CS 486 - A2

Alexander Maguire
amaguire@uwaterloo.ca
20396195

January 27, 2015

Problem 1a:

Paths are expanded in the order:
S H K C A B D M E N G

Problem 1a i:

Yes, $h(state)$ is admissible since it never overestimates the cost of getting to g.

Problem 1a ii:

S H K C A B D M G

Problem 1a iii:

S H K C F P Q R T G
My tie-breaking strategy was to prefer nodes with a lower heuristic.

Problem 2a:

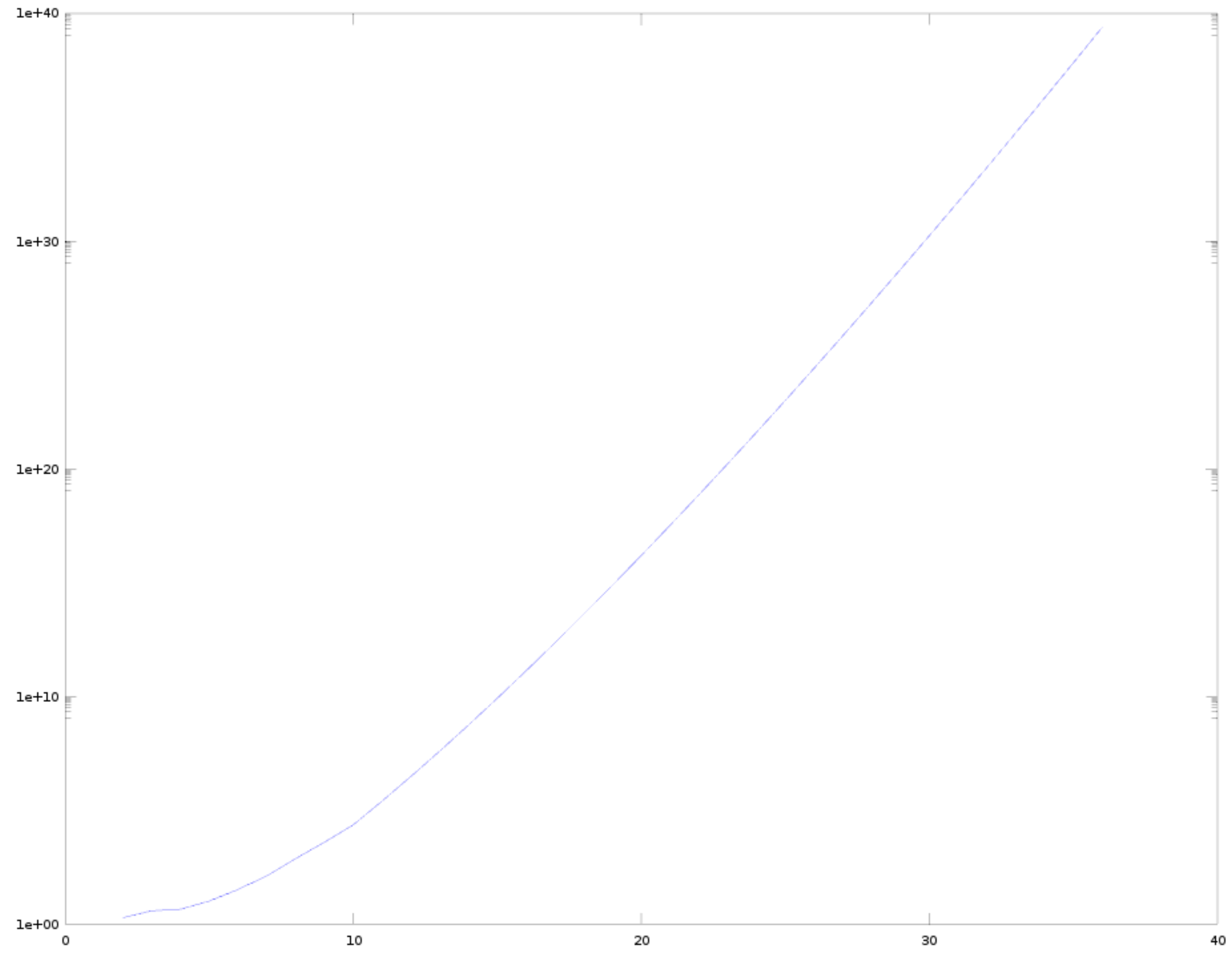
Consider a node to be a partial tour, with the first city always being A. Neighbors can be generated by appending an unvisited city to the end of the partial tour, with a goal state being one node with n cities visited, where n is the number of cities. A cost function between any two nodes is simply the distance of moving from the current city to the unvisited city, with an extra cost for completing the tour (to get back to the starting city).

Problem 2b:

After failing to come up with my own heuristic, some googling lead to the idea of using a minimal spanning tree of the unvisited cities as a heuristic. The page also described adding the costs from the current city to the nearest unvisited city, as well as the cost from the starting node to the nearest unvisited city. However, when I attempted to implement this, my heuristic would sometimes overestimate the total tour cost, resulting in incorrect solutions.

My final heuristic was simply to use the MST of the unvisited cities; this worked well.

Problem 2c:



Average neighbors expanded per number of cities

The number of generated nodes for each city progressed roughly exponentially; after 10 cities, however, my search became too slow, so everything afterwards is extrapolated by multiplying the previous number of nodes by the new number of cities, since this is what is to be expected in the worst case.

This extrapolation puts the number of generated nodes for 36 cities at a whopping 2.4536e+39. If every computer on Earth were running at 2.5GHz non-stop on this problem, it would take approximately 31 trillion years to complete.

Problem 3a:

To generate the neighbors of a node, I randomly swapped three cities in a tour (ensuring that the first city was never swapped. This is guaranteed to preserve a tour, since swapping any three cities in a tour will result in another tour.

Problem 3b:

My annealing schedule was made up of five parameters:

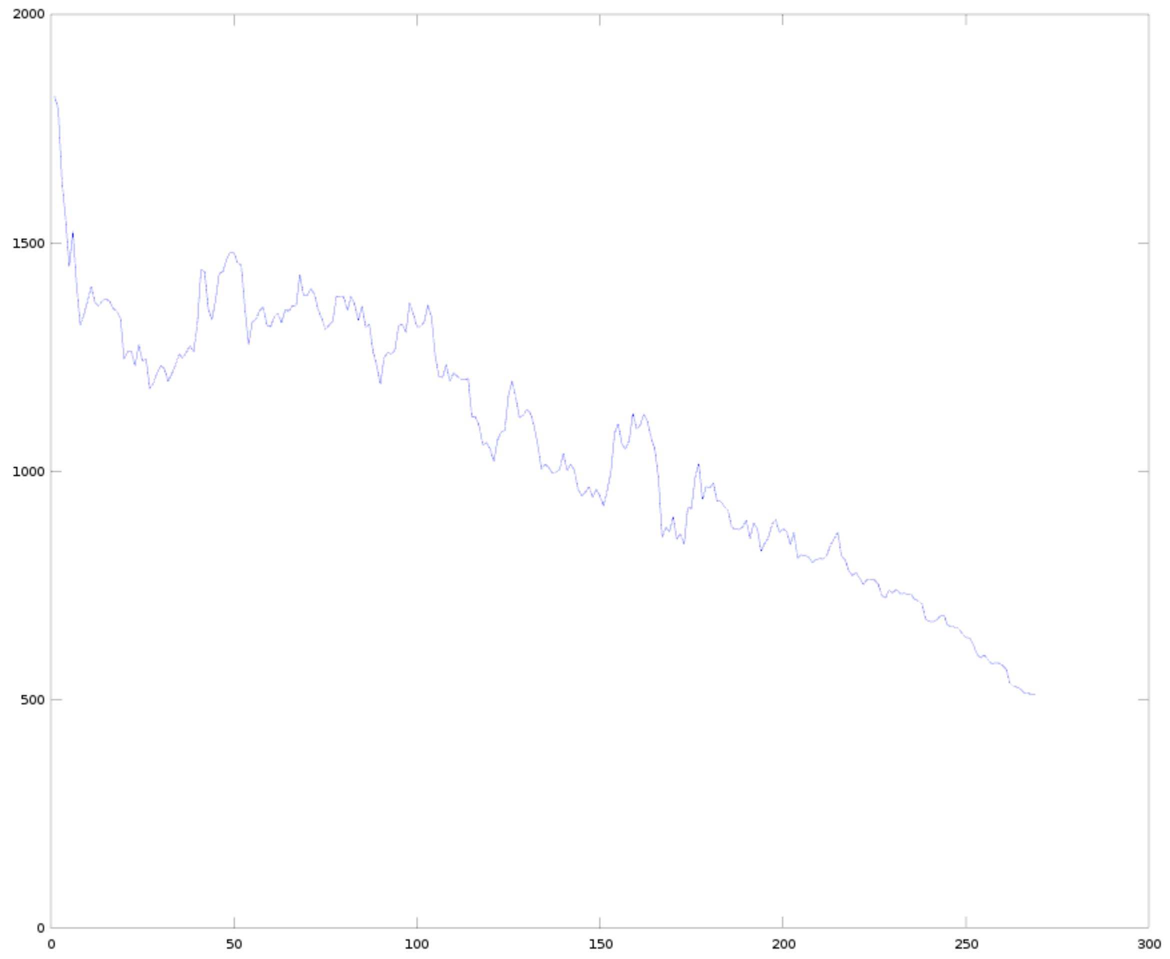
- $\alpha =$ **initTemp**: the starting temperature
- $\Omega =$ **minTemp**: the stopping temperature
- $\beta =$ **coolingFactor**: the rate at which the temperature is cooled
- $\Gamma =$ **itersPerTemp**: how many iterations the temperature remains constant
- $g =$ **isGeometric**: whether the cooling is geometric or arithmetic

Having taken a class on local-search last semester, I reused three of my best cooling schedules:

1. $\alpha = 13 \quad \Omega = 0.01 \quad \beta = 0.01 \quad \Gamma = 3 \quad g = 0$
2. $\alpha = 15 \quad \Omega = 0.0002 \quad \beta = 0.93 \quad \Gamma = 150 \quad g = 1$
3. $\alpha = 12 \quad \Omega = 0.00001 \quad \beta = 0.93 \quad \Gamma = 250 \quad g = 1$

And I found that (1) worked the best for small ($n < 5$) datasets, (2) for medium ($5 \leq n < 15$) datasets, and (3) for large($15 \leq n$) datasets. However, since (3) performed quite well on all datasets, I decided to use that as my chosen annealing schedule.

Problem 3c:



SA tour cost over time

The best solution I found was “AJQZXVPOADECACSUFGBAJHAGAERMAAAFTYIA-IAHLDNABWK”, with tour cost 511.12.

Problem 3d:

Simulated annealing is not complete, since it is not guaranteed to find the best answer (merely a *good* answer).

Problem 3e:

Simulated annealing is also not optimal, since it’s ability to search worse neighborhoods – while good for escaping local maxima – will usually jump to a neighborhood with significantly worse options.