Mathematics for Decision Making: An Introduction

Lecture 8

Matthias Köppe

UC Davis, Mathematics

January 29, 2009

Shortest Paths and Feasible Potentials

Feasible Potentials

Suppose for all $v \in V$, there exists some (directed) path P_v from r to v of cost y_v. Suppose there is one arc $(v, w) \in A$ with $y_v + c_{v,w} < y_w$. Then we know that there is a path $P'_w = \big(P_v, (v, w)\big)$ that is cheaper than P_w (**descent step**).

In particular: If for all $v \in V$, we have that y_v is the cost of a minimum-cost path P^*_v from r to v, then

$$y_v + c_{v,w} \geq y_w \quad \text{for all } (v, w) \in A. \tag{1}$$

We call any vector $\mathbf{y} = (y_v)_{v \in V} \in (\mathbf{R} \cup \{+\infty\})^V$ a **potential**; we call it a **feasible potential** if $y_r = 0$ and (1) holds.

Lemma (Feasible potentials provide **lower bounds**)

Let $\mathbf{y}$ *be a feasible potential and* P_v *be a path from* r *to* v*. Then* $c(P_v) \geq y_v$.

In particular, if $c(P_v) = y_v$, then P_v is a minimum cost path (**optimality criterion**).

The Algorithm of Ford [1956]

We have discovered two ingredients of a **descent algorithm**:

1. A **descent step** that moves from one solution to a better solution.
2. An **optimality criterion** that tells us when to stop.

We need one more thing:

3. An **initial solution:** We can start from a **y** with $y_r = 0$ and $y_v = +\infty$ for all $v \neq r$ (note this is not a feasible potential). We start with a predecessor vector **p** with $p(r) = 0$ and $p(v) = -1$ (to indicate we don't know any path yet)

Ford's Algorithm

Input: A digraph with arc costs, starting node r
Output: Shortest paths from r to all other nodes

```
Initialize y and p;
While y is not a feasible potential:
  Find an incorrect arc $(v, w)$ and correct it, updating predecessor information
Reconstruct shortest paths from p.
```

Ford's Algorithm is a prototype of a **label-correcting algorithm**.

Note that, as is, it is underspecified (not strictly an algorithm) because we do not say which incorrect arc should be taken.

Ford’s Algorithm and Negative-Cost Directed Circuits

On the second example of the homework, we noticed:

- Sometimes, Ford’s Algorithm does not terminate!
  Strictly speaking, because of this, it is **not an algorithm**, i.e., a finite procedure!
- This is caused by a **negative-cost directed circuit** in the problem.
  Along such a circuit, the algorithm will update the potentials y_v, so they become smaller and smaller, without bounds.
  We remark that in this example, for all nodes v on the directed cycle, **there does not exist a least-cost directed path from** r **to** v! We can go through the directed cycle an arbitrary number of times, to get directed paths that have arbitrarily low (negative) cost.

By changing the input specification and just excluding the case of negative-cost directed circuit, we can make Ford’s method always finite, i.e., turn it into an **algorithm**.
We emphasize that **this needs to be proved**, which we do next.
We also prove that it is **correct**, i.e., when it finishes, it has computed the desired result.

Finiteness and Correctness of Ford's Method, I

We need to prove theorems that establishes properties of the contents of our data structures (here: $\mathbf{y}$ and $\mathbf{p}$) during the execution of the algorithm.

Proof technique

Most algorithms have **loops** (here: a while loop). Then proofs of the theorems will be **inductive proofs**, where we show that

- a certain property holds before we enter the loop,
- if the property holds at the beginning of one loop iteration, it also holds at the end of this loop iteration.

This implies that the property also holds when the loop is left.

Lemma (Loop invariants in Ford's Algorithm)

If $\left(G = (V, A), \mathbf{c}\right)$ has no negative-cost directed circuit, then, ***at the beginning and end of any iteration of the while loop*** *in Ford's Algorithm, we have for each $v \in V$:*

1. *If $y_v < \infty$, then it is the cost of a* ***simple*** *directed path (i.e., a directed path where no vertex is visited twice) from r to v.*
2. *If $p(v) \neq -1$, then the predecessor vector* $\mathbf{p}$ *recursively defines a simple directed path from r to v* ***of cost at most*** *y_v.*

Finiteness and Correctness of Ford’s Method, II

Theorem (Finiteness and Correctness)

If $\big(G = (V, A), \mathbf{c}\big)$ has no negative-cost directed circuit, then Ford’s Algorithm terminates after a finite number of iterations. At termination, **p** *defines a least-cost directed path from r to v of cost y_v.*

Proof.

Finiteness:

- By the Loop Invariants Lemma, every y_v is the cost of a **simple** directed path.
- There are only finitely many simple directed paths, thus only finitely many possible values for each y_v.
- In each correction step, one y_v is lowered. Thus only finitely many steps.

Correctness:

- By the Loop Invariants Lemma, for each $v \in V$, the predecessor vector **p** defines a simple directed path P_v from r to v of cost **at most** y_v.
- But the loop has finished, so **y** is a **feasible** potential, so by the Lower Bound Lemma, any path from r to v has cost **at least** y_v.
- So P_v is optimal.

More About Feasible Potentials

Theorem (Characterization of networks with feasible potentials)

$(G, \mathbf{c})$ *has a feasible potential if and only if it has no negative-cost directed circuit.*
(This holds without assuming there always exists a directed path from a source!)

Proof.

If G has a feasible potential $\mathbf{y}$, then it does not have a negative-cost directed circuit:

- Let $v_0, v_1, \ldots, v_{k-1}, v_k = v_0$ be a directed circuit. As in the proof of the Loop Invariants Lemma, adding up $c_w \geq y_i - y_{i-1}$ along the circuit yields $c(P) \geq 0$.

Now suppose that G does not have a negative-cost directed circuit:

- Add an artificial source vertex r and zero-cost arcs (r, v), making a new graph G'.
- This graph G' satisfies our assumption that there always exists a directed path from the source r to any vertex.
- G' also does not have a negative-cost directed circuit.
- Apply Ford's Algorithm.
- By the Finiteness and Correctness Theorem, G' has a feasible potential $\mathbf{y}'$.
- This gives a feasible potential $\mathbf{y}$ for G. □

Efficiency of Algorithms

Now that we know that it terminates (in the case of no negative-cost directed cycles), we can ask how fast (**efficient**) it is!

- We could implement it on a computer, and systematically test how long it takes for graphs of various sizes, and, for a given size, how the running time depends on the actual data.
- (You did this test with a TSP formulation and SCIP as a **black box**.)
- However, in the case of shortest paths and Ford's Algorithm, we know the algorithm precisely (it's a **transparent box**), so we can do better
- We will make **mathematical statements** about the running time, and **prove them**.
- Because different computers have different speeds, we need a **mathematical abstraction of running time** or even a **mathematical abstraction of a machine**.
- Theoretical Computer Science has a wealth of such abstractions.
- We start with a very coarse abstraction that is sufficient for analyzing Ford's Algorithm: We just count the iterations of our *while* loop.

Theorem (Efficiency of Ford's Algorithm)

*If $\mathbf{c}$ is integer-valued and $G = (V, A)$ has no negative-cost directed circuit, then Ford's Algorithm terminates after **at most** Cn^2 steps, where $n = |V|$ and*

$$C = m(\mathbf{c}) = 1 + 2||\mathbf{c}||_\infty \quad \textit{with} \quad ||\mathbf{c}||_\infty = \max\{\,|c_{(a,b)}| : (a,b) \in A\,\}.$$

- This is a typical statement of an efficiency result: We establish an **upper bound** for (some abstraction of) the **running time**, with a **simple formula**. Note we do not predict the precise running time; the algorithm could be much faster than that.

Proof.

- The first correction step, leading to a finite potential value y_v of a vertex, gives y_v that is at most $||\mathbf{c}||_\infty(n-1)$, because it is the cost of a simple directed path (with at most $n-1$ arcs!) from the root.
- In every later correction step, y_v gets reduced by at least 1, because $\mathbf{c}$ is integral
- In the end, the potential value y_v is the cost of a least-cost path; this may be negative, but we certainly have $y_v \geq -||\mathbf{c}||_\infty(n-1)$.
- Thus at most $1 + 2 \cdot ||\mathbf{c}||_\infty(n-1) \leq Cn$ steps per vertex
- Hence, at most Cn^2 correction steps in total. □

Efficiency of Ford's Algorithm, II

- The theorem establishes that Ford's Algorithm is a **pseudo-polynomial algorithm**, i.e., its running time is bounded above by a polynomial expression in the "dimensions" (such as n_k) and the absolute values of its input data.
- Because the bound is monotonous in C and n, it is convenient to interpret this bound as an upper bound on the running time of the **worst case** that can happen among all problems $(G, \mathbf{c})$ with $|V| \leq n$ and and $m(\mathbf{c}) \leq C$.
- In a homework exercise, you show that there is a one-parameter family $\{ (G_k = (V_k, A_k), \mathbf{c}^k) : k \in \mathbf{N} \}$ of networks with $n_k = |V_k| = 2k + 4$ vertices and $C_k = 2^k$, such that Ford's Algorithm (with a specific, clever, evil way of choosing **which** incorrect arc should be corrected) takes more than 2^k steps.
- This shows that the upper bound is "not too far off" from the worst case

Better efficiency classes:

- We are not happy with pseudo-polynomial algorithms, because for the same graph G, the running time might grow quickly if we just use "large numbers"
- Better are **polynomial algorithms**, where the running time is allowed to grow polynomially in the "dimensions" (such as n), but only polynomially in the **number of digits** needed to write down the data (such as $c_{a,b}$)
- Even better are **strongly polynomial algorithms**, where the worst-case running time (#steps) is allowed to depend only on the dimensions, not on the data

Homework 1

1. Currency arbitrages
   1. Use Internet resources to learn about the notion of **currency arbitrage**.
   2. In a nutshell, given is a matrix R of currency exchange rates, i.e., for each pair (v, w) of currencies, an exchange rate $r_{v,w}$ is given, i.e., the number of units of currency w that can be paid for with one unit of currency v. Further given is a currency unit a (for instance, USD); we start with 1,000 of the currency unit a and buy and sell currencies using the given exchange rates, until we get currency unit a back. If we make a profit, this is called an **arbitrage opportunity**.
   3. Find a mathematical formulation for the problem of determining whether there exists a currency arbitrage, and finding the maximum possible arbitrage. If your formulation is nonlinear, find a way to make it linear by a change of variable. Create a ZIMPL model; the exchange rate data should be read from a separate data file.
   4. From Internet resources or the *Wall Street Journal*, obtain current exchange rates between USD, EUR, GBP, JPY, and CHF (at your choice, additional currencies). By solving your ZIMPL model with SCIP, determine whether there exists an arbitrage opportunity starting from your 1,000 US dollars, and, if so, determine the maximum possible arbitrage.
   5. If you discovered an arbitrage opportunity with these data, explain whether you would be able to make real profit from it.

Homework 2, 3

2. Modeling with graphs.
   Back in the GPS Navigation System problem, show how to model complicated street intersections, such as a standard north/south–east/west intersection where left turns are not allowed for cars coming from north or south.
3. A problem reduction.
   Packaged as a black box, you receive from me an algorithm that solves the following problem:

Least-cost simple directed path problem

Given a directed graph $G = (V, A)$, arc costs $c_{v,w} \in \mathbf{R}$, which are allowed to be negative, and two vertices $r, s \in V$, find a least-cost **simple** directed path from r to s in G.

Show that you can solve an arbitrary symmetric traveling salesman problem by making use of this algorithm.