COMPSCI 575/MATH 513

Combinatorics and Graph Theory

Lecture #32: Progressively Finite Games
(Tucker Section 10.1)
David Mix Barrington
5 December 2016

Progressively Finite Games

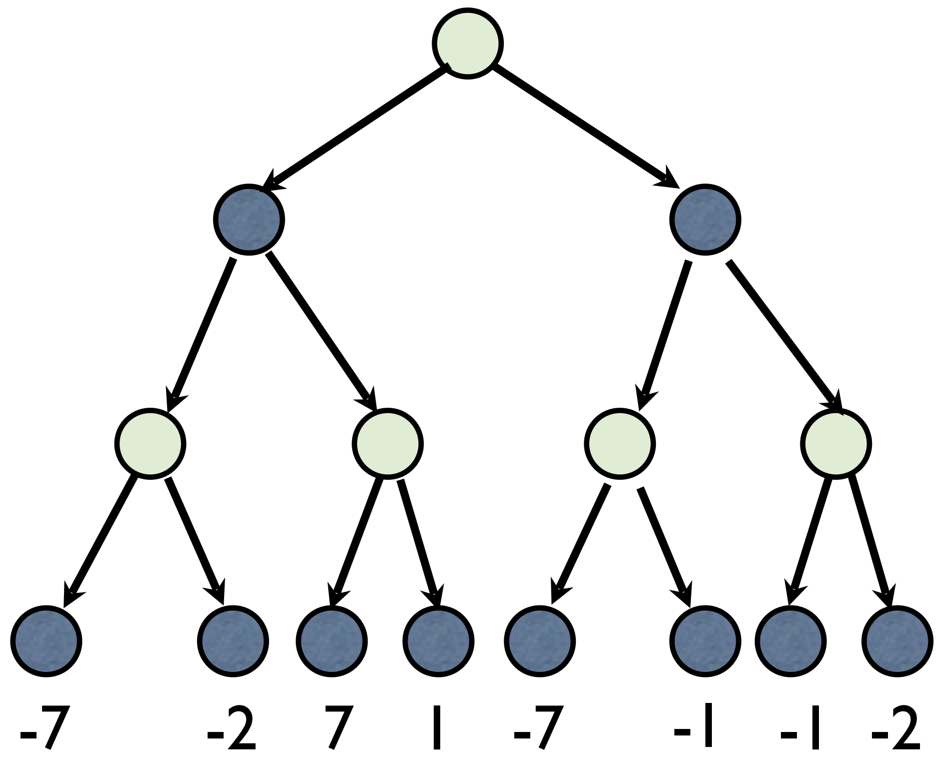
- Games We Are and Aren't Studying
- Examples: 1-4 Takeaway, Adding Coins
- Winning Strategies
- Winning and Losing Positions
- A Theorem on Winning Strategies
- The Grundy Function

How to Define a Game?

- In this last section of the course we look at one of the several branches of mathematics called **game theory**.
- Here we will look at games that have **two players**, are **deterministic**, **discrete**, **zero-sum**, and have **perfect information**.
- Lots of interesting games, of course, fail to have one or more of these properties.

Game Trees

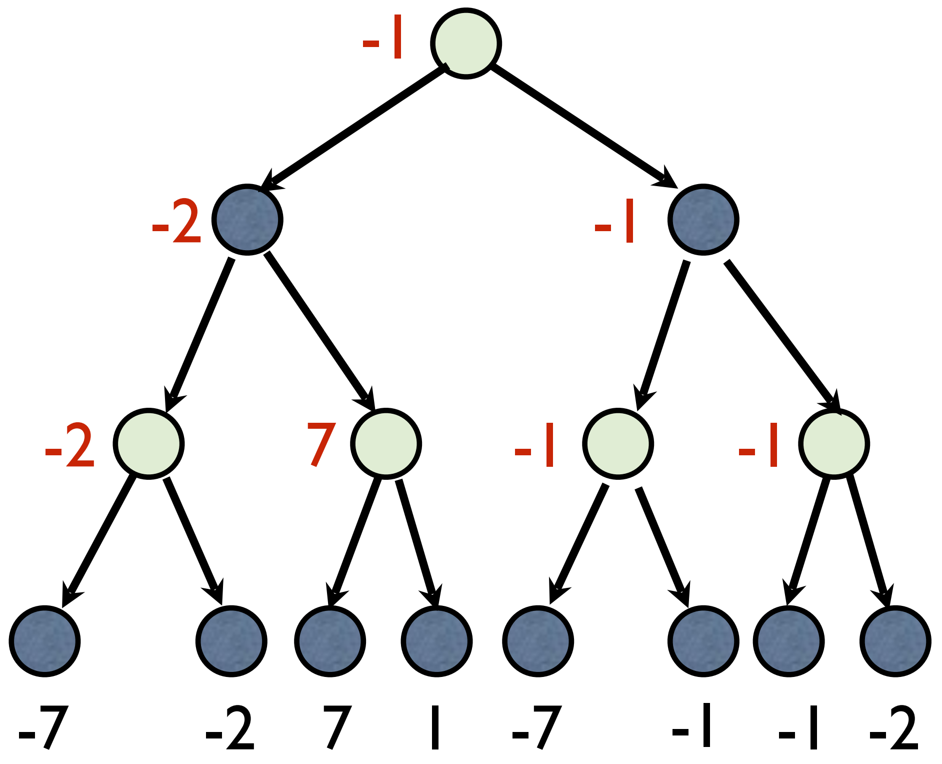
- In COMPSCI 250 we modeled games like this with **game trees**.
- Each node represents a position, and is labeled with the player to move, or with the payoff if it is a leaf.
- We actually need not alternate moves as we do here.



Evaluating Game Trees

- In COMPSCI 250 we proved that every node in such a tree has a **value**, and each player has a strategy that in the worst case will achieve that value for them.

- The value of a White node is the max of its childrens' values, and that of a Black node is the minimum.



Adversary Search

- So by evaluating the entire game tree, we could play the game optimally. Unfortunately, the tree is normally exponentially sized in the number of moves and it’s not feasible to do this.
- Actual programs to play games like Chess use an **evaluation function** on positions, then truncate the tree and play a game whose goal is to get the best possible position some number of moves from now.

Progressively Finite Games

- Tucker defines a class of games where the total number of positions is finite and there is a finite limit on the possible number of moves.
- His games are non-partisan, in that the available moves for either player are the same from any position. Equivalently, "whose move it is" is part of the position.
- He represents his games by directed acyclic graphs where nodes are positions and edges are moves. (A cycle could allow an infinite game.)

Adapting Chess to This Format

- Chess is perhaps the most familiar game that is deterministic, discrete, zero-sum, and has perfect information. But there is no finite limit on the number of moves. How can we fix this?
- Under the rules, either player may claim a draw by proving that the identical board position has occurred for a third time, or that fifty moves have passed since the last time a piece was captured or a pawn was moved.

Adapting Chess

- Suppose we make draws into wins for Black. Then Black wants to claim a draw whenever these situations occur, so we can make them automatic.
- But the "state" of the game has to include enough information about the history to enforce these rules.
- With these modifications, Chess has a finite game tree. So there exists either a winning strategy for White or a drawing for Black.

Games in Complexity

- Suppose I have a directed acyclic graph G, a start node s, and a goal node t.
- We can play a game where White starts a series of alternating moves along edges, and wins if and only the path so defined reaches t.
- This **alternating reachability** problem is "complete for the class P" in the language of COMPSCI 501.

Games in Complexity

- As we'll see later in this lecture, we can determine the winner (under optimal play) of such a game in polynomial time.
- Using the Cook-Levin construction from 501, we can convert *any* polynomial-time problem into an alternating reachability problem on a polynomial-size graph.
- But "polynomial" here is defined in terms of the *number of possible positions*, which is prohibitively big for most families of games.

Two Game Examples

- Let's now look at Tucker's examples of games.
- In the first, we start with a pile of stones, a legal move is to remove 1, 2, 3, or 4 stones, and the winner is the one takes the last stone.
- In the second, players add coins to a pile (1, 2, or five on a turn) until someone wins by making the total either n^2 for $n > 1$ or > 40.
- Each game is represented by a directed acyclic graph, which I'll sketch on the board.

Winning Strategies

- What do we mean by a winning strategy? A **strategy** is an assignment of a move to each position. A strategy is **winning** if playing it against *any* opponent's strategy leads to a win.
- We can ignore positions that are not reachable by the player in the course of a game in which the strategy is being used.
- The strategy is a finite object but could be huge.

Winning and Losing Positions

- In 1-4 Takeaway, 0 is a **winning position** because if you can move to 0, your opponent cannot move and she loses.
- This makes 1, 2, 3, and 4 **losing positions** because if you move to one of them your opponent will take all the stones and win.
- But 5 then turns out to be winning, and by induction you can show that 5k is winning and 5k+1, 5k+2, 5k+3, and 5k+4 are losing for all non-negative integers k.

Winning and Losing Positions

- In the second game, the rules define 4, 9, 16, 25, 36, and 41 to be winning positions.
- So any position with a direct move to one of these (2, 3, 7, 8, 11, 14, 15, 20, 23, 24, 31, 34, 35, 37, 38, 39, 40) is a losing position. (4 would be losing if it weren't already winning).
- Now look at 6. A player at 6 has no choice but to move to a losing position (7, 8, or 11). So moving to 6 *wins* the game and 6 is a winning position. This makes 1 and 5 losing positions, so that 0 is a winning position.

Theorem on Winning Strategies

- Something similar happens in any game defined by a graph in this way.
- Theorem: In any PF game, there is a unique winning strategy for one player or the other, consisting of a "kernel" of winning positions, and we can compute this kernel.
- Formally, a kernel is a set of nodes K such that there are no edges from K to K, and every non-K node has an edge into K.

Theorem on Winning Strategies

- The strategy is always to move to a kernel node if you are at a non-kernel node. If you are at a kernel node, you are doomed anyway under optimal play by your opponent, so it doesn't matter what you do.
- If you keep moving to kernel nodes, your opponent must eventually have no move.
- So if the start node is in the kernel, Black has a winning strategy, and if it is not, White does.

Proof of the Theorem

- We use induction on the number of nodes.
- If there is one node, we make it the kernel.
- Now if every n-node graph has a kernel, and we have an n+1 node graph G, we find a node x of in-degree 0 and find the kernel of G - {x} by the IH. If x has an edge to a kernel node, we add it to the kernel of G, otherwise we do not. We now have a legal kernel for G.

Constructing a Kernel

- Given any directed acyclic graph, we can construct a kernel more directly than in this proof.
- Define Level 0 to be the nodes that are **sinks**,with out-degree 0. These nodes are in the kernel.
- Level 1 is the nodes not in Level 0 whose successors are all in Level 0. Similarly, Level n+1 is the nodes not in any Level 0-n whose successors are all in Levels 0-n.

Constructing a Kernel

- So the level of a node is the length of its *longest* path to a sink.
- Level 0 nodes are all winning (in the kernel) and Level 1 nodes are all losing (not in it).
- A Level 2 node is losing if it has an edge to a Level 0 node, and winning otherwise.
- In general a Level n+1 node is losing if it has an edge to a winning node in Levels 0-n, and winning otherwise. In this way we label all the nodes.

Extending the Theorem

- The theorem holds for infinite games as well, as long as they are guaranteed to end.
- For example, Iconsider 1-4 Takeaway where White's first move is to name any integer as the initial pile size. The game always ends but there is no limit on the length, and there are infinitely many states.
- As long as no state has an infinite path out of it, we can define level and complete the proof, though the computation may no longer be finite.

The Grundy Function

- The **Grundy function** on a directed graph (with no infinite paths) assigns a non-negative integer g(x) to every node x.
- If x is a sink, g(x) is 0. Otherwise g(x) is the smallest number that is not equal to g(y) for any y that is a successor of x.
- We can define this by induction on levels, and it is well-defined if every node has a finite level.
- The unique kernel is just {x: f(x) = 0}.