Expert Systems With Applications 130 (2019) 172–187

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Flying tourist problem: Flight time and cost minimization in complex routes

Rafael Marques[a,b], Luís Russo[a,b], Nuno Roma[a,b,*]

[a] Instituto Superior Técnico, Universidade de Lisboa, Portugal
[b] INESC-ID, Rua Alves Redol, 9, Lisboa 1000-029, Portugal

ARTICLE INFO

Article history:
Received 26 October 2018
Revised 11 March 2019
Accepted 11 April 2019
Available online 16 April 2019

Keywords:
Flight search
Traveling salesman problem
Combinatorial optimization
Evolutionary algorithms.

ABSTRACT

This work introduces and formalizes the Flying Tourist Problem (FTP), whose goal is to find the best schedule, route, and set of flights for any given unconstrained multi-city flight request. To solve the FTP, the developed work proposes a methodology that allows an efficient resolution of this rather demanding problem. This strategy uses different heuristics and meta-heuristic optimization algorithms, allowing the identification of solutions in real-time, even for large problem instances. The implemented system was evaluated using different criteria, including the provided gains (in terms of total flight price and duration) and its performance compared to other similar systems. The obtained results show that the developed optimization system consistently presents solutions that are up to 35% cheaper (or 60% faster) than those developed by simpler heuristics. Furthermore, when comparing the developed system to the only publicly available (but not-disclosed) alternative for flight search, it was shown that it provides the best-recommended and the cheapest solutions, respectively 74% and 95% of the times, allowing the user to save time and money.

1. Introduction

Consider a person who wants to visit N different cities in the most efficient way possible. In the combinatorial optimization domain, this problem is well known as the Traveling Salesman Problem (TSP) and it is considered to be part of one of the most complex classes of problems (Karp, 1972). This difficulty arises from the exponential growth of the number of possible solutions, given approximately by $N!$.

Upon the introduction and formalization of the TSP, this problem could simply be stated as *"Given a list of N cities and the distances between them, what is the best closed tour that visits every city exactly once?"* or, as considered in the graph theory domain, *"Given a complete undirected graph with weighted edges, what is the minimum cost Hamiltonian cycle?"*. This formulation has a vast number of applications and it is very useful for the majority of the routing problems that occur on a network that can be modulated as a graph (Applegate, Bixby, Chvátal, & Cook, 2007; Gross, Yellen, & Zhang, 2013).

In this paper, the classic problem of traveling through N different cities is revisited, but assuming the particular case applied to commercial flights transportation. This specific formulation is closely related to the generic TSP, in the sense that both problems aim to find the most efficient way to visit a given number of cities. However, there are some considerable differences. While the generic TSP (and its asymmetric variation) considers that the cost between two cities is always constant over time, such assumption is certainly not true for the case of commercial flights, as the tickets price depend not only on the date, but also on the direction of the trip (i.e., $price(A \rightarrow B) \neq price(B \rightarrow A)$).

As a result of this time (and direction) dependency, a reasonable assumption would be to consider this as a Time-Dependent Traveling Salesman Problem (TDTSP) (Fox, Gavish, & Graves, 1980). Due to the specific characteristics and goals of the problem, this is, in fact, the case. However, the majority of the literature around the TDTSP makes a number of assumptions that, in many cases, do not adequately describe the problem. An example of this is the TDTSP formulation introduced by Picard and Queyranne (1978), which considers that the waiting period in each city is exactly one time-period. Such an assumption is not always verified, not only due to existing restrictions of flight offers in such routes, but also because flying dates are also dependent on the traveler's convenience.

Another variation of the TSP that is directly related to the considered problem is the TSP with time windows, where each city must be visited in a given time window. There are several

* Corresponding author at: INESC-ID, Rua Alves Redol, 9, 1000-029 Lisboa, Portugal.
E-mail address: nuno.roma@inesc-id.pt (N. Roma).

https://doi.org/10.1016/j.eswa.2019.04.024
0957-4174/

approaches to solve this problem. A recent and efficient approach was presented by Boland, Hewitt, Vu, and Savelsbergh (2017), which uses a time-expanded Integer Linear programming (ILP) formulation that is exploited without ever explicitly creating the complete formulation. A carefully designed partially time-expanded network is used to produce upper and lower bounds, which are iteratively refined until optimality is reached.

The observed limitations of these formulations lead to the need of a more realistic formulation of the problem (herein referred to as the *Flying Tourist Problem* (FTP)). In particular, while the goal of the TSP is to find the best *route* which efficiently connects all N cities (minimizing the distance), the goal of the FTP is to find the best *route, schedule* and *set of flights* for the trip. Furthermore, the objective function of this problem must also reflect multiple objectives, particularly the total *cost* and *flight duration* of the trip.

To the best of the authors' knowledge, no formal solution exists to solve this specific problem. While there are several meta-search engines that are capable of responding to multi-city requests, the user must always specify a particular route and schedule. However, from the analysis of the search space perspective, such scenario corresponds to the inspection of a single solution among the $N!$ solutions to the problem. Furthermore, as the number of cities to be visited increases, finding the best set of flights rapidly becomes a slow, time consuming, and tedious process.

On the other hand, a commercial flight search application was recently launched by *Kiwi* (www.kiwi.com), denoted as *Nomad*. This web-service addresses the same problem as the presented work and it is currently the only publicly available (non-disclosed) tool for the resolution of this problem. Consequently, it will be treated as the state-of-the-art in the quantitative evaluation of the developed system. According to the obtained results, the presented optimization algorithm is able to provide cost and flight duration gains as high as 25% when compared to *Kiwi*'s *Nomad* implementation, when considering trips with more than 5 nodes.

With this in mind, the goals and major contributions of this work are the following:

- Formal definition of the *Flying Tourist Problem* that looks for the best schedule and set of flights that visit a given list of cities, by minimizing both the total *cost* and *flight duration*;
- Identification of several optimization methods that can be used to solve the stated problem;
- Implementation of a system prototype capable of providing a high-quality solution for the problem (in an efficient manner) when using real-world data and resources;
- Analysis and evaluation of the obtained results, not only in terms of the obtained set of solutions, but also in terms of the achieved gains (time and cost).

The remaining of this article is structured as follows. Section 2 presents a brief overview of the most relevant literature. This is followed by a formal definition of the problem, presented in Section 3. Section 4 covers in detail the considered optimization procedure and Section 5 presents the architecture of the developed system prototype and the most relevant implementation details. Section 6 presents a quantitative analysis of the obtained solutions based on several conducted experiments and compares it to the current state-of-the-art. Section 7 briefly describes other related TSP formulations that have been presented in recent literature and Section 8 presents the major conclusions and addresses possible future work directions.

2. Literature review

The TSP is a classical formulation in several domains, including routing and graph theories (Applegate et al., 2007; Gross et al., 2013). It is also frequently applied in other specific optimization scenarios, including the Vehicle Routing Problem (VRP) or the single-machine scheduling. Although its symmetric versions over an undirected graph are usually considered, other variations are also common, based on its asymmetric counterpart over a directed graph (Öncan, Altınel, & Laporte, 2009).

The several optimization algorithms that have been proposed for the TSP are usually grouped into *exact, heuristic* and *metaheuristic* approaches.

Most exact algorithms (Laporte, 1992a; Laporte, 1992b) rely on an Integer Linear Programming (ILP) formulation, while others are based on branch and bound (Lawler & Wood, 1966; Morrison, Jacobson, Sauppe, & Sewell, 2016) and minimum spanning tree (Bazlamaçci & Hindi, 2001) techniques. However, the long execution times that characterize these approaches make them impractical in most application scenarios.

As a result, many heuristic methods to solve either the TSP (Rego, Gamboa, Glover, & Osterman, 2011) and the VRP (Laporte, Gendreau, Potvin, & Semet, 2000) have been proposed. Among the most common approaches are improvement heuristics, such as the k-opt exchange (Golden, Bodin, Doyle, & Stewart, 1980), construction heuristics, including the nearest neighbor (Laporte, 1992a) and tabu search (Glover & Laguna, 1999). For the particular case of the TSP, the Lin-Kernighan heuristic (Lin & Kernighan, 1973) is a particularly efficient algorithm. It was the state-of-the-art (for asymmetric TSPs) for over a decade. In general, its results are within 2% of the lower bound and often generate optimal solutions (Johnson & McGeoch, 1997). Despite this, the Lin-Kernighan heuristic cannot be directly applied to the asymmetric TSP. Instead, it is necessary to apply a graph transformation, converting the asymmetric TSP into a symmetric instance, with twice as many nodes (Jonker & Volgenant, 1983).

In the last 30 years, a great interest has also been devoted to the usage of metaheuristic algorithms to solve the TSP. Metaheuristics can be seen as higher order heuristics: they take advantage of an underlying heuristic and guide the algorithm to produce an efficient search space exploration. The class of metaheuristics is vast and includes algorithms such as the Simulated Annealing (Kirkpatrick, Gelatt, & Vecchi, 1983), Genetic Algorithm (Goldberg, 1989), Ant Colony Optimization (Dorigo & Gambardella, 1997), Particle Swarm Optimization (Kennedy & Eberhart, 1995) and many more.

The Simulated Annealing (SA) was one of the first metaheuristic methods to be developed, and its success resides on its ability to escape local minimum, by performing hill-climbing techniques. During the development of these algorithms, the TSP was the first optimization problem to be solved using these metaheuristics (Malek, Guruswamy, Pandya, & Owens, 1989). This was primarily because the TSP served as a good benchmark test for evaluating the algorithm's performance. The VRP was a natural consequence of this (Osman, 1993). There are also several works which focus on TSP and VRP with time-windows (Czech & Czarnas, 2002; Ohlmann & Thomas, 2007), including real-world environments that consider that the travel time is stochastic instead of well defined (Laporte, 1992b), as well as several SA algorithms which focus on multi-objective optimization (Czyżak & Jaszkiewicz, 1998).

The Ant Colony Optimization (ACO) is actually a group of several different optimization algorithms, as the Ant System (AS), Elitist AS, Ant Colony System, and min-Max AS. Just like the SA, the TSP was one of the first optimization problems to be solved using ACO. The first of these algorithms, the AS, did not consistently present high-quality results. However, the later algorithms (including the Ant Colony System) were capable of competing with the state-of-the-art (Dorigo & Gambardella, 1997). After fine-tuning, the ACO was rapidly applied to a vast collection of

combinatorial optimization problems, as the VRP (Gambardella, Taillard, & Agazzi, 1999), quadratic assignment (Gambardella, Taillard, & Dorigo, 1999), and weighted tardiness (den Besten, Stützle, & Dorigo, 2000), where the TSP occurs as a special case of these three problems. Other multi-objective examples were presented by Doerner, Gutjahr, Hartl, Strauss, and Stummer (2004) and Lopez-Ibanez and Stutzle (2012). More recent works obtained fast and reliable TSP solutions by using parallelism and cooperation among multiple colonies (Gülcü, Mahi, Baykan, & Kodaz, 2018). Their parallel cooperative hybrid algorithm (PACO-3Opt) avoids local minima by sharing information among colonies. This process continues until the termination criterion meets. Thus, it can reach the global optimum.

Another important metaheuristic is the Particle Swarm Optimization (PSO), proposed by Kennedy and Eberhart (1995) and Shi and Eberhart (1998). In this method, each particle represents a potential solution to the search space and the algorithm proceeds by having several particles moving around. The movement of each particle, characterized by its position and speed, is influenced by its known optimum value and the optimum value of the other particles. This gives the particles swarm like behavior. The method was later applied to TSP by a series of authors, including Clerc and Kennedy (2002), Goldbarg, de Souza, and Goldbarg (2006), Rosendo and Pozo (2010) and Jianchao and Zhihua (2006).

Besides these classical approaches, there are also some other proposals that apply hybrid optimization algorithms, combining two or more meta-heuristics. An example of this is the genetic simulated annealing ant colony system (Chen & Chien, 2011). Another recent improvement to solve the TSP was obtained by Osaba, Ser, Sadollah, Bilbao, and Camacho (2018), which use the Water Cycle Algorithm (WCA), a nature-inspired meta-heuristic proposed in 2012. This algorithm is motivated by the natural surface runoff phase in the water cycle process and on how streams and rivers flow into the sea. The application of WCA to the TSP shows relevant improvements to the existing approaches, both in terms of convergence, speed, and optimality.

Another bio-inspired metaheuristic is the bat-algorithm, proposed in 2010, based on the echolocation or bio-sonar characteristics of microbats. In particular, Osaba, Yang, Diaz, Lopez-Garcia, and Carballedo (2016) presented a discrete version of this algorithm that can be applied to the symmetric and asymmetric TSPs. They showed that good performance results can be obtained with this method when compared to the state-of-the-art.

A heuristic that is particularly resilient to large instances is the Partial OPtimization Metaheuristic Under Special Intensification Conditions (POPMUSIC). An application of this heuristic to the TSP (considering instances of up to several million cities) was proposed by Taillard and Helsgaun (2019). Their approach considers only a subset of the edges connecting the cities and the candidate edges are found with a technique exploiting tour merging and POPMUSIC. Then, high quality solutions can be efficiently found by providing these candidate edges to a local search engine.

An interesting variation of TSP is the Pickup and Delivery Traveling Salesman Problem with Handling costs (PDTSPH), where a single vehicle has to transport loads from origins to destinations. Loading and unloading of the vehicle is operated in a Last-In-First-Out (LIFO) fashion. However, if the load that must be unloaded is not the one that was loaded last, additional handling operations are allowed to unload and reload other loads that block the access. The additional handling operations take time and effort, to which penalty costs are associated. The aim of the PDTSPH is to find a feasible route such that the total costs, consisting of travel costs and penalty costs, are minimized. This problem was recently studied by Veenstra, Roodbergen, Vis, and Coelho (2017), who used a Large Neighborhood Search (LNS) heuristic. The authors performed exhaustive experimental tests to validate this approach, which obtained new optimal solutions in several instances.

However, despite its tight relation with the TSP (and with some of its presented variations), the particular problem that is formulated in this paper (see Section 3 below) differs significantly from the previous alternatives. Recently, some other formulations more closely related to the proposed FTP were presented in the literature. Some of these works shall be surveyed in Section 7, after the FTP formulation and evaluation, to allow a more consolidated context and analysis.

3. Flying tourist problem (FTP) formulation

Consider a tourist who wishes to take a trip that visits every node (city) i in the set of nodes V, $|V| = N$, with no particular order. The start node will be denoted as v_0, while the return node as v_{n+1}, and the complete set of nodes is given by $V_c = V \cup \{v_0\} \cup \{v_{n+1}\}$. The trip must start at a time $t \in T_0 = [T_{0m}, T_{0M}]$. Upon visiting a node, the tourist will stay there for a duration of d time-units (days). Consider that for each node to be visited, there is a range for the value that d might take, restricted as $d \in d_i = [d_{im}, d_{iM}]$ and $d_{iM} \geq d_{im} \geq 1$. The complete set of durations associated to each city is given by $D = \{d_i | i \in V\}$, therefore $|D| = N$. Furthermore, to each city $i \in V$, there is an associated time-window w_i which defines the set of dates in which the city i may be visited. The set of all time windows is denoted $TW = \{w_i | i \in V\}$ and has size N, $|TW| = |V| = N$.

The FTP is completely defined by a structure $G = (V_c, A, T_0, D, TW)$, used to create a multipartite graph describing the request. This multipartite graph is divided into k layers, where each layer corresponds to a particular moment in time. Besides this, every node in a layer is connected to all nodes in the subsequent layer. The set of arcs that connects these nodes is given by A. To each arc $a \in A$, it is associated a cost c_a (ticket cost) and a processing time p_a (flight duration), which depend upon the routed nodes, as well as the time (t) in which the arc transition is initiated, that is, $\forall a_{ij}^t \in A$, $c_{ij}^t \geq 0$ and $p_{ij}^t \geq 0$.

A valid solution s to the formulated FTP is a set of arcs (commercial flights) which start from node v_0 during the defined start period, visit every node i in V during its defined time-window w_i, by considering the staying duration defined by d_i, and finally return to node v_{n+1}. The set of all valid solutions is given by S. The goal of the FTP is to find the global minimum $s^* \in S$, with respect to the considered objective function.

The objective function associated to this problem depends on the user criteria. While some users might consider the expended cost to be the most important factor, there are others who consider the total flight duration of crucial importance. Thus, a total of three different objective functions shall be herein considered: (i) the flight price F_p (see Eq. (1)), (ii) the flight duration F_d (see Eq. (2)), and (iii) a balanced cost F_{bc} (see Eq. (3)), corresponding to a weighted sum between the former two.

$$F_p(s) = \sum_{n=0}^{N+1} c(s[n]) \tag{1}$$

$$F_d(s) = \sum_{n=0}^{N+1} p(s[n]) \tag{2}$$

$$F_{bc}(s) = \sum_{n=0}^{N+1} w_c * c(s[n]) + w_p * p(s[n]) \tag{3}$$

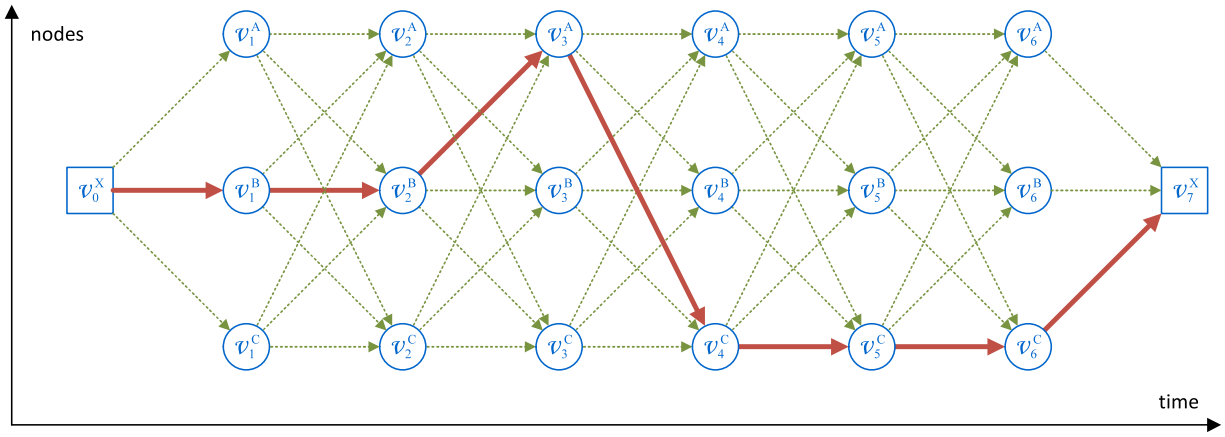


Fig. 1. Illustration of a Flying Tourist Problem using a multipartite graph. To each node (A,B,C) it is associated a waiting period of 1, 2, and 3 time-units, respectively. The red arrows represent a possible solution to the problem.

Fig. 1 illustrates the multipartite graph associated to a simple instance of the FTP with $v_{n+1} = v_0 = X$, one possible start date ($t = 0$), 3 nodes to visit (A, B, C), with a fixed duration of 1, 2, and 3 time-units, respectively, and no constraints relative to the time-window of each city. A possible solution to this problem instance corresponds to the set of arcs $(a^0_{X,B}, a^2_{B,A}, a^3_{A,C}, a^6_{C,X})$.

Despite the apparent complexity of the proposed definition, it can be used to state ordinary and real-life flight searches, including one-way and round-trip flights. For example, the problem of finding a single flight from A to B at date T can be instantiated as a FTP given by $v_0 = A$, $v_{n+1} = B$, $T_0 = T$, and $V = D = TW = \{\}$. In its turn, a round-trip flight involving the same two cities and the same start date, in which the staying period in B is b days, is given by $v_0 = v_{n+1} = A$, $T_0 = X$, $V = \{B\}$, $D = \{b\}$ and $TW = \{\}$. Thus, this definition is adequate either for simple and complex trips, which can be customized according to the user search criteria, by setting either an extended start period, or flexible waiting periods.

3.1. Relation to the TSP

As previously stated, the proposed FTP is closely related to the TSP and to its time-dependent variation. Given the following list of constraints:

1. $v_{n+1} = v_0$;
2. $T_0 = 0$;
3. $w_i = [0, +\infty[$, $\forall i \in V$;
4. $d_i = 1$, $\forall i \in V$;
5. $c^t_{ij} = c_{ij}$, $\forall i, j \in V$, $\forall t$;

one observes that constraints (1–4) provide a reduction of the devised FTP to a TDTSP, as proposed by Picard and Queyranne (1978), and the final constraint (5) reduces the problem to the classical TSP.

Since the FTP occurs as a generalization of the TSP, and given that the latter problem is well-known to be Np-hard complex, then so is the former one.

3.2. Graph construction

By considering the presented FTP definition, the total number of layers (k) of the devised multipartite graph represents the total time span between the earliest date at which the trip might start and the latest date in which it should finish. The arcs that connect those nodes are divided into three groups: *initial*, *transition* and *final* arcs.

The *initial* arcs are those which might initiate the trip. Consequently, they must start at node v_0, at a time $t \in T_0 = [T_{0m}, T_{0M}]$, connecting v_0 to every node in V. There are a total of $k_i = T_{0M} - T_{0m} + 1$ layers for the initial arcs.

Conversely, the *final* arcs are those that connect every node in V to the return node, v_{n+1}. There are as many final layers as there are initial layers, and the final layer extends from T_{fm} to T_{fM}, where $T_{fm} = T_{0m} + \sum(D)$ and $T_{fM} = T_{0M} + \sum(D)$, where $\sum(D)$ corresponds to the summation of all entries belonging to D. In the example depicted in Fig. 1, there is a single initial and final layer, since there is only one possible start date.

The *transition* arcs are those which fully connect the N nodes belonging to V. The earliest transition arc occurs at a time no sooner than $t_1 = T_{0m} + \min(D)$, where $min(D)$ corresponds to the lowest entry of the set of staying durations. Hence, if the trip starts by transiting an initial arc at time T_{0m}, the first transition arc might only be traversed $min(D)$ time-units later. By following a similar approach, the latest transition arc can occur no latter than $t_2 = T_{0M} + \sum(D) - \min(D)$. Thus, there are a total of $k_2 = t_2 - t_1 + 1$ transition layers, and $k_2 * n * (n-1)$ transition arcs.

The union of the initial, transition and final arcs gives the set A of all the arcs, which may be used to construct a solution to the requested trip.

Having the information relative to the multipartite graph associated to the devised FTP, it is now possible to construct a three-dimensional array matrix representing this problem, where each entry of the array corresponds to an arc connecting two nodes, at a particular moment in time. This weight matrix is initialized with a very high cost value (as to reject arcs which may not be part of the solution), and every entry of it is updated according to the information of the multipartite graph and the respective objective function. Finally, this weight matrix may be used as input for the optimization system (see Section 4).

Although it is clear that any arc $a \in A$ corresponds to a particular flight, it should be noted that no specific or limiting assumption was considered up until now. Instead, it was assumed an entirely abstract arc definition, connecting two nodes at a specific moment in time. In order to transform this set of arcs into a corresponding set of flights, it is necessary to obtain real-world flight data from some external source. This will be further detailed in Section 5.

4. Optimization system

Three distinct and widely known metaheuristic algorithms were considered to implement the devised optimization system: the Simulated Annealing (SA), the Ant Colony Optimization (ACO), and

the Particle Swarm Optimization (PSO). Each FTP instance is solved with these three algorithms and the best solution is selected.

4.1. Simulated annealing

The implemented SA solver receives, as input, the weight matrix of the problem instance, together with other parameters regarding the depot nodes and the waiting periods. Based on this input data, an initial solution x is randomly generated. This solution must be valid, that is, it may not violate any problem constrain. To create such a valid initial solution x, a closed trip is randomly generated based on the set of vertex V_c, on the starting date t_0 (according to T_0), and on the time of each node (according to t_0 and the waiting period of each previously visited node).

The SA metaheuristic relies on two iterative cycles: the outer cycle and the inner cycle (Markov chain) – see Algorithm 1. At each iteration step, the inner Markov chain is responsible for generating a new candidate solution y, according to an appropriate neighborhood function and a proper validation step, using a predefined acceptance criteria. Then, the outer cycle updates the state temperature based on a predefined cooling schedule. The considered length of the Markov chain (M) was set to the number of nodes m.

Algorithm 1 SA implementation of the FTP.

```
generateInitialSolution()
while termination condition not met do
  for i = 0 to i = M do
    generateLocalNeighbourhoodSolution()
    applyAcceptanceCriteria()
  end for
  updateTemperature()
end while
```

Procedure generateLocalNeighbourhoodSolution

```
Input: sol_cur - a solution to the FTP, v_0 - departure city;
  v_{n+1} - arrival city; D - nights in each destination;
Output: sol_neigh - neighbourhood solution
origin ← v_0
time ← random(T_0)
sol_neigh ← []
path_cur ← extractPathFromSolution(sol_cur)
path_neigh ← 2-opt(path_curr)
for destination in path_neigh do
  sol_neigh.append((time, origin, destination))
  time = time + nights[destination]
  origin ← destination
end for
destination ← n_f
sol_neigh.append((time, origin, destination))
return sol_neigh
```

The neighborhood function that was used for the generation of new candidate solutions is the 2-opt swap procedure. Hence, at each iteration step of the Markov chain, it selects two random nodes and swaps the corresponding path. Since this swapping procedure may change the dates at which each node is visited, it is necessary to adjust the dates and calculate the weight of the new solution.

The used acceptance criteria is the Metropolis criteria (Metropolis, Rosenbluth, Rosenbluth, Teller, & Teller, 1953), presented in Eq. (4). This criterion dictates that: (i) if a candidate solution y is *better* than the current solution x, it is always accepted; (ii) if the solution is worse, it may, or may not be accepted. The probability (p) by which a worse solution is accepted depends upon: a) the difference in the objective function values Δ_f of the two solutions; b) the current temperature of the system. As Δ_f increases, and as the temperature decreases, the probability of accepting a worse solution is reduced. With such an approach, the Metropolis acceptance criteria allows up-hill moves, which allows the algorithm to escape from local minimum. Notwithstanding, as the temperature reaches very low values, the algorithm becomes increasingly greedy.

$$p = \begin{cases} 1, & \text{if } f(y) \leq f(x), \\ e^{-\frac{f(y)-f(x)}{t}}, & \text{otherwise} \end{cases} \tag{4}$$

The developed SA optimization uses a geometric cooling schedule. It starts with an initial temperature t_0, and at each outer iteration, the temperature is decreased, using Eq. (5), where k is the iteration counter of the outer loop and λ is the cooling parameter.

$$t_{k+1} = \lambda * t_k \tag{5}$$

The t_0, t_f and λ terms must be calculated beforehand based on the probability of accepting a worse solution during the first iteration (p_0) and during the last iteration (p_f), and on the total number of outer iterations (k). The defined algorithm establishes p_0 as 0.98 and p_f as a positive close to zero value. The total number of iterations is set according to the time available for the optimization process.

To calculate the value of t_0 and t_f, the algorithm starts by generating some candidate solutions using the neighborhood function and the current solution x (Wang, Lin, Zhong, & Zhang, 2015). These candidate solutions are used to calculate the average absolute difference in the objective function Δ_{avg}. This allows the calculation of the t_0 value according to Eq. (6), based on the Metropolis criteria. The final temperature t_f is given by $t_f = \lambda^k t_0$. This allows the calculation of λ with Eq. (7). Given t_0, t_f and λ, the geometric cooling schedule is completely defined.

$$t_0 = \frac{-\Delta_{avg}}{ln(p_0)} \tag{6}$$

$$\lambda = \left(\frac{-\Delta_{avg}}{ln(p_f)t_0}\right)^{1/k} \tag{7}$$

4.2. Ant colony optimization

Similarly to the SA, the developed ACO algorithm receives, as input, a weight matrix with the information concerning the solution components of the problem. It must also receive other relevant parameters for the solution construction process, as the initial and final node and the set of waiting periods D – see Algorithm 2.

The initialization of the ACO metaheuristic requires the construction of an initial pheromone matrix. The initial pheromone value is set according to Eq. (8), where n is the number of nodes and C^{nn} is the cost of the nearest neighbor heuristic.

$$\tau_{ij}^t = \tau_0 = \frac{1}{nC^{nn}} \tag{8}$$

The initialization of the metaheuristic also requires the definition of a variety of algorithm-specific parameters, such as the number of ants m, the pheromone evaporation rate ρ, the heuristic relative influence β, the pheromone relative influence α, and the exploration rate Q_0.

After the initialization, and until the termination condition is met, the algorithm enters into an iterative cycle, where every ant belonging to the colony constructs a solution to the problem. This is followed by a pheromone update phase, to reflect the colony search experience. A new iteration may only start after all ants

Algorithm 2 ACO implementation of the FTP.

Input: G - weight matrix, v_0 - departure city; v_{n+1} - arrival city; V - array of cities to visit; D - nights in each destination; m - number of ants;
initPheromone()
while termination condition not met **do**
  for all ants **do**
    antProcedure()
  end for
  updateGlobalBest()
  updatePheromoneMatrix()
end while

Procedure antProcedure

Input: v_0 - departure city; v_{n+1} - arrival city; V - array of cities to visit; D - nights in each destination;
Output: solution - an array of triplets (time, origin, destination)
$time \leftarrow$ random(T_0), $origin \leftarrow n_0$
$cities\_to\_visit \leftarrow V$, $nights \leftarrow D$
$solution \leftarrow []$
while ctv not empty **do**
  if random(0, 1) $\leq Q_0$ **then**
    $destination \leftarrow$ exploitation()
  else
    $destination \leftarrow$ exploration()
  end if
  $solution.append((time, origin, destination))$
  $ctv.remove(destination)$
  $time = time + nights[destination]$
  $origin = destination$
end while
$destination \leftarrow n_f$
$solution.append((time, origin, destination))$

have finished the solution construction process and the pheromone matrix has been updated.

The construction process that is undertaken by each ant is as follows. First, the current time is set to a value belonging to the allowable trip starting dates, $t \in T_0$, and the current node is set to the start node v_0. Each ant enters an iterative cycle until all nodes belonging to V have been visited. At every step of this cycle, an ant chooses a solution component by either *exploiting* or *exploring* the search space. The decision of exploiting or exploring depends on the algorithm parameter Q_0 and on a pseudo-random value q, calculated at runtime. The selection of the solution component j, which identifies the next city to be visited, is given by Eq. (9). After the selection of each solution component, it is necessary to update the time, incrementing it by the duration relative to the selected city.

$$j = \begin{cases} exploitation\ (\text{Eq. } 10), & \text{if } q \leq Q_0 \\ exploration\ (\text{Eq. } 11), & \text{otherwise} \end{cases} \tag{9}$$

The *exploitation* of the search space utilizes the random-proportional rule, defined by Eq. (10), which determines the next solution component of the ants' solution. The $J_k(i,\ t)$ term represents the set of solution components that might be selected to form a valid solution component by an ant in its current *state*, where the state refers to the current ant position of the trip it has constructed so far.

$$arg\,max_{j \in J_k(i,t)}[\tau(i,j,t)][\eta(i,j,t)]^{\beta} \tag{10}$$

On the other hand, the *exploration* is given by Eq. (11), with $p_a(i, j, t)$ representing the probability of ant a (which is currently at node i at time t) selects j as the next node to visit. In the presented equations, η is the inverse of the weight matrix value.

$$p_a(i,j,t) = \begin{cases} \dfrac{[\tau(i,j,t)][\eta(i,j,t)]^{\beta}}{\sum_{u \in J_k(i,t)}[\tau(i,u,t)][\eta(i,u,t)]^{\beta}}, & \text{if } j \in J_k(i,t) \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

By following an iterative construction procedure, an incomplete (but valid) solution is found. To complete this solution, it is necessary to add an extra solution component, which closes the route by adding the return node, v_{n+1}.

After each ant finishes its iterative solution construction process, the ACO metaheuristic enters into its pheromone update step. Depending on the chosen ACO algorithm, the pheromone update may vary. This work follows the Ant Colony System (ACS) strategy, whose pheromone global update requires both a deposit and an evaporation step. Unlike many other ACO algorithms, the pheromone update applies only to the arcs belonging to the best solution found so far, S_{bs}. Furthermore, it is also necessary to apply a local pheromone update (see Eq. (12)), after the selection of each solution component, as to reduce the probability of other ants selecting the same one in the current iteration (Dorigo & Gambardella, 1997). This results in an update of the pheromone values, by means of Eq. (13), where $(\Delta\tau_{ij}^t)^{bs}$ is given by $1/C^{bs}$, where C^{bs} represents the objective function value of the best solution.

$$\tau_{ij}^t = (1-\rho)\tau_{ij}^t + \rho\tau_0 \tag{12}$$

$$\tau_{ij}^t = (1-\rho)\tau_{ij}^t + \rho(\Delta\tau_{ij}^t)^{bs} \tag{13}$$

ACO algorithms are often combined with local search heuristics that try to improve the quality of the ants' solutions, after each iteration. However, this was not considered in the implemented optimizer, due to the nonexistence of adequate local search procedures for the time-dependent TSP. In fact, even the k-opt exchange procedures, widely used in the classical TSP as local search, are not efficient for the time-dependent TSP because it requires, at each step, the computation of the entire trip cost, as opposed to just the cost difference regarding the k arcs, as in the symmetric TSP.

4.3. Particle swarm optimization

The implemented PSO algorithm receives, as input, the weight matrix of the corresponding FTP instance, as well as any other attributes relevant to the description of the problem, such as the initial and final city, respectively v_0 and v_{n+1}, the list of cities to visit V, the start window T_0 and the list of number of nights in each destination D. This is followed by the initialization of a swarm of particles, whose size (M) is equal to the number of cities to visit ($|V|$).

To initialize the swarm of particles, it is necessary to set the position and velocity of each particle. These values are initialized with a randomly generated solution, by using the same process as used and described in Section 4.1. Likewise, the initial velocity is randomly generated, by using the *velocityCalculate* procedure described in Algorithm 3. Having all positions initialized, the global best solution (P_{gb}) is updated, and each particle registers its current position as the particle's local best solution (P_{lb}).

At this point, the PSO algorithm enters an iterative cycle until the termination condition is met. At each iteration step (Δ_t), every particle generates a new solution. This particle movement only depends on its current position and velocity, as defined in Eq. (14).

$$X_c^{t+1} = X_c^t + V_c^t \cdot \Delta_t \tag{14}$$

Algorithm 3 PSO implementation of the FTP.

```
swarm ← []
p_gb ← None, f_gb ← inf
for i=1 to i=M do
  swarm[i] ← initParticle()
end for
updateGlobalBest()
while termination condition not met do
  for i=1 to i=M do
    pos_next ← positionUpdate(particle)
    vel_next ← velocityUpdate(particle)
    obj_next ← calculateObjective(pos_next)
    updateParticleInfo(pos_next, vel_next, obj_next)
  end for
  updateGlobalBest()
end while
```

Procedure positionUpdate

```
Input: position - the particle's current position; velocity - the list of swap operators;
Output: the next particle's position
next_pos ← copy(position)
for swap_op in velocity do
  (i, j) ← swap_op
  x_i, x_j = next_pos[i], next_pos[j]
  next_pos[i], next_pos[j] = x_j, x_i
end for
return next_pos
```

Procedure velocityCalculate

```
Input: pos_current, pos_wanted
Output: velocity - the list of swap operators that when applied to pos_current yields pos_wanted
vel ← []
while True do
  pos_next ← positionUpdate(pos_current, vel)
  if pos_next equals pos_wanted then
    return velocity
  end if
  for index_curr = 1 to index_curr = length(pos_next) do
    if pos_next[index_curr] not equals pos_next[index_curr] then
      city_curr ← pos_next[index_curr]
      index_wanted ← pos_wanted.index(city_curr)
      swap_op ← (index_curr, index_wanted)
      vel.append(swap_op)
      break out for cycle
    end if
  end for
end while
```

It must be noted that the *velocity* of a particle corresponds to a *swap sequence* - a list of swap operators - that transforms the path of one solution into another, as proposed and described in the work of Wan, Huang, Zhou, and PhG (2003). This method is summarized in the *positionUpdate* procedure described in Algorithm 3.

Having calculated the next position (X_c^{t+1}), it is necessary to calculate the particle's next velocity (V_c^{t+1}), by applying Eq. (15). Note that the difference operation between two positions (e.g. $P_{lb} - X_c^{t+1}$) uses the method described in the *velocityCalculate* procedure of Algorithm 3 to calculate a swap sequence. When calculating the next velocity, α and β represent the relative influence of the particle's own best solution and the swarm's global best, respectively, which determine the probability of keeping each swap operator of the swap sequence.

$$V_c^{t+1} = V_c^t \oplus \alpha * (P_{lb} - X_c^{t+1}) \oplus \beta * (P_{gb} - X_c^{t+1}) \tag{15}$$

Every iteration cycle (which starts with the update of the particle's position and velocity) is finished by calculating the objective function of each particle and by updating the information of its local best solution. Finally, the global best solution of the swarm is updated, and the iteration cycle restarts, until the termination condition determines the end of the PSO algorithm.

5. System prototype

Due to the NP-hardness nature of the FTP for unconstrained multi-city requests, the associated optimization procedure tends to be computationally heavy. As a consequence, the developed system prototype consists of a distributed web service composed of a server-side application that satisfies the requests received from the clients (see Fig. 2).

5.1. Client-side application

The Client-Side Application (CSA) is responsible for the user interaction, allowing him to define the requested trip (number 1 in the figure), by supplying the following set of parameters:

- The start and return cities, v_0 and v_{n+1};
- A list of cities to visit V;
- The waiting periods (D) associated to each city in V;
- The start time/period (T_0) of the trip.

The response to a user request is not directly processed by the CSA, but rather by the Server-Side Application (SSA), which uses the issued parameters to produce a solution that minimizes the considered cost function. Such a solution contains (at least) one set of flights that satisfy the user-defined request. Nonetheless, several other valid solutions may also be provided, to allow the user to choose the one that is more adequate to his needs. The most relevant information about each of the suggested flights is also provided (number 6), including:

- The flight cost;
- The flight duration;
- The date, departure and arrival time;
- A hyperlink to a third-party API that allows a direct booking of the flight.

The developed CSA was implemented using the React JavaScript library (www.reactjs.org) and Redux framework (www.reduxframework.com), and the communication between the client and the server applications is done via Asynchronous JavaScript and XML (AJAX), which means that upon submitting the request (number 2), the user may continue interacting with the application until the SSA returns a response to the CSA (number 5).

5.2. Server-side application

The SSA was implemented in a Python environment, together with Django framework (www.djangoproject.com). It comprises the following two main components: (i) a Data Management module, to fetch the flights data; and (ii) an Optimization module, which implements the devised search algorithms that find the best set of flights that satisfy the user request.

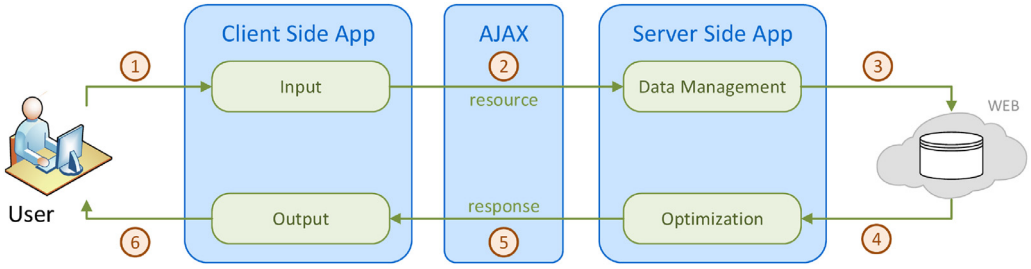


Fig. 2. Structure and data flow of the developed prototype.

5.2.1. Data management module

Since the issued user request only specifies an unordered list of trip nodes (i.e., the set of cities to be visited), the Data Management module is responsible for collecting all the flights information that is required to execute the devised optimization algorithms, thus completing the arcs (flights) that connect those nodes. Hence, an arc connecting two nodes corresponds to a flight between two cities, at a specific date. However, there are many flights that fit this description and each one may have several attributes that differentiate it from the others. For example, every flight has a particular cost, duration, departure and arrival time, airline company, bag limit or even different number of layover flights.

Due to the vast number of attributes that define every flight, it is impossible to know which particular flight is the most adequate for a specific user, because users often have different selection criteria. Hence, upon the construction of the multipartite graph, it makes sense to have a list of possible flights for every arc in A, instead of just a single one. This allows the selection of a specific flight according to the objective function being minimized. For example, if the goal is to minimize the total flight cost, it makes sense to select those flights that present the lowest cost, disregarding other attributes such as the airline company of the flights duration. This means that upon referring to an arc connecting two nodes, the Data Management module is actually considering a *family* of arcs that share key characteristics (such as the origin, destination and date), but which may vary regarding other attributes.

To collect the data corresponding to each arc belonging to A, the developed system communicates with a third-party API and sends a request seeking the required flight objects. Among the several free and public flight-data APIs that might be used, it was adopted the API provided by the *Kiwi* flight search company (docs.kiwi.com). Among the several useful features that are provided by this API is its ability to respond to a query over an extended search period. That is, upon requesting a flight between two cities, it is possible to specify a time window, instead of a single date. This is particularly useful because it allows the reduction of the total number of requests. Hence, while the total number of arcs in an FTP instance might be considerable, there is no need to make an individual request for every single arc. Instead, it is possible to submit a request for every pair of cities, by extending the search period to reflect the total number of necessary layers.

Having defined a complete graph, it is possible to run the optimization module (number 4), in order to produce a valid solution to the user request.

5.2.2. Optimization module

The implemented optimizer was developed by using a strictly modular approach, allowing the integration of several different optimization strategies and algorithms. In particular, the three considered metaheuristic algorithms (SA, ACO, and PSO) were implemented in the evaluated prototype, by using conventional and straightforward implementations and parameterizations.

Table 1
Algorithm specific parameters.

Alg.	Parameter	Value
SA	First iter. acceptance prob. (p_0)	0.98
	Last iter. acceptance prob. (p_f)	10^{-300}
	Initial temperature (t_0)	see Eq. (6)
	Final temperature (t_f)	see Eq. (5)
	Cooling parameter (λ)	see Eq. (7)
	Markov chain length (M)	N
ACO	Pheromone relative influence (α)	1
	Heuristic relative influence (β)	5
	Pheromone evaporation rate (ρ)	0.1
	Exploration rate (Q_0)	0.9
	Number of ants (m)	10
PSO	Swarm size	N
	Particle best solution relative influence (α)	0.9
	Global best solution relative influence (β)	0.9

All these algorithms require an initialization phase, which, among other things, defines some algorithm specific parameters. In order to inspect the same number of solutions in any given iteration of the algorithms, the total number of cities to visit in V was assigned to the Markov chain length (M) in the SA, to the number of ants (m) in the ACO, and to the swarm size (N) in the PSO. Table 1 summarizes the set of parameters used by the two metaheuristic algorithms of the implemented prototype.

6. Experimental results

In order to validate and evaluate the performance of the proposed system, several tests were developed and executed. First, the overall utility of the implemented system was evaluated, by performing a series of tests on the Flying Tourist Problem. In particular, the quality of the obtained solutions was compared with those provided by a *metric Nearest Neighbor* (mNN) heuristic, which promotes the nodes' proximity to define the traveling route (this straightforward approach closely approximates the strategy usually followed by a human solver). Then, a thorough comparison with a state-of-the-art alternative for the devised FTP was performed by considering a comprehensive set of real-world multi-city formulations using different objective functions.

These experiments were executed on a 2.6GHz Intel i7-6700 CPU, with 8GB of RAM, and all the code was developed using the Python3 programming language.

6.1. Flying tourist problem evaluation

To demonstrate and quantify the actual benefits of the proposed system, a series of FTP instances were defined, ranging from just 1 city to visit (which corresponds to a round-way flight), up to a total of 20 cities (see Table 2). For each problem instance, three different solutions were determined based on the following three optimization goals:

Table 2
Comparison of different Flying Tourist Problem solutions obtained with distinct algorithms and optimization goals.

	# Cities	1			2			3			4			5			6			7		
Optimization Goal	Optimization Method	Optimization Result																				
		Price	Duration	Balanced Cost	Price	Duration	Balanced Cost	Price	Duration	Balanced Cost	Price	Duration	Balanced Cost	Price	Duration	Balanced Cost	Price	Duration	Balanced Cost	Price	Duration	Balanced Cost
Price	mNN	438	2742	1128	722	5469	2145	952	7728	2982	1126	8685	3391	1297	8948	3589	1311	9549	3779	1550	10871	4343
	rNN	438	2742	1128	627	5205	1999	856	6418	2523	1016	7617	2994	1232	8152	3305	1266	8194	3341	1562	10419	4216
	SA	438	2742	1128	624	4753	1861	865	6517	2559	1169	7524	3073	1293	9703	3840	1417	9581	3864	1851	12712	5105
	ACO	438	2742	1128	627	5205	1999	873	6530	2568	1048	7963	3120	1218	8041	3262	1347	8959	3627	1562	10419	4216
	PSO	438	2742	1128	597	4440	1749	831	6295	2469	960	7230	2839	1114	7889	3144	1232	9215	3623	1339	10466	4073
	Best Method	mNN			PSO			PSO			PSO			PSO			PSO			PSO		
	Solution	438	2742	1128	597	4440	1749	831	6295	2469	960	7230	2839	1114	7889	3144	1232	9215	3623	1339	10466	4073
	Gain	0.0%	0.0%	0.0%	21.0%	23.2%	22.6%	14.5%	22.8%	20.8%	17.2%	20.1%	19.4%	16.4%	13.4%	14.1%	6.4%	3.6%	4.3%	15.8%	3.9%	6.6%
Duration	mNN	597	1602	898	811	2955	1453	1129	4181	2042	1335	4813	2376	1506	5414	2675	1561	5427	2717	1840	6547	3248
	rNN	597	1602	898	766	2450	1271	1027	3919	1895	1198	4276	2122	1475	4600	2413	1520	4815	2509	1846	5716	3007
	SA	597	1602	898	726	2617	1293	1053	4142	1979	1371	5039	2472	1600	5387	2736	1789	5939	3035	2413	8437	4220
	ACO	597	1602	898	766	2450	1271	1030	3838	1873	1186	4287	2116	1457	4570	2391	1629	5090	2667	1846	5716	3007
	PSO	597	1602	898	769	2309	1231	1002	3204	1663	1216	3502	1902	1444	3975	2203	1527	4089	2296	1719	4589	2580
	Best Method		mNN			PSO			PSO			PSO			PSO			PSO			PSO	
	Solution	597	1602	898	769	2309	1231	1002	3204	1663	1216	3502	1902	1444	3975	2203	1527	4089	2296	1719	4589	2580
	Gain	0.0%	0.0%	0.0%	5.5%	28.0%	18.0%	12.7%	30.5%	22.8%	9.8%	37.4%	24.9%	4.3%	36.2%	21.4%	2.2%	32.7%	18.4%	7.1%	42.7%	25.9%
Balanced Cost	mNN	467	1653	822	787	2966	1439	1070	4247	2021	1274	4844	2342	1447	5471	2651	1512	5472	2696	1723	6634	3192
	rNN	467	1653	822	708	2496	1245	1027	3919	1895	1130	4426	2219	1360	4989	2450	1467	4778	2453	1677	6041	2986
	SA	467	1653	822	700	2597	1269	1052	4170	1988	1233	5274	2445	1413	5317	2584	1784	6703	3260	2208	8111	3978
	ACO	467	1653	822	708	2496	1245	1007	3860	1863	1122	4429	2114	1342	4922	2416	1500	4930	2529	1677	6041	2986
	PSO	467	1653	822	696	2363	1196	974	3254	1658	1056	3590	1816	1257	4040	2092	1389	4255	2249	1539	4795	2516
	Best Method			mNN			PSO			PSO			PSO			PSO			PSO				PSO
	Solution	467	1653	822	696	2363	1196	974	3254	1658	1056	3590	1816	1257	4040	2092	1389	4255	2249	1539	4795	2516
	Gain	0.0%	0.0%	0.0%	13.0%	25.5%	20.3%	9.9%	30.5%	21.9%	20.7%	35.0%	29.0%	15.1%	35.4%	26.7%	8.8%	28.6%	19.9%	12.0%	38.4%	26.9%

	# Cities	8			9			10			12			15			17			20		
Optimization Goal	Optimization Method	Optimization Result																				
		Price	Duration	Balanced Cost	Price	Duration	Balanced Cost	Price	Duration	Balanced Cost	Price	Duration	Balanced Cost	Price	Duration	Balanced Cost	Price	Duration	Balanced Cost	Price	Duration	Balanced Cost
Price	mNN	1835	11443	4713	1715	11658	4693	2070	13684	5549	2446	16215	6570	2422	16099	6517	3303	17755	7632	3029	21460	8548
	rNN	1641	11102	4476	1700	12058	4803	1828	11976	4868	2027	13240	5385	2473	16216	6589	2827	19243	7743	3018	23891	9280
	SA	2051	14872	5894	2067	15571	6114	2452	18019	7117	2861	18467	7536	3629	26553	10498	4103	23209	9780	5633	40056	15960
	ACO	1609	11198	4482	1714	12453	4931	1967	13058	5290	2027	13240	5385	2473	16216	6589	3247	22332	8965	2978	24519	9440
	PSO	1423	10714	4206	1441	11206	4366	1547	11207	4440	1809	13994	5458	2252	16471	6510	2788	20490	8091	2651	24370	9167
	Best Method	PSO			PSO			PSO			PSO			PSO			PSO			PSO		
	Solution	1423	10714	4206	1441	11206	4366	1547	11207	4440	1809	13994	5458	2252	16471	6510	2788	20490	8091	2651	24370	9167
	Gain	29.0%	6.8%	12.1%	19.0%	4.0%	7.5%	33.8%	22.1%	25.0%	35.2%	15.9%	20.4%	7.5%	-2.3%	0.1%	18.5%	-13.4%	-5.7%	14.3%	-11.9%	-6.7%
Duration	mNN	2119	6729	3497	2026	6768	3444	2398	8060	4092	2754	8592	4499	2740	9165	4660	5435	13402	7825	4633	13384	7259
	rNN	2009	5744	3130	2065	6075	3638	2185	6508	3482	2379	6964	3754	2780	6897	4022	5367	9480	6600	4720	14065	7524
	SA	2252	8036	3987	2388	8720	4287	3083	10881	5423	3324	12570	6097	3671	14203	6831	6422	15089	9022	5641	16490	8896
	ACO	2009	5744	3130	2095	6405	3388	2195	6675	3539	2579	8311	4299	3024	8345	4621	5435	13401	7825	3803	12679	6465
	PSO	2017	4609	2795	1920	4669	2744	2098	4990	2965	2343	5681	3344	2890	7212	4187	3705	10599	5764	3655	11689	6062
	Best Method		PSO			PSO			PSO			PSO			rNN			rNN			PSO	
	Solution	2017	4609	2795	1920	4669	2744	2098	4990	2965	2343	5681	3344	2780	6897	4022	5367	9480	6600	3655	11689	6062
	Gain	5.0%	46.0%	25.1%	5.5%	45.0%	25.5%	14.3%	61.5%	38.0%	17.6%	51.2%	34.5%	-1.4%	32.9%	15.9%	1.3%	41.4%	18.6%	26.8%	14.5%	19.7%
Balanced Cost	mNN	1993	6821	3437	1865	6912	3374	2296	8156	4049	2655	8642	4451	2635	9305	4636	3531	10783	5707	3334	11932	5913
	rNN	1878	5858	3072	1934	6313	3258	2050	6859	3493	2142	7024	3607	2666	8479	4410	3333	10995	5631	3646	10075	5574
	SA	2235	8356	4071	2463	9035	4434	3033	12732	5942	3417	11386	5807	4751	18280	8809	5410	18501	9202	6190	20499	10483
	ACO	1878	5858	3072	2010	6811	3450	2130	7531	3750	2143	7024	3607	2921	9943	5028	3815	13918	6845	3281	12081	5921
	PSO	1658	4771	2592	1728	4852	2665	1840	5171	2839	2090	5724	3180	2599	6776	3852	3282	10046	5311	2853	14027	6205
	Best Method			PSO			PSO			PSO			PSO			PSO			PSO				rNN
	Solution	1658	4771	2592	1728	4852	2665	1840	5171	2839	2090	5724	3180	2599	6776	3852	3282	10046	5311	3646	10075	5574
	Gain	20.2%	43.0%	32.6%	8.0%	42.5%	26.6%	24.8%	57.7%	42.6%	27.1%	51.0%	40.0%	1.4%	37.3%	20.4%	7.6%	7.3%	7.4%	-8.6%	18.4%	6.1%

- Total *price* of the trip;
- Total *duration* of the trip;
- A *balanced cost*, corresponding to a weighted sum between the former two, where the price and flight duration contribute with 70% and 30%, respectively.

These three solutions were obtained by individually considering the following optimization approaches:

- *metric Nearest-Neighbor* (mNN) heuristic;
- *regular Nearest-Neighbor* (rNN) heuristic;
- *Simulated Annealing* (SA) metaheuristic;
- *Ant Colony Optimization* (ACO) metaheuristic;
- *Particle Swarm Optimization* (PSO) metaheuristic.

In this experiment, each request was run 20 times, in order to obtain a representative averaged solution. Moreover, to obtain a system response time compatible with a real-time user experience, a maximum optimization time of 1 second was allowed to each execution. In all the considered cases, the trip starts and returns to the same randomly chosen city, and visits a given set of cities, randomly chosen from the following set: Abuja, Atlanta, Barcelona, Beijing, Cairo, Casablanca, Dubai, Dublin, Frankfurt, Hong Kong, Istanbul, Johannesburg, Kiev, Los Angeles, Madrid, Miami, Moscow, New-York (JFK), Oslo, San Francisco, Sidney, Singapore. The start date was set to be the same for all requests (*1 May 2019*), which, upon the execution of the tests, was 50 days into the future. The waiting period on each city was set to a random value between 1 and 5 nights.

6.1.1. Quantitative evaluation and improvement

The result of the execution of the described evaluation is presented in Table 2, where the solutions for the three considered optimization goals (total flight cost, total flight duration and balanced cost) are presented, as a function of the adopted optimization approaches and the number of cities of the trip.

By analyzing the obtained values, it can be observed that significant gains are obtained for each considered optimization metric when the system is configured to use that same metric as the optimization goal. Moreover, it can also be seen that such optimization tends to also favor the other optimization metrics (except in some setups with 15, 17 and 20 cities), resulting in solutions with considerable benefits in what concerns both the total flight price and duration.

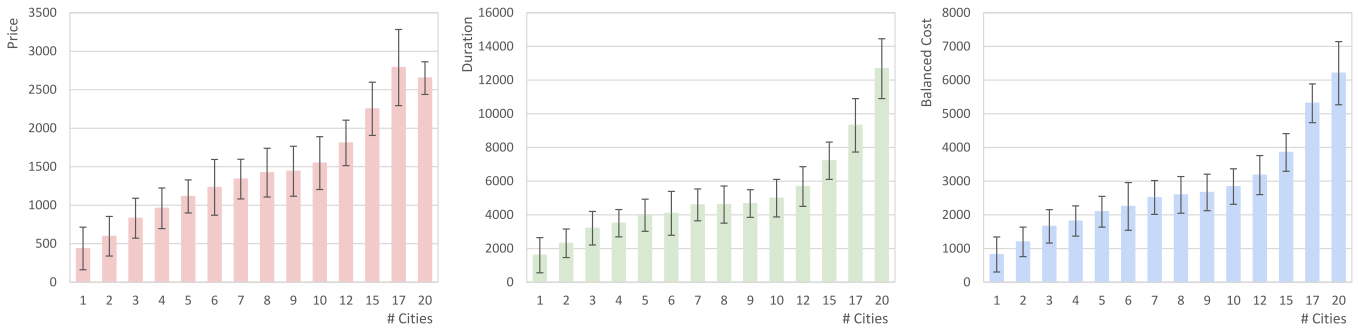
Although remarkable results are obtained with all the considered metaheuristics, it is also observed that the PSO tends to provide slightly higher gains than the other optimization methods for the defined FTP formulation. However, since this comparison covers averaged results, this observation does not imply that PSO always conducts to the best optimization. Sometimes, the ACO algorithm provides better values than PSO. Fig. 3 presents a brief statistical characterization of the results obtained with the PSO metaheuristic, depicting the relation between the mean value of the obtained gains (for each trip configuration) and the corresponding standard deviation, by using lines to illustrate one standard deviation from the average. On normal distributions, this range contains around 68% of all possible values. In the obtained results, the ACO average generally falls within this range. For example, for the price objective of 15 cities in Table 2, the ACO algorithm obtains an average cost of 2473 and this value is well within the corresponding lines for the 15 cities bar in Fig. 3a. Hence, the good performance of PSO is not a guarantee that it always achieves the best objective. Moreover, it should be noted that all these algorithms were limited to a 1 second execution time, in order to keep the systemâs response time within tolerable limits required by a human user using this web-service. If more execution time was allowed, the ACO and SA algorithms would obtain better results. Hence, the good results provided by PSO within this time limit are mostly due to its ability to escape to local minima and not necessarily to its ability to converge to the global minimum. Keeping these caveats in mind, it was decided to use PSO as the chosen metaheuristic in the following discussion.

The chart and data presented in Fig. 4 summarize the gains (price, duration and balanced cost) that are obtained using the PSO metaheuristic, by using the *metric nearest-neighbor* solutions as reference. The first insight into these results allows a preliminary evaluation of the utility of the proposed system. In fact, it can be seen that whenever the trip visits more than one city, the considered optimization system greatly improves the total flights cost (≈6–35%) and duration (≈15–60%).

6.1.2. Balancing the total flight price and duration

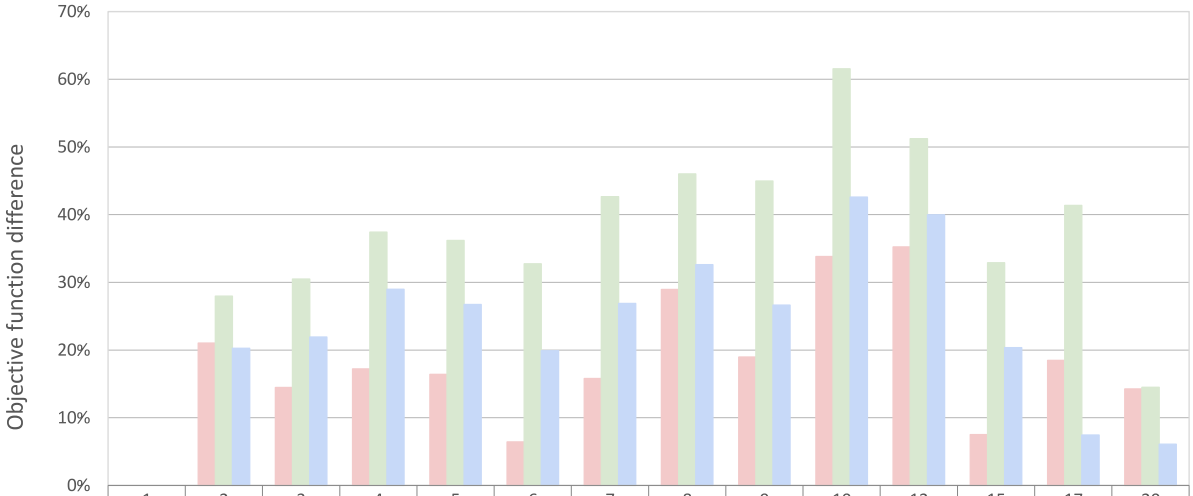
As it was referred before, the considered *Balanced Cost* optimization goal envisages a better compromise between the total price of the trip and the total flight duration, by minimizing a weighted value where the price and flight duration contribute with 70% and 30%, respectively.

As it can be observed in Table 2, such minimization also provides significant gains (when compared to the straightforward *metric Nearest-Neighbor*), although it introduces slight penalizations when compared with the best gains that are obtained when individually considering the total flight price and the total flight duration optimization goals. This happens because the two objective functions (price and duration) can hardly be simultaneously minimized, and thus, a compromise has to be reached. In this case,



(a) Total flight price. (b) Total flight duration. (c) Balanced cost.

Fig. 3. Statistical characterization of the results obtained with the PSO metaheuristic. The colored bars represent the mean value of the observed metric, whereas the error bars represent the corresponding standard deviation.



	1	2	3	4	5	6	7	8	9	10	12	15	17	20
Price	0.0%	21.0%	14.5%	17.2%	16.4%	6.4%	15.8%	29.0%	19.0%	33.8%	35.2%	7.5%	18.5%	14.3%
Duration	0.0%	28.0%	30.5%	37.4%	36.2%	32.7%	42.7%	46.0%	45.0%	61.5%	51.2%	32.9%	41.4%	14.5%
Balanced Cost	0.0%	20.3%	21.9%	29.0%	26.7%	19.9%	26.9%	32.6%	26.6%	42.6%	40.0%	20.4%	7.4%	6.1%

Fig. 4. Average gains provided by the PSO metaheuristic when considering the price, duration and balanced cost optimization goals.

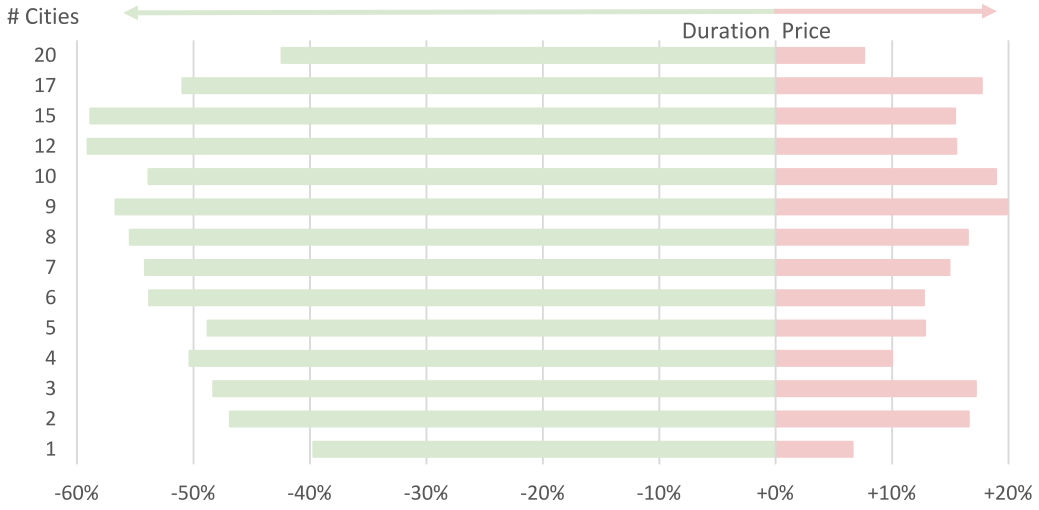


Fig. 5. Variation of the total flight price and duration when minimizing the *Balanced Cost* objective function.

compromising means slightly increasing the price to significantly reduce the duration (or vice-versa).

This compromise between flight price and duration is also illustrated in Fig. 5, which presents the relative duration gain as a consequence of the increase in price. This figure shows that, in general, increasing the price by around 15% leads to a decrease in the flight duration by around 55%, when compared to the price-only metaheuristic.

6.1.3. Impact of the trip start interval

To evaluate the influence of the trip start interval on the obtained results, the same queries and data sets were used to solve these same FTPs using trip start windows of different lengths. The resulting gains (decrease of the total price) are illustrated in Fig. 6, when considering start periods of length 1, 15 and 31 days, respectively. The solution corresponding to the 1-day start window was used as reference.

By analyzing the results presented in Fig. 6, it can be observed that increasing the interval of the start date may lead to even greater benefits, with flight price improvements as high as 24%.

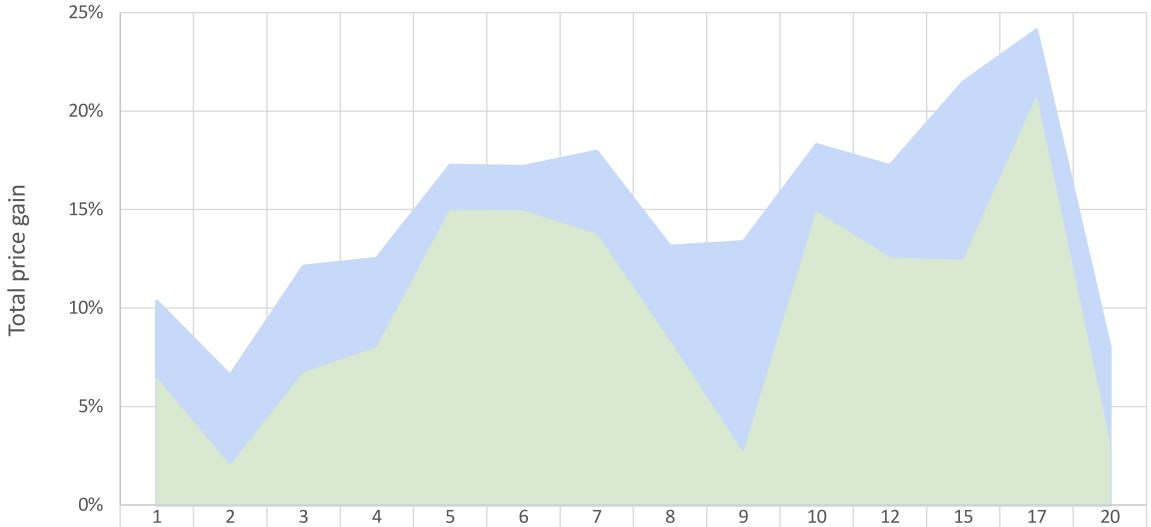
6.1.4. Response time

The response time of any FTP query depends on two different procedures: the data gathering and the optimization. In this case, the optimization time is mostly constant and established *apriori*. In contrast, the data gathering is usually the bottleneck of the process, because the construction of the cost matrix requires massive amounts of data.

The total time that is necessary to respond to a request, as a function of the number of cities and the length of the start interval, is illustrated in Fig. 7. From the analysis of this figure, it can be seen that requests with up to 10 cities can be solved in less than 60 seconds. It can also be seen that the response time increases non-linearly as the number of visited cities increases. On the other hand, increasing the length of the start interval has low influence for small instances (up to 10 cities), but has a significant impact for greater instances.

6.2. Comparison with Kiwi's nomad

At the present time, *Kiwi*'s *Nomad* is the only publicly available (but non-disclosed) tool that is capable of addressing the defined



	1	2	3	4	5	6	7	8	9	10	12	15	17	20
31-day start window	10%	7%	12%	13%	17%	17%	18%	13%	13%	18%	17%	21%	24%	8%
15-day start window	6%	2%	7%	8%	15%	15%	14%	8%	3%	15%	13%	12%	21%	3%

Fig. 6. Price improvement as a function of the trip start interval.

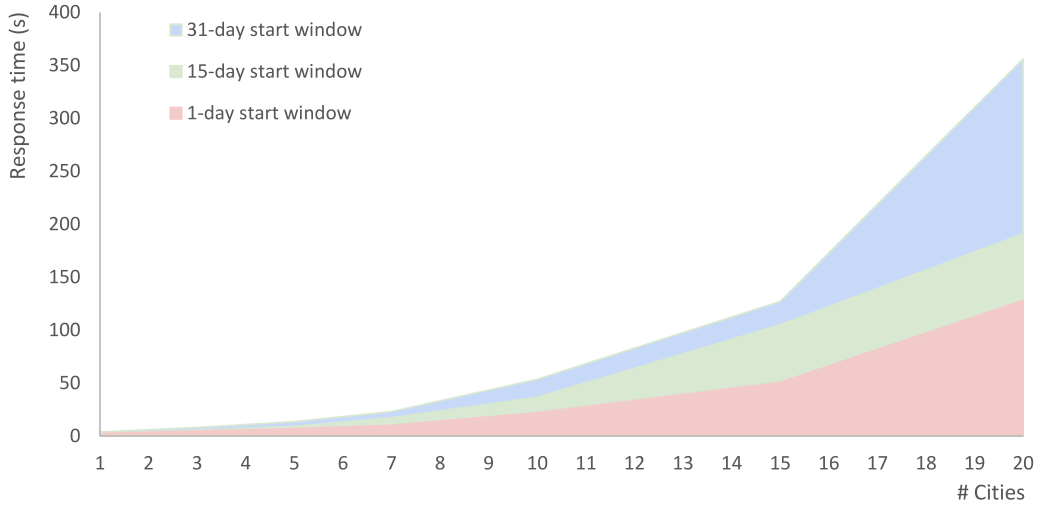


Fig. 7. Total response time to a request, as a function of the number of visited cities and length of start period.

Flying Tourist Problem in the form of an unconstrained multi-city routing problem, although its queries are limited to only 10 different cities. To facilitate the comparison of the conceived optimization system with this tool, the definition of the user requests of the proposed FTP (see Section 3) was kept as similar as possible to *Kiwi*'s *Nomad* interface. The user is asked to specify the departing/arriving city, together with the start date, the set of cities to be visited and the duration of the stay in each city.

The results provided by both applications were extensively compared against each other, according to each considered objective function. The difference in the total flight price and duration (for each query) was also measured and analyzed as a function of the query parameters. The former evaluation will be called *absolute comparison*, while the latter *quantative evaluation*.

The execution of these tests involved over 100 different queries, by varying not only the number of cities (2–10), but also the length of the trip start interval (1–15 days). All queries that were performed on both applications had its start and return city set to Lisbon (Portugal), while each city to be visited belongs to the same set of hub airports that were considered in the previous subsections. These queries were executed during the period between 15 and 16 of June 2018 and the base start date was set to the 1st of August 2018, which, at the time of the tests, was 45 days in the future. The staying period in each city was set to a random value between 1 and 5 days. For extended start periods, the base start date was extended by 31 days.

6.2.1. Absolute comparison

Both applications respond to each query with three different sets of flights, serving the following different optimization criteria: the *cheapest*, the *fastest* and the *recommended*. For each query, a winner was determined according to these criteria. The cheapest set of flights is determined according to the total flight price, while the fastest depends solely on the total flight duration. The recommended set of flights depends on both the price and the duration, and the winner for this criteria must have both lower prices and duration.

Fig. 8 illustrates the obtained comparison, by presenting the total number of times that an application outperformed the other, for each of the three different optimization criteria. It also shows the number of cases in which the responses were very similar.

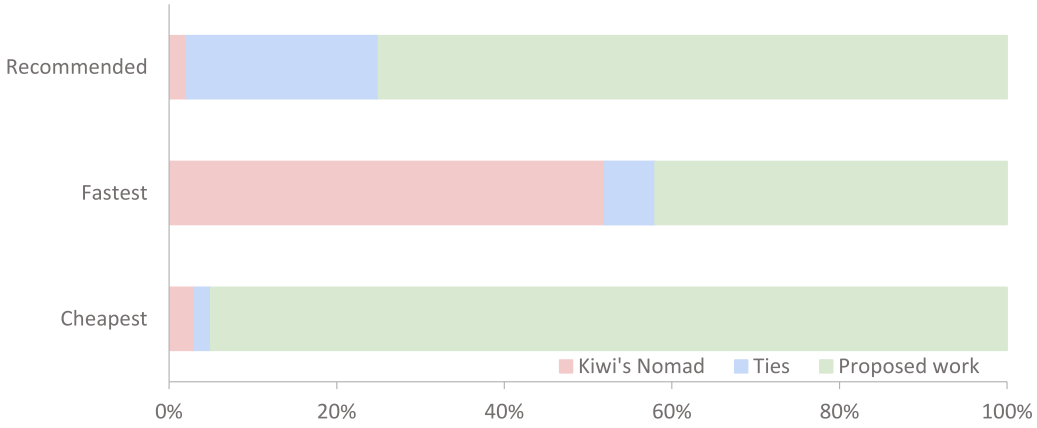


Fig. 8. Comparison of the results provided by the proposed tool and by *Kiwi*'s *Nomad* application.

The analysis of this figure indicates that the developed application presents better solutions for a significant amount of queries. In fact, while the fastest set of flights is only achieved in 42% of the queries, it presents the cheapest set of flights 95% of the times and the best recommended result 75% of the times.

6.2.2. Quantitative evaluation

To evaluate the difference of the responses provided by both applications, the total flight price and duration of the recommended set of flights was also quantitatively measured (see Fig. 9). The values presented in these graphs refer to the developed application response and were normalized using the *Kiwi*'s *Nomad* response as reference.

Fig. 9a presents the results of the queries performed for a single start date. Its analysis shows that, for a small number of nodes (2 and 3), the developed application recommends flights that are slightly more expensive (10 to 19%) than those presented by *Kiwi*. In contrast, the flight duration of these flights is much lower (33-46%). For requests with more nodes (5 to 10), the results presented by the developed application have both lower prices (2-18%) and flight duration (9-24%). Fig. 9b depicts the obtained results when the length of the start interval was extended to 31 days. With such an extended start period, every recommended set of flights provided by the proposed application has a lower price and duration. The price presents the most significant change: the minimum improvement is 8%, while the maximum is 29%.

Finally, it is worth noting that all the presented experiments only consider up to 10 different cities to be visited by the traveler. The reason why more cities were not considered arises not from the developed application (which could easily accommodate more cities), but it is motivated by a strict limit presented by *Kiwi*'s *Nomad* user interface, which does not support more than 10 cities in the planned route.

7. Related work

Before the release of *Kiwi*'s newest flight search service, this travel agency launched a *"Travelling Salesman Challenge"*, which attracted the attention of several researchers. In particular, Duque, Cruz, Cardoso, and Oliveira (2018) considered the *Kiwi* challenge and presented a formulation similar to the one that is now presented in this manuscript. Although their motivation is similar, their formulation is significantly different, as they do not consider a staying duration associated with each city. They consider only variable costs, which, to our knowledge, can not be used to model a duration in a simple way. As such, the problem they consider is less restricted and can be directly solved with a time dependent TSP approach. Interestingly, these authors also applied conventional SA and ACO meta-heuristics. They also use genetic algorithms to optimize the ACO approach and consider a hybrid algorithm that combines ACO and SA.

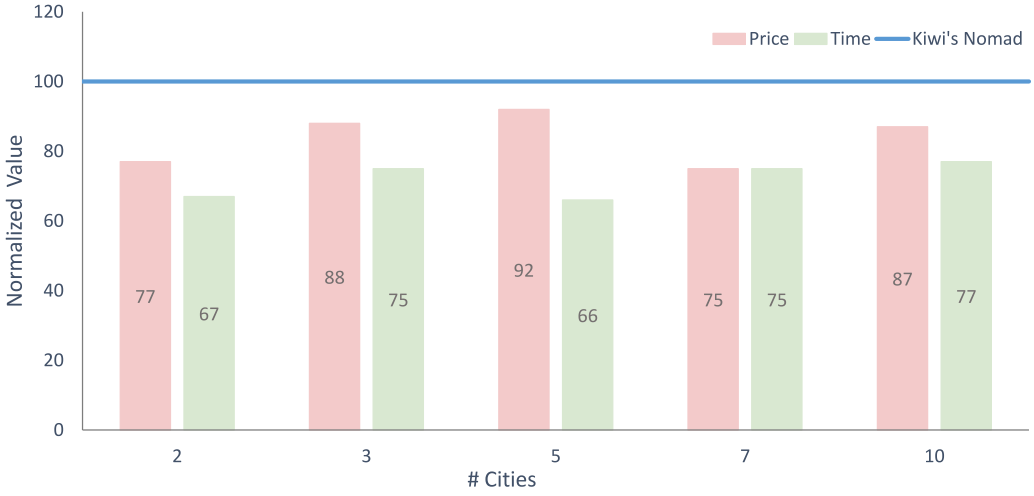
Although a direct comparison with the techniques presented by Duque et al. (2018) is not possible (since the considered problems are somewhat different), it is possible to present some observations about their data set. First, their approach was designed to scale well for a very large number of cities (they consider up to 100 cities). Considering the intrinsic motivation of a touristic passenger, the proposal that is now presented was only tested with smaller datasets, of at most 20 cities. On the other hand, their approach optimizes only the cost of the trip, whereas this proposal also considers the total duration and a balance of these two. When considering only up to 15 cities, their results (Table 1) show that SA obtained better results than ACO and even ACO-SA. This is interesting, as a different result was obtained in this proposal, possibly because of the different nature of the problem that was considered. Moreover, this FTP implementation only allowed 1 second of the solver execution, whereas these authors used 30 seconds on their test.

Another previous approach that is very similar to FTP was proposed by Li, Zhou, and Zhao (2016), where they presented the Travel Itinerary Problem (TIP). TIP also considers duration constraints but their constraints are only lower bounds, meaning that each city must be visited by a certain minimum number of days, but no upper bound is defined. There is a global up-bound that limits the total amount of days for the trip, but there is no limit per city. This differs from the FTP approach, where an upper bound is set in each city but not on the global trip. Their approach is very flexible, as they formulate this problem as a 0–1 integer programming model. These authors only consider optimizing the travel cost, not the duration. The focus of the paper is mostly on formulating the problem and in setting up the information system to process the necessary data. From an algorithmic point of view, their approach does not use any meta-heuristics approach. They refer to an enumeration approach which seems to require exponential time, as the number of cities increases (Table 5).

A time-dependent TSP with time windows was also considered by Montero, Méndez-Díaz, and Miranda-Bront (2017), by using an integer programming formulation and an upper and a lower bound for the duration in each city. However, the focus of this problem is in modeling the flow of transit in a city and therefore it does not consider costs, only trip duration. The authors present an exact algorithm for this formulation that scales up to 40 customers.

Hence, the existence of these alternative TSP formulations, that are somewhat close to the proposed FTP, is a clear evidence of the growing interest on this kind of TSP formulations, highlighting the importance of the presented FTP. Naturally, part of this interest is

(a) Single start date.



(b) Extended start period (31 days).

Fig. 9. Comparison of the recommended flights price and duration, as a function of the number of nodes and the length of start interval. The presented values refer to the proposed application response, and were normalized with respect to *Kiwi*'s *Nomad* response value.

due to the *Kiwi* travel agency. However, the presented work began even before their challenge, as a consequence of the growing amount of airline traveling information that is now available.

Nevertheless, it should be noted that an exact and quantitative comparison of these approaches with the work that was now presented is not a straightforward task, as they all consider slightly different problems. Still, the authors believe that this comparison highlights the relevance of the presented contributions. Just as TIP (Li et al., 2016), the FTP formulation was driven by a concrete problem that can be solved with existing data. This real-world application means that FTP formulation is more elaborated that the other time-dependent TSP formulations, that abstract away part of the problem. Besides this focus on the problem formulation, classical meta-heuristics that allowed the obtention of viable solutions for large problem sizes (20 cities, in only 1 second) were studied and evaluated. These problem sizes are well within the desired application range, which (in general) uses fewer cities.

8. Conclusions

Despite the existence of numerous flight search applications, most of them lack the ability to properly address unconstrained multi-city flight requests, since this problem is generally not tractable. To circumvent this absence, the present work formalizes and addresses the Flying Tourist Problem (FTP), a NP-hard problem that occurs as a generalization of the Traveling Salesman Problem (TSP), and whose goal is to find the best schedule, route, and set of flights, for an unconstrained multi-city flight request.

An effective methodology that allows an efficient resolution of this rather demanding problem was proposed, based on different heuristics and meta-heuristic optimization algorithms, including the Simulated Annealing, the Ant Colony Optimization and the Particle Swarm Optimization, allowing the identification of solutions in real-time, even for large instances. The developed methods were integrated into a web application prototype, allowing a fast resolution of user-defined requests.

The implemented system was evaluated using different criteria, including the provided gains (in terms of total flight price and duration) and its performance compared to other similar systems. The obtained results show that the developed optimization system consistently presents solutions that are up to 35% cheaper (or 60% faster) than those developed by simpler heuristics. Furthermore, when comparing the developed system to the only publicly

available (but not-disclosed) alternative, it was shown that it provides the cheapest and the best-recommended solutions, respectively 95% and 74% of the times.

As a result, upon the planning of a complex multi-city trip, the developed system showed to allow the user to save a significant amount of time and money.

Credit authorship contribution statement

Rafael Marques: Conceptualization, Methodology, Software, Validation, Investigation, Writing - original draft. **Luís Russo:** Methodology, Formal analysis, Writing - review & editing, Supervision. **Nuno Roma:** Conceptualization, Methodology, Resources, Writing - review & editing, Visualization, Supervision, Project administration.

Acknowledgement

This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia under projects UID/CEC/50021/2019 and PTDC/EEI-HAC/30485/2017.

Conflict of interest

The authors have no affiliation with any organization with a direct or indirect financial interest in the subject matter discussed in this manuscript.

References

Applegate, D. L., Bixby, R. E., Chvátal, V., & Cook, W. J. (2007). The traveling salesman problem: A computational study. *Princeton series in applied mathematics*. Princeton, NJ, USA: Princeton University Press.

Bazlamaçci, C. F., & Hindi, K. S. (2001). Minimum-weight spanning tree algorithms a survey and empirical study. *Computers and Operations Research, 28*(8), 767–785. doi:10.1016/S0305-0548(00)00007-1.

den Besten, M., Stützle, T., & Dorigo, M. (2000). Ant colony optimization for the total weighted tardiness problem. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, & H.-P. Schwefel (Eds.), *Parallel problem solving from nature PPSNVI* (pp. 611–620). Berlin, Heidelberg: Springer Berlin Heidelberg.

Boland, N., Hewitt, M., Vu, D. M., & Savelsbergh, M. (2017). Solving the traveling salesman problem with time windows through dynamically generated time-expanded networks. In D. Salvagnin, & M. Lombardi (Eds.), *Integration of AI and OR techniques in constraint programming* (pp. 254–262). Cham: Springer International Publishing.

Chen, S.-M., & Chien, C.-Y. (2011). Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques. *Expert Systems with Applications, 38*(12), 14439–14450. doi:10.1016/j.eswa.2011.04.163.

Clerc, M., & Kennedy, J. (2002). The particle swarm - Explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions of Evolutionary Computation, 6*, 58–73.

Czech, Z. J., & Czarnas, P. (2002). Parallel simulated annealing for the vehicle routing problem with time windows. In *Proceedings 10th Euromicro workshop on parallel, distributed and network-based processing* (pp. 376–383). doi:10.1109/EMPDP.2002.994313.

Czyżżak, P., & Jaszkiewicz, A. (1998). Pareto simulated annealing – a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis, 7*(1), 34–47. doi:10.1002/(SICI)1099-1360(199801)7:1<34::AID-MCDA161)3.0.CO;2-6.

Doerner, K., Gutjahr, J. W., Hartl, R., Strauss, C., & Stummer, C. (2004). Pareto ant colony optimization: a metaheuristic approach to multiobjective portfolio selection. *Annals of Operations Research, 131*, 79–99.

Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation, 1*(1), 53–66. doi:10.1109/4235.585892.

Duque, D., Cruz, J. A., Cardoso, H. L., & Oliveira, E. (2018). Optimizing meta-heuristics for the time-dependent tsp applied to air travels. In H. Yin, D. Camacho, P. Novais, & A. J. Tallón-Ballesteros (Eds.), *Intelligent data engineering and automated learning – ideal 2018* (pp. 730–739). Cham: Springer International Publishing.

Fox, K. R., Gavish, B., & Graves, S. C. (1980). Technical note-an *n*-constraint formulation of the time-dependent traveling salesman problem. *Operations Research, 28*(4), 1018–1021. doi:10.1287/opre.28.4.1018.

Gambardella, L. M., Taillard, E., & Agazzi, G. (1999a). Macs-vrptw: A multiple ant colony system for vehicle routing problems with time windows. In D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli, & K. V. Price (Eds.), *New ideas in optimization* (pp. 63–76). Maidenhead, UK, England: McGraw-Hill Ltd., UK.

Gambardella, L. M., Taillard, É. D., & Dorigo, M. (1999). Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society, 50*(2), 167–176. doi:10.1057/palgrave.jors.2600676.

Glover, F., & Laguna, M. (1999). Tabu search. In D.-Z. Du, & P. M. Pardalos (Eds.), *Handbook of combinatorial optimization: Volume1–3* (pp. 2093–2229). Boston, MA: Springer US. doi:10.1007/978-1-4613-0303-9_33.

Goldbarg, E. F. G., de Souza, G. R., & Goldbarg, M. C. (2006). Particle swarm for the traveling salesman problem. In J. Gottlieb, & G. R. Raidl (Eds.), *Evolutionary computation in combinatorial optimization* (pp. 99–110). Berlin, Heidelberg: Springer Berlin Heidelberg.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning* (1st ed.). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Golden, B., Bodin, L., Doyle, T., & Stewart, W. (1980). Approximate traveling salesman algorithms. *Operations Research, 28*(3), 694–711.

Gross, J. L., Yellen, J., & Zhang, P. (2013). *Handbook of graph theory* (2nd ed.). Chapman & Hall/CRC.

Gülcü, Ş., Mahi, M., Baykan, Ö. K., & Kodaz, H. (2018). A parallel cooperative hybrid method based on ant colony optimization and 3-opt algorithm for solving traveling salesman problem. *Soft Computing, 22*(5), 1669–1685. doi:10.1007/s00500-016-2432-3.

Jianchao, Z., & Zhihua, C. (2006). A new unified model of particle swarm optimization and its theoretical analysis. *Journal of Computer Research and Development, 43*(1), 96.

Johnson, D. S., & McGeoch, L. A. (1997). The traveling salesman problem: A case study in local optimization. In E. H. L. Aarts, & J. K. Lenstra (Eds.), *Local search in combinatorial optimization* (pp. 215–310). Chichester, United Kingdom: John Wiley and Sons.

Jonker, R., & Volgenant, T. (1983). Transforming asymmetric into symmetric traveling salesman problems. *Operations Research Letters, 2*(4), 161–163. doi:10.1016/0167-6377(83)90048-2.

Karp, R. (1972). Reducibility among combinatorial problems. In R. Miller, & J. Thatcher (Eds.), *Complexity of computer computations* (pp. 85–103). New York: Plenum Press.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of IEEE international conference on neural networks: Vol. 4* (pp. 1942–1948).

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95 - international conference on neural networks: 4* (pp. 1942–1948 vol.4). doi:10.1109/ICNN.1995.488968.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671–680. doi:10.1126/science.220.4598.671.

Laporte, G. (1992a). The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research, 59*(2), 231–247. doi:10.1016/0377-2217(92)90138-Y.

Laporte, G. (1992b). The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research, 59*(3), 345–358. doi:10.1016/0377-2217(92)90192-C.

Laporte, G., Gendreau, M., Potvin, J.-Y., & Semet, F. (2000). Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research, 7*(4), 285–300. doi:10.1016/S0969-6016(00)00003-4.

Lawler, E. L., & Wood, D. E. (1966). Branch-and-bound methods: A survey. *Operations Research, 14*(4), 699–719. doi:10.1287/opre.14.4.699.

Li, X., Zhou, J., & Zhao, X. (2016). Travel itinerary problem. *Transportation Research Part B: Methodological, 91*, 332–343. doi:10.1016/j.trb.2016.05.013.

Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research, 21*(2), 498–516. doi:10.1287/opre.21.2.498.

Lopez-Ibanez, M., & Stutzle, T. (2012). The automatic design of multiobjective ant colony optimization algorithms. *IEEE Transactions on Evolutionary Computation, 16*(6), 861–875.

Malek, M., Guruswamy, M., Pandya, M., & Owens, H. (1989). Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. *Annals of Operations Research, 21*(1), 59–84. doi:10.1007/BF02022093.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculations by fast computing machines. *The Journal of Chemical Physics, 21*(6), 1087–1092. doi:10.1063/1.1699114.

Montero, A., Méndez-Díaz, I., & Miranda-Bront, J. J. (2017). An integer programming approach for the time-dependent traveling salesman problem with time windows. *Computers & Operations Research, 88*, 280–289. doi:10.1016/j.cor.2017.06.026.

Morrison, D. R., Jacobson, S. H., Sauppe, J. J., & Sewell, E. C. (2016). Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization, 19*, 79–102. doi:10.1016/j.disopt.2016.01.005.

Ohlmann, J., & Thomas, B. (2007). A compressed-annealing heuristic for the traveling salesman problem with time windows. *INFORMS Journal on Computing, 19*(1), 80–90.

Öncan, T., Altınel, I. K., & Laporte, G. (2009). A comparative analysis of several asymmetric traveling salesman problem formulations. *Computers and Operations Research, 36*(3), 637–654. doi:10.1016/j.cor.2007.11.008.

Osaba, E., Ser, J. D., Sadollah, A., Bilbao, M. N., & Camacho, D. (2018). A discrete water cycle algorithm for solving the symmetric and asymmetric traveling salesman problem. *Applied Soft Computing, 71*, 277–290. doi:10.1016/j.asoc.2018.06.047.

Osaba, E., Yang, X.-S., Diaz, F., Lopez-Garcia, P., & Carballedo, R. (2016). An improved discrete bat algorithm for symmetric and asymmetric traveling salesman problems. *Engineering Applications of Artificial Intelligence, 48*, 59–71. doi:10.1016/j.engappai.2015.10.006.

Osman, I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research, 41*(4), 421–451. doi:10.1007/BF02023004.

Picard, J.-C., & Queyranne, M. (1978). The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations Research, 26*(1), 86–110. doi:10.1287/opre.26.1.86.

Rego, C., Gamboa, D., Glover, F., & Osterman, C. (2011). Traveling salesman problem heuristics: Leading methods, implementations and latest advances. *European Journal of Operational Research, 211*(3), 427–441. doi:10.1016/j.ejor.2010.09.010.

Rosendo, M., & Pozo, A. (2010). A hybrid particle swarm optimization algorithm for combinatorial optimization problems. In *IEEE congress on evolutionary computation* (pp. 1–8). doi:10.1109/CEC.2010.5586178.

Shi, Y., & Eberhart, R. (1998). A modified particle swarm optimizer. In *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (cat. no.98th8360)* (pp. 69–73). doi:10.1109/ICEC.1998.699146.

Taillard, E. D., & Helsgaun, K. (2019). Popmusic for the travelling salesman problem. *European Journal of Operational Research, 272*(2), 420–429. doi:10.1016/j.ejor.2018.06.039.

Veenstra, M., Roodbergen, K. J., Vis, I. F., & Coelho, L. C. (2017). The pickup and delivery traveling salesman problem with handling costs. *European Journal of Operational Research, 257*(1), 118–132. doi:10.1016/j.ejor.2016.07.009.

Wan, K., Huang, L., Zhou, C., & PhG, W. (2003). Particle swarm optimization for traveling salesman problem. In *Proceedings of the 2003 international conference on machine learning and cybernetics (IEEE cat. no.03ex693): 3* (pp. 1583–1585 3). doi:10.1109/ICMLC.2003.1259748.

Wang, C., Lin, M., Zhong, Y., & Zhang, H. (2015). Solving travelling salesman problem using multiagent simulated annealing algorithm with instance-based sampling. *International Journal of Computing Science and Mathematics, 6*(4), 336–353. doi:10.1504/IJCSM.2015.071818.