Algorithmic Game Theory and Applications

Lecture 11: Games of Perfect Information & Games on Graphs

Kousha Etessami

finite games of perfect information

A *perfect information* (PI) game: 1 node per information set.

Theorem([Kuhn'53]) Every finite n-person extensive PI-game, $\mathcal{G}$, has a pure subgame-perfect NE (SPNE), s^*.

For proving this, we need some definitions. For a game $\mathcal{G}$ with game tree T, and for $w \in T$, define the **subtree** $T_w \subseteq T$, by: $T_w = \{w' \in T \mid w' = ww'' \text{ for } w'' \in \Sigma^*\}$.
Since tree is finite, we can just associate payoffs to the leaves. Thus, the subtee T_w, in an obvious way, defines a "**subgame**", $\mathcal{G}_w$, which is also a PI-game.
The **depth** of a node w in T is its length $|w|$ as a string. The depth of tree T is the maximum depth of any node in T. The depth of a game $\mathcal{G}$ is the depth of its game tree.

Proof of Kuhn's theorem: "backward induction" algorithm

The proof provides a bottom-up "*backward induction*" algorithm for computing a pure SPNE in a finite PI-game:
We inductively "attach" to the root of every subtree T_w, a SPNE s^w for the subgame $\mathcal{G}_w$, together with the expected payoff vector $h^w := (h_1^w(s^w), \ldots, h_n^w(s^w))$.

1. Initially: Attach to each **leaf** w the empty profile $s^w = (\emptyset, \ldots, \emptyset)$, & payoff vector $h^w := (u_1(w), \ldots, u_n(w))$.
2. **While** ($\exists$ unattached node w whose children are attached)
   - if ($w \in Pl_0$) then
     $s^w := (s_1^w, \ldots, s_n^w)$, where $s_i^w := \bigcup_{a \in Act(w)} s_i^{wa}$;
     hence h^w is: $h_i^w(s^w) := \sum_{a \in Act(w)} q_w(a) * h_i^{wa}(s^{wa})$;

     else if ($w \in Pl_i$ & $i > 0$) then
     Let $s^w := (s_1^w, \ldots, s_n^w)$, & $h^w := h^{wa'}$, where
     $a' := \arg\max_{a \in Act(w)} h_i^{wa}(s^{wa})$,
     $s_{i'}^w := \bigcup_{a \in Act(w)} s_{i'}^{wa}$, for $i' \neq i$, and
     $s_i^w := (\bigcup_{a \in Act(w)} s_i^{wa}) \bigcup \{w \mapsto a'\}$;

proof of Kuhn's theorem (backward induction)

We can turn the "backward induction" algorithm into a proof by induction on the depth of a subgame $\mathcal{G}_w$ that it has a pure SPNE, $s^w = (s_1^w, \ldots, s_n^w)$. Then $s^* := s^\epsilon$ is a SPNE for $\mathcal{G}$.

Base case, depth 0: In this case we are at a leaf w. there is nothing to show: each player i gets payoff $u_i(w)$, and the strategies in the SPNE s^* are "empty" (it doesn't matter which player's node w is, since there are no actions to take.)

Inductive step: Suppose depth of $\mathcal{G}_w$ is $k+1$. Let $Act(w) = \{a'_1, \ldots, a'_r\}$ be the set of actions available at the root of $\mathcal{G}_w$. The subtrees $T_{wa'_j}$, for $j = 1, \ldots, r$, each define a PI-subgame $\mathcal{G}_{wa'_j}$, of depth $\leq k$.
Thus, by induction, each game $\mathcal{G}_{wa'_j}$ has a pure strategy SPNE,
$s^{wa'_j} = (s_1^{wa'_j}, \ldots, s_n^{wa'_j})$.
To define $s^w = (s_1^w, \ldots, s_n^w)$, there are two cases to consider
......

two cases

1. $w \in Pl_0$, i.e., the root node, w, of T_w is a chance node (belongs to "nature").
Let the strategy s_i^w for player i be just the obvious "union" $\bigcup_{a' \in Act(w)} s_i^{wa'}$, of its pure strategies in each of the subgames.
(Explanation of "union" of disjoint strategy functions.)
Claim: $s^w = (s_1^w, \ldots, s_n^w)$ is a pure SPNE of $\mathcal{G}_w$. Suppose not.
Then some player i could improve its expected payoff by switching to a different pure strategy in one of the subgames.
But that violates the inductive hypothesis on that subgame.

2. $w \in Pl_i$, $i > 0$: the root, w, of T_w belongs to player i. For $a \in Act(w)$, let $h_i^{wa}(s^{wa})$ be the expected payoff to player i in the subgame $\mathcal{G}_{wa}$. Let $a' = \arg\max_{a \in Act(w)} h_i^{wa}(s^{wa})$. For players $i' \neq i$, define $s_{i'}^w = \bigcup_{a \in Act(w)} s_{i'}^{wa}$.
For i, define $s_i^w = (\bigcup_{a \in Act(w)} s_i^{wa}) \cup \{w \mapsto a'\}$.
Claim: $s^w = (s_1^w, \ldots, s_n^w)$ is a pure SPNE of $\mathcal{G}_w$. □

Computing a SPNE for a general EFGs

We can use the same idea of Kuhn's backward induction algorithm to compute a SPNE in behavior strategies (not necessarily a pure one) for *any* finite extensive form game of perfect recall (not just PI games). Basic idea:

Repeat

> compute a NE for a "*bottom-most*" subgame in the game tree. (If necessary, convert that subgame to normal form in order to compute a (possibly mixed) NE for it.) Compute the expected payoffs for each player in that NE for that subgame. Update the game tree by removing that subgame and replacing it with a "leaf" with those payoffs.

Until remaining game tree is trivial (only the root remains).

consequences for zero-sum finite PI-games and Chess

Recall that, by the Minimax Theorem, for every finite zero-sum game Γ, there is a value v^* such that for any NE (x_1^*, x_2^*) of Γ, $v^* = U(x_1^*, x_2^*)$, and

$$\max_{x_1 \in X_1} \min_{x_2 \in X_2} U(x_1, x_2) = v^* = \min_{x_2 \in X_2} \max_{x_1 \in X_1} U(x_1, x_2)$$

But it follows from Kuhn's theorem that for extensive PI-games $\mathcal{G}$ there is in fact a pure NE (in fact, SPNE) (s_1^*, s_2^*) such that $v^* = u(s_1^*, s_2^*) := h(s_1^*, s_2^*)$, and thus that

$$\max_{s_1 \in S_1} \min_{s_2 \in S_2} u(s_1, s_2) = v^* = \min_{s_2 \in S_2} \max_{s_1 \in S_1} u(s_1, s_2) \qquad (1)$$

A finite zero-sum PI-game is called **determined** if (1) holds.[1]

Proposition ([Zermelo'1912]) Every finite zero-sum PI-game is determined, and the game's value, v^*, and pure minimax profile s^*, can be computed "efficiently" given $\mathcal{G}$'s game three.

[1] Note: an *infinite* zero-sum PI-game is called **determined** if $\sup_{s_1 \in S_1} \inf_{s_2 \in S_2} u(s_1, s_2) = v^* = \inf_{s_2 \in S_2} \sup_{s_1 \in S_1} u(s_1, s_2)$.

chess

Chess as a finite PI-game: after 50 moves with no piece taken, it ends in a draw. It's a **win-lose-draw** PI-game: no chance nodes, and only possible payoffs are 1, -1, and 0.

Proposition([Zermelo'1912]) In Chess, either:

1. White has a "winning strategy", or
2. Black has a "winning strategy", or
3. Both players have strategies to force a draw.

A "**winning strategy**", e.g., for White (Player 1) is a pure strategy s_1^* that guarantees value $u(s_1^*, s_2) = 1$, for all s_2.

chess

Chess as a finite PI-game: after 50 moves with no piece taken, it ends in a draw. It's a **win-lose-draw** PI-game: no chance nodes, and only possible payoffs are 1, -1, and 0.

Proposition([Zermelo'1912]) In Chess, either:

1. White has a "winning strategy", or
2. Black has a "winning strategy", or
3. Both players have strategies to force a draw.

A "**winning strategy**", e.g., for White (Player 1) is a pure strategy s_1^* that guarantees value $u(s_1^*, s_2) = 1$, for all s_2.

Question: Which of (1.), (2.), or (3.) is the correct answer??

chess

Chess as a finite PI-game: after 50 moves with no piece taken, it ends in a draw. It's a **win-lose-draw** PI-game: no chance nodes, and only possible payoffs are 1, -1, and 0.

Proposition([Zermelo'1912]) In Chess, either:

1. White has a "winning strategy", or
2. Black has a "winning strategy", or
3. Both players have strategies to force a draw.

A "**winning strategy**", e.g., for White (Player 1) is a pure strategy s_1^* that guarantees value $u(s_1^*, s_2) = 1$, for all s_2.

Question: Which of (1.), (2.), or (3.) is the correct answer??

We still don't know! **Problem:** The tree is far too big!!

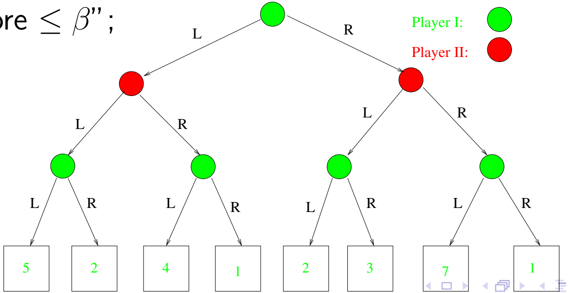
Even with ~ 200 depth & ~ 5 moves per node:

$$5^{200} \text{ nodes!}$$

Despite having an "efficient" algorithm to compute the value v^* given the tree, we can't even look at the whole tree! We need algorithms that don't look at the whole tree.

70 years of game-tree search

There's $>$ 70 years of research on chess & other game playing programs, (Shannon, Turing, . . .). Heuristic game-tree search is now very refined. See any AI text (e.g., [Russel-Norvig]). If we have a function *Eval*(w) that heuristically "evaluates" a node's "goodness" score, we can use *Eval*(w) to stop the search at, e.g., desired depth. While searching "top-down", we can "prune out" irrelevant subtrees using α-β-**pruning**. Idea: while searching minmax tree, maintain two values: α- "maximizer can assure score $\geq \alpha$"; & β- "minimizer can assure score $\leq \beta$";



minmax search with α-β-pruning

Assume, for simplicity, that players alternate moves, root belongs to Player 1 (maximizer), and $-1 \leq Eval(w) \leq +1$. Score -1 ($+1$) means player 1 definitely loses (wins). Start the search by calling: **MaxVal**$(\epsilon, -1, +1)$;

MaxVal(w, α, β)

If $depth(w) \geq MaxDepth$ then **return** $Eval(w)$.

Else, for each $a \in Act(w)$

$\alpha :=$ max$\{\alpha,$ **MinVal**$(wa, \alpha, \beta)\}$;

if $\alpha \geq \beta$, then **return** β

return α

MinVal(w, α, β)

If $depth(w) \geq MaxDepth$, then **return** $Eval(w)$.

Else, for each $a \in Act(w)$

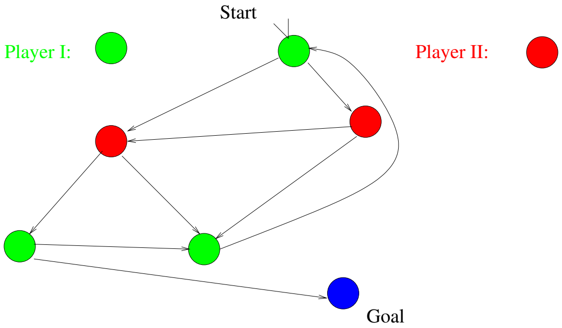
$\beta :=$ min$\{\beta,$ **MaxVal**$(wa, \alpha, \beta)\}$;

if $\beta \leq \alpha$, then **return** α

return β

Taster: infinite zero-sum PI-games & games on graphs

Instead of a tree, suppose we have a finite directed graph:



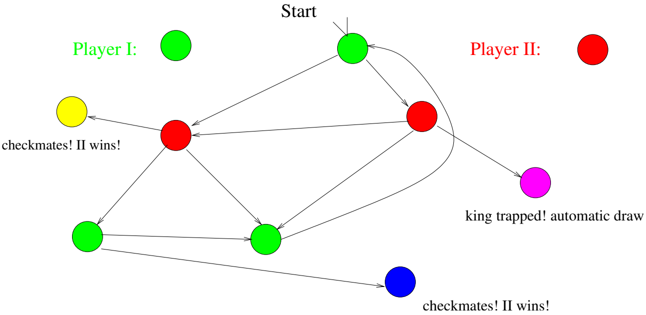
▷ Starting at "Start", does Player I have a strategy to "force" the play to reach the "Goal"?

▷ Note: this is a possibly *infinite* win-lose PI-game.

▷ Is this game determined for all finite graphs?

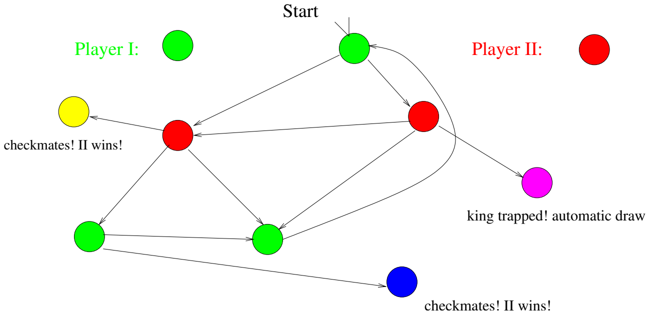
▷ If so, how do we compute a winning strategy for Player 1?

one motivation: unbounded chess



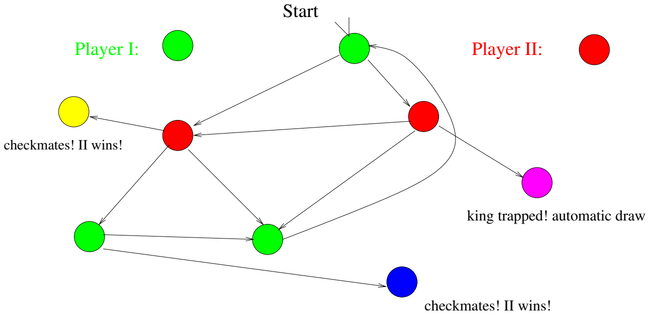
▷ Chess: the same "position/configuration" might recur in the game, but the (infinite) "game tree" does not reflect this.

one motivation: unbounded chess



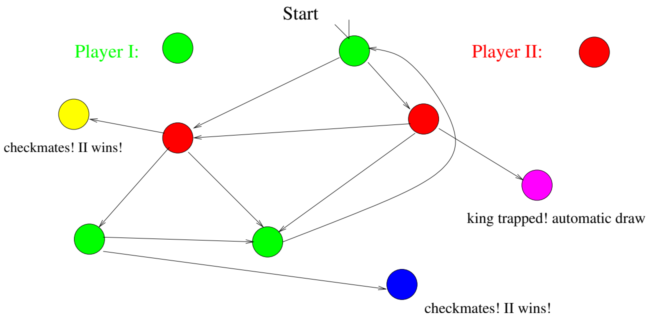
▷ Chess: the same "position/configuration" might recur in the game, but the (infinite) "game tree" does not reflect this.
▷ There are finitely many positions ($\leq 64^{32}$). After some depth, every "play" contains recurrences of positions.

one motivation: unbounded chess



▷ Chess: the same "position/configuration" might recur in the game, but the (infinite) "game tree" does not reflect this.
▷ There are finitely many positions ($\leq 64^{32}$). After some depth, every "play" contains recurrences of positions.
▷ Consider "unbounded chess" without artificial stopping conditions: an infinite play is by definition a draw.

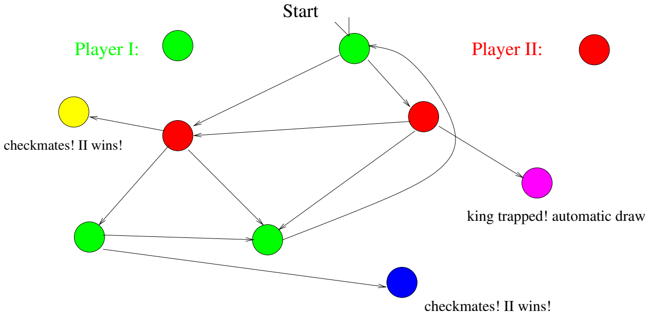
one motivation: unbounded chess



▷ Chess: the same "position/configuration" might recur in the game, but the (infinite) "game tree" does not reflect this.
▷ There are finitely many positions ($\leq 64^{32}$). After some depth, every "play" contains recurrences of positions.
▷ Consider "unbounded chess" without artificial stopping conditions: an infinite play is by definition a draw.

Is this win-lose-draw game determined? I.e., does Zermelo's theorem still hold?

one motivation: unbounded chess



▷ Chess: the same "position/configuration" might recur in the game, but the (infinite) "game tree" does not reflect this.
▷ There are finitely many positions ($\leq 64^{32}$). After some depth, every "play" contains recurrences of positions.
▷ Consider "unbounded chess" without artificial stopping conditions: an infinite play is by definition a draw.

Is this win-lose-draw game determined? I.e., does Zermelo's theorem still hold? Yes!

the "reachability" game: easy algorithm

Consider the "reachability" win-lose game: player 1 wants to reach a "goal" vertex, player 2 wants to avoid it. Algorithm to compute who has a winning strategy from each node:

Input: Game graph $G = (V, V_1, V_2, E, pl, v_0)$.

$Bad := \{v \in V \mid v$ a dead end that's winning for player 2$\}$.

1. Initialize: $Win_1 := \{\text{"goal"}\}$; $St_1 := \emptyset$;
2. **Repeat**

   Foreach $v \in V \setminus (Win_1 \cup Bad)$:

   If $(v \in V_1 \;\; \& \;\; \exists\,(v, v') \in E : \; v' \in Win_1)$

   $Win_1 := Win_1 \cup \{v\}$; $St_1 := St_1 \cup \{v \mapsto v'\}$;

   If $(v \in V_2 \;\; \& \;\; \forall\,(v, v') \in E : \; v' \in Win_1)$

   $Win_1 := Win_1 \cup \{v\}$;

   Until The set Win_1 does not change;

Fact: player 1 has a winning strategy iff $v_0 \in Win_1$ when the algorithm halts. If so, St_1 is a (memoryless) winning strategy for player 1.

generalizing to unbounded chess

The generalization is not hard: unbounded chess is a win-lose-draw game, with 3 distinct possible payoffs, -1, 0, or 1. The payoff 0 (a "draw") is for all infinite plays or plays that end a "stalemate" node. Consider the "reachability" game where player 1 wins if it attains payoff 1 (reaches its "checkmate" node), and loses if its payoff is any less. Use the "reachability" game algorithm on this game to find a strategy for player 1 that is winning from all vertices in Win_1 where payoff 1 terminal nodes can be reached. We can then eliminate Win_1 vertices and the payoff 1 nodes. We get a new game, with payoffs -1 and 0 only. In this new game, we again use the "reachability" game algorithm, but this time from the point of view of player 2, to determine from which nodes player 2 has a "winning strategy" to reach the terminal node labeled -1 ("checkmate" for player 2).

This was only a taster of a vast topic

- Infinite-horizon PI-games, and games on graphs, are a vast topic, and we've only scratched their surface.
- What if, in a game graph, instead of 2 players, there are player 1 nodes as well as "chance/nature" nodes? This amounts to what's called a *Markov Decision Process*.
- What if the game graph has 2 (or more) players as well as chance/nature nodes? This amounts to what's called a *Stochastic Game*.
- Each of these is a vast and rich topic that we don't have time to cover.