

2. Computational aspects of equilibria

Computational issues

In this chapter, we look at game theory from the point of view of a computer scientist.

We are interested in finding **algorithms** for computing equilibria, and also in understanding the computational **complexity** of this problem.

We will concentrate on games with two players, and we consider two cases:

1. Two player zero-sum games
2. Arbitrary two player strategic games

We start by some preliminaries on linear programming.

Linear programming basics

A **linear program** consists of:

- a set of real-valued **variables**
- a **linear objective function** (whose value we aim to maximize)
 - this is just a weighted sum of the variables
- a set of **linear constraints**
 - each constraint requires that a weighted sum of the variables be greater than or equal to some constant (can also use less than, or equality)

Linear programming problem

Typical form for a linear programming problem with n variables and m constraints:

“minimize”
can also
be used

maximize

subject to

$$\sum_{i=1}^n w_i x_i$$

objective function

$$\sum_{i=1}^n a_{ij} x_i \leq b_i \quad j = 1, \dots, m$$

linear constraints

$$x_i \geq 0 \quad i = 1, \dots, n$$

Goal: find values for the variables which satisfy the constraints and maximize (minimize) the value of the objective function

Solving linear programming problems

Many interesting practical problems can be phrased as linear programming problems.

Good news: **these problems can be solved in polynomial time.**

Two general types of algorithms exist:

Interior point methods:
terminate in polynomial time

Simplex method:
worst-case exponential time but works very well in practice

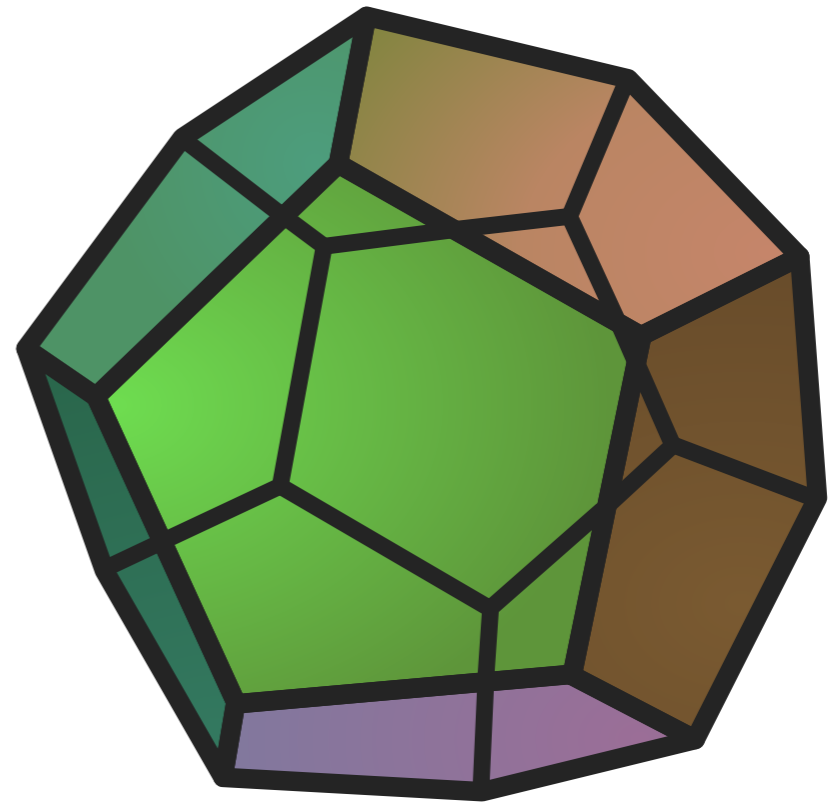
Intuitions for simplex method

Feasible region:

- Assignments of values to variables which satisfy all of the constraints
- Geometrical interpretation: convex polytope
- Optimal value of objective function is reached at some corner point

Simplex method:

- Identify an initial corner point
- Move to an adjacent corner point until local (=global) optimum found



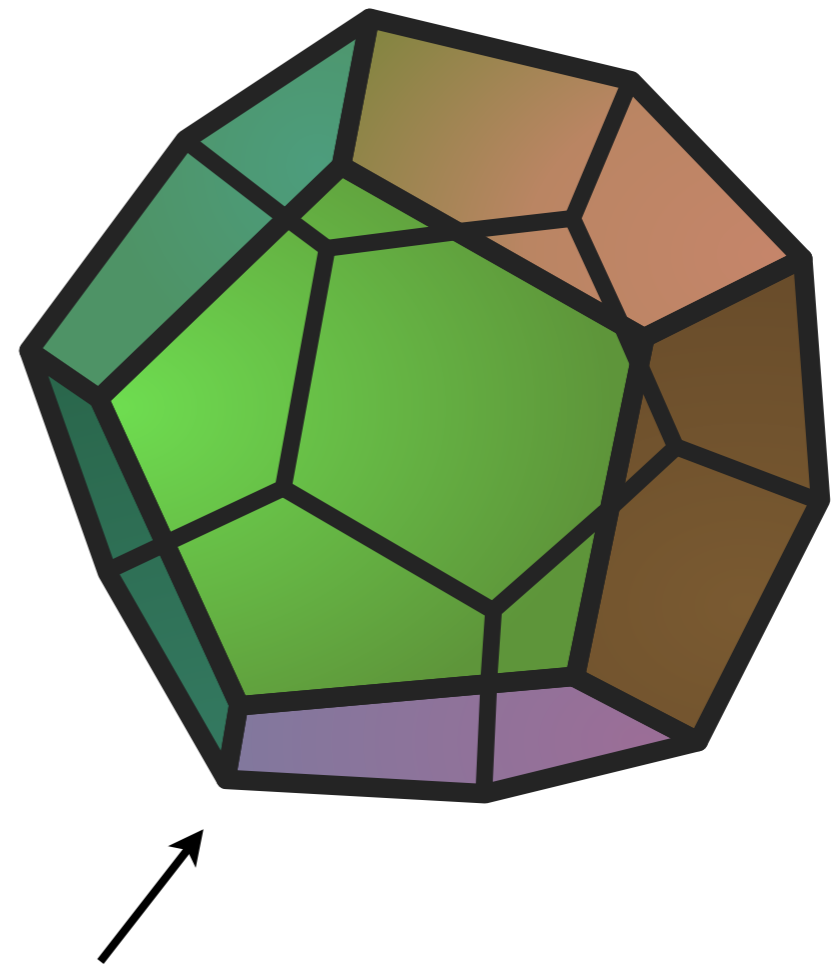
Intuitions for simplex method

Feasible region:

- Assignments of values to variables which satisfy all of the constraints
- Geometrical interpretation: convex polytope
- Optimal value of objective function is reached at some corner point

Simplex method:

- Identify an initial corner point
- Move to an adjacent corner point until local (=global) optimum found



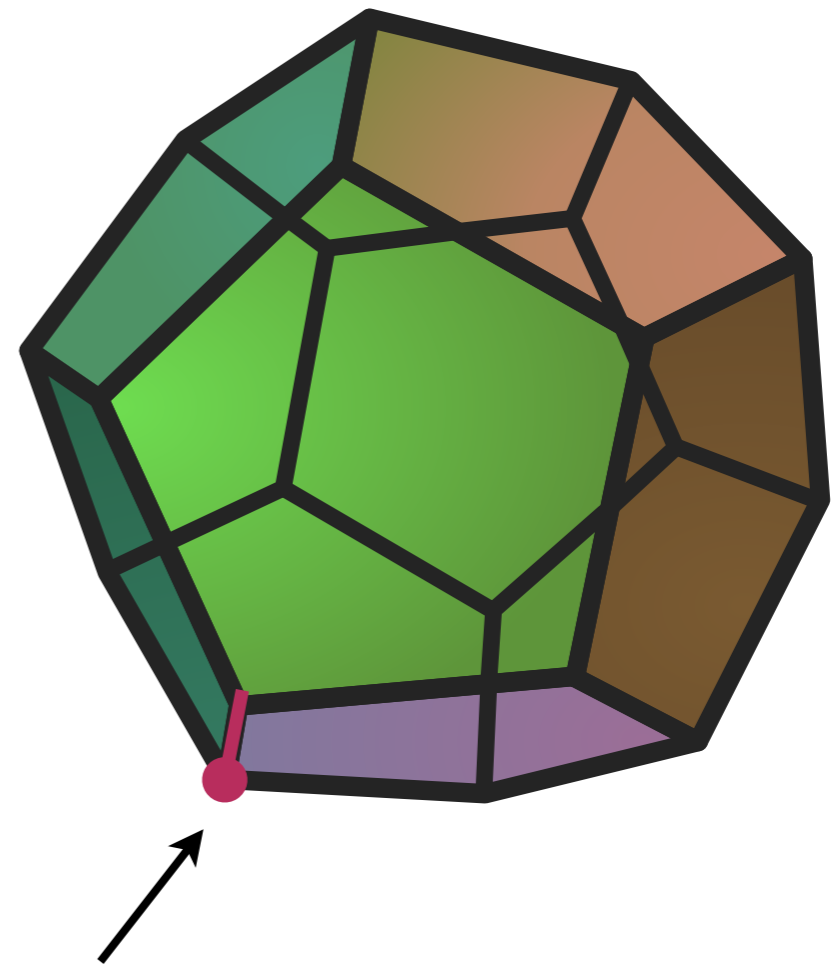
Intuitions for simplex method

Feasible region:

- Assignments of values to variables which satisfy all of the constraints
- Geometrical interpretation: convex polytope
- Optimal value of objective function is reached at some corner point

Simplex method:

- Identify an initial corner point
- Move to an adjacent corner point until local (=global) optimum found



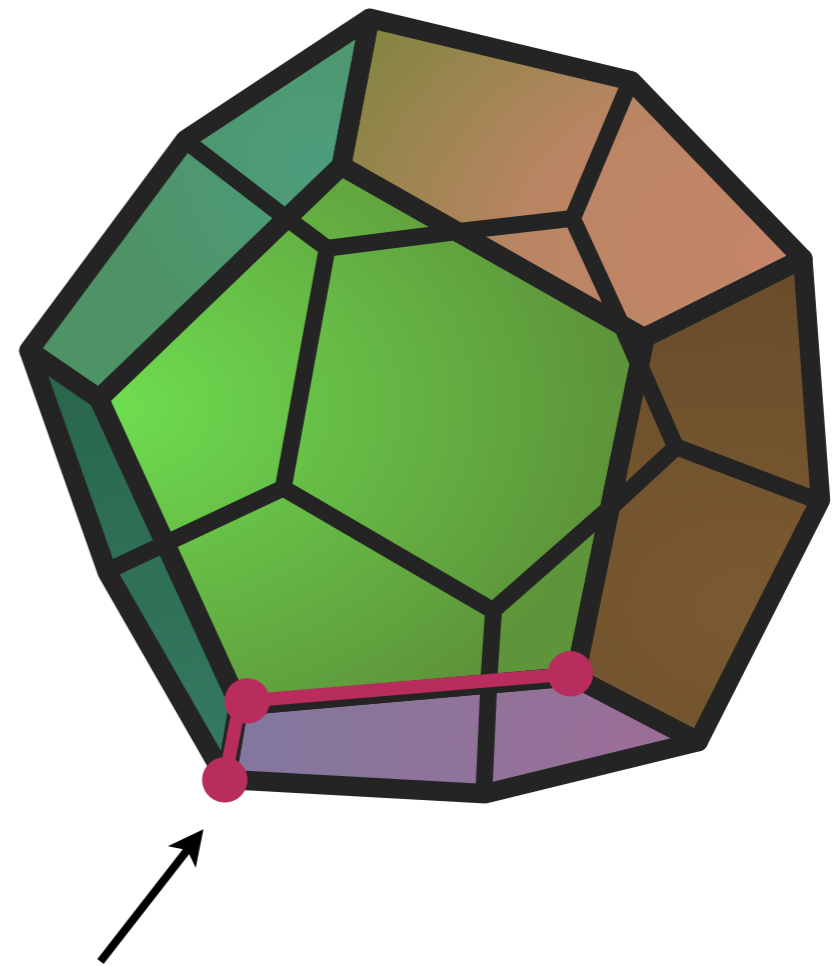
Intuitions for simplex method

Feasible region:

- Assignments of values to variables which satisfy all of the constraints
- Geometrical interpretation: convex polytope
- Optimal value of objective function is reached at some corner point

Simplex method:

- Identify an initial corner point
- Move to an adjacent corner point until local (=global) optimum found



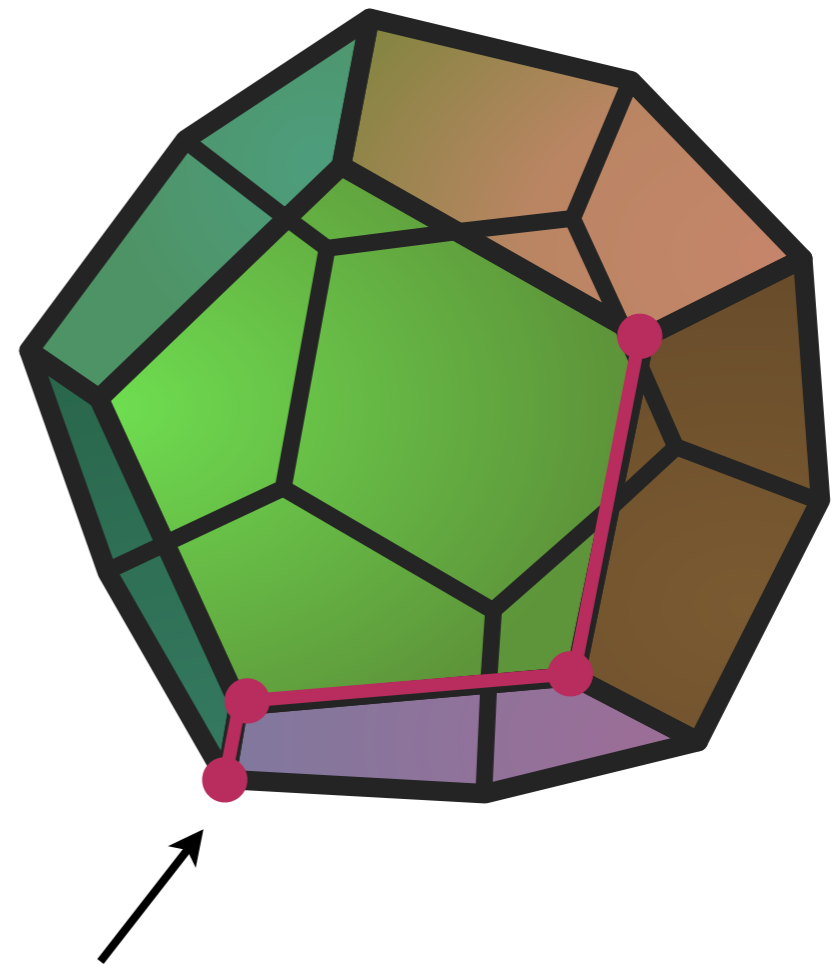
Intuitions for simplex method

Feasible region:

- Assignments of values to variables which satisfy all of the constraints
- Geometrical interpretation: convex polytope
- Optimal value of objective function is reached at some corner point

Simplex method:

- Identify an initial corner point
- Move to an adjacent corner point until local (=global) optimum found



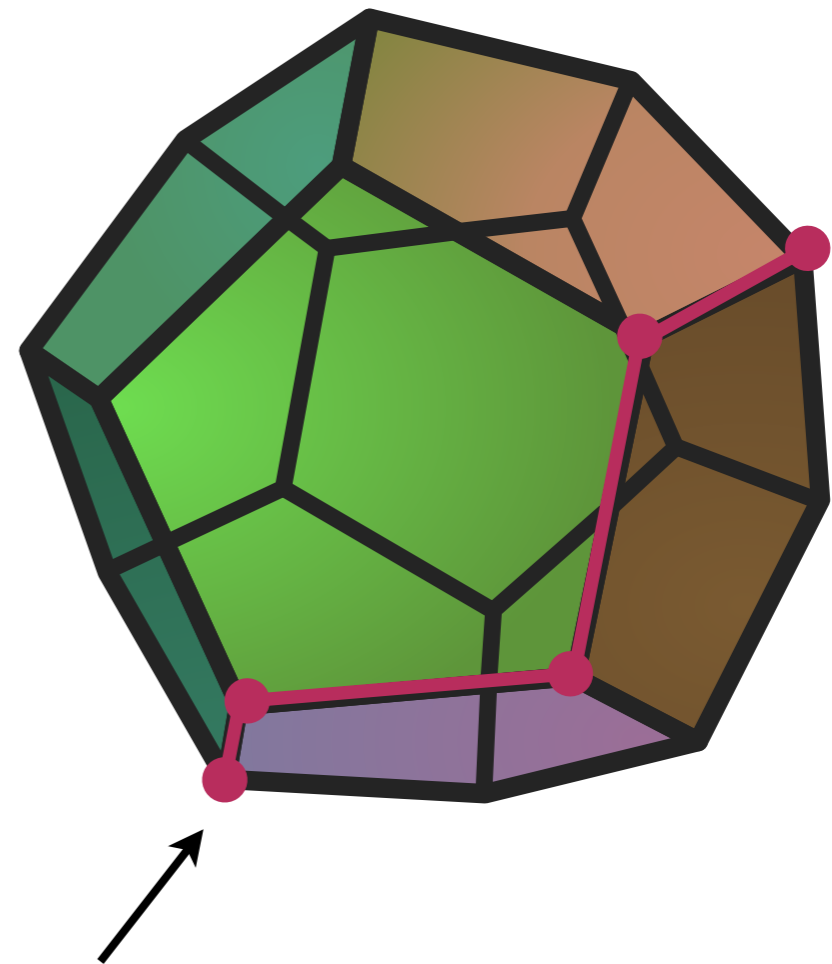
Intuitions for simplex method

Feasible region:

- Assignments of values to variables which satisfy all of the constraints
- Geometrical interpretation: convex polytope
- Optimal value of objective function is reached at some corner point

Simplex method:

- Identify an initial corner point
- Move to an adjacent corner point until local (=global) optimum found



Back to two-person zero-sum games

We aim to show how the problem of finding a Nash equilibrium can be rephrased as a linear programming (LP) problem.

This will show that a Nash equilibrium can be computed in polynomial time using existing algorithms for LP.

We will make use of the Minmax Theorem which tells us:

A strategy profile is a Nash equilibrium if and only if it is composed of minmax (= maxmin) strategies.

Therefore, we can find a Nash equilibrium by computing minmax strategies for both players.

We will show how to compute minmax strategies using LP.

Computing player 2's minmax strategy

The following LP program computes a minmax strategy for player 2 against player 1.

minimize v_1

subject to $\sum_{a_2 \in A_2} u_1(a_1, a_2) \cdot s_2^{a_2} \leq v_1$ for all $a_1 \in A_1$

$$\sum_{a_2 \in A_2} s_2^{a_2} = 1$$

$s_2^{a_2} \geq 0$ for all $a_2 \in A_2$

Computing player 2's minmax strategy

The following LP program computes a minmax strategy for player 2 against player 1.

minimize v_1

subject to $\sum_{a_2 \in A_2} u_1(a_1, a_2) \cdot s_2^{a_2} \leq v_1$ for all $a_1 \in A_1$

$$\sum_{a_2 \in A_2} s_2^{a_2} = 1$$

$s_2^{a_2} \geq 0$ for all $a_2 \in A_2$

Here v_1 is a variable representing the utility for player 1, and $s_2^{a_2}$ is a variable representing the probability of action a_2 in player 2's mixed strategy. The $u_1(a_1, a_2)$ are constants.

Computing player 2's minmax strategy

The following LP program computes a minmax strategy for player 2 against player 1.

minimize v_1

subject to $\sum_{a_2 \in A_2} u_1(a_1, a_2) \cdot s_2^{a_2} \leq v_1$ for all $a_1 \in A_1$

$$\sum_{a_2 \in A_2} s_2^{a_2} = 1$$

$s_2^{a_2} \geq 0$ for all $a_2 \in A_2$

The last two constraints make sure that the probabilities in the mixed strategy are non-negative and sum up to 1.

Computing player 2's minmax strategy

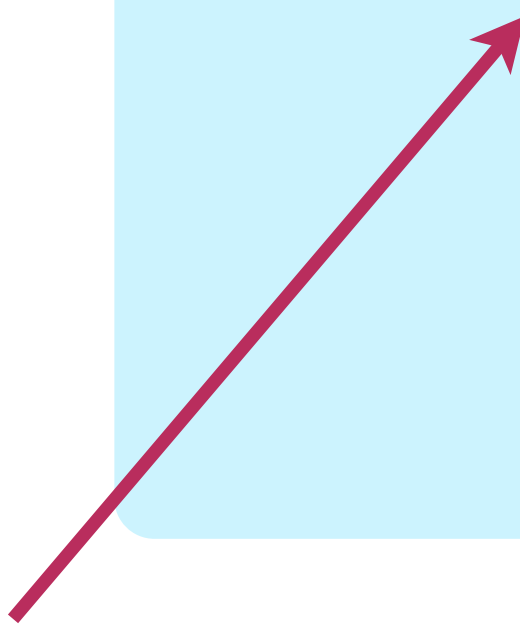
The following LP program computes a minmax strategy for player 2 against player 1.

minimize v_1

subject to $\sum_{a_2 \in A_2} u_1(a_1, a_2) \cdot s_2^{a_2} \leq v_1$ for all $a_1 \in A_1$

$$\sum_{a_2 \in A_2} s_2^{a_2} = 1$$

$s_2^{a_2} \geq 0$ for all $a_2 \in A_2$



The first constraint ensures that no matter what strategy player 1 chooses, his expected utility is at most v_1

Computing player 2's minmax strategy

The following LP program computes a minmax strategy for player 2 against player 1.

minimize v_1

subject to $\sum_{a_2 \in A_2} u_1(a_1, a_2) \cdot s_2^{a_2} \leq v_1$ for all $a_1 \in A_1$

$$\sum_{a_2 \in A_2} s_2^{a_2} = 1$$

$s_2^{a_2} \geq 0$ for all $a_2 \in A_2$

Since player 2 wants to minimize the utility of player 1, we must minimize the variable v_1

Computing player 2's minmax strategy

The following LP program computes a minmax strategy for player 2 against player 1.

minimize v_1

subject to $\sum_{a_2 \in A_2} u_1(a_1, a_2) \cdot s_2^{a_2} \leq v_1$ for all $a_1 \in A_1$

$$\sum_{a_2 \in A_2} s_2^{a_2} = 1$$

$s_2^{a_2} \geq 0$ for all $a_2 \in A_2$

A LP for computing a minmax strategy for player 1 can be constructed in a similar manner.

General two-player games

For general 2-player games, we formulate our NE problem as a **mixed integer programming problem (MIP)**.

Mixed integer programming problems are just like LP problems except that we can add constraints forcing some variables to only take integer values.

Mixed integer programming problems are thus more general than linear programming problems.

Unfortunately, this extra expressivity leads to an increase in computational complexity. There are no known polynomial time algorithms for solving MIP.

MIP formulation

find $s_i^{a_j} \geq 0$, $u_{a_i} \geq 0$, $r_{a_i} \geq 0$, $v_i \geq 0$, and $b_{a_i} \in \{0, 1\}$ such that

$$u_{a_1} = \sum_{a_2 \in A_2} u_1(a_1, a_2) \cdot s_2^{a_2} \quad \text{for all } a_1 \in A_1$$

$$u_{a_2} = \sum_{a_1 \in A_1} u_2(a_1, a_2) \cdot s_1^{a_1} \quad \text{for all } a_2 \in A_2$$

$$\sum_{a_1 \in A_1} s_1^{a_1} = 1 \quad \sum_{a_2 \in A_2} s_2^{a_2} = 1$$

$$s_1^{a_1} \leq 1 - b_{a_1} \quad s_2^{a_2} \leq 1 - b_{a_2} \quad \text{for all } a_1 \in A_1$$

$$r_{a_1} = v_1 - u_{a_1} \quad r_{a_2} = v_2 - u_{a_2} \quad \text{for all } a_2 \in A_2$$

$$r_{a_1} \leq d_1 \cdot b_{a_1} \quad r_{a_2} \leq d_2 \cdot b_{a_2}$$

Notes on the MIP formulation

In this particular MIP, there is no objective function to optimize, we just have to find values for the variables which satisfy the constraints.

d_1 and d_2 are constants defined as follows:

$$d_i = \max_{\substack{a_1^1, a_1^2 \in A_1 \\ a_2^1, a_2^2 \in A_2}} u_i(a_1^1, a_2^1) - u_i(a_2^1, a_2^2)$$

Thus, d_i represents the maximum difference in utility between two pure strategy profiles.

Binary variable b_{a_i} is used to indicate whether a_i is played with non-zero probability. If it is, we will have value 0, otherwise 1.

MIP formulation, step by step

First two constraints:

$$u_{a_1} = \sum_{a_2 \in A_2} u_1(a_1, a_2) \cdot s_2^{a_2} \quad \text{for all } a_1 \in A_1$$

$$u_{a_2} = \sum_{a_1 \in A_1} u_2(a_1, a_2) \cdot s_1^{a_1} \quad \text{for all } a_2 \in A_2$$

The first constraint ensures that u_{a_1} is assigned the utility for player 1 of playing the pure strategy a_1 against player 2's mixed strategy.

The second constraint does the same, but for player 2.

MIP formulation, step by step

Third and fourth constraints:

$$\sum_{a_1 \in A_1} s_1^{a_1} = 1 \quad \sum_{a_2 \in A_2} s_2^{a_2} = 1$$

As before, these constraints are used to ensure that the variables $s_i^{a_j}$ define a proper probability distribution.

We require the probabilities of each player's actions to sum to 1.

MIP formulation, step by step

Fifth and sixth constraints:

$$s_1^{a_1} \leq 1 - b_{a_1} \quad \text{for all } a_1 \in A_1$$

$$s_2^{a_2} \leq 1 - b_{a_2} \quad \text{for all } a_2 \in A_2$$

If $b_{a_1} = 0$, constraint trivially holds.

If $b_{a_1} = 1$, require that a_1 not played, i.e. $s_1^{a_1} = 0$.

Similarly for the second constraint above.

MIP formulation, step by step

Remaining constraints:

$$r_{a_1} = v_1 - u_{a_1} \quad \text{for all } a_1 \in A_1$$

$$r_{a_1} \leq d_1 \cdot b_{a_1}$$

$$r_{a_2} = v_2 - u_{a_2} \quad \text{for all } a_2 \in A_2$$

$$r_{a_2} \leq d_2 \cdot b_{a_2}$$

Variable v_i represents the highest possible expected utility that player i can obtain given the other player's mixed strategy.

MIP formulation, step by step

Remaining constraints:

$$r_{a_1} = v_1 - u_{a_1} \quad \text{for all } a_1 \in A_1$$

$$r_{a_1} \leq d_1 \cdot b_{a_1}$$

$$r_{a_2} = v_2 - u_{a_2} \quad \text{for all } a_2 \in A_2$$

$$r_{a_2} \leq d_2 \cdot b_{a_2}$$

Variable r_{a_i} represents the *regret* of playing a_i , i.e. the difference in utility between playing a_i and playing a best response to the other player's strategy.

MIP formulation, step by step

Remaining constraints:

$$r_{a_1} = v_1 - u_{a_1} \quad \text{for all } a_1 \in A_1$$

$$r_{a_1} \leq d_1 \cdot b_{a_1}$$

$$r_{a_2} = v_2 - u_{a_2} \quad \text{for all } a_2 \in A_2$$

$$r_{a_2} \leq d_2 \cdot b_{a_2}$$

In other words, every action played with non-zero probability must be a best response to the other player's mixed strategy.

MIP formulation

find $s_i^{a_j} \geq 0$, $u_{a_i} \geq 0$, $r_{a_i} \geq 0$, $v_i \geq 0$, and $b_{a_i} \in \{0, 1\}$ such that

$$u_{a_1} = \sum_{a_2 \in A_2} u_1(a_1, a_2) \cdot s_2^{a_2} \quad \text{for all } a_1 \in A_1$$

$$u_{a_2} = \sum_{a_1 \in A_1} u_2(a_1, a_2) \cdot s_1^{a_1} \quad \text{for all } a_2 \in A_2$$

$$\sum_{a_1 \in A_1} s_1^{a_1} = 1 \quad \sum_{a_2 \in A_2} s_2^{a_2} = 1$$

$$s_1^{a_1} \leq 1 - b_{a_1} \quad s_2^{a_2} \leq 1 - b_{a_2} \quad \text{for all } a_1 \in A_1$$

$$r_{a_1} = v_1 - u_{a_1} \quad r_{a_2} = v_2 - u_{a_2} \quad \text{for all } a_2 \in A_2$$

$$r_{a_1} \leq d_1 \cdot b_{a_1} \quad r_{a_2} \leq d_2 \cdot b_{a_2}$$

- Every solution to the above MIP yields a Nash equilibrium.
- Every Nash equilibrium corresponds to some solution.

Solving MIP problems

MIP problems are search problems.

Lots of work in computer science on search algorithms.

MIP solvers with sophisticated optimizations can be used.

Various heuristics can be used to help guide the search.

Experimental results show that adding an objective function to our MIP formulation can greatly improve performance.

Examples of objective functions one can add:

- minimize number of actions played
- maximize sum of players' utilities

Complexity of finding Nash equilibria

We know that for two player zero-sum games, we can find a Nash equilibrium in polynomial time.

But what about general two player games ?

Or games with more than two players ?

Known algorithms all run in exponential time, but so far nobody has shown that a polynomial algorithm cannot exist.

“Together with factoring, the complexity of finding a Nash equilibrium is in my opinion the most important concrete open question on the boundary of P today” (Papadimitriou)

Is this problem NP-complete ?

NP-complete problems (SAT, travelling salesman, ...) are a class of problems for which no polynomial time algorithms are known, but no one has shown that they don't exist.

So is the problem of finding a NE an NP-complete problem ?

Probably not.

First: a search problem, not a decision problem.

Second: NE always exist, whereas NP-complete problems usually involve testing whether a solution exists

The class PPAD

NE problem is related to a less well-known class called **PPAD**.

Canonical PPAD-complete problem

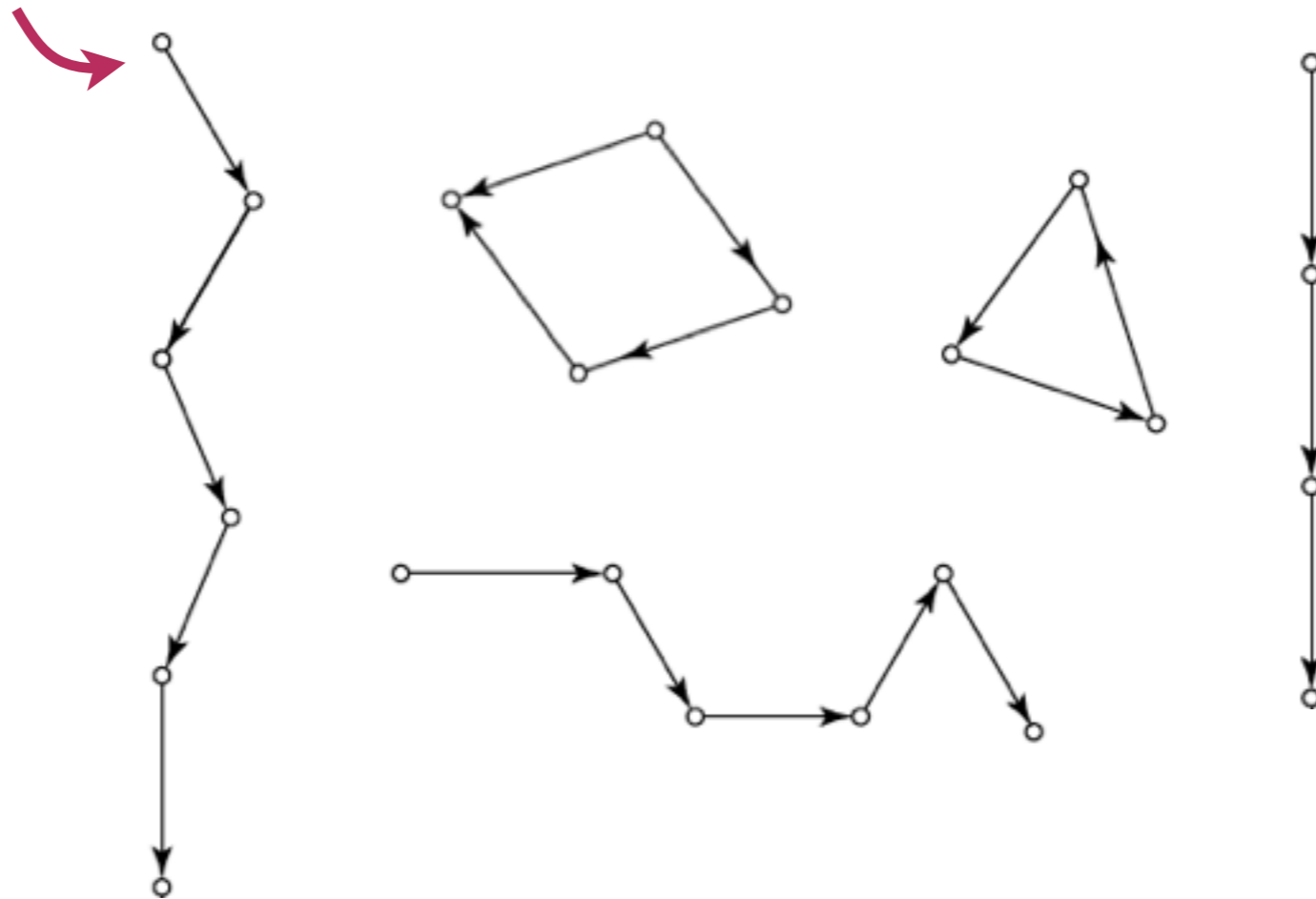
- Suppose we have an exponential-size graph where the in-degree and out-degree of each node is at most 1
- Given any node in the graph, we have a polynomial-time algorithm that finds the neighbours of the node [succinct encoding of the graph]
- **Problem:** given a parentless node, output a childless node

Note: at least one childless node must exist

The class PPAD

Typical PPAD problem:

given parentless node



Completeness for PPAD

Finding a Nash equilibrium is **PPAD-complete**

- for general n -player games with $n > 3$ [2005]
- for general 3-player games [later in 2005]
- for general 2-player games [even later in 2005]

Finding the Nash equilibrium even for general 2-player games is no easier than doing it for n -player games !

Currently very little is known about the class PPAD, and in particular, its relation to the class P.

Related NP-complete problems

Many problems related to Nash equilibria can be shown to be **NP-complete** (and thus probably cannot be solved in polytime):

- **Uniqueness**: Is there a unique Nash equilibrium?
- **Pareto optimality**: Does there exist a Pareto-optimal Nash equilibrium?
- **Guaranteed payoff**: Given a value v , does there exist a Nash equilibrium in which some player obtains an expected payoff of at least v ?
- **Guaranteed social welfare**: Does there exist a Nash equilibrium in which the sum of agents' utilities is at least k ?

What about correlated equilibria?

Good news: the equations we gave for determining whether a strategy profile is a correlated equilibrium define a linear program.

This means we can use LP solvers to compute correlated equilibria.

It follows that a correlated equilibrium of a given game can be found in polynomial time.

Can also compute a correlated equilibrium maximizing some linear objective function. Example: maximize social welfare.

More good news: the Uniqueness, Pareto-optimality, Guaranteed payoff, and Guaranteed social welfare problems can all be solved in polynomial time.

Computational aspects of IESDS / IEWDS

Computing result of IESDS / IESWS can be done in polytime.

Some relevant decision problems:

- **Strategy elimination:** Does there exist some elimination path under which the strategy s_i is eliminated?
- **Reduction identity:** Given action subsets $B_i \subseteq A_i$ for each player i , does there exist a maximally reduced game where each player i has the actions B_i ?
- **Reduction size:** Given constants k_i for each player i , does there exist a maximally reduced game where each player i has exactly k_i actions?

These problems are in P for IESDS, but NP-complete for IEWDS.

What about extensive form games?

One option: just use translation to strategic form representation

- Misses issues of subgame perfection, etc.
- Strategic form can be exponentially larger, so add another exponential to the running time
- Mixed strategy takes exponential space to represent

Another option: try to work directly on the game tree

- sequence form: different representation in terms of paths of tree
- for 2-player, zero-sum perfect recall games, obtain polytime algorithm for computing NE in behavioral strategies