsize-2; k ≥ 0; k--) {
        forall (i ≥ 0, j ≥ 0, i + j + k < boardsize) {
            Δ.eval[i][j][k] := f( Δ.eval[i][j][k + 1],
                                  Δ.eval[i + 1][j][k + 1],
                                  Δ.eval[i][j + 1][k + 1]);
        }
    }

    /* calculate the move values */
    move[0][0][0] := 1;
    for (k := 1; k < boardsize; k++) {
        forall (i ≥ 0, j ≥ 0, i + j + k < boardsize) {
            Δ.move[i][j][k] := Δ.move[i][j][k − 1]
                                  * g(Δ.eval[i + 1][j][k],Δ.eval[i][j + 1][k])
                               +  Δ.move[i − 1][j][k − 1]
                                  * g(Δ.eval[i][j][k],Δ.eval[i − 1][j + 1][k])
                               +  Δ.move[i][j − 1][k − 1]
                                  * g(Δ.eval[i + 1][j − 1][k],Δ.eval[i][j][k])
        }
    }

    return Δ;
}
```

Table 3. Calculating Y values