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

The Travellers Problem (TP) is a combinatorial optimisation problem whose extensions and variations are often encountered by travellers around the world. Given a set of airports, a set of flights, a set of destinations that is a subset of the airports, and a special airport A_0, a solution is a travel schedule that starts and finishes at A_0, visits all destinations and is in accordance with additional constraints specified by the traveller. For instance, the traveller may wish to spend a certain amount of days in each destination, to take a minimum number of connection flights, or to give minimum amount of money for flights. This work gives a formal description of TP and some of its main extensions, proves a result concerning its complexity, investigates existing methods to solve similar problems and presents a plan of action for modelling TP, generating TP datasets from the Skyscanner flights data and carrying out empirical evaluation.

This work is organised as follows. Section 2 gives a formal definition of TP and Section 3 presents some example TP instances. Section 4 lists the formulation of all problems studied in this work. Section 5 justifies the complexity of TP. Section 6 presents the background survey, where we discuss common techniques to solve problems similar to TP with respect to their complexity and formulation. Section 7 presents the prioritised project requirements. Our proposed approach is described in Section 8. It includes two distinct models for TP, evaluation plan and a discussion about how datasets for experiments will be created. Finally, Section 9 presents a detailed plan of work and time estimation of each task of high priority.

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:

$$A_{i_1}^d = A_{i_k}^a = A_0 \tag{1}$$

$$A_{i_j}^a = A_{i_{j+1}}^d, \quad 0 < j < k \quad j \tag{2}$$

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

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

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

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.

1. 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.
2. 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.
3. 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:

1. 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.
2. Travellers may wish to avoid taking connection flights. In such requirement, we wish to maximise the number of flights to and from destinations.
3. 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.

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.

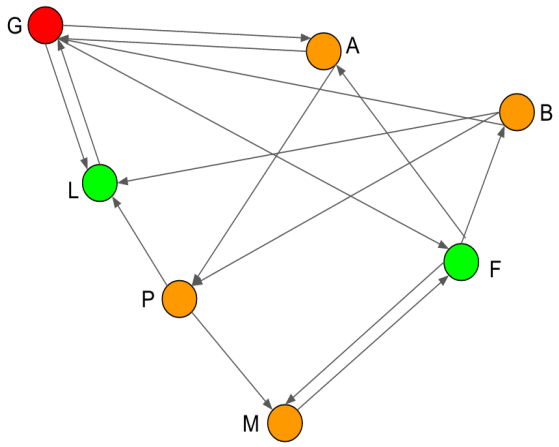


Figure 1: All airports in Example 1. Green vertices are connection airports, orange vertices are destinations and the red vertex is the home point. The links between each two airports are available at certain dates and vary in price, as indicated in Table 1.

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 $x + 1$.

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.

	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.

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 Investigated 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 [24].

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

   Instance. A set A of n cities, where $D \in A$ is the *depot* and each city i has demand c_i, a fleet of m vehicles V, where vehicle k has capacity q_k, a distance $d(i, j)$ between two cities i and j 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 for every vehicle k $(1 \leq k \leq m)$:

   - $s_k = \langle A_{s_{k,1}}, ..., A_{s_{k,p_k}} \rangle$, where p_k is the number of the cities in the tour of k,
   - $\sum_{i=1}^{p} c_{A_{s_{k,i}}} \leq q_k$,
   - $A_{s_{k,1}} = D$,
   - $s_i \cap s_j = \{D\}, \quad (1 \leq i < j \leq m)$,
   - $s_1 \cup s_2 ... \cup s_m = A$,

   and

   $$\sum_{k=1}^{m} C(s_k) \leq B, \text{ where } C(s_k) = \left(\sum_{i=1}^{p_k} d(A_{s_{k,i}}, A_{s_{k,i+1}}) \right) + d(A_{s_{k,kp}}, A_{s_{k,1}}) \text{ ?}$$

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 $\gamma = \langle A_{\pi_1}, ..., A_{\pi_n} \rangle$ 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 $\gamma = \langle A_{\pi_1}, ..., A_{\pi_n} \rangle$ 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, where time is a scalar transformation of distance.

   Question. Is there a permutation of cities $\gamma = \langle A_{\pi_1}, ..., A_{\pi_n} \rangle$ of A, such that each city A_{π_i} is visited at time t_{π_i}, where $l_{\pi_i} \leq t_{\pi_i} \leq u_{\pi_i}$, $t_{\pi_i} \geq t_{\pi_{i-1}} + d(A_{\pi_{i-1}}, A_{\pi_i})$ for $(2 \leq i \leq n)$, $t_{\pi_1} \geq t_{\pi_n} + d(A_{\pi_n}, A_{\pi_1})$, and

$$L_\gamma = \left(\sum_{i=1}^{n-1} d(A_{\pi_i}, A_{\pi_{i+1}}) \right) + d(A_{\pi_n}, A_{\pi_1}) \leq B \quad ?$$

5. **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?

6. **Time-Constrained Vehicle Routing Problem* (TCVRP)** [4]

   Instance. A set A of n cities, where $D \in A$ is the *depot* and each city i has demand c_i, a fleet of m vehicles V, where vehicle k has capacity q_k, a distance $d(i, j)$ 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, where time is a scalar transformation of distance, 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 for every vehicle k $(1 \leq k \leq m)$:

[2] Note that this problem is a special case of TSP.
[3] Note that this problem is a generalisation of TSP.
[4] Note that this problem is a generalisation of VRP.

- $s_k = \langle A_{s_{k,1}}, ..., A_{s_{k,p_k}} \rangle$, where p_k is the number of the cities in the tour of k,
- $\sum_{i=1}^{p} c_{A_{s_{k,i}}} \leq q_k$,
- $A_{s_{k,1}} = D$,
- $s_i \cap s_j = \{D\}, \quad (1 \leq i < j \leq m)$,
- $s_1 \cup s_2 ... \cup s_m = A$,
- each city $A_{s_{k,i}}$ is visited at time $t_{s_{k,i}}$ with $l_{s_{k,i}} \leq t_{s_{k,i}} \leq u_{s_{k,i}}$,
- $t_{s_{k,i}} \geq t_{s_{k,i-1}} + d(A_{s_{k,i}}, A_{s_{k,i-1}}), \quad (2 \leq i \leq n)$[5],

and

$$\sum_{k=1}^{m} C(s_k) \leq B, \text{ where } C(s_k) = \left(\sum_{i=1}^{p_k} d(A_{s_{k,i}}, A_{s_{k,i+1}}) \right) + d(A_{s_{k,k_p}}, A_{s_{k,1}}) \text{ ?}$$

7. **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 [24].

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.

[5] We do not care about the time window on the depot

- 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 $A_{j_q}^d = A_{i_p}$ and $A_{j_q}^a = A_{i_{p+1}}$, where subscripts are taken modulo n. Therefore, s satisfies property (1) of a valid solution. For each q $(1 \leq q \leq n)$, $A_{j_q}^a = A_{j_{q+1}}^d$ 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 $\gamma' = \langle A_{i_1}, A_{i_2}, ..., A_{i_m} \rangle$. We will prove that γ' is a solution of π.

By construction of π', all 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 $A_{j_1}^d = A_{j_n}^a = A_1$. 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 $d(A_k^d, A_k^a)$ 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 $n(n\text{-}1)/2$ 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 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 6.2 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 6.3 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 Branch and Bound Algorithms

The term "branch and bound" was first coined by Little et al. [40] and it refers to a widely-used method for solving optimisation problems. Given an instance π of an optimisation problem, the method repeatedly breaks up the set of candidate solutions into successively smaller subsets, called *subproblems* and calculates a bound on their value. If the bound of a subproblem is "worse" than the best one found so far, the subproblem is discarded. Otherwise, it is added for future investigation. If its bound is the "best" found so far, its value is remembered and used for comparison with future subproblems. The process of generating subproblems is called *branching*. We illustrate the branch and bound procedure with the next example.

Example 3. Let π be the following problem:

$$\text{minimise } x + y,$$

where $x \in \{1, 3, 6\}$ and $y \in \{0, 5\}$.

Figure 2 illustrates two possible executions of the branch and bound procedure, depending on the branching rule. Each execution is represented by a tree, where nodes represents subproblems

Figure 2: All subproblems for π checked by a branch and bound procedure with two different branching rules.

and edges represents variable assignments. The root represents the entire solution set, the green node is an optimum, red nodes denote subproblem sets that were discarded because they have bounds that are "worse" compared to the "best" bound found so far. Without loss of generality, we assume that the nodes in each tree are visited in a pre-order traversal. For each node i that represents a partial solution, a lower bound l_i on its value is calculated, where l_i is the sum of the value of all assigned variables. If i represents a single feasible solution, an upper bound u_i is calculated, equal to the sum of the assigned value to each variable.

In the left-most tree, the branching rule explores the feasible solutions, assigning values for x and y in a non-increasing order. One can see that it needs to check all possible assignments of x and y in order to find an optimum. This is due to the poor choice of branching rule, which leads to a series of ineffective bounds which can not discard subproblems.

The right-most tree has branching rule that explores the feasible solutions, assigning values for x and y in a non-decreasing order. This is more efficient branching rule for π, since the size of the tree is smaller. This is because the first encountered feasible solution is an optimal assignment for x and y, which sets 1 as an upper bound on the value of the feasible solutions, which discards partial solutions.

As shown in Example 3, the choice of bounding and branching procedures is crucial to the size of the explored search space. Existing literature has spent substantial effort in constructing efficient algorithms for branching and bounding [40, 19, 31]. Examples of such algorithms are discussed throughout this work for the studied NP-hard problems.

Branch and bound algorithms are one of the most successful methods for solving optimisation problems [14] and they are typically implemented in Constraints Programming (CP) and Integer Programming (IP) solvers [39, 45].

6.2 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 [36, 18] and it has many applications in the industry for resource

optimisation [34, 25]. However, Dantzig [17] 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.

6.2.1 Integer Programming Formulation of TSP

Dantzig et al. [19] 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:

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

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 3 and Figure 4. Both of them satisfy constraint (7). However, Figure 4 does not represent a valid tour, because it contains two subtours, each of size 3. To tackle this problem, Dantzig et al. [19] 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 3 and Figure 4 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 4 as invalid, as there is no city in S that is connected to a city in $\bar{S}$. Figure 3 will be accepted, since A_1 and A_3 are connected with A_6 and A_4 respectively.

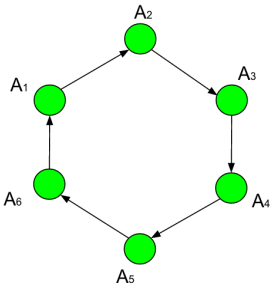


Figure 3: A valid TSP tour

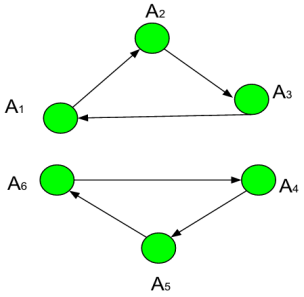


Figure 4: Two subtours of size 3

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).

6.2.2 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 [23, 35, 22, 37]. 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 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 7. 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 5 and Figure 6 are both examples of a perfect matching, where Figure 5 has $(o_1 \rightarrow i_2)$, $(o_2 \rightarrow i_3)$, $(o_3 \rightarrow i_4)$, $(o_4 \rightarrow i_5)$, $(o_5 \rightarrow i_6)$ and $(o_6 \rightarrow i_1)$. Every $sol(\pi^\star)$ can be translated to a path through the cities in π as follows: for every matched pair of vertices $(o_p \rightarrow i_q)$ 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 5 gives the tour shown in Figure 3 and the matching in Figure 6 is equivalent to the subtours in Figure 4.

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 6 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

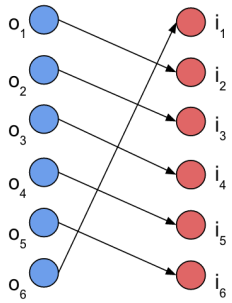


Figure 5: AP solution equivalent to Figure 3

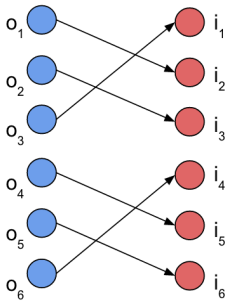


Figure 6: AP solution equivalent to Figure 4

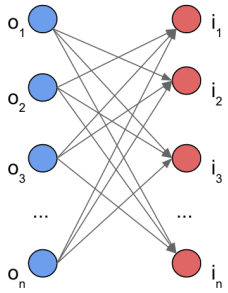


Figure 7: Problem $\pi^{\star}$ represented as a graph $G(V, E)$

bound on the cost of $opt(\pi)$ [35, 19, 16]. This approach is frequently combined with a branch and bound technique [38], which is discussed in Section 6.1.

Dantzig et al. [19] 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 [19] and [16], where in the latter the cutting-plane method is explained in more detail using a 10 city TSP instance as an example.

The number of solved instances using the AP relaxation varies. Dantzig et al. [19] executes the algorithm by hand on a 49 city problem instance and there is no data on the computational performance of the algorithm. Jonker et al. [35] report on experiments with up to 150 city instances solved. However, the algorithm's running time is not specified. The greatest number of cities solved by Fischetti and Toth [23] is 200. The longest time taken to solve a 200 city instance is 4.42 seconds, using a VAX 11/780 computer architecture.

6.3 Constraint Programming

Constraint Programming (CP) is a widely used method for solving optimisation problems [12, 41, 43]. 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.

6.3.1 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 [30, 27]. 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. [7] develop a framework for analysing the effectiveness of heuristics. Hooker and Vinay [32] 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 [29] 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 [28] propose a heuristic `dom` that chooses next assigned variable to be the one with the smallest number of values remaining in its domain. Brélaz [11] 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 [8]. 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 [47] 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. [46, 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 [47] 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 8. 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*.

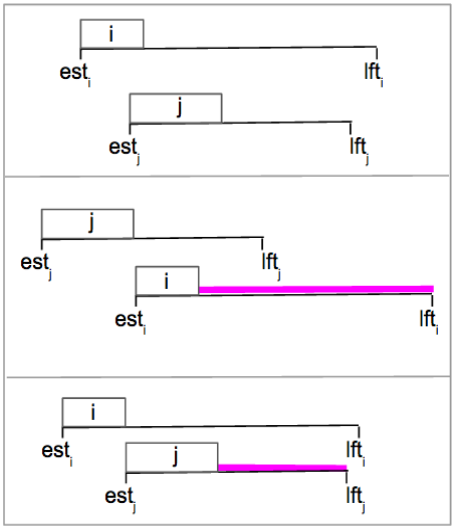


Figure 8: The values of $slack(i, j)$ (the length of the bottom pink line) and $slack(j, i)$ (the length of the top pink line), where est_x and lft_x refer to the earliest start and finish time of an operation x respectively.

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 8. Max-slack heuristic orders the operations, such that there is greatest time flexibility for the tasks. In the example in Figure 8, 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 [26] which proposes a principle that each variable should be assigned the least constraining value in its domain.

Smith and Cheng [47] 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. [46, p. 105], slack-based heuristics are described as "effective". However, Crawford and Baker [15] argue that their effectiveness is mainly due to their problem representation method.

6.4 Algorithms for the Time-Constrained TSP

Time-Constraint TSP (TCTSP), also known as TSP with Time Windows (TSPTW) is a generalisation of TSP, defined in Section 4. Although there is a substantial amount of work done on TCTSP, most of the attention in the literature seems to be for TSP [38] and many techniques for TCTSP are TSP models, adapted to include time windows constraints. This section is a discussion of common approaches to model and solve TCTSP.

Lawler et al. [38] present a summary of the most used techniques to model TCTSP. One of them is similar to the TSP IP formulation in Section 6.2.1, having each of the variables adapted to consider specific time periods. Lawler et al. [38] also discuss the dual model introduced by Baker [6] as an example of an effective technique.

Baker [6] works with instances with symmetric inter city distances under the triangle inequality. TCTSP is modelled as a system of linear integer inequalities, solved by a branch and bound procedure with bounds calculated using an algorithm for the longest path problem [6].

Baker [6] conducts experiments using 20 problem instances of size 9, 13, 22, 30 or 51 cities with varying percent of overlapping windows.

Arigliano et al. [4] work on TCTSP with asymmetric distances and construct an integer linear programming model using a variation of the cutting planes algorithm. The paper argues that a limitation of existing TCTSP algorithms such as [6] is that they assume that the travel time is constant. Arigliano et al. [4] 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. Arigliano et al. [4] can solve instances with up to 40 cities.

Hurkala [33] 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. [4] that the constant travel time assumption is unrealistic and the TCTSP model is also adapted to changing travel times. Hurkala [33] introduces a novel algorithm for this re-formulation of TCTSP and combines it with three already existing metaheuristics to produce three TCTSP solvers. For the empirical evaluation, Hurkala [33] uses 45 randomly generated instances with 13 to 23 cities, which is almost twice as small as the maximum number of cities that Arigliano et al. [4] can solve. This suggests that multiple time windows add additional complexity to the problem.

6.5 Algorithms for the Time-Constrained Vehicle Routing Problem

This section outlines the main approaches to the Time-Constrained Vehicle Routing Problem (TCVRP), often known as the Vehicle Routing Problem with Time Windows (VRPTW). TCVRP is a generalisation of TSP and it is defined in Section 4.

A common approach in the literature is to model VRP and TCVRP as a *set partitioning problem* [20, 2, 21, 3]. 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:

$$\text{minimise} \sum_{r \in R} c_r x_r, \tag{9}$$

[6] The longest path problem is the problem of finding a path of maximum length in a graph with no repeated vertices.

$$\sum_{r \in R} \delta_{i,r} x_r = 1, \quad \forall i \in A \setminus \{D\}, \tag{10}$$

where constraint (16) 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 [20, 2].

The method outlined above is somewhat similar to the AP LP relaxation for TSP. This shows that this general approach, dating back to the work of Dantzig et al. [19], is highly applicable to different NP-hard problems.

Some approaches are based on heuristics [48, 13]. Bräysy and Gendreau [9, 10] 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 the evaluation of heuristics. For more detailed review, we refer the interested reader further to Toth et al. [49], where the practical aspects of TCVRP are discussed and common methods to solve TCVRP as well as some of its other variations are outlined.

The number of cites in a solved instance varies across the different algorithms. For the algorithms based on the set partitioning model, Agarwal et al. [2] solve instances with up to 25 cities in their computational experiments. Desrosiers et al. [21] work with real-world bus transportation problems and they solve instances with up to 55 schools, where schools are equivalent to cities. Desrochers et al. [20] is the first work capable of optimally solving instances with 100 cities. They say that "... this problem size is six times larger than any reported to date by other published research." Heuristics-based approaches can also find solutions to instances with up to 100 cities [48]. However, one should note that these solutions need not be optimal.

6.6 Summary

This section outlines the main observations from the literature survey and shows their influence on the proposed approach in Section 8.

Most of the existing approaches to NP-hard problems generally follow a similar pattern. For instance, this can be a relaxation to another problem which need not be polynomial-time solvable in order to obtain a lower bound on the solution, used as part of a branch and bound procedure. Many algorithms model the problem as a constraint satisfaction problem or as a system of linear integer inequalities.

The average size of TSP instances tackled by existing algorithms outweighs the size of the solvable instances of the other studied problems, namely VRP, TCTSP and TCVRP for which TSP is a special case. TCVRP is a generalisation of each of TSP, VRP and TCTSP and it is the problem with smallest maximum size of solved instances by the existing literature. This observation is later used in Section 8.5 when deciding on feasible sizes of the datasets.

	Must Have (M)	Should Have (S)	Could Have (C)
1	Implement the TP CP model	Implement all soft and hard constraints specified in the proposal.	Define additional soft and hard constraints and implement them.
2	Implement the TP IP model	Evaluate the added soft and hard constraints	Improve the implemented algorithms, based on the evaluation results.
3	Implement one soft or hard constraint from Section 2		Generate Random TP instances
4	Implement a basic user interface (UI) that enables the user to interact with the TP solver		Evaluate the improved versions of the algorithms.
5	Prepare the datasets for the evaluation		Add some heuristics to the model.
6	Run the algorithms for each instance and collect appropriate metrics.		
7	Analyse the results.		
8	Write a paper about the project		
9	Create and deliver project presentation		

Figure 9: Project requirements.

When it is not necessarily vital that a given instance of TP is solved to optimality, there are algorithms that can return their "best" solution found so far after a given period of time. CP or IP solvers for instance offer this flexibility. Typically, much larger instances can be solved, since in many cases the algorithm does not need to explore the entire search space to prove the optimality of a solution, that may have been found relatively early on in search (and which itself may even be optimal). This will be investigated during the project evaluation.

7 Project Requirements

This section lists the project requirements. They are separated into three categories: must have, should have and could have, based on the MoSCoW technique for requirements prioritization [5]. Requirements under the *must have* category are critical to the success of the project. *Should have* requirements are important, but not necessary for the successful completion of the project. *Could have* requirements are desirable, but not necessary and their purpose is to improve the overall quality of the project. Figure 9 shows all project requirements. To refer to a specific requirement we use its *id*, which is the string composed of the first letter of the requirement category (M, S or C) and the number of the row the requirement is in. For instance, requirement "Define additional soft and hard constraints and implement them" has id $C1$. This list is later referred to in Section 9 where we describe the work plan.

8 Proposed Approach

Based on the analysis of existing work in Section 6, we model TP as Integer and Constraint Programming problems. The Constraint Programming (CP) approach is explained in Section 8.1 and the Integer Programming (IP) model is presented in Section 8.2. Having two distinct TP models would allow for each of the models to be used as an additional verification step for the

correctness of the other. Since optimal solutions of each TP instance have equal numeric value, if TP CP and TP IP return the same numeric value of an optimal solution for a large number of sufficiently-large instances, this will provide strong evidence that both models are correct. A difference in their solutions would indicate a mistake in either of them. Having more than one TP algorithm also gives opportunities for performing more detailed evaluation, learn more about the problem and about CP and IP.

8.1 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:

$$\sum_{i=0}^{m-1} c_{\mathcal{S}[i]} \tag{11}$$

subject to constraints (12-18).

$$\mathcal{S}[i] > 0 \wedge \mathcal{S}[i+1] = 0, \quad (0 \leq i \leq z-1) \tag{12}$$

$$\text{allDiff}(\mathcal{S}[0], ..., \mathcal{S}[z-1]) \tag{13}$$

Constraint (12) restricts that once the end of the trip is reached at some position z, no flights are further added to $\mathcal{S}$. Constraint (13) enforces that every flight is taken only once, using an all-different constraint [50].

The trip properties are added to the model as follows:

$$dom_{\mathcal{S}[0]} = \{j \in \{1, ..., m\} : A^d_j = A_0\} \tag{14}$$

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

$$dom_{\mathcal{S}[i]} = \{j \in \{1, ..., m\} : A^d_j = A^a_p,\ p = \mathcal{S}[i-1]\}, \quad (1 \leq i < z) \tag{15}$$

$$t_p + \Delta_p + C_r \leq t_q, \text{ where } p = \mathcal{S}[i], q = \mathcal{S}[i+1], r = A^a_q,\ (0 \leq i < z) \tag{16}$$

$$t_q + \Delta_q \leq T, \quad \text{where } q = \mathcal{S}[z-1] \tag{17}$$

$$\forall A_k \in D, |\{i : (0 \leq i < z) \wedge A^a_{\mathcal{S}[i]} = A_k\}| > 0 \tag{18}$$

Constraint (14) 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 (15). 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 (16) and (17) correspond to trip properties (3) and (4) respectively. Constraint (18) restricts that the number of the flights that arrive at every destination in the trip is positive, and thus enforces trip property (5).

8.2 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 $A^d_j = A^a_j = A_0$ and cost $c_j = 0$, added to F.

The objective function of TP is to minimise:

$$\sum_{i}^{m-1} \sum_{j=0}^{m} c_{j=1} x_{i,j} \tag{19}$$

subject to constraints (20-26).

First, we restrict that only one flight is taken at each step i with constraint (20). Constraint (21) is equivalent to the all-different constraint (13): it enforces every flight to be taken at most once.

$$\forall i \, (0 \leq i < m), \quad \sum_{j=1}^{m} x_{i,j} = 1 \tag{20}$$

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

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 (22) and (23) 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 $A_j^a = A_0$. Thus, constraint (23) indirectly enforces that $A_j^a = A_0$.

$$\sum_{j \in S_1} x_{0,j} = 1, \quad \text{where } S_1 = \{j \in \{1, ..., m\} : A_j^d = A_0\} \tag{22}$$

$$x_{m,0} = 1 \tag{23}$$

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 (26) restricts that every flight always departs from the arrival airport of the previous taken flight.

$$\sum_{j \in S_1} x_{i-1,j} = \sum_{j' \in S_2} x_{i,j'}, \quad \forall i \, (1 \leq i \leq m) \wedge \forall y \in A, \text{ where} \tag{24}$$

$$S_1 = \{j \in \{0, ..., m\} : A_j^a = y\} \text{ and } S_2 = \{j' \in \{0, ..., m\} : A_{j'}^d = y\}$$

Trip properties (3) and (4) are represented by constraint (31), which enforces that every flight f_j cannot depart until the previous flight $f_{j'}$ has arrived, adding connection time, if needed.

$$x_{i+1,j} + \sum_{j' \in F'} x_{i,j'} \leq 1, \quad \forall i \, (0 \leq i < m) \text{ and } \forall j \, (0 \leq j \leq m) \tag{25}$$

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

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

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

$$\forall A_k \in D, \sum_{i=0}^{m-1} \sum_{j \in F_k} x_{i,j} \geq 1, \quad \text{where } F_k = \{j : f_j \in F \wedge A_j^a = A_k\} \tag{26}$$

8.3 Modelling soft and hard constraints

This section shows how soft and hard constraints in Section 2 can be added to the constructed models for TP, using hard constraint 2 in Section 2.1 as an example.

Let $U \subseteq D$ be the set of all destinations which must be visited at some specific date. The TP formulation allows for destination airports to be used as connections. Therefore, pruning all flights departing from or arriving at a constrained destination A_i before or respectively after the required date τ_i for A_i potentially removes feasible solutions. Instead, we add constraint (27) which ensures that for every destination A_i in U, there is a scheduled flight arriving at A_i before the required date τ_i and the next scheduled flight departs from A_i after τ_i.

$$\forall A_i \in U, \exists j\, (0 \leq j < z), \text{ such that:} \tag{27}$$
$$A^a_{\mathcal{S}[j]} = A_i,$$
$$t_{\mathcal{S}[j]} + \Delta_{\mathcal{S}[j]} + C_i < \tau_i, \text{ and}$$
$$t_{\mathcal{S}[j+1]} > \tau_i + 1$$ [7]

Constraint (27) is defined as an addition to the CP model presented in Section 8.1. Other required soft or hard constraints can be enforced using similar methods for either of the two models.

8.4 Choice of CP and IP solvers

We were unable to find a solver that supports both IP and CP and this is the reason why the TP IP and CP models will be implemented using two different solvers. In order to decrease the implementation difference between the two models as much as possible, we have chosen solvers that support the same language, namely Java.

Choco [44] is an open-source Java library for Constraint Programming that has wide range of already implemented constraints and heuristics. One only states the variables and constraints and then the problem is solved using specific internally implemented search algorithms. The TP CP model will be implemented using Choco.

Gurobi is a mixed-integer programming solver first released in 2009. It is widely used mathematical optimisation tool [1] and it is licensed freely to academia. We will use Gurobi for the implementation of the TP IP model.

Both Gurobi and Choco are well-documented with good support from the community and easy to set up and use, which are the main reasons why we choose to use them.

[7] We assume that the traveller has to stay at A_i for at least one day after the required date τ_i.

8.5 Evaluation

This section is the plan of the project evaluation. The main goal of the evaluation is to analyse the implemented algorithms for TP and learn more about the problem and its extensions and variations. To achieve this, it is important to create instances that are good representatives of instances of TP that are likely to be encountered in practice and to measure appropriate metrics. Section 8.5.1 presents our choice of evaluation metrics and Section 8.5.2 gives more background on the experimental datasets.

8.5.1 Evaluation Measurements

The main metrics that will be taken when running each of the algorithms are the following:

1. The number of *search nodes* taken to solve each instance.
2. The running time of each of the algorithms for each instance.
3. The "best" returned solution when running for a given period of time and how far from an optimum it is.
4. The number of optimal solutions for each instance.
5. The number of airports, the number of destinations, the size of the flights set and the travel time period of the largest instances.

The number of *search nodes* denotes the number of recursive calls taken to find a solution if the problem is soluble or prove that the problem is insoluble. This metric is used by the existing literature to measure the hardness of an instance [42].

For metric 3, we measure how far from an optimum the returned solution is by comparing it with an optimal solution, returned after running the exact TP algorithms. We plan to find optimal solutions with respect to the following objective functions:

- Minimise the sum of the cost of each flight;
- Minimise the length of the flights schedule;
- Minimise the number of airports taken as connections;
- Minimise the time spent on flying;
- Minimise the connection time;
- Minimise a set of the aforementioned objectives.

By collecting metrics 1 and 2, we aim to learn more about how the size of TP instances influences the behaviour of the implemented algorithms. Metric 3 measures the trade-off between computation time and optimality of a solution. Metric 4 shows some of the properties of the datasets. Metric 5 aims to show the properties of the largest solvable instance.

8.5.2 Experimental Datasets

This section describes the planned procedures for generating the datasets for evaluation and their desired properties.

The size of the TP instances

The TP instances can be divided in four main categories with respect to their size. Category 1 instances should be easy to solve by hand and they will be constructed for testing purposes. Category 2 should be of "medium" size: neither too easy, nor too hard to solve in terms of search nodes and running time. Category 3 should consist of larger and thus more challenging instances. Category 4 instances will aim to test the maximum capability of the evaluated algorithms.

We note that the size of TP instances without including any additional soft and hard constraints can vary with respect to three main properties: the number of flights m, the holiday duration T and the number of airports n, where n is equal to the sum of the number of destinations n_d, connection airports n_c plus 1 (for the home point A_0). A TP instance can be small with respect to some parameters and large with respect to others. We will aim to generate a dataset that captures this variety.

Our decision on the size of the instances from each of the categories is based on the size of the related problems, solved by the existing literature. Problems belonging to category 1 will have $n \leq 10$, where $n_d \leq 4$, $T \leq 10$ and $m \leq 20$. Category 2 TP instances will have $10 \leq n \leq 30$, where $4 \leq n_d \leq 8$, $10 \leq T \leq 20$ and $30 \leq m \leq 50$. Category 3 problems will have $30 \leq n \leq 60$, where $15 \leq n_d \leq 30$, $20 \leq T \leq 30$ and $300 \leq m \leq 500$. The size of category 4 instances will be decided based on the running time results with the previous three categories. Taking the largest instance from category 3, we will iteratively increase its size and run the implemented algorithms with it until the instance becomes insoluble.

Constructing TP instances from the Skyscanner flights data

Skyscanner is a metasearch engine that enables people to find and compare flights in terms of price, date and duration. Skyscanner offers us some of their flights data for the project empirical evaluation. The data that we will be given consists of all flights that depart from and arrive at any airport in the world for the course of two months. Each flight f'_j has departure and arrival airport A^d_j and A^a_j respectively, duration Δ'_j, price c'_j, departure date g'_j and departure time q'_j. Each flight f'_j in the given data has to be transformed to a valid TP flight f_j. This can be done as follows.

1. For each f'_j, associate A^d_j and A^a_j to f_j.
2. For each f'_j, obtain Δ_j, equal to Δ'_j, represented as a fraction of a day.
3. Store the departure date of the earliest flight in the data in a separate variable T_0. For each flight f'_j, g_j is equal to the number of days difference between g'_j and T_0.
4. For each f'_j, obtain q_j, equal to q'_j, represented as a fraction of a day.
5. For each f_j, $t_j = g_j + q_j$.
6. Choose some currency ϵ. For each f'_j, obtain c_j, equal to c'_j converted to ϵ.

Let $\mathcal{F}$ be the transformed Skyscanner flights data and let $\mathcal{A}$ be the set of all airports in the world. With the defined size ranges for each of the four categories, we compose A by picking airports from $\mathcal{A}$ and then compose F by picking flights from $\mathcal{F}$ that only fly to and from airports in A and depart at time greater than or equal to T'_0 and arrive by time that is less than or equal to $T'_0 + T$, where T'_0 is an arbitrary chosen integer between 0 and the number of days difference between the earliest and the latest flight in $\mathcal{F}$.

Due to the expected enormous size of $\mathcal{F}$, we need to be specifically careful not to construct only instances without any solution, because this would not allow for taking all planned measurements. We will need to ensure that F contains at least one flight from and to each destination [8]. Ideally, the dataset should contain few instances without a solution and the majority of instances with more than one solution.

Randomly Generated TP instances

This task is of low priority and it will be done only after all high priority tasks are finished earlier than expected.

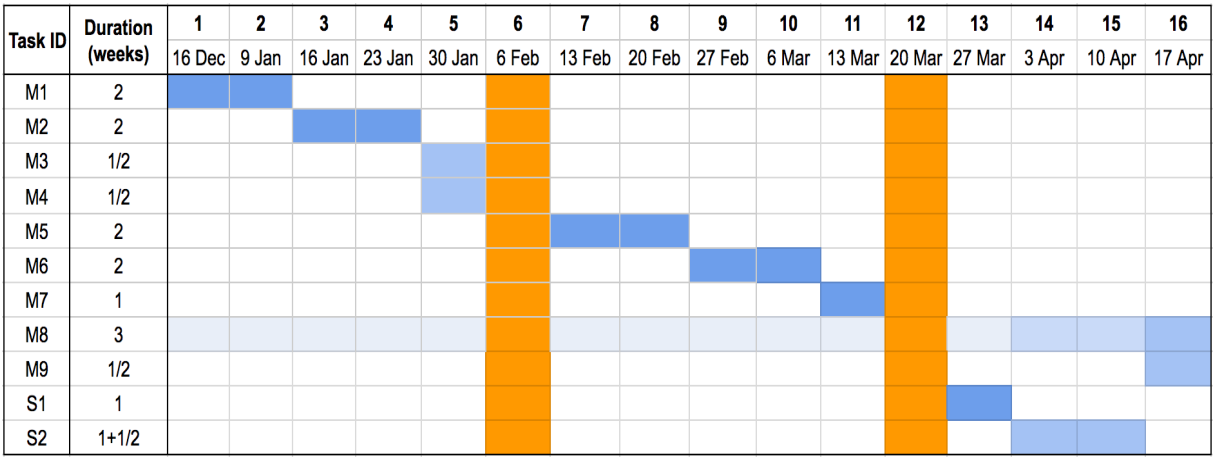
9 Work Plan

Figure 10 presents a plan of work on the must-have and should-have tasks for the remainder of this project, assuming that the project proposal is finished by 16^{th} of December [9]. There are 16 weeks of work until the submission deadline, which is on 21^{st} of April. Each week is equal to 4 full work days dedicated to this project on average. The must-have and should-have tasks from the requirements capture in Figure 9 are listed in column 1 in Figure 10, referenced by their ids. Column 2 is a time estimation for each corresponding task in terms of weeks. The shade of blue is an indication of the amount of effort planned to be spent on the particular task for the particular

[8] Even if this is the case, instances can still have no solutions.

[9] However, the proposal submission deadline is 18^{th} of December

Figure 10: The time estimation and plan of completion for each project requirement of high importance.



week. For instance, the project paper (task $M7$) will be written gradually every week with more emphasis during week 16.

Note that the task estimations are mainly based on previous experience and the actual time taken for each task may vary. Weeks 9 and 15, both marked in orange, have no assigned tasks. This time is allocated for finishing up some tasks that require more than the predicted time. In case all tasks are finished within the estimated time, these two weeks will be spent on implementing some of the could-have requirements, shown in Figure 9.

References

[1] Industries using Gurobi. URL http://www.gurobi.com/products/industries/industry-overview.

[2] 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.

[3] 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.

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

[5] Steve Ash. Moscow prioritisation. *DSDM Consortium*, 2007.

[6] 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.

[7] J. Christopher Beck, Patrick Prosser, and Richard J. Wallace. Toward understanding variable ordering heuristics for constraint satisfaction problems. In *Proceedings of the 14th Irish Artificial Intelligence and Cognitive Science Conference*, pages 11–16, 2003.

[8] 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.

[9] 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.

[10] 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.

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

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

[13] 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.

[14] Nicos Christofides, Aristide Mingozzi, and Paolo Toth. State-space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11(2):145–164, 1981. doi: 10.1002/net.3230110207. URL http://dx.doi.org/10.1002/net.3230110207.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] Matteo Fischetti and Paolo Toth. An additive bounding procedure for the asymmetric travelling salesman problem. *Math. Program.*, 53(2):173–197, January 1992. ISSN 0025-5610. doi: 10.1007/BF01585701. URL http://dx.doi.org/10.1007/BF01585701.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

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

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

[30] 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.

[31] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970. doi: 10.1287/opre.18.6.1138. URL http://dx.doi.org/10.1287/opre.18.6.1138.

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

[33] 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.

[34] 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.

[35] R. Jonker, G. De Leve, J. A. Van Der Velde, and A. Volgenant. Technical note-rounding symmetric traveling salesman problems with an asymmetric assignment problem. *Oper. Res.*, 28(3-part-i):623–627, June 1980. ISSN 0030-364X. doi: 10.1287/opre.28.3.623. URL http://dx.doi.org/10.1287/opre.28.3.623.

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

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

[38] 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.

[39] Sven Leyffer. Integrating SQP and branch-and-bound for mixed integer nonlinear programming. *Comp. Opt. and Appl.*, 18(3):295–309, 2001. doi: 10.1023/A:1011241421041. URL http://dx.doi.org/10.1023/A:1011241421041.

[40] 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.

[41] 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.

[42] Ciaran McCreesh, Patrick Prosser, and James Trimble. Heuristics and really hard instances for subgraph isomorphism problems. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 631–638, 2016. URL http://www.ijcai.org/Abstract/16/096.

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

[44] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016. URL http://www.choco-solver.org.

[45] Ignacio Quesada and Ignacio E Grossmann. An lp/nlp based branch and bound algorithm for convex minlp optimization problems. *Computers & chemical engineering*, 16(10-11): 937–947, 1992.

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

[47] 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.

[48] K. C. Tan, L. H. Lee, and K. Q. Zhu. Heuristic methods for vehicle routing problem with time windows. In *In Proceedings of the 6th AI and Math*, pages 281–295, 1999.

[49] 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.

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