TSP

A survey and profile

Dhasharath Shrivathsa

December 9, 2016

The problem

The Travelling salesman problem is a simple problem.
Answering it is hard

The question is?

The Euclidean Symmetric Travelling Problem is the most commonly solved.

- Asymmetric exists too

"Given $N \in \mathbb{I}$ cities in $D \in \mathbb{I}$ dimensions, find the optimal tour where the distance metric is the Euclidean norm."

or

$$
\begin{aligned}
G &= (V, E) \\
T &= (V \in G_V) \\
\text{Minimize} \sum_{n=0}^{|T_V|} norm(T_V[n] - T_V[n+1]) & \qquad (1) \\
\text{subject to } T_V \equiv G_V & \qquad (2)
\end{aligned}
$$

Context

The ShopBot needs to solve (almost) this every day

Approaches

Algorithim	**Complexity**	**Approximation**
Brute-force	$O(V!)$	1
Held-Karp	$O(V^2 2^V)$	1
Greedy	$O(V\lambda)$, not all TSP's	1.25 (for $D = 2$)
Christofides	$O(EV^{2.376})$	1.5
Ant colony (iterative)	$O(V\lambda)$	Depends on G
m-guillotine subdivision	$(O\left(n^{O(m)}\right)$	$(1 + \frac{c}{m})$
PTAS	$O\left(V\,(\log V)^{\left(O\left(c\sqrt{D}\right)\right)\,D-1}\right)$	$1 + \frac{1}{c}$

Oh my, what do I choose for my algorithm, given my space/time constraints? (Probably the PTAS)

Some assumptions

- Approximate solutions are OK, but we want a lower bound
- The time an algorithm takes to complete (`%%timeit`, `%%time`) is a reasonable proxy to how complex an algorithm is
- Space complexity doesn't matter (otherwise Greedy starts to look like $O\left(V^2\lambda\log\lambda\right)$)

Exact algorithm

Brute-force

English explanation

Consider all tours of a graph → Sum their weights → take the argmin. How many tours are there of a graph? $O(V!)$

Python

```
import itertools as it
import networkx as nx
import numpy as np

def brute_force(G):
  tours = list(it.permutations(G.nodes())) #O(V!)
  costs = []
  for tour in tours:
  cost = 0
  for n1, n2 in zip(tour, tour[1:]): #O(V)
    cost += G[n1][n2]['weight']
  costs.append(cost)
  return tours[np.argmin(costs)]
```

Exact algorithm

Brute-force

We can reason about the approximation factor of 1 by looking how it improves iterativley with a slight tweak to the code, a stop after evaluating n permutations out of $V!$

(figure to come)

Exact algorithm

Bellman-Held-Karp

English explanation

TBD

Mathematical formulation

$$\text{decision variable } x_E == 1 \text{ if } x_E \text{ is on the optimal tour.}$$

$$\text{Minimize: } \sum_E W_E \cdot x_E \tag{3}$$

$$\text{Subject to: } \forall_V, \sum_{E \subset adj(V)} x_E = 2 \tag{4}$$

$$\forall G' \subset G, \sum_{E = V(G') \text{ to } V(G)} x_E \geq 2 \tag{5}$$

English explanation

Pick an arbitrary node $\rightarrow$ find the lowest-weight edge $\rightarrow$ travel down the edge if it leads to a node that the agent has not "visited" before. Terminate once all edges are visited.

Algorithm

Start at a node $n \in V$, go down the edge $\forall m, min_{W\,mn} adj(n)$ to pick a new node m, subject to $m \notin$ previous n.

Approximation algorithm

Greedy

Let's look at some data to see how the algorithm performs. This graph has the optimal tour cost of 2579.

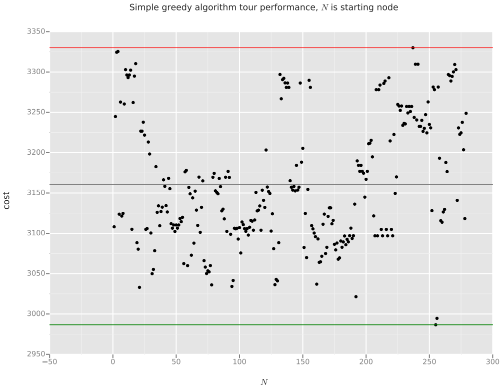

Figure: N is the starting node, the red, grey, and green lines denote worst, best and average case

Approximation algorithm

Greedy

It's distribution of costs is:

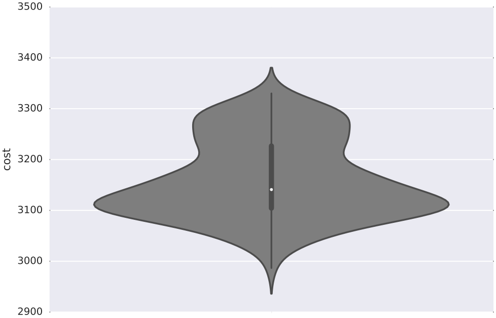

Figure: Relative frequency vs cost (KDE interpolation)

Approximation algorithm

Greedy

And here's how the agent progresses through the greedy approach.

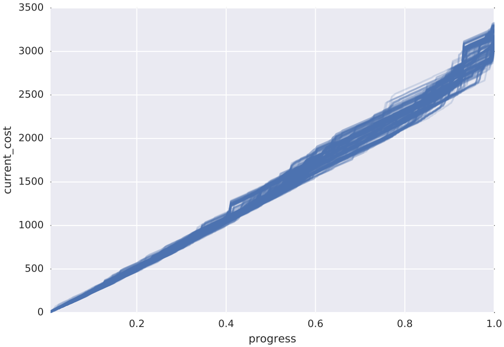

Figure: How the cost for the agent differs in time, each trace is a different starting node N.

TODO

run analysis of greedy agent average performance vs optimal tour to substantiate 1.25 approximation factor, rerun large graph analysis with new visualization code, build visualizations for m-guillotine subdivisions and provide a broken-down explination of the algorithim, write ant trail algorithim and run visualization code.