Optimization
Homework 2

Dr. Aaron Koehl

October 17, 2017

In this assignment, we will be practicing some of the discrete optimization algorithms which we have been learning about. These algorithms represent the very best way we know how to solve certain optimization problems. The ideal scenario is when your business problem can be put in terms of a problem solved by one of these provably *correct* and *efficient* algorithms. Fortunately, there are many hundreds of problems for which these algorithms are useful.

Due to msba.aaronkoehl.com by Tuesday Oct 24, 11:59pm, to folder `hw2`.

1 Setup and Deliverables

You will need the R packages `optrees`, `igraph`, and `qgraph`. I highly recommend R Studio, and you can install packages using the menu `Tools > Install Packages`. Once installed, you can load packages using, for example, `library(optrees)`.

You should complete this assignment as a single R Markdown Notebook. Your deliverables will be 1) `hw2.Rmd`, and 2) `hw2.PDF` generated from your notebook using Knit to PDF.

2 Matrix Representation

Using R, we can generate a random $n \times n$ adjacency matrix as follows. In the example below, each edge has a 20% chance of occuring. (There are many ways to do this, here is just one.)

```
n <- 1000
d <- runif(n*n)
d[d < 0.80] <- NA
d <- matrix(d,nrow=n,ncol=n)    #reshape the vector
```

```
diag(d) <- NA   # no self-loops
d[upper.tri(d)] = t(d)[upper.tri(d)]   # undirected graphs are symmetric
```

There is a function in the `optrees` package called `Cmat2ArcList` which converts an adjacency matrix to an adjacency list representation. It works well for $n \approx 50$. Unfortunately, for larger N (e.g. $n \approx 1000$) it is inefficient. (Try it and see how long or even *if* it returns.) Instead, we will write our own. Looking at the structure of d above, it is an $n \times n$ adjacency matrix with symmetric edge weights $d_{ij} = d_{ji}$. Write code which will produce, from a matrix d, the sparse matrix ds with appropriate column header names (dimnames) which match the structure as below. You can examine the structure of any object in R using `str()`. ds is similar to an adjacency list in that it is a list of edges, where each row contains a source node, destination node, and edge weight.

```
str(d)
   num [1:1000, 1:1000] NA NA NA 0.861 NA ...
str(ds)
    num [1:99858, 1:3] 1 1 1 1 1 1 1 1 1 1 ...
   - attr(*, "dimnames")=List of 2
   ..$ : NULL
   ..$ : chr [1:3] "head" "tail" "weight"
head(ds)
        head tail    weight
   [1,]    1   15 0.9205357
   [2,]    1   16 0.9938016
   [3,]    1   29 0.9480700
```

```
AdjMatrix2List <- function (d) { ... }
```

Note: *This is a general R coding exercise.* Please limit yourself to basic R constructs, such as `for` loops, `dimnames`, `list`, `c()`, and `matrix`, rather than searching for some other package or `igraph` function to solve this problem.

3 Euclidean Minimum Spanning Tree

Your new function (above) may be useful. In this exercise, you will create a Euclidean Minimum Spanning Tree (E-MST) on a set of random (X, Y) coordinates. Think of each of these coordinates as locations for a facility, and you want to build and visualize a road network that connects all of these facilities at minimum cost. (Note: We could easily use longitude and latitude if the points are close enough together, within the same state or so. Beyond that, we have to take into account the curvature of the earth to get accurate distances.)

The Euclidean distance between any two points $(x_1, y_1), (x_2, y_2)$ is given by

$$d_{ij} = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$$

We can generate a random set of 50 facility locations as follows. When you are finished with this exercise, you might want to try $n = 1000$.

```
n <- 50
x <- round(runif(n)*1000)
y <- round(runif(n)*1000)
plot(x,y,pch=16)
```

1. Create an adjacency matrix d by calculating the Euclidean distance between *every pair* of points $(x_i, y_i), (x_j, y_j)$. This will be a **complete** graph. (Why?)
2. Using your function `AdjMatrix2List`, create an adjacency list from d called ds. This *just so happens* to be the format needed for `msTreeKruskal` in `optrees`.
3. Calculate the minimum spanning tree using Kruskal or Prim. You don't need to implement these algorithms on your own. You can store the result in the variable `ds.mst`.

Here is an example of the structure produced by the MST algorithm, for a different n. Importantly, it contains tree.nodes and tree.arcs. You won't need the stages variables.

```
ds.mst <- msTreePrim(1:n, ds)    # nodes, arcs
str(ds.mst)
List of 4
   $ tree.nodes : num [1:25] 1 2 6 7 8 13 12 17 16 21 ...
   $ tree.arcs  : num [1:24, 1:3] 1 1 6 7 8 13 12 17 16 8 ...
   ..- attr(*, "dimnames")=List of 2
   .. ..$ : NULL
   .. ..$ : chr [1:3] "ept1" "ept2" "weight"
   $ stages     : num 24
   $ stages.arcs: num [1:24] 1 2 3 4 5 6 7 8 9 10 ...
```

4. The last step is to produce a plot visualizing your minimum spanning tree, using `ds.mst$tree.arcs`. Using the `segments` function, draw straight line segments on your `plot(x,y)` corresponding to the edges in the minimum spanning tree. `ept1` is the source node, and `ept2` is the destination node; you can safely ignore the weights.

   ```
   ds.mst <- msTreePrim(1:n, ds)    # nodes, arcs

   # Write this function
   ```

```
plot.mst <- function (arcList) { ... }

# After computing the MST, test as follows:
plot(x,y,pch=16)
plot.mst (ds.mst$tree.arcs)
```

Take a moment. Admire your handiwork. This is cool.

4 Hostile Agents

An intelligence service has n agents in a non-friendly country. Each agent knows some of the other agents and has in place procedures for arranging a rendezvous with anyone he or she knows. For each such possible rendezvous, say between agents i and j, any message passed between these agents will fall into hostile hands with a certain probability p_{ij}. The government wants to transmit a confidential message among all the agents while maximizing the total probability that no message is intercepted.

1. Specify this problem as a graph optimization problem.
2. How do you structure your inputs to this problem?
3. Which algorithm do you use to solve it?
4. What is the computational efficiency of your chosen solution?

5 Project Scheduling

The objective of project scheduling is to plan and control a project effectively. One common objective is to determine, given a set of tasks that need to be completed, a *schedule* of start and finish times for individual tasks that lead to the earliest completion time for an overall project. (This sounds like an optimization problem!)

Given these optimized times as a starting point, we can then use probability to estimate the likelihood that a project will be completed within a certain time period. (We'll save this one for later. If you are interested now, you can read about PERT/CPM.)

Further, with the aid of our model, we can perform a sensitivity analysis to see how delays in certain activities will affect the overall completion time of a project.

Lastly, during project execution we can compare our schedule against actual completion times to determine whether the project is proceeding at an acceptable rate.

5.1 1693 Analytics™

1693 Analytics is a small manufacturer about to prototype and market test a new product which involves three major activities: manufacturing, training staff and vendor representatives, and marketing. After a set of proposed specifications are reviewed and a design determined, materials are purchased and prototypes are manufactured. The prototypes are then tested and analyzed by trained testers, who then provide feedback. Based on their input, refinements to the prototype are incorporated and an initial production run is scheduled.

Once the production run is underway, the company staff and sales force undergo full-scale training. Advertising will happen in two phases. First, a small group works with the design team to produce a preproduction advertising campaign, drumming up interest on social media, and testing the market. With the final design revisions completed, the product is positioned and a full-scale marketing campaign is launched. The entire project is concluded when manufacturing, training, and advertising activities are underway.

	Task	Description
Manufacturing	A	Prototype model design
	B	Purchase of materials
	C	Manufacture of prototypes
	D	Revision of design
	E	Initial production run
Training	F	Staff training
	G	Staff input on prototypes
	H	Sales force training
Advertising	I	Preproduction advertising campaign
	J	Post-redesign advertising campaign

Project scheduling is a great application of shortest paths. In project management, consider a task A which must be completed before some task B. We can represent all such tasks in a directed acyclic graph (DAG), such that if A happens before B, we include a directed edge from A to B. Here are a set of predecessors and estimated completion times (in days) for the 1693 Analytics project above.

Task	Predecessor(s)	Estimated Completion Time
A	-	90
B	A	15
C	B	5
D	G	20
E	D	21
F	A	25
G	C,F	14
H	D	28
I	A	30
J	D,I	45

Your first task is to represent this scheduling network in R. Associate a completion time with each node. Here's an example to get you started.

```
s.labels <- c('a','b')
s.nodes <- c('90','15')
```

5.2 Enter, the Management.

Management at 1693 Analytics would like to schedule the activities of the new project so they will be completed in minimal time. In particular, we wish to know:

1. The earliest completion date for the project if it commences on Nov 1, 2017.
2. The earliest and latest *start* times for each task which will not alter the completion date.
3. The earliest and latest *finish* times for each activity that will not alter the completion date.
4. The list of tasks which *must* adhere to a rigid schedule so as not to delay the overall project, as well as a list of those tasks which have more flexibility.

In order to accomodate the above, we must calculate four quantities for each task: the earliest start times (ES), earliest finish times (EF), latest start times (LS), and latest finish times (LF). Additionally, we want to calculate the *slack* in our schedule—this represents the amount of scheduling flexibility we have in days for each task. Tasks which have no slack are said to be on the **critical path**. Any amount of delay of a task on the critical path delays the overall project by that amount. Similarly, given appropriate monetary resources, sometimes it is possible to *crash* tasks on the critical path by adding overtime resources to a task and thereby reducing the time estimates. [1] We would not want to crash a task that contains slack.

5.3 A Linear Programming Problem

Given a set of estimated durations for tasks and a set of predecessor relationships, PERT/CPM networks (as in the DAG we have modeled) can be modeled as linear programs, with a simple objective function and one constraint per edge.

Let $(X_A \dots X_J)$ represent the start times of each activity. For each edge, we include one constraint. Here's an example for task G:

[1] For some good required reading in project management, a role in which you may find yourself fast-tracked by your analytics degrees, see *The Mythical Man Month* by Frederick Brooks. One big takeaway: adding manpower to a late software project makes it *later*. This is also true for analytics projects, and it's due in large part to the amount of communication.

$$X_G \geq X_C + 5$$
$$X_G \geq X_F + 25$$

This effectively says that the start time for an activity must be greater than the start time for the predecessor activity, plus the predecessor's duration.

The **objective function** would be to minimize $\sum X_i$, e.g., $min X_A + X_B + X_C + \ldots + X_J$

Inequalities like the above constraint, which have the simple pattern of exactly two variables and one quantity, are said to be *difference constraints*. We can rewrite them as follows:

$$X_G - X_C \leq 5$$

Any system of difference constraints has a special structure which can be solved as a shortest path problem! Since we already have a graph, and we already have an efficient shortest path algorithm from the `optrees` package, let's use our specialized shortest path algorithms instead.

5.4 Earliest Start Times (ES)

In terms of graph theory, the critical path is the longest path through a directed acyclic graph. As we picked up in lecture, we can convert the longest (maximum) path problem into the shortest (minimum) path problem by multiplying the edge weights by -1, and using a suitable shortest path problem that works with negative edge weights.

To calculate the earliest start times for each node, you will need to solve the longest path problem, starting at source node A, with negative edge weights. The distances output from the algorithm are your earliest start times. Use the shortest path from the `optrees` package. The output is very similar to the minimum spanning tree algorithms, and the vector of interest is called `$distances`.

The key is to choose your edge weights carefully. The outgoing edge weight for any node is the source node's duration, which you happen to have stored in `s.nodes`. For example, weights $w_{AB} = w_{AF} = w_{AI} = 90$.

5.5 Earliest Overall Project Completion Time

This is given by the maximum of all the earliest finish times. It should be reported in days, and a date in the future relative to Nov 1.

5.6 Earliest Finish Times

After computing ES, compute the earliest finish times for each task, which is the start time plus that task's duration.

$$EF = ES + \text{node duration}$$

.

5.7 Latest Finish Times

The latest finish times are computed on the *transpose* of the graph. That is, for each edge from A to B, we reverse its direction and instead have an edge from B to A. Recall that the weight of an outgoing edge will be the duration from the *source* node, so you will have to rebuild your edge weights. We are still looking for finish times along the (longest) critical path, so again, we flip the edge weights to negative, and solve the shortest path problem.

However, there is one additional consideration. Which of the three terminal nodes E, H, or J do you use as the starting node? You can run the algorithm once for each node, or you can just add an 11th node—a dummy node with duration 0 which has outgoing edges to E, H, and J.

5.8 Latest Start Times

As before, the latest start times are computed by subtracting the node duration from the latest finish times.

$$LS = LF - \text{node duration}$$

5.9 Slack

Once you have computed the above quantities, the slack along the critical path is easy to calculate. It's either $LF - EF$ or $LS - ES$.

1. Which tasks have scheduling flexibility? (positive slack)
2. Which tasks are *on the critical path*? (slack=0)

5.10 Gantt Chart (Extra)

Yes, there are packages that do this quite well. Using your finish times, you can easily visualize them in a Gantt chart. Here is a line of code to get your creative juices started. This part is optional.

```
# s.lf   Finish times for nodes 1..N
# s.n    Node durations
# s.desc Short task description
barplot(rbind(s.lf-s.n,s.n),horiz=TRUE,col=c("white","gray"),
        border=0,names.arg = s.desc,las=1)
```

Or just enough to whet your appetite..

```
# s.lf   Finish times for nodes 1..N
# s.n    Node durations
```

```r
# s.desc Short task description
library(timevis)      # Install this first
gnt <- data.frame(id = 1:10, content=s.desc, start = s.lf-s.n, end = s.lf)
timevis(gnt)
```