Solving Large Hex Games

CMPUT 355: Games, Puzzles, and Algorithms

Lecture Outline

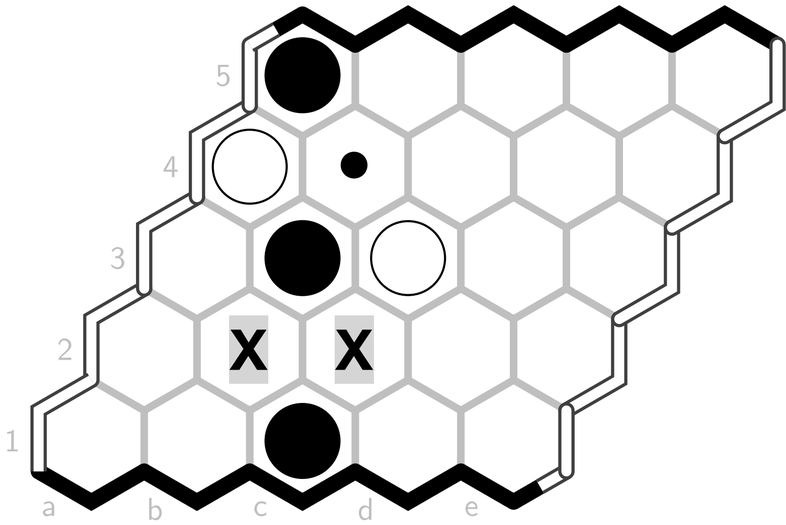
1. Logistics & Recap
2. Virtual connection patterns
3. Win-threat pruning

Logistics

- **Practice questions #5** will be available on **Friday** (Mar 20)
- **Quiz #3** marks have been posted
  - Solutions are available on the website
- **Quiz #4** marking is in progress

Recap: Virtual Connections

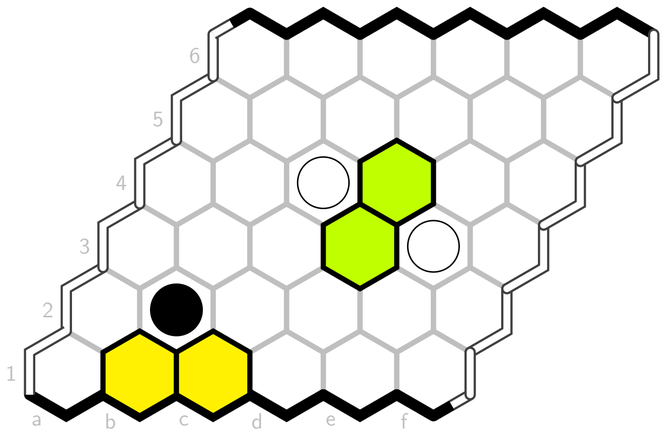
- **Virtual connection:**
  A player **can** connect two cells, but they haven't **yet**
- **Semi connection:** A player can force a connection between two cells, but only if they are the **current** player to move
  - **Example:** Black can force a connection between **b3** and **a5**, **if it is Black's turn**.
- **Full connection:** A player can connect two cells whether they are the **current** player to move *or* the **next** player to move.
  - **Example:** Black can force a connection from **b3** to **c1** edge **even if it's White's turn**.
  - Two nodes are fully connected if they have **2 or more semi-connections**



Virtual Connection Patterns

- Virtual connections are easily recognized by looking for **cell patterns**
- **Recognizing** opponents' virtual connections lets you recognize **win threats**
  - *Without* having to perform a full search
- Recall: If an opponent has a win threat, you **must** block it (**why?**)
- **Question:** What can you do if an opponent has **multiple** win threats?

Pattern: Bridge

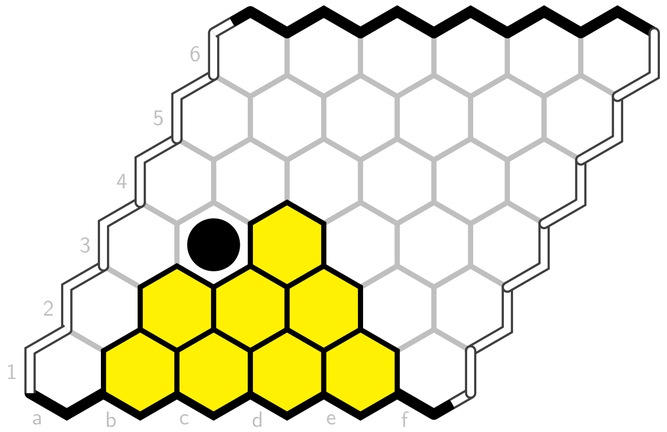


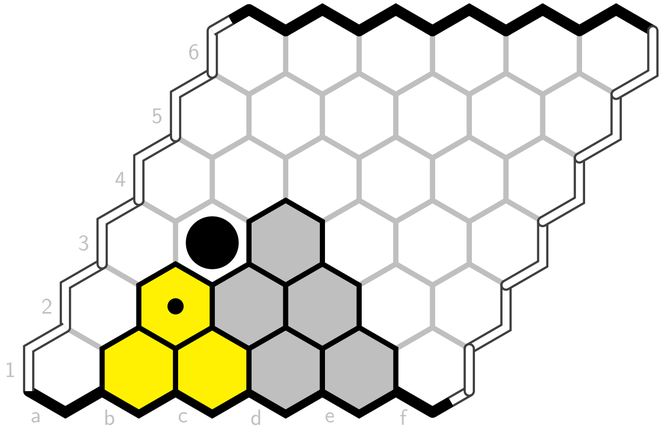
Examples:

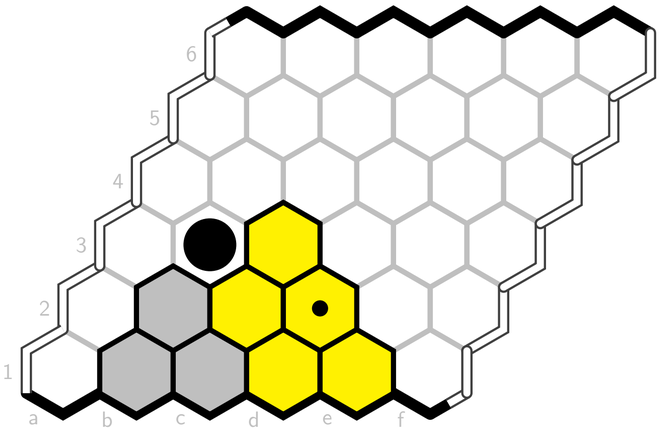
- The yellow cells are a bridge between **b2** and the **bottom** border
- The green cells are a bridge between **c4** and **e3**

- In a **bridge** pattern:
  - **Origin** stone is adjacent to **two empty cells**
  - **Both** empty cells are adjacent to the **destination** stone
- A bridge between two stones **fully connects** those stones
  - **Current** player can connect the stones by playing on **either** of the empty cells
  - **Next** player can connect the stones by playing on whichever cell the **current player does not play on**

Pattern: 2-3-4

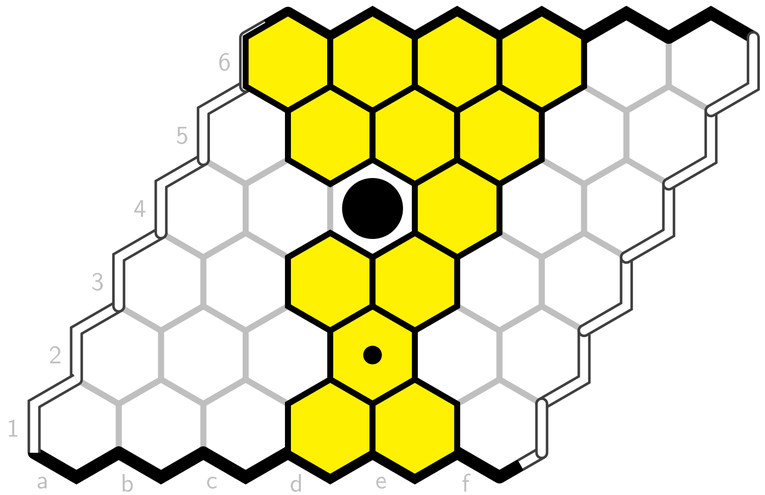






- In a **2-3-4** pattern:
  - **Origin** stone is adjacent to an empty cell on **same row**
  - Both cells on the origin row are **doubly-adjacent** to a line of **three** empty cells on "next" row
  - Each of the three empty cells are doubly-adjacent to a line of four empty cells on the "next" row
  - The row of four empty cells is adjacent to a border
- A 2-3-4 pattern **fully connects** a stone to a border:
  - **Next** player can play on next cell on **own** side of next row
    - That cell has a bridge to the border
  - OR, Next player can play on **opposite** side of the next row
    - There is a bridge from stone to that cell
    - And a bridge from that cell to the border
- **Question:** What should the **current** player do?

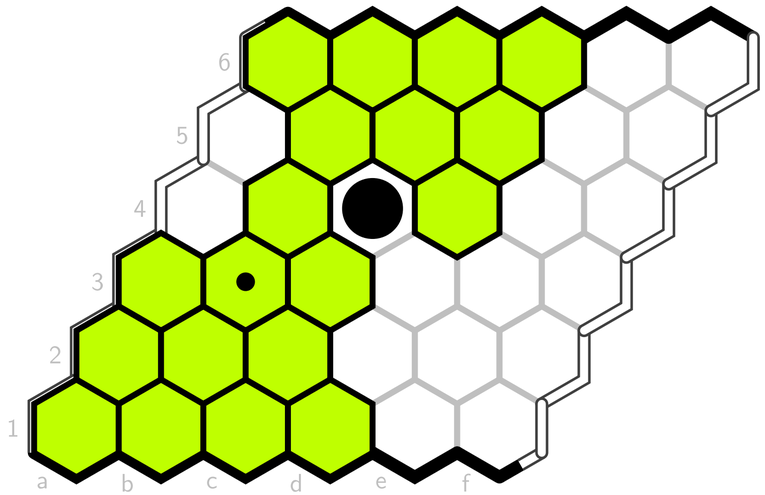
Example: Win threat 1



Questions:

1. Is this a full connection or a semi-connection between **top** and **bottom**?
2. How could we block this connection?

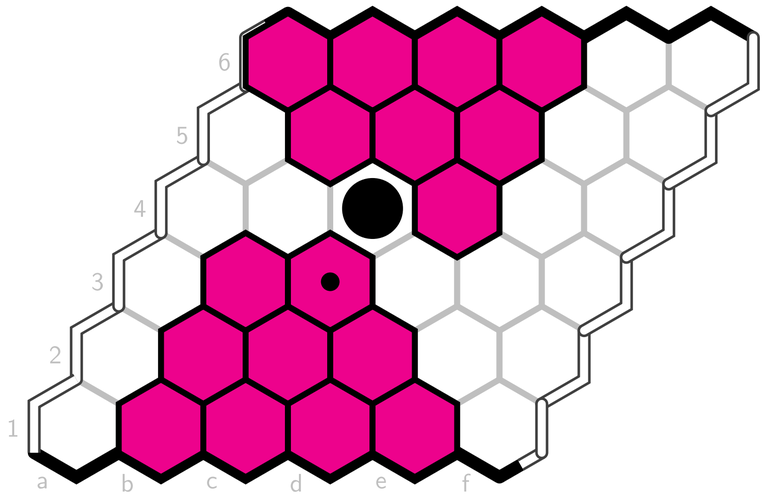
Example: Win Threat 2



Questions:

1. Is this a full connection or a semi-connection between **top** and **bottom**?
2. How could we block this connection?

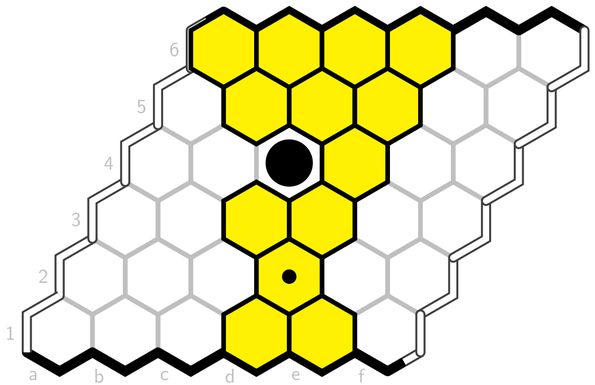
Example: Win Threat 3

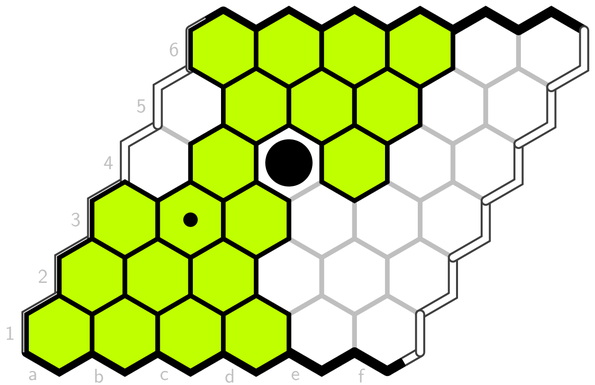


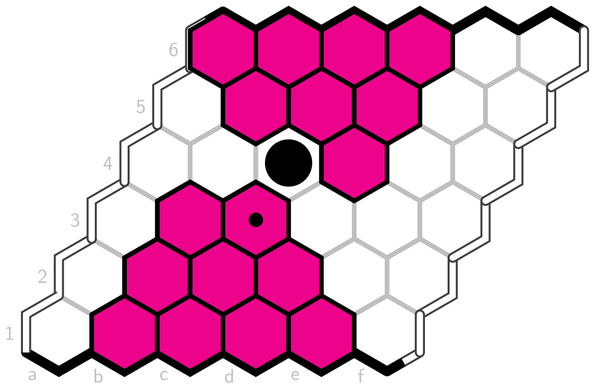
Questions:

1. Is this a full connection or a semi-connection between **top** and **bottom**?
2. How could we block this connection?

Blocking Multiple Threats

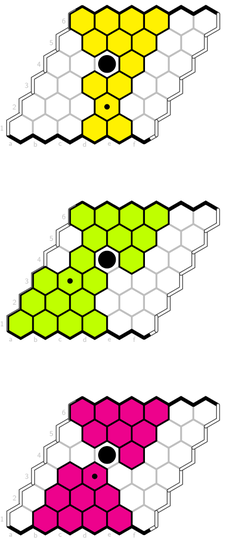


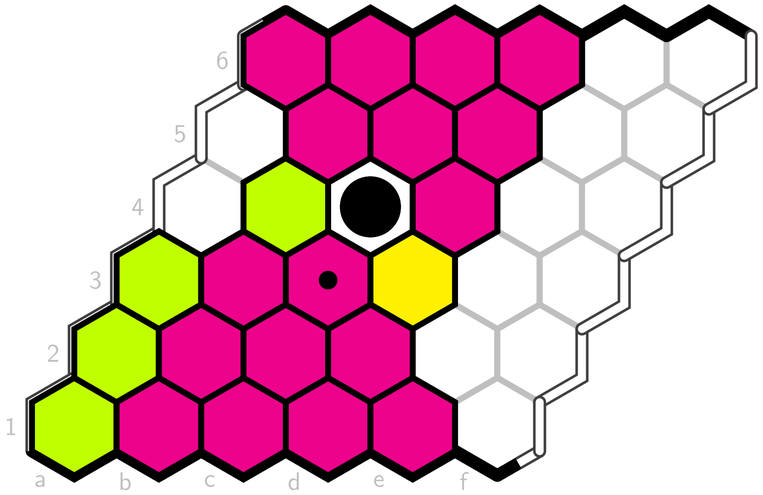


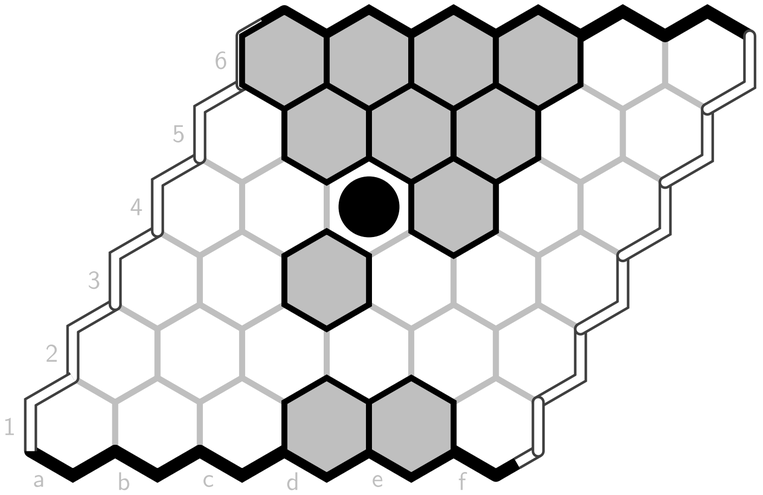


- We can block each semi-connection by playing on the "bottleneck" (the dot)
- But each of these win threats has a different bottleneck!
- **Question:** Can we block all of them at once?

Must-Play Region







- The only way to block multiple win threats with a single move is to play on their **intersection**
- The intersection need not be a bottleneck (**why not?**)
- The intersection of multiple win threats is called the **must play** region
- It is sufficient to only consider moves in the must play region (**why?**)
- This can dramatically speed up search by reducing the number of moves to consider

Win-Threat Pruning

- **Question:** How can we use win-threat detection to prune our search?
- One approach: **Find** all the win-threats and take their intersection
- Another approach: Use the results of the **recursive search** itself
  - If a given move is a losing move, then the stones used by the opponent are part of a win threat!
- Example implementation: **win_move** in **hex/hex_vc3.py**

Example Implementation: hex/hex_vc3.py

```
def win_move(s, ptm): # assume neither player has won yet
  """
  s        board, as string
  ptm      player to move, as character
  return   winning move if ptm has winning move, else ''
           count   total number of calls
           win_set virtual connection for winner
             if ptm:  win_move U win_set is ptm winning s-c
             if op't:           win_set is opt winning v-c
  """
  optm = oppCH(ptm)

  calls = 1
  mustplay, opt_win_threats = set(), []
  for j in CELLS:
    if s[j]==ECH: mustplay.add(j)
```

```
  while len(mustplay) > 0:
    for move in CELLS:
      if move in mustplay: break
    t = change_str(s, move, ptm) # resulting board
    if has_win(t, ptm):
      return point_to_alphanum(move, COLS), calls, {move}
    omv, ocalls, oset = win_move(t, optm)
    calls += ocalls
    if not omv: # opponent has no winning response to ptm move
      oset.add(move)
      return point_to_alphanum(move, COLS), calls, oset
    mustplay = mustplay.intersection(oset)
    opt_win_threats.append(oset)
```

```
if z > 1: ovc = ovc.union(opt_win_threats[z-2])
if z > 2:
  inter = opt_win_threats[z-1].intersection(opt_win_threats[z-2])
  j = z - 3
  while len(inter) > 0:
    inter = inter.intersection(opt_win_threats[j])
    ovc = ovc.union(opt_win_threats[j])
    j -= 1
return '', calls, ovc
```

Summary

- **Virtual connections** are easily recognized using **patterns**
  - Example 1: **Bridge**
  - Example 2: **2-3-4**
- These patterns can be **combined** to create compound virtual connections
- A virtual connection between two borders is a **win threat**
- A win threat **must be blocked** when it is present
- This means we can **prune** our search to **only include** moves that block win threats