University of Glasgow | School of Computing Science

The Traveller's Problem

Iva Stefanova Babukova

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Masters project proposal

18 December 2016

Contents

1 Introduction

In this work we present the Traveller's Problem (TP), a computational task whose extensions and variations are often encountered by travellers around the world. The task is concerned with creating a valid travel schedule, using airplanes as a means of transportation and in accordance with certain constraints specified by the traveller.

2 Problem Formulation

Each instance of TP consists of:

1. A set of airports $A = \{A_0, ..., A_n\}$ for $n > 0$. Each airport $A_i \in A$ represents a location the traveller can begin their commute in, visit as a desired destination, or connect in on the way to their destination.
2. The trip starts and ends at the same airport A_0, which is referred to as the *home point*.
3. The total travel time T, within which the traveller must have visited all destinations and returned to the home point.
4. A set of flights $F = \{f_0, ..., f_m\}$. Each flight f_j has:
   - departure airport A_j^d,
   - arrival airport A_j^a,
   - date t_j,
   - duration Δ_j,
   - cost c_j,

   for some non-negative integer j less than or equal to n. The date t_j is a positive rational number less than or equal to T that shows at which day f_j leaves its departure airport. The duration Δj is a positive fraction that shows the amount of time that takes for flight f_j to go from A_j^d to A_j^a. The cost c_j is a positive number that denotes the number of units of some currency ϵ that the traveller pays in order to be able to board flight f_j.
5. Each airport A_i has a *connection time* C_{A_i}, that is the time that takes to switch from any flight f_p with $A_p^a = A_i$ to any flight f_q with $A_q^d = A_i$.
6. A set of *destinations* $D = \{D_1, ..., D_l\}$, $D \subseteq A$, $l \leq n$.

A solution to any instance of TP is a sequence s of k valid flights, $\langle f_{i_1}, f_{i_2}, ..., f_{i_k} \rangle$ $\subseteq F$. We say that s is valid if the flights in s have the following properties:

1. $A^d_{i_1} = A^a_{i_k} = A_0$
2. $A^a_{i_j} = A^d_{i_{j+1}}, \quad 0 < j < k \quad j$
3. $t_{i_j} + \Delta_{i_j} + C_r \leq t_{i_{j+1}}, \quad 0 < j \leq k, \quad$ where $r = A^d_{i_{j+1}}$
4. $t_{i_k} + \Delta_{i_k} \leq T$
5. $\forall D_p \in D, \ \exists \ f_{i_j} \in s$, such that $A^a_{i_j} = D_p$

Note that a valid sequence of flights may contain one or more flights to and from airports that are not destinations. Such airports are called *connections*.

The *optimization* version of TP (TPO) asks for an *optimal* solution s which minimizes the total sum of the prices of the flights in s, denoted by $c(s)$.

The *decision* version of TP (TPD) asks whether there exists a valid sequence of flights s, such that $c(s)$ is less than or equal to some given integer B. The solution of this problem is a 'yes' or 'no' answer.

There exist a variety of additional constraints and extensions that can be added to TP. Our problem formulation has only presented the hard constraints which every valid solution to a TP instance must satisfy. In real-world problems, travellers may have additional preferences (soft constraints) and requirements (hard constraints) with regards to their travel. These are discussed in the next two sections.

2.1 Hard Constraints

This section presents some additional constraints that might be imposed on a TP instance. If any of them is required, then a solution that does not satisfy the requirement is considered as invalid.

- Travellers may wish to spend a certain amount of days at a given destination, specified by both upper and lower bounds. The days may additionally be constrained to be consecutive or not.
- Travellers may require to spend a given date at a given destination, for example due to an event occurring on that date in this destination.
- Travellers may require not to fly through a given airport more than once.

2.2 Soft Constraints

It may be desirable to search for a solution that satisfies some of the following requirements:

- Travellers may wish to spend a certain amount δ_i of days in each destination D_i, where δ_i may be specified as a lower or an upper bound.
- Travellers may wish to avoid taking connection flights. In such requirement, we wish to maximise the number of flights to and from destinations.
- Travellers may want to spend as little time on flying as possible. In such case, we wish to find a solution that minimises the sum of the durations of all flights.

Note that we may have an instance for which all soft constraints can not be satisfied simultaneously. In such case, the traveller may be required to rank his requirements in an order of preference. The instance becomes a lexicographic optimisation problem, where we first optimise the highest ranked objective, and subject to this we optimise the second ranked objective and so on. If all requirements are equally important for the traveller, we have to solve a multiobjective optimisation problem, where the objectives are all constraints required by the traveller. Each of the objectives is given a weight of importance. The problem is then to optimize the objective function, composed by the constraints, each of them multiplied by its weight. Multiobjective and lexicographic optimisation problems are discussed later in this work.

Note that most of the aforementioned constraints can be viewed as either hard or soft, depending on the user requirements. It is therefore suggested that any attempt at an investigation of TP assumes as an additional non-functional requirement that any proposed model to solve TP is flexible and can be easily extended by adding, removing and modifying the aforementioned constraints.

3 Worked Examples

We present an example instance of TP and comment on some of its solutions.

Example 1. A traveller wishes to visit 4 airports from a set of 7 airports available to travel to and from:

Glasgow (G), Berlin (B), Milan (M), Amsterdam (A), Paris (P), Frankfurt (F), London (L).

Airport G is the home point, F and L are connections, and B, M, A and P are the destinations. The travel time of the traveller is 15 days. All available flights are listed on Table 1. For simplicity, the duration of each flight is assumed to be 1 day. This means that if the traveller gets a flight at date x, they will reach the arrival airport at day $x + 1$.

Flight No	**Departs**	**Arrives**	**Date**	**Price**
GA1	G	A	1	74
GF1	G	F	1	86
FB2	F	B	2	156
GL3	G	L	3	25
MF3	M	F	3	78
BP4	B	P	4	67
AP4	A	P	4	58
PM6	P	M	6	71
FM8	F	M	8	234
MF9	M	F	9	39
FA10	F	A	10	220
FB11	F	B	11	122
FM12	F	M	12	250
PL12	P	L	12	45
BG13	B	G	13	335
BL13	B	L	13	102
AG13	A	G	13	90
LG14	L	G	14	24

Table 1: List of flights with departure and arrival airports, flight date and price.

Solution. A valid solution of the TP instance in the example above is the sequence of flights s, where the each flight is represented by its flight number, specified in the first column of Table 1:

$$s = \langle GA1, AP4, PM6, MF9, FB11, BG13 \rangle$$

The total flights cost $c(s)$ is 699.

A valid solution with lower cost is the following sequence:

$$s' = \langle GA1, AP4, PM6, MF9, FB11, BL13, LG14 \rangle$$

Here $c(s')$ is 483 and hence s is not optimal.

Example 2. Given the same problem instance as in Example 1, suppose that the traveller has booked a ticket for a concert in B on day 3. The traveller requires to attend the concert.

Solution. In such case, neither s, nor s' from Example 1 are solutions, because both of them assign the traveller to be at a different location (airport A) at day 3. The following sequence is a solution:

$$s'' = \langle GF1, FB2, BP4, PM6, MF9, FA10, AG13 \rangle$$

The total cost $c(s'')$ is equal to 729, which is more expensive than s and s'.

4 List of Problems

This Section gives a list of known problems, referred to in this work when proving the NP-hardness of TP (Section 5) and when reviewing the existing work (Section 6). The problems, marked with an asterisk ($*$) next to their title, are known to be NP-complete [6].

1. **Vehicle-Routing Problem * (VRP)**
   Instance. .
   Question. .

2. **Travelling Salesman Problem * (TSP)** [1]
   Instance. Set A of n cities, distance $d(A_i, A_j)$ between each pair of cities A_i, $A_j \in A$, positive integer B.

[1] Note that TSP is a special case of VRP when only one vehicle is allowed.

Question. Is there a tour of A having length B or less, i.e., a permutation of cities $\gamma = \langle A_{\pi(1)}, A_{\pi(2)}, ..., A_{\pi(n)} \rangle$ of A such that the total travel distance L_γ:

$$L_\gamma = \left(\sum_{i=1}^{n-1} d(A_{\pi(i)}, A_{\pi(i+1)}) \right) + d(A_{\pi(n)}, A_{\pi(1)}) \leq B \quad ?$$

3. **Travelling Salesman Problem Under the Triangle Inequality* (TSP-Δ)** [2]

   Instance. Set A of n cities, distance $d(A_i, A_j)$ between each pair of cities A_i, $A_j \in A$ that satisfies the triangle inequality, positive integer B.

   Question. Is there a tour of A having length B or less, i.e., a permutation of cities $\gamma = \langle A_{\pi(1)}, A_{\pi(2)}, ..., A_{\pi(n)} \rangle$ of A such that the total travel distance L_γ:

$$L_\gamma = \left(\sum_{i=1}^{n-1} d(A_{\pi(i)}, A_{\pi(i+1)}) \right) + d(A_{\pi(n)}, A_{\pi(1)}) \leq B \quad ?$$

4. **Time-Constrained TSP* (TCTSP)** [3]

   Instance. Set A of n cities, distance d(A_i, A_j) between each pair of cities A_i, $A_j \in A$, positive integer B, lower and upper bounds l_i and u_i respectively for each city A_i that specify its time window.

   Question. Is there a permutation of cities $\gamma = \langle A_{\pi(1)}, A_{\pi(2)}, ..., A_{\pi(n)} \rangle$ of A, such that each city A_{π_j} is visited at time t_j, where $l_j \leq t_j \leq u_j$, and

$$L_\gamma = \left(\sum_{i=1}^{n-1} d(A_{\pi(i)}, A_{\pi(i+1)}) \right) + d(A_{\pi(n)}, A_{\pi(1)}) \leq B \quad ?$$

5. **The Assignment Problem (AP)**

   Instance. Set A and set B with equal size, cost $C(a, b)$ of matching $a \in A$ to $b \in B$.

   Question. Find a bijection $f : A \leftarrow B$ such that $\sum_{a \in A} C(a, f(a))$ is minimised.

6. **Vehicle-Routing Problem (VRP)**

7. **Job-Shop Scheduling Problem (JSSP)**

8. **Longest Path Problem**

5 Complexity of TP

We state a theorem about the complexity of TP and prove it.

Theorem 1. *TPD is NP-complete.*

[2] Note that this problem is a variant of TSP.

[3] Note that this problem is a variant of TSP.

Proof. This proof first shows the membership of TPD in the NP class of problems. Second, we prove the NP-hardness of TP by constructing a polynomial-time reduction from a known NP-complete problem Π to TPD, where Π is chosen to be TSP-Δ, defined in Section 4. Its NP-hardness follows by a reduction from the Hamiltonian Cycle problem. The proof is presented by Garey and Johnson [6].

Given an instance of TPD and s, which is a sequence of flights from F, we can write an algorithm that checks in polynomial time whether s is a solution. To accept or reject validity, the algorithm only needs to traverse s and check that it satisfies all required properties. Therefore, TP is in NP.

Let π be an instance of TSP-Δ. Let π' be an instance of TPD with the following properties:

- The set of airports in π' is identical to the set of cities in π and it is similarly denoted as A (a city in π is called an airport in π'). Airport A_1 is the home point.
- Each airport in A is also a destination.
- The connection time C_{A_i} for each airport A_i is equal to 0.
- T is equal to n and the days start from 0.
- Let C be the Cartesian product of the airports in A with itself, that is $C = A \times A = \{(A_i, A_j) : A_i \in A, A_j \in A, i \neq j\}$. Then F is a set of flights, such that for every $(A_i, A_j) \in C$, there exists a flight f_k in F, such that $A_k^d = A_i$ and $A_k^a = A_j$ for every date $0 \leq t < T$.
- For every $f_k \in F$, c_k is equal to $d(A_k^d, A_k^a)$ in π. Therefore, the flight costs also satisfy the triangle inequality.
- For every $f_k \in F$, $\Delta_k = 1$.
- B is the upper bound on the allowed total cost.

Suppose that $\gamma = \langle A_{i_1}, A_{i_2}, ..., A_{i_n} \rangle$ is a solution to π, where $\langle i_1, ..., i_n \rangle$ is a permutation of $\langle 1, ..., n \rangle$ and the total travel distance $L_\gamma \leq B$. In π', γ is equivalent to the order of visited airports by some sequence of flights $s = \langle f_{j_1}, f_{j_2}, ...f_{j_n} \rangle$, such that for each p $(1 \leq p \leq n)$ there exists q $(1 \leq q \leq n)$ such that $A_{j_q}^d = A_{i_p}$ and $A_{j_q}^a = A_{i_{p+1}}$, where subscripts are taken modulo n. For each q $(1 \leq q \leq n)$, $A_{j_q}^a = A_{j_{q+1}}^d$ and $t_{j_q} = q - 1$.

Since γ is a tour in π, γ contains the home point in π' A_1 at some position i_p $(1 \leq p \leq n)$, where subscripts are taken modulo n. Therefore, there exist flights f_{j_u} and f_{j_v} in s, with $A_{j_u}^d = A_{i_p}$ and $A_{j_u}^a = A_{i_{p+1}}$, and $A_{j_v}^d = A_{i_{p-1}}$ and $A_{j_v}^a = A_{i_p}$. Therefore, $A_{j_u}^d = A_{j_v}^a = A_{i_p}$ and s satisfies the first property of a valid solution.

From the construction of s, it follows that property 2 also holds. To satisfy property 3 and 4, it is sufficient to choose flights from F such that for every $f_{j_u} \in s$, $t_{j_u} = u - 1$. We know

that such flights exist in F by the construction of the set F. Property 5 is satisfied, since all airports in A are destinations.

Since the cost of every flight in F is equal to the distance between the two cities in π that correspond to its departure and arrival airport, it follows that $c(s) = L_\gamma \leq B$.

The sequence s satisfies all requirements for a valid solution to π'. Therefore, a solution of π is also a solution to π'.

Conversely, suppose that $s = \langle f_{j_1}, ..., f_{j_k} \rangle$ is a solution to π', where the flights in s visit destinations in the sequence $\gamma' = \langle A_{i_1}, A_{i_2}, ..., A_{i_m} \rangle$. We will prove that γ' is a solution of π.

By construction of π', all n airports in A are also destinations. Therefore, γ' contains all cities in A, that is $m \geq n$. Suppose that $m > n$ and an arbitrary airport A_p $(i_1 \leq p < i_n)$ is included more than once in γ'. Then s must contain one than one flight with arrival airport equal to A_p. The duration of each flight in π' is one day. Therefore, for every q $(1 \leq q < n - 1)$, $t_{j_{q+1}} = q$. Since the traveller has only n days of total travel time, $k = n$ and s is restricted to contain exactly n flights. The only way to visit n destinations, given n flights is that all flights in s have unique arrival airports. Assuming that A_p is visited more than once means that there is more than one flight with arrival airport equal to A_p, which is a contradiction. Therefore, $m = n$ and each airport in A is visited exactly once.

From the properties of s it follows that $A^d_{j_1} = A^a_{j_n} = A_1$. Therefore, γ' is a cycle of size n. We know that $c(s) \leq B$. We assigned a cost of each flight f_k in F to be equal to $d(A^d_k, A^a_k)$ in π. Therefore, the total travel distance $L_{\gamma'} = c(s)$ which is less than or equal to B.

According to the specifications of π, γ' is a solution. Therefore, a solution to π' is also a solution to π.

The transformation from a TSP-Δ instance to an instance of TPD can be done in polynomial time. For each of the $n(n\text{-}1)/2$ distances d(A_i, A_j) that must be specified in π, it is sufficient to check that the same cost is assigned to the flights from A_i to A_j for all dates.

Therefore, TP is in NP and the decision version of TSP-Δ can be reduced to TPD in polynomial time, from which it follows that TPD is NP-complete.

□

6 Background Survey

We present some existing approaches to dealing with NP-hard problems, namely branch and bound (Section 6.2), approximation algorithms (Section 6.3), constraint programming (Section 6.4) and integer programming (Section 6.1). The way each of these methods tackles NP-hardness is illustrated with a specific NP-hard problem as an example, defined in Section 4.

6.1 Integer Programming

Integer programming (IP) is a method for solving optimisation problems. Given an optimisation problem π with a set of constraints ζ_π and a goal G_π, integer programming involves optimising G_π, subject to all constraints in ζ_π, where G_π is modeled as an expression, called an *objective function* and the constraints in ζ_π are expressed as mathematical equations. All variables in the model are restricted to take only integer values. As an example of a problem, modeled and solved with IP, we give TSP.

6.1.1 Integer Programming Formulation of TSP

We present the integer programming model for TSP and its variations, formulated by Dantzig et al. [4].

Let π be and instance of TSP. Let $x(i, j)$ be a variable that denotes whether the next city visited after i is j in some solution of π. The value of $x(i, j)$ is equal to 1 if this is true, or 0 if it is false. If $x(i, j)$ is 1, then i is called the *incoming* city and j - the *outgoing* city. One of the properties of a valid TSP tour is that each city is visited exactly once. This can be enforced with the following constraint:

$$\begin{aligned} \sum_{j \in A} x(i,j) &= 1, \qquad i \in A, \\ \sum_{i \in A} x(i,j) &= 1, \qquad j \in A, \end{aligned} \tag{1}$$

Constraint (1) ensures that each city in A is picked exactly once as an incoming and an outgoing city. However, it allows for the existence of one or more cycles of $n_1 < n$ cities, called *subtours*. For instance, consider Figure 1 and Figure 2. Both of them satisfy constraint (1). However, Figure 2 does not represent a valid tour, because it contains two subtours, each of size 3. To tackle this problem, Dantzig et al. [4] introduce the *subtour elimination* constraint, that is:

$$\sum_{S\bar{S}} x(i,j) \geq 2, \qquad \text{for all } S \subset A, \bar{S} = A \S S \neq \emptyset \tag{2}$$

Constraint

$$\sum_{i \in V} \sum_{j \in V} c(i,j) x(i,j) \tag{3}$$

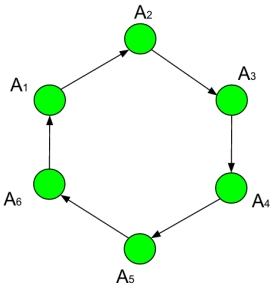

Figure 1: A valid TSP tour

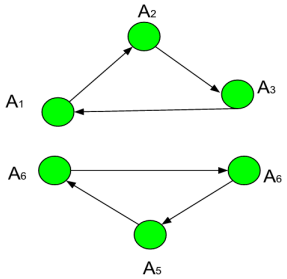

Figure 2: Two subtours of size 3

Equation (3) constrains that in a valid solution of π, each city has to be connected with exactly two cities. Note that formula (3) allows for solutions that include subtours, i.e. tours of length less than n. Equation (2) is also known as *subtour elimination* constraint and it serves the purpose to filter such invalid solutions.

In this formulation, one can remove constraint (2) to obtain an instance of AP with the same cost function as P [10, 5, 2].

By dropping constraint (2), the solution of P_{AP} can be either a tour, or a collection of subtours. Hence, every tour of P is a solution to AP, but not all solutions of AP are tours [2]. This allows for using the cost of the optimal solution of AP as a lower bound for the most optimal solution of P. In [9] it is showed with an experiment that both in theory and in practice the most optimal solution to AP is very likely to be a strong bound on P. The AP can be solved in at most $\mathcal{O}(n^3)$ steps [7, 3]. The "bound" procedure, that is called for every problem P_{ij} in Algorithm 1, computes AP_{ij}. It is shown in [9] that computing AP_{ij} for each problem P_{ij} in at most $\mathcal{O}(n^2)$ steps by the Hungarian method [7], provided that the algorithm is started from the optimal solution of the parent problem P_i.

6.2 Branch and Bound Algorithms

We present this technique for instances of TSP-Δ. The origins of the branch and bound technique date back to the work by Dantzig et al. [4], where an instance with 49 cities is solved and proved as optimal. The term "branch and bound" itself was introduced by Little et al. [10].

The branch and bound procedure repeatedly breaks up the set of feasible solutions into successively smaller subsets and calculates a *bound* on the objective function value for each subset. The bounds are obtained by performing *relaxation*, i.e. replacing the problem over a given subset with an easier (*relaxed*) problem, such that the solution value of the latter bounds that of the former. Bounds are used to discard certain subsets from further consideration.

Branch and bound procedure can be represented as a rooted tree whose root node corresponds to the original problem and the leafs of the tree form the set of all candidate

solutions. The successor nodes of each non-leaf node i correspond to the subproblems, defined by the branching rule.

Algorithm 1 presents a pseudo code of the branch and bound technique. An active problem set s is set to contain the problem set initially (line 3). Then, we branch on each problem P_i chosen from s. The branching rule generates a set of subproblems s_i (line 10). Each subproblem P_{ij} in s_i is then checked whether it is "worse" than the best one found so far by calculating its bound (line 12, 13). Problems with "worse" bounds are being discarded (line 14), otherwise they are added to s for further investigation (line 19). If the current bound is the "best" found so far, its value is remembered and used for comparison with next problems. On termination, the algorithm returns the most optimal solution of P (line 23).

Algorithm 1 Branch and Bound Procedure

```
1:  procedure BB (Problem P)
2:      begin
3:      s ← {P}                                  ▷ put P in the list of active problems set s
4:      bestVal ← NULL
5:      currentBest ← NULL
6:      currVal ← NULL
7:      while s not ∅ do
8:          P_i ← CHOOSE(s)                                  ▷ choose next subproblem
9:          s ← s - P_i
10:         s_i = BRANCH(P_i)                ▷ branch on P_i, returns a list of subproblems
11:         for P_ij in s_i do
12:             currVal ← BOUND(P_ij)                        ▷ calculate the bound of P_ij
13:             if currVal worse than bestVal then
14:                 kill P_ij                          ▷ P_ij can not lead to an optimal solution
15:             else if P_ij is a complete solution then
16:                 bestVal ← currVal
17:                 currentBest ← P_ij
18:             else
19:                 s ← s + P_ij
20:             end if
21:         end for
22:     end while
23:     return currentBest
24: end procedure
```

Algorithm 1 does not tell much about the "choose", "branch" and "bound" procedures. This is because there are many variations of their implementations. We discuss some of them in the following parts of this section.

6.2.1 Bounds based on the Assignment Problem

This relaxation is based on expressing the integer programming version of an optimisation problem P as an assignment problem (AP) after dropping one or more constraints from P. The AP relaxation is successfully used for solving TSP [10, 5, 2, 9, 1] and for the Vehicle Routing Problem (VRP) [8, 11]. We explain the principles of this relaxation using TSP as a use case.

The AP relaxation is reported to perform differently, depending on the type of the problem [9, 2]. In particular, its performance highly depends on whether the problem has symmetric or asymmetric cost matrix. In the case with TSP, symmetric cost matrix refers to the property that the distance $d(i, j)$ from i to j is equal to $d(j, i)$. TSP with symmetric cost matrix is called Symmetric TSP (STSP) and Asymmetric TSP (ATSP) otherwise.

Experiments with ATSP instances have shown that the AP relaxation provides an excellent lower bound [9].

6.2.2 Lagrangian relaxation

6.2.3 Choosing techniques

6.2.4 Branching rules

6.2.5 Performance of Branch and Bound Methods

Until the late 1980s, the most effective exact approaches for the vehicle routing problem were mainly branch and bound algorithms [11].

6.3 Approximation Algorithms

6.4 Constraint Programming

6.4.1 Local Search

6.4.2 Heuristics

6.4.3 Use Case

job-shop scheduling problem solution with CP

7 Proposed Approach

state how you propose to solve the software development problem. Show that your proposed approach is feasible, but identify any risks.

8 Work Plan

show how you plan to organize your work, identifying intermediate deliverables and dates.

References

[1] Edward K. Baker. Technical note - an exact algorithm for the time-constrained traveling salesman problem. *Operations Research*, 31(5):938–945, 1983. doi: 10.1287/opre.31.5.938. URL `http://dx.doi.org/10.1287/opre.31.5.938`.

[2] Mandell Bellmore and John C. Malone. Pathology of traveling-salesman subtour-elimination algorithms. *Oper. Res.*, 19(2):278–307, 1971. ISSN 0030-364X. doi: 10.1287/opre.19.2.278. URL `http://dx.doi.org/10.1287/opre.19.2.278`.

[3] Nicos Christofides. *Graph Theory: An Algorithmic Approach (Computer Science and Applied Mathematics)*. Academic Press, Inc., Orlando, FL, USA, 1975. ISBN 0121743500.

[4] George B. Dantzig, D. Ray Fulkerson, and Selmer M. Johnson. Solution of a large-scale traveling-salesman problem. *Operations Research*, 2(4):393–410, 1954. doi: 10.1287/opre.2.4.393. URL `http://dx.doi.org/10.1287/opre.2.4.393`.

[5] W.L. Eastman. *Linear Programming with Pattern Constraints*. PhD thesis, Harvard University, Cambridge, MA, 1958.

[6] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Bell Telephone Laboratories, Incorporated, United States of America, 1979.

[7] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[8] G. Laporte, H. Mercure, and Y. Nobert. An exact algorithm for the asymmetrical capacitated vehicle routing problem. *Networks*, 16(1):33–46, 1986.

[9] Eugene L. Lawler, David B. Shmoys, Alexander H. G. Rinnooy Kan, and Jan K. Lenstra. *The Traveling salesman problem : a guided tour of combinatorial optimization*. Wiley interscience series in discrete mathematics and optimization. J. Wiley and sons, Chichester, New York, Brisbane, 1987. ISBN 0-471-90413-9. URL `http://opac.inria.fr/record=b1117783`.

[10] John D. C. Little, Katta G. Murty, Dura W. Sweeney, and Caroline Karel. An algorithm for the traveling salesman problem. *Oper. Res.*, 11(6):972–989, December 1963. ISSN 0030-364X. doi: 10.1287/opre.11.6.972. URL `http://dx.doi.org/10.1287/opre.11.6.972`.

[11] Paolo Toth and Daniele Vigo, editors. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001. ISBN 0-89871-498-2.