cmput 355 hex notes by Prof Hayward part 1/2

solving games

The games we have discussed so far have been solved by mostly exploiting the search space. In tic-tac-toe the space is small, so minimax alone works ok. In nim (after grouping isomorphic positions together, and considering transpositions) the state d.a.g. is small, so minimax again works ok. (Nim also has a formula that allows large positions to be solved almost instantly.)

In chess, the state space is too large to allow states to be solved exactly, but bots finally outperformed humans in the 1990s, based mainly on alpha-beta-style programs that use strong move ordering heuristics to perform narrow but deep searches.

In checkers the game simplifies towards the end, usually turning into a battle between kings only on each side. Using an end-game database — similar to the approach in nim, where game values are computed from the end game back to the starting positions — 8×8 checkers was solved exactly by a UAlberta team led by Jonathan Schaeffer. 10×10 checkers, also called draughts, has not yet been solved.

In Hex we will see a different approach. The strongest Hex programs need to exploit a certain kind of mathematical knowledge about the game that greatly reduces the search space. Today's Hex programs are not quite as strong as the strongest humans, but using AlphaGo-style learning, in a few years they probably will be.

solving Hex states

Two ideas are useful when solving Hex states: inferior cells and virtual connections.

First, some definitions. A *state* or *situation* $S = (P, X)$ is defined by the board *position* P and the *player-to-move (ptm)* X, ie. the player X whose turn it is to move next. An algorithm is *explicit* if, for any state reachable from the start state, it can computes the next move of the strategy in time polynomial in the boardsize.

To *solve* S is to determine the outcome, assuming perfect (ie. minimax) play by both players. Hex has no draws, so the winner of S is either ptm (ie. 1st player to play from S), or the opponent of ptm (ie. 2nd player to play from S).

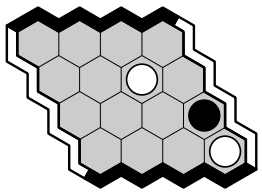
If we solve S and learn only the outcome, but not how to play to achieve that outcome, then we have *weakly solved* S. If in addition we find an explicit provably-winning strategy (ie. we can prove that it wins against all possible opponent strategies), then we have *strongly solved* S.

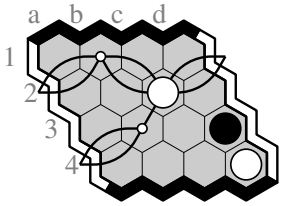
An example. Let S be the state consisting of the empty 100×100 board. By the strategy stealing argument, we know that there exists a winning strategy for the first player. So S is weakly solved.

The only algorithms that I know that solve Hex positions take time at least exponential in the boardsize. As far as I know, no one has strongly solved S. Today, the best solver takes several months to solve the hardest 9x9 position.

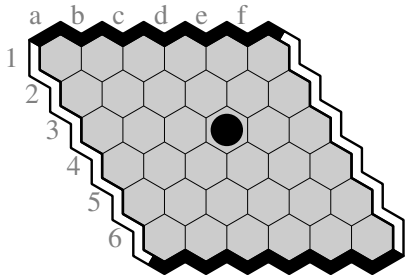
Another example. Strongly solve this position, with Black to play. Hint: the second diagram shows you a winning strategy. The white stone at c2 is virtually connected to the right side using

cell set {d1,c2}. On the left, if the opponent plays in any of {a1,a2,b1,b2,c1} then reply at b3, virtually connected to the left with {a3,a4}; if the opponent plays in any of {b3,a3,a4}, reply at b1, virtually connected to the left with {a1,a2} and to c2 with {c1,b2}.





Another example. Let S be the 6×6 position with a black stone on the centermost cell. It is not too difficult to find an explicit provably winning strategy for Black (so White, who plays next, loses). So S is strongly solved.

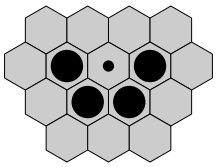


Challenge problem. White to play: find a winning Black strategy.

inferior cells

When searching for a winning move, sometimes some cells can be easily ruled out. The idea in inferior cell analysis is to identify cells that are inferior in some sense to other cells. In the search for a single winning move, moves to inferior cells can be ignored.

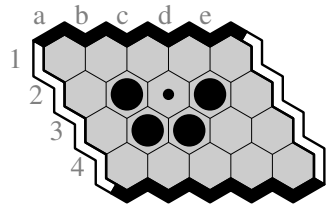
Consider the following part of a Hex position. The dotted cell is useless: neither player should play there. Can you see why?



dead cells

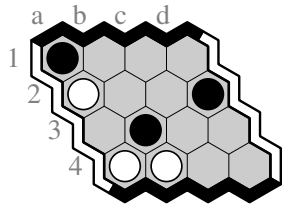
For a Hex position P and player X, a *joinset* is a set of empty cells which, when X-colored, joins X's two sides. A *joinset* is minimal if is not a proper subset of some other joinset.

Eg. below, find a joinset for Black, find a joinset for White, find a minimal joinset for Black, find a minimal joinset for White. Can you find a minimal joinset for Black that includes the dotted cell `c2`? For White? What does this tell you about the dotted cell?

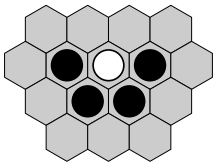


Eg. in the position below, the set { `a3 b2 c2 d1` } is a white joinset, but it is not minimal, because it is a superset of the white joinset {`b2 c2 d1` }. For a position P, a cell is *live* if it is in some minimal joinset (for either player), otherwise it is *dead*.

Eg. cells `b2, d2, d1` are all live: they form a white joinset, and if we remove any one cell from the joinset, it is no longer a joinset. So the joinset is minimal, so each cell in it is live. Exercise: explain why `a3` is dead.

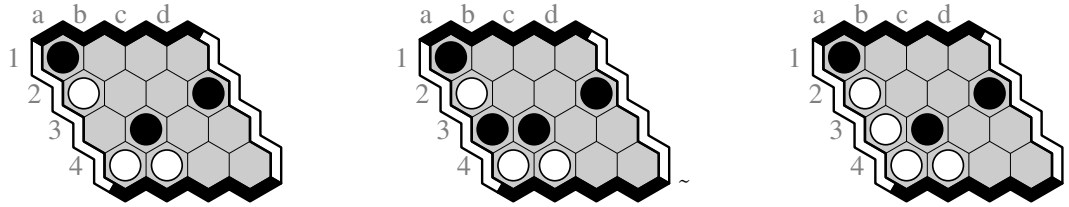


We extend the definition of dead from empty cells to colored cells: we call a colored cell *dead* if, after uncoloring, it is dead. Eg. the white stone here is dead.



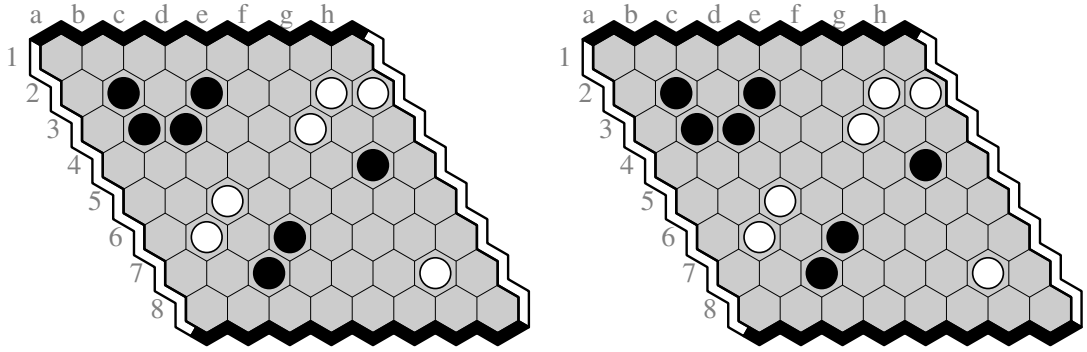
theorem For any Hex position, changing the color (empty, black, white) of a dead cell does not change the winner of the position.

Eg. fix the player to move: then the winner of each position below is the same.



So, when searching for a winning move, if we see that a cell is dead, we can prune it from our search.

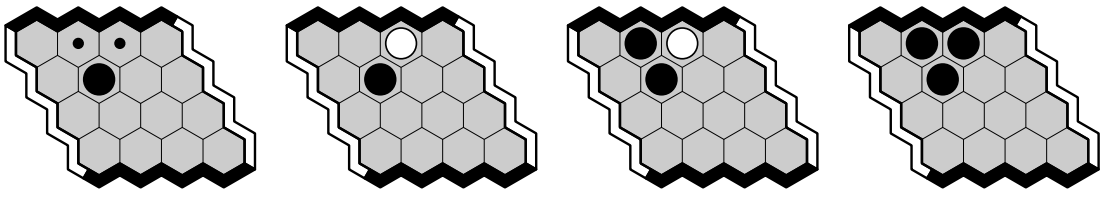
Sometimes, as above, we can tell that a cell is dead just by looking at the neighbouring cells. How many dead cells can you find here?



captured cells

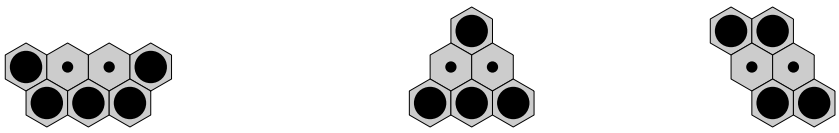
In the 1st diagram below, if White ever plays at one dotted cell (as in the 2nd diagram), Black can reply at the other dotted cell (as in the 3rd diagram), and so kill the White cell. So: it is useless for White to play at either dotted cell. We call a set of cells *black-captured* if Black has a replying strategy that leaves every cell in the set black or dead.

useful in solving Hex states: fill each Black-captured sets with black stones. This reduces the size of the search tree.



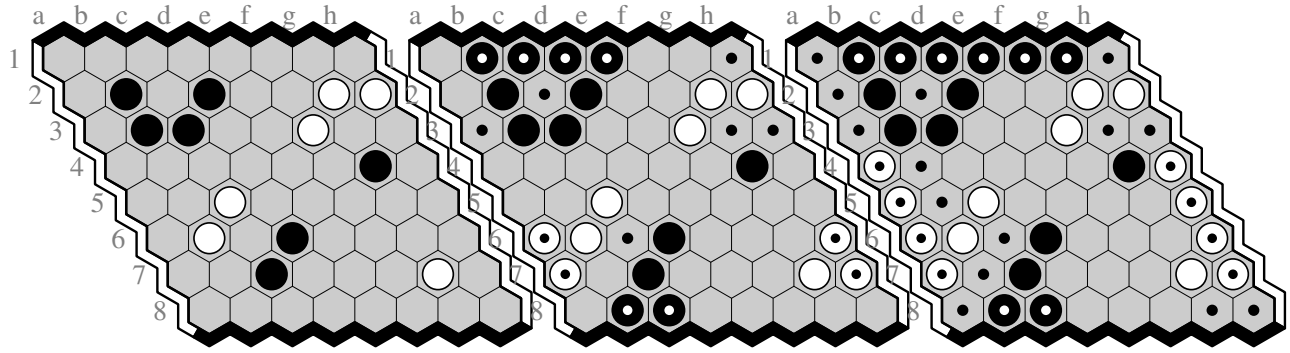
Here are three common captured set patterns. Page 55 of this paper shows others:

`http://www.cs.ualberta.ca/~hayward/papers/bergeParis.pdf`



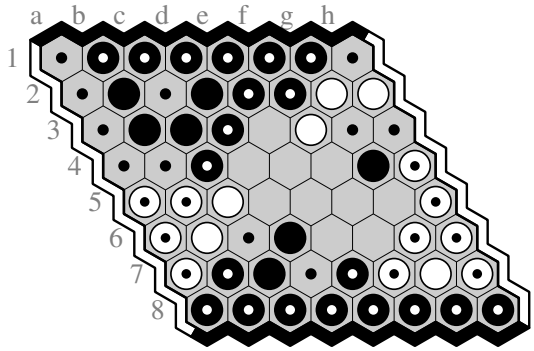
On the 8x8 board above, fill in as many dead and captured cells as you can.

Here, with each step I fill in more dead and captured cells. Whoever wins the first state wins the last state. Obviously, this kind of fill-in reduces the search space.



Finally, here is one more fillin pattern, called *permanently inferior*: whenever you see the pattern at the left, you can add a black stone as on the right. If you want to know why this is ok, read this paper `https://www.cs.ualberta.ca/~hayward/papers/revDom.pdf`.

If you fill in the above 8x8 board using dead, captured, and permanently inferior fillin, you get end up with something like this:



solving 3x3 Hex

The number of possible 3x3 Hex games is less than 9! and checking whether a player has won is easy: use a shortest path algorithm or a union-find algorithm to check whether a player has joined their two sides. (For small boards such as 3x3 you can also check all minimal cells that that join a player's two sides.) So negamax is sufficient to solve 3x3 Hex. Program `simple/hex/hex-3x3.py` uses the latter method to check for wins, and solves any 3x3 position with a negamax tree that has under 10 000 nodes. With better move ordering, the tree is smaller by a factor of about 10.

solving 4x4 Hex

Our bound on the number of 4x4 games is 16! or about 2.1×10^{13}, so now we have to be more careful. One approach is to use a transposition table and so a search d.a.g. instead of a search tree. Using `simple/hex/hexCount.py` you can count the number of non-isomorphic Hex board positions where the number of black and white stones differ by at most 1: for the 4x4 board this is around 5 million, so this approach would work. (The number of reachable Hex positions is even smaller, because many of these positions already have a winning condition.)

With smart move ordering, negamax searches around 7 million nodes so solve the empty board.

One way to improve negamax search in Hex is to back up win and loss information and compute mustplay sets in this way, as in `simple/hex/hex-vc.py` . This program solves the empty board with fewer than 300 000 nodes.

The improved version `simple/hex/hex-vc.py` tries to find smaller mustplayer regions, it solves the empty board with fewer than 80 000 nodes.

solving 5x5 and larger Hex

Compute virtual connections, especially bridges: use this to check for wins long before a player has a finished winning path. Fill in dead and captured cells.

solving 7x7 Hex

Here is how mustplay analysis works. When it is your turn to play, consider the opponent's threats. Is there any empty cell, such that if the opponent plays there, they then have a winning virtual connection? If yes, then the union of that cell plus the cells used by the virtual connection is an *opponent threat-set*. So, find all opponent threat-sets that you can, and then take the intersection of all of them: this combined intersection is your *mustplay region*. Your next move must be in this region: if it is not, then you have missed at least one opponent threat-set, and after you move your opponent can win.

Around 2003, a group at UAlberta (Björnsson, Hayward, Johanson, Kan, Po, Van Rijswijck) were trying to solve Hex positions.

By pruning dead cells, computing virtual connections and using mustplay analysis they could solve positions on 6x6 but not 7x7.

And then they realized that filling captured cells with stones of the capturing player does not change the outcome of the game. So they added that feature to their program and then could solve all 7x7 positions. More details are here

`http://webdocs.cs.ualberta.ca/~hayward/talks/hex.solve7x7.pdf`

towards 11x11

As with the first-move Black strategies below, many Hex positions have concisely describable winning strategies. To date, all opening moves on all board sizes up to 9x9 have been solved, and two opening moves on 10x10. See

`http://webdocs.cs.ualberta.ca/~hayward/talks/hex.sol10.pdf`

David Pankratz wrote a Hex strategy visualizer that might help find a new 10x10 winning strategy. It would be great if someone could find a winning 11x11 Hex strategy. This is the original boardsize introduced by Piet Hein in 1942 when he introduced the game (then called Polygon) to readers of the Danish newspaper *Politiken*.

$b2 \wedge (b1 \vee c1) \wedge (a3 \vee b3)$

```
b2.(b1|c1).(a3|b3)
```

```
.   x   x                  b2   b2+
  .   1   .                |     |
    y   y   .            b1 c1 a3 b3
```

$a3 \wedge (a2 \wedge (a1 \vee b1) \ \vee \ c1 \wedge (b2 \vee c2 \wedge (b3 \vee c3)))$

```
a3. (a2.(a1|b1) | c1.(b2| c2.(b3|c3)))
```

```
      x   x   2y                  a3
        2x  y   y           a2          c1
          1   z   z       a1 b1     b2    c2
                                       b3  c3
```

$a2 \wedge (a1 \vee b1) \wedge (a3 \ \vee \ c2 \wedge (b2 \vee c1) \wedge (b3 \vee c3))$

```
a2.(a1|b1).(a3| c2.(b2|c1).(b3|c3))
```

```
x   x   y                       a2            a2+
  1   y   2y                 a3   c2    c2+    a1  b1
    2x   z   z                  b2 c1  b3 c3
```

review questions

1. Define *Hex state*. Give an example.
2. For a given Hex state, define *explicit algorithm*. Give an example.
3. For a 2-player game state, define *strongly solved*. Show that 3×3 Hex (from the empty board with Black to play) is strongly solved: give an explicit winning strategy.
4. For a 2-player game state, define *weakly solved*. Explain why 11×11 Hex (from the empty board with Black to play) is weakly solved but not strongly solved.
5. In words, describe the winning white virtual connection represented by the curvy-line diagram at the top of page 2 (on the right).
6. For a Hex position, define *joinset*, *minimal joinset* and *dead cell*. For the position at the top of page 3, give a non-minimal White joinset, a minimal White joinset, and a dead cell.
7. In the middle position on page 3, explain why cell `a3` is dead.
8. Explain why on page 3 we needed to *extend* the definition of dead cell.
9. In the position in the second row of page 4, identify the 3 local patterns that have 1 empty cell that is dead because of 4 adjacent colored cells.
10. at the top of page 5, explain exactly how each cell in the middle diagram has been filled in.
11. at the top of page 5, explain exactly how each cell in the right diagram has been filled in.
12. on page 5, explain exactly how each cell in the bottom diagram has been filled in.
13. prove the theorem on page 4

hints and answers to review questions

1. definition in the notes. e.g. any Hex position, Black to play (or White to play).

2. definition in the notes. e.g. for 2×2 Hex, here is an explicit first-player strategy for Black: 1.B[a2]. If 2.W[a1] then 3.B[b1] and the game ends; if 2.W[b1] then 3.B[a1] and the game ends; if 2.W[b2] then 3.B[a1] and the game ends. You could also describe this strategy with a proof tree. You could also describe this strategy like this: 1.B[a2]. Now Black maintains the virtual connection between a2 and the top: if White plays in either of {a1, b1}, Black plays in the other.

3. definition in the notes. e.g. here is an explicit first-player winning strategy for Black: play in the center at b2. Now main the virtual connection between b2 and the top using {b1, c1} and the virtual connection between b2 and the bottom using {a3, b3}.

4. definition in the notes. By Nash's strategy stealing argument, there exists a winning first-player strategy for any $n \times n$ Hex board.

5. The white cell at $c2$ is virtually connected to the right side using $\{d1, c2\}$. Also, this cell is virtually connected to the left side using $\{a1, a2, b1, c1, b2, b3, a3, a4\}$.

   If Black plays at any of the first five cells listed in this set, White plays at $b3$ and maintains the connection using $\{a3, a4\}$.

   If Black plays at any of the last three cells listed in this set, White plays at $b1$ and maintains the connection using $\{a1, a2\}$ and $\{c1, b2\}$.

6. definition in the notes.

   $S = \{a1, b1, c1, c2, d1, e1\}$ is a White joinset, but not minimal.

   $S - c2 = \{a1, b1, c1, d1, e1\}$ is a minimal White joinset.

   in this position, $c2$ is dead and every other empty cell is live.

7. Recall that a *path* is a sequence of nodes, with no repeated node, such that each consecutive pair is adjacent (touching). So any path that includes $a3$ and joins Black's two sides must include both $b2$ and $b3$, and these two are already adjacent. So if Black has a joinset J that includes $a3$, then $J - a3$ is still a Black joinset. So no Black minimal joinset includes $a3$. So $a3$ is dead.

8. The original defintion of dead cell assumes that the cell is empty. So we needed to extend the original definition to also include cells that have a stone.

9. Each of c2, g3, c6 is the center of a dead cell pattern.

   But so is a3: use the two black cells at b2, b3, and put white cells at z3 and z3 (where z is the white column on the left side of the board).

   And so is h3: use the black cell at g4 and white cells at h2, i2, i3 (where i is the white column on the right side of the board).

   And so is h1: use the black cell at h0 and white cells at g2, h2, i1.

10. As explained in the previous answer, cells a3, c2, c6, g3, h1, h3 are dead, so they be filled with either color. Because of the black cells at b2, b0, c0, d0, cells b1,c1 are black-captured. Similarly, d1,e1 are black-captured, b8,c8 are black-captured, a6,a7 and h6,h7 are white-captured.

11. After filling cells as described in the previous answer, every other marked cell can be filled by applying one of the three dead-cell patterns or one of the three captured-cell patterns.

12. Fill cells as described in the previous answer. By the inferior-cell pattern, black-color d8 is black. Now d7 is dead, so black-color d7. Assume that dead cell g8 is black, so use captured-cell pattern with black cells d8, d9, e9, f9, g8 and black-color e8, f8. Now use the inferior-cell pattern and black-color e7. Now notice that a white move to g6 kills f7 and vice versa so, by capturing, white-color g6,f7. A black move to e2 kills f2 and vice versa, so black-color e2,f2. A black move to c4 kills d3 and vice versa, so black-color c4,d3.

13. **proof** For a state X, let c be a dead cell. Assume that Black has a winning strategy S that at some point plays at c. Let S' be this Black strategy obtained from S: whenever the strategy calls for a move to c, pass (or play anywhere else); whenever the strategy calls for a move to a cell that already has a Black stone, pass (or play anywhere else).

    At each terminal position of the game tree when Black follows S, Black wins, so Black has some winning path. This winning path is still winning when Black follows S'. The only cell that Black occupies under S but not S' is c, but c is not on any minimal joinset, so at every terminal position reached by following S, Black has a winning path, and a minimal subset of these cells is also a winning path following S'.