The Traveller's Problem

Iva Babukova

October 16, 2016

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

1 Problem Formulation

Each instance of TP consists of:

1. A set of airports $A = \{A_0, ..., A_n\}$ for $n > 0$. Each airport $A_i \in A$ represents a location the traveller can begin their commute in, visit as a desired destination, or connect in on the way to their destination. Each airport A_i has a *connection time* C_{A_i}, that is the time that takes to commute to A_i.

2. The trip starts and ends at the same airport A_0, which is referred to as the *home point*.

3. The total travel time T, within which the traveller must have visited all destinations and returned to the home point.

4. A set of flights $F = \{f_0, ..., f_m\}$. Each flight f_j has:

   - departure airport A_j^d,
   - arrival airport A_j^a,
   - date t_j,
   - duration Δ_j,
   - cost c_j,

   for some non-negative integer j less than or equal to n. The date t_j is a positive rational number less than or equal to T that shows at which day f_j leaves its departure airport. The duration Δj is a positive fraction that shows the amount of time that takes for flight f_j to go from A_j^d to A_j^a. The cost c_j is a positive number that denotes the number of units of some currency ϵ that the traveller pays in order to be able to board flight f_j.

5. A set of *destinations* $D = \{D_1, ..., D_l\}$, $D \subseteq A$, $l \leq n$.

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

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

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

Also note that the current problem formulation accepts asymmetric flight costs, i.e. one might have two flights f_i and f_j such that $A^d_i = A^a_j$ and $A^a_i = A^d_j$. That does not imply that $c_i = c_j$. Moreover, the problem formulation does not restrict the flight cost to depend on the flight duration. Consequently, this implies that TP is *asymmetric* and the flight cost does not satisfy the *triangle inequality*.

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

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

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

1.1 Hard Constraints

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

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

1.2 Soft Constraints

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

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

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

Note that most of the aforementioned constraints can be viewed as either hard or soft, depending on the user requirements.

It is therefore suggested that any attempt at an investigation of TP assumes as an additional non-functional requirement that any proposed model to solve TP is flexible and can be easily extended by adding, removing and modifying the aforementioned constraints.

2 Worked Examples

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

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

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

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

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

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

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

$$s = \{GA1, AP4, PM6, MF9, FB11, BG13\}$$

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

A valid solution with lower cost is the following sequence:

$$s' = \{GA1, AP4, PM6, MF9, FB11, BL13, LG14\}$$

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

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

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

$$s'' = \{GF1, FB2, BP4, PM6, MF9, FA10, AG13\}$$

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

3 Use Cases

This Section gives a list of known NP-complete problems [Garey and Johnson, 1979] referred to in this work when proving NP-hardness of TP (Section 4) and as use cases when reviewing the existing work (Section 5).

1. **Travelling Salesman Problem (TSP)**

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

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

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

2. **Time-Constrained TSP (TCTSP)**

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

   Question. It there a TSP tour such that for each city A_i visited at time t_i, $l_i \leq t_i \leq u_i$?

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

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

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

4 Complexity of TP

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

Theorem 1. *The decision version of TP 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 the traveling salesman problem (TSP), defined in Section 3. Its NP-hardness follows by a reduction from the Hamiltonian Cycle problem. The proof is presented by Garey and Johnson [1979].

Given an instance of TPD and s, which is a sequence of flights from F, we can write an algorithm V that checks in polynomial time whether s is a solution. To accept or reject validity, V 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, as defined in Section 3. 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').
- 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 $n + 1$.
- 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 $t \leq T$.
- For every $f_k \in F$, c_k is equal to $d(A_k^d, A_k^a)$ in π. Therefore, the flights cost also satisfies the triangle inequality.
- For every $f_k \in F$, $\Delta_k = 1$.
- B is the upper bound on the allowed total cost.

Suppose that $\gamma = A_{i_1}, A_{i_2}, ..., A_{i_n}$ is the most optimal solution of π, where $< i_1, ..., i_n >$ is a permutation of $< 1, ..., n >$ and the total travel distance $L_\gamma \leq B$. In π', γ is equivalent to the order of visited airports by some sequence of flights $s = f_{i_1}, f_{i_2}, ...f_{i_n}$, such that for every $f_{i_k} \in s, 1 \leq k \leq n$:

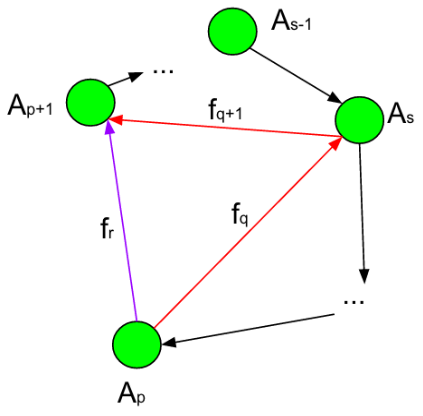

Figure 1: Tour γ'

- $A^d_{i_k} = A_{i_k}, A_{i_k} \in \gamma$, and
- $A^a_{i_k} = A_{i_{k+1}}$, where subscripts are taken modulo n.

From the two conditions above it follows that s satisfies property 1 and 2 of a valid solution. To satisfy property 3 and 4, it is sufficient to choose flights from F such that for every $f_k \in s$, $tk = k$. We know that such flights exist in F by the construction of the set F. Property 5 is satisfied, since all airports in A are destinations. Therefore, s is a solution of π', from which it follows that a solution of π is also a solution of π'.

Suppose that $s = \{f_{j_1}, ..., f_{j_m}\}$ is the most optimal solution of π', where flights in s visit destinations in the sequence $\gamma' = \{A_{i_1}, A_{i_2}, ..., A_{i_n}\}$. We will prove that γ' is a solution of π. By construction of π', γ' contains all airports in A. Suppose that an arbitrary airport A_s, for $i_1 \leq s < i_n$, is included more than once in γ'. Therefore, A_s is listed at least twice in the tour: $\gamma' = \{A_{i_1}, ..., A_{s-1}, A_s, ..., A_p, A_s, A_{p+1}\}$ for $s < p < i_n$ and
$s = \{f_{i_1}, ..., f_q, f_{q+1}, ..., f_{i_m}\}$. The tour is shown in Figure 4, where the green circles represent the airports, the red and black arrows are flights in s and the purple flight f_r in in F, but not in s. Since s is the most optimal solution, $c_q + c_{q+1} < c_r$. However, by construction of π' the cost of the flights satisfies the triangle inequality and $c_q + c_{q+1}$ must be greater than c_r. This is a contradiction. Therefore, A_s can not be added more than once to γ'. Since A_s was arbitrary chosen, the tour contains each airport in A exactly once, which makes it a valid solution to π. Therefore, the most optimal solution of π' is a solution of π.

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

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

□

5 Existing work

We present some existing approaches to dealing with NP-hard problems, namely branch and bound (Section 5.1), constraint programming (Section 5.3), approximation algorithms (Section 5.2) and integer programming (Section 5.4). Each method is illustrated with the aid of a use case, listed in Section 3.

5.1 Branch and Bound Algorithms

The origins of the branch and bound technique date back to the work by **?**, where an instance with 49 cities is solved and proved as optimal. The term "branch and bound" itself was introduced by **?**.

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

Branch and bound procedure can be represented as a rooted tree whose root node corresponds to the original problem and the leafs of the tree form the set of all candidate solutions. The successor nodes of each non-leaf node i correspond to the subproblems, defined by the branching rule.

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

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

5.1.1 Bounds based on the Assignment Problem

This relaxation is based on expressing the integer programming version of an optimisation problem P as an assignment problem (AP) after dropping one or more constraints from P. The AP relaxation is successfully used for solving

Algorithm 1 Branch and Bound Procedure

```
procedure BB (Problem $P$)
    begin
    s ← $\{P\}$                              ▷ put $P$ in the list of active problems set $s$
    bestVal ← NULL
    currentBest ← NULL
    currVal ← NULL
    while s not $\emptyset$ do
        $P_i$ ← CHOOSE(s)                     ▷ choose next subproblem
        s ← s - $P_i$
        $s_i$ = BRANCH($P_i$)                 ▷ branch on $P_i$, returns a list of subproblems
        for $P_{ij}$ in $s_i$ do
            currVal ← BOUND($P_{ij}$)         ▷ calculate the bound of $P_{ij}$
            if currVal worse than bestVal then
                kill $P_{ij}$                  ▷ $P_{ij}$ can not lead to an optimal solution
            else if $P_{ij}$ is a complete solution then
                bestVal ← currVal
                currentBest ← $P_{ij}$
            else
                s ← s + $P_{ij}$
            end if
        end for
    end while
    return currentBest
end procedure
```

TSP **?????** and for the Vehicle Routing Problem (VRP) [**??**]. We explain the principles of this relaxation using TSP as a use case.

Let P be an instance of TSP. The constraints of P can be expressed with the following integer programming model [**???**]:

Let $G(V, E)$ be a complete directed graph with vertex set $V = \{1, ..., n\}$, where each vertex corresponds to a city in P and set of edges $E = \{(i, j)|i, j \in V\}$ and non-negative cost $c(i, j)$ for each edge (i, j) that is equal to the distance $d(i, j)$ in P. For all $i \in V$, $c(i, i) = \infty$ and for all edges (i, j) that one wishes not to include in a tour, $c(i, j) = \infty$. Let $x(i, j)$ be either 1, if edge (i, j) is in the solution, or 0 otherwise. Then, following **?**, P can be formulated as the problem of minimising

$$\sum_{i \in V} \sum_{j \in V} c(i, j) x(i, j) \tag{1}$$

subject to

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

$$\sum_{i \in S} \sum_{j \in S} x(i, j) \leq |S| - 1, \qquad \text{for all } S \subset V, S \neq \emptyset \tag{3}$$

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

In this formulation, one can remove constraint (3) to obtain an instance of AP with the same cost function as P [**???**].

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

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

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

5.1.2 Lagrangian relaxation

5.1.3 Choosing techniques

5.1.4 Branching rules

5.1.5 Performance of Branch and Bound Methods

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

5.2 Approximation Algorithms

5.3 Constraint Programming

5.3.1 Local Search

5.3.2 Heuristics

5.3.3 Use Case

job-shop scheduling problem solution with CP

5.4 Integer Programming

References

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

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

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

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

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

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

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

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

Eugene L. Lawler, David B. Shmoys, Alexander H. G. Rinnooy Kan, and Jan K. Lenstra. *The Traveling salesman problem : a guided tour of combinatorial optimization.* Wiley interscience series in discrete mathematics and optimization. J. Wiley and sons, Chichester, New York, Brisbane, 1987. ISBN 0-471-90413-9. URL `http://opac.inria.fr/record=b1117783`. La 1e ed. est de 1985. Une reimpr. avec corr. a ete realisee en 1986.

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`.

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