Automatic Strategy Verification for Hex[⋆]

Ryan B. Hayward, Broderick Arneson, and Philip Henderson

Department of Computing Science,
University of Alberta, Edmonton, Canada
{hayward,broderic,ph}@cs.ualberta.ca

Abstract. We present a concise and/or-tree notation for describing Hex strategies together with an easily implemented algorithm for verifying strategy correctness. To illustrate our algorithm, we use it to verify Jing Yang's 7×7 centre-opening strategy.

1 Introduction

Hex is the classic two-player board game invented by Piet Hein in 1942 and independently by John Nash around 1948 [1,2,6,7,8,9]. The game is named after the board, which consists of a parallelogram-shaped $m \times n$ array of hexagons, also called cells. Each player is assigned a set of stones and two opposing board sides; players alternately place a stone on an unoccupied cell; the first player to form a path connecting her[1] two sides with her stones wins the game. For example, Fig. 1 shows the start and end of a game on a 3×3 board. White succeeds in joining her two sides, so White wins this game. For more on Hex, see the recent survey by Hayward and Van Rijswijck [3] or the web page by Thomas Maarup [7].

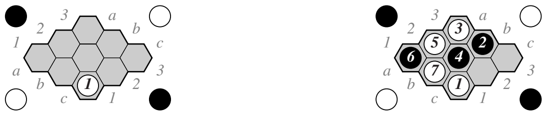

Fig. 1. The start (left) and finish (right) of a Hex game on a 3×3 board

An intriguing aspect of the game of Hex is that for all $n \times n$ boards, although a winning first-player strategy is known to exist [1,2,9], explicit such strategies have been found only for small boards. While finding such strategies is routine on very small boards, the task quickly becomes challenging as board size increases. This is not surprising since, as Stefan Reisch has shown, determining the winner of arbitrary Hex positions is PSPACE-complete [11].

[⋆] Authors gratefully acknowledge the support of NSERC and the University of Alberta Games Group.

[1] For brevity we use 'she' and 'her' whenever 'she or he' and 'his or her' are meant.

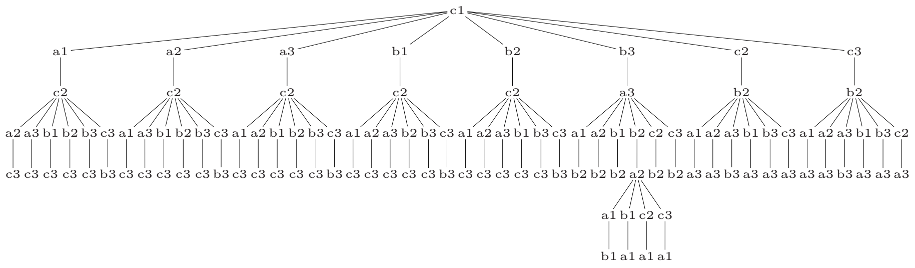

Fig. 2. A winning first-player 3×3 Hex strategy. Fig. 1 shows one line of this strategy.

For 7×7, 8×8, and 9×9 boards, Jing Yang found strategies by hand [12,13,15,16]. Later, Hayward *et al.* found other 7×7 strategies by computer [4,5], while Noshita found 7×7 strategies and one 8×8 strategy similar to Yang's by hand [10]. For boards 10×10 or larger, no winning strategies are known.

As the search for winning strategies on larger boards continues, it is of interest to provide algorithms for verifying strategy correctness. Recently, Noshita described strategies in a manner that arguably facilitates human verification [10]. By contrast, in this paper we present a system that allows for computer verification. To demonstrate the utility of our system, we use it to confirm the correctness of Yang's original 7×7 strategy [13].

2 Excised Trees and Autotrees

The underlying feature of our verification system is the condensed tree notation we use to represent strategies.[2] Our notation allows the standard tree description of a strategy to be condensed in three ways. First, it permits the use of an "and" operation corresponding to the combinatorial sum of independent substrategies. Second, it permits the use of a macro descriptor for representing repeatedly occurring substrategies. Third, it allows all opponent moves to be excised from the tree by replacing each set of opponent responses with a single anonymous meta-response.

The first two of these three ideas are well known; for example, they were used by Yang in his description of his proofs [12,13,15,16]. The third idea, namely using excised trees, is new. In the rest of this section we illustrate the excision process and show that it does not hamper verification.

To begin, consider the first-player strategy tree in Fig. 2. The nodes at even depth indicate first-player moves; the nodes at odd depth indicate second-player moves; the game in Fig. 1 follows one root-to-leaf path through the tree. Notice that the first-player strategy described by the tree is *complete*: after each second-player move, there is a unique first-player response; after each first-player move,

[2] This notation could also be used for other two-player board games in which game pieces are fixed once they have been placed.

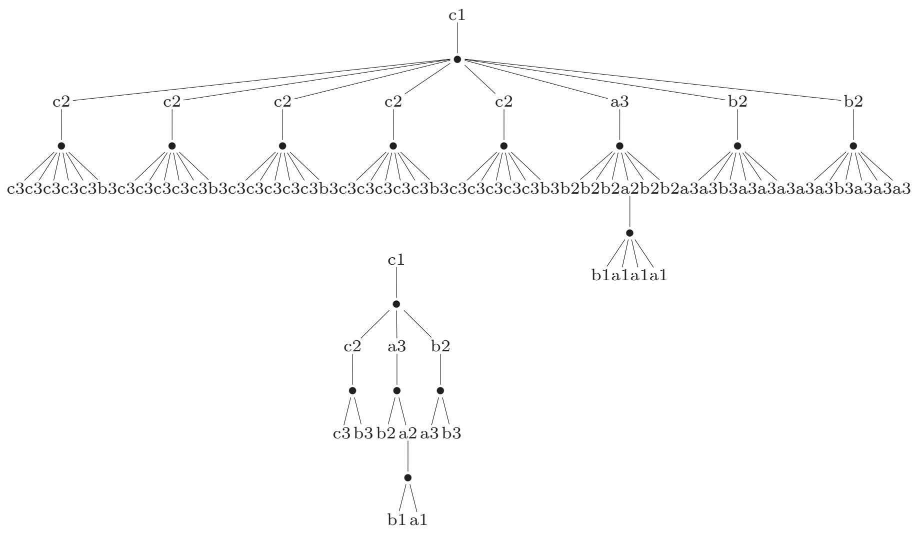
c1
c2 c2 c2 c2 c2 a3 b2 b2
c3c3c3c3c3b3c3c3c3c3c3b3c3c3c3c3c3b3c3c3c3c3c3b3c3c3c3c3c3b3b2b2b2a2b2b2a3a3b3a3a3a3a3a3b3a3a3a3
b1a1a1a1
c1
c2 a3 b2
c3 b3 b2 a2 a3 b3
b1 a1

Fig. 3. The tree obtained from the strategy tree in Fig. 2 by replacing each set of opponent response nodes with a single "•" meta-node (top), and the excised tree obtained by then repeatedly merging identical subtrees (bottom)

there is every possible second-player response. Also, each leaf node establishes a first-player win, so this is a winning strategy for the first player.

Next, consider the two trees shown in Fig. 3. The top tree is obtained from the tree in Fig. 2 by excising nodes corresponding to second-player moves; each set of second-player moves is replaced with a single meta-node, indicated in our diagrams by a dot (•). The bottom tree is obtained from the top tree by repeatedly merging identical subtrees into a single subtree until, for each node, all subtrees are distinct. We refer to the bottom tree as an *excised tree.*

More generally, given any complete (but not necessarily winning) strategy tree, the following process, which we call *excision*, replaces the tree with an excised tree.

> *For each non-leaf first-player node, merge the children into a single meta-node. Next, as long as some second-player node has two identical subtrees, remove one of these subtrees.*

Excised trees represent equivalence classes of strategies, so some information is lost when a strategy tree is replaced with its excised tree. However, excision can be reversed in the following sense: for any excised tree E for a player, there is a set $\mathcal{S}$ of strategy trees such that E is the excised tree of every tree in $\mathcal{S}$. Furthermore, it is easy to construct all elements of $\mathcal{S}$ from E via the following process, which we call *restoration*:

> *At each meta-node m, for each possible opponent move to a cell c, select for the player's responding move any cell r that is the root of a subtree of m in which c does not appear.*

For example, consider the restoration process for the excised tree shown at the bottom of Fig. 3. Start with the top-most meta-node m^*, namely the child of $c1$. For this board position, the cell set of possible opponent moves is $\{a1$, $a2$, $a3$, $b1$, $b2$, $b3$, $c2$, $c3\}$. Consider the first such cell, $a1$. The cell sets of the subtrees of m^* are $\{c2, b3, c3\}$, $\{a3, b2, a2, b1, a1\}$, and $\{b2, a3, b3\}$. Since $a1$ is not in the first or third of these three cell sets, we can select the root of either the first or third subtree of m^*. Let us assume in this example that we always select the root of the first available subtree. Thus, as the response to $a1$ we select the root of the first subtree, namely $c2$. Continuing in this fashion, we select $c2$ as the response for opponent moves to $a2$, $a3$, $b1$, and $b2$, and we select $a3$ as the response for opponent moves to $b3$, $c2$, and $c3$. Having selected all responses to m^*, we continue in top-down order to process meta-nodes until all such nodes have been dealt with and the excised tree has been replaced with a complete strategy tree S' of $\mathcal{S}$.

Notice that S' is different from the strategy tree S of Fig. 2 from which E was derived; for example, in the restoration process we never selected the root of the third subtree of m^* as a response to an opponent move. However, by repeating the restoration process once for each of the possible permutations of choices for r, we would construct all possible strategy trees associated with E, including S.

In the restoration process it will always be possible to find at least one value of r at each meta-node as long as the excised tree being restored was obtained from a complete strategy tree. This follows from Observation 1, which in turn follows from the fact that in Hex, stones never move once played.

With respect to a strategy, a π-move is a move made by player π. With respect to a strategy tree, a π-node is a node associated with a π-move, and a $\overline{\pi}$-node is a node associated with π's opponent.

Observation 1. *Let T be a complete Hex strategy tree for a player π, let p be a π-node of T that is not a leaf, let S_1,...,S_k be the subtrees of T rooted at the children of p, and for each S_j let P_j be the set of cells associated with the π-nodes of S_j. Then the combined intersection $I = P_1 \cap \ldots \cap P_k$ is empty.*

Proof. For each index j, let q_j be the cell associated with the root of S_j. T is complete, so $Q = \{q_1, \ldots, q_t\}$ corresponds to all possible opponent responses to p, namely all the unoccupied cells after the move p. Also, for each index j, q_j is occupied by an opponent's stone and so is not in P_j, and so is not in I. Thus I is empty.

The following is a corollary of the preceding observation.

Observation 2. *Let $E(T)$ be the excised tree obtained from a complete Hex strategy tree T for a player π, let m be a meta-node of $E(T)$ that is not a leaf, let S_1,...,S_k be the subtrees of $E(T)$ rooted at the children of m, and for each S_j let P_j be the set of cells associated with the π-nodes of S_j. Then the combined intersection $I = P_1 \cap \ldots \cap P_k$ is empty.*

We refer to the class of trees that we use in our verification system as "autotrees"; we use this term since such trees make explicit mention only of a player's *own* moves. Autotrees have the same form and function as excised trees; however, they may not have arisen via excision, and so we do not define them with respect to excision. An *autotree* is defined as follows: each node at one set of alternating levels is a special node called a *meta-node*; each node at the other set of alternating levels is labeled with a board cell.

We call an autotree *elusive* if it satisfies the conditions of Observation 2. Notice that restoration generates a complete strategy tree from an autotree if and only if the autotree is elusive.

As an initial step in our verification algorithm, we check whether the input autotree is elusive. The second and final step in our verification algorithm is to determine whether the strategies associated with the input autotree are winning. We call an autotree of a player *satisfying* if, for every leaf, the cells of the root-to-leaf path satisfy the conditions of a win, namely join the player's two sides on the Hex board. An elusive autotree represents a winning strategy if and only if the autotree is satisfying. This follows from the following theorems, which in turn follow by straightforward arguments from our definitions and the discussion to this point; we omit the details of the proofs.

Theorem 3. *For Hex, for any complete strategy tree there is a unique associated elusive excised tree, and for any elusive autotree there is a unique set of associated complete strategy trees. Furthermore, for any complete strategy tree S and the excised tree $E(S)$ derived from S, S is winning if and only if all strategy trees S' created via restoration from $E(S)$ are winning.*

Theorem 4. *An autotree represents a winning strategy if and only if the autotree is elusive and satisfying.*

3 And/or Autotrees with Leaf Patterns

To complete the description of our notation, we need only to describe how we add two features to autotrees: *and*-nodes and leaf patterns.

Notice that the children of a meta-node in an autotree correspond to an "or" decision in a strategy; depending on the opponent's move at the meta-node, the player will play the strategy corresponding to the first subtree, *or* the next subtree, *or* the next subtree, and so on; see the excised tree in Fig. 3. By contrast, in Hex as in many other games, a particular strategy often decomposes into two or more independent substrategies that each need to be followed.

Such "and" operations are easily incorporated into our notation by allowing each labeled node (namely, not a meta-node) of a modified autotree to have any number of children. We refer to autotrees that are modified in this way as *and/or autotrees* since, when interpreting them as strategies, the odd depth nodes (the meta-nodes) are *or*-nodes while the even depth nodes (with cell labels) are *and*-nodes.

Consider for example Fig. 4, which shows an *and/or* autotree for a winning 4×4 strategy. The root is an *and*-node, so we have to play all substrategies simultaneously; in this case, there is only one subtree so there is only one substrategy

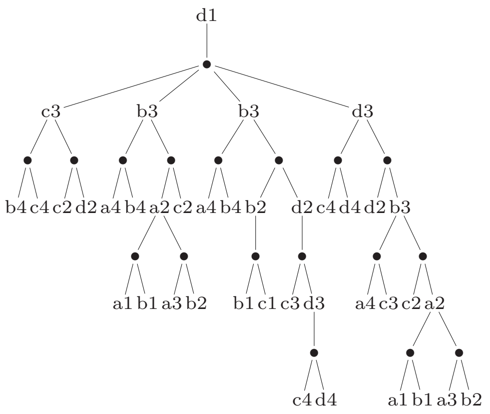
d1
c3 b3 b3 d3
b4 c4 c2 d2 a4 b4 a2 c2 a4 b4 b2 d2 c4 d4 d2 b3
a1 b1 a3 b2 b1 c1 c3 d3 a4 c3 c2 a2
c4 d4 a1 b1 a3 b2

Fig. 4. An *and/or* autotree for a winning first-player 4×4 Hex strategy. Odd depth nodes (•) are "or"-nodes; even depth nodes (cell labels) are "and"-nodes. Fig. 5 shows one line of this strategy.

to follow. Suppose that the opponent's response to the player's initial move $d1$ is $b3$. Then the player can select any subtree not containing $b3$, say the first subtree; thus the player moves to $c3$, the root of the first subtree. This root is an *and*-node with two subtrees, so now the player has to follow these two substrategies simultaneously; the player must ensure that she reaches a leaf node in each of the subtrees of every *and*-node. For example, if the opponent's next move is at one of $\{b4, c4\}$, the player must immediately reply with the other of these two cells or risk not reaching a leaf of the $\{b4, c4\}$ subtree. Similarly, if the opponent's next move is at one of $\{c2, d2\}$, the player must immediately reply with the other of these two cells. If the opponent's next move is not in $\{b4, c4\}$ or $\{c2, d2\}$, the player can move anywhere. Fig. 5 illustrates another line of play of this strategy.

Finally, subtrees of *and/or* autotrees that correspond to isomorphic substrategies can be replaced with a special node corresponding to such substrategies. This is illustrated in Fig. 6, where two substrategy macros are used to simplify the tree of Fig. 4.

Modifying our verification algorithms to handle *and*- and *or*-nodes is straightforward. For *or*-nodes, the test for the elusive property is the same as with unmodified autotrees: check whether the combined intersection of all child nodes is

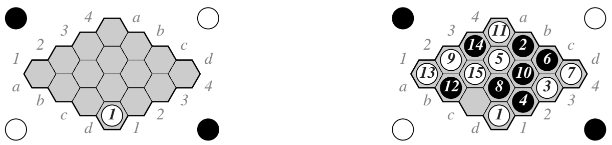

Fig. 5. The start (left) and finish (right) of one line of the strategy of Fig. 4

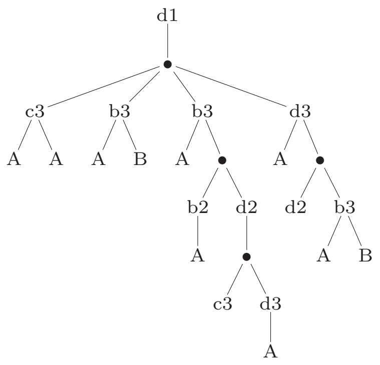

Fig. 6. An *and/or* autotree with two macro pattern nodes. This tree is equivalent to the tree in Fig. 4; pattern parameters have been omitted.

the empty set. For *and*-nodes, it is necessary to check whether the intersection of each pair of child nodes is empty. Another algorithmic approach one might take here is to expand the *and/or* autotree into the corresponding equivalent autotree; however, the resulting trees can be large,[3] so this approach would require significantly more space than our approach.

Testing the satisfying property on *and/or* autotrees involves checking every root-to-leaf path in the associated expanded autotree. For reasons of efficiency we do not want to generate the expanded autotree; we thus carry out this task in an implicit fashion. By using a simple indexing scheme for each root-to-leaf path in the *and/or* autotree, we can reconstruct the cell sets for each possible root-to-leaf path in the associated autotree. Each node stores the number of root-to-leaf paths it contains. We consider all such paths and verify that each satisfies the winning condition.

We implement the isomorphic substrategy feature in the simplest possible way, namely using macro substitution to generate the equivalent *and/or* autotree.

4 Verifying Yang's Proof

As a benchmark for testing our system, we used it to verify the first known winning 7×7 Hex strategy, namely Yang's original 7×7 center-opening strategy [14,13]. Yang described his strategy in an easily understood notation similar to that used in the C programming language; an applet that follows this strategy is available on his homepage[12]. The version of the strategy that we tested is from a preprint also available from his web page [14]. In Yang's notation, his strategy uses about 40 patterns (not counting pattern variations) comprising about six pages of text. A recursion tree indicating the hierarchy of his patterns is shown in Fig. 7.

[3] For example, an *and*-node with k subtrees of two nodes each corresponds in the expanded autotree to a node with 2^k subtrees.

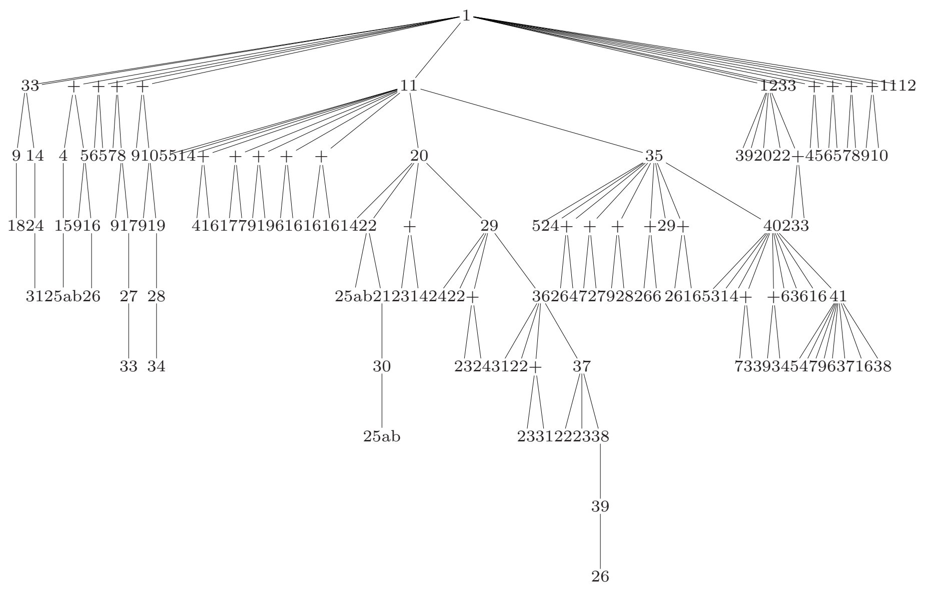

Fig. 7. Part of the recursion tree for Yang's proof. References to frequently occurring small patterns have been omitted. Labels indicate pattern numbers. Nodes labeled + are *and*-nodes; all other nodes are *or*-nodes.

```
( pattern8
  // called by: 1
  ((c6 BR) (d4 BR))
  (d6 e3 e4 e5 e6 f2 f3 f4 f5 f6 g1 g2 g3 g4 g5 g6)
  (c6 d4 BR)

  [(f3 [(pattern2ab (e3 e4) (d4 f3))]
       [(pattern2ab (g2 g3) (f3 BR))])
   (e5 [(d6) (e4)]
       [(pattern13 (e6 f4 f5 f6 g3 g4 g5 g6) (e5 BR))])
   (f2 [(pattern2ab (g1 g2) (f2 BR))]
       [(pattern9 (g5 g4 f5 f4 f3 e5 e4 e3) (BR f2 d4))])
   (e3 [(pattern17 (d6 e5 e6 f2 f3 f4 f5 g1 g2 g3 g4 g5) (c6 d4 e3 BR))]) ])
```

Fig. 8. Yang's Pattern 8 in our notation

We translated Yang's proof into our notation by hand, following his pattern naming convention. As an example of our notation, see Fig. 8. The first line gives the name of the pattern. The second line is a comment noting that the only pattern calling this pattern is Pattern 1. The third line gives the connections that are achieved by the pattern; in this case at least one of two connections is achieved, either between *c*6 and the bottom right side of the board, or between *d*4 and the bottom right side; this information is given only to aid in human debugging purposes and is not used by our algorithm. The fourth line lists the cells that

```
pattern1 connect: (TL BR)
  empty: (a1 a2 a3 a4 a5 a6 a7 b1 b2 b3 b4 b5 b6 b7 c1 c2 c3 c4 c5 c6 c7
          d1 d2 d3 d5 d6 d7 e1 e2 e3 e4 e5 e6 e7 f1 f2 f3 f4 f5 f6 f7 g1
          g2 g3 g4 g5 g6 g7)
 played: (TL d4 BR)
  stats: AND = 1480, OR = 2339, Leafs = 3514
  paths: 25574/25574
VALID pattern.
```

Fig. 9. Diagnostics returned after verifying Yang's proof

must be unoccupied at this point; the fifth line lists the cells that the player must already occupy. The subsequent lines describe the *and*/*or* autotree, where parentheses surround the subtrees of an *or*-node and square brackets surround the subtrees of an *and*-node.

In the process of verifying the description of Yang's proof, we found only one typographical error: in the description of Pattern 11 there is a call to Pattern 17 that should instead be a call to Pattern 19.

Our notation represents Yang's strategy in about 700 lines of text. The diagnostic message returned by our program after recursively verifying Yang's proof is shown in Fig. 9. The resulting tree had 1,480 *and*-nodes, 2,339 *or*-nodes, 3,514 leaves, and 25,574 implicit root-to-leaf paths. The verification took less than one second to execute on our computer, a single-processor Athlon64 3200+ with 1 gigabyte of memory.

5 Conclusions

We have introduced the notion of an *excised tree* as a compressed representation of a complete strategy tree from which all explicit opponent moves have been excised. We used excised trees in a simple algorithm that verified the correctness of Yang's original winning 7×7 Hex strategy.

One way in which our system could be improved would be to automate the process of translating strategies from other notations into our notation.

Another improvement concerns the number of paths that our algorithm checks in verifying the correctness of a strategy. Currently our system explicitly verifies that every possible cell set that a player might end up with contains a winning path. For example, for Yang's strategy this was a total of 25,574 cell sets that were checked. The problem with this approach is that the number of such cell sets, corresponding to the number of root-to-leaf paths in the complete strategy tree, increases exponentially in the board size.

Consider for example Martin Gardner's winning second-player strategy for the player with the longer sides on an $n{\times}n-1$ board [2]. The strategy consists of the *and* of $f(n) = n \times (n-1)/2$ substrategies each consisting of the *or* of two moves. The associated excised tree thus has $2^{f(n)}$ root-to-leaf paths. Even for n as small as 14, $2^{f(n)} = 2^{91}$, and checking this many paths individually is currently computationally infeasible.

Thus, as board size increases, verification algorithms will be required that do not explicitly check the winning condition for each root-to-leaf path.

References

1. Gardner, M.: Mathematical Games. Scientific American 197, July, pp. 145–150, August, pp. 120–127, October, pp. 130–138 (1957)
2. Gardner, M.: The Scientific American Book of Mathematical Puzzles and Diversions, pp. 73–83. Simon and Schuster, New York (1959)
3. Hayward, R.B., van Rijswijck, J.: Hex and Combinatorics (formerly Notes on Hex). Discrete Mathematics 306, 2515–2528 (2006)
4. Hayward, R.B., Björnsson, Y., Johanson, M., Kan, M., Po, N., van Rijswijck, J.: Solving 7×7 Hex: Virtual Connections and Game-state Reduction. In: van den Herik, H.J., Iida, H., Heinz, E.A. (eds.) Advances in Computer Games (ACG10), Many Games, Many Challenges, pp. 261–278. Kluwer Academic Publishers, Boston (2004)
5. Hayward, R.B., Björnsson, Y., Johanson, M., Kan, M., Po, N., van Rijswijck, J.: Solving 7×7 Hex with Domination, Fill-in, and Virtual Connections. Theoretical Computer Science 349, 123–139 (2005)
6. Maarup, Th.: Hex – Everything You Always Wanted to Know About Hex but Were Afraid to Ask. Master's thesis, Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark (2005)
7. Maarup, Th.: Hex Webpage (2005), `http://maarup.net/thomas/hex/`
8. Nasar, S.: A Beautiful Mind. Touchstone, New York (1998)
9. Nash, J.: Some Games and Machines for Playing Them. Technical Report D-1164, Rand Corp. (1952)
10. Noshita, K.: Union-Connections and Straightforward Winning Strategies in Hex. ICGA Journal 28(1), 3–12 (2005)
11. Reisch, S.: Hex ist PSPACE-vollständig. Acta Informatica 15, 167–191 (1981)
12. Yang, J.: Jing Yang's Web Site (2003), `http://www.ee.umanitoba.ca/~jingyang`
13. Yang, J., Liao, S., Pawlak, M.: A Decomposition Method for Finding Solution in Game Hex 7x7. In: International Conference on Application and Development of Computer Games in the 21st Century, pp. 96–111 (November 2001)
14. Yang, J., Liao, S., Pawlak, M.: On a Decomposition Method for Finding Winning Strategy in Hex Game (2001), `http://www.ee.umanitoba.ca/~jingyang/hexsol.pdf`
15. Yang, J., Liao, S., Pawlak, M.: Another Solution for Hex 7x7. Technical report, University of Manitoba, Winnipeg, Canada (2002), `http://www.ee.umanitoba.ca/~jingyang/TR.pdf`
16. Yang, J., Liao, S., Pawlak, M.: New Winning and Losing Positions for 7x7 Hex. In: Schaeffer, J., Müller, M., Björnsson, Y. (eds.) CG 2002. LNCS, vol. 2883, pp. 230–248. Springer, Heidelberg (2003)