The Distributed Retired Traveling Salesman Problem

Daniel Seita DTS1@WILLIAMS.EDU
Ziang Zhang ZZ2@WILLIAMS.EDU
Department of Computer Science, Williams College, Williamstown, MA 01267 USA

Abstract

The use of major online travel agencies has made scheduling long-term travel much easier by allowing users to easily select a set of flights to purchase tickets from. Current travel agencies allow users to plan out trips involving airlines to more than one major city, but they require (1) specific dates for each city and (2) an ordering. We propose a system that does not require to burden the user with this type of decision. It essentially solves a harder version of the Traveling Salesman Problem since costs are not constant between two cities. Assuming the number of cities is limited to a reasonable number for a trip (e.g., four) then this algorithm should successfully find the cheapest cost flight under a number of weak assumptions. We present the theoretical and systematic components of the project, discuss empirical results, and suggest future work.

1. Introduction and Motivation

The use of major online travel agencies, such as Travelocity and Kayak, has made scheduling long-term travel much easier by allowing users to easily select a set of flights to purchase tickets from. People use a combination of factors to help them make their decision, such as the price of all the flights and the days they wish to land and depart from a city. Current travel agencies allow users to plan out trips involving airlines to more than one major city, but they require (1) specific dates for each city and (2) an ordering.

For instance, if the first author of this paper wants to visit Paris, London, and Tokyo from June 1 to July 1, and he starts out in New York City (NYC), he may want to schedule flights such that he departs from NYC to arrive in Paris on June 1, and then departs from Paris so that he arrives in London on June 10, and then depart there to be in Tokyo on June 20, and then fly back to NYC on July 1. But suppose the first author is retired (and presumably, is rich, or stole money from the second author). Thus, he has substantial free time, so he is not restricted on which days he can be in particular cities. He is still prudent, so he wants to schedule trips such that airline travel costs are *minimal*. This is challenging using current agencies, which motivates us to produce a more general service that can solve this problem:

- a date range from X to Y (with X occurring earlier than Y),
- and a list of destinations to reach, $d_1, d_2, \ldots, d_m$, with d_1 equal to d_m only if the user wants to end up at the same spot where he started. (We assume for the sake of realism that these destinations are all reachable by air travel; a number of other assumptions will be discussed in Section 5)

The goal is to find the **cheapest**, valid schedule of flights $f_1, f_2, \ldots, f_n$, with $n > m$, such that it touches each city at least once[1]. We will use the Matrix Airfare Search[2] database to obtain information about our flights; the information itself will be extracted with our own web crawler. We coin this "The Retired Traveler Problem," primarily because if we assume the dates are set far enough apart, it would interfere with a non-retired person's occupation. For details on the real problem, see sources such as (Applegate et al., 2007).

Our project is a combination of mathematics and systems. Sections 2 and 3 describe our objectives in further detail.

2. Mathematical Component

2.1. An Integer Programming Formulation

We will solve this problem using a technique known as **integer programming**. This is loosely based off of the many ways of solving the *traveling salesman problem*, where the

Proceedings of the 1st International Conference on Distributed Systems, Williamstown, USA, 2014. Program Chair: Jeannie Albrecht.

[1] The cheapest route of flights to visit these cities may, in fact, require the traveler to visit an airport more than once.

[2] http://matrix.itasoftware.com/

goal is to hit each city exactly once with fixed costs. Our problem is more difficult than the traveling salesman problem, because our costs are not fixed. The cheapest way of going from city i to city j will vary depending on the day. (The other major difference between our problem and the traveling salesman problem is that we are assuming we can touch each city more than once[3].)

We define our problem formulation as follows.

- We have n cities to reach; we index cities by i or j (where $i, j \in \{1, 2, \ldots, n\}$).
- We have t (consecutive) days when we can travel: $t \in \{1, 2, \ldots, m\}$. Let
- The minimum cost for traveling from city i to j on day t is c_{ijt}. To start out this project, we assume that all flights span one day and are not overnight, for simplicity. As soon as possible, we will expand this to include potential hub stops, overnight flights, etc.

This allows us to define the following binary variables:

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

The goal is to solve this problem:

$$\text{Minimize} \sum_{t=1}^{m} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ijt} x_{ijt}, \tag{1}$$

subject to the following constraints:

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

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

$$\sum_{t=1}^{m} \sum_{j=1}^{n} x_{ijt} \geq 1 \text{ for all } i \in \{1, 2, \ldots, n\}. \tag{4}$$

The first constraint ensures that we have at most one flight per day. The second constraint ensures we enter each city at least once. The third constraint ensures we leave each city at least once. Sadly, this is **not** enough to solve our problem! The issue here is with potential cycles. Figure 1 shows how the integer programming solution could theoretically give us a solution consisting of disjoint cycles.

[3]This may not actually make our problem much harder but the varying costs certainly will.

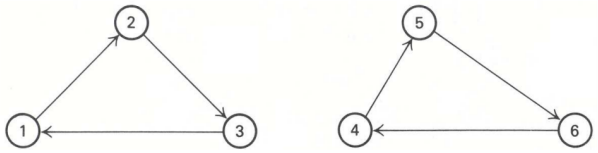

Figure 1. This represents two sets of disjoint cycles.

To counteract this issue, we will need to ensure we have additional variables. With the six-variable situation in Figure 1, for instance, we will need constraints such as

$$\sum_{t=1}^{m} (x_{14t} + x_{15t} + x_{16t} + x_{24t} + x_{25t} + x_{26t} + x_{34t} + x_{35t} + x_{36t}) \geq 1. \tag{5}$$

The previous constraint ensures that at least one leg of the tour connects cities 1, 2, and 3 with cities 4, 5, and 6. We need constraints like these for every way we can split up the cities into groups. Of course, the need to implement these kind of constraints to prevent sub-tours is why integer programming is so difficult. Fortunately, we will assume the user is going to keep the number of cities reasonably small.

2.2. Implementation Details

How to best implement this is still an open question. There are algorithms for implementing integer programming. The question is whether we can efficiently use our *distributed systems*. There are a number of results based on parallelizing integer programming in the literature. We will read through them and by the first checkpoint (see Section 4.1) we will have made a decision. There are also some online implementations, such as CPLEX Optimizer[4], but I think it is in our best interests to implement this ourselves.

3. Systems Component

We will have a front-end server, a master server, and a group of slave servers in the system, as shown in Figure 2. We get all the servers from the PlanetLab Platform (thanks to Jeannie).

3.1. Front-end Server

This server will serve as a front-end interface for the ignorant masses. It receives user input, forwards the request to the master server (shown in the image as step `1`), listens

[4]http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html

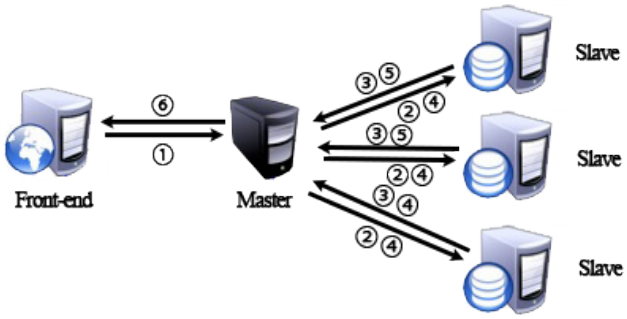

Figure 2. This represents our ideal machine setup.

to the master for the result (shown as step `6`), and displays the result when it is ready. More specifically, the front-end server will have a simple website to interact with the user.

3.2. Master Server

The Master server is the brain of the system. It listens to the user request sent from the front-end server (step `1`). When a request comes in, it first comes up with a list of flight prices on all possible dates and for all possible destinations in the problem that we need to get for the computation, distributes the flight price queries among all the slaves (step `2`), gathers all the price data back from the slaves (step `3`), and combines them. It then upload the combined data to a distributed file system to make the data available for parallel computing on all the slaves. Then it starts the parallel computing for the cheapest route using a distributed zero-one linear programming algorithm on all slaves (steps `4` and `5`). After the computation, it sends the results of the cheapest route back to the front-end server (step `6`).

3.3. Slave Server

We have a group of slave servers that listens to the command of the master server and does the distributed tasks of flight price querying (using the web crawler we write) and parallel zero-one linear programming.

4. Checkpoints

4.1. Checkpoint 5/2

By this date, we will have our machine/cluster and the web crawler working. We will finalize the mathematical portion of our work by choosing which distributed algorithm to implement.

4.2. Checkpoint 5/9

By this date, we will have implemented (1) the distributed systems requesting process[5] and (2) the (possibly distributed) integer programming problem. In other words, the code will solve the problem for the two of us any reasonable input (it just lacks a front-end for the rest of the ignorant masses).

UPDATE!!!

We are now at the second checkpoint. Here is what we have accomplished:

- We have implemented a web crawler
- We have implemented a web crawler server that can talk to the crawler to get the price
- We have implemented Balas' Additive Algorithm (Balas, 1965) for efficiently solving binary integer programming problems.
- We have gotten the basic setup for our writeup (i.e., this PDF document) ... it just needs to be heavily revised.

We have sent all of our code to Jeannie by email.

I (Daniel Seita) have to admit that we're a bit behind where we want, because we haven't been able to actually combine our crawler's results with the input to the integer programming problem, but we're going to do nothing but 339 work over the weekend. We're also thinking of expanding the work to include other integer programming formulations (this is going to be an additional part of our mathematical side of this project).

4.3. Presentation and Final Product

By this date, we will have implemented a fancy front-end website where users can query a flight schedule. We will also be prepared to give a talk by the May 13 (**UPDATE: May 11**) date. Then we will work on fixing up this paper and doing some more programming.

5. Limtiations

TODO

6. Conclusions and Future Work

TODO

[5] As stated earlier, this is when the master distributes requests to slave machines to determine the cheapest flight for a certain date and city destinations.

Acknowledgments

TODO

References

Applegate, David L., Bixby, Robert E., Chvatal, Vasek, and Cook, William J. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, Princeton, NJ, USA, 2007. ISBN 0691129932, 9780691129938.

Balas, Egon. An additive algorithm for solving linear programs with zero-one variables. *Operations Research*, 13(4):517–546, 1965. doi: 10.1287/opre.13.4.517. URL http://pubsonline.informs.org/doi/abs/10.1287/opre.13.4.517.