The Distributed Retired Traveling Salesman

Applying Integer Programming Concepts to Solve a Harder Version of the Traveling Salesman Problem

Daniel Seita and Lucky Zhang

Department of Computer Science, Williams College

May 21, 2014

Problem 1 (Traveling Salesman Problem).

Given a list of cities and their pairwise distances, what is the shortest possible route that visits each city exactly once and returns to the starting city?

- One of the most studied problems in all of computer science.
- Decision version (does there exist such a cycle with cost $\leq B$) is NP-complete[1].

[1] Recall from CS 256: NP-complete means (1) in NP and (2) NP-hard.

Our problem is harder!

Problem 2 (Distributed Retired TSP).

We have the following (real-world) problem:

- *A list of cities* $a_1, a_2, \ldots, a_n$
- *A starting date* d_{start} *and ending date* d_{end}.

The goal is to find **the minimum cost** *of any a series of flights* $f_1, \ldots, f_m$ *such that we visit every city* **at least once**. *Hence, "Retired" TSP because you'd need to be wealthy and retired, and "Distributed" because we plan on using multiple machines to speed up algorithm.*

Our problem is harder!

Problem 2 (Distributed Retired TSP).

We have the following (real-world) problem:

- *A list of cities* $a_1, a_2, \ldots, a_n$
- *A starting date* d_{start} *and ending date* d_{end}*.*

The goal is to find **the minimum cost** *of any a series of flights* $f_1, \ldots, f_m$ *such that we visit every city* **at least once**. *Hence, "Retired" TSP because you'd need to be wealthy and retired, and "Distributed" because we plan on using multiple machines to speed up algorithm.*

Biggest challenge: for any two cities, costs are no longer fixed!

We use **binary integer programming** to help us solve Retired TSP.

- minimize $\mathbf{c}^{\mathrm{T}}\mathbf{x}$
- subject to $A\mathbf{x} \geq \mathbf{b}$,
- and $x_i \in \{0, 1\}$

We use **binary integer programming** to help us solve Retired TSP.

- minimize $\mathbf{c}^{\mathrm{T}}\mathbf{x}$
- subject to $A\mathbf{x} \geq \mathbf{b}$,
- and $x_i \in \{0, 1\}$

Here is how we **formulated** Retired TSP as a binary IP problem:

- Let c_{ijt} denote the *minimum cost* of flying from city i to j on day t.

Our key variables:

$$x_{ijt} = \begin{cases} 1 & \text{if we go from cities } i \text{ to } j \text{ on day } t, \\ 0 & \text{otherwise.} \end{cases}$$

Here are our first three constraints:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijt} \leq 1 \text{ for all } t \in \{1, 2, \ldots, m\}. \tag{1}$$

Ensures we have at most one flight per day.

Here are our first three constraints:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijt} \leq 1 \text{ for all } t \in \{1, 2, \ldots, m\}. \tag{1}$$

Ensures we have at most one flight per day.

$$\sum_{t=1}^{m}\sum_{i=1}^{n} x_{ijt} \geq 1 \text{ for all } j \in \{1, 2, \ldots, n\}. \tag{2}$$

Ensures we enter each city at most once.

Here are our first three constraints:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijt} \leq 1 \text{ for all } t \in \{1, 2, \ldots, m\}. \tag{1}$$

Ensures we have at most one flight per day.

$$\sum_{t=1}^{m}\sum_{i=1}^{n} x_{ijt} \geq 1 \text{ for all } j \in \{1, 2, \ldots, n\}. \tag{2}$$

Ensures we enter each city at most once.

$$\sum_{t=1}^{m}\sum_{j=1}^{n} x_{ijt} \geq 1 \text{ for all } i \in \{1, 2, \ldots, n\}. \tag{3}$$

Ensures we leave each city at most once.

Here are our first three constraints:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijt} \leq 1 \text{ for all } t \in \{1, 2, \ldots, m\}. \tag{1}$$

Ensures we have at most one flight per day.

$$\sum_{t=1}^{m}\sum_{i=1}^{n} x_{ijt} \geq 1 \text{ for all } j \in \{1, 2, \ldots, n\}. \tag{2}$$

Ensures we enter each city at most once.

$$\sum_{t=1}^{m}\sum_{j=1}^{n} x_{ijt} \geq 1 \text{ for all } i \in \{1, 2, \ldots, n\}. \tag{3}$$

Ensures we leave each city at most once.

What's the problem?

Problem 1: Doesn't prevent cycles!

- Flight 1: BOS to NYC
- Flight 2: NYC to BOS
- Flight 3: SEA to SFO
- Flight 4: SFO to SEA

Problem 1: Doesn't prevent cycles!

- Flight 1: BOS to NYC
- Flight 2: NYC to BOS
- Flight 3: SEA to SFO
- Flight 4: SFO to SEA

Problem 2: Doesn't prevent logistical impossibilities!

- Flight 1 on June 2: BOS to NYC
- Flight 2 on June 5: SEA to BOS
- Flight 3 on June 8: NYC to SEA

Our implementation:

- We have a master server that sets up and calls everything (uses **xmlrpc**!)
- We have a web crawler that crawls **Matrix Airfare Search** to obtain prices
- We have a slave server that gets called by the master, and then uses the web crawler to obtain prices.
- The algorithm to solve binary IP is **Balas' Additive Algorithm**. Basic idea: depth-first search of all possible variable assignments with pruning and look-aheads to avoid searching "impossible" solutions.

Let's do a 3 city, 3 day demonstration!

- We ran it on an example with 4 cities over the span of 6 possible travel days. This means $4 \cdot 3 \cdot 6 = 72$ total variables X_{ijt}.
- Our solution worked and it analyzed 73,976 nodes in the DFS tree, rather than the full 2^{72} set of solutions.
- Took just about 10 minutes, and price-checking took almost all the time.
- This is still an exponential time solution in general, but **much** faster than brute force.

Still a number of things to do before May 23:

- **Distribute price checking** (e.g., using PlanetLab)
- Evaluate speed and practical usefulness
- Expand to get multiple flights for one ticket
- Make a front-end server for the ignorant masses
- Finish paper: 8 pages overview, 15 pages of dense, technical details

Thank you for your attention.

Any questions?

Please contact one of us if you have any comments or feedback about our presentation. We view it as a privilege to be able to give these talks, and want to make sure that our viewers are as happy as possible.