Adversarial Search

Toolbox so far

- Uninformed search
  - BFS, DFS, uniform cost search
- Heuristic search
  - A*

Common environmental factors:
static, discrete, fully observable, deterministic actions.
Also: single agent, non-episodic.

There's More!

- Add a second agent, but not controlled by us.
- Assume this agent is our adversary.
- Environment (for now)
  - Still static
  - Still discrete
  - Still fully observable (for now)
  - Still deterministic (for now)

Games!

- Deterministic, turn-taking, two-player, zero-sum games of perfect information.

2007

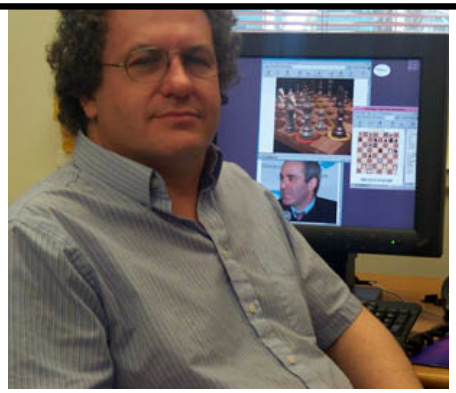
Checkers Is Solved

Jonathan Schaeffer,* Neil Burch, Yngvi Björnsson,† Akihiro Kishimoto,‡
Martin Müller, Robert Lake, Paul Lu, Steve Sutphen

The game of checkers has roughly 500 billion billion possible positions (5×10^{20}). The task of solving the game, determining the final result in a game with no mistakes made by either player, is daunting. Since 1989, almost continuously, dozens of computers have been working on solving checkers, applying state-of-the-art artificial intelligence techniques to the proving process. This paper announces that checkers is now solved: Perfect play by both sides leads to a draw. This is the most challenging popular game to be solved to date, roughly one million times as complex as Connect Four. Artificial intelligence technology has been used to generate strong heuristic-based game-playing programs, such as Deep Blue for chess. Solving a game takes this to the next level by replacing the heuristics with perfection.

best known is the four-color theorem (*9*). This deceptively simple conjecture—that given an arbitrary map with countries, you need at most four different colors to guarantee that no two adjoining countries have the same color—has been extremely difficult to prove analytically. In 1976, a computational proof was demonstrated. Despite the convincing result, some mathematicians were skeptical, distrusting proofs that had not been verified using human-derived theorems. Although important components of the checkers

The game of checkers has roughly 500 billion billion possible positions (5×10^{20}). The task of solving the game, determining the final result in a game with no mistakes made by either player, is daunting. Since 1989, almost continuously, dozens of computers have been working on solving checkers, applying state-of-the-art artificial intelligence techniques to the proving process. This paper announces that checkers is now solved: Perfect play by both sides leads to a draw. This is the most challenging popular game to be solved to date, roughly one million times as complex as Connect Four. Artificial intelligence technology has been used to generate strong heuristic-based game-playing programs, such as Deep Blue for chess. Solving a game takes this to the next level by replacing the heuristics with perfection.



UNIVERSITY OF ALBERTA
EDMONTON, ALBERTA, CANADA

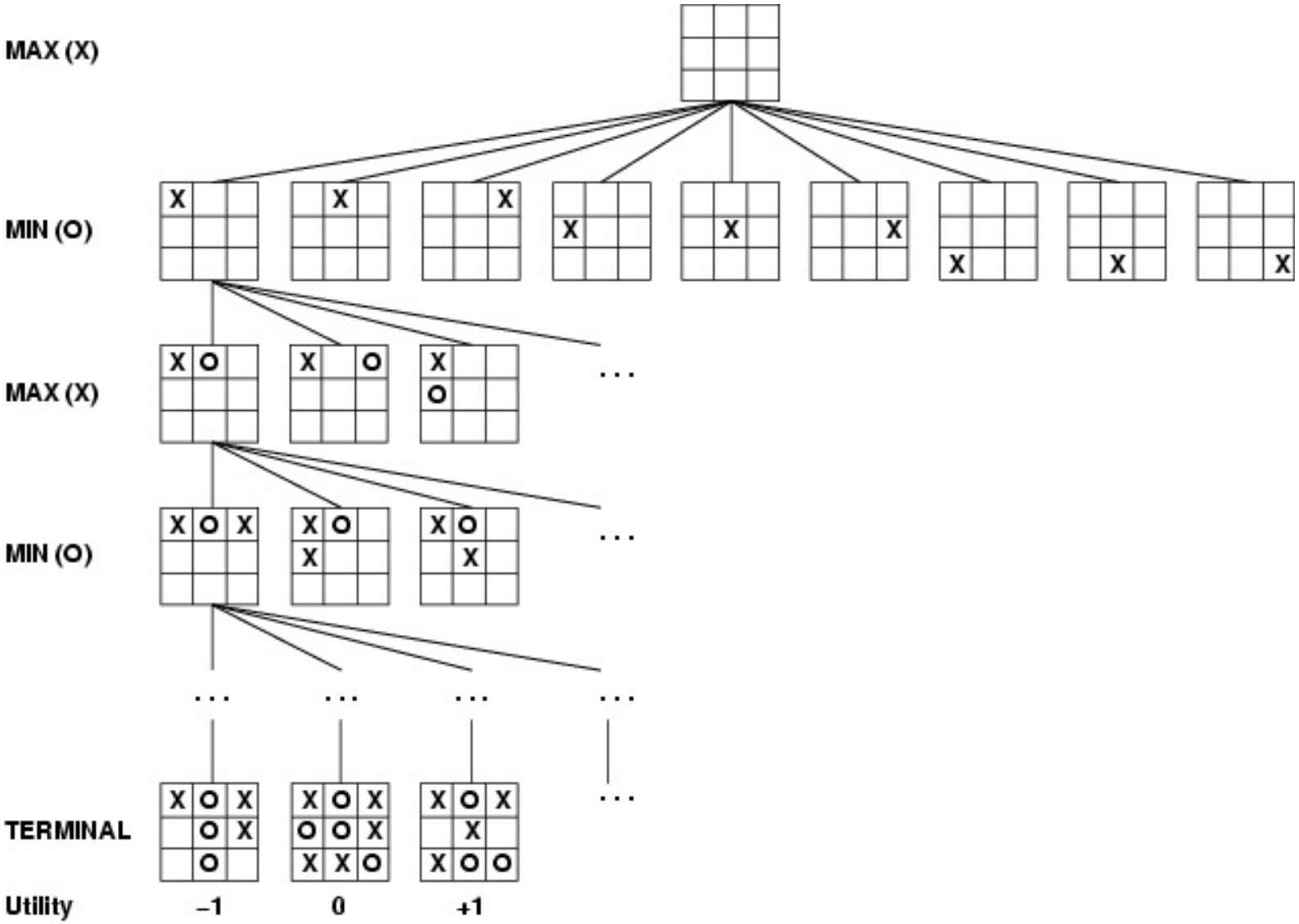
DEPARTMENT OF COMPUTING SCIENCE

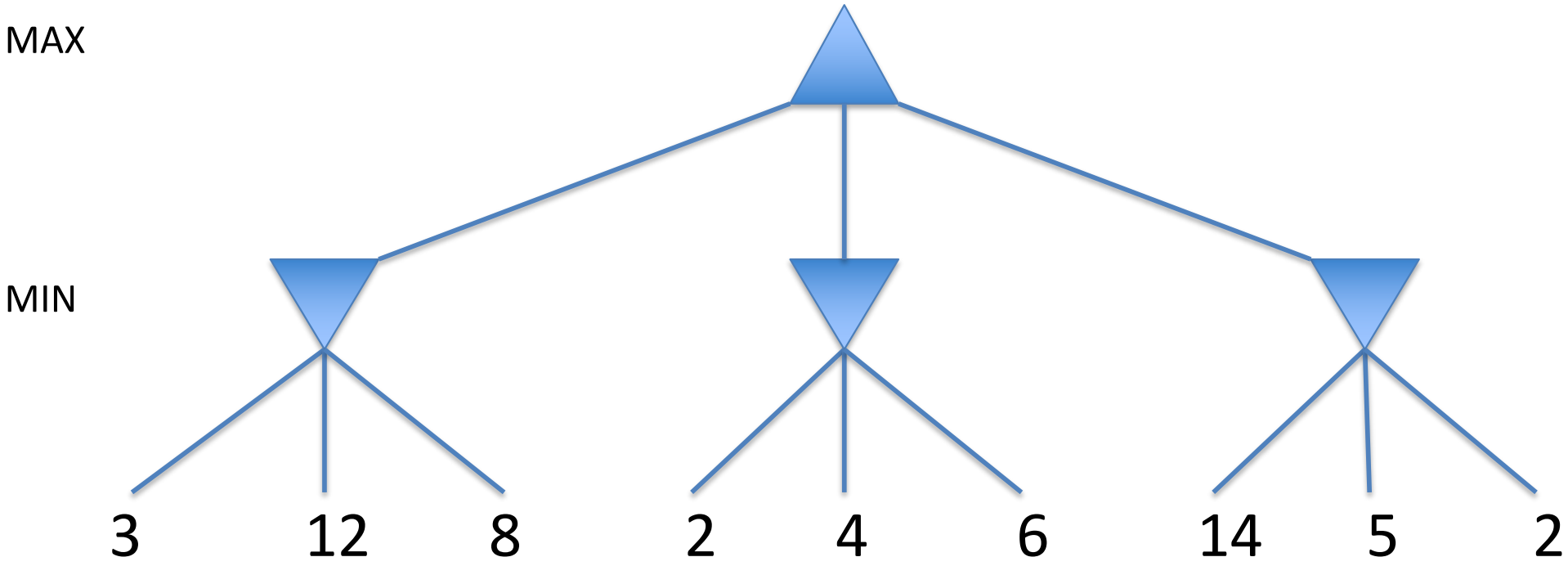
Adversarial search

- Still search!
  - But another agent will alternate actions with us.
- Main new concept:
  - Two players are called MAX and MIN.
  - Only works for zero-sum games.
    - Strictly competitive (no cooperation).
    - What is good for me is equally bad for my opponent (in regards to winning and losing).
  - Most “normal” 2-player games are zero-sum.

- Most all of our concepts from state-space search transfer here.
- S_0: initial state
- TO-MOVE(s): Defines who makes the next move at a state.
- ACTIONS(s): Returns the set of legal moves in a state.
- RESULT(s, a): Returns what state you go into (*transition model*)
- IS-TERMINAL(s): Returns true if s is a *terminal state*.
- UTILITY(s, p): Numeric value of a terminal state s for player p.

Game Tree




MAX
MIN
3
12
8
2
4
6
14
5
2

Minimax algorithm

- Select the best move for you, assuming your opponent is selecting the best move for themselves.
- Works like DFS.

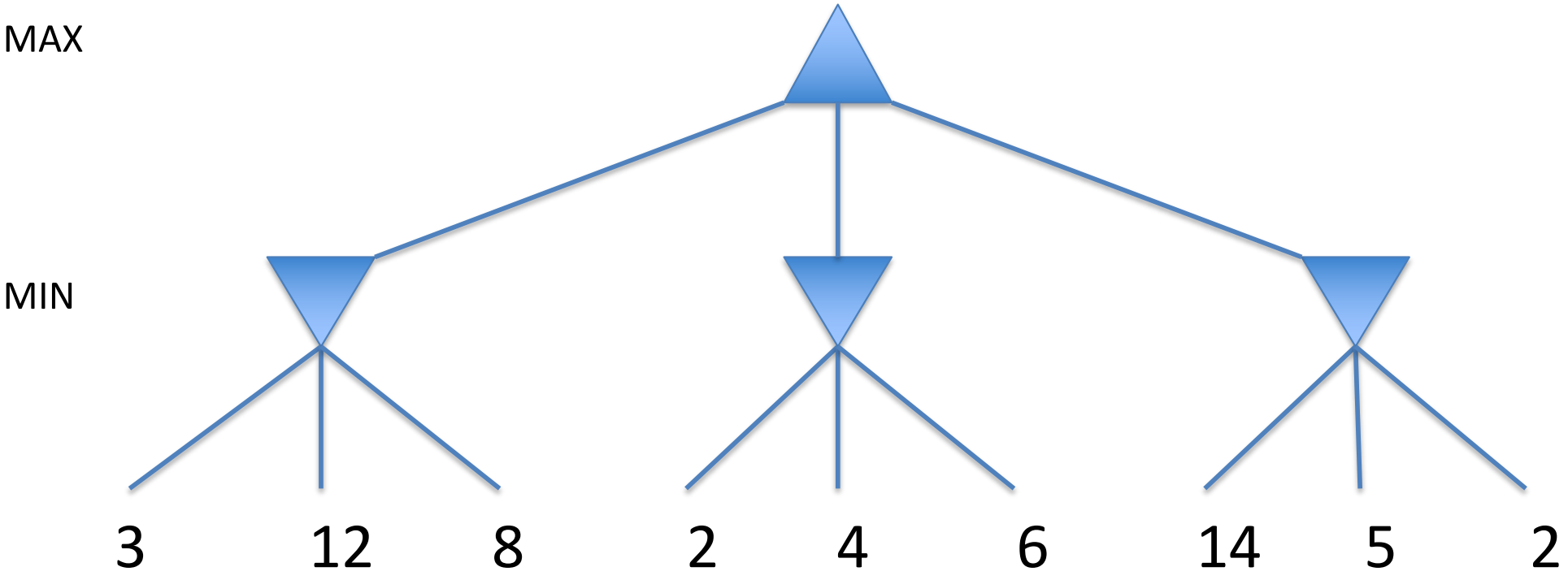
Minimax algorithm

(assuming it is MAX's turn)

minimax(s) =

utility(s, MAX)	if IS-TERMINAL(s)
$\max_{a \text{ in actions(s)}}$ minimax(result(s, *a*))	if TO-MOVE(s)=MAX
$\min_{a \text{ in actions(s)}}$ minimax(result(s, *a*))	if TO-MOVE(s)=MIN

result(s, a) means the new state generated
by taking action *a* in state *s*.



minimax(s) =

utility(s, MAX)	if IS-TERMINAL(s)
$\max_{a \text{ in actions(s)}}$ minimax(result(s, *a*))	if TO-MOVE(s)=MAX
$\min_{a \text{ in actions(s)}}$ minimax(result(s, *a*))	if TO-MOVE(s)=MIN

Properties of minimax

- Complete?
  - Yes (assuming tree is finite)
- Optimal?
  - Yes (assuming opponent is also optimal)
- Time complexity: $O(b^m)$
- Space complexity: O(bm) (like DFS)
- But for chess, b ≈ 35, m ≈ 100, so this time is completely infeasible!

Real-World Minimax

- The minimax algorithm given here only stores the utility values; "real-world" minimax should store utility values *and* the move that gives you the value.
- This is usually done by keeping an auxiliary data structure called a transposition table; this table also cuts down on search time.
  - Table stores, for every state, the minimax value and corresponding best move.

Nim

Nim

- How to represent a state?
- How to represent an action?