Matheuristics to solve the Traveling Analyst Problem

Alexandre Chanson, Vincent t'Kindt, Nicolas Labroche, Patrick Marcel

Matheuristics to solve the Traveling Analyst Problem

Alexandre Chanson[a,b,*], Vincent T'Kindt[a,b], Nicolas Labroche[a], Patrick Marcel[a]

[a] *University of Tours, Laboratoire d'Informatique Fondamentale et Appliqué (EA 6300), 64 avenue Jean Portalis, Tours, 37200, Indre et Loire, France*
[b] *EMR CNRS 7002 Recherche Opérationnelle, Ordonnancement et Transport,*

Abstract

Exploratory Data Analysis (EDA), the notoriously tedious task of interactively analyzing datasets to gain insights, has attracted a lot of attention lately in the data management community. We recently proposed a formal definition of EDA as a multi-objective optimization problem, coined the Traveling Analyst Problem (TAP) and inspired by the Orienteering Problem (OP). The present work investigates the use of matheuristics to compute near optimal solutions to the TAP. We introduce four heuristics including two Variable Partitioning Local Search matheuristics and two Local Branching matheuristics. We present the results of experiments on realistic instances of various sizes to illustrate their effectiveness.

Keywords: matheuristic, orienteering problem, multiobjective optimization

1. Introduction

Exploratory data analysis (EDA) is the tedious interactive analysis of large datasets to extract insights. EDA is typically conducted by an analyst who specifies queries on a database system either using a text language like SQL or through a dedicated GUI. The resulting sequence of queries is called an EDA session. EDA became more popular as data volumes grew over the last two decades. Furthermore, with the development of open data the main issue for data consumers is no longer how to get access to the data but rather how to fully explore and exploit it.

To answer this problem the data management scientific community has recently moved from assisted exploration approaches ([10]), which still rely on the human expert, to more holistic and automated approaches such as the generation of EDA sessions ([9, 11, 20, 22]). From the literature on expert EDA sessions and recent works on EDA sessions generation, it emerges that EDA sessions must (1) present relevant and interesting information to the user, (2) require a limited computational time, and finally (3) be composed of a sequence of queries which appear coherent to the user. We introduced in previous works ([4, 5]) a formal definition of the problem of automatically generating EDA sessions. This problem, named Traveling Analyst Problem (TAP), relates to a well-known family of routing problems called orienteering problems (OP). For instance, assuming that we dispose of a set of database queries, the TAP is defined by an undirected graph $G = \langle V, E \rangle$ in which each vertex $v_i \in V$ represents a query and $(v_i, v_j) \in E$ represents the action of running query v_i before v_j. To meet the three requirements of EDA

*Corresponding Author

sessions, we introduce for each vertex an interestingness score p_i and a service time t_i for each vertex $v_i \in V$. Besides, for each $(v_i, v_j) \in E$, we introduce d_{ij} as the conceptual distance between the two corresponding queries as defined in [1]. The objective of TAP is to produce a routing s over G such that the overall interestingness score $\bar{P}(s) = \sum_{i=1}^{|s|} p_i$ is maximum, the overall distance $\bar{D}(s) = \sum_{i=1}^{|s|-1} d_{i,i+1}$ is minimal and the service time $\bar{T}(s) = \sum_{i=1}^{|s|} t_i$ is minimal. When there is no ambiguity regarding the routing/solution s we use $\bar{P}, \bar{D}, \bar{T}$ instead of $\bar{P}(s)$, $\bar{D}(s)$, $\bar{T}(s)$. The TAP has been shown to be strongly NP-Hard in [4].

The TAP is closely related to the orienteering problem, it differs from it as they is no service times on the vertices in TAP. The OP was formally introduced by Tsiligirides ([25]) who proposed two heuristics to solve it. The first heuristic introduced in [25] is based on a monte-carlo process generating several feasible solutions and choosing the best one. The second heuristic is based on an earlier vehicle routing heuristic by [26]. It relies on circular subdivisions of the euclidean space to construct solutions. It is therefore only compatible with instances giving explicit vertex coordinates. Solutions provided by both algorithms are then improved by a local search called route improvement ([25]) which tries to either introduce new vertices without exceeding the distance budget or to shorten the path length by changing the order of vertices in the route. Chao et al. ([7]) propose a two step heuristic which first builds a set of feasible solutions exploiting geometric features before applying a local search. This one consists in exchanging vertices between the current best solutions and other feasible solutions to improve the former.

Several works describe exact methods for solving the OP. In [13] a branch-and-cut algorithm is proposed and tested on many instances including those from [25] but also larger ones with up to 500 vertices. They solve those large instances to optimality in a few hours. Most of the extensions of the OP tackled in the literature turn out to be more complex to solve to optimality. In [3], Bianchetti et al. propose a branch-and-cut algorithm to solve the Team Orienteering Problem (TOP). In contrast, to the OP in the TOP multiple vehicles are available. This algorithm relies on a MIP formulation, with a polynomial number of variables and constraints, along with a custom branch-and-bound procedure. It is able to solve optimally instances up to 102 vertices and 4 vehicles. Two MIP formulations for the OP are proposed in [19] with a polynomial number of constraints and variables.

In [17], Hue and Linn tackle an extension of the OP, the Team Orienteering Problem with Time Windows (TOPTW), in which a set of identical vehicles are considered with the same travel capabilities. A vertex can be visited by only one vehicle, and the vertices can only be visited within their specified time windows. Hu and Lin propose a metaheuristic to solve this problem. Several routes are generated, one for each agent. Those routes are then stored in a fixed size pool. Routes from this pool are combined to form complete solutions for the problem. The algorithm also features several operators enabling crossover of routes or swaps of vertices in a route to improve the built solutions. This work, along with [7] and [25], points out the importance of reordering vertices when constructing solutions heuristically.

Among the heuristics available in the operations research community, matheuristics have been the matter of a growing interest in the last decade. In the context of the OP and its extensions, two previous works propose matheuristics. In [2] the authors tackle the arc routing team orienteering problem by using a matheuristic, which combines a tabu search and a MILP solver. The algorithm is able to solve 78% of the tested instances to optimality. In [27], Yu et al. focus on time dependent profits and provide a matheuristic which yields high quality solutions for instances with up to 200 vertices. This algorithm solves first the problem of sequencing vertices before using a MILP solver to find appropriate service times for the computed sequence.

In this paper we propose to leverage the recent works made in the field of matheuristics to produce near-optimal solutions for the TAP within reasonable time. We solve instances with up to 700 vertices within a few minutes which corresponds, in our original EDA problem, in processing small-size databases. For a human expert it would require up to a few hours to sift through for interesting information. It is important to notice that researchers working on the automatic generation of EDA sessions have not yet investigated the use of sophisticated heuristics stemming from operations research. This paper provides consequently a strong contribution by making the link between two research communities. It is organized as follows. In Section 2.2, we propose a mathematical programming model of the TAP. In Section 3, we present two constructive heuristics as well as four matheuristics. In Section 4, we report on experimental evaluations of the proposed algorithms and show their efficiency. In Section 5, we conclude this work and propose further research directions.

2. Modeling and properties of the TAP

2.1. Structural properties

In [4], the TAP is defined as a multi objective optimization problem. Therefore, the optimal solution to a TAP instance is not unique: it is a set of incomparable solutions, called Pareto optima ([24]).

Definition 1. *Let $\mathcal{S}$ be the set of feasible solutions to a TAP instance. The set of Pareto optima of a TAP instance is $\mathcal{P} = \{s \in \mathcal{S} : \nexists s' \in \mathcal{S}, \bar{P}(s') \geq \bar{P}(s) \wedge \bar{D}(s) \geq \bar{D}(s') \wedge \bar{T}(s) \geq \bar{T}(s')$, with at least one strict inequality$\}$. A solution $s \in \mathcal{S}$ is called a Pareto optimum.*

Besides the Pareto front, we also establish several properties of the instances, notably a dominance condition on vertices.

Definition 2 (Dominance condition)**.** *$\forall v_i \neq v_j \in V, v_i$ dominates v_j **iff** $t_i \geq t_j, p_j \leq p_i$ and $\forall v_k \in V \setminus \{v_i, v_j\}, d_{ik} \leq d_{jk}$, with at least one strict inequality.*

Lemma 1 (Dominated vertices in Pareto optima)**.** *$\forall s \in \mathcal{P}, \forall v_i \in s, \forall v_j \neq v_i \in V$, **if** v_j dominates v_i **then** $v_j \in s$*

Proof: Let s be a Pareto optimum and two vertices v_i and v_j such that $v_i \in s$, $v_j \notin s$ and v_j dominates v_i. Construct s' by replacing v_i by v_j in s. Then, we have $\bar{P}(s') \geq \bar{P}(s), \bar{D}(s') \leq \bar{D}(s)$ and $\bar{T}(s') \leq \bar{T}(s)$ with at least one strict inequality, which contradicts the fact that $s \in \mathcal{P}$.□

As this dominance condition is unlikely to be satisfied in most instances due to the very restrictive condition on distances, we propose another condition.

Definition 3 (Pseudo-dominance condition)**.** *$\forall v_i \neq v_j \in V, v_j$ dominates v_i **iff** $t_j \geq t_i, p_j \leq p_i$, with at least one strict inequality.*

Unlike the condition stated in Definition 2, the one stated in Definition 3 is not a dominance condition as applying it may lead to discard optimal solutions. That is the reason why the condition of Definition 3 is called a pseudo-dominance condition. Applying the pseudo-dominance condition we define the notion of pseudo dominance set.

Definition 4 (Pseudo-dominance set)**.** *Given an instance, let the pseudo-dominance set of a vertex v_j be $I_j = \{v_i \in V : v_j \neq v_i, v_j$ satisfies the pseudo-dominance condition over $v_i\}$.*

As previously mentioned, dominance condition as stated in Definition 2, is unlikely to happen in practice. At the contrary, the pseudo-dominance condition of Definition 3 may be used to prune the search space in the context of a heuristic solution of the TAP. The benefits of using pseudo-dominance conditions will be evaluated in Section 4. Pseudo-dominance conditions can be used during the solving process as follows: whenever a vertex v_j is selected, all vertices in its pseudo-dominance set I_j are also selected.

In this work we assume that the user is capable of formulating and adjusting bounds on the total distance and time for a given instance. This enables us to use the ϵ-constraint method ([24]) to compute a Pareto optimum for the TAP: we maximize $\bar{P}$ under constraints that $\bar{T} \leq \epsilon_t$ and $\bar{D} \leq \epsilon_d$. This method still enables to enumerate all solutions in $\mathcal{P}$ by solving all problems with different (ϵ_t, ϵ_d) values. In the remainder we focus on the solution of the ϵ-constraint problem. We now describe how the pseudo-dominance conditions can be used to filter any given instance.

Let N be an upper bound on optimal solution sizes, i.e. such that $|V| \geq N \geq |s|$ for any optimal solution s. In this paper N is computed as $N = min(k, k')$, where k and k' are two bounds calculated by exploiting the ϵ-constraints of the problem. Assume that service time are ordered such that $t_1 \leq ... \leq t_{|V|}$. Then, k is defined as $\sum_{j=1}^{k} t_j \leq \epsilon_t < \sum_{j=1}^{|V|} t_j$. Besides, k' is obtained by solving a relaxation of the TAP in which all p_i are set to 1, the $\epsilon-$constraint on $\bar{T}$ is dropped along with sub-tour elimination constraints. An optimal solution to this relaxed problem is found in polynomial time by constructing the shortest 2-cycles between vertices until ϵ_d is reached.

For any given instance we can use the pseudo-dominance sets of vertices and the bound N to design a filtering step: remove vertices v_i such that $|I_i| > N$, as any vertex with a pseudo-dominance set larger than N is unlikely to be in an optimal solution. Remember that this filtering is not optimal as the pseudo-dominance condition may lead to consider vertices as dominated while they are part of some optimal solutions. However, we will see in practice how impacting is this filtering.

2.2. A mixed integer programming (MIP) model for the TAP problem

Now, let us introduce a MIP formulation of the TAP. This model relies on two sets of binary variables to represent vertices selection and sequencing of the selected vertices. We have:

$$\forall i \in 1..n, y_i = \begin{cases} 1 & \text{if vertex } i \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

and

$$\forall i, j \in 0..n+1, i \neq j, x_{ij} = \begin{cases} 1 & \text{if vertex } i \text{ precedes vertex } j \\ 0 & \text{otherwise} \end{cases}$$

We also introduce integer variables $u_i \in \{2, ..., n\}$, $\forall i \in 1..n$, used for sub-tour elimination.

Objective.

$$max \sum_{i=1}^{n} p_i y_i \tag{1}$$

Constraints.

$$\sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} d_{i,j} x_{i,j} \leq \epsilon_d \tag{2}$$

$$\sum_{i=1}^{n} t_i y_i \leq \epsilon_t \tag{3}$$

$$\sum_{i=0, j\neq i}^{n} (x_{i,j}) - y_j = 0, \forall j \in 1..n \tag{4}$$

$$\sum_{j=1, j\neq i}^{n+1} (x_{i,j}) - y_i = 0, \forall i \in 1..n \tag{5}$$

$$\sum_{j=1}^{n} x_{0j} = \sum_{i=1}^{n} x_{i,n+1} = 1 \tag{6}$$

$$u_i - u_j + 1 \leq (n-1)(1 - x_{ij}), \forall i, j \in 1..n, i \neq j \tag{7}$$

This model involves (n^2+5n+1) variables and (n^2+2n+4) constraints. The objective (1) aims at maximizing the total score, i.e. the interestingness of the sequence of vertices. Constraint (2) ensures that the total distance does not exceed a threshold ϵ_d. Similarly constraint (3) ensures that the total service time does not exceed a threshold ϵ_t. Constraints (4) and (5) ensure the solution is a path (if a vertex is selected in the solution, then one arc must enter it and one must leave it). Constraint (6) ensures there is only one start and one end vertex. Finally, we use classic TSP sub-tour elimination constraints (7) to ensure a single sequence is computed. Here, we chose those presented in [21].

Constraints corresponding to pseudo-dominance conditions. Using Definition 3 we propose to add an additional set of constraints to the model.

$$y_i \leq y_j, \forall i \in 1..n, \forall j \in I_i \tag{8}$$

Adding those constraints may lead the optimal solution to be infeasible as they are based on the pseudo-dominance condition. We evaluate the impact of adding these constraints in Section 4.

3. Variable partitioning local search and a local branching heuristics

Matheuristics are hybrid approaches combining metaheuristics and exact methods ([8]) by embedding a MIP solver into traditional heuristic processes. The role of the MIP solver can vary from a method to another. A classic one, called Variable Partitioning Local Search (VPLS), has proved to be efficient especially on hard permutation problems ([8]). Another method, called Local Branching and introduced in [14], has also proved to be effective. In this section, we will briefly discuss both methods and propose two algorithms based on each method to solve the TAP. Since both approaches require an initial solution, we present in Section 3.3 two possible initialization heuristics.

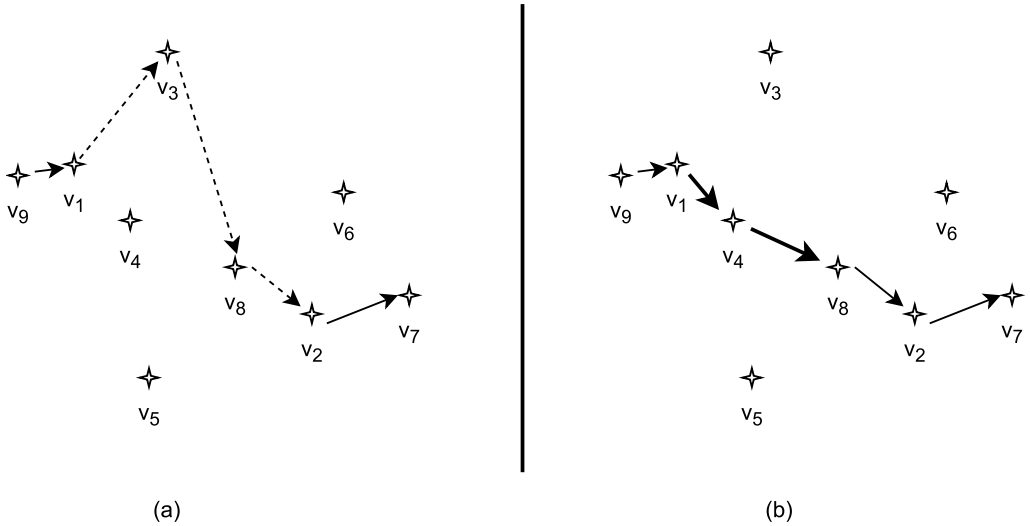

Figure 1: Example of a solution improved by VPLS algorithm

3.1. VPLS

The VPLS method improves an incumbent solution by iteratively reoptimizing a part of it. As the reoptimization is done via the solution of a MIP, at each iteration two sets of variables are defined. One set of variables is fixed as in the incumbent solution while the variables of the other set are let free so that they define a small MIP to be solved. Hopefully, this new small MIP can be solved quickly. In the case of the TAP the solution is a sequence: thus, we apply the VPLS method by selecting a sub-sequence of vertices (called the reoptimization window) and freeze the remaining vertices of the solution. Given a solution s^t at iteration t, let us denote by w_{start} (respectively w) the starting position (respectively length) of the window. The set of variables fixed to their current values for iteration $t+1$ is defined as: $F_{t+1} = \{y_i, i \in s^t \setminus s^t[w_{start} : w_{start} + w]\} \cup \{x_{ij}, i \in s^t \setminus s^t[w_{start} : w_{start} + w], j \in 1..n, j \notin s^t[w_{start} : w_{start} + w]\} \cup \{x_{ji}, i \in s^t \setminus s^t[w_{start} : w_{start} + w], j \in 1..n, j \notin s^t[w_{start} : w_{start} + w]\}$. With $s^t[a : b]$ being the sub-sequence starting from position a and ending at position b (inclusive). This is an iterative process which can be stopped by a specific convergence criterion like a maximum time limit or a maximum number of iterations.

We provide an example in Figure 1. Figure 1.a shows the current feasible solution $s^t = [v_9, v_1, v_3, v_8, v_2, v_7]$ at iteration t, together with the unselected vertices $\{v_4, v_5, v_6\}$. Now, assume that we decide to reoptimize the window $[v_3, v_8]$, which is represented by dotted arrows. Figure 1.b shows $s^{t+1} = [v_9, v_1, v_4, v_8, v_2, v_7]$ which is the solution obtained after solving the corresponding MIP with a reduced set of variables. One of the key points of VPLS relates to the choice of the reoptimization window at each iteration. We propose two methods and both assume that the window length w is a given parameter. The first heuristic called *vpls-det*, described in Algorithm 1, moves the reoptimization window from the start to the end of the sequence. An additional parameter o may induce an overlap of windows between iterations if set to a non-zero value by the user. This window is positioned back at the beginning of the sequence when, at an iteration t, the solution is improved. The second heuristic, called *vpls-random* and described in Algorithm 2, randomly selects the window along the sequence.

Algorithm 1 vpls-det

Input: An instance (scores, service times, distances, ϵ_t, ϵ_d), the window width w, window overlap o, the maximum number of iterations, and the maximum iteration time. A feasible solution s^f

Output: A solution to the TAP, of service-time at most ϵ_t and overall distance at most ϵ_d.

$t \leftarrow 0$
$s^t \leftarrow s^f$
$w_{start} \leftarrow 0$
while $t <$ max. iterations **do**
    $s^{t+1} \leftarrow MIP(s^t, w_{start}, w,$ max. iteration time) ▷ solve reduced MIP
    if $\bar{P}(s^{t+1}) > \bar{P}(s^t)$ **then**
        $w_{start} \leftarrow 0$
        $s^t \leftarrow s^{t+1}$
    else
        $w_{start} \leftarrow w_{start} + w - o$
    end if
    if $w_{start} + w > |s^t|$ and $\bar{P}(s^t) = \bar{P}(s^{t+1})$ **then**
        return s^t
    end if
    $t \leftarrow t + 1$
end while
return s^t

Algorithm 2 vpls-random

Input: A TAP instance (scores, service times, distances, ϵ_t, ϵ_d), the window width w, the maximum number of iterations, and the maximum iteration time. A feasible solution s^f

Output: A solution to the TAP, of service-time at most ϵ_t and overall distance at most ϵ_d.

$t \leftarrow 0$
$s^t \leftarrow s^f$
while $t <$ max. iterations **do**
    $w_{start} \leftarrow$ pick a random window starting position
    $s^t \leftarrow MIP(s^t, w_{start}, w,$ max. iteration time) ▷ solve reduced MIP
    $t \leftarrow t + 1$
end while
return s^t

3.2. Local branching

The two other matheuristics we introduce are a direct application of the Local Branching ([8]) method with no diversification phase. They rely on the Hamming distance ([15]) between the two sets of decision variables of the MIP, as described in Section 2.2: variables y_i represent the presence or absence of a vertex in the solution while the $x_{i,j}$'s represent the order between selected vertices. Thus, any two solutions can be compared by the Hamming distance, either on the complete set of decision variables or on a subset. For example, assume it is computed w.r.t. only the x_{ij}'s, then the Hamming distance $\Delta(x, x^s)$ to a known solution x^s is given by:

$$\Delta(x, x^s) = \sum_{i,j=1,x^s_{ij}=1}^{n} (1 - x_{ij}) + \sum_{i,j=1,x^s_{ij}=0}^{n} x_{ij} \tag{9}$$

This distance can be used to build a constraint that effectively constrains the solver to search in the neighborhood of an incumbent solution. In contrast to the VPLS approaches, which are limited to modifications within a specific sub-sequence, this approach enables broader transformations of the solutions such as swapping the first vertex and the last vertex of the solution. The two local branching heuristics we propose are denoted by *lb-y* and *lb-yx* and are described in Algorithm 3. They simply vary on the constraints they use, (10) for *lb-y* and (11) for *lb-yx*. Let s^t be the best solution known at iteration t and let (x^t, y^t) be its associated variable vector. Then, have:

$$\Delta(y, y^t) < h \tag{10}$$

$$\Delta(x, x^t) + \Delta(y, y^t) < h \tag{11}$$

with h a parameter limiting the maximum number of variables to be changed.

Algorithm 3 lb-y / lb-yx

Input: A TAP instance (scores, service times, distances, ϵ_t, ϵ_d), the maximum hamming distance h, the maximum number of iterations and the maximum iteration time. A feasible solution s^f

Output: A solution to the TAP, of service-time at most ϵ_t and overall distance at most ϵ_d.

1: $t \leftarrow 0$
2: $s^t \leftarrow s^f$
3: **while** $t <$ max. iterations **do**
4: $\quad s^t \leftarrow MIP(s^t,$ constraint (10) or (11), max. iteration time$)$
5: $\quad t \leftarrow t + 1$
6: **end while**
7: return s^t

3.3. Initial solutions

The importance of the initial solution given to the matheuristics is crucial, as the matheuristics can be seen as metaheuristics, which improve a given solution. Hopefully, the latter is obtained quickly and is sufficiently good so that the matheuristic can reach near-optimal solutions. We propose two different heuristics to construct initial solutions. The first one, called *h-ks*, runs in

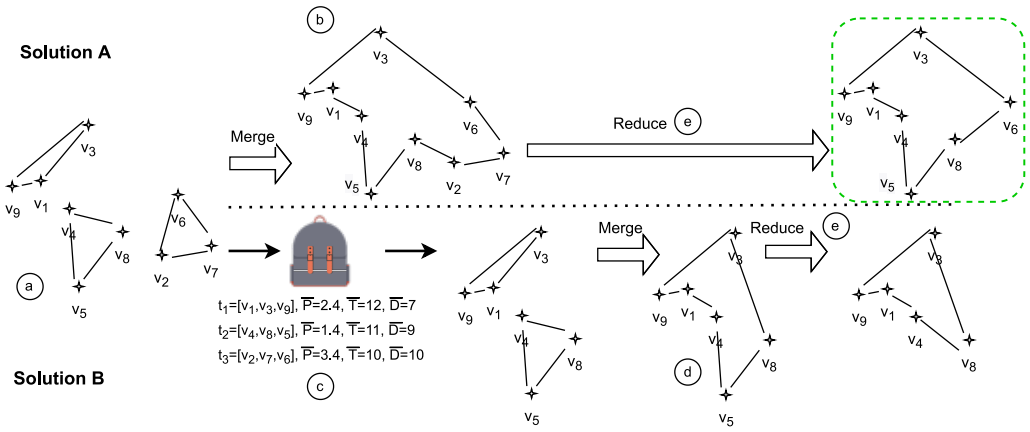

Figure 2: General principle of h-tsp for generating Solution A (top) and Solution B (bottom) starting from a solution to the assignment problem over the distance matrix

$O(n^2)$ time and exploits the knapsack structure of the problem. First, vertices are selected by decreasing order of their ratio p_i/t_i, until constraints (2) and (3) prevent from more insertions. In addition to this classic knapsack heuristic, whenever a vertex is selected, it is inserted at the position in the partial solution which minimizes the $\bar{D}$ criterion of the selected solution under construction.

The second algorithm, called *h-tsp*, is an approach inspired by sub-tour merging heuristics for the traveling salesman problem (TSP). This algorithm produces 2 solutions which are compared and the best one is returned. The algorithm (see Figure 2) starts by solving a relaxed version of the TSP over G with distances d_{ij} when sub-tour elimination constraints are removed. This yields an assignment problem that is solved by the successive shortest path method ([12]) in $O(n^3 log(n))$ time. This may lead to a potentially non feasible solution composed of sub-tours (see step ⓐ Figure 2). From this point the algorithm computes two solutions. First to compute, **Solution A**, the algorithm merges all sub-tours (see step ⓑ Figure 2). This step is skipped if the assignment problem solution is composed of a single tour. The merging of sub-tours is done by applying an implementation of the minimum spanning tree approach described in [18]. This merging approach was chosen as it outperforms other tour construction heuristics ([18]). Since after merging sub-tours the distance and/or time ϵ-constraint may be violated, we design an additional redcution step to fix this issue (see step ⓔ Figure 2). First, if the ϵ-bound on time is not answered, queries are eliminated in increasing order of their ratio p_i/t_i until it is no longer violated. Then, if the ϵ-bound on distance is not respected, queries are eliminated in increasing order of p_i value, every η eliminated queries the tour is reoptimized by the LKH heuristic[1] ([16]).

Solution B is obtained by considering all sub-tours as single elements in a Multi-Dimensional Knapsack. Each sub-tour is associated to its total length, service-time and interestingness score (see step ⓒ Figure 2). This Multi-Dimensional Knapsack problem is solved to extract a set of sub-tours that are next merged (see step ⓓ Figure 2) using the same merging approach as Solution A. The same reduction step as **Solution A** can be applied if any ϵ-constraint is violated.

Finally, **Solution A** and **Solution B** are compared, and the best (feasible) one in terms of $\bar{P}$

[1]We use the implementation provided by Helsgaun http://webhotel4.ruc.dk/ keld/research/LKH/

value is returned.

4. Computational experiments

We organize this experimental section as follows: first, we describe the different types of instances we use and their properties; second, we evaluate the impact of filtering and adding constraints as described in Section 2.2; third, we compare the quality of solutions produced by initialization heuristics; fourth, using the best initialization heuristic for each instance we evaluate the solution quality of the four matheuristics. Finally, we compare the performance of the heuristics and matheuristics to CPLEX given a similar time budget on large instances.

For all experiments, we used the CPLEX solver version 20.10 running on a Fedora Linux (kernel 5.11.13-200) workstation, with two 2.3 Hz Intel Xeon 5118 with 377GB of memory. To show realistic running times the CPLEX solver is run in single-thread mode whenever a MIP model has to be solved. We provide an open source implementation of all algorithms in our Git repository `https://github.com/AlexChanson/TAP-Matheuristics`. This repository also contains all TAP instances used in this section.

4.1. Instances generation

No suitable benchmark for our problem is available in the literature. This due to the added service time and the use of a metric with no triangular inequality. We instead use four families of randomly generated instances.

- The first family of instances, denoted by *f1*, is designed to be mainly sort of knapsack instances. The interestingness score is a real number drawn from an uniform distribution between 0 and 1. The distance is an integer number drawn from a uniform distributions between 5 and 6. Finally the service time is an integer number drawn from a uniform distribution between 5 and 50.

- The second family of instances, denoted by *f2*, is designed to be mainly sort of TSP instances. The interestingness score is an integer number drawn from an uniform distribution between 1 and 3. The distance is an integer number drawn from a uniform distributions between 1 and 14. Finally the service time is an integer number drawn from a uniform distribution between 5 and 6.

- The third family of instances, denoted by *f3*, is designed to contain particularly hard knapsack instances (service time and profit are strongly correlated [23]), while still having a strong routing component. The distance is an integer number drawn from a uniform distributions between 1 and 10. The service time is an integer number drawn from a uniform distribution between 5 and 50. The interestingness score is equal to the service time plus 5.

- The last family of instances, denoted by *f4*, is similar to real world instances. The interestingness score is a real number drawn from an uniform distribution between 0 and 1. The distance is an integer number drawn from a uniform distributions between 1 and 10. Finally the service time is an integer number drawn from a uniform distribution between 5 and 50. These constants were set by manually generating a few instances with databases found on the open data platform Kaggle ([6]).

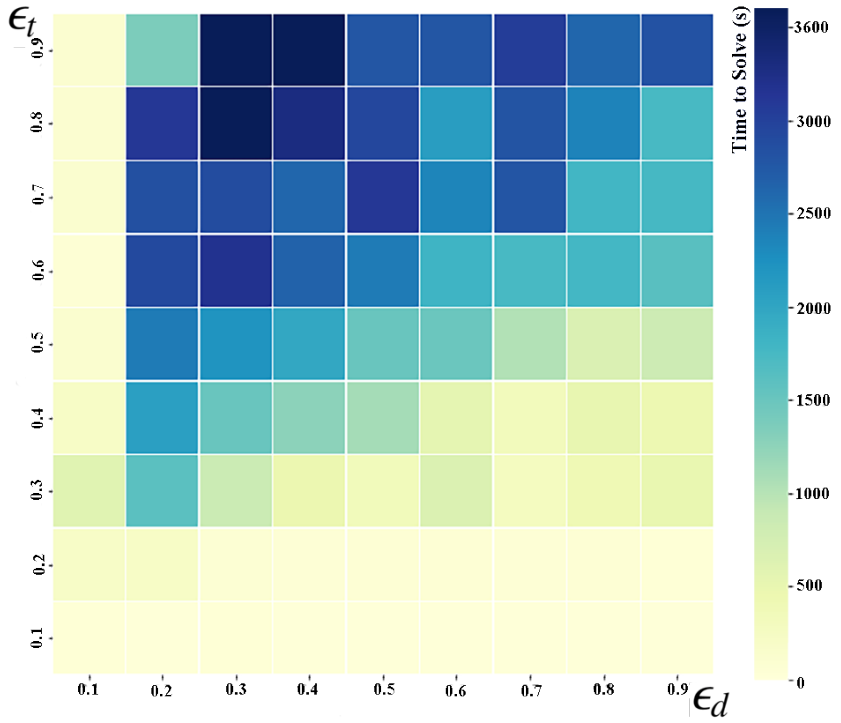

Figure 3: Average time taken by CPLEX to solve an instance from set *f3* to optimallity

For each family of instances, we generate 30 instances for each size $n \in \{40, 60, 80, 100, 200, 300, 400, 500, 600, 700\}$.

In this work, we will need solve single objective TAP instances. We intend to fix ϵ_t and ϵ_d to simplify the interpretation of results. Due to the varying size of instances we express ϵ_t and ϵ_d as the expected fraction of queries in the instances in the solution. For example if ϵ_t is set to 0.35 for a 200 vertices, it means we expect that the time constraint will, on average, lead to solutions containing 70 queries. By choosing different values for ϵ_t and ϵ_d we can make one $\epsilon-$constraint more restrictive than the other.

To choose a combination of $\epsilon-$constraints that is not too simple to solve. We measured the time taken by CPLEX to produce an optimal solution (1h time limit) on 30 instances of *f4* with 300 vertices, with various $\epsilon-$constraints combinations. We report those results in Figure 3. We decide on ϵ_t set to 0.6 and ϵ_d set to 0.3 for the remainder of this work. As this appears to be a hard scenario for the solver, but where it is still practically possible to obtain optimal solutions in a few hours.

4.2. Evaluation of pseudo-dominance and filtering

In this section we evaluate the effectiveness of filtering pseudo-dominance conditions. Both approaches were proposed in Section 2.1. We only conduct experiments on instances of family *f4*, as for those of family *f3* pseudo-dominance conditions never apply.

First, we propose to evaluate the impact of adding constraints based on pseudo-dominance to the solution of the MIP. We use CPLEX with default parameters and timeout of two hours, with

and without pseudo-dominance. In Table 1 we report the number of nodes #Nodes explored by CPLEX, and the average and maximum deviation, respectively Δ_{avg} and Δ_{max} from the optimal solution value (computed without pseudo-dominance). Finally, we report the average and maximum time gained ΔT_{avg} and ΔT_{max} compared to CPLEX without pseudo-dominance: negative values denote lost time over running the vanilla solver. In a set of preliminary experiments it was noted that the pseudo-dominance produced a large number of constraints potentially spoiling the solver. Thus, we also report results for only adding a fraction of the valid constraints, which is achieved by only selecting constraints that relate to vertices with dominance sets larger than $\kappa \times |V|$, with $\kappa \in \{0, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$.

Table 1 shows that adding the constraints (even partially) seems to marginally degrade the solution quality. However, this approach is always requiring more CPU time than CPLEX to solve instances. Even with very few added constraints. We note though that the best case scenario seems to be for $\kappa = 0.8$.

As we may eventually use pseudo-dominance in the matheuristics, we test solution quality of CPLEX alone, against CPLEX with added constraints (for $\kappa = 0.8$) in a limited 60 seconds time budget. This would be typical of the time taken by one iteration of the matheuristics. Solution quality is reported in terms of average relative deviation to the optimal solution in Table 2.

Size	CPLEX	CPLEX with pseudo-dominance ($\kappa = 0.8$)
40	0	0
60	0	0
80	0	0
100	0.05	0.14
200	30.23	9.31
300	91.11	90.71
400*	99.69	99.68

Table 2: Average relative deviation to the optimal solution objective after 60 seconds for varying instance sizes (* deviation reported against near-optimal solutions, MIP gap < 1%)

Results from Table 2 show that the approach can provide some improvement for instances of size 200. However, since this trend doesn't extend to instances beyond this size, we choose to discard this approach for the remainder of this work.

For the filtering approach, a preliminary test was first conducted to evaluate the number of vertices the filtering step removes on instances of family *f4*. Its results are reported in Table 3. This shows that applying the method based on the computed bound N only filters out about 3% of the vertices. This is unlikely to be enough to achieve significant performance improvements.

κ	Size	Δ_{avg}	Δ_{max}	ΔT_{avg} (s)	ΔT_{max} (s)	#Nodes_{avg}	#Nodes_{max}
0	40	0	0	0	0	0	0
	60	0	0	0	0	0	0
	80	0	0	0	0	0	0
	100	0	0	0	0	0	0
	200	0	0	0	0	0	0
	300	0	0	0	0	0	0
0.2	40	0	0	-0.74	0.2	69	580
	60	0	0	-1.51	0.84	129	1443
	80	0	0.01	-1.86	17.21	118	1191
	100	0	0	-45.02	51.98	575	3629
	200	0	0.01	-1003.11	1743.38	1702	5950
	300	1.98	21.83	-2363.51	404.39	1720	4265
0.3	40	0	0	-0.39	0.44	66	630
	60	0	0	-1.01	2.43	83	1024
	80	0	0.01	-2.67	17.77	197	1000
	100	0	0	2.36	99.35	333	2104
	200	0	0.01	-679.46	2068.54	1423	3326
	300	0.05	0.52	-2985.96	-408.19	1748	2804
0.4	40	0	0	-0.4	0.08	67	503
	60	0	0	-1.17	1.9	68	671
	80	0	0.01	1.34	19.96	42	494
	100	0	0.01	-10.26	68.16	433	2144
	200	0	0.01	-196.64	2043.44	1316	4549
	300	0	0.01	-2099.43	916.81	1351	2717
0.5	40	0	0	-0.45	0.4	71	585
	60	0	0.01	-0.52	1.63	48	490
	80	0	0.01	-1.54	18.96	149	772
	100	0	0	-10.77	95.47	558	2560
	200	0	0.01	-489.51	1495.5	1457	3434
	300	0	0	-978.76	451.26	974	1584
0.6	40	0	0	-0.39	0.49	78	482
	60	0	0.01	-1.35	1.06	144	1440
	80	0	0.01	-1.54	21.37	237	1976
	100	0	0	-7.35	81.4	489	1688
	200	0	0.01	-461.65	2107.72	1285	3428
	300	0	0	-2243.49	853.16	1264	2304
0.7	40	0	0	-0.35	0.42	57	439
	60	0	0	-0.96	1.95	100	1282
	80	0	0	-3.83	19.42	202	1129
	100	0	0	-10.92	113.96	476	1332
	200	0	0.13	-547.39	1786.61	1483	3667
	300	0.01	0.09	-834.5	1164.47	1157	3531
0.8	40	0	0	-0.24	0.25	61	612
	60	0	0	-0.98	1.79	68	649
	80	0	0.01	-6.87	7.43	312	2012
	100	0	0	-16.14	52.61	486	1827
	200	0.01	0.35	-112.72	1847.14	1050	2550
	300	0	0.05	-1189.3	2137.28	943	2228

Table 1: Solution quality and running time for CPLEX solving instances with pseudo-dominance

Size	Min. Filtered (%)	Avg. Filtered (%)	Max. Filtered (%)
40	0.00	2.25	7.50
60	0.00	3.06	8.33
80	0.00	3.00	7.50
100	1.00	3.37	6.00
200	1.50	3.13	5.00
300	1.33	3.23	5.00
400	1.25	3.15	5.25
500	0.80	2.93	5.40
600	2.17	2.88	3.67
700	2.00	2.99	4.43

Table 3: Proportion of vertices removed from instances by the filtering step

However, since our work focuses on an heuristic approach and the pseudo-dominance condition is not exact, we propose to remove more vertices by sorting them in decreasing order of their dominance set size and by removing the first 10%, 15% and 20% of the most dominated queries.

vertices removed	Size	Δ_{avg}	Δ_{max}	ΔT_{avg} (s)	ΔT_{max} (s)	#Nodes_{avg}	#Nodes_{max}
10%	40	0	0	-0.1	0.84	70	510
	60	0	0.01	-0.38	3.14	151	2486
	80	0	0.01	0.65	22	117	1048
	100	0	0.01	0.03	101.14	444	1745
	200	0	0.01	160.61	2252	1462	4563
	300	0.24	1.83	1493.71	5770.97	1017	2420
15%	40	0	0	0.07	0.76	41	360
	60	0	0	-0.51	2.53	182	1347
	80	0	0.01	1.67	21.97	65	634
	100	0	0	2.89	119.44	430	1200
	200	0	0.01	528.63	3786.35	1201	3919
	300	0.18	1.14	1409.66	6512.28	1220	1924
20%	40	0.27	2.88	238.87	3586.31	86528	1329708
	60	0.06	1.01	0.19	2.38	13	138
	80	0.02	0.28	1.51	18.71	198	1202
	100	0	0.06	4.22	122.9	562	2021
	200	0.02	0.32	516.67	2800.96	1037	3229
	300	0.03	0.17	148.61	6510.58	1089	1966

Table 4: Solution quality and run time for CPLEX solving Filtered instance

We report in Table 4 the the number of nodes #Nodes explored by CPLEX, and the average and maximum deviation, respectively Δ_{avg} and Δ_{max} from the optimal solution value (computed without filtering). Finally, we report the average and maximum time gained ΔT_{avg} and ΔT_{max} compared to running CPLEX on unfiltered instances. Similarly to previous experiments negative values represent lost time. The results in Table 4 show that it is beneficial to remove more queries than only the 3% that would have been removed by using the bound N on optimal the solution size. Removing 10% to 15% of queries yields a significant improvement in the running time (of

more than 20 minutes on larger instances) without deteriorating to much the solution quality: the deviation is at most 1.83% with respect to optimal solutions. We observe diminishing returns with 20% of vertices filtered with a lower ΔT_{avg}, we will therefore use a 15% filtering on all further experiments.

4.3. Evaluation of the initialization heuristics

We now propose to evaluate the quality of the initialization heuristics. We compare the running times of *h-ks* and *h-tsp* along with their deviation to the optimal solutions computed by CPLEX. We conduct this series of experiments on instances of family *f1* and *f2*. Those sets of instances respectively constitute mainly knapsack and TSP like instances. Thus, we expect *h-ks* to perform better on *f1*, and *h-tsp* to perform better on *f2*. Additionally, given the results of the previous experiments we propose to test the effect of the filtering approach on the heuristics. We report the average and maximum deviation, respectively Δ_{avg} and Δ_{max} from the optimal solution value along with the average running time for each heuristic in Table 5.

Table 5: Run time and quality of solutions produced by initialization heuristics on unfiltered instances

Familly	Algorithm	Size	Δ_{avg}	Δ_{max}	Avg. running time
f1	*h-ks*	40	0.27	1.87	0.01
f1	*h-ks*	60	0	0	0.01
f1	*h-ks*	80	0.05	0.63	0.01
f1	*h-ks*	100	0.7	4.81	0.01
f1	*h-ks*	200	0.28	1.34	0.01
f1	*h-ks*	300	0.09	0.35	0.01
f1	*h-tsp*	40	6.9	12.49	0.56
f1	*h-tsp*	60	4.34	8.44	1.35
f1	*h-tsp*	80	5.8	9.55	2.58
f1	*h-tsp*	100	6.03	7.61	4.51
f1	*h-tsp*	200	5.74	8.32	28.66
f1	*h-tsp*	300	5.73	8.4	390.28
f2	*h-ks*	40	0.6	1.82	0.01
f2	*h-ks*	60	0.29	1.16	0.01
f2	*h-ks*	80	0.22	0.87	0.01
f2	*h-ks*	100	0.32	0.69	0.01
f2	*h-ks*	200	0.11	0.33	0.01
f2	*h-ks*	300	0.07	0.22	0.01
f2	*h-tsp*	40	0.32	1.82	0.15
f2	*h-tsp*	60	0.36	1.11	0.58
f2	*h-tsp*	80	0.19	0.84	0.88
f2	*h-tsp*	100	0.26	0.69	0.33
f2	*h-tsp*	200	0.13	0.33	1.35
f2	*h-tsp*	300	0.04	0.22	3.71

As expected, the results in Table 5 show a clear advantage of *h-ks* for the knapsack like instances (family *f1*). The average deviation of *h-ks* remains mostly under 1%, with its running time is under 10ms. Meanwhile, *h-tsp* produces solutions with a deviation of 5-6% with a particularly long running time. When examining the results on instances from family *f2* however

we note the superiority of *h-tsp* over *h-ks* is marginal. Yet, results in Table 6 show that *h-tsp* produces better solutions than *h-ks* in some cases. However, due to the low cost of running those heuristics on most instances types, we still believe it is better to run both and pick the best result.

Family	*h-ks*	*h-tsp*	tie
f1	175	0	5
f2	24	32	119
f3	23	3	154
f4	152	18	10

Table 6: Number of instances which an heuristic performs better than the other, wuth number of ties.

Now considering the filtering step, we report the result of running both heuristics on filtered instances in Table 7. The filtering seems marginally degrade the heuristics in a few scenarios. We also note a reduction in the running time of *h-tsp* on family *f1* as a side effect of the filtering.

Table 7: Running time and quality of solutions produced by initialization heuristics on filtered instances

Familly	Algorithm	Size	Δ_{avg}	Δ_{max}	Avg. running time
f1	*h-ks*	40	0.27	1.87	0.01
f1	*h-ks*	60	-0	0	0.01
f1	*h-ks*	80	0.05	0.63	0.01
f1	*h-ks*	100	0.7	4.81	0.01
f1	*h-ks*	200	0.28	1.34	0.01
f1	*h-ks*	300	0.09	0.35	0.01
f1	*h-tsp*	40	5.19	10.39	0.43
f1	*h-tsp*	60	4.18	8.43	0.94
f1	*h-tsp*	80	4.29	9.2	1.73
f1	*h-tsp*	100	5.21	9.89	3.00
f1	*h-tsp*	200	5.17	7.01	18.10
f1	*h-tsp*	300	4.49	6.62	226.61
f2	*h-ks*	40	0.6	1.82	0.01
f2	*h-ks*	60	0.29	1.16	0.01
f2	*h-ks*	80	0.3	1.61	0.01
f2	*h-ks*	100	0.39	2	0.01
f2	*h-ks*	200	0.22	1.67	0.01
f2	*h-ks*	300	0.09	0.22	0.01
f2	*h-tsp*	40	0.27	1.82	0.10
f2	*h-tsp*	60	0.32	1.11	0.46
f2	*h-tsp*	80	0.27	1.61	0.76
f2	*h-tsp*	100	0.37	2	0.64
f2	*h-tsp*	200	0.21	1.67	0.95
f2	*h-tsp*	300	0.07	0.22	2.33

4.4. Comparison of the matheuristics with optimal solutions

We now compare the four matheuristics by evaluating their deviation to the optimal solution when they are all given a 10 minutes time budget. We conduct this experiment on both instances

of family *f3* and *f4* and with and without the filtering step. For each matheuristic, we run preliminary tests to find an efficient combination of its parameters, and the obtained result can be found in Table 8. Note that for each instance, the matheuristics are given the best initial solution from either *h-tsp* or *h-ks*.

Algorithm	m_{iter}	t_{iter}	w	k	h
vpls-det	5	120	15	7	N.A.
vpls-random	7	90	20	10	N.A.
lb-y	7	90	N.A.	N.A.	15
lb-yx	5	120	N.A.	N.A.	50

Table 8: Best parameters for the four matheuristics given our preliminary experiments

In Table 9 we report the results of matheuristics running on unfiltered instances, the average and maximum deviation, respectively Δ_{avg} and Δ_{max} from the optimal solution value along with the average running time for each matheuristic. For uncorrelated instances (*f3*) the four algorithms provide extremely low deviations with *vpls-det* appearing to slightly outperform. However, this advantage doesn't appear to hold for instances of family *f4*. With those correlated instances it appears that *lb-yx* performs significantly better with an average deviation under 5% on the largest instances. As explained in Section 4.2 we only report in Table 10 results using the filtering approach on instances of family *f4*. With this filtering step we observe better results on large instances as matheuristics do not reach their time limit. The deviations are overall extremely low. As for the fastest algorithm , *vpls-random* manages an average of 66 seconds on instances of 300 vertices.

4.5. Performance on large instances

As a final experiment we propose to evaluate the capabilities of the matheuristics with the filtering step to solve large instances of the TAP. We compare the performance of the four matheuristics with a 10 minutes time limit on the largest instances of family *f4*. We report δ^I_{avg} the deviation to the initial (*h-tsp* or *h-ks*) solution for each matheuristic in Table 11.

The results in Table 11 show that all matheuristics are equally able to improve the solutions provided by the initialization heuristics. A 2% to 4% improvement is achieved depending on the size of the instances. With the largest instances only improved by 2%.

5. Conclusion

In this paper we studied the solutions of the Traveling Analyst Problem (TAP) using simple and effective matheuristics based on the VPLS and Local Branching methods. We investigated possible techniques to further accelerate solution of the MIP using filtering and additional constraints both based on pseudo-dominance. We showed that adding constraints to the MIP while reducing the number of possible solutions was detrimental to the solver in both exact mode and limited time. The filtering approach showed however no significant degradation of the solutions and provided significant time savings. We also proposed and studied two polynomial heuristics based on TSP and Knapsack heuristics. They proved effective in solving instances of the TAP close to these two problems. They provide good starting solutions to the matheuristics which are capable of converging to near optimal solutions in a few iterations. The four matheuristics are

Family	Algorithm	#vertices	Δ_{avg}	Δ_{max}	Avg. time (s)
f3	*vpls-det*	40	0	0	0.97
f3	*vpls-det*	60	0	0	2.71
f3	*vpls-det*	80	0	0	8.26
f3	*vpls-det*	100	0	0	14.65
f3	*vpls-det*	200	0	0.02	143.54
f3	*vpls-det*	300	0.19	0.5	514.97
f3	*vpls-random*	40	0	0	0.92
f3	*vpls-random*	60	0	0	2.68
f3	*vpls-random*	80	0	0	8.51
f3	*vpls-random*	100	0	0	16.91
f3	*vpls-random*	200	0	0.02	172.30
f3	*vpls-random*	300	0.25	0.53	451.24
f3	*lb-y*	40	0	0	1.14
f3	*lb-y*	60	0	0	3.19
f3	*lb-y*	80	0	0	9.36
f3	*lb-y*	100	0	0	18.90
f3	*lb-y*	200	0.04	0.57	173.03
f3	*lb-y*	300	0.26	0.53	588.67
f3	*lb-yx*	40	0	0	1.01
f3	*lb-yx*	60	0	0	2.84
f3	*lb-yx*	80	0	0	8.83
f3	*lb-yx*	100	0	0	17.27
f3	*lb-yx*	200	0.01	0.22	195.25
f3	*lb-yx*	300	0.26	0.53	555.34
f4	*vpls-det*	40	0	0	0.71
f4	*vpls-det*	60	0	0.01	1.70
f4	*vpls-det*	80	0	0	2.78
f4	*vpls-det*	100	0	0.1	10.59
f4	*vpls-det*	200	0.02	0.05	68.17
f4	*vpls-det*	300	18.16	99.85	211.16
f4	*vpls-random*	40	0	0	0.65
f4	*vpls-random*	60	0	0.01	1.54
f4	*vpls-random*	80	0	0	3.73
f4	*vpls-random*	100	0	0.1	12.53
f4	*vpls-random*	200	0.01	0.08	85.17
f4	*vpls-random*	300	9.16	99.8	217.31
f4	*lb-y*	40	0	0	1.04
f4	*lb-y*	60	0	0.01	2.69
f4	*lb-y*	80	0	0	6.23
f4	*lb-y*	100	0	0	21.69
f4	*lb-y*	200	0.45	12.86	210.37
f4	*lb-y*	300	6.18	34.96	323.70
f4	*lb-yx*	40	0	0	0.89
f4	*lb-yx*	60	0	0.01	2.25
f4	*lb-yx*	80	0	0	5.14
f4	*lb-yx*	100	0	0	19.70
f4	*lb-yx*	200	0.23	6.69	198.66
f4	*lb-yx*	300	4.81	26.86	227.97

Table 9: Solution quality and running time of matheuristics on instances of families f3 and f4 (without filtering)

Algorithm	#vertices	Δ_{avg}	Δ_{max}	Avg. time (s)
vpls-det	40	0	0	0.57
vpls-det	60	0	0.01	1.30
vpls-det	80	0	0.01	3.12
vpls-det	100	0	0.05	9.07
vpls-det	200	0.02	0.07	49.87
vpls-det	300	0.01	0.04	103.94
vpls-random	40	0	0	0.56
vpls-random	60	0	0.01	1.28
vpls-random	80	0	0.01	3.20
vpls-random	100	0	0	10.84
vpls-random	200	0	0.04	94.95
vpls-random	300	0.01	0.04	66.75
lb-y	40	0	0	0.82
lb-y	60	0	0.01	2.13
lb-y	80	0	0.01	5.43
lb-y	100	0	0	19.17
lb-y	200	0	0.03	213.71
lb-y	300	0.01	0.04	274.33
lb-yx	40	0	0	0.68
lb-yx	60	0	0.01	1.80
lb-yx	80	0	0.01	5.21
lb-yx	100	0	0	16.81
lb-yx	200	0	0.01	147.67
lb-yx	300	0	0.04	204.48

Table 10: Solution quality and running time of matheuristics on filtered instances of family f4

shown to produce an optimal or near-optimal solution on many instances. We note a slight dominance of *lb-yx* among these matheuristics. Notably, when coupled with filtering all matheuristics are able to produce better solutions in less time. Finally, we showed that the matheuristics are still able to improve initial solutions on instances of up to 700 vertices.

References

[1] Julien Aligon, Matteo Golfarelli, Patrick Marcel, Stefano Rizzi, and Elisa Turricchia. Similarity measures for OLAP sessions. *Knowledge and Information Systems (KAIS)*, 39(2):463–489, 2014.

[2] Claudia Archetti, Ángel Corberán, Isaac Plana, José Maria Sanchis, and M. Grazia Speranza. A matheuristic for the team orienteering arc routing problem. *European Journal of Operational Research*, 245(2):392–401, 2015.

[3] Nicola Bianchessi, Renata Mansini, and M. Grazia Speranza. A branch-and-cut algorithm for the Team Orienteering Problem. *International Transactions in Operational Research*, 25(2):627–635, 2018.

[4] Alexandre Chanson, Ben Crulis, Nicolas Labroche, Patrick Marcel, Verónika Peralta, Stefano Rizzi, and Panos Vassiliadis. The traveling analyst problem: Definition and preliminary study. In Il-Yeol Song, Katja Hose, and Oscar Romero, editors, *Proceedings of the 22nd International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data co-located with EDBT/ICDT 2020 Joint Conference, DOLAP@EDBT/ICDT 2020, Copenhagen, Denmark, March 30, 2020*, volume 2572 of *CEUR Workshop Proceedings*, pages 94–98. CEUR-WS.org, 2020.

[5] Alexandre Chanson, Nicolas Labroche, Patrick Marcel, Stefano Rizzi, and Vincent T'kindt. Automatic generation of comparison notebooks for interactive data exploration. In Julia Stoyanovich, Jens Teubner, Paolo Guagliardo,

#vertices	Algorithm	δ^I_{avg}
400	*vpls-det*	4.04
400	*vpls-random*	4.05
400	*lb-y*	4.05
400	*lb-yx*	4.01
500	*vpls-det*	3.28
500	*vpls-random*	3.28
500	*lb-y*	3.28
500	*lb-yx*	3.28
600	*vpls-det*	2.88
600	*vpls-random*	2.88
600	*lb-y*	2.88
600	*lb-yx*	2.88
700	*vpls-det*	2.57
700	*vpls-random*	2.57
700	*lb-y*	2.57
700	*lb-yx*	2.57

Table 11: Relative improvement over initial solutions for instances of family *f4*

Milos Nikolic, Andreas Pieris, Jan Mühlig, Fatma Özcan, Sebastian Schelter, H. V. Jagadish, and Meihui Zhang, editors, *Proceedings of the 25th International Conference on Extending Database Technology, EDBT 2022, Edinburgh, UK, March 29 - April 1, 2022*, pages 2:274–2:284. OpenProceedings.org, 2022.

[6] Alexandre Chanson, Faten El Outa, Nicolas Labroche, Patrick Marcel, Verónika Peralta, Willeme Verdeaux, and Lucile Jacquemart. Generating personalized data narrations from EDA notebooks. In Kostas Stefanidis and Lukasz Golab, editors, *Proceedings of the 24th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data (DOLAP) co-located with the 25th International Conference on Extending Database Technology and the 25th International Conference on Database Theory (EDBT/ICDT 2022), Edinburgh, UK, March 29, 2022*, volume 3130 of *CEUR Workshop Proceedings*, pages 91–95. CEUR-WS.org, 2022.

[7] I-Ming Chao, Bruce L. Golden, and Edward A. Wasil. A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research*, 88(3):475–489, 1996.

[8] Federico Della Croce, Andrea Grosso, and Fabio Salassa. Matheuristics: Embedding MILP solvers into heuristic algorithms for combinatorial optimization problems. *Heuristics: Theory and Applications*, pages 53–68, 02 2013.

[9] Rui Ding, Shi Han, Yong Xu, Haidong Zhang, and Dongmei Zhang. Quickinsights: Quick and automatic discovery of insights from multi-dimensional data. In Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska, editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 317–332, 2019.

[10] Krista Drushku, Julien Aligon, Nicolas Labroche, Patrick Marcel, and Verónika Peralta. Interest-based recommendations for business intelligence users. *Information Systems*, 86:79–93, 2019.

[11] Ori Bar El, Tova Milo, and Amit Somech. Automatically generating data exploration sessions using deep reinforcement learning. In David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo, editors, *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 1527–1537, 2020.

[12] Michael Engquist. A successive shortest path algorithm for the assignment problem. *INFOR: Information Systems and Operational Research*, 20(4):370–384, 1982.

[13] Matteo Fischetti, Juan José Salazar González, and Paolo Toth. Solving the Orienteering Problem through Branch-and-Cut. *INFORMS Journal on Computing*, 10(2):133–148, 1998.

[14] Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical Programming*, 98(1):23–47, Sep 2003.

[15] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950.

[16] Keld Helsgaun. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[17] Qian Hu and Andrew Lim. An iterative three-component heuristic for the team orienteering problem with time

windows. *European Journal of Operational Research*, 232(2):276–286, 2014.

[18] Andrew B. Kahng and Sherief Reda. Match twice and stitch: a new TSP tour construction heuristic. *Operations Research Letters*, 32(6):499–509, 2004.

[19] Imdat Kara, Papatya Sevgin Bicakci, and Tusan Derya. New Formulations for the Orienteering Problem. *Procedia Economics and Finance*, 39:849–854, 2016.

[20] Pingchuan Ma, Rui Ding, Shi Han, and Dongmei Zhang. Metainsight: Automatic discovery of structured knowledge for exploratory data analysis. In Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava, editors, *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, pages 1262–1274, 2021.

[21] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7(4):326–329, 1960.

[22] Aurélien Personnaz, Sihem Amer-Yahia, Laure Berti-Équille, Maximilian Fabricius, and Srividya Subramanian. DORA THE EXPLORER: exploring very large data with interactive deep reinforcement learning. In Gianluca Demartini, Guido Zuccon, J. Shane Culpepper, Zi Huang, and Hanghang Tong, editors, *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, pages 4769–4773, 2021.

[23] David Pisinger. Where are the hard knapsack problems? *Computers & Operations Research*, 32(9):2271–2284, 2005.

[24] Vincent T'kindt and Jean-Charles Billaut. *Multicriteria Scheduling - Theory, Models and Algorithms (2. ed.)*. Springer, 2006.

[25] Theodore Tsiligirides. Heuristic Methods Applied to Orienteering. *Journal of the Operational Research Society*, 35(9):797–809, 1984.

[26] Anthony Wren and Alan Holliday. Computer scheduling of vehicles from one or more depots to a number of delivery points. *Operational Research Quarterly (1970-1977)*, 23(3):333–344, 1972.

[27] Qinxiao Yu, Kan Fang, Ning Zhu, and Shoufeng Ma. A matheuristic approach to the orienteering problem with service time dependent profits. *European Journal of Operational Research*, 273(2):488–503, 2019.