Simplex: Odds & Ends

- Slightly more general objective functions

- Linear programming is a good modelling tool

- Simplex is (probably) not polynomial but LP can be solved in polynomial time using Ellipsoid or Interior Point Algorithms

- For a *user* LP is a *modelling tool*. You don't need to worry about how to solve LPs. Just plug them into Matlab, Maple, LP-Solve, etc.

- **Integer Linear Programming**, which restricts the x to be integers, is NP-Hard.

- In practice, one can *bound* the optimal solution of NP-hard problems by modelling them as **Integer Linear Programs (ILPs)** and then *relax* the ILPS to be **Linear Programs**.

We usually write simplex in the form

$$\min_{x \in F} z = c'x \quad \text{where} \quad F = \{x : Ax = b,\ x \geq 0\}$$

for A an $m \times n$ matrix and $x \in R^n$. We already saw that

$$\max_{x \in F} c'\,x \quad \text{is equivalent to} \quad \min_{x \in F} -c'\,x$$

so LP can also model maximization problems.

In many cases, e.g., modelling quality of service problems in network routing, we might be asked to optimize the more complicated

$$\max_{x \in F} \left(\min_{i \in [1,n]} x_i \right)$$

or even

$$\max_{x \in F} \left(\min_{j \in [1,t]} d_j'\,x \right)$$

where $d_j \in R^n$, $j = 1, \ldots, t$. At fist glance this looks like something totally unrelated, but this can actually easily be modelled using LPs. How?

Modelling with LPs

Linear programming is widely appliable.

One common use is in optimizing graph related problems (and since graphs are often used to model many problems this leds to many applications).

Routing problems can often be modelled using LPs as can many graph covering problems (leading to many database applications).

One important observation is that you, as a user, only have to worry about modelling your problem as a LP. You don't have to worry about actually solving the LP since there is so much good code around.

All of the standard math programs, e.g., Matlab, Maple, Macsyma, have LP solvers built in. For larger problems there is a good freeware package, LP-Solve. For industrial strength problems there are highly optimized expensive pieces of software, such as CPLEX.

The simplex algorithm, as taught, **is not polynomial**.

The essential problem is that if matrix A is an $m \times n$ matrix then there can, in the worst case, be $\binom{n}{m} \sim n^m$ BFSs, an exponential number.

In order for simplex not to be exponential, we would have to find a pivot-rule that managed to always permit us to skip most BFSs and, with a short (polynomial) sequence of pivots, always find the optimum.

Unfortunately, for every pivot-rule known, there is a counterexample with exponential number of BFSs that forces simplex with that pivot rule to visit (almost) every BFS.

Big Open Question: Does there exist a pivot-rule that always gives polynomial time behavior.

What is known is that, for some pivot rules, simplex runs in polynomial time "on average".

The best simplex code is highly optimized and “usually” runs very quickly.

There are also two known polynomial time algorithms for solving LP, the Ellipsoid algorithm and Karmakar’s method.

The Ellipsoid method is primarily of theoretical interest but Karmakar’s method (an “interior point” method) is, on some types of LPs, actually faster than simplex.

An Integer Linear Program (ILP) is formulated exactly the same as a Linear Program (LP) with the additional constraint that all of the x_i must be be integers .

$$\min_{x \in F} z = c'x$$

where

$$F = \{x \; : \; Ax = b, \; x \geq 0, \; \text{and } x_i \text{ integers}\}$$

ILP is NP-Hard!

We now see an example of how ILP can model, minimum, set cover, an NP-Hard problem. So, ILP being solvable in polynomial time would imply that set cover is solvable in polynomial time.

The Set Covering Problem

Let X be a set and $\mathcal{F}$ a family of subsets of X such that $X = \cup_{F \in \mathcal{F}} F$.

For example $X = \{1, 2, 3, 4, 5, 6\}$ and $\mathcal{F}$ contains the subsets

$$\begin{aligned} F_1 &= \{1, 3, 5\} \\ F_2 &= \{2, 3, 6\} \\ F_3 &= \{2, 5, 6\} \\ F_4 &= \{2, 3, 4, 6\} \\ F_5 &= \{1, 4\} \end{aligned}$$

A subset $F \in \mathcal{F}$ *covers* its elements.

The problem is to find a *minimum-size subset* $\mathcal{C} \subseteq \mathcal{F}$ that covers X, i.e., $X = \cup_{F \in \mathcal{C}} F$.

For example $\{F_1, F_2, F_4\}$ covers X but is not a minimal size solution.

$\mathcal{C} = \{F_1, F_4\}$ is a minimal size solution.

Finding a minimal-size set cover is NP-Hard.

Let $X = \{u_j\}_j$. Construct an integer linear program with one variable x_i for each set F_i.

$$\boxed{\begin{array}{l} z = \min \sum_i x_i \\ \text{subject to conditions} \\ \forall j,\ \sum_{u_j \in F_i} x_i \geq 1 \\ \forall i,\ x_i \leq 1 \\ \forall i,\ x_i \geq 0 \\ \forall i,\ x_i \text{ is an integer} \end{array}}$$

Note that conditions imply that if x is feasible then

$$\forall i,\ x_i \in \{0, 1\}.$$

Now, given a feasible solution x to the ILP define

$$\mathcal{C}(x) = \{F_i\ :\ x_i = 1\}.$$

The condition $\forall j,\ \sum_{u_j \in F_i} x_i \geq 1$ means that $\mathcal{C}$ is a cover, so every feasible x corresponds to a cover.

Working backwards, given cover $\mathcal{C}$, we can define $x_i(\mathcal{C}) = 1$ if $F_i \in \mathcal{C}$ and 0 otherwise. $x(\mathcal{C})$ is a feasible solution. This gives us a one-one correspondence between feasible solutions and set covers.

Since $|\mathcal{C}| = \sum_i x_i$, minimizing the objective function is equivalent to solving minimal set cover.

Given ILP that models our problem, e.g.,

$$\boxed{\begin{array}{l} z = \min \sum_i x_i \\ \text{subject to conditions} \\ \forall j,\ \sum_{u_j \in F_i} x_i \geq 1 \\ \forall i,\ x_i \leq 1 \\ \forall i,\ x_i \geq 0 \\ \forall i,\ x_i \text{ is an integer} \end{array}}$$

we can relax the ILP to get an LP

$$\boxed{\begin{array}{l} \bar{z} = \min \sum_i x_i \\ \text{subject to conditions} \\ \forall j,\ \sum_{u_j \in F_i} x_i \geq 1 \\ \forall i,\ x_i \leq 1 \\ \forall i,\ x_i \geq 0 \end{array}}$$

Note that, by definition,

$$\bar{z} \leq z$$

This might look trivial but it is **extremely powerful**.

In general, suppose you have a situation in which you have an NP-Hard problem to solve, e.g., QOS networking applications. You've designed a heuristic that you think works well but how can you convince people of this?

In most cases you try to run your heuristic on (real or benchmark) problems and then would like compare your solutions to the "real" optimal solutions to show that your solutions are very close. But, the problem is NP-hard, so how can you find the real optimal solution to which to compare it?

Often, this difficulty is finessed by comparing your results to the benchmark results derived by others but that is often not intellectually satisfying (and might not really mean much anyway; what justifies the benchmark?).

Another approach is to

- Write your minimization problem as an ILP.
  Denote unknown optimal solution to ILP as z.

- Relax the ILP to an LP.

- Solve the associated LP (in polynomial time) to get "optimal relaxed solution" $\bar{z}$.

- Run your heuristic on original problem to get heuristic solution s.

- We would like to know $s - z$ or $\frac{s-z}{z}$ but can't calculate them.

  We can, though, calculate $s - \bar{z}$ and $\frac{s-\bar{z}}{\bar{z}}$.

  Since $s - z \le s - \bar{z}$ and $\frac{s-z}{z} < \frac{s-\bar{z}}{\bar{z}}$, this permits us to upperbound (relative) error to optimum **without knowing the optimum.**