University of Glasgow | School of Computing Science

Algorithms for Multi-City Travel Itineraries

Iva Babukova (2030458b)

September 23, 2019

ABSTRACT

The Traveller's Problem (TP) is a combinatorial optimisation problem that is 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 home 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. These may require spending a certain amount of days in each destination or taking a minimum number of connecting flights for example. The review of existing literature indicates that TP is unlikely to have been studied: we were unable to find a formal description of TP or an attempt to solve it. This work formally defines TP and constructs two solutions for the problem using Integer Programming (CP) and Constraint Programming (CP). We also build a framework for creating TP instances using flight data given by a flight metasearch engine called Skyscanner. The evaluation of the models shows that the IP model is faster then the CP model for all instances. The results further suggest that both models are not efficient enough to be used as part of a real-time flight booking platform: the number of flights in the instance has significant impact on the running time of the algorithms. We suggest ways to improve the performance of the models that can be carried out as part of future work.

1. INTRODUCTION

1.1 Problem Motivation

Online travel planning is the process of booking holidays over the Internet using one or more websites that offer details about available itineraries. It is one of the fastest growing sectors in the travel industry. [20]. At the moment, online booking systems provide no efficient solutions for finding itineraries involving multiple cities. If a traveller wants to visit several destinations with unspecified order and dates, they have to manually query for all permutations of dates and destinations in order to find the best trip. This easily turns into an intractable task and travellers are often forced to book suboptimal trips.

TP can be classified as a member of the family of routing problems. *Routing Problems* are a set of problems within the area of combinatorial optimisation and operational research that ask for an optimal route through a set of destinations that satisfies all time window, capacity and any additional instance-specific constraints. Examples of routing problems are the Travelling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP). Routing problems find direct applications in multiple areas: transportation and logistics [22], planning inspections of remote sites [34], genome sequencing [4], data clustering [45] and many others.

This work is organised as follows. Section 1.2 formulates TP and other routing problems discussed throughout this work. Existing work is presented in Section 2. The design and implementation of the TP algorithms and the TP instance generator are discussed in Section 3 and Section 4 respectively. Section 5 presents the empirical analysis of the algorithms. Section 6 presents a summary of this work, gives suggestions for improvement and proposes future directions this project can take.

1.2 Routing Problems Formulations

This section includes formulations of TP and two existing NP-hard [23] routing problems studied throughout this work, namely the Travelling Salesman Problem and the Vehicle-Routing Problem.

1.2.1 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

$$\sum_{k=1}^{m} C(s_k) \leq B, \text{ where}$$

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

1.2.2 Travelling Salesman Problem (TSP)

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 $\gamma = \langle A_{\pi_1}, ..., A_{\pi_n} \rangle$ of A such that the total travel distance L_γ:

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

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

1.2.3 Traveller's Problem (TP)

Instance. Each instance of TP consists of:

- 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 trip in, visit as a desired destination, or connect in on the way to their destination.
- A set of *destinations* $D = \{D_1, ..., D_l\}$, $D \subseteq A$, $l \leq n$.
- The trip starts and ends at the same airport A_0, which is referred to as the *home airport.*
- The total travel time T, within which the traveller must have visited all destinations and returned to the home airport. The first day is day 0.
- A set of flights $F = \{f_0, ..., f_{m-1}\}$. Each flight f_j has a departure airport A^d_j, an arrival airport A^a_j, 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^d_j to A^a_j. 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.
- Each airport A_i has *connecting time* C_{A_i}, that is the time that takes to switch from any selected flight f_p with $A^a_p = A_i$ to any selected flight f_q with $A^d_q = A_i$, where f_q is immediately after f_p in a solution.

Question. Is there a sequence $s \subseteq F$ of k valid flights: $s = \langle f_{i_1}, ..., f_{i_k}\rangle$, referred to as a *trip*, where the flights in s satisfy the following *trip properties*:

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

Note that s may contain one or more flights to and from airports that are not destinations. Such airports are called *connecting airports.* Flight that either departs from or arrives at connecting airport is called *connecting flight.*

The *optimisation* version of TP (TPO) asks for an *optimal* solution $opt(s)$ of s with respect to a given objective function. The objective function may minimize or maximise on connecting airports, cost, travel duration or the number of flights in s.

The *decision* version of TP (TPD) asks whether there exists a valid sequence of flights s, such that some properties of s have values less than or equal to a given integer B. TPD is shown to be is NP-complete (Appendix A), from which it follows that TPO is NP-hard.

This problem formulation has only presented the constraints which every trip must satisfy. In reality, travellers may have additional preferences with regards to their travel. These are discussed in the next two sections.

Hard Constraints

Additional constraints might be imposed on an instance of TP. A solution that does not satisfy all required constraints is considered invalid. Examples of hard constraints include:

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.

Soft Constraints

It may be desirable to search for solutions that satisfy some requirements, but still accept trips that violate them. Such requirements are called *soft constraints* and for TP they can be some of the following:

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 connecting 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 there may exist an instance for which all soft constraints can not be satisfied simultaneously. In such case, the traveller may be required to rank their requirements in an order of preference. The instance becomes a lexicographic optimisation problem, where the highest ranked objective is optimised first, then the next highest ranked objective and so on. If all requirements are equally important for the traveller, the instance becomes a multiobjective optimisation problem, where the objectives are all constraints required by the traveller. Each of the objectives is assigned a weight of influence on the final solution set. The problem is then to optimize the objective function, composed by the constraints, each of them multiplied by its weight. Multiobjective optimisation is further investigated in this work.

Most of the constraints presented in this section can be viewed as either hard or soft, depending on the user requirements. Therefore this work assumes as an additional non-functional requirement that any constructed TP models are flexible and easy to extend by adding, removing and modifying constraints and objective functions. The choice of techniques for solving TP, discussed later in this work, is highly influenced by this requirement.

2. EXISTING WORK

This section is a discussion of two widely used methods for solving routing problems: Integer Programming (IP) and Constraint Programming (CP) and the principles of Branch and Bound methods, typically employed as IP and CP search algorithms. The background of the real-world data used for generating TP instances is also discussed.

2.1 Integer Programming

Integer Programming (IP) is a way to solve optimisation problems by modelling their requirements as linear inequalities and equations, called *constraints*, where variables are restricted to take only integer values. Optimal solutions are obtained by optimising an *objective function* subject to the constraints. This task is done by an IP *solver*, which incorporates implementations of some specialised search algorithms. Solving a problem with IP requires building a suitable model and passing it to the IP solver. There is a range of solvers one can use.

The next section presents an example IP model of the Travelling Salesman Problem (TSP), defined in Section 1.2, constructed by Dantzig et al. [14].

Dantizg, Fulkerson and Johnson

Let $x(i, j)$ be a variable that denotes whether city j immediately succeeds city i in a TSP tour. The value of $x(i, j)$ is equal to 1 if this is true, or 0 otherwise. If $x(i, j)$ is 1, then i is called an *outgoing* city and j - an *incoming* city. The objective function of TSP is to minimise the total length of the tour, which Dantzig et al. [14] express as the following function:

$$\min \sum_{i\in A}\sum_{j\in A} d(i,j)x(i,j), \tag{1}$$

where A is the set of all cities and $d(i, j)$ is the distance between cities i and j.

In each TSP tour every city has to be visited once, i.e each city has to be incoming and outgoing exactly once. This can be enforced with the following constraint:

$$\begin{aligned} \sum_{j\in A} x(i,j) &= 1, \qquad i \in A, \\ \sum_{i\in A} x(i,j) &= 1, \qquad j \in A, \end{aligned} \tag{2}$$

Constraint (2) 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*, where some cities are visited more than once. For example, consider Figure 1 and Figure 2. Both of them satisfy constraint (2), but Figure 2 does not represent a valid tour, because it contains two subtours, each of size 3. To tackle this problem, [14] introduce the *subtour elimination* constraint:

$$\begin{aligned} \sum\{x(i,j) : (i,j) \in (S \times \bar{S}) \cup (\bar{S} \times S)\} \geq 2, \\ \forall \emptyset \subset S \subset A, \text{ where } \bar{S} = A \setminus S \end{aligned} \tag{3}$$

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 (3) enforces that at least two cities in S are connected with cities from $\bar{S}$. Figure 1 and Figure 2 serve as examples. Let $S = \{A_1, A_2, A_3\}$, then $\bar{S} = \{A_4, A_5, A_6\}$. Constraint (3), would detect the tour on Figure 2 as invalid, as there is no city in S that is connected to a city in $\bar{S}$. Figure 1 will be accepted, since A_1 and A_3 are connected with A_6 and A_4 respectively.

Solving TSP is done by minimising the objective function (1), subject to constraints (2) and (3).

Dantzig et al. [14] solve this TSP IP model using a novel methods for that time, such as the *cutting-plane* method and the *assignment problem relaxation*, where the latter is discussed in the next section. We point the interested reader to [14] and [15], where the cutting-plane method is explained in more detail using a 10 city TSP instance as an example.

Other example solutions of routing problems with IP include [2, 8]. Achuthan and Caccetta [2] give an IP formulation for the VRP. Baker [8] presents a model for the Time-Constrained TSP (TCTSP), defined in Section 1.2. 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 [1].

2.2 Constraint Programming

Constraint Programming (CP) is an alternative method for solving optimisation problems. Similarly to IP, CP first requires building a model of the problem. Modelling a problem instance π yields a constraint satisfaction problem (CSP) $CSP(\pi)$, which consists of a set of decision variables V, a set of rules, called *constraints* concerning the assignment of values to variables and an *objective.* Unlike IP, CP constraints need not be linear integer inequalities. Moreover, CP supports specialised constraints like "if-then" and "all-different". Each variable in V has a set of possible values, called *domain.* A *solution* to π is an assignment of each variable in V to a value in its domain, such that all constraints are satisfied and the objective is optimised. The program that searches for a solution $CSP(\pi)$ is called a *solver.* Similarly to IP, implementing a CP solver is not needed.

The remaining of this section discusses existing formulations of some routing problems as CSPs.

TSP with CP

We present an alternative approach to model TSP using CP. The technique is developed by Caseau and Laburthe [10] and it can handle small to medium size problem instances with size equal to 40 cities on average. Caseau and Laburthe [10] motivate the choice of this size by arguing that it is both frequently encountered in practice and not extensively studied.

The $CSP(TSP)$ constructed by Caseau and Laburthe [10] is similar to the TSP IP model discussed in the previous section. Let the set of cities A be duplicated into two sets A_1 and A_2. Caseau and Laburthe [10] represent each city $i \in A_1$ as a variable with domain equal to all cities in A_2 that are not equal to i. The assignment $i = j$, $i \in A_1$, $j \in A_2$ implies that the next visited city after i is j. The assignment of values in A_2 to variables in A_1 is equivalent to a bipartite perfect matching from A_1 to A_2 [10], which has the same effect as constraint (2) from the TSP IP model.

Subtour elimination is imposed by restricting the domains of certain variables upon each assignment [10]. Whenever variable i is assigned the value j, all paths adjacent to cities i and j are checked. Let x be a path that ends in i and y be a path that starts from j. If $|x| + |y| < n - 2$, then the variable at the end of y can not be assigned the city at the start of x as value, because that would lead to a cycle of size less than n.

Caseau and Laburthe [10] also introduce a *lookahead* technique that predicts when certain partial variable assignments will inevitably lead to an invalid tour. The technique detects assignments that are necessary to enforce the cyclic

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

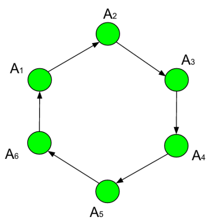


Figure 1: A valid TSP tour

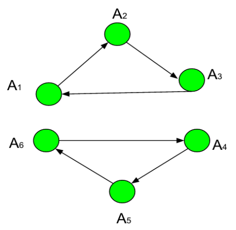


Figure 2: Two subtours of size 3

property of a TSP tour and marks as invalid all partial assignments that do not satisfy these properties.

Caseau and Laburthe [10] prune suboptimal partial solutions by imposing a lower bound on the tour distance, which is the sum of the distances between each city i and the city nearest to i. The authors also use this bound for pruning values from the domain of some variables. For instance, when $d(i, j)$ has a value that is much higher than the best assignment for i and j, the procedure predicts that the assignment $i = j$ would inevitably fail.

Caseau and Laburthe [10] note that CP is a "truly flexible approach", where flexibility is defined as the ability to easily add or remove extra constraints to the initially constructed model. For instance, one can represent TSP with time windows by only adding few more constraints to the TSP CSP constructed above. This flexibility of CP has been noted before [30, 35].

Other CP algorithms for routing problems include GENIUS-CP [35], Backer et al. [7] and Rousseau et al. [39]. GENIUS-CP [35] is a combination of novel vehicle routing heuristics and a CP model for VRP. The model can be easily adapted to solve problems from many different contexts by exploiting the flexibility of CP. Backer et al. [7] also construct a CP model for VRP and use iterative improvement techniques and heuristics to speed up the search. Rousseau et al. [39] use an approach that combines CP with the column generation method introduced by Dantzig and Wolfe [13]. Their method targets VRPs with time window constraints.

2.3 Branch and Bound methods

The term "branch and bound" was first coined by Little et al. [33] and it refers to a widely-used method for solving optimisation problems. Branch and bound algorithms are one of the most successful methods for solving optimisation problems [12] and they are typically implemented in Constraints Programming (CP) and Integer Programming (IP) solvers [6, 32, 38].

A branch and bound algorithm iteratively divides the current set of problems into successively smaller subsets, called *subproblems*, calculates bounds on their values and discards subproblems that would lead to suboptimal solutions. After the termination of the algorithm, the best solution found so far is considered an optimum.

Branch and bound algorithm iterates over the entire solution space. However unlike exhaustive search, the use of bounds on the value of optimal solutions enables the algorithm to provably ignore parts of the search space.

The process of dividing each solutions set into subproblems is called *branching*. The choice of bounding and branching procedures is shown to be crucial to the size of the explored search space [33, 14, 28]. Existing literature has spent substantial effort in constructing efficient algorithms for branching and bounding. Some of these algorithms are discussed in this section.

Branching Strategies

Branching algorithms can be based on either *variable* or *node* selection. Variable selection strategies are rules about how to divide the current set of solutions into subproblems, so that they are easier to solve than the original problem. Node selection methods choose which set of solutions should be branched next.

Common branching strategies estimate the impact of the variables on the objective functions or on the constraints. For example the next variable to branch on can be the one that imposes highest or lowest number of restrictions. We point the interested reader to the works of Atamtürk and Savelsbergh [6] and Achterberg et al. [1], which discuss most common branching strategies and evaluate their influence on the performance of the branch and bound algorithm.

Calculating Bounds

Relaxations are commonly used bounding strategies. A *relaxation* of a problem π is the process of obtaining another problem π' that is believed to be easier to solve by removing some constraints from π, where a feasible solution in π maps to a feasible solution in π', but the converse need not be true in general. Solutions to π' give information about the lower or the upper bound of the values of π.

Some examples of relaxations include the Assignment Problem (AP) relaxation, the Lagrangian relaxation and the Set Partitioning Problem (SPP) relaxation. The AP relaxation consists of removing the subtour elimination constraint from the TSP IP model defined in Section 2.1 and minimising the objective function (1) only subject to constraint (2). The solution of the relaxed problem is used as a lower bound of the original problem. The AP relaxation is extensively used in the existing literature [21, 29, 19, 31].

The set partitioning problem (SPP) relaxation, discussed in the remainder of this section, is commonly used for VRP [16, 3, 17, 5]. It is performed 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, VRP 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:

$$\text{minimise} \sum_{r \in R} c_r x_r, \tag{4}$$

$$\sum_{r \in R} \delta_{i,r} x_r = 1, \quad \forall i \in A \setminus \{D\}, \tag{5}$$

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 [16, 3].

2.4 Travel Booking Systems

A *metasearch engine* queries multiple search engines and returns a ranked list of the aggregated results. A class of online booking systems work on top of a metasearch engines in order to help users find their desired itineraries. Given a query composed of home airport, destination, inbound and outbound dates, supplied by the traveller once, the travel metasearch engine returns a ranked list of all relevant flights.

Skyscanner. Skyscanner [40] is a travel metasearch engine, available in over 30 languages and used daily by over 7.6 million people [41]. Whenever the user clicks on one of the query results on the Skyscanner website [40] in order to complete their booking, Skyscanner saves the information about the booking to a database. The data from all bookings is referred to as *redirects data*.

Every booking b_k in the redirects data contains a list of flights H_k and their total price c_k, expressed as number of units of a some currency ϵ_k. Each flight f_{k_j} in H_k has departure date t_{k_j}, arrival date a_{k_j}, departure airport $A^d_{k_j}$ and arrival airport $A^a_{k_j}$. Redirects data for half a year for the time period between 1st of July 2016 and 1st of January 2017 was given by Skyscanner for the empirical study and testing of the constructed CP and IP TP models. The data is of size 38 GB and it contains 321,465,401 unique bookings. The process of obtaining TP instances from the redirects data is presented in Section 4.

3. SOLUTIONS TO TP

Two solutions to TP using IP and CP have been designed and implemented. This section presents the models, the choice of IP and CP solvers and the flights and airports representation.

3.1 Data Structures

The Java built-in implementation of a hash table, called *HashMap* [18], is used for representing all flights and airports as follows.

Each flight f_j is assigned an integer k_j, for $0 \leq k_j \leq m$ as a unique id and each airport A_i has a unique code $c(A_i)$.

All flights are kept in a hash table H_F, where for each flight f_j, k_j is a key and A^d_j, A^a_j, t_j, Δ_j and c_j are the values associated with this key.

All airports are kept in three hash tables H_A H_d and H_a. For each $A_i \in A$, H_A maps $c(A_i)$ to C_{A_i}. Hash tables H_d and H_a record the ids of the flights that depart from and arrive at each airport respectively, that is, for every $f_j \in F$, k_j is recorded as a value associated with key A^d_j in H_d, and again as a value associated with key A^a_j in H_a.

This representation is used for both CP and IP models. Its choice is motivated by the fact that both models require frequent access to all departing or arriving flights for a given airport, as will be seen in the next two sections. This usage of HashMaps promises $\mathcal{O}(1)$ complexity for getting and putting elements for each hash table, provided that risks of rehashing are avoided [18]. Since the number of flights and airports is specified on the first line of each TP instance, their sizes are known when the hash table is created and no rehashing occurs.

3.2 The CP model

The CP model for TP is constructed as follows. Let $m = |F|$. The decision variables are an array $\mathcal{S}$ of size $m+1$ (indexed $0, 1, ..., m$) that represents the TP tour and a variable z with domain $dom_z = \{1, ..., m\}$ that denotes 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 $(1 \leq j \leq m)$. The end of the tour is marked by setting $\mathcal{S}[z] = 0$. 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 $dom_v = \{0, ..., m\}$.

Constraints. The constraints added to the model enforce the five trip properties and make $\mathcal{S}$ form a valid sequence of flights.

The following constraint restricts that once the end of the trip is reached at some position z, no flights are further added to $\mathcal{S}$.

$$\mathcal{S}[i] > 0 \land \mathcal{S}[i+1] = 0, \ (0 \leq i \leq z-1) \tag{6}$$

The all-different constraint below enforces that every flight is taken only once [43].

$$\text{allDiff}(\mathcal{S}[0], ..., \mathcal{S}[z-1]) \tag{7}$$

The trip properties are enforced by the following five constraints, where constraint 1 corresponds to trip property (1), constraint 2 to trip property (2), etc:

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

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

   $\forall i (1 \leq i < z)$
3. $t_p + \Delta_p + C_r \leq t_q, \, p = \mathcal{S}[i], q = \mathcal{S}[i+1], r = A^a_q$,

   $\forall i (1 \leq i < z)$
4. $t_q + \Delta_q \leq T, \quad$ where $q = \mathcal{S}[z-1]$
5. $\forall A_k \in D, \, |\{i : (0 \leq i < z) \land A^a_{\mathcal{S}[i]} = A_k\}| > 0$

Constraint 1 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 airport. For constraint 2, the domain of each variable in $\mathcal{S}$ is set to include only flights that depart from the arrival airport of the previous flight. Constraint 5 restricts that the number of the flights that arrive at every destination in the trip is positive.

Objective Functions. The four objective functions are modelled as follows.

The cost objective function 8 optimises the sum of the costs of all flights in $\mathcal{S}$.

$$\text{optimise} \sum_{i=0}^{m-1} c_{\mathcal{S}[i]} \tag{8}$$

The trip duration objective function 9 optimises the sum of the departure date and duration of the last flight in $\mathcal{S}$.

$$\text{optimise } t_j + \Delta_j, \quad f_j = \mathcal{S}[z-1] \tag{9}$$

The number of connecting flights objective function 10 optimises the cardinality of the intersection between $\mathcal{S}$ and the set F' of all flights that arrive to connecting airport.

$$|\mathcal{S} \cap F'|, \quad F' = \{j : f_j \in F \wedge A^a_j \notin D\} \tag{10}$$

The number of flights is optimised by restricting the value of z to be either the minimum or the maximum possible.

TP can then be formulated as the problem of either maximising or minimising one or more of the objective functions, subject to all of the presented constraints.

3.3 The IP model

The model described in this section is a modification of the TP CP model with constraints in the form of integer linear inequalities.

Let $m = |F|$. We introduce a variable $x_{i,j}$ for every $i \in \{0, ..., m-1\}$ and $j \in \{0, ..., m-1\}$, such that $x_{i,j} = 1$ if $(i+1)^{th}$ flight is f_{j+1} 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, a set of variables $x_{m,j}$ is introduced, for $(0 \leq j \leq m)$, where flight f_m 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 $c_j = 0$. Connecting time is not added, when connecting from A_0 to A_0 as part of these flights. The special flight is used for enforcing the valid ending of the trip at the home airport without the need of an additional variable, as it is the case with z in the CP model, by setting $x_{m,m} = 1$. The special flight can be scheduled more than once, provided that if $x_{i,m} = 1$ for $(0 < i < m)$, then for every i' $(i < i' \leq m)$, $x_{i',m} = 1$ and the first occurrence of f_m marks the end of the trip. Flight f_m is added to F.

The variable $x_{i,j}$ is implemented as a two dimensional matrix of size $m' \times m'$.

Constraints. Constraints (11) and (12) are equivalent to the all-different constraint used in the CP model. Constraint (11) imposes that exactly one flight is scheduled at each step by adding a requirement that each row i in the matrix takes the value 1 once. Constraint (12) ensures that each flight is taken at most once. Constraints (13) and (14) accept as valid flight schedules that do not contain all flights in F.

$$\forall i\,(0 \leq i < m), \quad \sum_{j=0}^{m} x_{i,j} = 1 \tag{11}$$

$$\forall j\,(0 \leq j < m), \quad \sum_{i=0}^{m-1} x_{i,j} \leq 1 \tag{12}$$

$$\sum_{i=0}^{i=m} x_{i,m} \geq 1 \tag{13}$$

$$x_{m,m} = 1 \tag{14}$$

Constraint (15) enforces that once the trip is ended, no more flights can be scheduled. Assume that flight f_{z-1} is the final flight returning to A_0, where $1 \leq z \leq m$. Then, constraint (15) adds $m - z + 1$ many f_m flights and the variables $x_{z,m}, ..., x_{m,m}$ are all equal to 1.

$$\forall i\,(1 \leq i < m), \quad x_{i-1,m} \leq x_{i,m} \tag{15}$$

Each of the trip properties is expressed as an integer linear inequality as follows.

Constraints (16) and (14) enforce trip property (1). From trip property (2) it follows that there must exist some $z < m$, $x_{z,j} = 1$, for which $j \neq m$ and $A^a_j = A_0$. Thus, $A^a_j = A_0$ is indirectly enforced.

$$\sum_{j \in S_1} x_{0,j} = 1, \tag{16}$$

where $S_1 = \{j \in \{0, ..., m-1\} : A^d_j = A_0\}$

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. Every flight is restricted to depart always from the arrival airport of the previous taken flight. This is equivalent to the following linear integer inequality:

$\forall i\,(1 \leq i \leq m) \wedge \forall y \in A :$

$$\sum_{j \in S_1} x_{i-1,j} = \sum_{j' \in S_2} x_{i,j'}, \tag{17}$$

where:

$S_1 = \{j \in \{0, ..., m\} : A^a_j = y\}$,

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

Constraint (18) enforces trip properties (3) and (4): every flight f_j cannot depart until the previous flight $f_{j'}$ has arrived, adding connection time, if needed.

$\forall i\,(0 \leq i < m), \forall j\,(0 \leq j \leq m) :$

$$x_{i+1,j} + \sum_{j' \in F'} x_{i,j'} \leq 1, \tag{18}$$

where:

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

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

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

$$\forall A_k \in D, \sum_{i=0}^{m-1} \sum_{j \in F_k} x_{i,j} \geq 1, \tag{19}$$

where $F_k = \{j : f_j \in F \wedge A^a_j = A_k\}$

Objective functions. The four objectives are modelled as follows.

Objective function (20) optimises the sum of the costs of all taken flights:

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

Objective function (21) optimises the number of flights taken during the trip: scheduling more "special" flights results in shorter trips and vice versa.

$$\sum_{i=0}^{i=m} x_{i,m} \tag{21}$$

The trip duration objective function is implemented as follows. A set of variables $y_{i,j}$ is introduced for every $i \in \{0, ..., m-1\}$ and $j \in \{0, ..., m-1\}$, such that if $y_{i,j} = 1$, then the last flight is f_{j+1} and it is taken at the $(i+1)^{th}$ step of the trip (that is, $x_{i,j} = 1$), using the observation that $x_{i,j} \times x_{i+1,m} = 1$ if and only if f_{j+1} is the last flight in the trip, taken at position i. Each $y_{i,j}$ is equal to:

$$\begin{aligned} &\forall i (0 \leq i < m) \, \forall j (0 \leq j < m), \\ &\quad y_{i,j} = x_{i,j} \times x_{i+1,m} \end{aligned} \tag{22}$$

This multiplication is equivalent to the following set of IP constraints:

$$\begin{aligned} &\forall i (0 \leq i < m), \, \forall j (0 \leq j < m): \\ &\quad y_{i,j} \geq 0, \\ &\quad y_{i,j} \leq x_{i,j}, \\ &\quad y_{i,j} \leq x_{i+1,m}, \\ &\quad y_{i,j} \geq x_{i,j} + x_{i+1,m} - 1 \end{aligned} \tag{23}$$

Then, the trip duration objective function is:

$$\sum_{i=0}^{m-1} \sum_{j=0}^{m-1} (t_j + \Delta_j) \times y_{i,j} \tag{24}$$

The number of connecting flights can be optimised using the following objective function:

$$\sum_{i=0}^{m-1} \sum_{j \in F'} x_{i,j}, \quad F' = \{j : f_j \in F \wedge A^a_j \notin D\} \tag{25}$$

TP can then be formulated as the problem of either maximising or minimising one or more of the objective functions, subject to all of the presented constraints.

3.4 IP and CP 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 [37] 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 already implemented search algorithms. The TP CP model is implemented using Choco.

Gurobi is a mixed-integer programming solver first released in 2009. It is widely used mathematical optimisation tool [25] and it is licensed freely to academia. Gurobi is used for the implementation of the TP IP model.

The CP and IP models presented in Section 2.2 and Section 2.1 respectively are given to Choco and Gurobi, which find solutions. Both solvers use the default search strategies and heuristics. The CP model uses a branching heuristic that chooses the variable with current minimum domain size first (first-fail heuristic [26]), where the sequence in which the values in its domain are tried (value ordering) follows a non-decreasing order. The IP solver uses the same value ordering strategy and a default branching heuristic that "...makes an automatic choice, depending on problem characteristics...", where the rules of the automatic choice are not supplied.

3.5 Hard Constraints

Hard constraints 1 and 2 (HC1, HC2), formulated in Section 1.2.3 are implemented in the CP model. This section presents the implementation of HC2, noting that HC1 is incorporated to the CP model using similar techniques.

Let $P = \{p_1, ..., p_h\}$ denote the list of every HC2 for a given TP instance π, where each HC2 is represented as a pair $p_r = \langle A_{k_r}, t_{i_r} \rangle \in P$ for $0 \leq r \leq h \leq |D|$, where the traveller must be at destination $A_{k_r} \in D$ at date $t_{i_r} < T$.

We introduce an array $\mathcal{D}$ of size h. The domain of each variable in $\mathcal{D}$ is $\{1, ..., m\}$. If $\mathcal{D}[r] = i$, then the flight at position $\mathcal{S}[i]$ departs from A_{k_r} at least one day after date t_i [2] and the flight $\mathcal{S}[i-1]$ arrives at A_k before date t_{i_r}, satisfying p_r. This is achieved by the following constraints:

$$\text{allDiff}(\mathcal{D}[1], ..., \mathcal{D}[h]) \tag{26}$$

$$\begin{aligned} &\forall \, p_r \in P, \, \exists \, i \, (0 < i < m): \\ &\mathcal{D}[r] = i \wedge \mathit{dom}_{\mathcal{S}[i-1]} = S_1 \wedge \mathit{dom}_{\mathcal{S}[i]} = S_2, \end{aligned} \tag{27}$$

where:

$$S_1 = \{j : f_j \in F \wedge A^a_j = A_{k_r} \wedge t_j + \Delta_j < t_{i_r}\},$$

$$S_2 = \{j' : f_{j'} \in F \wedge A^d{}_{j'} = A_{k_r} \wedge t_{j'} > t_{i_r} + 1\}$$

3.6 Multiobjective Optimisation

Both models can optimise subject to more than one of the four objectives simultaneously and implement two different techniques for multiobjective optimisation: scalarisation and determination of the Pareto frontier. Their implementations are discussed in this section.

Let $O = \{o_1, ..., o_k\}$ be the set of all objectives for minimisation, subject to which some TP instance π has to be solved. Let $\mathcal{F}(\pi)$ be the set of feasible solutions to π, where $g_r(S)$ is the value of objective o_r in feasible solution $S \in \mathcal{F}(\pi)$ and $S(\pi, o_r)^*$ denotes the minimum measure of objective o_r across all feasible solutions. Let w_r denote the weight of importance of objective function o_r. For both models, $w_r = \frac{1}{k}$ for $1 \leq r \leq k$, i.e. each objective function has the same weight.

The IP model implements a multiobjective optimisation technique called *scalarisation*, which is performed as follows. First, π is solved subject to each individual objective in O. The multiobjective optimisation for an instance π is then formulated as the following single-objective function:

[2] It is assumed that the traveller wants to stay at least for a day after the required date. Ideally, the solver should allow for taking this as an input parameter.

$$\underset{S \in \mathcal{F}(\pi)}{\text{optimise}} \sum_{r=1}^{k} \frac{w_r \times g_r(S)}{S(\pi, o_r)^*}, \tag{28}$$

where *optimise* refers to the minimum or the maximum value of the function.

The CP model multiobjective optimisation consists of finding the set of *Pareto optimal* solutions, where a solution is Pareto optimal if none of the objective functions can be improved in value without making some of the other objectives have worse values. Given a CP model and a set of objective functions, the Choco solver has already implemented functionality to return the set of Pareto optimal solutions for the modelled instance subject to the objectives.

4. DATASETS FOR TP

The process of obtaining TP instances from the redirects data, introduced in Section 2.4, consists of extracting all relevant information about airports and flights from each booking and passing the data to a program, called *TP instance generator*. The program takes the processed data, the required instance size (m, n, d and T) and the number of required instances as an input and returns a set of TP instances that satisfy all these properties. This section presents the utilisation of the redirects data in more detail.

Extracting airports data. Let $\mathcal{A}$ be the set of airports that will be used by the TP instance generator, where $\mathcal{A}$ is chosen to consist of the airport codes and connecting times of the 100 busiest airports in Europe [44]. The connecting time C_i of each airport $A_i \in \mathcal{A}$ is expressed as a fraction of a day and it is extracted from the redirects data as follows. Initially, for each A_i, C_i is set to infinity. For every sequence of flights H_k, where H_k follows a non-decreasing order by departure date, if $|H_k| > 1$, for every two consecutive flights f_j and f_{j+1} in H_k with $A_j^a = A_{j+1}^d = A_i$, where $A_i \in \mathcal{A}$:

$$C_i = \min\{C_i, t_{j+1} - a_j\},$$

where t_{j+1} is the departure time of f_{j+1} and a_j is the arrival time of f_j.

Extracting flights data. Let $w = (l, u)$ be the time window of the flights that will be used by the TP instance generator, where l is the earliest allowed flight departure date and u is the latest allowed flight arrival date, where $l =$ 1st of November 2016, 00:00 UTC and $u =$ 31st of December 2016, 23:59 UTC. This choice of values for l and u is motivated by the assumption that during these months there is an increased number of flight bookings, because of the holidays. It is desirable for the flights data to be composed by consecutive months, so that the models can be evaluated later with instances with $T > 31$.

The list of flights that will be used for generating TP instances is obtained after executing the following steps:

1. For every H_k in the redirects data:
   - if $|H_k| > 1$, discard H_k, since no information on the price of each separate flight is available. Discard H_k and repeat step 1.
   - if $|H_k| = 1$, let $f_k \in H_k$. If $A_k^a \in \mathcal{A}$ and $A_k^d \in \mathcal{A}$ and $l \leq t_k \leq a_k \leq u$, go to step 2. Otherwise, discard H_k and repeat step 1.
2. Convert the currency of f_k to GBP, using currency exchange rates from 4th of February, assuming that the currency rates are not volatile. Go to step 3.
3. Add the duration Δ_k of f_k as a fraction of a day, where $\Delta_k = a_k - t_k$ and remove a_k from the properties of f_k. Go to step 1.
4. Sort all extracted flights in non-decreasing order by departure date, breaking ties on flight duration, departure and arrival airport codes and price. This yields the list of flights $\mathcal{F}$, which may contain duplicate flights with same or different price[3]. Go to step 5.
5. Remove all duplicate flights from $\mathcal{F}$ as follows. For every sequence of duplicate flights, preserve the flight with median price. This is easy to do, since the duplicate flights are a contiguous subsequence[4] of $\mathcal{F}$, ordered by non-decreasing price. Return $\mathcal{F}$ and terminate the program.

Step 1 is executed for every booking in the redirects data. Steps 2, 3 are computed only for bookings that contain one flight. Step 4 is reached after all bookings in the redirects data have been visited and it returns $\mathcal{F}$ of size equal to 1.7 Gigabytes that contains 12,580,487 flights. Step 5 reduces the size of $\mathcal{F}$ to 5.6 Megabytes, which is 147,172 unique flights, where 67,657 flights depart in November and 79,515 flights arrive by the end of December.

TP instance generator. The TP Instance Generator is a program that takes as an input $\mathcal{F}$, $\mathcal{A}$ and a configuration file f_c, and outputs two files f_1 and f_2, where f_1 represents a valid TP instance and f_2 can be used to regenerate f_1. The file f_c is a list of values for m, n, d and T. The files are generated by executing the following steps for every setting of m, n, d and T in f_c:

1. Using $\mathcal{A}$, randomly pick a set of n airports $A \subseteq \mathcal{A}$, a set of d destinations $D \subset A$ and home airport $A_0 \in (A \setminus D)$. Remember the positions of the airports in $\mathcal{A}$ and their purpose, so that they can be written to f_2 later. Go to step 2.
2. Randomly decide on earliest departure flight date T_0, where $l \leq T_0 \leq u - T$. Go to step 3.
3. Extract all flights in $\mathcal{F}$ that depart at time later than or equal to T_0 and arrive at time no later than $T_0 + T$ from and to airports only in A. The obtained set of flights is denoted as F'. Go to step 4.
4. If $|F'| < m$, go to step 1. If $|F'| \geq m$, remember the positions of the flights in $\mathcal{F}$ and their purpose, so that they can be written to the properties file later. Go to step 5.
5. Randomly pick m flights from F' to obtain the set F. Reformat the departure date of each $f_j \in F$ to be equal to $t_j - T_0$, that is a positive rational number that represents a fraction of a day. Go to step 6
6. Compose a directed graph $G = (V, E)$ with $V = A$ and $E = \{(i_1, i_2) | \exists f_j \in F, A_j^d = i_1 \wedge A_j^a = i_2\}$. Check whether all destinations in D and A_0 are part of one

[3]Two flights are said to be duplicates if they have the same departure and arrival dates and departure and arrival airports. They do not need to have the same price.

[4]A contiguous subsequence of a list S is a subsequence made up of consecutive elements of S.

Setting	m	n	T
1	$10 \times d$	$2 \times d$	$(d+1) \times 3$
2	$14 \times d$	$2 \times d$	$(d+1) \times 3$
3	$20 \times d$	$2 \times d$	$(d+1) \times 3$
4	$25 \times d$	$2 \times d$	$(d+1) \times 3$

Table 1: The values of m, n and T, depending on d, used for generating TP instances.

strongly connected component (SCC)[5] in G. If that is true, go to step 7. Otherwise, go to step 1.

7. The sets A and F form a valid TP instance, which is saved to F_1. The last positions of picked airports and flights in $\mathcal{A}$ and $\mathcal{F}$, remembered at steps 1 and 4, are saved to f_2. The program terminates.

Step 6 acts as filtering of unsolvable instances. Tarjan's algorithm [42] is used for finding all SCCs in G. Note that an instance that passes the SCC test still may have no solutions.

5. EMPIRICAL STUDY

This section presents an empirical study of the CP and IP TP models. All experiments presented in this section were carried out on a MacBook Pro laptop with 2.9GHz dual-core Intel Core i5 processor, 16 GB 1867 MHz DDR3 memory and OS X El Capitan operating system. A time limit of 900 seconds (15 minutes) was imposed on both solvers for every trial, where for Section 5.4 the incumbent solutions are recorded.

In this section, the objectives for minimising cost, connecting airports, trip duration and number of flights are also referred to as o_1, o_2, o_3 and o_4 respectively and $O = \{o_1, o_2, o_3, o_4\}$ is the set of all objectives. The problem of finding optimal solutions for a given TP instance π subject to a given objective function $o_r \in O$ is called a *trial* and it is denoted as $tr(\pi, o_r)$.

5.1 Correctness of the models

The correctness of the two TP models is verified using a test dataset, generated as outlined in Section 4. The test dataset contains 40 unique TP instances with number of flights m, number of airports n, and holiday time T calculated as functions of the number of destinations d according to settings 1 and 2 in Table 1. For each $d \in \{1, 2, 3, 4\}$ and for each of the two settings, five TP instances were generated. The smallest instance has size $m = 10$, $n = 2$, $d = 1$, $T = 6$, that is $(10, 2, 1, 6)$, and the largest instance has size $(56, 8, 4, 15)$.

Each of the solvers was run 4 times on each of the test instances, optimising on one of each objectives, asking for all optimal solutions without enforcing time limits. That makes 160 calls to each solver. Each set of solutions is saved to a file and the returned flight schedules from the solvers for each trial were compared. The two solvers returned the same answers for each trial. With this result, we *assume* that the models and their implementations are correct.

[5]A strongly connected component (SCC) of a directed graph G is a subgraph G' with the property that every vertex in G' is reachable from every other vertex in G', and no additional edges or vertices from G can be included in G' without breaking this property.

5.2 Comparison of the models

This section compares the performance of the two implemented TP models.

Experiment setup. Two datasets called $\mathcal{D}_3$ and $\mathcal{D}_4$ were generated using the TP instance generator described in Section 4 as follows. For each value of d ($1 \leq d \leq 15$), 20 TP instances were generated for each dataset, having m, n and T calculated as functions of d according settings 3 and 4 for $\mathcal{D}_3$ and $\mathcal{D}_4$ respectively, as shown in Table 1. For example, in $\mathcal{D}_3$ the smallest instance has $m = 20$, $n = 2$, $d = 1$, $T = 6$, that is $(20, 2, 1, 6)$, the largest instance has size $(300, 30, 15, 48)$, the smallest instance in $\mathcal{D}_4$ has size $(25, 2, 1, 6)$ and the largest instance has size $(375, 30, 15, 48)$. This makes 300 TP instances for each dataset.

For each dataset, each instance was solved subject to each objective function in O with both models. Whenever the time limit of 900 seconds was reached, the solver stopped execution, the trial is considered as not solved and no incumbent solutions are recorded.

The results are shown in Figure 3, Figure 4 and Figure 5. Each figure is discussed in the following paragraphs.

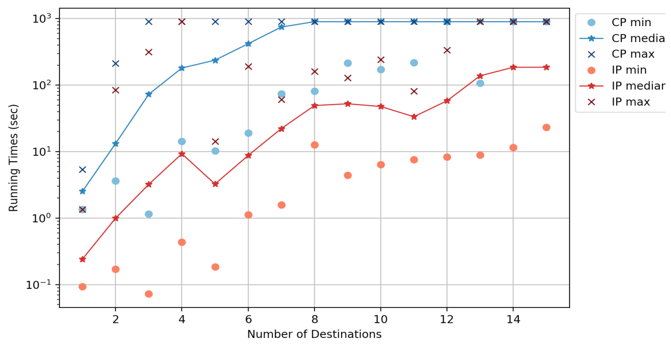
CP and IP. The minimum, median and maximum running time of all trials per model for every d is shown in Figure 3a for $\mathcal{D}_3$ and in Figure 3b for $\mathcal{D}_4$. For each dataset, both models show an increase in terms of the median running time with the increase of d, where this relationship is more evident in the CP model.

The results suggest that the IP model is better than the CP model for both $\mathcal{D}_3$ and $\mathcal{D}_4$. For example, in Figure 3a the minimum running time of CP for every $d \notin \{3, 13\}$ is worse than the median running time of IP. The IP model reached the time limit for $d = \{13, 14, 15\}$, whereas the CP model reached the time limit for every $d \notin \{1, 2\}$. For $\mathcal{D}_4$, the median running time of IP is faster than the minimum running time of CP for 9 settings of d. Figures 13 - 12 further show that for both datasets the CP model reached the time limit for a much higher proportion of trials than the IP model.

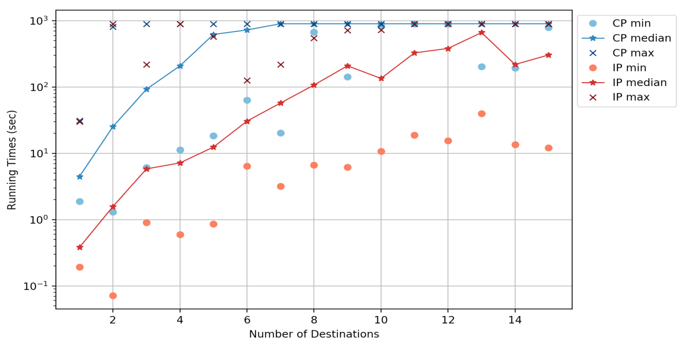
CP and IP objective functions. For each dataset, model, objective function and value of d we calculated the median running time over all trials. The results can be seen in Figure 4a for $\mathcal{D}_3$ and Figure 4b for $\mathcal{D}_4$, where the CP results are marked in blue and the IP results are in red and the median running time over trials with objective functions o_1, o_2, o_3 and o_4 are marked by a circle, cross, triangle and asterisk respectively. For example, the median running time of the CP minimum cost objective function for $d = 4$ is 200 seconds for trials in $\mathcal{D}_3$ and about 880 seconds for trials in $\mathcal{D}_4$.

The figures show that each objective function implemented in the IP model solves trials faster than any objective function that is part of the CP model. With the increase of d, the superiority of IP over CP becomes more visible.

Higher variation between the running time of each objective function in the CP model for instances with $d \leq 10$ for $\mathcal{D}_3$ and $d \leq 7$ for $\mathcal{D}_4$ can be observed, where o_1 and o_4 tend to perform worse than o_2 and o_3. The reason that for high values of d all objective functions are shown to perform equally well is that most trials reached the time out. The implementation of the objective functions in the IP model have similar running times for most values of d, where higher variation in terms of running time can be observed only in $\mathcal{D}_4$ for high values of d.

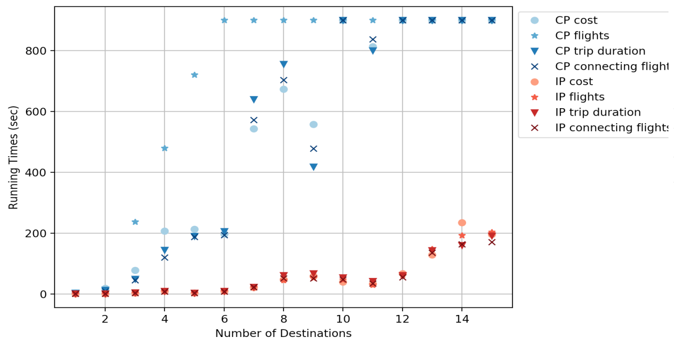


(a) $\mathcal{D}_3$

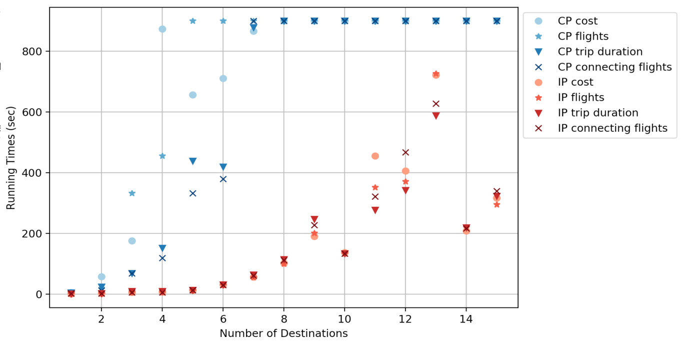


(b) $\mathcal{D}_4$

Figure 3: Minimum, median and maximum running time of CP (blue) and IP (red) models for each dataset.

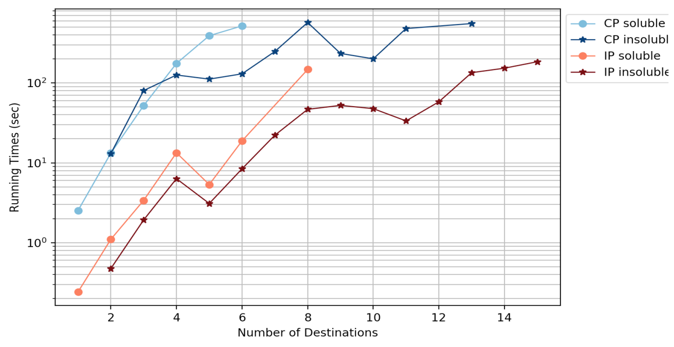


(a) $\mathcal{D}_3$

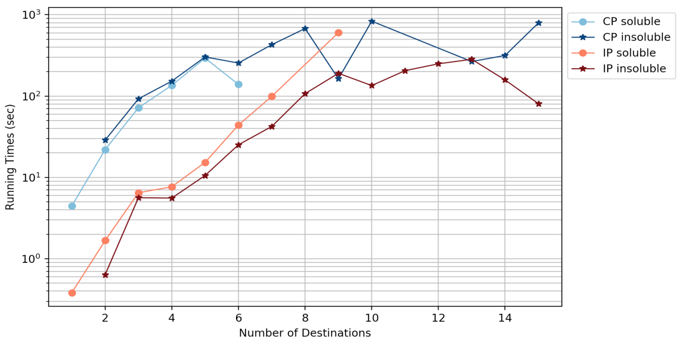


(b) $\mathcal{D}_4$

Figure 4: Median running time of each objective function for each dataset.



(a) $\mathcal{D}_3$



(b) $\mathcal{D}_4$

Figure 5: Median running time of IP and CP models on soluble and insoluble trials for each dataset.

CP and IP soluble and insoluble trials. We divided the results produced by each model into three groups: those that were found soluble, insoluble and not solved in order to compare the performance of the models for each group. The median running time of insoluble and soluble trials for every value of d was calculated and all not solved trials were discarded, since their running time is always 900 seconds. Figure 5 shows the results, where Figure 5a is for $\mathcal{D}_3$ and Figure 5b for $\mathcal{D}_4$. We note that the numbers of soluble, insoluble and not solved trials for each dataset differ across the two models. Their proportion for each value of d for each model and for each dataset are shown in Figure 13 and Figure 14.

Figure 5a shows that the IP model has faster median running time than the CP model for both soluble and insoluble trials. An increase of the median running time with the increase of d is observed for both groups of trials for each model, where this relationship is more evident in the CP model: for $d = 8$, the CP model solved 0 soluble trials and for $d = \{14, 15\}$, CP reached the time limit for all trials.

Figure 5b shows similar results. The IP model is faster than the CP model for every $d \notin \{9, 13\}$, where for $d = 13$ insoluble CP and IP trials have equal median running time and for $d = 9$, the CP model is slightly quicker.

For $d = 0$, both models encountered 0 insoluble trials in both datasets and for high values of d none of the models found soluble trials. Figure 13b and Figure 14b show the reason: increasing d decreases the proportion of soluble instances for both datasets. This suggests that the chosen relationship between d and m, n and T in $\mathcal{D}_3$ and $\mathcal{D}_4$ shown in Table 1 is not successful in creating soluble instances for high values of d.

The IP model finds proving insolubility easier than finding an optimal solution: for every value of d for both $\mathcal{D}_3$ and $\mathcal{D}_4$, insoluble trials were solved quicker than soluble trials (Figure 5). A possible reason for obtaining these results is that once a trial is found to be insoluble, the solver can terminate execution, whereas finding a solution to a trial does not immediately guarantee its optimality.

5.3 Multiobjective Optimisation

This section evaluates the performance of the CP and IP multiobjective optimisation implementations and uses notation previously introduced in Section 3.6.

Let $\omega(S, o_r)$ be the *score* of a solution $S \in \mathcal{F}(\pi)$ relative to objective $o_r \in O$, where $\omega(S, o_r)$ is calculated as follows:

$$\omega(S, o_r) = \frac{g_r(S) + 1}{S(\pi, o_r)^* + 1}, \tag{29}$$

where the addition of 1 to both the numerator and the denominator is done in order to avoid division by 0 or obtaining score equal to 0. Whenever $g_r(S) = S(\pi, o_r)^*$, then $\omega(S, o_r) = 1$ and this means that solution S is an optimum with respect to o_r. Note that $\omega(S, o_r) \geq 1$, since o_r is a minimisation objective. Therefore, solutions with high score have value far from the optimum relative to the measured objective. Calculating scores of solutions is used as a normalisation technique: the measure of performance over all objectives is adjusted to a common scale, so that comparison between them is possible.

Let $\mathcal{D}_5$ be the union of all TP instances in datasets $\mathcal{D}_3$ and $\mathcal{D}_4$, discussed in the previous section, that have known optimal solutions for each objective function by both models. For every $\pi \in \mathcal{D}_5$, we solved $tr(\pi, O)$ with both models with O as defined in the beginning of Section 5. The CP model returned the Pareto frontier of $tr(\pi, O)$, which in general comprises multiple solutions for every trial, whereas the IP model returned one solution for each trial. The weight w_r of each objective o_r is equal to $\frac{1}{4}$ (that is, the four objectives have equal weight). For each objective in O, the score of every solution returned by each model was calculated. The obtained results from each trial and objective, grouped by d, are shown in Figure 6 for the IP model and in Figure 7 for the CP model, where Figure 7 shows all members of the Pareto frontier for each trial.

The scores of all trials relative to o_1, o_3 and o_4 tend to decrease with the increase of d for both models (Figure 6 and Figure 7). Possible explanation of this result could be the following. It was observed in the previous section that with the increase of d the number of soluble trials decreases and the number of trials that reach the time limit increases. This suggests that as d increases, the number of solutions returned by each model is smaller and it is produced after solving only "easier" trials (that is, trials that did not reach the time limit), which causes having less variation and lower scores for higher values of d. However, this hypothesis does not explain the results for o_2 (Figures 6b and 7b).

The highest scores were obtained for the solutions returned by the CP model relative to o_1 (Figure 7a) and the lowest scores were obtained for the solutions returned by the IP model relative to o_2 (Figure 6b). For every objective, the scores of the solutions obtained from the IP model are lower than the scores of the solutions returned by the CP model. This suggests that Pareto optimality is much weaker than optimality with respect to the scalarisation objective.

5.4 Empirical study of the IP model

The experiments in this section investigate the influence of the number of flights m, airports n, destinations d and holiday time T on the running time of the IP model and the solubility of the TP instances.

Similarly to the experiments in the previous section, each instance was solved subject to each of the 4 implemented objective functions, that is 4 trials per instance. For each trial, a time limit of 15 minutes was imposed on the solver. Whenever the time limit is reached, the solver returns the incumbent solution, if found, and stops execution.

The results from each experiment are represented by three plots, where each left-most plot (Figure 8a, Figure 9a, Figure 10a and Figure 11a) is generated as follows. For each varied setting of either m, n, d or T, the running times of all trials are sorted from least to greatest, divided into four quartiles and plotted using a *box and whiskers* plot as follows. A rectangle represents all running times within the second and the third quartile. The horizontal orange bar across the rectangle is the median and it divides the data into two parts. The vertical black lines connected to the rectangle from below and from above are referred to as *upper whisker* and *lower whisker* respectively. They represent running times in the fourth and the first quartile respectively. The black horizontal bar at the end of each whiskers marks the lowest or highest running time, excluding outliers. *Outliers* are running times that are at least 1.5 times lower than the range of values in the lower whisker or 1.5 times higher than the data range in the upper whiskers. Outliers are represented by green diamonds.

For each experiment, the middle plot (Figure 8b, Figure 9b, Figure 10b and Figure 11b) divides all solved trials into soluble and insoluble and shows the minimum, median and maximum running time of the IP solver for each group and each varied parameter.

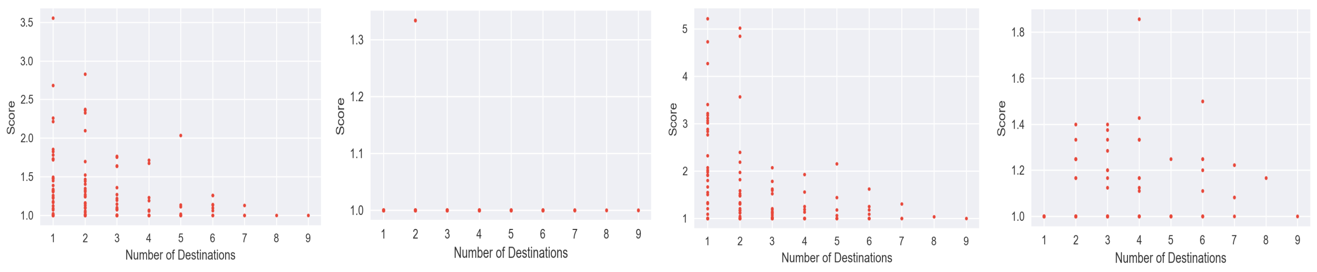
For each experiment, the right-most plot (Figure 8c, Figure 9c, Figure 10c and Figure 11c) shows the percentage of solved (red for soluble and blue for insoluble instances) and not solved (green) trials for each varied parameter.

Experiment 1. In the first experiment, we increased the number of flights m while maintaining constant numbers of airports n, destinations d and holiday time T. For values of m ($100 \leq m \leq 500$) in increments of 40, 10 TP instances were generated with $n = 11$, $d = 6$ and $T = 14$.

Figure 8 shows the obtained results. Figure 8a suggests that the running time rapidly increases with the increase of m. Most trials with $m \leq 220$ are solved within 200 seconds. However, for values of $m \geq 340$ most of the trials reached the time limit: the running time of all quartiles is 15 minutes, where for $m = 460$, all trials reached the time limit.

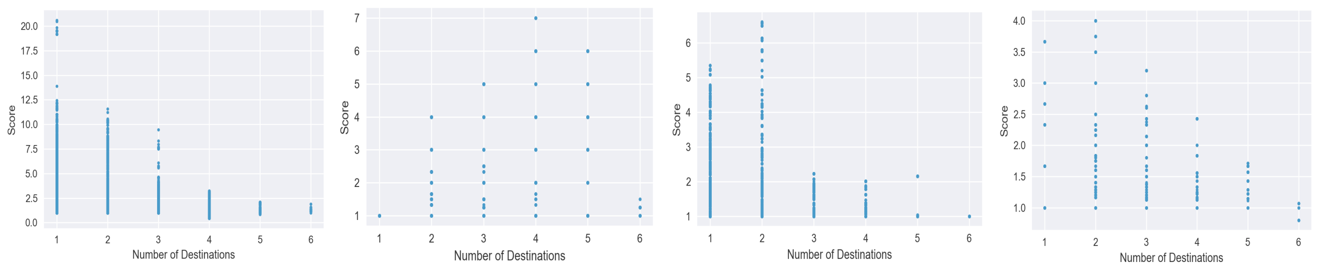
A slight decrease of the running time for trials with $m = 500$ can be seen, showing that for the chosen values of n, d and T, the instances with $m = 460$ take most time to solve. The region where problems change from being soluble to insoluble and their difficulty increases dramatically is referred to as *phase transition.* Figure 8 shows a possible existence of a phase transition for Experiment 1, occurring at $m = 460$. Effective methods to predict problem difficulty using phase transitions have been developed [36, 11].

Figure 8b shows that the IP model is faster at solving



(a) min cost (o_1) (b) min connecting airports (o_2) (c) min trip duration (o_3) (d) min number of flights (o_4)

Figure 6: The scores of all optimal solutions returned by the IP model, relative to each objective and grouped by d.



(a) min cost (o_1) (b) min connecting airports (o_2) (c) min trip duration (o_3) (d) min number of flights (o_4)

Figure 7: The scores of all optimal solutions returned by the CP model, relative to each objective and grouped by d.

insoluble trials than soluble trials. The same result was observed in the experiments described in Section 5.2.

Figure 8c shows an increase in the proportion of not solved trials as m increases, where for $m = 460$ all trials were not solved. For values of $m \leq 220$ an increase of the proportion of soluble trials can be observed with the increase of m, whereas for $m > 220$ there is lower number of soluble trials. These results can be explained as follows. Experiments show that the IP model is faster when run on insoluble trials. Therefore, soluble trials are expected to hit the time limit more often than insoluble trials, provided that the trials have the same size. Since there is a trend of increase in soluble trials for $m \leq 220$, we believe that the number of green trials that are soluble is higher than the number of green trials that are insoluble.

Experiment 2. In the second experiment, we increased n while maintaining constant numbers of m, d and T. For each n ($7 \leq n \leq 20$), 10 TP instances were generated, with $m = 150$, $d = 6$ and $T = 14$.

Figure 9 shows the obtained results: in Figure 9a it can be seen that only five trials reached the time limit and most of the trials were solved within 200 seconds time. A slight decrease of the running time with the increase of n can be observed. Figure 8b confirms previous observations that the IP model runs faster on insoluble trials.

Figure 9c shows that increasing n results in an increase of the proportion of insoluble trials, where for $17 \leq n \leq 20$ all trials are insoluble. This is due to the TP instance generator: adding more airports to an instance while maintaining m constant reduces the proportion of flights that visit destinations. The SCC test does not account for flight dates and therefore it is insufficient for filtering instances with high number of airports, compared to flights.

Experiment 3. In the third experiment, we increased d while maintaining constant numbers of m, n and T. For each d ($1 \leq d \leq 10$), 10 TP instances were generated, containing $m = 150$, $n = 11$ and $T = 14$.

The results in Figure 10 show that there is no particular relation between the running time and the value of d: the median for each d is about 50 seconds and almost all trials were solved for less than 200 seconds.

Similarly to Figure 9c, Figure 10c suggests that as d is increased, a larger proportion of the generated TP instances are insoluble most likely for similar reasons.

Experiment 4. In the fourth experiment, we increased T while maintaining constant numbers of m, n and d. For each T ($7 \leq T \leq 20$) in increments of 2, 10 TP instances were generated, containing $m = 150$, $n = 11$ and $d = 6$.

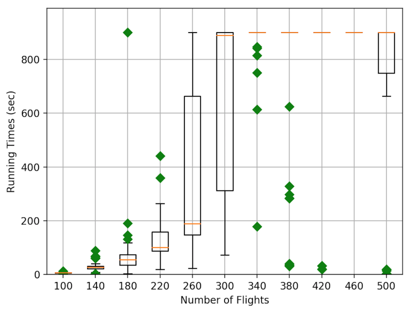
The results can be seen in Figure 11, where Figure 11a shows that most trials were solved in under 200 seconds time and that no particular relation between the value of T and the running time can be observed. Figure 11b confirms the previously noted observation that the IP model is better at solving insoluble instances. Figure 11c suggests that there is no particular relation between the value of T and the instance solubility, where the reason for having only insoluble trials for $T = \{7, 17\}$ is unknown.

5.5 Size of real-world TP instances

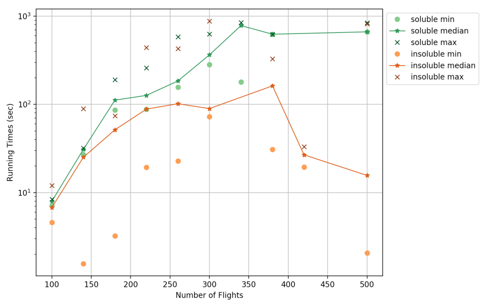
Knowing more about the size of the TP instances encountered in reality can give ideas for improvement of the models. This section quantifies *realistic size* in terms of m, n, d and T using some assumptions and the Skyscanner redirects data.

Let us assume that travellers in Europe typically visit at most 8 destinations, that their holiday lasts for no more than 30 days and that they don't put an upper limit of the number of connecting airports available for usage. The Skyscanner data was used to predict the approximate number of flights m for the assumed values of n, d and T as follows. For each value of d ($1 \leq d \leq 8$), where $n = 100$ $T = (d + 1) \times 3$ (settings 1-4 in Table 1), 50 TP instances were generated and their average, minimum and maximum m was calculated.

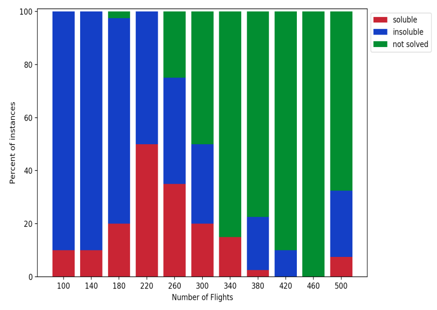
Table 2 shows the obtained results: for each value of d,



(a) all trials

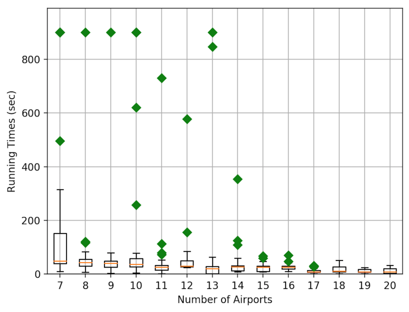


(b) soluble vs insoluble trials

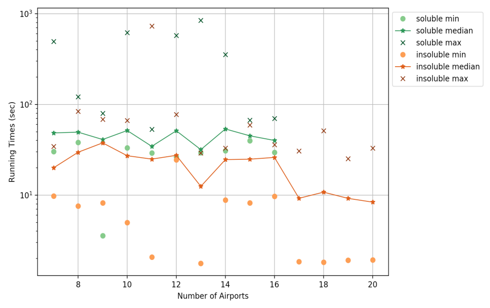


(c) solubility of trials

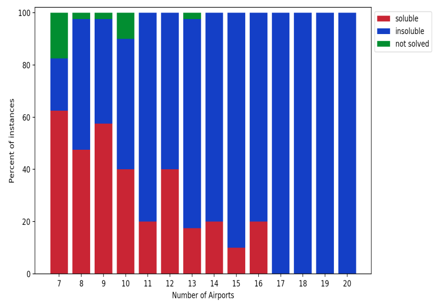
Figure 8: Experiment 1



(a) all trials

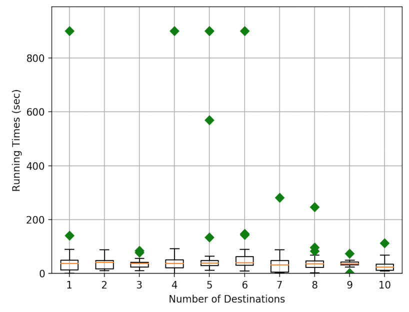


(b) soluble vs insoluble trials

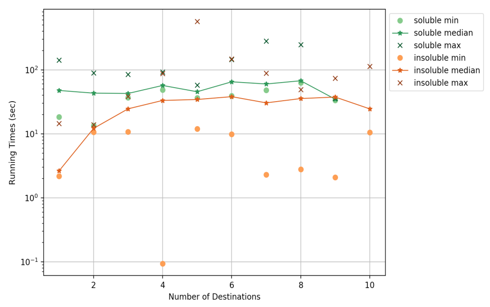


(c) solubility of trials

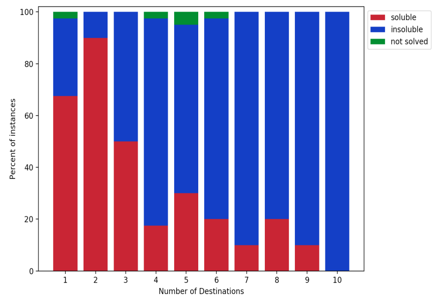
Figure 9: Experiment 2



(a) all trials

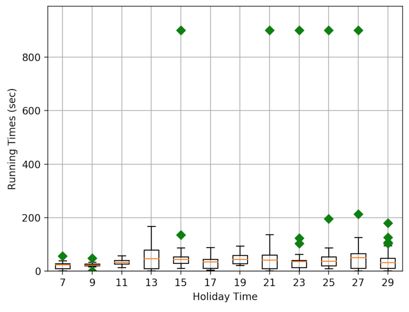


(b) soluble vs insoluble trials

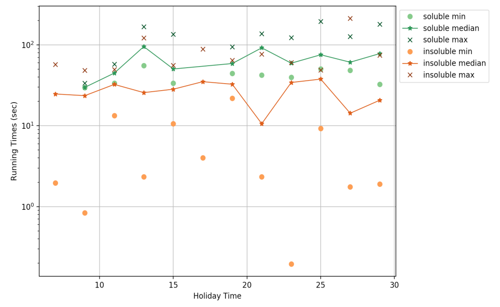


(c) solubility of trials

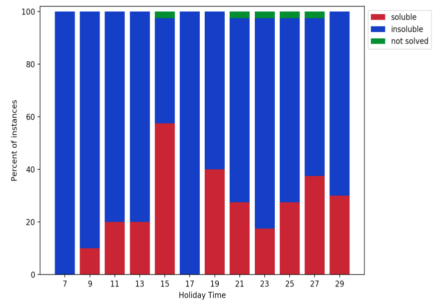
Figure 10: Experiment 3



(a) all trials



(b) soluble vs insoluble trials



(c) solubility of trials

Figure 11: Experiment 4

$m \sim d^2 \times 100$ on average, where small TP instances tend to have $m \leq 1000$ and large TP instances have $m \sim 7166$. This suggests that the choices of m in Table 1 were too small. However, the results in Section 5.2 and Section 5.4 show that both models would struggle to solve instances with higher m in terms of d.

5.6 Summary of Findings

The results from the comparison of the two models show that the IP model outperforms the CP model and its superiority becomes more evident with the increase of the instance

d	T	Average m	Minimum m	Maximum m
1	6	120.71	4	508
2	9	431.04	10	1,459
3	12	860.2	88	2,879
4	15	1,606.18	402	5,213
5	18	2,107.72	183	4,784
6	21	3,106.16	537	7,010
7	24	4,819.54	875	10,786
8	27	7,166.08	1,506	13,206

Table 2: The minimum, average and maximum number of flights m depending on d and T, where $T = (d + 1) \times 3$.

size. Furthermore, the solutions returned by the IP multiobjective optimisation tend to be close to the optimal values, whereas the CP model also returns solutions that have one or more objective values far from the optimum.

It was noted that for $d \geq 7$ most instances were insoluble. This suggests that further empirical evaluation with datasets that consist of higher proportion of soluble instances is needed to determine whether this behaviour is replicated.

The results from the evaluation of the IP model show that m has significant impact on the instance running time, where for TP instances with $m \geq 340$ the IP solver reached the time limit for most trials. This is not surprising. When m is increased, the search space grows in two dimensions for both models: for CP, the size of variables' domains and the number of variables in $\mathcal{S}$ both increase and for IP, this involves creating an additional row and column in the two-dimensional matrix comprising $x_{i,j}$ variables. Therefore, if the same experiments were conducted on the CP model, we would expect to observe similar behaviour.

For each of the evaluated datasets, the IP model is quicker at proving instances as insoluble than solving instances to optimality. This suggests that the solver spends substantial time searching for an optimum, once solubility is proved.

No strong correlation between the running time of the IP solver and the choice of values for T was observed. The running time tends to decrease with the increase of d or n, because the number of soluble (that is, "harder") instances in the dataset decreases with the increase of d or n.

We also estimated the size of real-world TP instances using the Skyscanner data. The results show the following relation between m and d: $m \sim d^2 \times 100$, which means that at their current state, both models are too inefficient to solve real-world TP instances as part of a real-time application. Improving their efficiency requires reducing the influence of m on their time complexity.

6. CONCLUSION AND FUTURE WORK

This work investigated the Traveller's Problem (TP): a combinatorial optimisation problem that finds direct applications in the travel industry. We constructed two algorithms for solving TP and some of its extensions and variations. The algorithms use Integer Programming (IP) and Constraint Programming (CP) and both can return optimal solutions subject to four objectives: cost, number of connecting airports, trip duration, number of flights, as well as any combination of them. In addition, the CP model implements two hard constraints that a TP solution may be required to satisfy.

The author of this project was given access to flight booking data for the period of half a year by Skyscanner; a travel metasearch engine widely used for booking itineraries online. The utilisation of this data involved building a framework for processing the data and creating TP instances from it.

The evaluation was performed on each model with TP instances generated from Skyscanner data. The results show that the IP is much quicker at solving problems than the CP model, where the difference between their running times increases with the increase of the instance size. The time complexity of both models is highly influenced on the number of flights in the instance and this makes them too inefficient to solve real-world TP instances, where it was observed that $m \sim d^2 \times 100$.

Future work should aim to build an algorithm for solving real-world TP instances. Achieving this milestone is worthwhile, because it will have a significant impact on the travel metasearch industry and will improve travelling and trip planning experience of travellers around the world. The rest of this section gives suggestions for improving and evaluating the models.

An alternative way to model TP is to have all destinations as decision variables instead of flights. For example, the CP model can have an array $\mathcal{D}$ of d variables that denotes the order in which the destinations are visited for the first time, where the domain of each variable is $(1, ..., d)$. For example, having $\mathcal{D}[i] = j$ would mean that destination with id j is the i^{th} visited destination, given that each destination has a unique integer from 1 to d assigned as an id.

Both models do not implement any heuristics and both solvers execute default search algorithms (Section 3.4). Existing work has shown that the choice of heuristics can have significant impact on the search effort [27, 24, 9]. Future work should investigate existing heuristics and search strategies and look into developing specialised versions for TP.

In order to reduce the size of the search space, each TP instance can be preprocessed before its model is created. Preprocessing can consist of concatenation of pairs of flights as follows: let f_j and f_k be two flights having $A^a_j = A^d_k = A_i$ for $A_i \notin D$ and $t_j + \Delta_j + C_{A^a_j} \leq t_k$. Then a new flight $f_{j,k}$ called a *meta-flight* can be constructed with $A^d_{j,k} = A^d_j$, $A^a_{j,k} = A^a_k$, $t_{j,k} = t_j$, $\Delta_{j,k} = t_k - t_j + \Delta_k$ and $c_{j,k} = c_j + c_k$. This process is repeated for the concatenated flights until a new set F' of flights only to and from destinations is obtained. Creating meta-flights removes the need to model connecting airports and potentially reduces the number of flights in the instance. However, it requires storing extra data, such as the set of flights each meta-flight is composed of (if solution with minimum number of flights is required). Future work should further investigate this idea and its trade-offs.

During evaluation, we repeatedly encountered the issue of having too few soluble TP instances for high values of d. In future work, modifying the TP instance generator to build trips from the Skyscanner data and then build TP instances around these pre-generated trips could solve this issue.

In real-world applications, incumbent solutions that are sufficiently close to the optimum are often accepted. Future work can investigate the trade-off between the quality of the incumbent solution and the time taken to compute it. This can be done by repeatedly solving a set of soluble instances, each time varying the imposed time limit t and calculating the scores of the returned solutions using Equation 29.

References

[1] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Oper. Res. Lett.*, 33(1):42–54, January 2005. ISSN 0167-6377. doi: 10.1016/j.orl.2004.04.002. URL `http://dx.doi.org/10.1016/j.orl.2004.04.002`.

[2] NR Achuthan and L Caccetta. Integer linear programming formulation for a vehicle routing problem. *European Journal of Operational Research*, 52(1):86–89, 1991.

[3] 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`.

[4] R. Agarwala, D. Applegate, D. Maglott, G. Schuler, and A. Schäffer. A fast and scalable radiation hybrid map construction and integration strategy. *Cold Spring Harbor Laboratory Press*, pages 350–364, 2000. ISSN 1088-9051.

[5] 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`.

[6] Alper Atamtürk and Martin W. P. Savelsbergh. Integer-programming software systems. *Annals of Operations Research*, 140(1):67–124, 2005. ISSN 1572-9338. doi: 10.1007/s10479-005-3968-2. URL `http://dx.doi.org/10.1007/s10479-005-3968-2`.

[7] Bruno De Backer, Vincent Furnon, Paul Shaw, Philip Kilby, and Patrick Prosser. Solving vehicle routing problems using constraint programming and metaheuristics. *Journal of Heuristics*, 6(4):501–523, 2000. ISSN 1572-9397. doi: 10.1023/A:1009621410177. URL `http://dx.doi.org/10.1023/A:1009621410177`.

[8] E. 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`.

[9] 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.

[10] Yves Caseau and François Laburthe. Solving small tsps with constraints, 1997.

[11] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the really hard problems are. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'91, pages 331–337, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc. ISBN 1-55860-160-0. URL `http://dl.acm.org/citation.cfm?id=1631171.1631221`.

[12] 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`.

[13] G. B. Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1):101–111, 1960.

[14] G. B. Dantzig, D. R. Fulkerson, and S. 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`.

[15] 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`.

[16] 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`.

[17] 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`.

[18] Java Documentation. Java hashmap, 2017. URL `https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html`. Accessed: 2017-03-16.

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

[20] Eurostat. Booking transport online within EU, 2014. URL `http://ec.europa.eu/eurostat/statistics-explained/index.php/File:Online_booking_for_transport_(%25_of_all_trips,_EU-28,_2014_(%C2%B9)_new.png#file`. Accessed: 2017-02-21.

[21] 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`.

[22] M. M. Flood. The traveling-salesman problem. *Operations Research*, 4(1):61–75, 1956. doi: 10.1287/opre.4.1.61. URL `http://dx.doi.org/10.1287/opre.4.1.61`.

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

[24] Ian Gent, Ewan MacIntyre, Patrick Prosser, Barbara 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] Gurobi. Industries using Gurobi. URL `http://www.gurobi.com/products/industries/industry-overview`.

[26] R. M. Haralick and G. 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] 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`.

[29] 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`.

[30] Philip Kilby, Patrick Prosser, and Paul Shaw. A comparison of traditional and constraint-based heuristic methods on vehicle routing problems with side constraints. *Constraints*, 5(4):389–414, 2000. doi: 10.1023/A:1009808327381. URL `http://dx.doi.org/10.1023/A:1009808327381`.

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

[32] 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`.

[33] 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`.

[34] P. C. Mahalanobis. A sample survey of the acreage under jute in bengal. *Sankhya: The Indian Journal of Statistics (1933-1960)*, 4(4):511–530, 1940. ISSN 00364452. URL `http://www.jstor.org/stable/40383954`.

[35] Gilles Pesant, Michel Gendreaul, and Jean-Marc Rousseau. *GENIUS-CP: A generic single-vehicle routing algorithm*, pages 420–434. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997. ISBN 978-3-540-69642-1. doi: 10.1007/BFb0017457. URL `http://dx.doi.org/10.1007/BFb0017457`.

[36] Patrick Prosser. An empirical study of phase transitions in binary constraint satisfaction problems. *Artificial Intelligence*, 81(1):81 – 109, 1996. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/0004-3702(95)00048-8. URL `http://www.sciencedirect.com/science/article/pii/0004370295000488`.

[37] 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`.

[38] 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.

[39] Louis-Martin Rousseau, Michel Gendreau, and Gilles Pesant. Solving small vrptws with constraint programming based column generation. *Proceedings of CPAIORâĂŹ02*, 2002.

[40] Skyscanner. Skyscanner: Travel Booking Website. URL `http://skyscanner.net`. Accessed: 2017-02-21.

[41] Skyscanner. Skyscanner Traffic Overview, 2017. URL `https://www.similarweb.com/website/skyscanner.com`. Accessed: 2017-03-09.

[42] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

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

[44] Wikipedia. The busiest airports in Europe. URL `https://en.wikipedia.org/wiki/List_of_the_busiest_airports_in_Europe`. Accessed: 2017-02-26.

[45] Jr. William T. McCormick, Paul J. Schweitzer, and Thomas W. White. Problem decomposition and data reorganization by a clustering technique. *Operations Research*, 20(5):993–1009, 1972. doi: 10.1287/opre.20.5.993. URL `http://dx.doi.org/10.1287/opre.20.5.993`.

APPENDIX

A. COMPLEXITY OF TP

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 1.2. Its NP-hardness follows by a reduction from the Hamiltonian Cycle problem. The proof is presented by [23].

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 airport.
- 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 $A_{j_q}^d = A_{i_p}$ 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.

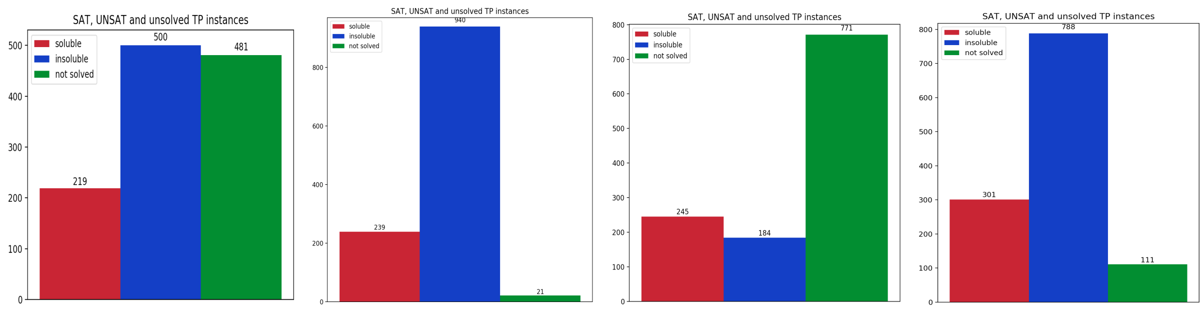
The transformation from a TSP instance to an instance of TPD can be done in polynomial time. For each of the $n(n$-$1)/2$ distances d(A_i, A_j) that must be specified in π, it is sufficient to check that the same cost is assigned to the flights from A_i to A_j for all dates.

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. $\square$

B. ADDITIONAL EVALUATION RESULTS

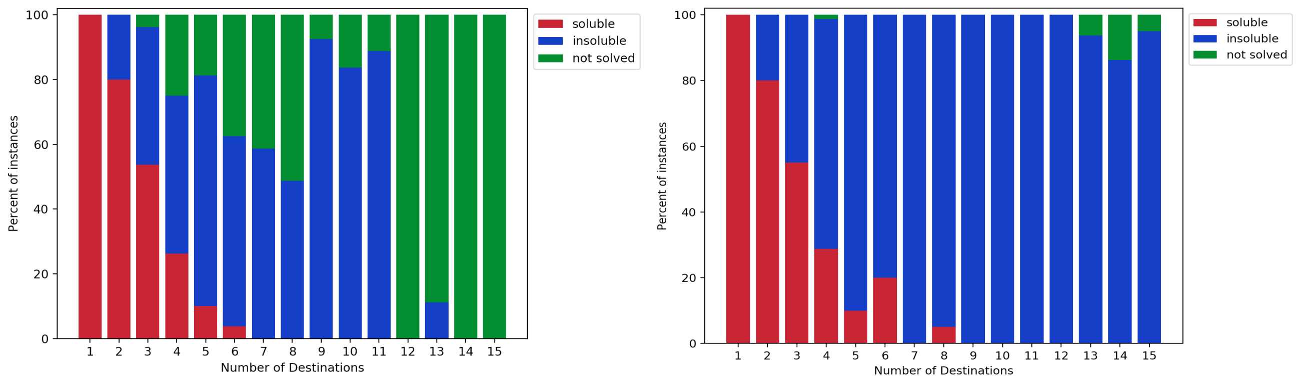
B.1 Evaluation of IP

Running time per objective function. Figures 15a, 15b, 15c and 15d show the median running time of each objective function for each varied parameter in Experiment 1, 2, 3 and 4 respectively. All four objective functions performed similarly during Experiment 1. For Experiment 2, an unusual rise in the running time of the number of connecting flights objective is observed. Figures 15c and 15d show that the trip duration objective has the slowest running time and the rest have similar performance in terms of running time. A reason why trip duration tends to be less fast could be that it involves creating an additional set of variables $y_{i,j}$ and satisfying additional constraints, as presented in Section 3.3.



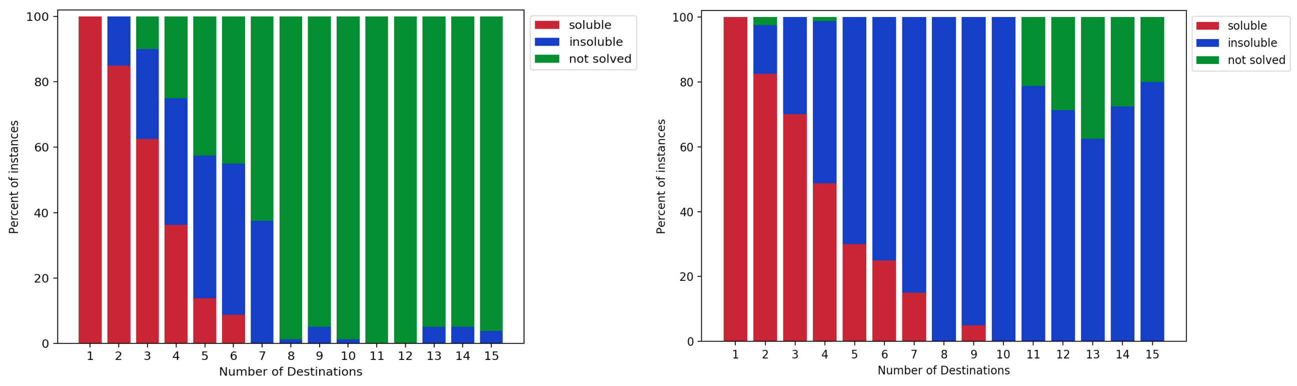
(a) CP $\mathcal{D}_3$ (b) IP $\mathcal{D}_3$ (c) CP $\mathcal{D}_4$ (d) IP $\mathcal{D}_4$

Figure 12: The number of soluble, insoluble and not solved trials for each model and dataset.



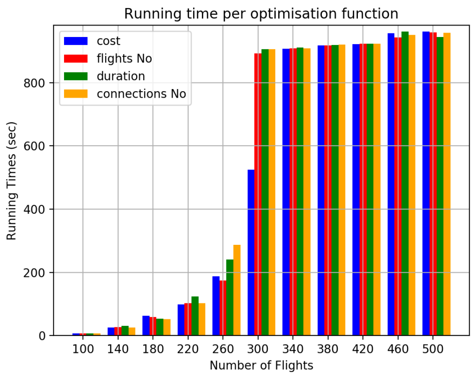
(a) CP $\mathcal{D}_3$ (b) IP $\mathcal{D}_3$

Figure 13: The percentage of soluble, insoluble and not solved instances for each setting of m, n, d and T in dataset $\mathcal{D}_3$ for the IP and CP model.

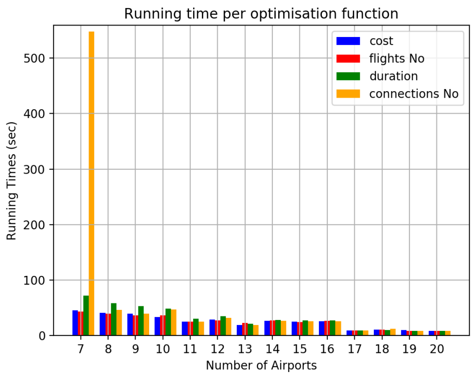


(a) CP $\mathcal{D}_4$ (b) IP $\mathcal{D}_4$

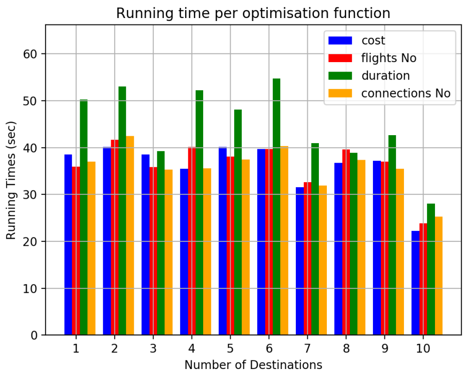
Figure 14: The percentage of soluble, insoluble and not solved instances for each setting of m, n, d and T in dataset $\mathcal{D}_4$ for each model.



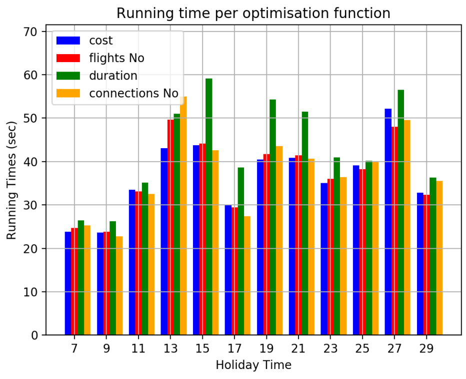
(a) Experiment 1



(b) Experiment2



(c) Experiment 3



(d) Experiment 4

Figure 15: Median running time of each objective for each of the experiments