MoHex 2.0: A Pattern-Based MCTS Hex Player

Shih-Chieh Huang[1,2], Broderick Arneson[2], Ryan B. Hayward[2(✉)], Martin Müller[2], and Jakub Pawlewicz[3]

[1] DeepMind Technologies, London, UK
[2] Computing Science, University of Alberta, Edmonton, Canada
hayward@ualberta.ca
[3] Institute of Informatics, University of Warsaw, Warsaw, Poland

Abstract. In recent years the Monte Carlo tree search revolution has spread from computer Go to many areas, including computer Hex. MCTS-based Hex players now outperform traditional knowledge-based alpha-beta search players, and the reigning Computer Olympiad Hex gold medallist is the MCTS player MoHex. In this paper we show how to strengthen MoHex, and observe that—as in computer Go—using learned patterns in priors and replacing a hand-crafted simulation policy by a softmax policy that uses learned patterns significantly increases playing strength. The result is MoHex 2.0, about 250 Elo points stronger than MoHex on the 11×11 board, and 300 Elo points stronger on the 13×13 board.

1 Introduction

In the 1940s Piet Hein [22] and independently John Nash [26–28] invented Hex, the classic two-player alternate-turn connection game. The game is easy to implement — in the 1950 s Claude Shannon and E.F. Moore built an analogue Hex player based on electrical circuits [29] — but difficult to master. It has often been used as a testbed for artificial intelligence research.

Around 2006 Monte Carlo tree search appeared in Go [11] and soon spread to other domains. The four newest Olympiad Hex competitors — MoHex from 2008 [4], Yopt from 2009 [3], MIMHex from 2010 [5], Panoramex from 2011 [20] — all use MCTS.

In this paper we show how to strengthen MoHex, the reigning Computer Olympiad Hex gold medallist [20]. Among several improvements, we observe that — in Hex as in Go — replacing a hand-crafted MCTS simulation policy by one based on learned board patterns can increase a player's strength. Following Go players such as Erica, we apply a minorization-maximization algorithm [12] to measure the win correlation of board patterns, and use the resulting pattern weights in both the leaf selection and simulation phases of MCTS. The result is MoHex 2.0, a player that is about 250 Elo[1] points stronger than MoHex on the 11×11 board, and 300 Elo points stronger on the 13×13 board.

[1] The Elo gain from win rate r is $400 * -\log((1/r) - 1)$.

H.J. van den Herik et al. (Eds.): CG 2013, LNCS 8427, pp. 60–71, 2014.
DOI: 10.1007/978-3-319-09165-5_6,

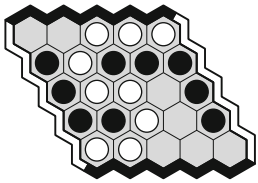

Fig. 1. A Hex game won by White.

In Sect. 2 we review computer Hex. In Sect. 3 we review MoHex. In Sect. 4 we describe MoHex 2.0. In Sect. 5 we give experimental results. In Sect. 6 we discuss future work.

2 Computer Hex

We begin with the rules of Hex. One player has black stones and is assigned two opposite sides, or borders, of the board. The other player has white stones and is assigned the other two sides. Players alternate placing a single stone of their color on an empty cell. The winner is the player who joins their two sides with their stones. Black plays first. To mitigate the first-player advantage, this extra rule is adopted: the first player places a black stone, and the second player then chooses whether to play as white or as black. See Fig. 1. For more on Hex, see [8,9,21].

In the 1950s, as part of his seminal research into games and artificial intelligence, Claude Shannon (with help from E.F. Moore) built electrical circuit machines to play the connection games Birdcage [17] and Hex [29]. Birdcage — also known as Gale or Bridg-It — is similar to Hex, except it is played on the edges of a graph rather than on the nodes [9].

In Shannon's Birdcage circuit, cells have resistors (respectively shorted or removed if occupied by the player or opponent), voltage is applied across the board, and the cell with largest voltage drop is selected as the next move.

This model can also be used for Hex, and cross-board conductance is a good indication of position strength. Following Anshelevich [2], recent alpha-beta search Hex players such as Hexy [1], Six [19], and Wolve [4] use evaluations that are based on an augmented circuit that adds information about virtual connections.

A *virtual connection*, or VC, (respectively *virtual semi-connection*, or VSC) is a second-player (first-player) point-to-point connection strategy, where a point is an empty cell, or a *chain* (as in Go, namely a maximal group of stones connected by adjacency), or a board side. Thus a VC between a player's two sides is a winning strategy. Many Hex players find a restricted class of VCs defined by an algebra described by Anshelevich [2]: initialize a set of base VCs, then find larger VCs by applying the closure of two combining rules. The *and-rule* combines VCs in series to form a new VC or VSC. The *or-rule* combines sets of collectively non-interfering VSCs in parallel to form a new VC. See Fig. 2.

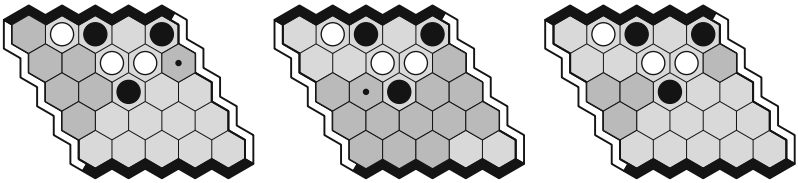

Fig. 2. The left and middle diagrams each show the cells of a side-to-side White VSC. The right diagram shows Black's corresponding mustplay region: a Black move outside this region leaves White with a winning VSC.

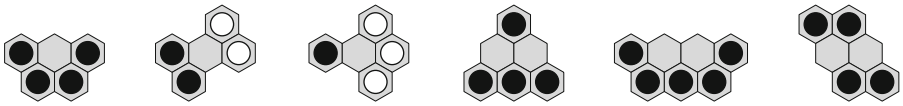

Fig. 3. Some inferior cell patterns. In the first three (from left) patterns, the empty cell is dead and so can be filled for either player. In the last three patterns, the empty cell set is Black-captured and so can be filled for Black.

In addition to strengthening the augmented circuit of which the conductance is the evaluation function, VCs and VSCs are useful in move pruning: if an opponent has a side-to-side VSC, then a player move that fails to interfere with this VSC is losing. With respect to the set of (known) opponent side-to-side VSCs, a player's *mustplay* region is the set of cells that interferes with all of these VSCs. See Fig. 2.

A second form of move pruning is based on inferior moves, especially those deducible from board patterns. A cell is *dead* if it is provably (without changing the win/loss value of the game) never needed by either player. A cell set is *captured* if it can be provably assigned to the capturing player (again, without changing the win/loss value of the game). See Fig. 3. Some Hex players have inferior cell engines, which identify these and other kinds of inferior moves. See [7,23] for more on inferior cell analysis.

3 MoHex

MoHex is a relatively simple first-generation MCTS player [6]. It is built on top of Fuego, an open-source game-independent MCTS platform especially well known for its Go player [15]. MoHex uses UCT (MCTS plus UCB as in [30]) and the RAVE all-moves-as-first UCT heuristic [18]. When a node becomes heavy (i.e., the number of node visits reaches a threshold), it runs its virtual connection and inferior cell engines on that position, yielding information that often prunes inferior moves. In simulations it uses uniformly random moves augmented by only one (up to symmetry) Hex-specific pattern, *savebridge*: if an opponent move attacks a player's bridge (a virtual connection with two empty cells), the player deterministically replies to save the bridge; if the opponent move simultaneously threatens more than one bridge, the player randomly saves one bridge. See Fig. 4.

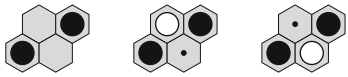

Fig. 4. The bridge (left). In a MoHex simulation, when the opponent (White) threatens a bridge, the player (Black) replies to save it (middle, right).

MoHex is 180 Elo points weaker without RAVE, and a further 100 Elo points weaker without savebridge. For comparison, doubling the number of simulations strengthens MoHex by 36 Elo points. See [6] for further details.

3.1 MoHex Weaknesses

Although MoHex has won three of the past four Olympiad Hex competitions [3,5,6,20], its success is arguably due more to the strength of its virtual connection engine than to its MCTS engine. For example, in recent Olympiad competitions the MCTS players Panoramex, MIMHex, and Yopt each played into winning positions against MoHex only to play eventually outside of the must-play region and then lose the game (whenever an opponent misses the mustplay, MoHex plays out the win without further use of MCTS).

MoHex is rather weak without its virtual connection and inferior cell engines. This is presumably due to its simulation policy, which is oblivious to global virtual connectivity. In Hex, as in Go, a position can decompose into separate subgames. Consider a position with two subgames in which Black must win both to win the game. If Black's winning probability is .6 for each subgame, then Black's game-winning probability is $\min\{.6, .6\} = .6$, whereas the simulation win rate will be closer to $.6 \times .6 = .36$. Such mismeasurement is a serious flaw, especially against an opponent capable of reasoning about positions with multiple subgames.

Figures 5 and 6 show examples of weak MoHex performance.

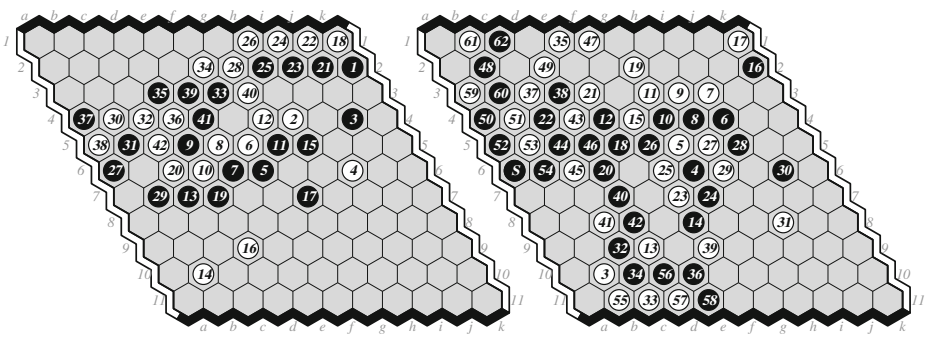

Fig. 5. MoHex (Black) loses to MIMHex (left) and Wolve (right) in 2010.

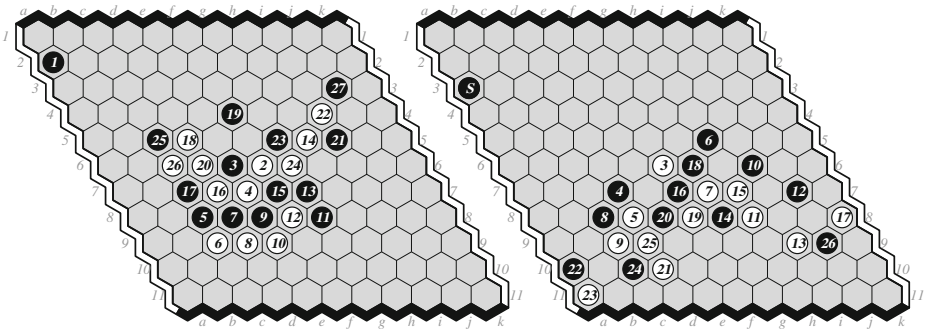

Fig. 6. MoHex (left) and Wolve (right) losing to Panoramex (White) in the 2011 Olympiad. The latter position is hard for MCTS players: `27.W[j8]` connects `13.W[i9]` directly to the side but loses; `27.W[i10]` connects indirectly to the side and wins.

4 MoHex 2.0

Given the success of patterns in UCT Go players, we wanted to revise MoHex by also using patterns. This required substantial revision, so we took the opportunity to add other updates as well. One update is to the virtual connection engine: we now use an algorithm by Jakub Pawlewicz that finds smaller connection sets, but in about half the time. A second update is to tune parameters using the optimization tool CLOP by Remi Coulom [14]. Other changes concern the MCTS engine, as follows.

4.1 Extend on Unstable Search

MCTS is *unstable* if the move with the highest win rate differs from the move with the highest visit count. MoHex moves are frequently unstable, so we implement a search extension policy similar to that of [25]: extend an unstable search by half of its original search time. We extend a search at most once.

4.2 Improved MCTS Formula

In Fuego, the score for a move j is computed using the formula

$$Score(j) = (1 - w) \times U + w \times R + E, \tag{1}$$

where U is the UCT mean (wins over visits), R is the RAVE mean (wins over visits), w is the computed weight of the RAVE term, and E is the exploration term. E is the usual UCT exploration formula, i.e., $E = c_b \times \sqrt{\frac{\ln n}{n_j}}$, where n is the parent visit count, n_j is the node visit count, and c_b is a constant.

The RAVE weight w is calculated taking into account the ratio of RAVE visits to UCT visits [18]. For details on Fuego's RAVE implementation, see [15]. When n_j is small, w is nearly 1, and so $Score(j)$ is almost exactly the

RAVE score. As n_j increases, w decays to 0 and the RAVE score becomes less important.

Notice that the exploration term is separate and outside of the RAVE and UCT terms. Thus large exploration terms (as in the case of a rarely visited child of a popular parent node) can overwhelm the RAVE information, possibly forcing exploration where it is not needed. Indeed, Fuego and MoHex both find the best setting of c_b to be 0, so such unneeded exploration never occurs. But a negative side effect of this setting is to rely solely on RAVE for exploration.

As with other recent MCTS players, we find that moving the exploration term inside the UCT term improves performance:

$$Score(j) = (1 - w) \times (U + E) + w \times R. \tag{2}$$

Formula (2) allows exploration (a) to rely on RAVE initially, but (b) to occur for already visited children, which could not happen with formula (1) where $c_b = 0$.

4.3 Patterns

Following Go players such as Erica, we apply the supervised learning algorithm minorization-maximization (MM) [12] to learn the win correlation of board patterns, and use the resulting pattern weights in both the leaf selection and simulation phases of MCTS.

For learning we used two data sets of about 35 000 games. One set consists of 19 760 13×13 games among strong human players on the Little Golem site, extracted in July 2012. The other set consists of 15 116 games among MoHex and Wolve, and also recent Olympiad games.

Most of the computer games in our training data set come from tournaments in which we iterate over all possible opening moves, so the first moves in these games are generally not the strongest available move. Moreover, the human games are played with the swap rule, so the first moves in those games are also not the strongest for that position. Thus, as input for the MM pattern-learning algorithm, for all games we used every move except the first.

The human games often end with a resignation, specifically before either player has an absolute connection between their two sides. By contrast, most of the computer games end with an absolute connection. We obtained the best results when learning on the combined data set rather than on just one of the sets. This suggests that perhaps MoHex benefits from learning how to play endgames to completion, which is perhaps not surprising given that the simulations have no virtual connection knowledge.

We considered 6-patterns, 12-patterns, and 18-patterns, where a $6t$-pattern consists of the $6t$ cells nearest the pattern center.

In Hex it matters where a player's two sides are. For example, in order to know whether to extend a ladder, a player should know whose side the ladder runs into. Thus, in recording patterns, we allowed only one symmetry, namely rotation by 180 degrees.

Figure 7 shows some patterns learned by MM. In the figures, a is the number of times the pattern appeared, p is the number of times it was played when it was

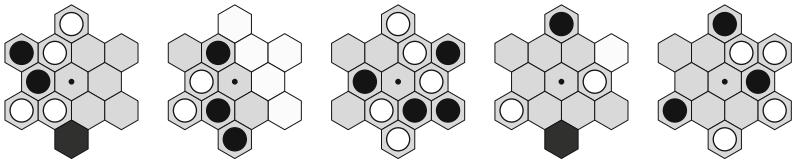

Fig. 7. Learned 12-patterns. Black to move to dot. Shaded cells are board sides. From left: P1 $(\gamma, p, a) = (886, 439, 479)$, P3 $(754, 179, 194)$, P13 $(754, 179, 194)$, P16 $(321, 48, 64)$, P36 $(213, 52, 65)$.

an option, and γ is the feature weight (as defined in [12]) learned by MM, where a larger value is more urgent. Not surprisingly, many of the high-γ patterns are some version of savebridge.

The patterns with high-γ do indeed look urgent. P1, with $(\gamma, p, a) = (886, 439, 479)$, is the pattern with largest γ, played 439 times out of 479: Black splits two White chains and threatens to connect to the Black side. In P3 Black joins two chains and splits the White chain from the White side. In P13 Black joins two chains and splits White chains; this is the highest γ pattern with no side cell. As with most high-γ patterns, this one probably occurs most often near the end of a game. In P16 Black splits White and virtually connects to the Black side via bridges. In P36 Black splits White with a bridge.

Figure 8 shows more patterns. P138 is the 6-pattern with largest γ. Black saves the bridge and splits White from the side.

The default γ-value for moves about which we know nothing is 1.0, so patterns such as P35518, P35474, P35461 — with γ less than .1 — are probably bad moves. P35461 is provably inferior: playing on the other side of the White bridge kills the dotted cell.

A pattern is *global* if it matches anywhere on the board, and *local* if it includes the opponent's most recent move. From our data sets we extracted 65 932 global patterns (33 212 asymmetric: 565 6-patterns, the rest 12-patterns). The maximum γ is 5 820, the minimum is 6.6×10^{-5}. We then ran our inferior cell engine on these patterns; 30 593 of them are *prunable*, namely correspond to a move that is dead, captured, or dominated.

Similarly, we extracted 11 602 local patterns (5 869 asymmetric: 550 6-patterns, the rest 12-patterns), of which 3 659 are prunable. The maximum γ is 11 281, the

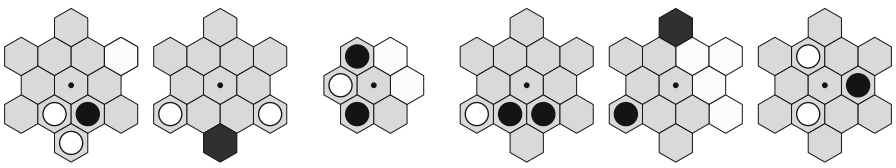

Fig. 8. More learned patterns. P49 (194, 2247, 3259), P135 (100, 86, 182), P138 (98, 94, 191), P35518 (.04, 0, 10190), P35474 (.05, 3, 14270), P35461 (.05, 6, 17351).

minimum is .0262. Prunable patterns corresponding to dead or captured (respectively dominated) moves have γ reset to .00001 (.0001). Patterns that did not occur in the data sets have γ set to 1.0.

4.4 Estimating Prior Knowledge

Following common MCTS practice [18], we use pattern weights as prior knowledge, namely to estimate the relative strengths of a node's unvisited children. When a tree node is visited for the first time, for every empty cell in that position, we identify its unique pattern. If the move is prunable, the move is not considered as a child option for this node. For non-prunable moves, we add the global and local γ-values, and then scale by dividing by the sum of the global and local γ-values over all non-prunable moves.

We use the resulting value as the prior estimated strength value ρ for that move. In addition to the above, an unvisited node has its RAVE value set to .5 and its RAVE count set to 8. Using 12-patterns (respectively 6-patterns, 18-patterns).

4.5 Progressive Bias

Following MCTS practice, e.g., as in Mango [10], we added a progressive bias term to the UCT formula. In our experiments, following Timo Ewalds's Havannah program [16], adding a square root to the denominator of the progressive bias term works best. Here ρ is the prior estimate from Sect. 4.4.

$$Score(j) = (1 - w) \times (U + E) + w \times R + PB. \tag{3}$$

Here $PB = c_{pb} \times \rho / \sqrt{n_j + 1}$, where $c_{pb} = 2.47$ from CLOP.

4.6 Probabilistic Simulations

Having learned the pattern weights, we implemented probabilistic simulations, in which moves are generated stochastically according to a softmax policy [25]. This worked well only after capping the maximum global γ value to a constant, $g = .157$ from CLOP. Thus moves with $\gamma \geq .157$ are equally likely to be played. By contrast, Timo Ewalds (private communication) found no improvement from using probabilistic simulations in the Havannah program Castro, possibly because any such improvement is overwhelmed by dynamic factors such as the sudden-death threat from rings.

Notice that resetting the γ values for inferior moves effectively prunes these moves in the simulation without having to call the inferior cell engine.

We are not sure why capping the global γ values is so critical. One possible factor is that the frequency of local moves in the simulations drops if the global values are not capped. In a typical MoHex 2.0 simulation on an empty 13×13 board, about .50 of the moves are local.

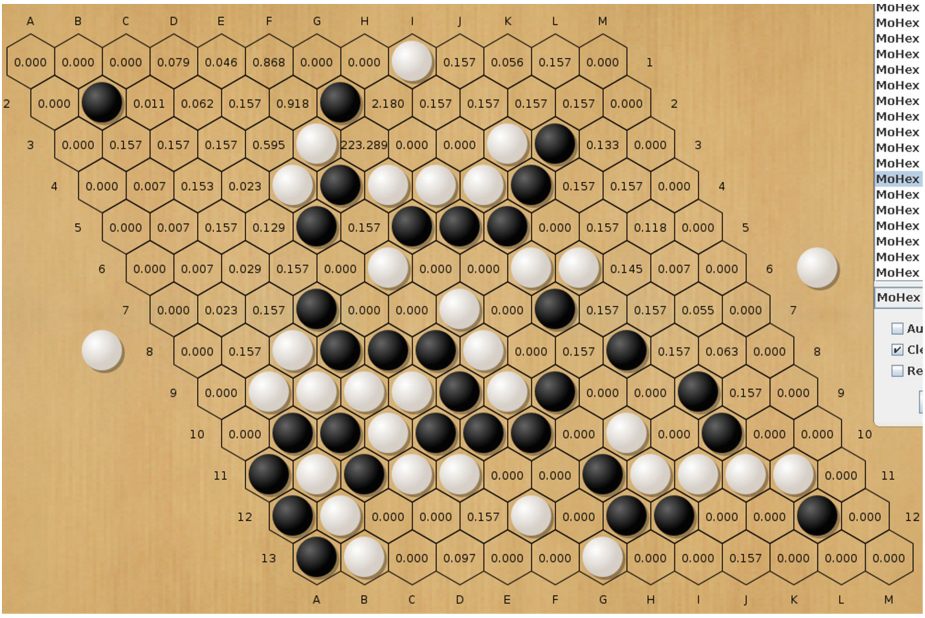

Fig. 9. Part way through a simulation.

Figure 9 is a screenshot from a simulation. The number in a cell is the corresponding move's total γ (local plus global). Notice that many cells have $\gamma = 0$, i.e., less than 0.0005, and so are bad moves or provably inferior, and so unlikely to be selected. The last move played was `black g2`, which threatens a win at `g3`. The total γ at `g3` is high — much higher than the capped global value of .157 — due to an urgent local pattern — so this move is likely to be selected next. If there was no global cap, this local response would be swamped by global values. It is a coincidence that this response blocks a win threat, because the simulation algorithm knows nothing about global connectivity.

Implementing probabilistic simulations is expensive and requires significant optimization for this feature to not have a negative effect on performance. In MoHex 2.0, this feature increased performance only slightly but was strong enough to overcome a significant (more than a factor of 2) decrease in simulation speed. We chose to keep this feature because we expect it will be useful in future developments.

4.7 Ideas that Did Not Work

In addition to the updates mentioned so far, we considered other changes. Most were not beneficial and so are not included in the final revision. Here are three changes that did not work out.

We were unable to find a stronger hand-crafted simulation policy. Here is a typical attempt: in addition to the 4-cell savebridge pattern, include 4-cell

patterns for breakbridge (break the connection when the opponent's last move leaves a partly broken bridge) and ladder continuation. For this set of three patterns, a promising win rate of .6 at 10 K simulations per move dropped to .5 at 100 K per move.

Degrading RAVE weight by distance to the last move, as in Fuego [24], did not work. Adding criticality (a measure of a cell's importance to the winner [13], perhaps less important in Hex than in Go) did not work.

5 Experimental Results

We ran 11×11 and 13×13 tournaments between MoHex 2.0 and the 2011 Olympiad version of MoHex. Each tournament iterated multiple times over all possible opening moves. Each player had 4 cores, 1.5 Gb memory, and time per game up to 5 min. Each thread ran MCTS, so neither player used the single-thread solver that MoHex has run in recent Olympiads. Over more than 1000 games, MoHex 2.0 had respective win rates of .81 and .85, meaning again of about 252 and 310 Elo points, respectively. See Fig. 10.

	allotted time per player		
board size	1 min	3 min	5 min
11×11		.811 ± .010	
13×13	.853 ± .006	.852 ± .006	.856 ± .010

Fig. 10. MoHex 2.0 v MoHex win rates. ± is standard error, 68 % confidence.

We also ran a 3000-game 3 min/game 13×13 tournament comparing these programs against the current version of Wolve. The MoHex and MoHex 2.0 win rates are .587±.008 and .854±.006 respectively. This is a 245 Elo points gain, showing that the improvements to MoHex are not just an artefact of self-play (Fig. 11).

6 Future Work

While MoHex 2.0 is stronger than MoHex, it is still far from perfect. For example, it does not find the winning move from the Panoramex-Wolve game shown in Fig. 6. There are several things to try next. We mention four of them.

(1) Larger patterns could be used, both in the tree and simulations. (2) Global connectivity information could be added to the simulations, which currently know nothing about winning or losing even when only one move away. (3) Simulation balancing could be used to learn simulation weights [25], which are currently those learned by MM and also used in the tree. (4) Adaptive simulation policies could be used.

program/feature	Elo gain
MoHex 1.0	—
new VC algorithm/MCTS formula	*
extend on unstable search	35
patterns in tree	175
probabilistic simulations	*
CLOP	46
total: MoHex 2.0	310

Fig. 11. Feature contributions to MoHex 2.0 on 13×13 board. * entries are not easily measurable in current framework.

References

1. Anshelevich, V.V.: Hexy wins Hex tournament. ICGA J. **23**(3), 181–184 (2000)
2. Anshelevich, V.V.: A hierarchical approach to computer Hex. Artif. Intell. **134**(1–2), 101–120 (2002)
3. Arneson, B., Hayward, R.B., Henderson, P.: Mohex wins Hex tournament. ICGA J. **32**(2), 114–116 (2009)
4. Arneson, B., Hayward, R.B., Henderson, P.: Wolve 2008 wins Hex tournament. ICGA J. **32**(1), 49–53 (2009)
5. Arneson, B., Hayward, R.B., Henderson, P.: Mohex wins Hex tournament. ICGA J. **33**(3), 181–186 (2010)
6. Arneson, B., Hayward, R.B., Henderson, P.: Monte-Carlo tree search in Hex. IEEE Trans. Comput. Intell. AI Games **2**(4), 251–258 (2010)
7. Björnsson, Y., Hayward, R.B., Johanson, M., van Rijswijck, J.: Dead cell analysis in Hex and the Shannon game. In: Bondy, A., Fonlupt, J., Fouquet, J.-L., Fournier, J.-C., Ramirez Alfonsin, J.L. (eds.) Graph Theory in Paris: Proceedings of a Conference in Memory of Claude Berge, pp. 45–60. Birkhäuser (2007)
8. Browne, C.: Hex Strategy: Making the Right Connections. A.K. Peters, Natick (2000)
9. Browne, C.: Connection Games: Variations on a Theme. A.K. Peters, Wellesley (2005)
10. Chaslot, G.M.J.-B., Winands, M.H.M., van den Herik, H.J., Uiterwijk, J.W.H.M., Bouzy, B.: Progressive strategies for monte-carlo tree search. New Math. Nat. Comput. (NMNC) **04**(03), 343–357 (2008)
11. Coulom, R.: Efficient selectivity and backup operators in Monte-Carlo tree search. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M.J. (eds.) CG 2006. LNCS, vol. 4630, pp. 72–83. Springer, Heidelberg (2007)
12. Coulom, R.: Computing "elo ratings" of move patterns in the game of go. ICGA J. 30(4), 198–208 (2007). http://remi.coulom.free.fr/Amsterdam2007/
13. Coulom, R.: Criticality: a monte-carlo heuristic for go programs (2009). http://remi.coulom.free.fr/Criticality/
14. Coulom, R.: CLOP: confident local optimization for noisy black-box parameter tuning. In: 13th Advances in Computer Games, pp. 146–157 (2011). http://remi.coulom.free.fr/CLOP/
15. Enzenberger, M., Müller, M., Arneson, B., Segal, R., Xie, F., Huang, A.: Fuego (2007–2012). http://fuego.sourceforge.net/

16. Ewalds, T.: Playing and solving havannah. Master's thesis, University of Alberta (2012)
17. Gardner, M.: Recreational topology. In: The 2nd Scientific American Book of Mathematical Puzzles and Diversions, Chap. 7, pp. 78–88. Simon and Schuster, New York (1961)
18. Gelly, S., Silver, D.: Combining online and offline knowledge in UCT. In: Ghahramani, Z. (ed.) ICML. ACM International Conference Proceeding Series, vol. 227, pp. 273–280. ACM (2007)
19. Hayward, R.B.: Six wins Hex tournament. ICGA J. **29**(3), 163–165 (2006)
20. Hayward, R.B.: Mohex wins Hex tournament. ICGA J. **36**(3), 180–183 (2013)
21. Hayward, R.B., van Rijswijck, J.: Hex and combinatorics. Discrete Math. **306** (19–20), 2515–2528 (2006)
22. Hein, P.: Vil de laere Polygon? Politiken, 26 December 1942
23. Henderson, P.: Playing and Solving the Game of Hex. Ph.D. thesis, University of Alberta, Edmonton, Alberta, Canada (2010)
24. Huang, S.-C.: New Heuristics for Monte Carlo Tree Search Applied to the Game of Go. Ph.D. thesis, Nat. Taiwan Normal Univ., Taipei (2011)
25. Huang, S.-C., Coulom, R., Lin, S.-S.: Time management for Monte-Carlo tree search applied to the game of go. In: 2010 International Conference on Technologies and Applications of Artificial Intelligence (TAAI), pp. 462–466, November 2010
26. Kuhn, H.W., Nasar, S. (eds.): The Essential John Nash. Princeton University Press, Princeton (2002)
27. Nasar, S.: A Beautiful Mind: A Biography of John Forbes Nash. Jr, Simon and Schuster (1998)
28. Nash, J.: Some games and machines for playing them. Technical report D-1164, RAND, February 1952
29. Shannon, C.E.: Computers and automata. Proc. Inst. Radio Eng. **41**, 1234–1241 (1953)
30. Wang, Y., Gelly, S.: Modifications of uct and sequence-like simulations for monte-carlo go. In: CIG, pp. 175–182. IEEE (2007)