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 identify a set of flights in just a few clicks. Current travel agencies allow users to plan out long-term trips involving flights to more than two cities, but they require specific arrival and departure dates for each city and a city ordering. We present a system that does not burden the user with these decisions. Specifically, our code takes in two required inputs: a list of cities the user wishes to travel to, and a date range over which they are willing to travel, and outputs the cheapest set of flights within that range that form a valid route. It essentially solves a harder version of the Traveling Salesman Problem since costs are not constant between two cities. Under several weak assumptions, and assuming that the number of cities and days is sufficiently limited, then our algorithm should successfully find the cheapest cost flight in a reasonable amount of time. We present the theoretical and systematic components of the project and discuss empirical results.

1. Introduction and Motivation

The use of major online travel agencies, such as Travelocity[1] and Kayak[2], 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 total price of 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 multiple destinations (see Figure 1 for an example using Travelocity), but they require (1) specific dates for each city and (2) an ordering.

[1]www.travelocity.com
[2]www.kayak.com

Proceedings of the 1st International Conference on Distributed Systems, Williamstown, USA, 2014. Program Chair: Jeannie Albrecht. Copyright 2014 by the author(s).

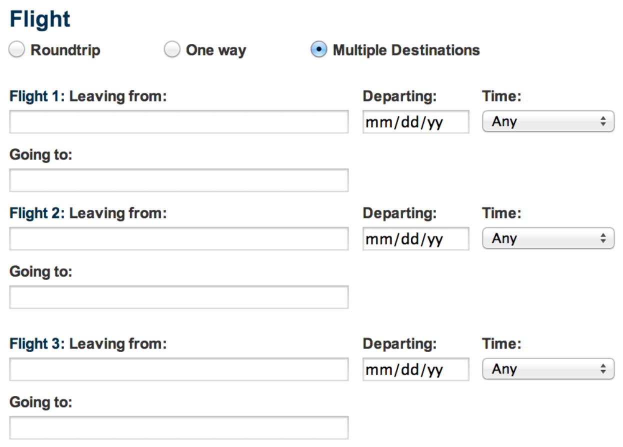
This limitation can make searching for flight routes time-consuming for users who are not restricted to arriving at cities on particular dates. For instance, suppose one resides in New York City (NYC) and wants to schedule a summer trip to Paris, London, and Tokyo from June 1 to June 20, and it does not matter to him how long he stays in a city. Suppose that it also does not matter the ordering that he arrives at the cities. Thus, the factors that matter[3] are:

- The full flight route occurs within the date range (in this case, between June 1 and June 20).
- The flight route must arrive and leave at least once for NYC, Paris, London, and Tokyo.
- The flight route is valid and logically consistent. For instance, if the first flight takes him from NYC to Tokyo, the second flight in the route should not depart from a place other than Tokyo.

There are a vast pool of valid flight routes. It is reasonable to assume that out of all these, one would want the *cheapest* route. Unfortunately, finding the cheapest route possible is challenging using current agencies, because they require knowing the arrival and departure dates for each city. The reason for this is simple: the problem of finding this flight route is a hard problem to solve.

In fact, this problem is very similar to the well-known Traveling Salesman Problem (Applegate et al., 2007), which asks if there exists a cycle in a graph that touches each node exactly once (i.e., a Hamiltonian Path). Each edge has a cost associated with it, and the decision version of the problem, which asks if there exists a valid Hamiltonian path with cost bounded by B, is NP-complete (Karp, 1972). In our case, we do not force the user to visit each city ex-

[3]For brevity, we list a high-level overview of the problem without getting into too many technical details; Section 6 describes additional assumptions we impose on the problem to make it practical.


Flight

Roundtrip One way [✓ Multiple Destinations]

Flight 1: Leaving from:	Departing: mm/dd/yy	Time: Any
Going to:		
Flight 2: Leaving from:	Departing: mm/dd/yy	Time: Any
Going to:		
Flight 3: Leaving from:	Departing: mm/dd/yy	Time: Any
Going to:		

Figure 1. Travelocity lets users select multiple destinations.

actly once because the cheapest flight through a sequence of cities *may require* visiting a city more than once.

What makes our problem (likely) much harder than the Traveling Salesman Problem is that the cost of an edge between two nodes in our graph (i.e., two cities) is not constant, because flight prices frequently change. We coin our problem the "Distributed Retired Traveling Salesman Problem." The "Retired" portion comes from how we assume that whoever is planning this trip is retired, because otherwise, how would he or she have the time and money? The "Distributed" part is because we incorporate concepts from the design and practice of distributed systems to build software that can tackle this problem. We discuss the mathematical and systems portions of our solution in Sections 3 and 4, respectively.

2. Related Work

The Traveling Salesman Problem (TSP) is one of the oldest and most well-known problems in combinatorial optimization. While no exact, polynomial time solution for the decision problem is known, there is a $3/2$-approximation algorithm due to Christofides (Christofides, 1976). That algorithm was long the standard approximation algorithm until a stunning recent result in (Gharan et al., 2011), which has thus spurred a cascade of additional research relating to the TSP (e.g., (Moemke & Svensson, 2011)).

In contrast to the attention devoted to TSP, there appears to be very little research focusing on the special case of our problem, which introduces additional challenges. These can be mathematically-oriented, such as how to manage the variable costs, or systems-oriented, such as how to even obtain the actual flight costs. We have found nothing in the literature that particularly fits our problem.

3. Mathematical Component

The mathematical portion of our work uses a technique known as *binary integer linear programming*, which falls under the broader category of optimization techniques. The goal in optimization is straightforward: given a set of variables and a set of constraints, the goal is to find the best, feasible solution according to some criteria, such as cost (in which case, "best" means "minimal").

3.1. Linear and Integer Programming

One of the most commonly-used optimization techniques is linear programming, introduced by Dantzig in (Dantzig, 1963).

Definition 3.1. *A* linear programming problem *consists of three components:*

1. *a finite collection of linear inequalities or equations in a finite number of unknowns, $x_1, \ldots, x_n$;*
2. *sign constraints $x_i \geq 0$ on some (possibly empty) subset of the unknowns;*
3. *a linear function to be minimized or maximized.*

An assignment to the variables $x_1, \ldots, x_n$ satisfying the first two conditions is a feasible *solution. If it also satisfies the third, then it is an* optimal *solution (Franklin, 2002).*

Linear programming has tremendous applications, and a full list of them would be impossible to create. Some problems well-suited to linear programming include investment management, scheduling problems, and the diet problem. (The last problem concerns the rather interesting question of what is the minimum cost of a nutritionally adequate diet.) The reason for linear programming's great versatility is the ease at which constraints can be added to a model.

More specific cases of linear programming are integer and binary linear programming. (Sometimes we drop the "linear" part for brevity.)

Definition 3.2. *An* integer programming problem *is a linear programming problem with the added restriction that all variables (i.e., unknowns) $x_1, \ldots, x_n$ are integers.*

Definition 3.3. *A* binary integer programming problem *is an integer programming problem with the added restriction that all variables $x_1, \ldots, x_n$ are such that $x_i \in \{0, 1\}$.*

We see that our problem is particularly suited to binary integer programming, because all the possible flights we could take can be viewed as a set of binary random variables, each of which has value 1 if we decide to take that flight, and 0 otherwise. We now formulate this problem in more detail.

3.2. An Integer Programming Formulation

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 2 shows how the integer programming solution could theoretically give us a solution consisting of disjoint cycles.

To counteract this issue, we will need to ensure we have additional variables. With the six-variable situation in Figure 2, for instance, we will need constraints such as

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

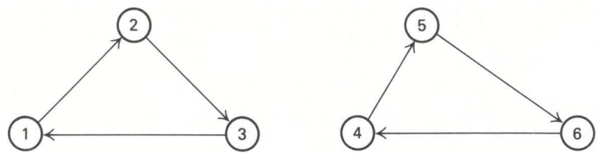


Figure 2. This represents two sets of disjoint cycles.

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.

here, we shoudl also add in prof miller's great example based on why int prog is so hard

3.3. Implementation Details

We will use the Matrix Airfare Search[4] 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.

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 5.1) we will have made a decision. There are also some online implementations, such as CPLEX Optimizer[5], but I think it is in our best interests to implement this ourselves.

4. 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 3. We get all the servers from the PlanetLab Platform (thanks to Jeannie).

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

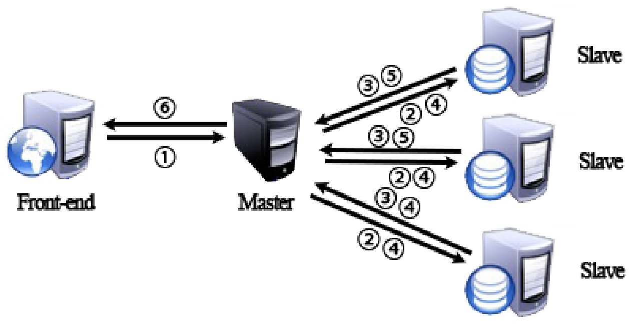


Figure 3. This represents our ideal machine setup.

4.1. 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 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.

4.2. 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`).

4.3. 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.

5. Checkpoints

5.1. 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.

5.2. Checkpoint 5/9

By this date, we will have implemented (1) the distributed systems requesting process[6] 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).

5.3. 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.

6. Limtiations

TODO

7. Conclusions and Future Work

TODO

[6]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

We thank Professor Jeannie Albrecht for teaching the course that inspired this project. We are also indebted to Professor Steven Miller for introducing us to the world of operations research.

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.

Christofides, Nicos. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

Dantzig, George B. *Linear programming and extensions*. Rand Corporation Research Study. Princeton Univ. Press, Princeton, NJ, 1963. URL http://gso.gbv.de/DB=2.1/CMD?ACT=SRCHA&SRT=YOP&IKT=1016&TRM=ppn+180926950&sourceid=fbw_bibsonomy.

Franklin, Joel N. *Methods of mathematical economics : linear and nonlinear programming, fixed-point theorems*. Classics in applied mathematics. SIAM, Philadelphia, 2002. ISBN 0-89871-509-1. URL http://opac.inria.fr/record=b1105716.

Gharan, Shayan Oveis, Saberi, Amin, and Singh, Mohit. A randomized rounding approach to the traveling salesman problem. In Ostrovsky, Rafail (ed.), *FOCS*, pp. 550–559. IEEE, 2011. ISBN 978-1-4577-1843-4. URL http://dblp.uni-trier.de/db/conf/focs/focs2011.html#GharanSS11.

Karp, R. Reducibility among combinatorial problems. In Miller, R. and Thatcher, J. (eds.), *Complexity of Computer Computations*, pp. 85–103. Plenum Press, 1972.

Moemke, Tobias and Svensson, Ola. Approximating graphic tsp by matchings. In *Proceedings of the 2011 IEEE 52Nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pp. 560–569, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-0-7695-4571-4. doi: 10.1109/FOCS.2011.56. URL http://dx.doi.org/10.1109/FOCS.2011.56.

A. Balas’ Additive Algorithm

TODO