Advances in Computer Games, Springer Lecture Notes in Computer Science, Vol. 6048, H.J. van den Herik and P. Spronck (eds), 2010, 88-98

Hex, Braids, the Crossing Rule, and XH-search

Philip Henderson, Broderick Arneson, and Ryan B. Hayward

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

Abstract. We present XH-search, a HEX connection finding algorithm. XH-search extends Anshelevich's H-search by incorporating a new crossing rule to find braids, connections built from overlapping subconnections.

1 Introduction

HEX is the connection game invented by Piet Hein [1] and John Nash [2]. The board is a rhombus of hexagonal cells. On alternating turns, each player places a stone of her colour on any vacant cell. The winner is the player who connects her two opposing edges with a path of her stones. HEX has many nice properties: additional stones of a player's colour are never disadvantageous, the game cannot end in a draw, and the first player has a winning strategy [2]. However, determining the winner of arbitrary positions is PSPACE-complete [3].

For a HEX position, a *point* is either a vacant cell or a maximal connected set of same-coloured stones; the latter is a *chain*. We assume that board edges are represented by stones, so a chain can include a board edge. In HEX, a common tactical question is whether a specified pair of points can be connected. For a HEX position, a subgame with a second-player strategy (respectively first-player strategy) to connect two points is a *virtual connection* or VC (resp. *virtual semi connection* or SC). For a VC/SC, the two points connected are its *endpoints*, while the set of vacant cells used in the connecting strategy is its *carrier*. For an SC, the initial move of the strategy is its *key*. See Fig. 1.

Anshelevich presented H-search [4, 5], a hierarchical VC/SC composition algorithm. H-search is the foundation of HEXY, SIX, and WOLVE, the only gold medal HEX programs of the Computer Games Olympiad [6–10]. H-search uses three rules, respectively *base/AND/OR*:
(1) Each player has an empty carrier VC between each pair of adjacent points.
(2) If a player has VCs α_1, α_2 with endpoint pairs $\{p_0, p_1\}$,$\{p_0, p_2\}$ and carriers C_1, C_2 such that $C_1 \cup \{p_1\}$ and $C_2 \cup \{p_2\}$ do not intersect, then

(i) if the midpoint p_0 is vacant, then combining α_1, α_2 forms an SC with endpoints p_1, p_2, carrier $C_1 \cup C_2 \cup \{p_0\}$, and key p_0,

(ii) if the midpoint p_0 is a chain for the player, then combining α_1, α_2 forms a VC with endpoints p_1, p_2 and carrier $C_1 \cup C_2$.

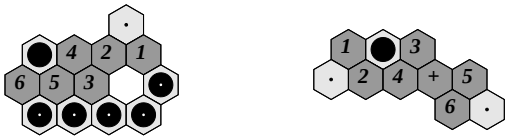

Fig. 1. Diagrams of a Black VC (left) and a Black SC (right). Carriers are shaded, endpoints are dotted, and the SC key is +. In Black's VC strategy, which connects the top dotted cell to the bottom dotted group, if White plays 1 then Black plays 2; then, if White plays 3 then Black plays 4 and then one of {5,6}. In Black's SC strategy, which connects the dotted cells, after playing at the key, Black plays to get one of each of {1,2}, {3,4}, {5,6}.

(3) If between endpoints p_1, p_2 a player has SCs $\alpha_1, \ldots, \alpha_k$ with carriers C_1, $\ldots, C_k$ such that $C_1 \cap \ldots \cap C_k$ is empty, then combining the SCs forms a VC with endpoints p_1, p_2 and carrier $C_1 \cup \ldots \cup C_k$.

When used in automated HEX players, H-search is usually restricted by limiting the number of SCs used in the OR rule, for otherwise it takes too long. However, as Anshelevich observed, even unrestricted H-search misses some connections [4, 5], including the SC shown in Fig. 2. We call this SC the *braid*, as its substrategies are tangled together. It is of interest to extend H-search in a way that allows for efficient discovery of new connections.

Melis extended Anshelevich's implementation of H-search by allowing board edges as AND rule midpoints [11, 9]. Rasmussen et al. extended H-search by triggering a VC search if the OR rule finds an SC set with small carrier intersection [12]; this finds more connections but the search time grows exponentially in the number of cells in the search. Yang described a decomposition notation for HEX connections, including his hand-crafted centre-opening 9×9 SC [13, 14], and Noshita introduced union-connections, which are useful in connection verification but not found by H-search [15, 16]; neither of these techniques has been used in an automated connection discovery algorithm.

In this paper we present XH-search, an easily implemented extension of H-search. XH-search finds all connections found by H-search, as well as connections

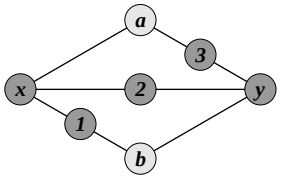

Fig. 2. The braid, an SC not found by H-search. Endpoints are a, b; the key is x or y. To connect, say after playing at key x, if the opponent blocks at 1 then respond at y and then claim one of $\{2, 3\}$. Notice that within the braid there are VCs between each of $\{a, x\}$, $\{b, y\}$, and SCs between each of $\{x, y\}$, $\{x, b\}$, $\{a, y\}$.

built up from braids. The Crossing Rule, a new HEX connection deduction rule, allows for an efficient implementation of XH-search.

2 Stepping Stones

In order to describe the Crossing Rule we first need to define stepping stones. Although defined as points that are internal to a connection, we will use them later as braid endpoints.

Stepping Stone. For a chain X and VC or SC C, X is a *stepping stone* of C if following C's strategy guarantees that C's endpoints will be connected by a chain containing X.

For a VC/SC C, $SS(C)$ denotes the set of C's stepping stones. See Fig. 3.

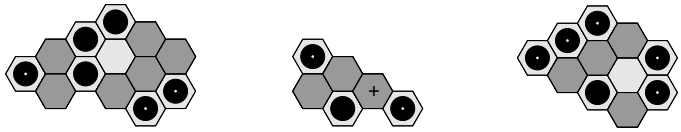

Fig. 3. A VC (left) and SC (middle) with stepping stone. A VC (right) with none.

Lemma 1. *Let C be a base rule VC. Then C has no stepping stone: $SS(C) = \emptyset$.*

Proof. In a base rule VC the two endpoints are neighbours and so already connected. □

Lemma 2. *Let C be a VC computed via the OR rule from SCs $C_1, \ldots, C_k$. Then each chain which is a stepping stone for every C_j is a stepping stone of C: $\cap_{j=1}^{k} SS(C_j) \subseteq SS(C)$.*

Proof. If a chain X is a stepping stone for every SC C_j then, regardless of which SC is maintained, there will be a chain connecting C's endpoints, equal to C_j's endpoints, that contains X. □

Lemma 3. *Let C be an SC with key k computed via the AND rule from VCs C_1, C_2 with vacant midpoint k. Then any chain which is a stepping stone of C_1 or C_2 is a stepping stone of C: $SS(C_1) \cup SS(C_2) \subseteq SS(C)$.*

Proof. The strategy that maintains C first plays at k and then maintains both C_1 and C_2. C_1 connects k to one endpoint, say p_1, of C and C_2 connects k to the other endpoint, say p_2. Each chain created by C also follows C_1 and so connects p_1 to k and contains each stepping stone s_1 of C_1; similarly, it follows C_2 and so connects p_2 to k and contains each stepping stone s_2 of C_2. □

Notice in Lemma 3 that k is vacant, so not in a chain, so not in $SS(C)$.

Lemma 4. *Let C be a VC or SC computed via the AND rule from connections C_1, C_2 with chain midpoint X. Then $SS(C_1) \cup SS(C_2) \cup \{X\} \subseteq SS(C)$.*

Proof. Following the strategies for C_1 and C_2 ensures that the union of X with connecting chains for C_1 and C_2, each of which connects X to an endpoint of C, forms a connecting chain for C containing X. Thus X is in $SS(C)$. The inclusion of $SS(C_1)$ and $SS(C_2)$ in $SS(C)$ follows as in the proof of Lemma 3. □

For each connection C discovered in XH-search we compute a subset $SS^*(C)$ of $SS(C)$. $SS^*(C)$ is defined by applying the preceding lemmas, with this exception: for each VC C computed via the OR rule, $SS^*(C)$ is the empty set. We refer to this as the *SS algorithm*.

Lemma 5. *Let C be a connection with endpoints p_1, p_2. Then for any stepping stone s in $SS^*(C)$ there is a partition S_1, S_2 of the carrier of C such that S_1 is the carrier of a VC from s to p_1, and if C is a VC (resp. SC) then S_2 is the carrier of a VC (resp. SC) from s to p_2.*

Proof. Argue by induction. If C is a base rule VC then $SS^*(C)$ is empty and the lemma holds vacuously. Assume next that C is a VC built from connections whose stepping stones satisfy the lemma. If C is built by the OR rule, then $SS^*(C)$ is empty and again the lemma holds vacuously. Suppose then that C is built by the AND rule, say from VCs C_1, C_2 with midpoint chain p_0 and endpoint pairs $\{p_0, p_1\}, \{p_0, p_2\}$. Then $SS^*(C) = SS^*(C_1) \cup SS^*(C_2) \cup \{p_0\}$. If $s = p_0$, then partitioning C into C_1, C_2 satisfies the lemma. Assume next that s is in $SS^*(C_1)$. Then by the induction hypothesis, the carrier of C_1 can be partitioned into A, B with A a VC from s to p_1 and B a VC from s to p_0. By the AND rule, taking the union of B and C_2 with midpoint p_0 yields a VC from s to p_2 that is disjoint from A, and the lemma holds. Similarly, the same holds if s is in $SS^*(C_2)$. Thus, by induction, the lemma holds for VCs. The proof is similar for SCs. □

It is of interest to know whether Lemma 5 holds if one replaces $SS^*(C)$ with $SS(C)$. This is the case if the OR rule is restricted to combining two SCs.

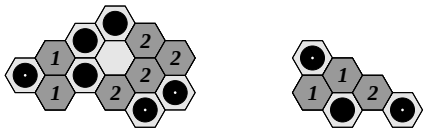

Fig. 4. Illustrating Lemma 5. The VC carrier (left) partitions into S_1, S_2, where each S_j is the carrier of a VC to the stepping stone. The SC carrier (right) partitions into S_1, S_2, where S_1 (resp. S_2) is the carrier of a VC (resp. SC) to the stepping stone.

3 The Crossing Rule

Observe in Fig. 2 that if the endpoints a, b of a braid are same-coloured chains then the braid "untangles" into three disjoint SCs between the internal vacant cells x, y such that two of these SCs have stepping stones. The following rule shows that finding two vacant cells with three such SCs is sufficient to conclude the existence of an SC between particular pairs of the SC's stepping stones.

Crossing Rule. Consider a HEX position with pairwise disjoint SCs C_1, C_2, C_3, each with vacant cell endpoints x, y, and with both $S_1 = SS^*(C_1) \setminus SS^*(C_2)$ and $S_2 = SS^*(C_2) \setminus SS^*(C_1)$ nonempty. Then for any endpoints s_1, s_2 in S_1, S_2 there is an SC C whose carrier is the union of $\{x, y\}$ with the carriers of C_1, C_2, C_3, and with key x or y.

SCs found by the Crossing Rule are shown in Fig. 5.

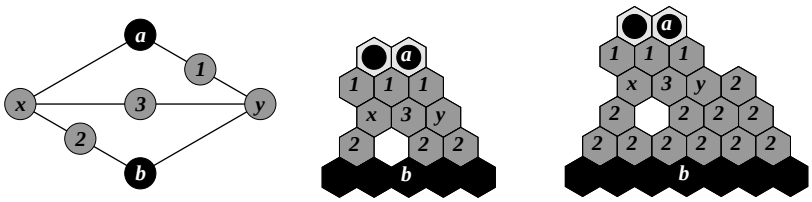

Fig. 5. Crossing Rule SCs. For $j = 1, 2, 3$, cells labelled j form carrier C_j of SC between x, y. $SS^*(C_1)$ contains a and not b. $SS^*(C_2)$ contains b and not a. Combining these SCs yields SC between a, b with carrier $\{x, y\} \cup C_1 \cup C_2 \cup C_3$ and key x or y.

Proof. By Lemma 5 we can partition C_1's carrier into VC V_1 and SC W_1, and partition C_2's carrier into VC V_2 and SC W_2. Assume first that VCs V_1, V_2 have a common endpoint: each connects their respective stepping stone endpoint s_1, s_2 to the same vacant cell, say x. Then by the AND rule, there exists an SC C^- connecting s_1 to s_2 with key x and whose carrier is the union of $\{x\}$ and the carriers of V_1, V_2. C is the same as C^-, except with an (unnecessarily) larger carrier, so C can follow the same strategy as C^- and we are done.

Assume next that V_1 and V_2 have no common endpoint. By relabelling cells if necessary, assume V_1 connects s_1 to x and V_2 connects s_2 to y. Thus W_1 connects s_1 to y and W_2 connects s_2 to x. Notice that the carriers of V_1, W_1, V_2, W_2, C_3 are pairwise disjoint by construction. The strategy to maintain SC C is as follows: play the key x as the first move, and then maintain VCs V_1, V_2 against any probes into their carriers. If the opponent's first probe outside V_1, V_2 is in W_1 or C_3, then playing the key of W_2 completes the connection from s_1 to x to s_2 via V_1, W_2. If instead the opponent's first probe is in W_2, then play y next, noting that y has a VC to s_1 via application of the OR rule to two disjoint SCs whose

carriers are W_1 and the series combination of C_3, V_1 through x. Since y also has a disjoint VC to s_2 (V_2), we are done. □

Notice that if the second case of the Crossing Rule's proof applies then either x or y can be key. Also, if the first case of the proof applies, then key selection matters but C is a connection whose carrier properly contains the carrier of a connection that can be deduced via H-search. We assume that H-search is implemented so that a connection is deleted whenever a second connection is discovered with the same endpoints but with a carrier that is a proper subset of the first connection's carrier.[1] Thus the first case is not relevant, and we can assume that any SC discovered by the Crossing Rule can have either x or y as its key. Also, the Crossing Rule deduces an SC joining any such s_1, s_2. Thus we can compute a single carrier and key, and then add the "same" SC to various different pair-connection lists. This also holds if C_3 has stepping stones. The Crossing Rule building block SCs do not share any endpoints with the deduced SC: in some sense, connections deduced by the Crossing Rule are orthogonal to their subconnections, whereas the connections deduced by the AND/OR rules are parallel to their subconnections.

The Crossing Rule can be strengthened. A cell is *dead* if it is not on any minimal path connecting either player's two edges. A set of vacant cells is *captured* by a player if she has a second player strategy on that set that leaves every opponent stone in that set dead. For example, the carrier of an *edge bridge*, shown in Fig 6, is captured. Filling in a player's captured set with her stones does not change the winner of a postion [17, 18].

Fig. 6. A black edge bridge (left). Black fill-in does not alter the winner (right).

Strong Crossing Rule. Consider a HEX position with SCs C_1, C_2, C_3, each with vacant cell endpoints x, y, and with both $S_1 = SS^*(C_1) \setminus SS^*(C_2)$ and $S_2 = SS^*(C_2) \setminus SS^*(C_1)$ nonempty. Further, assume that the player with these SCs captures a set B by playing at y, and that $C_1 \cap C_2$, $C_1 \cap C_3$, $C_2 \cap C_3$ are each a subset of B. Then for any endpoints s_1, s_2 in S_1, S_2 there is an SC C whose carrier is the union of $\{x, y\}$ and B with the carriers of C_1, C_2, C_3, and with key y.

Proof. (sketch) The result follows from the Crossing Rule and the fact that filling in captured sets does not change the winner of a game. □

[1] Such connections are provably useless and are pruned by HEX programs such as SIX and WOLVE [11].

An SC found by the Strong Crossing Rule is shown in Fig. 7. XH-search applies the Strong Crossing Rule by checking whether either of the potential keys x or y forms a bridge with the edge; the carrier of the edge bridge is the captured set B. When XH-search finds such an SC C, it updates $SS^*(C)$ by applying the next lemma.

Lemma 6. *Let C be an SC computed via the (Strong) Crossing Rule on SCs C_1, C_2, C_3. Then $SS(C)$ is empty.*

Proof. Recalling the proof of the Crossing Rule, there is no common substrategy among the potential outcomes: any portion of C_1, C_2, C_3 could be omitted from a winning path. Indeed, even the intersection of any of these SCs cannot be valid as their strategies were partitioned into disjoint carriers. Thus, in general we cannot conclude the existence of any stepping stones for an SC deduced from the (Strong) Crossing Rule. □

Using this lemma in the SS algorithm does not change the validity of Lemma 5, which holds vacuously for any empty set of stepping stones. Thus the (Strong) Crossing Rule still holds.

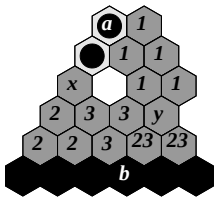

Fig. 7. An SC found by the Strong Crossing Rule. For $j = 1, 2, 3$, cells labelled j form carrier C_j of SC between x, y. Cells labelled both 2,3 form set B and are captured if Black plays y. $SS^*(C_1)$ contains a and not b. $SS^*(C_2)$ contains b and not a. Combining these SCs yields an SC between a, b with carrier $\{x, y\} \cup B \cup C_1 \cup C_2 \cup C_3$ and key y.

4 Crossing Rule Connections

We now show some Hex connections found by XH-search but not H-search. Our implementation of XH-search computes stepping stones via the SS algorithm, with each connection storing all of their stepping stones, and applies the Strong Crossing Rule with edge bridges as the only captured sets considered. The three VCs in Fig. 8 are common edge VCs, also known as templates (see King's web-page for more templates [19]). In each of these examples H-search fails to find an SC which does not use the marked cell. These respective three "missing SCs", found by XH-search, are those shown in Figs. 5 and 7.

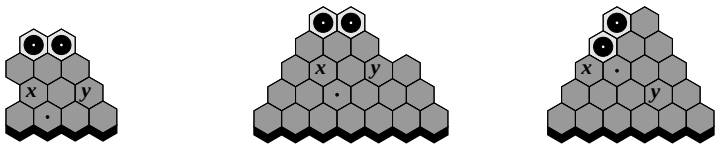

Fig. 8. Edge VCs found by XH-search but not H-search. Each SC found by H-search uses the marked cell. The "missing SCs", which do not use this cell, are in Fig. 5 or 7.

If the Crossing Rule could only find common edge VCs, then adding a library of VC patterns to check would be an effective alternative. Thus, in Fig. 9 we show more connections from games played by our HEX programs in which XH-search proved advantageous.

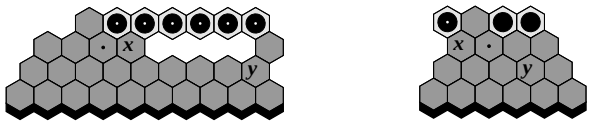

Fig. 9. More edge VCs found by XH-search but not H-search. Each SC found by H-search uses the marked cell. The two "missing SCs" are each similar to that of Fig. 7.

The Crossing Rule requires both endpoints of the deduced SC to be chains, and most connections found by XH-search but not H-search are near an edge. Thus, when using XH-search we ensure that the AND Rule allows edges as midpoint; this allows edges to be stepping stones.

XH-search is not complete: there are connections, some easily recognizable by humans, that it cannot find. See Fig. 10. It is of interest to find some efficient connection recognition algorithm which can find all connections recognizable by human players.

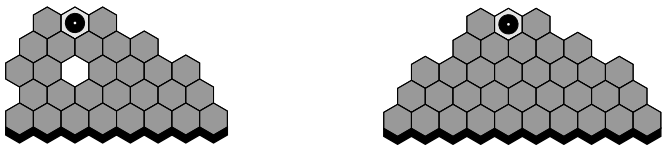

Fig. 10. XH-search finds neither the SC (left) nor the VC (right).

5 Implementation and Complexity

It is straightforward to implement XH-search by starting with H-search, adding stepping stone deductions to the AND/OR rules, and adding the (Strong) Crossing Rule. In the following pseudocode, the queue holds endpoint pairs, each of which has a VC carrier list and an SC carrier list; omitting the crossing rule computation leaves H-search.

```
Algorithm XH-search.
   initialize VC/SC carrier lists:
       for each pair E of endpoints
           E.VCList.makeEmpty(); E.SCList.makeEmpty();
   initialize queue Q with base VCs:
       Q.makeEmpty()
       for each pair E of adjacent endpoints
            Q.add(E); E.VCList.add(baseVC(E))
   while (not Q.isEmpty())
       E <- Q.remove()
       compute crossing rule on E's SCs:
           for each new SC Z with endpoint pair F found,
               Q.add(F); F.SCList.add(Z)
       compute OR rule on E's SCs:
           for each new VC Z found
               E.VCList.add(Z)
       compute AND rule on E's VCs with both of E's endpoints:
           for each new VC/SC Z with endpoint pair G found,
               Q.add(G); G.(VC/SC)List.add(Z)
   end while
end XH-search
```

Extending H-search by adding stepping stone deductions does not increase the runtime complexity:

Lemma 7. *Let $f(n)$ be the worst-case running time of H-search on a board with n cells. Then the worst-case running time of H-search with stepping stone deductions is in $O(f(n))$.* □

Regarding memory requirements, our H-search implementation stores the two endpoints and carrier for each connection. In order to store stepping stones, two options are possible: store all stepping stones in the same manner as the carrier, or store only a single stepping stone per connection. The second solution is appealing, since a complete set of stepping stones is not required in order to find new connections, and since many connections have few stepping stones; however, it forces us to choose which stepping stone to keep if there are several available. The first solution provides more information, but nearly doubles the memory per connection; the second increases the memory per connection only marginally. Since memory is not a bottleneck for our HEX program, we opted for the first solution.

The running time of XH-search depends not only on the number of points and the sizes of connection lists, but also on the number of discovered connections as well as their type (VC or SC). Nonetheless, we can at least analyze the relative computational efficiency of the different deduction rules in terms of the known factors.

Aside from the AND rule, OR rule, and Crossing Rule, we also include naive deduction of missing SCs of the form shown in Fig. 2. This naive deduction would involve finding four distinct points a, b, x, y such that x, y are vacant, and such that there exists five pairwise disjoint connections with two VCs joining pairs a, x and b, y and three SCs joining pairs a, y and b, x and x, y. Since the OR rule's complexity is parameterized by the maximum number of SCs it may combine in parallel, we have included entries for parameter values of both three and four, which are the common selections for HEX programs. Table 1 summarizes our analysis, with n representing the number of points and l_V, l_S representing the list sizes for VCs and SCs, respectively. We distinguish between these two list lengths, as the number of SCs usually far exceeds the number of VCs.

Deduction rule	running time
AND rule	$n^3 l_V^2$
OR-3 rule	$n^2 l_S^3$
OR-4 rule	$n^2 l_S^4$
crossing rule	$n^2 l_S^3$
naive rule	$n^4 l_V^2 l_S^3$

Table 1. Worst-case runtime (within constant factor) of connection deducing rules.

We can now clearly see the benefit of using stepping stones: the complexity of the Crossing Rule is roughly the square root of a naive implementation, and roughly the same order of complexity as the regular deduction rules in H-search.

6 Experiments

To test the effectiveness of the crossing rule, we played an 11×11 tournament between Monte Carlo players H (using H-search) and XH (using XH-search). Each player is a version of MOHEX, the UCT HEX player that won silver at the 2008 Computer Games Olympiad in Beijing [7]. Each player uses 100K rollouts to analyze UCT tree nodes and computes connections with our normal tournament settings: the AND rule is computed over the edge (so the edge of the board can be the midpoint of an AND connection) and the OR-rule combines up to 4 SCs. Neither player uses an opening book or endgame solver. The tournament comprises 4-rounds, with each player opening at each position once as Black and once as White (with no swap move allowed), for a total of 4×121×2=968 games.

XH defeated H 501 games to 467, 17 games above a breakeven score of 484, taking on average 1.099 times as long as H per move. The time increase

is roughly in line with the analysis in Table 1: an 11×11 board has roughly $n = 100, l_V = 10, l_S = 25$, and finding more connections with the Crossing Rule increases the number of iterations for all rules.

The crossing rule thus added roughly 12 ELO in strength in exchange for 9.9% more computation time on average. While this gain might seem negligible, strength gains measured via all-opening no-swap tournaments are muted due to the forced inclusion of many very strong and very weak opening moves. By comparison, each doubling of MoHex's number of rollouts results in an average ELO gain of 31.8 (averaged over 7 rollout doublings, from 1s to 128s, when competing against a 1s version). Thus, per unit time invested, the crossing rule represents a more significant strength gain than simply increasing the number of rollouts.

7 Conclusions

XH-search is efficient and easily implemented. Furthermore, it identifies important Hex connections that cannot be found with H-search.

In future work we hope to identify further efficient deduction rules, particularly those that identify the most common connection omissions. The captured sets of the Strong Crossing Rule need not be restricted to edge bridges; it would be interesting to find other efficient methods that allow for carrier overlap within connection deduction rules.

Acknowledgements

We gratefully acknowledge NSERC and iCORE for funding this work, and the University of Alberta's Hex and GAMES group members for providing support and feedback.

References

1. Hein, P.: Vil de laere Polygon? Series of articles in *Politiken* newspaper (December 1942)
2. Nash, J.: Some games and machines for playing them. Technical Report D-1164, RAND (1952)
3. Reisch, S.: Hex ist PSPACE-vollständig. Acta Informatica **15** (1981) 167–191
4. Anshelevich, V.V.: The game of Hex: An automatic theorem proving approach to game programming. In: AAAI/IAAI. (2000) 189–194
5. Anshelevich, V.V.: A hierarchical approach to computer Hex. Artificial Intelligence **134**(1–2) (2002) 101–120
6. Anshelevich, V.V.: Hexy wins Hex tournament. ICGA Journal **23**(3) (2000) 181–184
7. Arneson, B., Henderson, P., Hayward, R.B.: Wolve Wins Hex Tournament. ICGA **31**(4) (2008)
8. Hayward, R.B.: Six wins Hex tournament. ICGA Journal **29**(3) (2006) 163–165

9. Melis, G., Hayward, R.: Six wins Hex tournament. ICGA Journal **26**(4) (2003) 277–280
10. Willemson, J., Björnsson, Y.: Six wins Hex tournament. ICGA Journal **27**(3) (2004) 180
11. Melis, G.: Six. `http://six.retes.hu/` (2006)
12. Rasmussen, R., Maire, F.: An extension of the H-search algorithm for artificial Hex players. In: Australian Conference on Artificial Intelligence. (2004) 646–657
13. Yang, J.: Jing Yang's web site. `www.ee.umanitoba.ca/~jingyang/` (2003)
14. Yang, J., Liao, S., Pawlak, M.: A decomposition method for finding solution in game Hex 7x7. In: ADCOG. (2001) 96–111
15. Kohei, N.: Union-connections and a simple readable winning way in 7×7 Hex. Proceedings of 11th Game Programming Workshop (2004) 72–79
16. Kohei, N.: Union-connections and straightforward winning strategies in Hex. ICGA Journal **28**(1) (2005) 3–12
17. Hayward, R.B.: A note on domination in Hex. Technical report, University of Alberta (2003)
18. 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. Volume 263 of International Federation for Information Processing. Kluwer Academic Publishers, Boston (2003) 261–278
19. King, D.: Hall of hexagons - the game of Hex. `http://www.drking.plus.com/hexagons/hex/index.html` (2007)