appeared as *Theoretical Computer Science* **349** (2005) 123-139.

Solving 7×7 Hex with domination, fill-in, and virtual connections

Ryan Hayward [a] Yngvi Björnsson [b] Michael Johanson [a]
Morgan Kan [a] Nathan Po [a] Jack van Rijswijck [a]

[a] *Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, {hayward,johanson,mkan,nathan,javhar}@cs.ualberta.ca*

[b] *School of Computer Science, Reykjavik University, Iceland, yngvi@ru.is*

Abstract

We present an algorithm that determines the outcome of an arbitrary Hex game-state by finding a winning virtual connection for the winning player. Our algorithm recursively searches the game-tree, combining fixed and dynamic game-state virtual connection composition rules to find a winning virtual connection for one of the two players. The search is enhanced by pruning the game-tree according to two new Hex game-state reduction results: under certain conditions, (i) some moves dominate others, and (ii) some board-cells can be "filled-in" without changing the game's outcome.

The algorithm is powerful enough to solve arbitrary 7×7 game-states. In particular, we use it to determine the outcome of a 7×7 Hex game after each of the 49 possible opening moves, in each case finding an explicit proof-tree for the winning player.

Key words: Hex, virtual connection, pattern set, move ordering, move domination, game-state reduction, fill-in

1 Introduction

Hex is the classic two-player board game invented by Piet Hein in 1942 and independently by John Nash around 1948 [11,12,20]. The board consists of a rhombus-shaped $n \times n$ array of hexagons, also called cells. Each player is assigned a set of stones and two opposing board

⋆ This is a revised and expanded version of [17]. The support of NSERC is gratefully acknowledged.

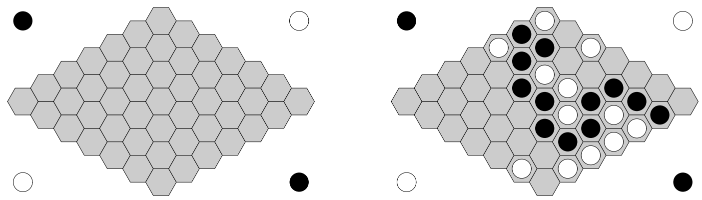


Fig. 1. An empty 7×7 board ...and a finished game; Black wins.

sides, all with the same colour; say Black gets black stones and sides, while White gets white stones and sides. Players alternately place a stone on an unoccupied cell. The first player to form a path connecting his/her two sides with his/her stones wins the game. See Figure 1. For more on Hex, see the text by Browne [8] or the survey by Hayward and Van Rijswijck [14].

In Hex, an unrestricted opening allows the first player to gain a considerable advantage. In particular, it is known that there exists a winning strategy for the first player [12]. While no explicit strategy that holds for arbitrary sized boards is known, many players believe that opening in the centermost cell in particular is a winning move. In order to offset this opening move advantage, the game is often started according to the following "swap rule": colours are assigned to the four sides of the board, but not to the players; one player then places a stone on any cell; the other player then chooses which colour stones to play with. The second move is played by the player whose stones are the opposite colour of the first stone. From then on, the game continues in normal fashion, namely with players alternating moves.

The swap rule has a balancing effect on the first move: if the first player makes an obviously winning move, the second player can swap and easily win the game; if the first player makes an obviously losing move, the second player can continue without swapping, and again win the game. The first player is thus led to search for an opening move that is neither too strong nor too weak, namely a move whose outcome is difficult to discern. While the swap rule transforms Hex from a game with a first-player winning strategy to a game with a second-player winning strategy, the second player can exploit this property only by knowing the theoretical outcomes of all opening moves, and furthermore knowing how to play well enough to win for each opening.

Thus the Hex swap rule raises two questions: (i) what is the theoretical outcome for each opening move, and (ii) among all opening moves, what are the comparative difficulties of playing perfectly after the move? In this paper we present a computer algorithm that we use to answer these questions for the game of Hex played on a 7×7 board. In order to describe our algorithm and the ideas behind it, we first need to introduce some terminology.

We begin with game-state notation. The terms we use in this paper are consistent with those used in [14]. With respect to Hex, a *board-state* B describes a particular placement of some number of black stones and some number of white stones, such that each cell has at most one stone. We assume no constraint on the relative number of stones of each colour, as the game may have started with a handicap advantage for one of the players; also, we introduce a form of game-state analysis that occasionally requires the placement of extra stones on the board. The *empty board-state* has no stones on the board. A *k-opening* is a board-state with

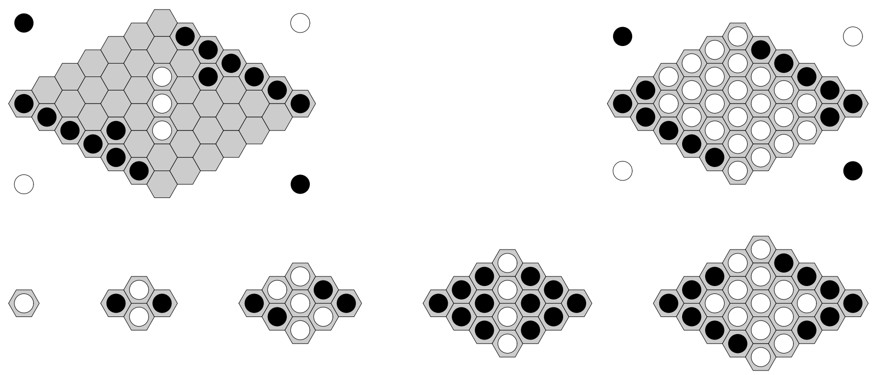


Fig. 2. Previously known 1-opening results. The stone on each cell indicates the winner with perfect play if White's first move is to that cell. For cells with no stone, the winner was not previously known. The 6×6 results were reported by Enderton [9] and verified by Van Rijswijck by computer [24]. The 7×7 results were obtained by Yang et al. by hand [27–29].

exactly k stones on the board. A *game-state*, or simply a *state*, $G = (B, P, Q)$ is defined by specifying a board-state B, the player P with the next move, and the opponent Q of player P. In the definition of game-state, notice that it would be sufficient to list only the player whose turn it is to play next; we list both players in the definition, since the opponent of a player is often explicitly mentioned in our proofs and discussions.

We say that a player *wins* a game-state if there exists a winning strategy for that player in that game-state. Hex cannot end in a draw, so for any game-state exactly one of the players wins the game-state. The *value* of a game-state is the player who has a winning strategy; thus for any fixed board-state B the value of $G = (B, P, Q)$ is either P or Q.

A state is *solved* if its value is known, and *explicitly solved* if a winning strategy is known. As we have already remarked, for the empty board-state on arbitrarily large boards, Hex has been solved but not explicitly solved.

In this paper we present an algorithm that explicitly solves arbitrary Hex states. The worst-case running time of our algorithm is exponential in the number of cells in the board, which is not surprising given that solving arbitrary Hex states is PSPACE-complete [21]. Our algorithm is fast enough to solve 7×7 states in a reasonable amount of time, while solving 8×8 states is currently beyond reach. As a benchmark for the efficiency of our algorithm, we solve all 7×7 1-openings. Previously known 1-opening results are summarized in Figure 2.

In solving these 1-openings our algorithm constructs proof-trees whose terminal nodes correspond to game-states in which a winning virtual connection is detected; since such virtual connections are typically detected many moves before the actual end of the game, each proof-tree found by our algorithm is a proper subset of the complete game-tree, in which each leaf node corresponds to a game-state in which one player has a complete winning chain. Since the size of the proof-trees we find can be used as a measure of the difficulty of playing the opening perfectly, the data we present can be used to answer the two questions raised earlier.

Our results yield the first computer solution of any Hex state on a 7×7 or larger board. Solving Hex states on 5×5 or smaller boards is computationally routine; for larger boards, the problem is more challenging. Enderton reported the values of all 6×6 1-openings, although with no explanation of how he obtained his results [9]. Van Rijswijck independently verified these results by computer, using an alpha-beta search guided by a Hex-specific evaluation function. By providing useful move ordering, his heuristic function evaluation led to the discovery of winning moves sooner than by using unguided search; in this way, his algorithm solved all 1-openings and many longer openings [24,23]. As this method was not strong enough to solve 7×7 states, Van Rijswijck further described but did not implement an alternative recursive algorithm [25]. Recently Yang et al. solved by hand several 7×7 1-openings [27–29], one 8×8 1-opening [26] and one 9×9 1-opening [26].

Our algorithm solves an arbitrary Hex state by computing a winning virtual connection according to dynamic-state composition rules. Following the recursive game-tree search proposed by Van Rijswijck, our algorithm is enhanced by the computation of fixed-state virtual connections; additionally, some new Hex move domination and fill-in results allow significant pruning of the game-tree.

Before presenting our algorithm in §4 and our 7×7 results in §5, we provide necessary background information on virtual connections in §2 and domination and fill-in in §3.

2 Virtual Connections

Roughly, a *connection subgame* in Hex is a subgame in which one of the players can form a connection between two specified sets of cells. If the player can connect the two sets even if the opponent moves first, the connection subgame is called a *virtual connection*; if the player must have the first move in order to guarantee the connection, the connection subgame is called a *weak connection* or a virtual semiconnection.

We now define these terms more formally. Our definitions are essentially those from [3], although our notation is slightly different.

We consider each of the four boundary sides of the board as an occupied set of cells. Let S, X, Y be pairwise non-intersecting sets of cells such that the cell set forming X is connected (namely, for any two cells in X, there is a cell-to-cell path that stays in X), the cell set forming Y is connected, and all cells in S are unoccupied. A P-stone is a stone belonging to P. For a fixed board-state B and a player P, the *subgame* $P(B{:}X{,}S{,}Y)$ is the game of Hex restricted to playing in S, where P wins by forming a chain of P-stones connecting X and Y. $P(B{:}X{,}S{,}Y)$ is a *weak connection for* P if P has a winning strategy for this subgame assuming that P plays first, and a *virtual connection for* P if P has a winning strategy for this subgame even if P's opponent plays first. For a virtual connection or weak connection $P(B{:}X{,}S{,}Y)$, S and X, Y are usually referred to as the *carrier* and *ends* respectively.

These notions are illustrated in Figure 3. For example, the leftmost diagram in this figure shows a Black weak connection between the black stone and the bottom-right side: if each of the dotted cells is unoccupied and it is Black's turn to move, then Black can force a connection by moving to the white-dotted cell. The next two diagrams in this figure show two more Black weak connections. Notice that the common intersection of the cell sets that form these three Black weak connections is empty. This implies that the union of the three

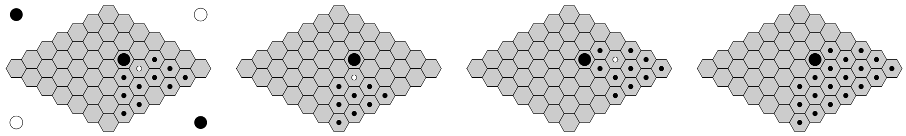


Fig. 3. Three weak connections and a virtual connection.

cell sets, indicated in the rightmost diagram in the figure, forms a Black virtual connection from the black stone to the bottom right side: if all the dotted cells in this diagram are unoccupied and it is White's turn to move, then whatever White does will leave all the cells of at least one of these weak connections unoccupied with Black to move; Black can then move to the white-dotted cell of that weak connection and force a connection.

Although defined slightly differently by different authors, virtual connections have long been recognized as being central to Hex strategy. References to virtual connections permeate the Hex literature, where they are also referred to as "connections" or "safe groups". For example, various forms of virtual connections are discussed by Berge [6,15] [1] , Schensted and Titus [22], Browne [8], and Anshelevich [3].

Virtual-connections are useful in solving Hex states since, when accompanied by an explicit strategy, a virtual connection serves as a proof or certificate that a pair of cells can be connected. The most important virtual connections and weak connections are those which connect a player's two sides, since such a connection occurs by definition if and only if there is an associated winning strategy. We call such subgames *win-links* and *win-weaklinks* respectively. We use $P(B{:}-, S, -)$ to denote such a connection joining P's two sides.

Connection subgames are particularly useful in Hex end-game analysis. For example, Berge observed that a move to a cell that is not in the carrier of some opponent win-weaklink is a losing move, since on the next move the opponent can turn the win-weaklink into a win-link.

For a fixed game-state, we call the intersection of the set of all unoccupied board cells with all known opponent win-weaklink carriers a player's *must-play*. For a player X and an unoccupied cell c of a board-state B, we define $B + c_X$ as the state obtained from B by adding an X-stone at c. Using this notation, Berge's remark can be restated as follows:

Theorem 1 *(Berge [6,15]) If c is not in P's must-play for (B, P, Q) then Q wins $(B + c_P, Q, P)$.*

Proof. If c is not in P's must-play for (B, P, Q) then there is a win-weaklink $Q(B{:}-, S, -)$ such that c is not in S. Since $Q(B{:}-, S, -)$ is a win-weaklink, there is some cell d in S such that $Q(B + d_Q{:}-, S - d_Q, -)$ is a win-link. Since c is not in S, $Q(B + c_P + d_Q{:}-, S - d_Q, -)$ is also a win-link, so Q wins $(B + c_P, Q, P)$. □

Notice that the computation of a mustplay region is a form of null move analysis, as it involves the consideration of what can occur if a player skips a turn.

A useful feature of virtual connections is that smaller ones can be combined in various ways to form larger ones. The knowledge of this fact is as old as Hex itself; for example, it is

[1] A translated version of [6] appears in [15]

discussed in detail in [6]. Recently, Anshelevich [1,3,2] used the following set of combining rules to compute connection subgames in an inductive or "bottom-up" fashion.

Theorem 2 *(Anshelevich [1,3,2])*

- AND-*rule: Let $P(B{:}X,S,Y)$ and $P(B{:}Y,T,Z)$ be virtual connections with X and S each disjoint from T and Z. Then (i) if each cell of Y has a P-stone then $P(B{:}X,S \cup T \cup Y,Z)$ is a virtual connection (ii) if Y consists of a single cell then $P(B{:}X,S \cup T,Z)$ is a weak connection.*
- OR-*rule: Let $P(B{:}X,S_j,Y)$ be a weak connection for $j = 1, \ldots, t$ such that the intersection of the sets S_j is empty. Then $P(B{:}X,S,Y)$ is a virtual connection, where S is the union of the sets S_j.* □

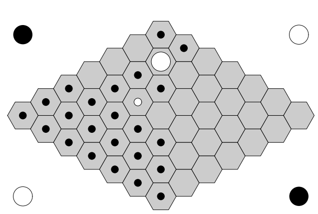
Notice that part (i) of the AND-rule is essentially a restatement of the previous Berge observation. Also notice that this set of rules is static, in that it yields a class of connection subgames for a fixed state. Anshelevich pointed out that this set of rules is not sufficient to establish all virtual connections of a state, and is thus not strong enough to solve all Hex states; however, the rules do yield a sufficiently large class of virtual connections to provide an effective subroutine of a strong Hex-playing program [1,3,2].

As Van Rijswijck observed, an alternative method of computing connection subgames is to proceed through the game-tree dynamically. For a state $G = (B, P, Q)$ and for each unoccupied cell x of B, define $G + x_P$ as $(B + x_P, Q, P)$; when the player P is clear from context we shall sometimes write $G + x$ in place of $G + x_P$. Van Rijswijck's discussion in [25] suggests the following set of rules. The symbol ϕ denotes the empty set. For a player P, a P-winning chain is a chain of P-stones that connects P's two sides.

Theorem 3 *(Van Rijswijck [25]) Consider a board-state B and players P, Q. If B has a P-winning chain then $P(B{:}-, \phi, -)$ is a win-link. If B has no P-winning or Q-winning chain then P wins (B, P, Q) if and only if P wins $(B + c_P, Q, P)$ for some unoccupied cell c of B; in particular*

- *if P wins $(B + c_P, Q, P)$ with a win-link $P(B + c_P{:}-, S, -)$ then $P(B{:}-, S + c_P, -)$ is a win-weaklink,*
- *if, for each unoccupied cell c^j, Q wins $(B + c^j_P, Q, P)$ with some win-link $Q(B + c^j_P{:}-, S^j, -)$ then the intersection of the sets S^j is empty,*
- *if there is a set C of unoccupied cells such that, for each c^j in C, Q wins $(B + c^j_P, Q, P)$ with some win-link $Q(B + c^j_P{:}-, S^j, -)$ and the intersection of the sets S^j is empty, then $Q(B{:}-, S, -)$ is a win-link, where S is the union of the sets S^j.*

Proof (sketch). The proof follows by elementary game-theory arguments from the fact that



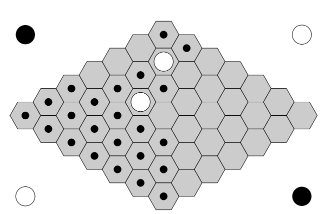


Fig. 4. A white win-weaklink . . . and a corresponding win-link.

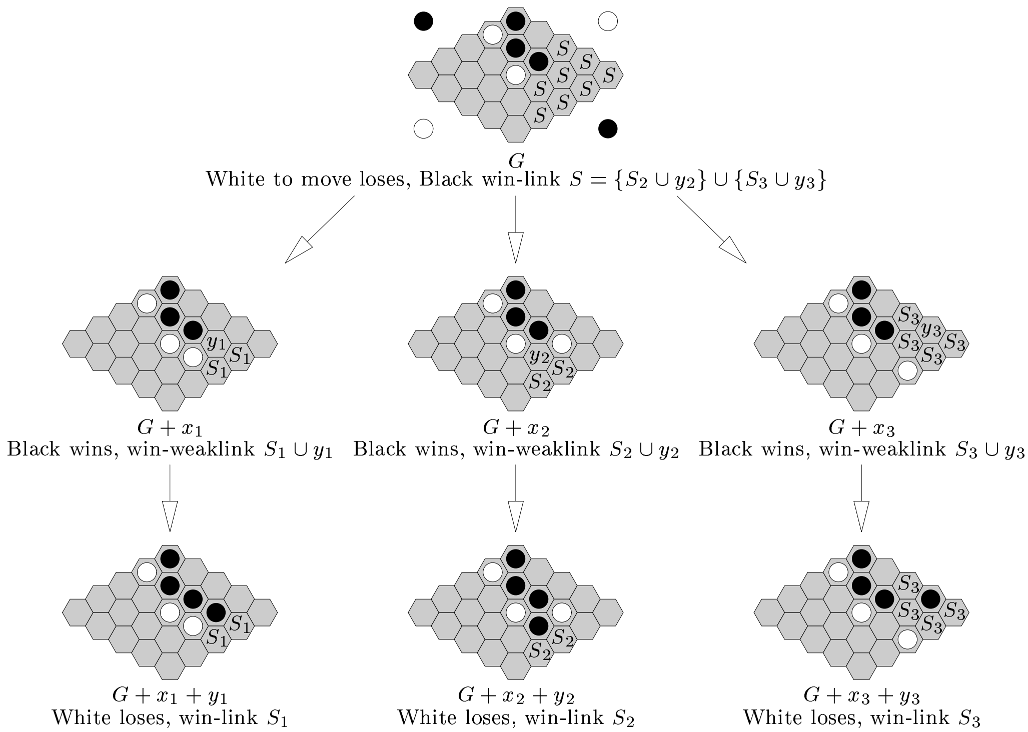


Fig. 5. An example illustrating Theorem 3.

any Hex state has exactly one winner. This fact in turn requires some care to prove; see for example [4,10,14]. □

Figure 5 illustrates Theorem 3. The root state G is a loss for White. Three of White's possible moves are explored. In each state $G + x_i$, the move y_i yields a black win; the resulting state $G + x_i + y_i$ has a black win-link carrier S_i, so $G + x_i$ has a black win-weaklink with carrier $S_i \cup y_i$; this win-weaklink implies that x_i loses in G, and moreover that *any* white move outside of $S_i \cup y_i$ loses. The intersection of these three win-weaklink carriers is empty. Indeed, the intersection of just the two win-weaklink carriers $S_2 \cup y_2$ and $S_3 \cup y_3$ is already empty, which means that the union of these two win-weaklink carriers is a black win-link in G. It also means that the exploration of these two branches of the game-tree is sufficient to determine that White loses G; the consideration of any other move is unnecessary.

Notice that these rules are by their definition complete: they can be used to solve any arbitrary Hex state.

From a computational point of view, the difficulty with the two preceding sets of rules is that the number of possible connection subgames that can be computed in this way is exponential in the number of cells. For this reason, an exhaustive approach to computing connection subgames based on either rule set will be forced to limit the number of intermediate connection subgames computed.

For both the static and dynamic computational processes, what is needed is some way of distinguishing those intermediate connection sets that are critical to solving the particular state from those that are not. We close this section by giving evidence that this is likely to be a difficult problem.

Assume that at some point in a computation involving the dynamic rules it is discovered that player P has no winning move in a state G. It follows that P's opponent Q has a win-weaklink S_x after each possible move x by P and that the union of any collection of these win-weaklinks which have an empty intersection establishes a win-link for Q. If G is an intermediate state in the process of solving some earlier state, then P needs to compute such a win-link to pass back to the state that gave rise to G. It is reasonable to expect that a useful win-link to pass back would be one that has the smallest number of cells, among all such possible win-links. However, it is also reasonable to expect this problem to be computationally difficult, since it seems to be intimately related to determining the outcome of a Hex game, which we have already noted is PSPACE-complete. Saks observed that this problem is indeed computationally difficult, as we now explain.

The *Minimum-Union Empty Intersection Problem (MUEIP)* is the decision problem that takes as input an integer k together with a set $S = \{S_1, \dots, S_t\}$ of subsets of a finite set V and asks whether there is a subset T of S such that the intersection of the sets in T is empty and the union of the sets in T has size at most k. *Minimum Cover* is the decision problem that takes as input an integer k together with a set $A = \{A_1, \dots, A_t\}$ of subsets of a finite set V and asks whether there is a subset of at most k elements of A whose union is V. Minimum Cover, referred to as Problem [SP5] in the text by Garey and Johnson [13] and sometimes also known as Minimum Set Cover, was shown to be NP-complete by Karp [18].

Theorem 4 *(Michael Saks, private communication) MUEIP is NP-complete.*

Proof. Consider an instance of Minimum Cover, where k, A, and V are as defined above and $n = |V|$. This instance can be transformed in polynomial time into an instance of MUEIP, as follows.

For each index j, let B_j be the set complement (with respect to V) of A_j; also, let $B = \{B_1, \dots, B_t\}$. Observe that the union of k elements of A is equal to V if and only if the intersection of the corresponding k elements of B is empty. Let V' be the set obtained by adding $t(n+1)$ new elements to V. For each index j, let B'_j be the set obtained by adding $n+1$ of the new elements to B_j in such a way that each B_j gets expanded by a set of new elements disjoint from all other new elements. Let $B' = \{B'_1, \dots, B'_t\}$. Observe that a set of k elements of B has empty intersection if and only if the corresponding set of k element of B' has empty intersection, and this occurs if and only if the same set of k elements of B' has empty intersection and union with size at most $k(n+1)+n$.

Since MUEIP is clearly in NP, the theorem follows from the preceding transformation and the fact that Minimum Cover is NP-complete [18]. □

Kiefer recently showed that computing virtual connections using only the static combining rules described earlier is PSPACE-complete [19]. Theorem 4 suggests that computing virtual connections using dynamic combining rules is also likely to be computationally difficult. Thus any algorithm that solves arbitrary Hex states by computing virtual connections dynamically and/or statically will probably need to use some extra game knowledge in order to reduce the complexity of searching through the game-tree. We discuss some such reductions in the next section.

3 Game-State Reduction: Domination and Fill-in

One reason that Hex is a challenge for computers to play or solve is the high branching factor; especially in the early stages of the game, the number of possible moves is high. In this section we describe two kinds of game-state reductions. Each reduction strengthens the algorithm implicitly described by the virtual connection composition rules of the previous section by allowing the game-tree to be pruned. We refer to these reductions as domination and fill-in.

Informally, one move dominates another if the former is at least as good as the latter. Since we are interested here only in determining the value of a game-state, one move is "at least as good as" another if the former yields a win whenever the latter yields a win. Formally, for possible moves u, v from a state $G = (B, P, Q)$, we say that u *dominates* v if P wins $(B + u_P, Q, P)$ whenever P wins $(B + v_P, Q, P)$.

Domination results are useful for our purposes since, in searching the game-tree for a winning move, a dominated move at a particular state can be ignored as long as at least one dominating move at that state is considered. Unfortunately, few Hex domination results are known. One such result is due to Beck, who proved that on an empty board size 2×2 or larger, moving to an acute corner is a losing, and so dominated, move [4]. The board cell coordinate system we use in this paper is shown in Figure 6 and subsequent figures. The two acute corners of the boards of Figure 6 are A1 and G7.

"Fill-in" refers to placing stones on a board that do not alter the value of an associated game-state. Hayward recently established a domination and fill-in result involving a certain three-cell configuration [16]; this is explained in Theorem 5.

Our algorithm uses Theorem 5 in two ways. Firstly, for any state (B, P, Q) with an empty P-triangle, the two P-moves to the side of the triangle can be ignored since they are dominated by the P-move to the tip. Secondly, for any state (B, Q, P) with a P-triangle with a P-stone at the tip and the two side cells unoccupied, each cell of this side pair can be filled with a P-stone without changing the outcome of the game. As can be seen from Figure 12, the second result is particularly useful when combined with our virtual connection computation approach.

The key idea in the proof of Theorem 5 is that if P ever plays at the tip of an empty P-side triangle, it is thereafter pointless for Q to play into either of the two adjacent side cells. The reason for this is that P could immediately reply into the other side cell and make the Q-stone "dead", in the sense that the Q-stone will not be in any minimal winning Q-chain. This idea has recently been developed to establish more general domination and fill-in results; see [14,7] for more details.

For a player P, a P*-side cell* is any cell that borders one of P's two boundary sides, a P-

side pair is a set $\{x, y\}$ of two adjacent P-side cells, and a *P-triangle* is an ordered triple (x, y, t) where $\{x, y\}$ is a P-side pair and t is the unique board cell adjacent to both x and y; t is called the *tip* of the side triangle. A P-triangle is *empty* if each cell of the triangle is unoccupied. See Figure 7.

Theorem 5 *(Hayward [16]) Let B be a board-state with an empty P-triangle (x, y, t). Then a player R in $\{P, Q\}$ wins any one of the four states $(B + t_P, Q, P)$, $(B + t_P + x_P, Q, P)$, $(B + t_P + y_P, Q, P)$, $(B + t_P + x_P + y_P, Q, P)$ if and only if R wins all of them. Also, for each cell z in $\{x, y\}$, P wins $G_t = (B + t_P, Q, P)$ if P wins $G_z = (B + z_P, Q, P)$.*

Proof. Hex is a so-called "regular" game: altering a game-state by adding an extra stone is never disadvantageous for the player whose stone was added [11]. This property can be stated more formally as follows: for each unoccupied cell x of a board-state B and for each player R in $\{P, Q\}$, R wins $(B + x_R, P, Q)$ if R wins (B, P, Q).

The second statement of the theorem follows from the first statement and the regularity of Hex, since P wins $(B+t_P+x_P+y_P, Q, P)$ if P wins $(B+z_P, Q, P)$ and P wins $(B+t_P, Q, P)$ if P wins $(B + t_P + x_P + y_P, Q, P)$.

Consider then the first statement of the theorem. Since Hex cannot end in a draw [11], it suffices to prove this statement for one of the two players, say for $R = P$. Also, regularity simplifies the task of proving this statement, since by regularity P wins $(B+t_P+x_P+y_P, Q, P)$ if P wins $(B + t_P + y_P, Q, P)$ or $(B + t_P + x_P, Q, P)$, and P wins $(B + t_P + y_P, Q, P)$ and $(B + t_P + x_P, Q, P)$ if P wins $(B + t_P)$. Thus to prove the first statement of the theorem it suffices to show that

$$P \textit{ wins } G' = (B + t_P, Q, P) \textit{ if } P \textit{ wins } G = (B + t_P + x_P + y_P, Q, P). \qquad (*)$$

We prove $(*)$ by constructing a strategy S' for P and G' from a strategy S for P and G such that S' wins G' if S wins G. S' is defined as follows:

(i) in response to a Q-move to either cell of $\{x, y\}$, P plays into the other cell
(ii) in response to a Q-move not to either cell of $\{x, y\}$, if there is a P-chain connecting the P-stone at cell t to P's other side of the board then P plays into either cell of $\{x, y\}$ to win the game; otherwise P plays as in the state of S obtained from the current state by changing the status of cells x and y so that both are occupied by P-stones.

Now suppose that S wins G for P and consider a continuation of G' in which P plays according to S'. It is easy to show by induction on the number of unoccupied board cells of B' that for each state $G'_j = (B'_j, P, Q)$ reached in this continuation, the associated state $G_j = (B_j, P, Q)$ reached by following S has the property that B_j and B'_j are equal except possibly for the contents of cells x and y; these two cells are both P-stones in B_j whereas in B'_j they are either both unoccupied or contain at least one P-stone.

We finish the proof by contradiction: suppose that there is a winning Q-chain in B'_j. Notice that such a chain contains at most one cell of x or y, since at least one cell of $\{x, y\}$ is unoccupied or has a P-stone. Suppose that the chain contains one of these two cells, say y. Since the six neighbours of y include x, t, and two cells from P's boundary and since none of these four cells contains a Q-stone, the two other neighbours of y are the neighbours of

y on the Q-chain. Since these two neighbours are themselves adjacent, removing y from the Q-chain leaves a shorter winning Q-chain. Thus the existence of a winning Q-chain in B'_j implies the existence of a winning Q-chain in B'_j that does not include any cell from $\{x, y\}$. But S and S' are the same with respect to all cells except x and y, so the existence of such a chain implies the existence of a winning Q-chain in B_j, contradicting our earlier assumption that S wins G for P. It follows that there is no winning Q-chain in B'_j, so it follows by induction on the number of unoccupied cells of B' that S' wins G' for P. □

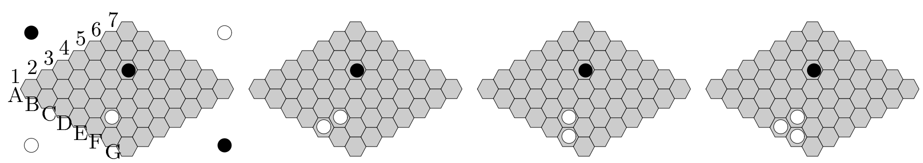


Fig. 6. Illustrating the first part of Theorem 5. Applying this result to the white side triangle with tip E2, it follows that a player has a winning strategy for one of these board-states if and only if that player has a winning strategy for all of these board-states.

4 The Algorithm

Our algorithm Solver combines the approaches suggested by Theorems 1, 2, 3, and 5. For a player P with opponent Q, the algorithm solves a state $G = [P, B]$ as follows.

Algorithm Solver

Input: A Hex game-state $G = (B, P, Q)$.

Output: The value of G together with the carrier of a win-link for G.

For each side triangle for which the first statement of Theorem 5 applies, add stones to the appropriate side cells; call the resulting board B^*. Statically compute virtual connections and weak connections. If a win-weaklink $P(B{:}-, S, -)$ is detected then return (P, S); if a win-link $Q(B{:}-, S, -)$ is detected then, if the win-link uses the the tip of a triangle whose side was filled in then add the side cells to S, and return (Q, S).

Otherwise, let T be the set consisting of all carriers of all Q-win-weaklinks found for G and let R be the P-mustplay region defined with respect to T, namely the intersection of the set of unoccupied cells of B with each of the elements of T. Remove from R any side-cells from any empty P-triangle. While R is not empty, pick a cell x in R, and do the following:

> *Let B^*_x be the state obtained from B^* by adding a P-stone at x and, if x was the tip of an empty P-triangle before this move, filling in the triangle. Recursively solve $G_x = (B^*_x, Q, P)$.*

If P wins G_x, say with win-link carrier X, then add to X the cell x as well as the two associated side-cells if x was the tip of an empty P-triangle, and return (P, X).
If Q wins G_x, say with win-weaklink X, then add X to T.

If execution reaches this point then the while loop terminated without discovering a win-weaklink for P, so the union U of elements of T is a carrier for a win-link for Q, so return (Q, U).

A sample execution of the algorithm is described in Figures 7 through 9. The correctness of our algorithm follows directly from the previous theorems.

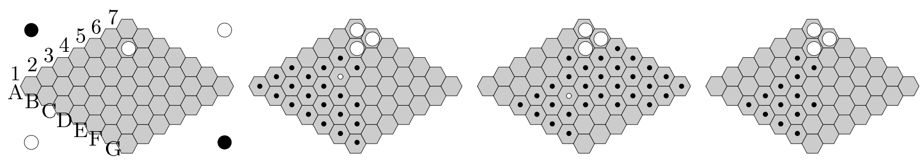


Fig. 7. SOLVER solves b6: initialization. After the initial move (left), the game-state is reduced by applying Theorem 5 and adding white stones to the two side-cells of the white side-triangle with tip b6. In the resulting state, White has two win-weaklinks (center-left and center-right) whose resulting intersection yields a 13-cell black mustplay region (right). If Black has a winning move, it has a winning move to one of these 13 cells.

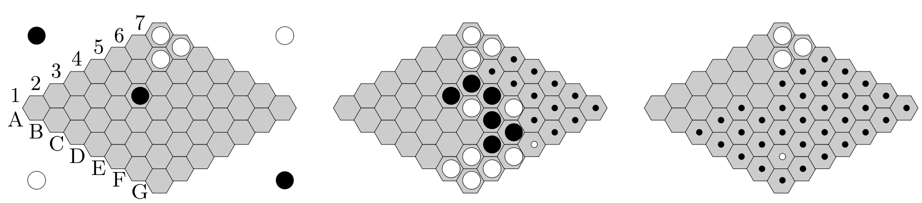


Fig. 8. SOLVER solves b6-c4. As shown by the SOLVER b6 recursion tree in Figure 13, c4 is the first black response considered to the white b6 opening (left). Following the topmost path b6-c4-f2-d5-d4-c5-e5-e4-g3-f3-g2-f4 in the recursion tree and applying Theorem 5 after f2 leads to the first solved state (center, with white win-weaklink); since f4 is a leaf of the recursion tree, the white win-weaklink here was discovered statically. SOLVER continues solving the c4-subtree, eventually determining that c4 is a black loss (right, with white win-weaklink). This win-weaklink does not contain c4 or b5, so, of the 13 possible b6-responses corresponding to the initial black mustplay region described in Figure 7, 11 moves remain to be checked.

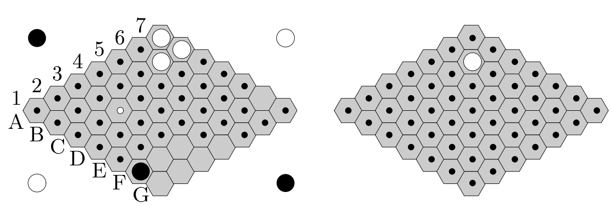


Fig. 9. SOLVER solves b6: conclusion. The move to f1 is the last black reply considered in response to the white b6 opening (left, with white win-weaklink), since after the discovery of this last white win-weaklink, the set of such win-weaklinks has empty intersection. The union of these 11 white win-weaklinks gives the final win-link for White (right).

5 SOLVER 7×7 1-Opening Solutions

As mentioned earlier, SOLVER is strong enough to solve arbitrary 7×7 states. Figures 10 and 11 summarize the results obtained by running SOLVER on all 49 7×7 1-openings. Figures 13 and 14 show the SOLVER recursion trees from two of these executions, while Figure 15 shows a longest line of play from each of the 49 solutions. Each execution was performed on a single processer machine[2]; in each case, the run time was roughly proportional to the number of nodes in the SOLVER recursion tree, taking about one minute for the five 1-openings with the smallest node-counts, and about 110 hours for the 1-opening with the largest node-count; the total run time for all 49 1-openings was about 615 hours. A listing of all 49 trees (including a tree viewer) is available at `http://www.cs.ualberta.ca/~hayward/hex7trees`.

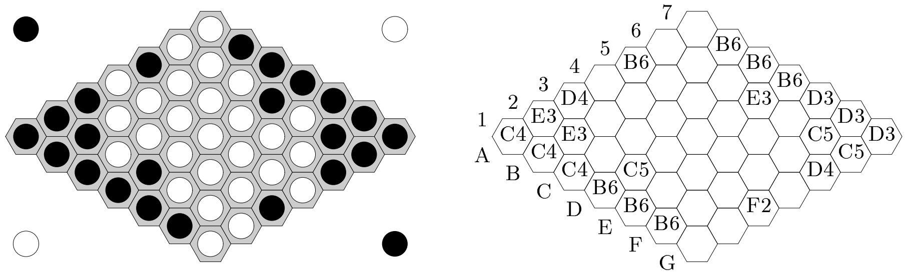


Fig. 10. All 7×7 1-opening results, as found by SOLVER. The stone on each cell indicates the winner with perfect play if White's first move is to that cell. The move indicated on each losing cell is the winning countermove discovered.

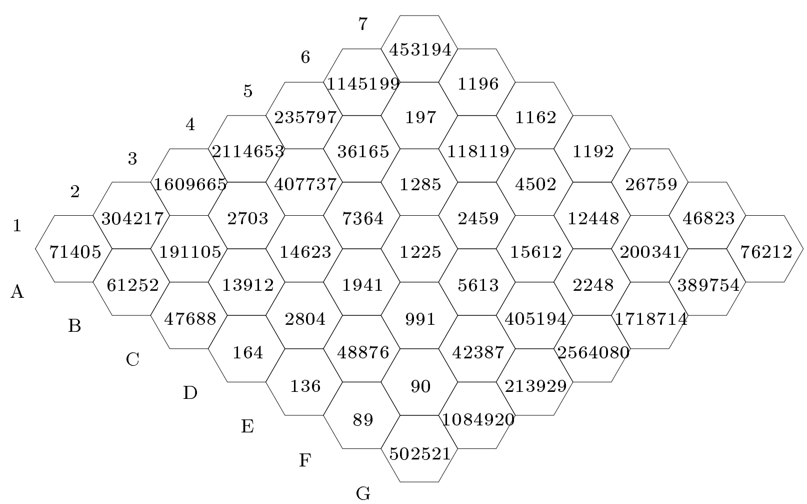


Fig. 11. Number of nodes in the SOLVER 7×7 1-opening recursion trees.

For any size Hex board, the set of winning open-move cell locations is symmetric with respect to reflection through the center of the board. Notice that the SOLVER node-counts do not share this symmetry, as neither the order in which SOLVER considers moves nor SOLVER's

[2] The program was compiled with gcc 3.1.1 and run on an AMD Athlon 1800+ MHz processor with 512 MB memory running Slackware Linux.

implementation of the static computation of virtual connections is designed to reflect this symmetry.

Figure 12 demonstrates the relative strength of the three key parts of our algorithm, namely virtual connection computation, side-triangle move domination, and side-triangle fill-in, by showing SOLVER node-counts when various of these features are turned off. In particular, notice that adding side-triangle fill-in to virtual connection computation results in a substantial decrease in the number of nodes considered, while further adding side-triangle domination has little effect.

As expected, SOLVER with domination and fill-in usually needs to explore fewer game-tree nodes to solve a state than SOLVER-D, which uses fill-in but not domination; however, there are a few exceptions. In particular, for each of the 6×6 openings d4, e5, f1, and f4 SOLVER-D explored slightly fewer nodes than SOLVER. One reason for this anomalous behaviour is that fill-in can cause the carrier of the win-link returned by our algorithm to be unnecessarily large.

For example, if in a certain state there is an empty side triangle for which both the tip and a side cell are winning moves, then the carrier of the win-link discovered by making the move to the tip will contain both side cells and so be a proper superset of the carrier discovered by making the move to the side cell; in this case, if domination is not in effect the smaller carrier will be discovered if move ordering happens to process the side cell before the tip, and this smaller carrier may result in fewer recursive calls being made.

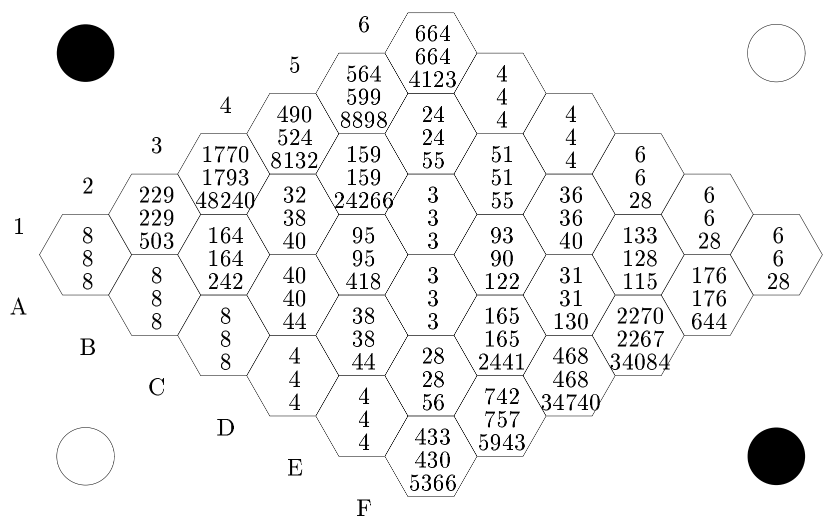


Fig. 12. Number of nodes in the 6×6 1-opening recursion trees for SOLVER (top entry), SOLVER−D, namely SOLVER without side-cell domination (middle entry), and SOLVER−FD, namely SOLVER with neither side-cell domination nor fill-in (bottom entry). While corresponding data were obtained for some 7×7 1-openings, SOLVER−FD in particular was too slow to execute for all such openings. For example, the b7 SOLVER−FD tree has 824796 nodes, compared to only 1196 for SOLVER.

In comparing the winning 7×7 opening moves (Figure 10) with winning opening moves on smaller boards (Figure 2), some features common to each of these $n \times n$ boards are worth noting. For example,

- the n cells on the short diagonal (obtuse corner to obtuse corner) are all first-player winning openings,
- the $n-1$ cells on each of the first-player's sides (except for the cell in the short diagonal) are all first-player losing openings.

It would be of considerable interest to show whether these results hold in general, especially if the proof is positive (as opposed to say a single counterexample), since to date, for arbitrarily large $n \times n$ boards,

- no particular move is known to be a first-player win,
- the only moves that are known to be first-player losses are for $n \geq 2$, the two acute corner cells [4], and for $n \geq 3$, the two cells each in the first-player's side and adjacent to the acute corner cell [5].

6 Conclusions and Open Problems

We have shown how combining static and dynamic virtual connection computation methods with some move domination results yields an algorithm strong enough to solve arbitrary 7×7 Hex states. A next step is to design an algorithm strong enough to solve 8×8 states; preliminary results suggest that this is considerably more difficult and that further techniques will be required. Another direction is to use SOLVER to gather 7×7 information that can be used to find better move ordering heuristics for Hex game-tree search on (much) larger boards; for example, such data would be useful in analyzing any local configuration with effective board size at most 7×7.

Acknowledgements

The authors gratefully acknowledge the support of the Natural Sciences and Engineering Research Council of Canada, the University of Alberta Research Excellence Envelope, and the University of Alberta GAMES Research Group. Also, the fourth and fifth authors gratefully acknowledge the support of an NSERC Summer Undergraduate Research Award.

We thank Michael Buro, Maryia Kazekevich, Martin Müller, and Jonathan Schaeffer for their assistance in sustaining the Mongoose Hex project that was the starting point for this work. We also thank the referees for their detailed comments on earlier versions of this article.

References

[1] Vadim Anshelevich. The Game of Hex: An Automatic Theorem Proving Approach to Game Programming. *Proc. 17th National Conf. on AI (AAAI-2000)*, pages 189–194, 2000.

[2] Vadim Anshelevich. The Game of Hex: The Hierarchical Approach. In Richard J. Nowakowski, editor, *More Games of No Chance*, volume 42 of *MSRI Publications*, pages 151–165. Cambridge

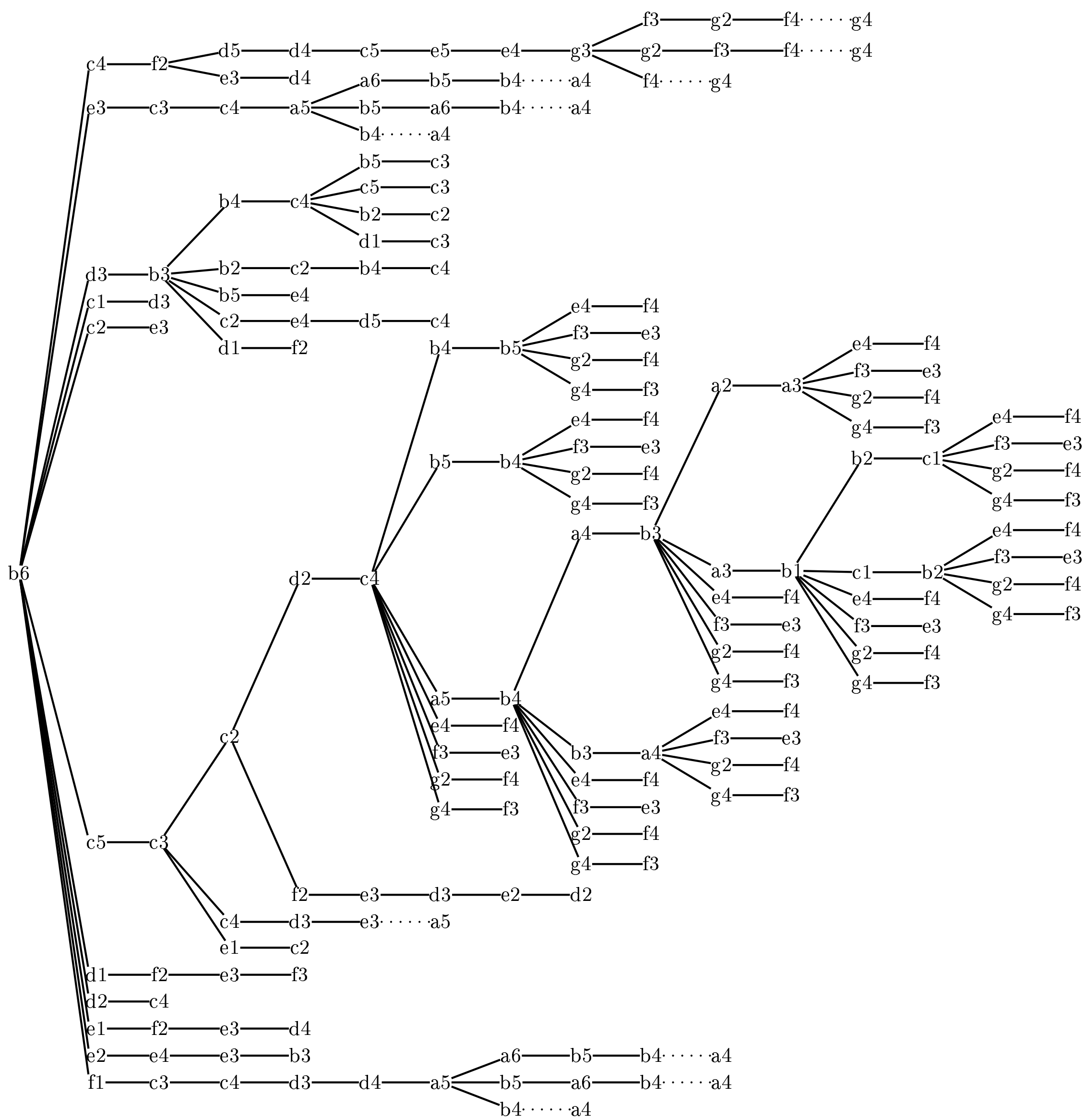


Fig. 13. The SOLVER recursion tree for the 7×7 opening White-b6 (with the ten nodes connected by dotted edges added so that every path ends with a winning move). For each node, the order of child generation is top-to-bottom. Each SOLVER recursion tree is a subtree of the complete game-tree, as the only replies to a winning move which appear in the recursion tree are those replies in that state's mustplay region. For example, consider for the tree shown here the state G after White plays b6. As shown in the last diagram in Figure 8, White has a win-weaklink created by playing at c4 which does not contain b5; thus b5 is not in the Black mustplay region for G, so SOLVER never needs to consider the Black move to b5, so b5 does not appear as a child of b6 in this recursion tree. Notice from the tree shown here that in solving the b6 opening the selection of d2 as the first move considered at the b6-c5-c3-c2 subtree was unfortunate, as d2 leads to a White loss whereas f2, the second move considered, leads to a White win. If f2 had been considered first, the d2 subtree would not have been explored, and the resulting recursion tree would have had only 97 nodes instead of 197.

University Press, Cambridge, 2002.

[3] Vadim Anshelevich. A Hierarchical Approach to Computer Hex. *Artificial Intelligence*, 134:101–120, 2002. (Also in: J. Schaeffer and J. Van den Herik, (Eds). *Chips, Computers and Artificial Intelligence, Elsevier*, 141-160, 2002.).

[4] Anatole Beck, Michael N. Bleicher, and Donald W. Crowe. *Excursions into Mathematics*, chapter Games, pages 317–387. Worth, New York, 1969.

[5] Anatole Beck, Michael N. Bleicher, and Donald W. Crowe. *Excursions into Mathematics: the Millennium Edition*, chapter Appendix 2000. A. K. Peters, Natick, 2000.

[6] Claude Berge. L'Art Subtil du Hex. Manuscript, 1977.

[7] Yngvi Björnsson, Ryan Hayward, Michael Johanson, and Jack van Rijswijck. Dead Cell Analysis in Hex and the Shannon Game. Manuscript, `www.cs.cmu.edu/~hde/hex/hexfaq/`, submitted to Proc. Graph Theory 2004, 2004.

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

[9] Bert Enderton. Answers to infrequently asked questions about the game of Hex. `www.cs.cmu.edu/~hde/hex/hexfaq/`, 1995.

[10] David Gale. The Game of Hex and the Brouwer Fixed Point Theorem. *American Mathematical Monthly*, 86(10):818–827, 1979.

[11] Martin Gardner. Mathematical Games. *Scientific American*, 197, 1957. July pp. 145–150; August pp. 120–127; October pp. 130–138.

[12] Martin Gardner. *The Scientific American Book of Mathematical Puzzles and Diversions*, chapter The game of Hex, pages 73–83. Simon and Schuster, New York, 1959.

[13] Michael R. Garey and David S. Johnson. *Computers and Intractability.* W.H. Freeman and Co., San Francisco, 1979.

[14] Ryan Hayward and Jack van Rijswijck. Hex and Combinatorics. Manuscript, `www.cs.ualberta.ca/~hayward/publications.html`, submitted to Discrete Mathematics, 2003 (revised Oct. 2005).

[15] Ryan B. Hayward. Berge and the Art of Hex. Manuscript, `www.cs.ualberta.ca/~hayward/publications.html`, (to appear in A. Bondy, V. Chvátal, eds., A biography of Claude Berge, Princeton University Press, in preparation), 2003.

[16] Ryan B. Hayward. A Note on Domination in Hex. Manuscript, `www.cs.ualberta.ca/~hayward/publications.html`, January 2003.

[17] Ryan B. Hayward, Yngvi Björnsson, Michael Johanson, Morgan Kan, Nathan Po, and Jack van Rijswijck. Solving 7×7 Hex: Virtual Connections and Game-state Reduction. In H. Jaap van den Herik, Hiroyuki Iida, and Ernst A. Heinz, editors, *Advances in Computer Games*, volume 263 of *IFIP International Federation for Information Processing*, pages 261–278. Kluwer Academic Publishers, Boston, 2003.

[18] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[19] Stefan Kiefer. Die Menge der Virtuellen Verbindungen im Spiel Hex ist PSPACE-völlstandig. Studienarbeit Nr. 1887, Facultät für Informatik, Electrotechnik, und Informationstechnik, Universität Stuttgart, July 2003.

[20] Sylvia Nasar. *A Beautiful Mind.* Touchstone, New York, 1998.

[21] Stefan Reisch. Hex ist PSPACE-vollständig. *Acta Informatica*, 15:167–191, 1981.

[22] Craige Schensted and Charles Titus. *Mudcrack Y and Poly-Y.* Neo Press, Peaks Island, Maine, 1975.

[23] Jack van Rijswijck. Queenbee's home page. `www.cs.ualberta.ca/~queenbee`.

[24] Jack van Rijswijck. Computer Hex: Are Bees better than Fruitflies? Master's thesis, University of Alberta, Edmonton, Canada, 2000.

[25] Jack van Rijswijck. Search and Evaluation in Hex. Technical report, University of Alberta, 2002. `www.cs.ualberta.ca/~javhar/research.html`.

[26] Jing Yang. Jing yang's web site. `www.ee.umanitoba.ca/~jingyang`, 2003.

[27] Jing Yang, Simon Liao, and Mirek 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.

[28] Jing Yang, Simon Liao, and Mirek Pawlak. Another solution for Hex 7x7. Technical report, University of Manitoba, Winnipeg, Canada, 2002. `www.ee.umanitoba.ca/~jingyang/TR.pdf`.

[29] Jing Yang, Simon Liao, and Mirek Pawlak. New Winning and Losing Positions for 7x7 Hex. In *Computers and Games*, volume 2883 of *Lecture Notes in Computer Science*, pages 230–248. Springer, 2002.

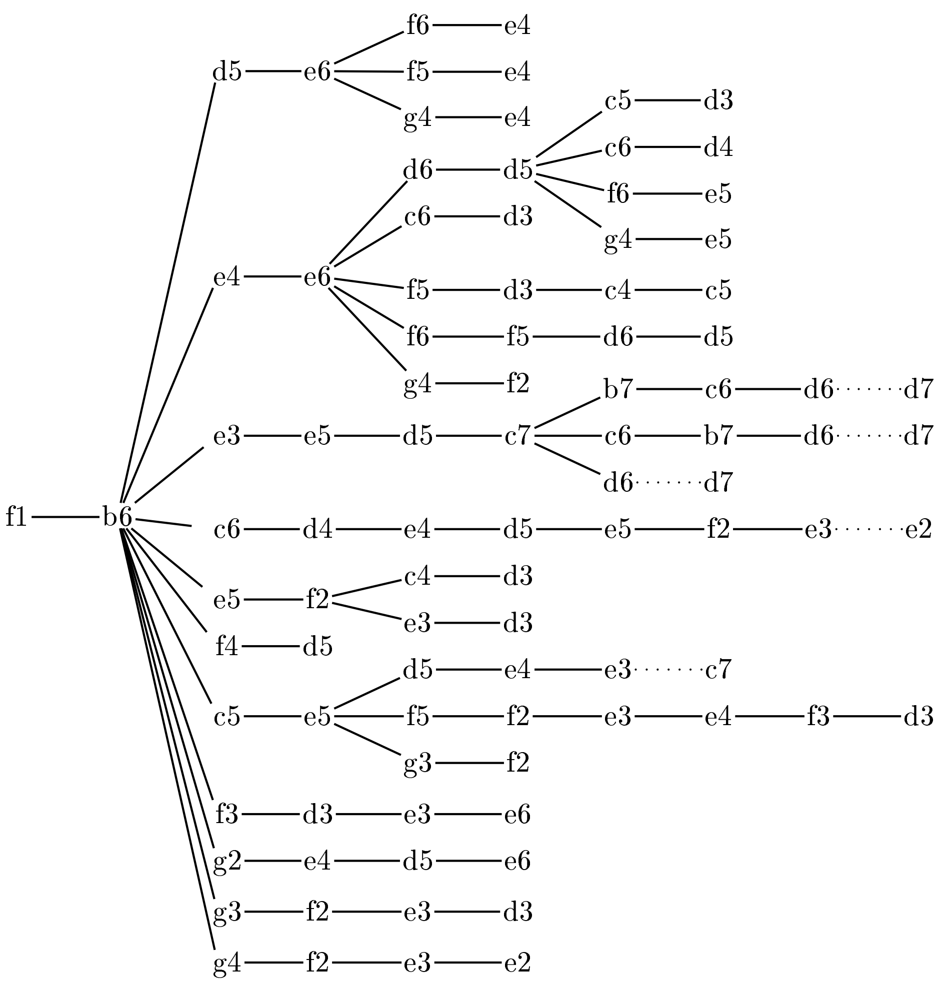


Fig. 14. The Solver recursion tree for the 7×7 opening White-f1 (with the five nodes connected by dotted edges added so that every path ends with a winning move). For each node, the order of child generation is top-to-bottom. Notice that the f1-b6 subtree, which establishes that b6 is a winning countermove to f1, is paradoxically smaller than the b6 subtree shown in Figure 13, in part because of the move ordering here is more fortunate than there. In this f1-tree, whenever it is White's turn to play, the first move considered turns out to be a winning move; this is not the case in the b6 tree shown in Figure 13.

```
1     2     3     4     5     6     7     8     9     10    11    12    13    14    15    16    17    18    19

a1 c4 a6 f2 d3 b6 e4 d4 e3 e5 d5 e1 e2 f1 a3 b2 c2 c3 a4 d2 b5 a2 b3 a5 b4 a7 c5 c7 b7 c6 d6 d7
a2 e3 f4 d5 c5 d3 g2 e6 e5 c7 d6 d7 g5 f7 e7 f6 b6 d2 c6 g1 f2 f1 e1 e2 a4 b3
a3 d4 c5 b4 c4 d2 c3 c2 b3 b1 b2 c1 e4 f2 e2 d5 d3 b7 a7 b6 a6 b5 e3 e5 c6 c7 d6 d7
a4 b3 d3 d5 c5 d2 c2 b7 a7 b6 a6 b5 e5 c3 a5 b1 a3 b2 e4 f1 e2 e1 g1 f2 g2 f3 g3 f4 g4 f5 g5 f7 f6 e7 e6 d7 c4
a5 b6 c5 e3 f4 d5 c6 c3 d3 e1 e2 f1 g1 f2 d2 d1 g2 f3 g3 e6 d6 e5 g5 f7 e7 f6 d4 e4
a6 f1 c3 b5 c6 d4 c4 c5 a5 b4 a4 b3 a3 a7 b6 b1 b2 c1 c2 d1 e1 d2 e2 d3 e4 d5 e5 e3 g2 f2 g1 f3 g3 f4 g4
a7 d5 c3 d2 c2 c4 a5 b4 a4 a6 e4 d4 e5 e3 b5 b3 a3 b1 b2 b6 c6 c5 g2 f3 g3 g1 f2 f1 e2 d3 e1 f4 g4
b1 c4 a6 f2 d3 b6 e4 d4 e3 e5 d5 e1 e2 f1 a3 b2 c2 c3 a4 d2 b5 a2 b3 a5 b4 a7 c5 c7 b7 c6 d6 d7
b2 e3 g2 g1 a6 b3 c3 d2 f2 f1 e1 e2 c2 b5 d4 c4 d3 d5 c5 b7 a7 b6 a5 b4 e4 e5 c6 c7 d6 d7
b3 b1 c4 c3 b6 f1 b4 c2 b2 c1 d2 d3 e3 d4 e4 e2 g1 d5 e5 f2 g2 f3 g3 f4 g4
b4 f1 c2 b7 a7 c3 b3 b5 c4 c5 e4 d4 e3 e5 d5 c7 b6 c6 d6 d7 e6 e7 g6 f6 g5 f5 g4 f4 g3
b5 f1 c3 b6 c6 d2 c5 c1 a2 b4 c4 b2 a3 b3 c2 d1 e2 d3 e3 d4 e4 e1 g1 d5 e5 f2 g2 f3 g3 f4 g4
b6 c4 f2 d5 d4 c5 e5 e4 g3 f3 g2 f4 g4
b7 b6 e4 d5 e5 e3 g2 f3 g3 e6 f5 f6 c6 c5 d6 f4 d4 d3 g4 g1 c4 c3 f2 f1 e2 e1 d2 d1
c1 c4 a6 f2 d3 b6 e4 d4 e3 e5 d5 e1 d2 c2 a3 d1 b4 a4 b3 b1 e2 b2 f1 c7 b7 c6 a7 b5 a5 c3 d6 d7
c2 d5 c5 b7 a7 b6 a6 b5 a5 b3 c4 f1 b4 c3 e2 d3 e3 d4 e5 e4 g3 f3 g2 g1 f2 d2 e1 f4 g4
c3 d4 b6 b5 c5 c1 c4 d2 a2 b2 a3 b3 a4 b4 c2 d1 e2 d3 e4 d5 e5 e3 g2 f2 g1 f3 g3 f4 g4
c4 c3 b6 d3 e3 f1 d2 e2 b3 b5 b4 c5 d4 d5 e5 e4 g3 f3 g2 g1 f2 b2 c2 f4 g4
c5 d5 c2 b7 a7 b6 a6 b5 a5 b3 c4 f1 b4 c3 e2 d3 e3 d4 e5 e4 g3 f3 g2 g1 f2 d2 e1 f4 g4
c6 d3 c3 c4 e3 d5 c5 d4 e5 e4 a5 b4 a4 b3 a3 b1 b2 b5 a6 a7 b6 c1 c2 d1 d2 e1 e2 f1 g1 f2 g2 f3 g3 f4 g4
c7 b6 e4 d5 e5 e3 g2 f3 g3 e6 f5 f6 c6 c5 d6 f4 d4 d3 g4 g1 c4 c3 f2 f1 e2 e1 d2 d1
d1 b6 c6 d4 e4 d5 e5 f2 e3 e2
d2 c5 e4 d3 f2 e5 d5 f3 e3 c7 b6 c6 b5 c3 c4 d4 a4 b2 b3 c2 d6 d7
d3 d4 b6 c4 a5 a6 b5 b3 c3 b4 e4 d5 e5 e3 g2 f3 g3 g1 f2 f1 e2 e1 c2 f4 g4
d4 d3 c4 c3 e3 f1 c2 d2 e1 e2 g1 f2 f3 g2 a4 b3 a3 b1 b2 b4 a5
d5 d4 f2 e4 g3 f3 g2 f5 e5 f4 c4 c5 a6 a7 b6 b7 c6 d3 c3 b5 a5 c7 e6 b4 a4
d6 e3 c4 d5 b6 c3 d3 b5 c5 e1 f2 e2 f3 e5 d4 e4 g4 f6 e6 f5 d2 d1
d7 b6 e4 d5 e5 e3 g2 f3 g3 e6 f5 f6 c6 c5 d6 f4 d4 d3 g4 g1 c4 c3 f2 f1 e2 e1 d2 d1
e1 b6 c6 d4 e4 d5 e5 f2 e3 e2
e2 d5 e5 e4 c5 d3 e3 d4 c3 c4 g3 f4 g4 f3 g2 f2 g1 f5 g5 f7 f6 e7 e6 d7 c6 d6 a5 b5 a6 a7 b6 b4 a4
e3 d5 b6 c4 c5 d4 b3 b4 e5 e4 g3 f2 f3 e2 d3 d2 c3 c1 a2 a3 b2 f4 g4
e4 e3 d4 d3 b3 b4 c3 c4 g2 g1 f2 f1 e2 e1 d2 b2 c2 d7 c6 f3 g3 f4 g4
e5 d5 e4 e3 d4 d3 b3 c2 b2 c4 a5 a4 b4 b6 c5 b5 g2 g1 f2 f1 e2 e1 d2 c3 d1 f3 g3
e6 f2 e3 d4 e2 e4 f3 f4 c4 c5 a6 a7 b6 b7 c6 d3 c3 b5 a5 c7 d6 b4 a4
e7 d3 a5 b3 e3 d4 e4 f1 e2 e1 d2 d1 c2 c1 a2 b2 g1 f2 g2 f3 g3 f4 g4 f6 f5 e6 e5 d6 c5 b4
f1 b6 e3 e5 d5 c7 b7 c6 d6 d7
f2 e4 b6 d3 d4 e3 c3 c4 a5 a6 b5 b4 a4
f3 d4 e5 f4 e4 f2 e2 e3 c4 c5 a6 a7 b6 b7 c6 d3 c3 b5 a5 c7 e6 d5 d6 b4 a4
f4 c5 f5 d3 e3 f1 g1 f3 d4 e4 c4 d5 a6 a7 b6 b5 a5 b7 c6 c7 d6 d7 f6 e5 e6 b4 a4
f5 d4 f2 f3 e3 e4 c4 c5 a6 a7 b6 b7 c6 d3 c3 b5 a5 c7 d6 d7 f6 e5 e6 b4 a4
f6 c5 a6 a7 g2 f5 e5 d6 b6 b7 c6 d5 c4 c7 e6 f3 e4 e3 d4 c3 d3 e1 e2 f1 g1 f2 f4 g3 d2 d1
f7 d3 a5 b3 e3 d4 e4 f1 e2 e1 d2 d1 c2 c1 a2 b2 g1 f2 g2 f3 g3 f4 g4 f6 f5 e6 e5 d6 c5 b4
g1 b7 e5 e4 f3 d6 g4 f2 e3 f4 g3 e7 g6 g5 f5 f6 d7 e6 c6 d5 c5 c7 a7 g2 e2 d4 c4 b6 a6 d3 c3 b5 a5 b4 a4
g2 b7 e5 f3 e2 d3 e3 e4 d4 d5 a7 c5 g3 f4 g4 f2 g1 f5 g5 f7 f6 e7 e6 d7 d6 c7 b6 c6 a5 b4 a4 b5 a6
g3 f2 e3 c5 d4 c2 a2 b3 e2 d5 e5 e4 f3 f4 g4 f5 g5 f7 f6 e7 e6 d7 d6 c7
g4 e7 c5 d3 c3 b6 c4 c7 c6 c1 b7 d2 a2 b2 a3 b3 a4 b4 a5 b5 c2 d1 e3 d4 e4 d5 e5 f1 e1 e2 g1 f2 g2 f3 g3 d6 e6
g5 d4 c4 d3 e3 e4 c3 c5 a6 b3 b5 a7 b6 b7 c6 c7 d6 d7 e6 e7 g6 f5 f6 d1 c2 c1 a2 b2 d2 e1
g6 c5 g2 f5 c4 c3 e5 f3 f4 g3 b4 d3 a3 d4 a6 a5 b5 a7 b6 b1 b2 a4 b3 b7 c6 c1 c2 d1
g7 d3 c3 c5 c4 e4 d5 d4 a6 a7 b6 b5 a5 b3 b4 b7 c6 c7 d6 d7 e6 e7 f6 f7 g6 d1 c2 c1 a2 b2 d2 e1
```

Fig. 15. Longest 7×7 SOLVER lines of play. For each of the 49 7×7 1-openings, the corresponding line shows a longest line of play from the the associated SOLVER solution. The top row shows the move number of that column.

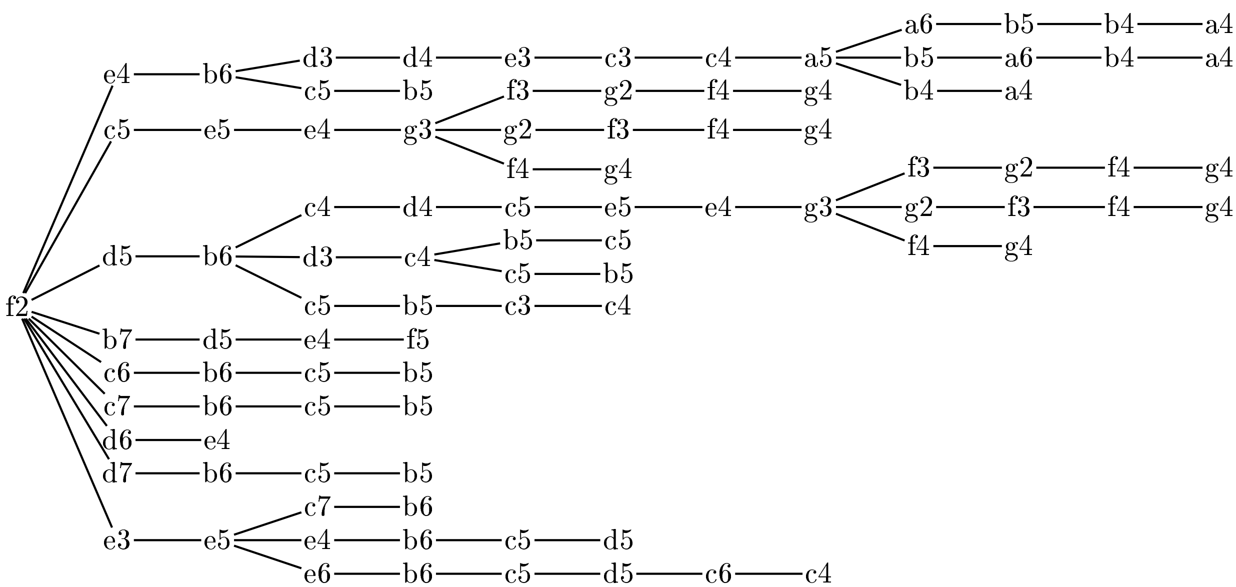


Fig. 16. A Solver proof tree for the 7×7 opening White-f2. The data for this tree was generated by first running a modified version of Solver in which the final move considered from any position is a winning move; this guarantees that every recursion tree path ends with a winning move. The resulting recursion tree was then pruned, at all points in the tree, by removing any losing moves made by the winning player before the winning move for that state was found.