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

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: ____________________ Signature: ______________________

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. The first day is day 0.
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 selected flight f_p with $A_p^a = A_i$ to any selected flight f_q with $A_q^d = A_i$, where f_q is immediately after f_p in a solution.
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, $\langle f_{i_1}, f_{i_2}, ..., f_{i_k} \rangle \subseteq F$, also called a *trip*. We say that s is valid if the flights in s have the following properties:

$$A^d_{i_1} = A^a_{i_k} = A_0 \tag{1}$$

$$A^a_{i_j} = A^d_{i_{j+1}}, \quad 0 < j < k \quad j \tag{2}$$

$$t_{i_j} + \Delta_{i_j} + C_r \leq t_{i_{j+1}}, \quad 0 < j \leq k, \quad \text{where } r = A^d_{i_{j+1}} \tag{3}$$

$$t_{i_k} + \Delta_{i_k} \leq T \tag{4}$$

$$\forall D_p \in D,\ \exists\ f_{i_j} \in s, \text{ such that } A^a_{i_j} = D_p \tag{5}$$

In this work, we refer to these properties as *trip properties*. 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 and the example is shown pictorially on Figure 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 $x + 1$.

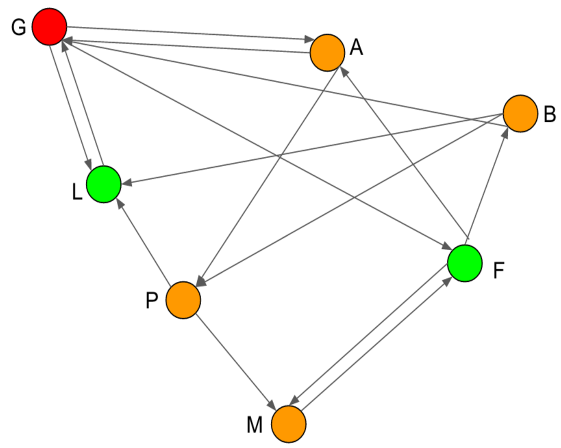

Figure 1: All airports in Example 1. The links between each two airports are available at certain dates and vary in price, as indicated in Table 1.

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

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

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 [21].

1. **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.

   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:

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

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

2. **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:

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

3. **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 $l_j \leq t_j \leq u_j$, $t_j < t_{j+1}$ for $(1 \leq j \leq n-1)$ and

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

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

   Instance. A set R of resources and a set N of jobs. Each job $J \in N$ consists of a sequence of operations O_J, a ready time rt_J and a deadline dt_J. Each operation i has processing time p_i and required resource r_i.

   Question. Is there a sequence $S = \{st_i : i \in J, \forall J \in N\}$ of starting times for every operation, such that every job meets its deadline and each resource is used by no more than one job at the same time?

5. **Time-Constrained Vehicle Routing Problem* (TCVRP)**

   Instance. A set A of n cities, where $D \in A$ is the *depot* and each city i has demand d_i, a fleet of m vehicles V, where vehicle k has capacity q_k, a travel cost $c(i, j)$ between two cities i and j, lower and upper bounds l_i and u_i respectively for each city i that specify its time window and a positive integer B.

   Question. Is there a set $\mathcal{S}$ consisting of m sequences $s_1, ..., s_m$, of the cities in A, called *tours* where $s_k = \langle A_{k(1)}, ..., A_{k(p)} \rangle$ and $\sum_{i=1}^{p} d_{A_{k(i)}} \leq q_k$ for $1 \leq k \leq m$ and $0 \leq p \leq n$ and $A_{k(1)} = D$ with $s_1 \cap s_2 ... \cap s_m = D$ and $s_1 \cup s_2 ... \cup s_m = A$, where each city $A_{k(i)}$ is visited at time t_i with $l_i \leq t_i \leq u_i$ and

$$\sum_{k=1}^{m} C(s_k) \leq B, \text{ for } C(s_k) = \left(\sum_{i=1}^{p-1} c(A_{k(i)}, A_{k(i+1)}) \right) + c(A_{k(p)}, A_{k(1)}) \ ?$$

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

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

6. **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.

5 Complexity of TP

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

Theorem 1. *TPD is NP-complete.*

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, defined in Section 4. Its NP-hardness follows by a reduction from the Hamiltonian Cycle problem. The proof is presented by Garey and Johnson [21].

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. 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.
- 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 $L_\gamma \leq B$. Without loss of generality, assume that $i_1 = 1$. 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 $A^d_{j_q} = A_{i_p}$ and $A^a_{j_q} = A_{i_{p+1}}$, where subscripts are taken modulo n. Therefore, s satisfies property (1) of a valid solution. For each q $(1 \leq q \leq n)$, $A^a_{j_q} = A^d_{j_{q+1}}$ and $t_{j_q} = q - 1$. We know that such flights exist in F by the construction of the set F.

From the construction of s, it follows that property (2) also holds. Properties (3) and (4) also hold, since we have chosen flights from F such that for every $f_{j_q} \in s$, $t_{j_q} = q - 1$ $((1 \leq q \leq n))$. 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 γ' = $\langle A_{i_1}, A_{i_2}, ..., A_{i_m} \rangle$. We will prove that γ' is a solution of π.

By construction of π', all 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 $(1 \leq p < n)$ is included more than once in γ'. Then s must contain more 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, and s is restricted to contain exactly n flights, that is $k = n$. The only way to visit n distinct 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 to π. Therefore, a solution to π' is also a solution to π.

The transformation from a TSP instance to an instance of TPD can be done in polynomial time. For each of the n(n-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 can be reduced to TPD in polynomial time, from which it follows that TPD is NP-complete.

□

6 Background Survey

This section presents a detailed review of some existing methods to approach NP-hard problems. In Section 6.1 we model TSP as a system of linear integer inequalities and showed a method for computing a lower bound on the cost of the tour that is widely used in the literature. Section 6.2 introduces Constraint Programming, where we discuss heuristics and give an example of how heuristics can be specifically designed for a given problem, using JSSP. In Section 6.4 and Section 6.5 we discuss some main approaches to solve two NP-hard problems that are somewhat similar to TP, namely TCTSP and TCTSP. We study 4 different NP-hard problems: TSP, JSSP, TCTSP and TCVRP, all defined in Section 4. The purpose of this is to identify successful techniques to model and solve hard problems that could be applied to TP.

6.1 Linear and Integer Programming

Linear Programming (LP) is a way to solve problems by modeling their requirements as a system of linear inequalities and equations and subject to them finding an optimal solution to the problem, expressed as a function that has to be either minimised or maximised. This function is called the *objective function*.

The field of LP is well-studied [31, 16] and it has many applications in the industry for resource optimisation [30, 22]. However, Dantzig [15] shows that there are many important optimisation problems that can not be modeled as a linear program, because their variables can take only integer values. Such problems can be modeled with the tools of integer programming (IP). Integer programming (IP) is an extension of linear programming (LP). The difference between LP and IP is that in the IP model variables are restricted to take only integer values.

The rest of this section presents different methods to solve problems with the tools offered by IP and comments on their performance by using TSP as an example problem.

6.1.1 Integer Programming Formulation of TSP

Dantzig et al. [17] formulate TSP as the following IP problem. Let $x(i,j)$ be a variable that denotes whether city j succeeds city i in a TSP tour. 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 *outgoing* city and j - the *incoming* city. The objective function of TSP is to minimise the total length of the tour, that is:

$$\min \sum_{i \in A} \sum_{j \in A} d(i,j)x(i,j) \tag{6}$$

In each TSP tour every city is visited once. Moreover, each city has to be incoming and outgoing exactly once. Therefore, the value of $x(i,j)$ for every i will be 1 for only one j and 0 for the rest. This can be enforced with the following constraint:

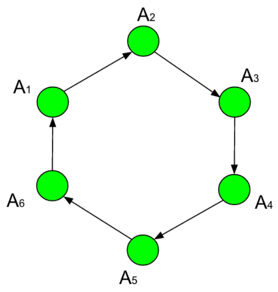

Figure 2: A valid TSP tour

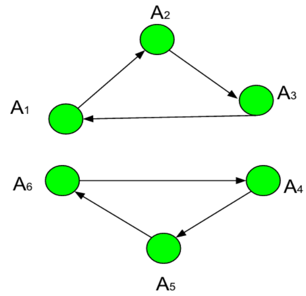

Figure 3: Two subtours of size 3

$$
\begin{aligned}
&\sum_{j \in A} x(i,j) = 1, \qquad i \in A,\\
&\sum_{i \in A} x(i,j) = 1, \qquad j \in A,
\end{aligned} \tag{7}
$$

Constraint (7) 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 2 and Figure 3. Both of them satisfy constraint (7). However, Figure 3 does not represent a valid tour, because it contains two subtours, each of size 3. To tackle this problem, Dantzig et al. [17] introduce the *subtour elimination* constraint, that is:

$$
\sum \{x(i,j) : (i,j) \in (S \times \bar{S}) \cup (\bar{S} \times S)\} \geq 2, \quad \forall \emptyset \subset S \subset A, \text{ where } \bar{S} = A \setminus S \tag{8}
$$

Here, S is a subset of A and $\bar{S}$ is the set of all cities that are in A and not in S. Constraint (8) enforces that at least two cities in S are connected with cities from $\bar{S}$. We give Figure 2 and Figure 3 as an example. Let $S = \{A_1, A_2, A_3\}$, then $\bar{S} = \{A_4, A_5, A_6\}$. Constraint (8), would detect the tour on Figure 3 as invalid, as there is no city in S that is connected to a city in $\bar{S}$. Figure 2 will be accepted, since A_1 and A_3 are connected with A_6 and A_4 respectively.

The formulation of the general TSP problem is given by the objective function (6), which has to be minimised, subject to constraints (7) and (8).

6.1.2 The Assignment Problem Relaxation

The assignment problem (AP) relaxation consists of removing the subtour elimination constraint from the TSP IP model and minimising the objective function (6) only subject to constraint (7). This is well known and extensively used relaxation [34, 20, 6, 4, 32]. In this section we explain how this method helps in solving NP-hard problems, using TSP as an example.

Let $\pi^\star$ be the resulting problem after performing AP relaxation on some instance π of TSP. The feasible solutions in π correspond to travelling salesman tours and we let $opt(\pi)$ denote an

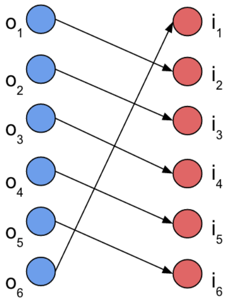

Figure 4: AP solution equivalent to Figure 2

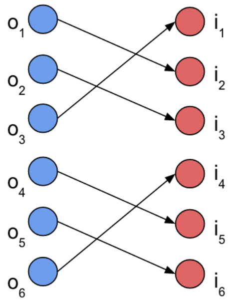

Figure 5: AP solution equivalent to Figure 3

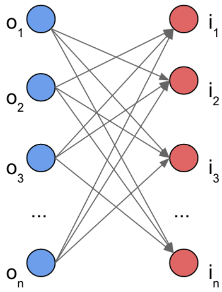

Figure 6: Problem $\pi^\star$ represented as a graph $G(V, E)$

optimal solution. The feasible solutions in $\pi^\star$ correspond to perfect matchings and we let $opt(\pi^\star)$ correspond to a perfect matching of minimum weight.

Problem $\pi^\star$ can be modelled as a bipartite graph $G(V, E)$, as shown on Figure 6. The vertex set of G is $V = O \cup I$, where $O = o_1, .., o_n$ and $I = i_1, ..., i_n$ both correspond to A. The set of edges is $E = \{(o_p, i_q) \,|\, a_p, a_q \in A,\ p \neq q\}$. Each edge $e = (o_p, i_q) \in E$ has weight w_e equal to $d(a_p, a_q)$ and it represents a link from a_p to a_q.

Figure 4 and Figure 5 are both examples of a perfect matching, where Figure 4 has $(o_1 \to i_2)$, $(o_2 \to i_3)$, $(o_3 \to i_4)$, $(o_4 \to i_5)$, $(o_5 \to i_6)$ and $(o_6 \to i_1)$. Every $sol(\pi^\star)$ can be translated to a path through the cities in π as follows: for every matched pair of vertices $(o_p \to i_q)$ in $sol(\pi^\star)$, it is sufficient to choose city a_q as the next visited city after a_p in π. For example, the matching in Figure 4 gives the tour shown in Figure 2 and the matching in Figure 5 is equivalent to the subtours in Figure 3.

Note that a feasible solution in π maps to a feasible solution in $\pi^\star$, but the converse need not be true in general, as the example in Figure 5 shows.

Let $opt(\pi^\star)$ have total edge weight equal to α. Suppose that $opt(\pi^\star)$ does not map to a tour in π. Therefore, $opt(\pi)$ must have length at least α. Conversely, suppose that $opt(\pi^\star)$ can be mapped to $opt(\pi)$. Therefore, the length of $opt(\pi)$ is equal to α. Hence, the length of $opt(\pi)$ is greater than or equal to the value of $opt(\pi^\star)$. In the existing work, this property is used to derive a lower bound on the cost of $opt(\pi)$ [34, 17, 4]. This approach is frequently combined with a branch and bound technique [33], which is discussed in Section 6.3.

Dantzig et al. [17] solve the constructed TSP model using a novel method for that time, which they call the "cutting-plane method". We point the interested reader back to [17] and [14], where in the latter the cutting-plane method is explained in more detail using a 10 city TSP instance as an example.

6.2 Constraint Programming

Constraint Programming (CP) is a widely used method for solving optimisation problems [11, 35, 36]. Given an instance π of an optimisation problem, we first model it as a constraint satisfaction problem (CSP) $CSP(\pi)$. Problem $CSP(\pi)$ consists of a set of decision variables V, each with a set of possible values, called its *domain*, and a set of rules, called *constraints* concerning the assignment of domain values to variables. A *solution* to π is an assignment of each variable in V to a value in its domain, such that all constraints are satisfied. The program that searches for a solution $CSP(\pi)$ is called a *solver*.

Whenever the domain dom_v of a variable v is empty in some temporal assignment of variables, we say that we have a *domain wipeout* and this variable's assignment is regarded as invalid. The *degree* of v is the number of constraints that involve v and at least one other unassigned variable.

Finding a solution to $CSP(\pi)$ involves a search through the assignments of the variables in $CSP(\pi)$. Finding an optimal solution or proving that no solution exists requires iterating over the entire search space in the worst case. There are multiple techniques to speed up the search, such as identifying and discarding poor temporal variable assignments from further investigation, implementing intelligent search methods, or adding *heuristics* for variable and value assignments. We discuss some heuristics that could be applicable to TP in the next section.

6.2.1 Heuristics

Heuristics are rules concerning the ordering of variables or assignments of values, used to "guide" the search, that aim to limit the total explored search space. Previous work has shown that heuristics can have a significant effect on search effort [27, 24]. However, they are not guaranteed to always work. Heuristics are applicable for the cases when the solution is not required to be an optimum and when the particular problem instance has no solutions. In the latter, heuristics can help to prove early in the search that a given partial solution leads to a domain wipeout. This section presents some of the most well-known heuristics and gives an example of two successful heuristics that are tailored specifically for the Job-Shop Scheduling Problem (JSSP), defined in Section 4.

Research effort is spent on understanding heuristics and the properties that improve their effectiveness. Beck et al. [5] develop a framework for analysing the effectiveness of heuristics. Hooker and Vinay [28] prove that heuristics that create simpler subproblems are successful in general, and heuristics that create subproblems that are more likely to be satisfiable are usually "bad". Haralick and Elliott [26] propose the intuition that "to succeed, try first where you are most likely to fail", known as the *fail-first principle*. It suggests that the variable to be assigned next should be the one which is most-likely to lead to a domain wipeout of some variable.

The fail-first principle is a basis for the development of various heuristics. For instance, Golomb and Baumert [25] propose a heuristic `dom` that chooses next assigned variable to be the one with the smallest number of values remaining in its domain. Brélaz [10] introduce a new generalisation of `dom`, denoted as `dom+deg`, which chooses the variable with the smallest number of values remaining in its domain, breaking ties on highest variable degree. Another generalisation of `dom` is `dom/deg`, which divides the domain size of a variable by the degree by its degree and chooses the variable that gives minimal value [7]. All of these are *variable ordering* heuristics, because they determine the next explored variable during search.

Slack-Based Heuristics

Slack-based heuristics were introduced by Smith and Cheng [38] for the JSSP, defined in Section 4. They help in determining how to sequence a given pair of operations in an instance of JSSP, modelled as CSP. There are various ways to model JSSP as CSP, which is a significant area of research on its own. We refer the interested reader to Rossi et al. [37, Chapter 22] for an extensive discussion. This section explains the main principles of the slack-based heuristics, comments on their performance and discusses our hypothesis that it could be applicable to TP (which is the main reason why they are discussed here).

Let π_J be an instance of the JSSP and let $CSP(\pi_J)$ be constructed using some CSP model for JSSP, which includes a variable $o(i,j)$ that determines the ordering of any pair of operations i and j in π_J. The value of $o(i,j)$ is 1 when i is scheduled before j, or 0 otherwise. There are four possible restrictions for the values of $o(i,j)$, imposed by the constraints in $CSP(\pi_J)$:

1. $o(i,j)$ can only be 1, i.e. i is before j in the current variable assignment.
2. $o(i,j)$ can only be 0, i.e. j is before i in the current variable assignment.
3. $o(i,j)$ can be neither 0 nor 1, i.e. there is a domain wipeout.
4. $o(i,j)$ can be either 0 or 1, i.e. there is no restriction on the ordering of i and j.

Slack-based heuristics are only applicable for case 4 and thus we skip discussion of the first three possibilities.

For every $o(i,j)$ that can be either 0 or 1, Smith and Cheng [38] compute $slack(i,j)$, which is the time remaining after ordering i before j and similarly $slack(j,i)$, as shown by the pink bars in Figure 7. If process i is sequenced before process j, then $slack(i,j)$ indicates the time window within which j has to be completed. Two different heuristics for the sequencing of i and j, based on the size of the time window are introduced: *min-slack* and *max-slack*.

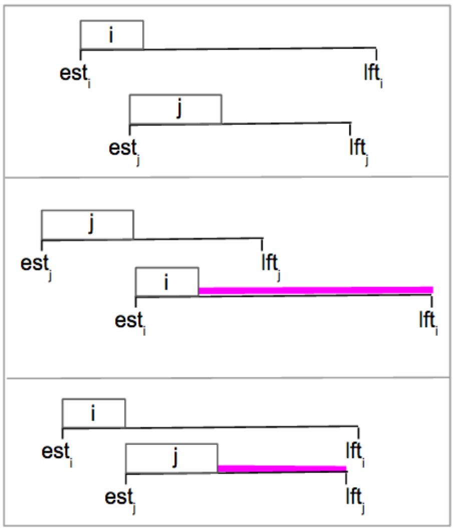

Figure 7: The values of $slack(i, j)$ (the length of the bottom pink line) and $slack(j, i)$ (the length of the top pink line), where est_x and lft_x refer to the earliest start and finish time of an operation x respectively.

Min-slack ordering selects an ordering of i and j that gives minimum time flexibility for the tasks. For instance, min-slack heuristics would assign $o(i, j) = 1$ for the example in Figure 7. Max-slack heuristic orders the operations, such that there is greatest time flexibility for the tasks. In the example in Figure 7, max-slack would assign $o(i, j) = 0$, since $slack(j, i) > slack(i, j)$.

We relate the idea behind min-slack to the principles of the `dom` heuristic. If the starting times of i and j are variables, then the assignment of $o(i, j)$ based on min-slack would decrease the domain of the starting time of the second scheduled process. As opposed to min-slack, max-slack would maximize the number of possible starting times. We recognise that this idea can be traced back to the work of Geelen [23] which proposes a principle that each variable should be assigned the least constraining value in its domain.

Smith and Cheng [38] compare the slack-based heuristics with two solution procedures over the same suite of benchmark problems and report on obtaining "comparable results at very low computational expense". In Rossi et al. [37, p. 105], slack-based heuristics are described as "effective". However, Crawford and Baker [13] argue that their effectiveness is mainly due to their problem representation method.

6.3 Branch and Bound Algorithms

6.4 Algorithms for Time-Constrained TSP

This section discusses some existing approaches to model and solve the Time-Constrained TSP (TCTSP). Although there is substantial amount of work done on TCTSP, most of the attention in the literature seems to be for TSP [33] and many of the methods to solve TCTSP are TSP models that are adapted to include time windows constraints. We first explain a popular method to model TCTSP and briefly outline few existing solution techniques. The second part of the section gives an example of an exact algorithm for TCTSP.

A disjunctive graph for TCTSP is a *mixed* graph $G = (V, E, A)$, where V is the set of vertices, representing all cities in the problem, E is the set of the undirected edges and A is the set of directed edges. For every two vertices i and j, $e(i, j) \in E$ if and only if i and j can be visited in either order, and $e(i, j) \in A$ if and only if i must be visited before j. The disjunctive graph is used extensively for TCTSP [33].

Lawler et al. [33] present a summary of the most used techniques to model TCTSP. One of the discussed models is similar to the TSP IP formulation in Section 6.1.1 with each of the variables adapted to consider specific time periods. Another model is based on the disjunctive graph model with dual labels on vertices, that is, each label is a tuple (i, t), meaning that city i is visited at time t. The work by Baker [4] presented in Section 6.4.1 is also an example of a solution that utilizes this model.

Arigliano et al. [3] work on TCTSP with asymmetric distances and constructs an integer linear programming model using a variation of the cutting planes algorithm by Dantzig et al. [17]. The paper argues that the limits of existing TCTSP algorithms like [4] is that they assume that the travel time is constant. Arigliano et al. [3] make TCTSP seem more related to real-world problems by introducing the notion of *travel speed*, *congestion factor* and *travel time*, where the latter is a function of the former two variables.

Hurkala [29] is a recent work that is believed to be the first to consider problems with multiple time windows. The paper agrees with Arigliano et al. [3] that the constant travel time assumption is unrealistic and the TCTSP model is adapted to changing travel times. As well as developing a novel algorithm for TCTSP with multiple time windows and changing travel time, it presents and empirical evaluation of three metaheuristics, which are often utilised for scheduling problems.

6.4.1 An Exact Algorithm for TCTSP

In this Section we discuss an exact algorithm for TCTSP introduced by Baker [4], based on a branch and bound procedure and on the disjunctive graph model. The model assumes that time is a scalar transformation of distance, i.e. time and distance may be used interchangeably. Baker [4] algorithm works for instances with symmetric inter city distances under the triangle inequality, where we say that the distance between two cities i and j is *symmetric* if $d(i, j) = d(j, i)$.

The term *time window* for a city i refers to the period of time between l_i and u_i in which i has to be visited. Baker [4] introduces a variable t_i for each city i that determines the time i is visited in the tour and an additional variable t_{n+1} that determines the time at which the tour is completed. The objective function of the problem is:

$$\text{minimise } \{t_{n+1} - t_1\} \tag{9}$$

subject to a set of 5 constraints that enforce that there is sufficient travel time between each two subsequent cities in the tour and that the time each city is visited is within the required time window. The model allows for the traveller to arrive at city i at time $t_i < l_i$, provided that she waits in the city until the time window opens before leaving i. Moreover, although a single upper and lower bound on the time window is assumed, the model could be easily extended to include sets of distinct time windows for each city [4].

The objective function of the model indirectly imposes the constraint that the number of visits to each city has to be one. Suppose the TCTSP algorithm returned a tour containing city i more than once. Then the tour would not be optimal, because one could do better by cutting one of the visits to i until i is visited only once [4].

Baker [4] models TCTSP as a disjunctive graph and uses a relaxation of the TCTSP model that is equivalent to the longest path problem[5] in a disjunctive graph with $n+1$ vertices. The solution to the relaxed problem is used as a lower bound on the tour distance during the bounding procedure of the branch and bound algorithm.

Baker [4] enforces that the time windows constraint by adding dual labelling on the vertices in the disjunctive graph within the branch and bound procedure. The second label of each vertex i is the value of the longest path from vertex 1 to i and this value is precisely t_i.

The experiments described in [4] show that this approach is effective on small to moderate-sized instances and that its efficiency greatly depends on the number of overlapping time windows. The results show that instances with greater number of overlapping time windows are easier to solve. We conjecture that the reason for this could be that the model is more constrained and therefore the number of possible subproblems that have to be checked during the branch and bound procedure is smaller.

6.5 Algorithms for the Time-Constrained Vehicle Routing Problem

This section outlines the main approaches to the Time-Constrained Vehicle Routing Problem (TCVRP), defined in Section 4.

A common approach in the literature is to model VRP and TCVRP as a *set partitioning problem* [18, 1, 19, 2]. This can be done as follows. Let $\mathcal{R}$ be the set of all feasible routes, where c_r denotes the cost of a route $r \in R$. Let $\delta i, r$ and x_r be two variables that can be either 0 or 1. For each $r \in R$, $\delta i, r$ is 1 if r visits city i and x_r is 1 if r is used in the solution. Then, TCVRP can be formulated as the problem of choosing a set $\mathcal{S} \subset R$ with minimal cost, known as the *set partitioning problem* (SPP), that is:

[4]This is true, since the distances are under the triangle inequality.

[5]The longest path problem asks to find a path of maximum length in a given graph.

$$\text{minimise} \sum_{r \in R} c_r x_r, \tag{10}$$

$$\sum_{r \in R} \delta_{i,r} x_r = 1, \quad \forall i \in A, \tag{11}$$

where constraint (11) imposes that each city is part of a set which is used in the solution. Some algorithms then construct a LP relaxation of this formulation and use the solution of the relaxation as a lower bound for a specific branch and bound procedure [18, 1].

The method outlined above is somewhat similar to the AP LP relaxation for TSP and the relaxation for TCTSP discussed earlier. This shows that this general approach, dating back to the work of Dantzig et al. [17], is highly applicable to different NP-hard problems.

There is a number of approaches based on heuristics [2, 12]. Bräysy and Gendreau [8, 9] show the importance of good heuristics for tackling TCVRP. They present a review of some route construction and route improvement heuristics and attempt to outline a guideline for heuristics evaluation.

Other algorithms for TCVRP include approximation algorithms and dynamic programming approaches. For more detailed review, we refer the interested reader further to Toth et al. [39], where the practical aspects of TCVRP are discussed and common methods to solve TCVRP as well as some of its other variations are outlined.

6.6 Summary

This section outlines some observations from the literature survey.

Existing literature shows that most approaches to NP-hard problems generally follow a similar pattern. For instance, this often is a relaxation to some other problem, the solution of which is used as a lower bound in a branch and bound procedure. All studied problems have successful solutions that employ IP and CP. Heuristics are topic of research and they are highly applicable for optimisation problems in general.

A successful methodology for one problem gives positive results when adapted and applied to a different problem. This can be explained as a consequence from a result of the NP-completeness theory, that is, if a polynomial time algorithm for one NP-complete problem is found, then all of them are solvable in polynomial time.

From the two observations above we can deduce that:

- A successful method for some NP-hard problem might give positive results for TP.

- If an efficient algorithm for TP is found, then it might give good results when applied to other problems.

Thus, our proposed approach is based on IP and CP: the two techniques that tend to give good results on the studied problems.

7 Proposed Approach

We model TP as Integer and Constraint Programming problems. The Constraint Programming (CP) approach is explained in Section 7.1 and the Integer Programming (IP) model is presented in Section 7.2.

7.1 CP Model

Let $m = |F|$. We introduce an array $\mathcal{S}$ of size $m + 1$ (indexed $0, 1, ..., m$) that represents the TP tour and a variable z with domain $dom_z = \{1, ..., m\}$ to denote the number of flights in the trip. If $\mathcal{S}[i] = j$, then flight $f_j \in F$ is the $(i+1)^{th}$ flight in the tour, for $(0 \leq i < m)$ and $(1 \leq j \leq m)$. To denote the end of the tour, we set $\mathcal{S}[z] = 0$. All subsequent variables in $\mathcal{S}$ will then have to be 0. Each variable v in $\mathcal{S}$ is either 0, if no flight is taken at that step, or it is equal to some flight number, that is $dom_v = \{0, ..., m\}$.

TP can then be formulated as the problem of minimising the objective function:

$$\sum_{i=0}^{m-1} c_{\mathcal{S}[i]} \tag{12}$$

subject to constraints (13-19).

$$\mathcal{S}[i] > 0 \wedge \mathcal{S}[i+1] = 0, \quad (0 \leq i \leq z-1) \tag{13}$$

$$\text{allDiff}(\mathcal{S}[0], ..., \mathcal{S}[z-1]) \tag{14}$$

Constraint (13) restricts that once the end of the trip is reached at some position z, no flights are further added to $\mathcal{S}$. Constraint (14) enforces that every flight is taken only once, using an all-different constraint [40].

The trip properties are added to the model as follows:

$$dom_{\mathcal{S}[0]} = \{j \in \{1, ..., m\} : A_j^d = A_0\} \tag{15}$$

$$dom_{\mathcal{S}[z-1]} = \{j \in \{1, ..., m\} : A_j^a = A_0\}$$

$$dom_{\mathcal{S}[i]} = \{j \in \{1, ..., m\} : A_j^d = A_p^a,\ p = \mathcal{S}[i-1]\}, \quad (1 \leq i < z) \tag{16}$$

$$t_p + \Delta_p + C_r \leq t_q, \text{ where } p = \mathcal{S}[i], q = \mathcal{S}[i+1], r = A_q^a, \ (0 \leq i < z) \tag{17}$$

$$t_q + \Delta_q \leq T, \quad \text{ where } q = \mathcal{S}[z-1] \tag{18}$$

$$\forall A_k \in D, \ |\{i : (0 \leq i < z) \wedge A_{\mathcal{S}[i]}^a = A_k\}| > 0 \tag{19}$$

Constraint (15) is equivalent to trip property (1). It restricts the domains of the first and the $(z-1)^{th}$ variable in $\mathcal{S}$ to contain only flights that depart from/arrive at the home point. Trip property (2) is enforced by constraint (16). The domain of each variable in $\mathcal{S}$ is set to include only flights that depart from the arrival airport of the previous flight. Constraints (17) and (18) correspond to trip properties (3) and (4) respectively. Constraint (19) restricts that the number of the flights that arrive at every destination in the trip is positive, and thus enforces trip property (5).

7.2 IP Model

Most of the constraints for the TP CP model need modification in order to be applicable to an IP model. In particular, IP does not allow for "if-then", "all-different" and other constraints that are not integer linear inequalities. The model described in this section is a modification of the TP CP model, described in the previous section, that gives constraints in the form of integer linear inequalities, which we also refer to as *constraints*.

Let $m = |F|$. We introduce a variable $x_{i,j}$ for every $i \in \{0, ..., m-1\}$ and $j \in \{1, ..., m\}$, such that $x_{i,j} = 1$ if $(i+1)^{th}$ flight is f_j or 0 otherwise. This variable is somewhat similar to $\mathcal{S}$, used for the CP model, where $\mathcal{S}[i] = p$ is equivalent to $x_{i,p} = 1$. In addition, we introduce a variable $x_{m,0} = 1$, where flight f_0 is a "special" flight with duration $\Delta_j = 0$, date $t_j = T$, departure and arrival airports $A_j^d = A_j^a = A_0$ and cost $c_j = 0$, added to F.

The objective function of TP is to minimise:

$$\sum_{i}^{m-1} \sum_{j=0}^{m} c_{j=1} x_{i,j} \tag{20}$$

subject to constraints (21-27).

First, we restrict that only one flight is taken at each step i with constraint (21). Constraint (22) is equivalent to the all-different constraint (14): it enforces every flight to be taken at most once.

$$\forall i\,(0 \leq i < m), \quad \sum_{j=1}^{m} x_{i,j} = 1 \tag{21}$$

$$\forall j\,(0 \leq j \leq m), \quad \sum_{i=0}^{m-1} x_{i,j} = 1 \tag{22}$$

Our method to express the properties of a trip as integer linear inequalities is as follows. Assume that flight $z-1$ is the final flight returning to A_0, where $1 \leq z \leq m$. We add $m-z+1$ many f_0 flights, so that the variables $x_{z,0}, ..., x_{m,0}$ are all equal to 1. We do not add connection time when connecting from A_0 to A_0 as part of these flights.

Constraints (23) and (24) enforce trip property (1). The first flight is restricted to depart from the home point and the last flight must arrive at the home point. From trip property (2) it follows that there must exist some $z < m$, $x_{z,j} = 1$, for which $j \neq 0$ and $A^a_j = A_0$. Thus, constraint (24) indirectly enforces trip property (1).

$$\sum_{j \in S_1} x_{0,j} = 1, \quad \text{where } S_1 = \{j \in \{1, ..., m\} : A^d_j = A_0\} \tag{23}$$

$$x_{m,0} = 1 \tag{24}$$

Property (2) is enforced by defining two sets of flights S_1 and S_2, such that all flights in S_2 depart from the arrival airport of S_1. Constraint (25) restricts that every flight always departs from the arrival airport of the previous taken flight.

$$\sum_{j \in S_1} x_{i-1,j} = \sum_{j' \in S_2} x_{i,j'}, \quad \forall i\,(1 \leq i \leq m) \wedge \forall y \in A, \text{ where} \tag{25}$$

$$S_1 = \{j \in \{0, ..., m\} : A^a_j = y\} \text{ and } S_2 = \{j' \in \{0, ..., m\} : A^d_{j'} = y\}$$

Trip properties (3) and (4) are represented by constraint (26), which enforces that every flight f_j cannot depart until the previous flight $f_{j'}$ has arrived, adding connection time, if needed. The connection time C'_r is equal to the connection time of the arrival airport, unless the current flight is f_0.

$$x_{i+1,j} + \sum_{j' \in F'} x_{i,j'} \leq 1, \quad \forall i\,(0 \leq i < m) \text{ and } \forall j\,(0 \leq j \leq m) \tag{26}$$

$$\text{where } F' = \{j' : f_{j'} \in F \wedge t_{j'} + \Delta_{j'} + C'_r > t_j\} \text{ and}$$

$$C'_r = \begin{cases} 0, & \text{if } j = 0 \\ C_{A^a_{j'}}, & \text{otherwise} \end{cases}$$

Trip property (5) is enforced by restricting that all destination airports must be visited by at least one flight:

$$\forall A_k \in D, \sum_{i=0}^{m-1} \sum_{j \in F_k} x_{i,j} \geq 1, \quad \text{where } F_k = \{j : f_j \in F \wedge A_j^a = A_k\} \tag{27}$$

7.3 Modelling soft and hard constraints

7.4 Choice of CP and IP solvers

7.5 Experimental Datasets

This section describes the instances that will be used for the empirical analysis of our algorithms for TP.

7.5.1 Skyscanner Flights API

Skyscanner is a metasearch engine that enables people to find and compare flights, hotels and cars for hire in terms of price, date and duration. They offer us access to their in-house flights API. Given a departure and an arrival airport and a date, it returns a set of flights with these attributes.

The Skyscanner flights API will be used during the empirical analysis to construct TP instances with varying size and properties. Testing our algorithms on these instances will show the effectiveness of our algorithms on real-world problems.

Generating instances from the Skyscanner data

7.5.2 Randomly Generated TP instances

8 Work Plan

Implement the IP and CP model for TP. (mid December-mid January)

Implement support for the soft and hard constraints. (mid January - mid February)

Generate TP instances for empirical evaluation. (mid February - 1st March)

Run the algorithms on the test data and analyse results. (March-April)

Submission deadline: 21 April 2017, 12PM

References

[1] Yogesh Agarwal, Kamlesh Mathur, and Harvey M. Salkin. A set-partitioning-based exact algorithm for the vehicle routing problem. *Networks*, 19(7):731–749, 1989. doi: 10.1002/net.3230190702. URL `http://dx.doi.org/10.1002/net.3230190702`.

[2] Guilherme Bastos Alvarenga, Geraldo Robson Mateus, and G. de Tomi. A genetic and set partitioning two-phase approach for the vehicle routing problem with time windows. *Computers & OR*, 34(6):1561–1584, 2007. doi: 10.1016/j.cor.2005.07.025. URL `http://dx.doi.org/10.1016/j.cor.2005.07.025`.

[3] Anna Arigliano, Gianpaolo Ghiani, Antonio Grieco, and Emanuela Guerriero. Time dependent traveling salesman problem with time windows: Properties and an exact algorithm. 2015.

[4] 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`.

[5] J. Christopher Beck, Patrick Prosser, and Richard J. Wallace. Toward understanding variable ordering heuristics for constraint satisfaction problems. In *IN: PROCEEDINGS OF THE FOURTEENTH IRISH ARTIFICIAL INTELLIGENCE AND COGNITIVE SCIENCE CONFERENCE*, pages 11–16, 2003.

[6] 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`.

[7] Christian Bessiere and Jean-Charles Régin. Mac and combined heuristics: Two reasons to forsake fc (and cbj?) on hard problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 61–75. Springer, 1996.

[8] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part I: route construction and local search algorithms. *Transportation Science*, 39(1):104–118, 2005. doi: 10.1287/trsc.1030.0056. URL `http://dx.doi.org/10.1287/trsc.1030.0056`.

[9] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part II: metaheuristics. *Transportation Science*, 39(1):119–139, 2005. doi: 10.1287/trsc.1030.0057. URL `http://dx.doi.org/10.1287/trsc.1030.0057`.

[10] Daniel Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

[11] Yves Caseau and Franois Laburthe. Solving small tsps with constraints, 1997.

[12] Chi-Bin Cheng and Keng-Pin Wang. Solving a vehicle routing problem with time windows by a decomposition technique and a genetic algorithm. *Expert Syst. Appl.*, 36(4):7758–7763, May 2009. ISSN 0957-4174. doi: 10.1016/j.eswa.2008.09.001. URL `http://dx.doi.org/10.1016/j.eswa.2008.09.001`.

[13] James M Crawford and Andrew B Baker. Experiment al results on the application of sat isfiability algorithms to scheduling problems. 1994.

[14] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson. On a linear-programming, combinatorial approach to the traveling-salesman problem. *Oper. Res.*, 7(1):58–66, February 1959. ISSN 0030-364X. doi: 10.1287/opre.7.1.58. URL `http://dx.doi.org/10.1287/opre.7.1.58`.

[15] George B. Dantzig. On the significance of solving linear programming problems with some integer variables. *Econometrica*, 28(1):30–44, 1960. ISSN 00129682, 14680262. URL `http://www.jstor.org/stable/1905292`.

[16] George B. Dantzig. *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`.

[17] 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`.

[18] Martin Desrochers, Jacques Desrosiers, and Marius Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2): 342–354, 1992. doi: 10.1287/opre.40.2.342. URL `http://dx.doi.org/10.1287/opre.40.2.342`.

[19] Jacques Desrosiers, François Soumis, and Martin Desrochers. Routing with time windows by column generation. *Networks*, 14(4):545–565, 1984. doi: 10.1002/net.3230140406. URL `http://dx.doi.org/10.1002/net.3230140406`.

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

[21] 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.

[22] W. W. Garvin, H. W. Crandall, J. B. John, and R. A. Spellman. Applications of linear programming in the oil industry. *Manage. Sci.*, 3(4):407–430, July 1957. ISSN 0025-1909. doi: 10.1287/mnsc.3.4.407. URL `http://dx.doi.org/10.1287/mnsc.3.4.407`.

[23] Pieter Andreas Geelen. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proceedings of the 10th European Conference on Artificial Intelligence*, ECAI '92, pages 31–35, New York, NY, USA, 1992. John Wiley & Sons, Inc. ISBN 0-471-93608-1. URL `http://dl.acm.org/citation.cfm?id=145448.145491`.

[24] Ian P. Gent, Ewan MacIntyre, Patrick Prosser, Barbara M. Smith, and Toby Walsh. An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In *Proceedings of the Second International Conference on Principles and Practice*

of Constraint Programming, Cambridge, Massachusetts, USA, August 19-22, 1996, pages 179–193, 1996. doi: 10.1007/3-540-61551-2_74. URL http://dx.doi.org/10.1007/3-540-61551-2_74.

[25] Solomon W Golomb and Leonard D Baumert. Backtrack programming. *Journal of the ACM (JACM)*, 12(4):516–524, 1965.

[26] Robert M Haralick and Gordon L Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial intelligence*, 14(3):263–313, 1980.

[27] Robert M. Haralick and Gordon L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artif. Intell.*, 14(3):263–313, 1980. doi: 10.1016/0004-3702(80)90051-X. URL http://dx.doi.org/10.1016/0004-3702(80)90051-X.

[28] John N Hooker and V Vinay. Branching rules for satisfiability. *Journal of Automated Reasoning*, 15(3):359–383, 1995.

[29] Jaroslaw Hurkala. Time-dependent traveling salesman problem with multiple time windows. In *Position Papers of the 2015 Federated Conference on Computer Science and Information Systems, FedCSIS 2015, Lódz, Poland, September 13-16, 2015.*, pages 71–78, 2015. doi: 10.15439/2015F311. URL http://dx.doi.org/10.15439/2015F311.

[30] P. M. Jacovkis, H. Gradowczyk, A. M. Freisztav, and E. G. Tabak. A linear programming approach to water-resources optimization. *Zeitschrift für Operations Research*, 33(5):341–362, 1989. ISSN 1432-5217. doi: 10.1007/BF01416081. URL http://dx.doi.org/10.1007/BF01416081.

[31] L. V. Kantorovich. Mathematical methods of organizing and planning production. *Management Science*, pages 366–422, 1960.

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

[33] 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.

[34] 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.

[35] David Manlove, Gregg O'Malley, Patrick Prosser, and Chris Unsworth. A constraint programming approach to the hospitals / residents problem. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 4th International Conference, CPAIOR 2007, Brussels, Belgium, May 23-26, 2007, Proceedings*, pages 155–170, 2007. doi: 10.1007/978-3-540-72397-4_12. URL http://dx.doi.org/10.1007/978-3-540-72397-4_12.

[36] Kevin Mcdonald and Patrick Prosser. A case study of constraint programming for configuration problems, 2002.

[37] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[38] Stephen F. Smith and Cheng-Chung Cheng. Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings of the 11th National Conference on Artificial Intelligence. Washington, DC, USA, July 11-15, 1993.*, pages 139–144, 1993. URL `http://www.aaai.org/Library/AAAI/1993/aaai93-022.php`.

[39] Paolo Toth, Daniele Vigo, Paolo Toth, and Daniele Vigo. *Vehicle Routing: Problems, Methods, and Applications, Second Edition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2014. ISBN 1611973589, 9781611973587.

[40] Willem Jan van Hoeve. The alldifferent constraint: A survey. *CoRR*, cs.PL/0105015, 2001. URL `http://arxiv.org/abs/cs.PL/0105015`.