# Optimization Theory (DS2) Lecture #8
# $A^*$ Algorithm, Duality, and Minimum Spanning Tree Via Linear Algebra

March 24, 2017

**Abstract**

Today we are going through the $A^*$ algorithm, which doesn't appear in the textbook, duality (which does), and looking at MSTs (which don't) using duality and LA.

*Some text and examples adapted from the paperback edition of* A Gentle Introduction to Optimization*, B. Guerin* et al.

## 1 Reorganizing the Semester

Now that we are about halfway through the semester, we can see a little more clearly the complete picture of where we are headed:

- Lec. 1: Introduction
- Basics of Linear Programming
  - Lec. 2: Examples of Linear, Integer and Graph Problems (formulating formal descriptions of problems)
  - Lec. 3: Three Kinds of Certificates, Standard Equality Form, and a Simplex Iteration
  - Lec. 4: The Heart of the Simplex Algorithm (bases and canonical forms) *In my opinion, this is the hardest week of the semester!*
  - Lec. 5: Sticking a Stake in the Heart of the Simplex Algorithm
- Graph Algorithms
  - Lec. 6: Basic Ideas of Graphs, Dijkstra's Shortest Path First Algorithm
  - Lec. 7: Several Minimum Spanning Tree Algorithms
  - Lec. 8: $A^*$ Algorithm, Duality, and Minimum Spanning Tree Via Linear Algebra

  - Lec. 9: Minimum Weight Bipartite Matching (and more Duality)
  - (Sadly, we are going to skip some important problems such as set cover, and min cut max flow. Semester projects, perhaps?)

- Integer Problems
  - Lec. 10: The Knapsack Problem
  - (Sadly, we are going to skip *many* important problems and solutions here!)

- Computational Complexity
  - Lec. 11: Basic Ideas of Computational Complexity

- Non-Linear Programming (NLP) and Advanced Techniques
  *Warning: These lectures require a little basic calculus! But don't worry too much if you're not up on your derivatives; there will be no required homework on these lectures.*
  - Lec. 12: Basics: Finding a Min/Max of a Function (Derivatives, Gradient Descent, Newton's Method)
  - Lec. 13: Deterministic Exact, Deterministic Heuristic, and Random Algorithms (AI-like methods, more on computational complexity)
  - Lec. 14: Pareto Optimality and Wrapping Up the Semester

## 2 Semester Projects

In the English Wikipedia page on the simplex algorithm, I like the explanation of the history, and I like the images, but the explanation of tableaus and pivots is not quite the way we're solving things, so I wouldn't invest too much time in trying to understand that.

The Japanese page on the simplex algorithm is actually poor. *I would* **gladly** *accept a student rewriting that Wikipedia page as an end-of-term project!* In fact, I encourage it! Any volunteers?

You should expect that your project will take 10-20 hours; if it takes less than that, it's too lightweight.

Types of projects:

1. Create or improve an optimization-related Wikipedia page, such as:
   (a) the Japanese page on simplex
   (b) the Indonesian page on just about anything

   Languages other than English, Japanese or Indonesian require prior approval, and will require an outside evaluator.

2. 3D print a teaching tool I can keep, e.g. a polytope for a *specific* linear problem, or a 2D optimization surface with a max, a min, and a saddle point.

3. A compiler that takes in graphviz and outputs OpenSCAD. Ideally, this would be a plugin for the existing graphviz tools.
4. Find and solve a medium sized optimization problem using one of the techniques from class. This should be a large enough problem that creating the formal definition is non-trivial, and presumably the solution will require the use of some computer method. For this, you may use existing tools, as the emphasis is on the problem.
5. Implement one of the algorithms from class. Your choice of language. This is *not* just learning to use some existing library!
6. Write a 5 page report on advanced optimization (e.g., one of the topics from the last two weeks of the semester).
7. Work on one of the sections from the textbook that we didn't cover.

## 3 Interlude: Most Important Algorithms of the 20th Century

The Society for Industrial and Applied Mathematics (SIAM) named the top ten algorithms of the 20th century. Simplex and Metropolis (arguably the first important Monte Carlo algorithm) are two of them.

There is also the book, *9 Algorithms That Changed the Future*, by John MacCormick, but it's more of a list of problems than algorithms.

Marcos Otero also created a list in reply to a list that I don't actually like. Even Oteros' is more of a list of problems than algorithms.

## 4 $A^*$ Algorithm

I don't know if $A^*$ is the way that Google Maps or Ekitan solves your routing problem to get here in the morning, but it *could*. Some people think of $A^*$ as an extension to or generalization of Dijkstra's shortest path algorithm, but it expands the range of applications so much that I think of it as a separate algorithm.

In Dijkstra's algorithm, the frontier (or "open set") consists of a set of nodes with links headed out from there, the links currently under consideration for addition to our path(s). We select the next link based on

$$\min(d(n) + c_e), n \in F, e \in \text{edges of } n \tag{1}$$

for $d()$ being the distance from the origin to the node $n$, $c_e$ being the cost of edge $e$ from node $n$. That is, we add a new node to the frontier by selecting the next node with the lowest total cost back to our origin.

Putting it another way, we can talk about $f(n)$, for all $n \in \mathrm{Succ}(F)$, the set of possible successor nodes, the candidates to be added to our tree. (This will be a little easier for talking about $A^*$.)

Then we can modify $f(n)$ to be

$$f(n) = g(n) + h(n) \tag{2}$$

where $g(n)$ is the distance to the node, and $h(n)$ is a *heuristic* for estimating the distance from $n$ to the destination.

$A^*$ is fantastically useful when Dijkstra fails us in practice, because it guides the set of things we look at next for efficiency, without ultimately preventing us from finding the solution even in "bad" cases. It is also good when our "graph" is effectively determined dynamically, as long as we can limit the set of things we add at each step.

## 5 Duality, marking a brief return to linear algebra

### 5.1 Examples

There are several cases of duality. Let's look at a couple. The minimization form of the linear program:

$$\min z(\vec{x}) = \vec{c}^{\mathsf{T}}\vec{x} \tag{3}$$

subject to

$$A\vec{x} \geq \vec{b} \tag{4}$$
$$\vec{x} \geq \vec{0} \tag{5}$$

has a *dual* program

$$\max z'(\vec{y}) = \vec{b}^{\mathsf{T}}\vec{y} \tag{6}$$

subject to

$$A^{\mathsf{T}}\vec{y} \leq \vec{c} \tag{7}$$
$$\vec{y} \geq \vec{0}. \tag{8}$$

The superficially similar problem where the $\geq$ in Eq. 4 is replaced with $=$ has a different dual:

$$\min z(\vec{x}) = \vec{c}^{\mathsf{T}}\vec{x} \tag{9}$$

subject to

$$A\vec{x} = \vec{b} \tag{10}$$
$$\vec{x} \geq \vec{0} \tag{11}$$

has the *dual* program

$$\max z'(\vec{y}) = \vec{b}^{\mathsf{T}}\vec{y} \tag{12}$$

subject to

$$A^{\intercal}\vec{y} \le \vec{c} \tag{13}$$

$$\vec{y} \text{ free.} \tag{14}$$

Note the difference in the last lines of the two dual problems.

In the textbook, the primal problem is the maximization problem, called $P$ or $P_{\text{max}}$ and the minimization problem is the dual problem, called $D$ or $P_{\text{min}}$, so technically our first problem above is actually the dual and its dual is the primal.

## 5.2 Constructing the dual

Personally, I found Table 4.1 in the textbook to be cryptic. Let me try to reconstruct it for you. In each block below, replace the "␣" place with the symbol below it, then read across for what it generates in the dual problem. For example, if $A\vec{x} = \vec{b}$, read down to "= constraint" and across to "free variable" to learn that the $y_i$ corresponding to a particular line in the set of constraints having an equals sign results in a new variable in $\vec{y}$ that is a free variable.

| $P(P_{\text{max}})$ | $D(P_{\text{min}})$ |
|---|---|
| $A\vec{x} \,␣\, \vec{b}$ | $\vec{y} \,␣\, \vec{0}$ |
| $\le$ constraint | $\ge 0$ variable |
| $=$ constraint | free variable |
| $\ge$ constraint | $\le 0$ variable |
| $\vec{x} \,␣\, \vec{0}$ | $A^{\intercal}\vec{y} \,␣\, \vec{c}$ |
| $\ge 0$ variable | $\ge$ constraint |
| free variable | $=$ constraint |
| $\le 0$ variable | $\le$ constraint |

## 5.3 A couple of comments

The dual of the dual of a problem is the original problem. (If it's not, you've done something wrong, such as the problem wasn't properly specified in the first place.)

The *weak duality theorem* says that the objective function value for the maximization problem $P$ is always less than or equal to the objective function for the minimization function. If they are equal, then you have found optimal solutions for both. It also implies that if the primal is unbounded, then the dual is infeasible, and vice-versa.

The *strong duality theorem* says that if the primal has a bounded optimal value for the objective function $z(\vec{x})$, then the the dual also has an optimal value for its objective function $z'(\vec{y})$, and in fact, *those optimal values are always the same!*

# 6 Minimum spanning tree via LP (and Duality)

We're going to try this on the graph we used last week, which consists of 9 nodes and 14 links, so it's bit of a large problem to do by hand, but I want you to see the techniques.

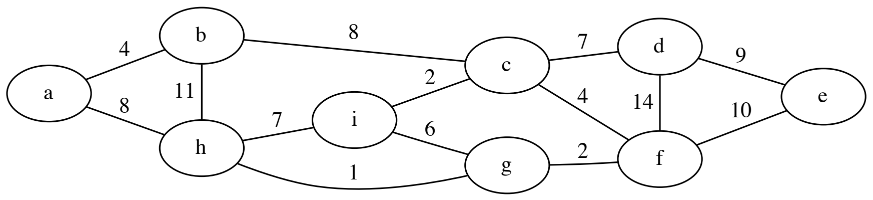

Figure 1: Our figure from Intro to Algorithms.

Although we're not going to cover everything you can do to graph problems using matrices, I also want you to get the idea that matrices can be used in graph problems.

A complete, proper formulation via linear programming would involve an exponential number of constraints. Basically, you write out every proper, non-empty subset of vertices and create a constraint that says that there must be at least one edge connecting a node *inside* the subset with a node *outside* the subset. For a graph with $v = |V|$ vertices, this would be $2^v - 2$ constraints. For our graph in Fig. 1, that would be 510 constraints, and I'm not planning on writing those out by hand! Even for a computer, such exponential growth *very* quickly leaves the ream of the possible.

Instead, I want to demonstrate the principles by creating a hybrid approach here. (I haven't seen this approach written down anywhere before. This is my own formulation, but I have no doubt both that it is correct and has been discovered before.) Basically, we're going to do Boruvka's algorithm where each iteration is done using an LP.

## 6.1 The program

The minimization form of the problem (note that our $x$ and $y$ here are swapped relative to the above):

$$\min z(\vec{x}) = \vec{c}^{\intercal}\vec{x} \tag{15}$$

subject to

$$A\vec{x} \geq \vec{b} \tag{16}$$

$$\vec{x} \geq 0, \vec{x} \text{ integer} \tag{17}$$

where

$$\vec{c}^{\intercal} = (4, 8, 11, 8, 7, 1, 2, 6, 7, 4, 2, 14, 9, 10) \tag{18}$$

$$\vec{x}^{\intercal} = (x_{ab}, x_{ah}, x_{bh}, x_{bc}, x_{hi}, x_{hg}, x_{ic}, x_{ig}, x_{cd}, x_{cf}, x_{gf}, x_{df}, x_{de}, x_{fe}) \tag{19}$$

$$A^{\intercal} = \begin{array}{c|ccccccccc} & a & b & c & d & e & f & g & h & i \\ \hline ab & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ ah & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ bh & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ bc & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ hi & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ hg & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ ic & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ ig & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ cd & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ cf & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ gf & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ df & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ de & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ fe & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{array} \tag{20}$$

$$\vec{b} = \vec{1} \tag{21}$$

Note that $A$ is written in transpose, that was easier here. $A$ actually has one column for each edge and one row for each vector, though we're going to swap them and use it the way it's written in just a minute.

Let's look at this in detail. Every member of the vector $\vec{x}$ should be either 0 or 1, either we use the link or we don't. Then the objective function will end up being the sum of the costs of the links that are in use, which are represented in $\vec{c}$. We constrain each node to having at least one link, which means that $\vec{x}$ has to have a 1 in some entry corresponding to a 1 in each column of $A^{\intercal}$.

Also, note that since you can't use half a link, we are insisting that $\vec{x}$ (or $\vec{y}$) all be integers, but we haven't yet really discussed integer problems. Instead, what we will solve here is called the *linear relaxation* of the integer problem.

On the first iteration, we *should* get the result

$$\vec{x}^{\intercal} = (1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0). \tag{22}$$

That is, our set of links we add in the first iteration of Boruvka's algorithm should be $\{ab, hg, ic, cd, gf, de\}$.

After this iteration, we now have several trees, called a forest:

$$\{\{a, b\}, \{c, d, e, i\}, \{f, g, h\}\}. \tag{23}$$

Now we can rewrite the program looking for the cheapest connections between each tree:

$$\min z(\vec{x}) = \vec{c}^{\mathsf{T}}\vec{x} \tag{24}$$

subject to

$$A\vec{x} \geq \vec{b} \tag{25}$$
$$\vec{x} \geq 0, \vec{x} \text{ integer} \tag{26}$$

where

$$\vec{c}^{\mathsf{T}} = (8, 11, 8, 7, 6, 4, 14, 10) \tag{27}$$
$$\vec{x}^{\mathsf{T}} = (x_{ah}, x_{bh}, x_{bc}, x_{hi}, x_{ig}, x_{cf}, x_{df}, x_{fe}) \tag{28}$$
$$A^{\mathsf{T}} = \begin{array}{c} \\ ah \\ bh \\ bc \\ hi \\ ig \\ cf \\ df \\ fe \end{array} \begin{array}{c} \begin{array}{ccc} ab & cdei & fgh \end{array} \\ \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \end{array} \tag{29}$$

### 6.2 And its dual

Our program has the dual

$$\max z'(\vec{y}) = \vec{b}^{\mathsf{T}}\vec{y} \tag{30}$$

subject to

$$A^{\mathsf{T}}\vec{y} \leq \vec{c} \tag{31}$$
$$\vec{y} \geq \vec{0}, \vec{y} \text{ integer.} \tag{32}$$

## 7 Interlude: the Adjacency and Distance Matrices

Note that the matrix $A$ above is related to but not the same as the *adjacency matrix* for a graph, which in a $v \times v$ square matrix just denoting which nodes are connected to each other, also generally denoted $A$ (for adjacency):

$$A = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{array} \begin{array}{c} \begin{array}{ccccccccc} a & b & c & d & e & f & g & h & i \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array} \tag{33}$$

The distance matrix (sometimes denoted $M$) replaces those simple 1/0 values with the distance (cost):

$$M = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{array} \begin{array}{c} \begin{array}{ccccccccc} a & b & c & d & e & f & g & h & i \end{array} \\ \begin{pmatrix} 0 & 4 & 0 & 0 & 0 & 0 & 0 & 8 & 0 \\ 4 & 0 & 8 & 0 & 0 & 0 & 0 & 11 & 0 \\ 0 & 8 & 0 & 7 & 0 & 4 & 0 & 0 & 2 \\ 0 & 0 & 7 & 0 & 9 & 14 & 0 & 0 & 0 \\ 0 & 0 & 0 & 9 & 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 4 & 14 & 10 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 1 & 6 \\ 8 & 11 & 0 & 0 & 0 & 0 & 1 & 0 & 7 \\ 0 & 0 & 2 & 0 & 0 & 0 & 6 & 7 & 0 \end{pmatrix} \end{array} \tag{34}$$

Note that $A = A^\intercal$ and $M = M^\intercal$, which is true for undirected graphs but not directed ones. In this class, we are limiting ourselves entirely to undirected graphs.

These two matrices have many uses; you can determine, for example, how closely two graphs resemble each other.

Matrix representations of graphs are used for e.g. the original Page Rank algorithm that is the heart of Google's search.

## 8 Homework

See the separate file uploaded to SFS.