Mathematics for Decision Making: An Introduction

Lecture 7

Matthias Köppe

UC Davis, Mathematics

January 27, 2009

Back to the pen plotter: An Insight from Graph Theory

We observed that there are some line drawings that can be drawn entirely in pen-down mode, but most drawings require using pen-up mode as well – we need to move from one "dead end" to a new start. This theorem gives us a characterization:

Theorem (Euler walk)

*A **connected** graph has an Euler walk if and only if there are 0 or 2 vertices of odd degree.*

A new idea for modeling the pen plotter problem is:

- We ignore the orientation of lines, i.e., in which direction we draw them
- Consider a set F of "movement" edges that we add to the graph (V, E), such that the (multi)graph $(V, E + F)$ has an Euler walk (here we allow parallel edges)
- We want to pick this set F such that the total movement time is minimized (note the actual drawing time is the same (constant) in every solution).
- An important observation is that it never pays off to connect edges to vertices that already have even degree.
- Thus, the set F can always be chosen as a set of edges connecting vertices of odd degree – to be precise, a pairing (**matching**) of the vertices of odd degree, i.e., each vertex of odd degree belongs to exactly one edge of F
  (To deal with the start/end point, we again introduce artificial vertices.)

Pen plotter problem: Matching formulation

We have reduced the problem to a **matching problem**:

Minimum-cost perfect matching problem

Given a graph (V, E) and costs $c_{a,b}$ for all $\{a, b\} \in E$, find a perfect matching F (i.e., in the case of an even $|V|$, a subset F of E such that every vertex is incident to precisely one edge) of minimum total cost.

The optimization model can be found in `plotter-matching.zpl`

- 50 points, 10% density: 276 variables, 0.04 seconds
- 50 points, 50% density: 325 variables, 0.05 seconds
- 100 points, 10% density: 1327 variables, 0.09 seconds
- 400 points, 10% density: 17579 variables, 10.5 seconds
- 1000 points, 1% density: 130817 variables, after 67 seconds: 0.01% gap, total 160.2 seconds
- 1000 points, 10% density: 126757 variables, after 80 seconds: 0.00% gap, total 150.6 seconds

Case Study: The Shortest Path Problem in GPS Navigation

The fundamental problem to be solved is to find the "shortest" path from r to s through the network of streets and roads.

- **Definition of distances:** Either road length, or expected driving time (depends on road length and maximum allowed speed or typical speed at a given time of day). Some GPSs take real-time traffic data into account.
- **Mathematical abstraction of the network:** Digraph (V, A) with V vertex set of street intersections, street numbers, or points of interest, A arc set of (directed) street segments. Complicated, asymmetric street intersections might need special treatment with auxiliary vertices and arcs.
- **Integer linear optimization model** as a minimum-cost flow problem. We send 1 indivisible unit of flow from r to s; distances $c_{a,b}$ are treated as per-unit costs.

$$
\begin{array}{lll}
\min & \sum_{(a,b)\in A} c_{a,b} x_{a,b} & \\
\text{s.t.} & f_{\mathbf{x}}(b) = 0 & \text{for all } b \in V \text{ with } b \neq r, s \\
& f_{\mathbf{x}}(r) = 1 & \\
& \mathbf{x} \in \{0,1\}^A &
\end{array}
$$

- Note that this formulation has feasible solutions that contain cycles, but they are never optimal solutions.

Conclusions about black-box optimization software

- Algebraic modeling languages like ZIMPL are convenient
- Integer programming solvers like SCIP, used as a “black box”, can efficiently solve a wide range of optimization problems, up to a certain size
- Not all mathematical models are equal!
  As we observed in the pen plotter problem, it pays off to find “good” formulations. Without explaining what is in the black box, we cannot know what makes a formulation good. *Not always* models with fewer variables are better!
- For very large scale problems, the black box solvers break down:
  - very many variables (matching formulation of pen plotter problem for ≥ 1000 points; formulation of shortest path problem as minimum-cost flow)
  - or very many constraints (all the short-cycle inequalities in the case of the TSP)

By opening the black box, we can solve many problems of much larger scale. These are complicated technologies!
We can deal with many variables by **Column Generation** (Revised Simplex Method, Branch-and-Price) and with many constraints by **Cut Generation** (Cutting Plane Algorithms, Branch-and-Cut), and by **decomposition techniques**.

←

In this class, we instead study fast **combinatorial algorithms** for important basic problems.

Shortest Paths and Rooted Trees

Assumptions:

- We assume there always exists a path from r to any other node v.
- The cost $c_{a,b}$ of each arc (a, b) is a real number; it is allowed to be negative (this has important applications!)

Remark

- We use the words “minimum cost path” and “shortest path” interchangably

To understand the problem better, we consider a **generalization**: Find a minimum cost path from r **to any other vertex** $v \in V$.

- A solution to the generalized problem is a collection of minimum cost paths P_v (from r to v), for $v \in V$ – quite complicated solution data!
- Important observation: **Subpaths of shortest subpaths are shortest.**
- In particular, this holds for special subpaths: $P_w = [r, P_1, v, P_2 = (v, w), w]$.
- Thus, there always is an optimal solution to the generalized problem that takes the form of a **rooted tree** T:

  *Each vertex w that is not the root r has a unique **predecessor** v, i.e., there is a unique arc $(v, w) \in T$ that leads into w. The root r has no predecessor.*

- We can store the whole tree by storing the predecessor $p(w)$ of every node w (**data structure**). Let $p(r) = 0$ (special value).

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**.