Traveling Salesperson Problem

Prof. Richey and Prof. Wright

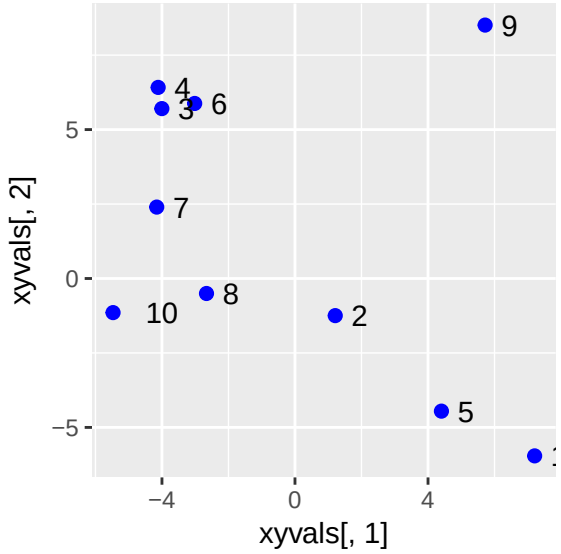
May 7, 2018

The Traveling Salesperson Problem is a classic problem in combinatorial optimization. Given N points in a plane, we want to find the shortest path that visits all N points and returns to the starting point. A nice introduction to the problem, along with the many ways to solve it can be found at the Wikipedia page https://en.wikipedia.org/wiki/Travelling_salesman_problem.

Implementation Setup

We will create N random points in a plane and plot them with ggplot. Note that the numbers plotted next to a point is the *row index* of that point in the matrix `xyvals`.

```
N <- 10
xyvals <- matrix(runif(2*N, -10, 10), nrow=N)
library(ggplot2)
ggplot(data=NULL, aes(x=xyvals[,1], y=xyvals[,2])) +
  geom_point(size=2, color="blue") +
  geom_text(aes(label=1:N),hjust=-1)
```



Tours, Permutations, and Distances

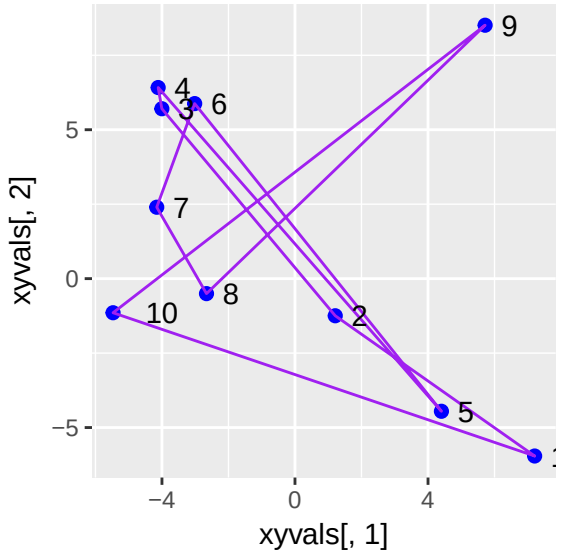
We want to find a path (also called a *tour*) that visits each point once and returns to where we started. One way to represent the tour is a permutation

$$t_1, t_2, \ldots, t_N$$

of the values $1, 2, \ldots, N$. For example, $1, 2, \ldots, N$ would be a tour. It's probably not the shortest tour:

```
tour <- 1:N
ggplot(data=NULL, aes(x=xyvals[,1], y=xyvals[,2])) +
  geom_point(size=2, color="blue") +
```

```
  geom_polygon(color="purple", fill=NA) +
  geom_text(aes(label=1:N),hjust=-1)
```



We need a function that computes the length of a tour. Since a tour is made up out of a number of segments, we will start with the distance between two points.

```
distPair <- function(i, j, tour){
  x1 <- xyvals[tour[i],1]
  # ...
  # ... finish the function here
  # ...
}
```

Check:

```
distPair(1, 2, tour)
```

```
## [1] 13.39051
```

Using this, we can calculate the distance around an entire tour.

```
distTour <- function(tour){
  tot <- 0
  N <- length(tour)
  # ... finish the function here ...
  # ... don't forget to go home!
  tot
}
```

Check:

```
trivialTour <- 1:N
distTour(trivialTour)
```

```
## [1] 104.5243
```

How about a “random” tour?

```
randTour <- sample(1:N, N, rep=FALSE)
randTour
```

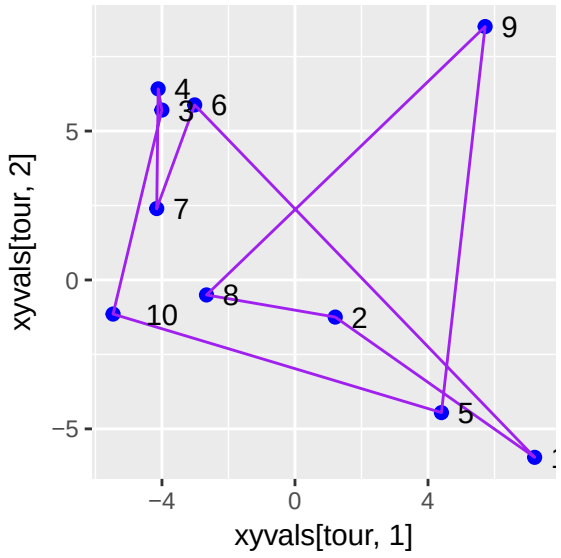
```
##  [1]  7  4  3 10  5  9  8  2  1  6
```

```
distTour(randTour)
```

```
## [1] 103.6257
```

We will draw the random tour, but first we will write a function to make future plots easier.

```
plotTour <- function(tour, ptSize=2){
  ggplot(data=NULL, aes(x=xyvals[tour,1], y=xyvals[tour,2])) +
    geom_point(size=ptSize, color="blue") +
    geom_polygon(color="purple",fill=NA)  +
    geom_text(aes(label=tour),hjust=-1)  # prints the index of the point in the vector xyvals
}
plotTour(randTour)
```



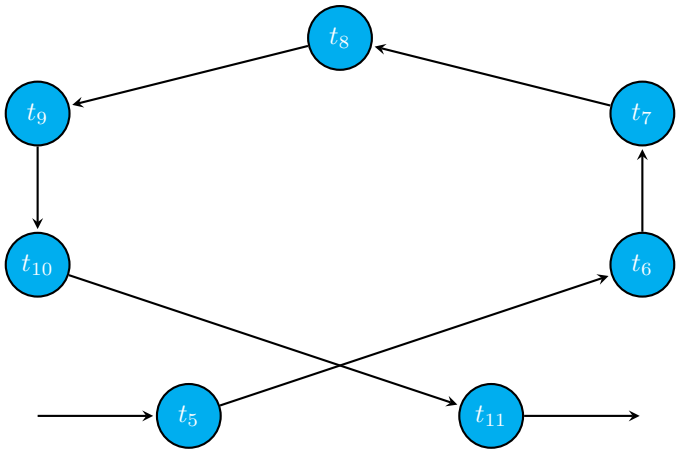
Simulated annealing and the TSP

Simulated annealing can find an approximate solution to the Traveling Salesperson Problem. By now, we know that for any simulated annealing scenario, we need the following:

- A well-defined set of states. In this case, all possible tours form the state space. Notice that there are $(N-1)!$ different tours (the first city can be fixed).
- A function to minimize. We have the total distance function.
- A natural proposal transition. This is a little tricky.

A proposal transition

Consider a section of a tour.



This tour is represented by the ordering:

$$t_1, \ldots, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}, \ldots, t_N$$

A simple proposal transition selects two locations, say 5 and 11, and makes a transition to a new state by simply reversing the order between the two locations.

That is, the proposed transition becomes

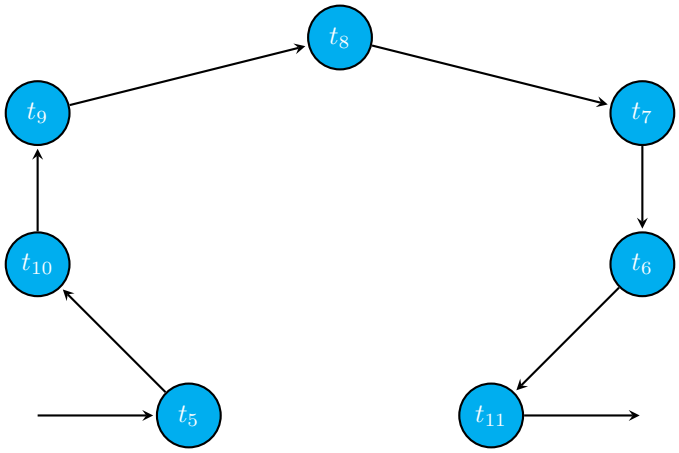
$$t_1, \ldots, t_5, t_{10}, t_7, t_8, t_9, t_6, t_{11}, \ldots, t_N$$

Geometrically, this uncrosses (or perhaps crosses) part of the path. Notice that:

- Before, $t_5 \to t_6$. After, $t_5 \to t_{10}$.
- Before, $t_{10} \to t_{11}$. After, $t_6 \to t_{11}$.
- All the directions from t_{10} to t_6 are reversed:

$$t_{10} \to t_9 \to t_8 \to t_7 \to t_6$$

The result is:



Generally speaking, the steps are:

- Pick two sites $i < j$ at random. Note: in the example above, we picked $i = 6$ and $j = 10$.
- We need to pay special attention the case where $i = 1$ or $j = N$. Don't allow both $i = 1$ and $j = N$.
- Reverse the positions tour from t_i to t_j. In our case, this means the new proposed tour is:

$$t_1, \ldots, t_{i-1}, t_i, t_{j-1}, t_{j-2}, \ldots, t_{i+2}, t_{i+1}, t_j, t_{j+1}, \ldots, t_N$$

- Compute the new total distance of the tour. As it turns out, this is really easy. Note between the before and after tours, almost all of the pairwise distances are unchanged. The only things that change between the two are:
  - The distances of $t_{i-1} \to t_i$ and $t_j \to t_{j+1}$ are removed.
  - The distances of $t_{i-1} \to t_j$ and $t_i \to t_{j+1}$ are added.
  - **Note:** If $i = 1$, replace $i - 1$ with N. If $j = N$, replace $j + 1$ with 1.
  - All the other distances are unchanged. (For the segments in the middle of the reversed section, only the direction changes.)

Hence, if D_{curr} is the current distance, you can compute the proposed distance D_{prop} simply by subtracting two distances and adding two distances. This can quite helpful if N is large. You do not have to recompute the total distance each time!

Strategy

To attack the TSP with simulated annealing, think about how to implement the proposal transition. While you are doing so, keep track of the key sites i and j, and the associated pairwise distances that contribute to the changes in the total distance. This means you will not directly implement a "proposal" function as we have done before, because you need to keep track of i and j in order to efficiently calculate the total distance.

A general outline of your code:

Start by implementing the two key function `distPair` and `distTour` described above.

Setup for simulated anealing:

```
N <- Some large value
# coordinates are randomly selected values between -10 and 10
xyvals <- matrix(runif(2*N,-10,10), nrow=N)
```

A quick look:

```
trivialTour <- 1:N
plotTour()
```

Compute the initial distance. This is the only time you need to use the `distTour` function.

```
dist0 <- distTour(trivialTour)
```

Set up the simulated annealing:

```
M <- number of steps
sig2 <- some sigma2 value
decFac <- decrease factor
```

The algorithm is best implemented as one large loop containing sections for:

- the proposal transtion
- making the move to a new state
- annealing

The outline for the loop is something like this:

```
for(m in 1:M){
  # Generate two positions between 1 and N such that pos0 < pos1, and
  # make sure it is not the case that pos0 = 1 and pos1 = N.
  # ...
  # ...
  # Do nothing if the positions are both at the endpoints.
```

```
  if(pos0 == 1 & pos1 == N){
    next # this causes R to return to the top of the loop and proceed with the next value of m
  }
  # Keep track of the distances lost and gained.
  # distLost <- ...
  # distGain <- ...
  #
  deltaDist <- distGain - distLost
  # Compute a new distance.
  dist1 <- dist0 + deltaDist
  # If you want, print out the distances every 100 steps;
  #   this is useful as a check but slows things down.
  # if(M %% 100 == 0){
  #
  #   print(dist1)
  # }
  # Traverse the tour between pos0 and pos1 to create the proposed tour.
  # Note: R is very good at slicing vectors -- you can do this in one line.
  # ....
  # ....
  # Compute rho
  rho <- exp(-deltaDist/sig2)
  if(runif(1,0,1) < rho){
    # Accept the proposed transition
    #
  }else{
    # Reject the proposed transition
    #
  }
  sig2 <- sig2*decFac
}
# Display results
plotTour()
dist0
```

It helps to pick reasonably large value of M and run this block several times. Keep track of your original distance and note how much of an improvement you see.

TSP Project

Due Monday, May 14

Use simulated annealing to find an approximate solution to the Traveling Salesperson Problem. Put all your work in a new R Markdown file. Set things up so you can easily vary the number of points N. See how large you can make N and still find a good tour in reasonable amount of time.

Then implement at least one of the following extensions of the TSP:

1. Suppose north-south travel is more expensive than east-west travel. Modify your distance function so that the difference between y-values is multiplied by some constant c. You might think of this new function as a "cost" function rather than a distance function. This is now the function to minimize along the tour.
2. Suppose that the points lie on either side of a border, and crossing the border involves an additional cost. For example, you might let the y-axis be the border. If an edge crosses the y-axis, then its "cost"

equals its distance plus some constant c. The cost of traveling along an edge that doesn't cross the border is simply the distance along the edge. The cost function is now the function to minimize along the tour.

What to turn in

Turn in a HTML or PDF file knit using R Markdown. Make sure that you clearly explain your work, and include the items mentioned in the grading rubric below. Your goal should be to communicate your work to another person (e.g., another student at your level who is not in this course).

Grading Rubric

Your notebook will be graded on a scale of 0 to 4, according to the following rubric.

4. Problems and goals are clearly stated, including relevant definitions or parameters. Computations are complete; code runs and is clearly explained. Conclusions are clearly stated and backed up by sufficient computational evidence. Limitations of the methodology, extensions for future work, and/or conjectures are discussed. Notebook is well-formatted and easy to read.

3. Problems and goals are stated well, though relevant definitions or parameters may be missing. Computations are mostly complete; code runs, but explanation is weak. Conclusions are unclear or not well justified. Insufficient discussion of limitations, extensions, and/or conjectures.

2. Statement of problem or goal is unclear. Computations are incomplete; explanation is ambiguous. Code may produce errors when run. Conclusions are possibly correct, but not justified. Little or no discussion of limitations, extensions, and/or conjectures. Notebook is difficult to read.

1. Serious misunderstanding of the problem or goal. Computation is inadequate for the task at hand. Work is not clearly explained. No discussion of limitations, extensions, and/or conjectures. Notebook is difficult to read.

0. Notebook is not turned in.