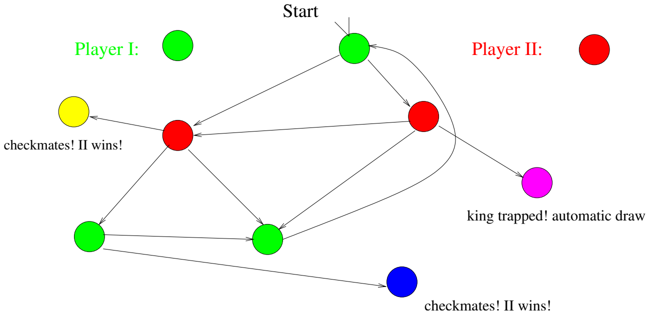
Algorithmic Game Theory and Applications

Lecture 12: Games on Graphs

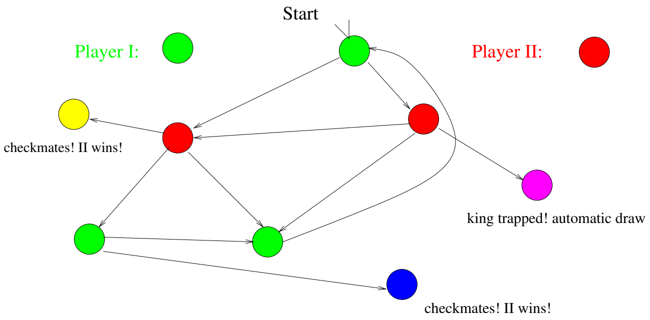
Kousha Etessami

unbounded chess



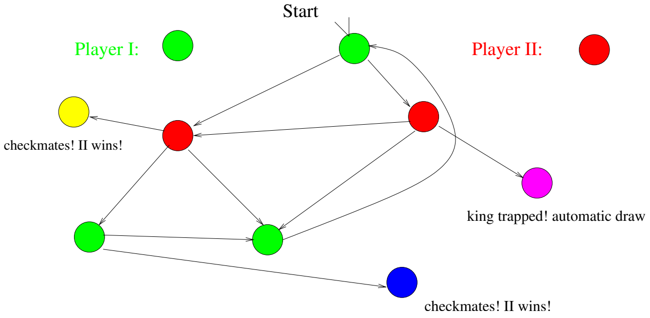
▷ Chess: the same "position/configuration" might recur in the game, but the (infinite) "game tree" does not reflect this.

unbounded chess



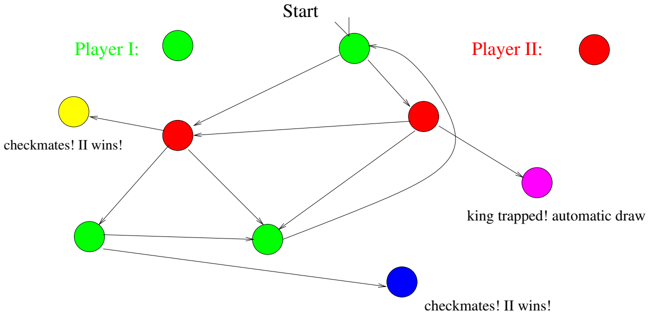
▷ Chess: the same "position/configuration" might recur in the game, but the (infinite) "game tree" does not reflect this.
▷ There are finitely many positions ($\leq 64^{32}$). After some depth, every "play" contains recurrences of positions.

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.

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?

more serious motivation

▷ We can often model the dynamics of a system (e.g., a running program) as a <u>state transition system</u>.

▷ If the system interacts with an environment, transitions out of some states can be viewed as "controlled by the environment". Can the environment force the system, with <u>some</u> sequence of inputs, into a "<u>bad state</u>"?

▷ Even for state machines without environments, certain temporal queries about the behavior of the system over time can be formulated as a game on a graph.

▷ Such queries, and much more, can be formalized in certain "<u>temporal logics</u>": formal languages for describing relationships between the occurrence of events over time. Efficiently checking such queries against a system model (e.g., a state transition system) is the task of "<u>model checking</u>". Some key model checking tasks are intimately related to efficiently solving certain games on graphs.

game graphs and their trees

A 2-player **game graph**, $G = (V, E, pl)$ consists of:

▷ A (finite) set V of vertices.

▷ A set $E \subseteq V \times V$ of edges.

▷ A partition (V_1, V_2) of the vertices $V = V_1 \cup V_2$ into two disjoint sets belonging to players 1 and 2, respectively.

game graphs and their trees

A 2-player **game graph**, $G = (V, E, pl)$ consists of:

▷ A (finite) set V of <u>vertices</u>.

▷ A set $E \subseteq V \times V$ of <u>edges</u>.

▷ A partition (V_1, V_2) of the vertices $V = V_1 \cup V_2$ into two disjoint sets belonging to players 1 and 2, respectively.

A game graph G together with a start vertex $v_0 \in V$, defines a game tree T_{v_0} given by:

▷ Action alphabet $\Sigma = V$. Thus $T_{v_0} \subseteq V^*$.

▷ $\epsilon \in T_{v_0}$, and $wv'' \in T_{v_0}$, for $v'' \in V$, if and only if

- $w = \epsilon$ and $(v_0, v'') \in E$, or
- $w = w'v'$, for some $v' \in V$, and $(v', v'') \in E$.

We extend the partition (V_1, V_2) to a partition (T'_1, T'_2) of the tree nodes of T_{v_0} as follows: if $v_0 \in V_i$, then $\epsilon \in T'_i$, and if $v' \in V_i$, then any tree node $wv' \in T_i$.

T_{v_0} is thus a game tree, where $Act(wv') = \{v'' \mid (v', v'') \in E\}$, whose plays are all paths in the graph G starting from v_0.

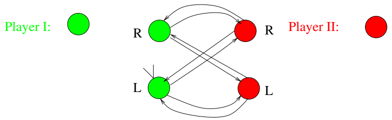
games on graphs

A **game on a graph**, $\mathcal{G}_{v_0}$, is given by:

> *A finite game graph G, vertex $v_0 \in V$, and payoff function $u : \Psi_{T_{v_0}} \mapsto \mathbb{R}$.*

These together define a 2-player zero-sum PI-game with game tree T_{v_0}.

Note: We already know that even for win-lose payoff functions u, games on finite graphs are not in general determined, because the infinite binary tree $\{L, R\}^*$ is the game tree for the following game graph:



and we already know (lecture 11) that there are sets Y of plays such that the win-lose game $\langle \{L, R\}^*, Y \rangle$ is not determined.

So, let's restrict the possible payoff functions.

"history oblivious" payoffs

▷ Suppose ∃ vertex v' of graph G that is a "dead end". E.g., in chess this could be "checkmate for Player I".

▷ There may be many ways to get to v', but the winner is the same for any finite play $wv' \in V^*$. I.e., $u(wv') = u(w'v')$, for all $wv', w'v' \in V^*$. So, the payoff is "history oblivious".

▷ What about for infinite plays π? We can think of π as an infinite sequence $v_0 v_1 v_2 v_3 v_4 v_5 \ldots\ldots$, where each $v_i \in V$. We use the notation $\pi \in V^\omega$.

▷ For $\pi = v_0 v_1 \ldots$, let

$$\inf(\pi) = \{v \in V \mid \text{for } \infty\text{-many } i \in \mathbb{N},\ v_i = v\}$$

▷ Let's call payoff function $u()$ history oblivious (**h.o.**), if for all infinite plays π & π', if $\inf(\pi) = \inf(\pi')$, then $u(\pi) = u(\pi')$, and for all finite complete plays wv and $w'v$,

$$u(wv) = u(w'v).$$

Call a graph game h.o. if its payoffs are h.o.

We will only consider h.o. games (and often less).

“finitistic” payoffs

▷ Note that in chess, if the play π is infinite, then the play is always a draw, i.e., $u(\pi) = 0$.

▷ Let's call an h.o. payoff function finitistic if for all infinite plays π and π', $u(\pi) = u(\pi')$. Let's call a game on a graph $\mathcal{G}_{v_0}$ finitistic if its payoff function is.

So, in win-lose-draw finitistic games, infinite plays are either all wins, all losses, or all draws, for player 1.

Question: Are all finitistic games on graphs determined?

Answer: Yes.............

In fact, more it true: for finitistic games there is always a pure memoryless strategy for each player that achieves the value of the game, and we can efficiently compute these strategies.

memoryless strategies and determinacy

Definition For a game $\mathcal{G}_{v_0}$, a pure strategy s_i for player i is a **memoryless strategy** if for all $wv, w'v \in Pl'_i$, $s_i(wv) = s_i(w'v)$, and if $wv_0 \in Pl'_i$ then $s_i(wv_0) = s_i(\epsilon)$.
I.e., the strategy always makes the same move from a vertex, regardless of the history of how it got there.
Let MLS_i denote the set of memoryless strategies for player i.
MLS_i is a finite set, even if S_i is not. In particular, if $m = |Pl_i|$ is the number of vertices belonging to player i, then $|MLS_i| \leq |\Sigma|^m$.
Definition $\mathcal{G}_{v_0}$ is **memorylessly determined** if both players have memoryless strategies that achieve "the value". I.e.,

$$\max_{s_1 \in MLS_1} \inf_{s_2 \in S_2} u(s_1, s_2) = \min_{s_2 \in MLS_2} \sup_{s_1 \in S_1} u(s_1, s_2)$$

Theorem A Finitistic games on finite graphs are memorylessly determined. Moreover, there is an efficient (P-time) algorithm to compute memoryless value-achieving strategies in such games.

the win-lose case: easy "fixed point" algorithm

We first prove the theorem for finitistic win-lose games via an easy "bottom up" fixed point algorithm.

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

Assume w.l.o.g. all infinite plays are win for player 2 (other case is symmetric). "Dead end": vertex with no outgoing edge. $Good := \{v \in V \mid v$ a dead end that wins for player $1\}$. $Bad := \{v \in V \mid v$ a dead end that wins for player $2\}$.

1. Initialize: $Win_1 := Good$; $St_1 := \emptyset$;
2. **Repeat**

   Foreach $v \notin Win_1$:

   If $(pl(v) = 1 \;\&\; \exists\,(v, v') \in E : \; v' \in Win_1)$

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

   If $(pl(v) = 2 \;\&\; \forall\,(v, v') \in E : \; v' \in Win_1)$

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

   Until The set Win_1 does not change;

Player 1 has a Win.-Strategy iff $v_0 \in Win_1$. If so, St_1 is a memoryless winning strategy for player 1.

why does this work?

Proof of Theorem A: (for the win-lose case)

$\triangleright$ First, we claim that for each $v \in Win_1$, St_1 is a winning strategy for player 1 in the game $\mathcal{G}_v$ (i.e., the game that starts at node v).

Suppose $v \in Win_1$. It must have entered Win_1 after, say, m iterations of the repeat loop. By induction on m, if player 1 plays according to (partial) strategy St_1, then it is guaranteed a win in the game $\mathcal{G}_v$ within m moves. Note that St_1 may be partial: it may only tell us how to move from some vertices. This won't matter.

<u>Base case:</u> $m = 0$, $v \in Good$.

<u>Inductively:</u> either v is player 1's vertex or 2's.

If it is player 1's, then $St_1(v) = v'$, where $(v, v') \in E$ and $v' \in Win_1$, and furthermore v' entered Win_1 by $m - 1$ iterations. By induction St_1 wins for player 1 from v' in $m - 1$ moves.

If v is player 2's, then we know that <u>for all</u> $(v, v') \in E$, $v' \in Win_1$, and furthermore v' entered Win_1 by $\leq m - 1$ iterations. Thus, no matter what move player 2 makes, in 1 move, by induction, we will be at a vertex $v' \in Win_1$ where player 1 wins with St_1 within $m - 1$ moves.

$\triangleright$ Now consider $v \notin Win_1$ when algorithm halts. For each $v' \in pl^{-1}(2)$, if $\exists\,(v', v'') \in E$, with $v'' \notin Win_1$, then pick one such v'', and let $St_2 := St_2 \cup \{v' \mapsto v''\}$. St_2 may also be partial.

We claim St_2 is a memoryless winning strategy for player 2 in every game $\mathcal{G}_v$, where $v \notin Win_1$. Suppose St_2 is not a winning strategy for some $v \notin Win_1$. Then player 1 must be able to win by reaching a *Good* vertex within say, m moves from v against St_2. Let's show this is a contradiction.

Base case: m$=0$, but then $v \in Good$. $\Rightarrow\Leftarrow$.

Inductively: either v is player 1's or player 2's.

If player 1's, then $\forall (v, v') \in E$, $v' \notin Win_1$, because otherwise by the algorithm $v \in Win_1$. Suppose player 1's winning strategy is to play $(v, v') \in E$. It must have a win within $m-1$ moves from $v' \notin Win_1$ against St_2. $\Rightarrow\Leftarrow$.

If it is player 2's move, then one possibility is $v \in Bad$, ($\Rightarrow\Leftarrow$).
Otherwise, $St_2(v) = v'$ must be defined: since $v \notin Win_1$, there must exist $(v, v') \in E$ with $v' \notin Win_1$. Otherwise, by the algorithm, $v \in Win_1$.

By induction, player 1 must have a $(m-1)$-winning strategy from $v' \notin Win_1$.$\Rightarrow\Leftarrow$.

generalizing to finitistic zero-sum

The generalization is not hard:
In a finitistic game, there can only be a bounded number, $r \leq |V| + 1$, of distinct payoffs $u(\pi)$,

$$j_1 < j_2 < j_3 < \ldots < j_r$$

and one of these, say j_k, is the payoff $u(\pi)$ for all infinite plays π. Suppose, w.l.o.g., that $k < r$. (If instead $1 < k$, then we work symmetrically with respect to player 2. If $1 = k = r$, then all payoffs are equal and there is nothing to do.)
Consider a new win-lose game where player 1 wins if it attains payoff j_r, and loses if its payoff is any less. Use the fixed point algorithm on this game to find a memoryless (partial) strategy for player 1 that is winning from vertices in Win_1 where payoff j_r can be obtained. We can then eliminate Win_1 vertices and the payoff j_r. We get a new finitistic zero-sum game, with payoffs $j_1 < \ldots < j_{r-1}$. Repeat!!

non-finitistic win-lose h.o. games: Muller games

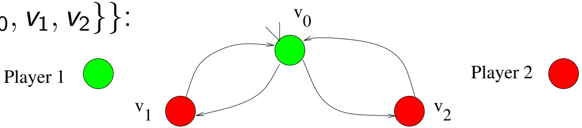
▷ We will only be interested in win-lose h.o. games.
By attaching a "self-loop" to every dead-end vertex, every play becomes infinite, and we can define the "payoffs" via a set $\mathcal{F} \subseteq 2^V$, where

$$\mathcal{F} = \{F \subseteq V \mid \text{player 1 wins if } \inf(\pi) = F\}$$

We call $\mathcal{F}$ the (Muller) winning condition. Let's call such win-lose h.o. games Muller games.

▷ **Question:** Are all Muller games determined? **Answer:** Yes.

▷ **Question:** Are all Muller games memorylessly determined?
Answer: No! Consider the following Muller game,
$\mathcal{F} = \{\{v_0, v_1, v_2\}\}$:



Does Player 1 have a winning strategy?
Does it have a memoryless winning strategy?

remarks

▷ Muller games and restricted variants of them are important in applications to model checking. We can't do them full justice here.

▷ Every Muller game can be converted to an "equivalent" (but potentially exponentially larger) game with a limited kind of Muller winning condition called a parity condition. These so called parity games *are* memorylessly determined.

▷ Can we find winning strategies in parity games efficiently (in P-time)? This is a tantalizing open problem.
It follows from memoryless determinacy that finding winning strategies for them is in **NP** $\cap$ **co-NP**: we can guess a memoryless strategy for either player and efficiently verify that it is a winning strategy.

▷ An older survey text on all this is:
[*"Automata, Logics, and Infinite Games"*,
edited by E. Grädel, W. Thomas, T. Wilke, 2002].

food for thought: back to LP

Consider the following LP, for solving a finitistic win-lose game with game graph G. (Suppose w.l.o.g., player 1 loses if the play is infinite.) Let $V = \{v_1, \ldots, v_n\}$ be vertices of G. We will have one LP variable x_i for each vertex $v_i \in V$.

Minimize x_m

Subject to:

$0 \leq x_i \leq 1$, *for* $i = 1, \ldots, n$;

$x_i = 1$, *for* v_i *a winning dead end for player 1.*

$x_i = 0$, *for* v_i *a losing dead end for player 1.*

For each x_i *where* $\mathrm{pl}(v_i) = 1$,

$\quad x_i \geq x_j$, *for each* $(v_i, v_j) \in E$.

For each x_i *where* $\mathrm{pl}(v_i) = 2$,

and $\{v_{j_1}, \ldots, v_{j_r}\} = \{v' \mid (v_i, v') \in E\}$,

$\quad x_i \leq x_{j_k}$, *for* $k = 1, \ldots, r$, *and*

$\quad x_i \geq x_{j_1} + \ldots + x_{j_r} - (r - 1)$

▷ The optimal value of the given LP is 1 iff player 1 has a winning strategy in $\mathcal{G}_{v_m}$.

▷ Now, what if instead of 2 players, player 1 was playing "alone against nature/chance"? Could you formulate an LP for 1's optimal payoff? This would be a simple instance of a "Markov Decision Process".