

# Lecture 13: Heuristics

CS442: Great Insights in Computer Science  
Michael L. Littman, Spring 2006

## Targum: Valid? NP? P?

- **Jumble:** Given a word list  $l$  and a jumble of letters  $w$ , can the letters of  $w$  be rearranged to form a word in  $l$ ?
- **Crossword clue:** Given a clue, and an answer word  $w$ , does  $w$  answer the clue?
- **Crossword grid:** Given answers for each slot, do they fit in the grid without conflicts?
- **Sudoku:** Given a partial grid, can it be completed properly?
- **Word find:** Does word  $w$  appear in grid  $G$ ?

# Algorithms and Reality

- Algorithms are a great way to solve problems.
- We saw last time that some problems don't seem to admit efficient algorithms (NP-completeness).
- A branch of CS studies "approximation algorithms", which guarantee near-optimal solutions.
- Such algorithms are rare and some problems are NP-complete to approximate.

# Heuristics

- Although the idea is often abused, there is a place for solutions to problems that don't *always* work, but often work well in practice.
- Hillis covers two very important categories:
  - search with heuristic evaluation functions.
  - local search.
- Principally studied in Artificial Intelligence.

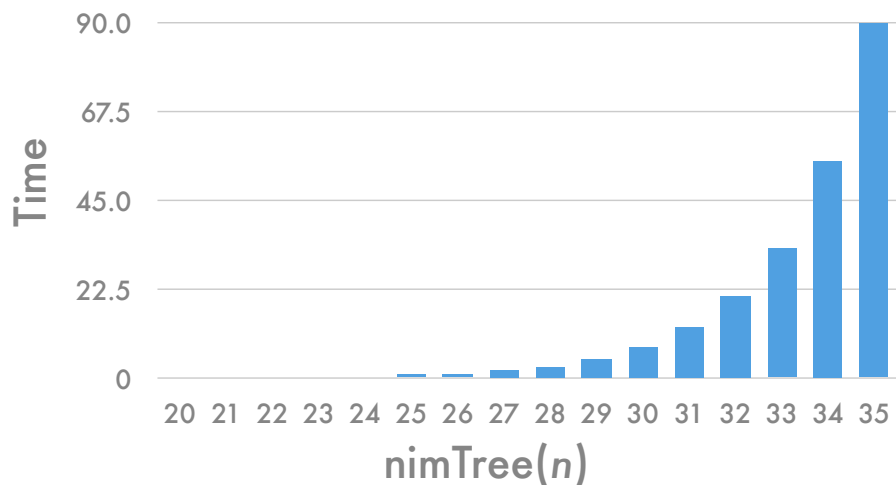


# Minimax Algorithm

```
def nimTree(x):  
    if x == 0: return -1  
    if x == 1: return -nimTree(x-1)  
    return max(-nimTree(x-1), -nimTree(x-2))
```

- 1 = I win, -1 = I lose
- If it's my turn and there are zero left, I lose.
- Take one or two. See if opponent wins with that many items. Negate the result to translate to a win or loss for me.

## Growth of nimTree

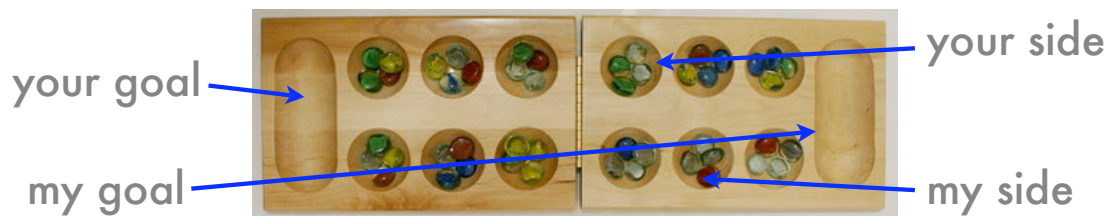


- Clearly exponential growth: time =  $10^n$ .

# Can't Go On Forever

- At this rate, nimTree(43) would take an hour.
- Chess games typically take around this many moves and the branching factor is much much higher.
- Need to stop searching at some point...

# Mancala, Awari, etc.



- On turn, choose a pit on your side. "Sew" stones, one per pit, counter-clockwise (skip opponent's goal).
- If last stone in goal, go again. If last stone on your side in empty pit, capture last stone and adjacent pit.
- Win if you have more stones in your goal when no more moves possible.

# Game Size

- Game starts with 4 stones in each of the 12 pits.
- Game lasts around 40 moves. Each move about 6 choices. So, game tree is about  $6^{40} = 10^{31}$  nodes.
- Lots of duplicate boards in the game tree, though. How many ways can we put the 48 stones in the 14 different bins?

# Stones in Bins

	2 bins	3 bins	4 bins	5 bins
0 stones	1	1	1	1
1 stone	2	3	4	5
2 stones	3	6	10	15
3 stones	4	10	20	35
4 stones	5	15	35	70
5 stones	6	21	56	126

How many ways can you put  $s$  stones into  $b$  bins?

Look familiar?

# Mini Awari

- So, for this game, there are about 6.5 trillion board positions.
- Smaller game: 6 pits, 3 stones per pit to start.
- “Only” 480k positions. Shorter games.



# Choosing a Line of Play

- Let's say you have two possible lines of play.
- One puts 4 stones in your goal and other 2.
- All other things being equal, the former moves you closer to winning.
- Idea: Rate boards as being “better” or “worse”. If you aren't sure whether you will win or lose, move to bring about better boards.

# Heuristic Function

- Lots of possibilities.
- Simple one: the goodness of the board is the number of stones in my goal minus the number of stones my opponent has (plus 0.5 if it's my turn).

# Against Random Player

- Player 1 chooses randomly.
- Player 2 uses the heuristic after search to the given depth.
- Play 10 games.
- Deeper search, generally better results.

search depth	record
random	2-8
1	6-2-2
2	8-2-0
3	9-1
10	10-0
15	9-1-0
16	10-0

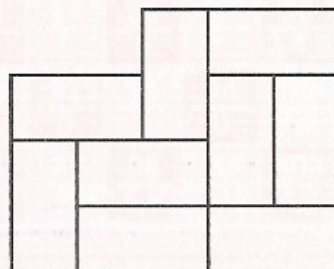
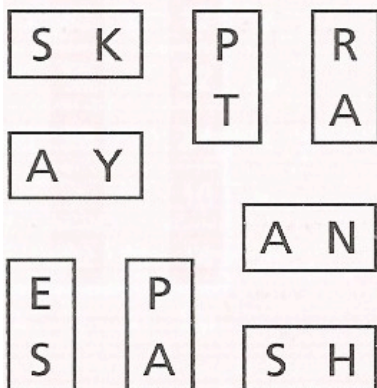
# Deep Blue

- IBM's champion chess program used three main ideas:
  - opening book
  - deep game-tree search
  - end-game database
- Evaluated 200M positions per second.
- Searched to a depth of about 12. 3.5-2.5 vs. human champion Kasparov (May 1997).

# Puzzle from Games

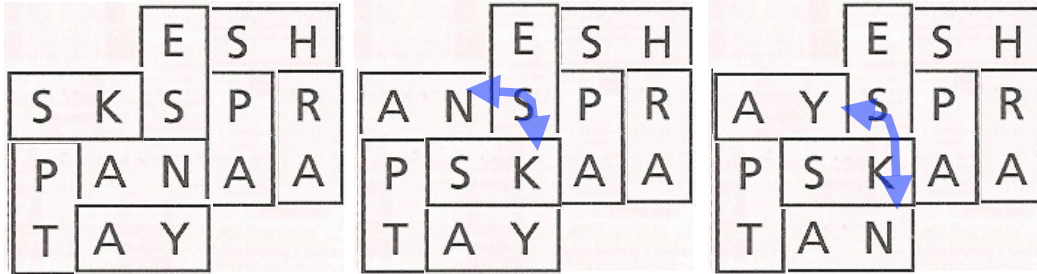
## PIECE OFFERING

In the puzzle below, fit the pieces into the frame to form familiar words reading across and down, in crossword fashion. You won't need to rotate the pieces; they'll fit as shown. Each piece will be used only once. Put the puzzle together correctly and you win the Piece Prize!



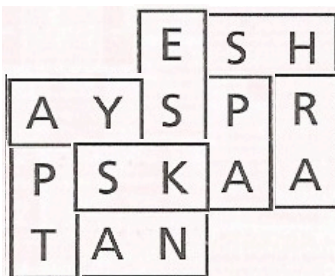
$4! \times 4! = 576$   
layouts

# Random Start State



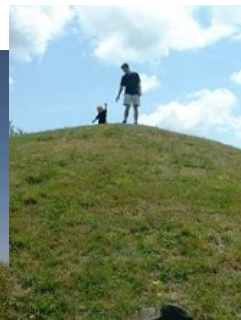
- across: SKSPR
  - down: spa
  - Score = 1
- across: ANSPR
  - down: apt, spa
  - Score = 2
  - Improvement
- across: tan
  - down: apt, spa
  - Score = 3
  - Improvement

# Now, We're Stuck

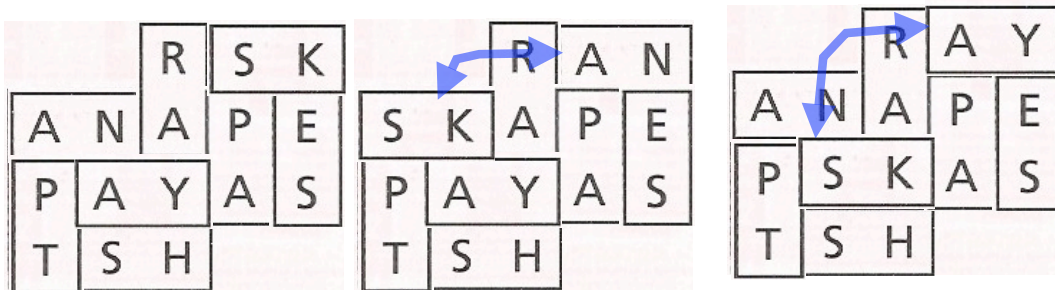


- across: tan
- down: apt, spa
- Score = 3

- I tried all 8 pairwise swaps.
- None result in any improvement.
- This state is called a *local maximum*, since there's no way up from here without going down first.

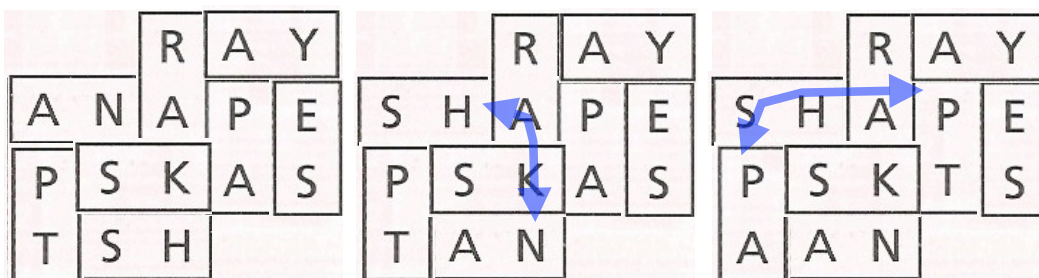


# Random Start State #2



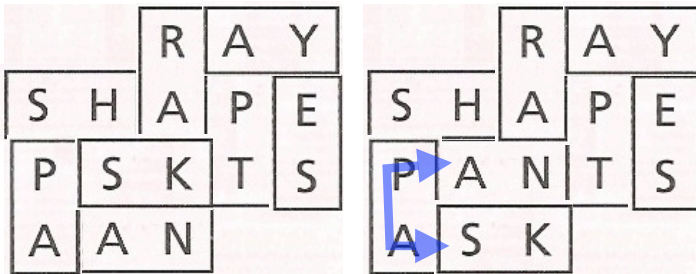
- |                  |                   |                  |
|------------------|-------------------|------------------|
| • across:        | • across: ran     | • across: ray    |
| • down: apt, spa | • down:           | • down: apt, yes |
| • Score = 2      | • Score = 1       | • Score = 3      |
|                  | • Worse (reject). | • Improvement    |

# Next Two Rounds



- |                  |                           |                       |
|------------------|---------------------------|-----------------------|
| • across: ray    | • across: ray, shape, tan | • across: ray, shape  |
| • down: apt, yes | • down: yes               | • down: spa, apt, yes |
| • Score = 3      | • Score = 4               | • Score = 5           |
|                  | • Improvement             | • Improvement         |

# Last Round



- across: ray, shape
- down: spa, apt, yes
- Score = 5
- across: ray, shape, pants, ask
- down: spa, has, rank, apt, yes
- Score = 9 (solved)

# Local Search

- Example of local search.
- Start with a partial solution.
- Heuristic says how good the current solution is.
- Consider “local changes” to improve the solution.
- “Hillclimb” to a better solution.

# Hillclimbing Code

```
def hillClimb(solution):
    stuck = false
    while not stuck:
        v = value(solution)
        newsolution = stepUp(solution)
        if newsolution == solution: stuck = true
        else: solution = newsolution
    print solution, "is a local maximum"
```

# Hill Climbing Observations

- Can get stuck.
- Sometimes must “undo” something that is correct to make progress (*apt* and *spa* left, then returned).
- Random restarts can help.
- Can be a long string of improvements (not polynomial even to find a local maximum).
- But, at least it doesn't look stupid!

# Next Time

- Computability.
- Finish Hillis, Chapter 4.