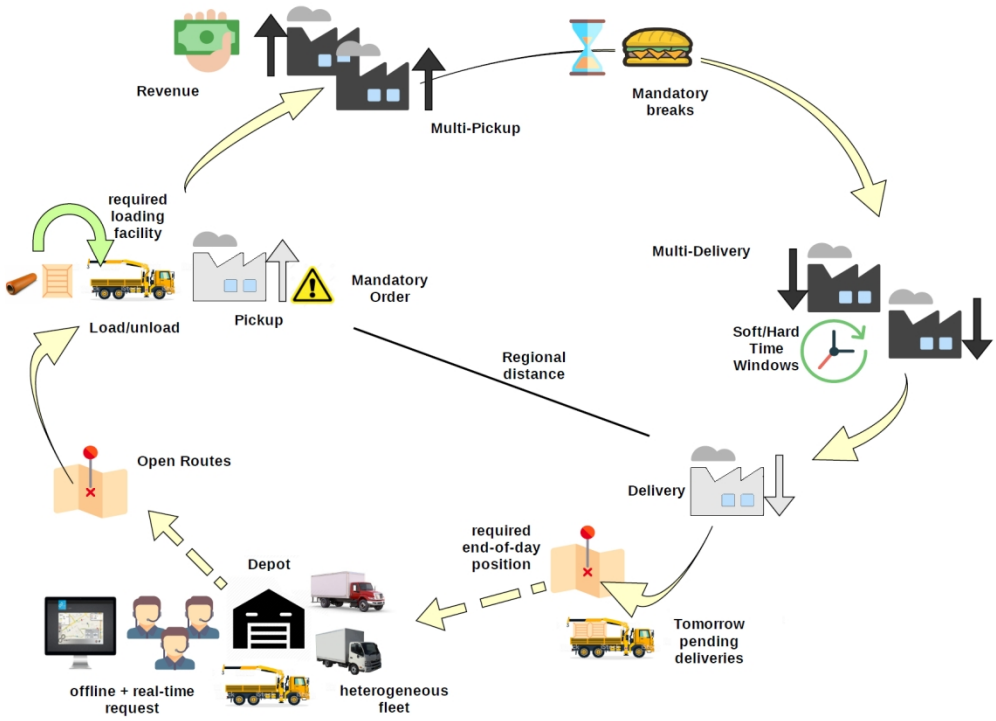
A two-level local search heuristic for pickup and delivery problems in express freight trucking

NETWORKS
An International Journal

A two-level local search heuristic for pickup and delivery problems in express freight trucking

Journal:	*Networks*
Manuscript ID	Draft
Wiley - Manuscript type:	Special Issue Article
Date Submitted by the Author:	n/a
Complete List of Authors:	De Giovanni, Luigi; Università degli Studi di Padova, Dipartimento di Matematica "Tullio Levi-Civita" Gastaldon, Nicola; Università degli Studi di Padova, Dipartimento di Matematica "Tullio Levi-Civita" Sottovia, Filippo; Trans-Cel snc
Keywords:	Multi-attribute VRP, Multi-pickup and Multi-delivery, Express Freight Transportation, Tabu Search, Parallel implementation, Column Generation
Abstract:	We consider a multi attribute vehicle routing problem inspired by a freight transportation company operating a fleet of heterogeneous trucks. The company offers an express service for requests including multiple pickup and multiple delivery positions spread in a regional area, with associated soft or hard time windows often falling in the same working day. Routes are planned on a daily basis and re-optimized on-the-fly to fit new requests, taking into account constraints and preferences on capacities, hours of service, route termination points. The objective is to maximize the difference between revenue from satisfied orders and operational costs. The problem mixes attributes from both intercity less-than-truckload and express couriers operations, and we propose a two-level local search heuristic. The first level assigns orders to vehicles through variable neighborhood stochastic tabu search; the second level optimizes the route service sequences. The algorithm, enhanced by neighborhood filtering and parallel exploration, is embedded in a decision support tool currently in use in a small trucking company. Results have been compared to bounds obtained from a mathematical programming model solved by column generation. Experience on the field and test on literature instances attest to the quality of results and the efficiency of the proposed approach.



The MAVRP under study

A two-level local search heuristic for pickup and delivery problems in express freight trucking

Luigi De Giovanni[♭], Nicola Gastaldon[♭,♯], and Filippo Sottovia[♯]

[♭]Dipartimento di Matematica "Tullio Levi-Civita", Università di Padova, Via Trieste 63, 35121 Padova (Italy), {luigi@math, nicola.gastaldon@phd}.unipd.it

[♯]Trans-Cel, Via L. da Zara 33, 35020 Albignasego (Italy), info@trans-cel.it

Abstract

We consider a multi attribute vehicle routing problem inspired by a freight transportation company operating a fleet of heterogeneous trucks. The company offers an express service for requests including multiple pickup and multiple delivery positions spread in a regional area, with associated soft or hard time windows often falling in the same working day. Routes are planned on a daily basis and re-optimized on-the-fly to fit new requests, taking into account constraints and preferences on capacities, hours of service, route termination points. The objective is to maximize the difference between the revenue from satisfied orders and the operational costs. The problem mixes attributes from both intercity less-than-truckload and express couriers operations, and we propose a two-level local search heuristic. The first level assigns orders to vehicles through a variable neighborhood stochastic tabu search; the second level optimizes the route service sequences. The algorithm, enhanced by neighborhood filtering and parallel exploration, is embedded in a decision support tool currently in use in a small trucking company. Results have been compared to bounds obtained from a mathematical programming model solved by column generation. Experience on the field and test on literature instances attest to the quality of results and the efficiency of the proposed approach.

keywords: Multi-attribute VRP, Multi-pickup and Multi-delivery, Express Freight Transportation, Tabu Search, Parallel implementation

1 Introduction

Modern supply chains ask for highly customized medium- and long-haul freight transportation which is often convenient for trucking, able to provide the required service flexibility. In order to profitably fulfill these requests, trucking companies need a clever setup of daily operations and ask for computerized support tools. This gives rise to new and interesting routing problems, defined by the carriers according to their specific business models and operational contexts. The Vehicle Routing Problem (VRP), in its basic version, considers a set of clients and a set of vehicles, and asks for devising a set of minimum cost routes starting and ending at a central depot, with the constraint that each client has to be visited by one vehicle. Clearly, this statement rarely fits real-world applications, and several versions of VRP include constraints on capacity, service time windows, pickup and delivery precedence etc. (see [25], among others). One of the trends in recent research on VRPs is to take more and more attributes simultaneously into account, giving rise to the so called Multi-Attribute (or Rich) VRP (MAVRP [15, 26]).

We consider a problem inspired by the daily operations at Trans-cel, a small trucking company based in the Venice area (Italy) and targeting a market of highly customized freight transportation demands. The company owns a fleet of heterogeneous trucks and offers an express pickup-and-delivery service for freight of various size to be moved in a regional area of up to about 500 kilometers from the company premises. Each request specifies one or more pickup and delivery points with related hard or soft time windows and other operational preferences. Customers ask for a just-in-time service to be fulfilled in the same working day or two consecutive days. A revenue is associated to orders, together with penalties for, e.g., soft time windows violations. Routes determine the sequence of pickup and deliveries for each truck: they start from the last position of the day before and are subject to several operational constraints or preferences on, e.g., hours-of-service, maximum duration, ending position. The status of orders, tasks, vehicles and

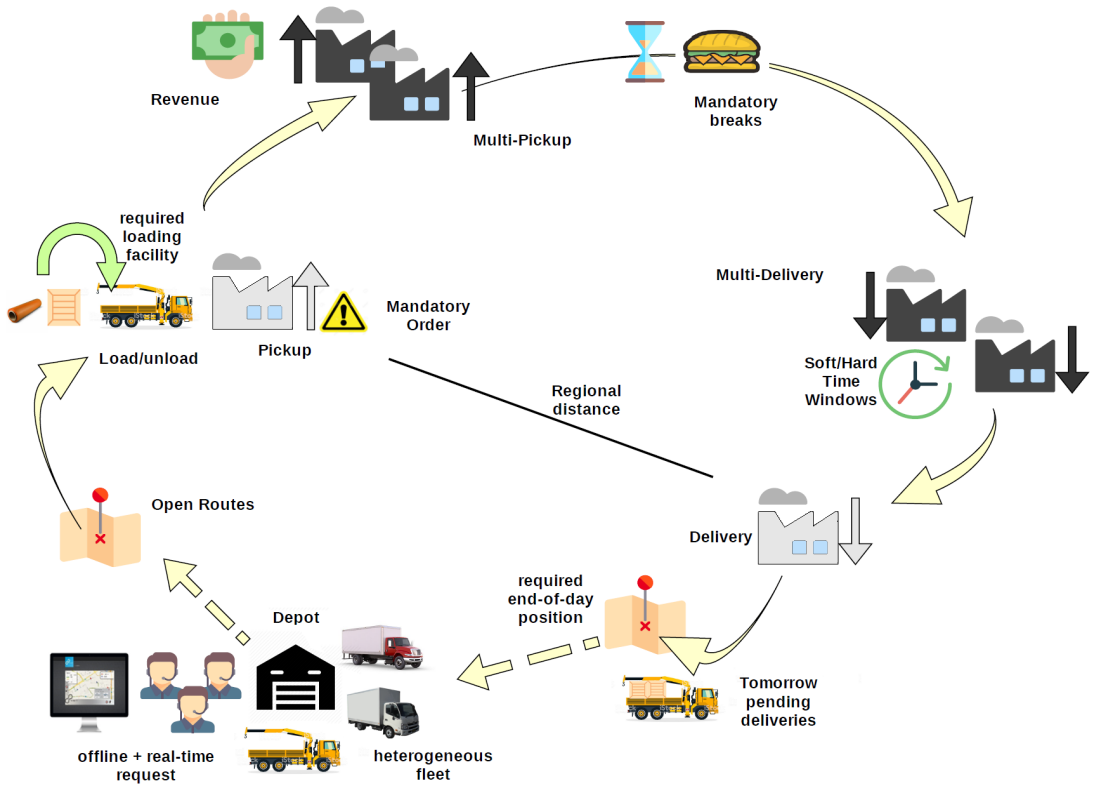


Figure 1: The MAVRP under study

road network is available from a decision support platform able to make real-time data available [10] and, based on this information, the problem is to determine a set of routes that maximize the difference between the revenue from satisfied requests and route costs, taking operational constraints and preferences into account.

The problem belongs to the class of MAVRP and has to be solved in static settings to define initial daily routes, and in dynamic settings to respond to a variety of such events as on-line issued requests, delayed tasks, truck failures, road network modifications. The main attributes of the MAVRP under study are illustrated in Figure 1.

The scope of this paper is the development of an algorithm to solve the MAVRP stated above and support the operations management office during the daily and real-time assignment of requests to routes. Section 2 reviews some relevant literature on MAVRPs. The problem is described in Section 3, providing the details of its attributes and pointing out its peculiarities. Section 4 reports the proposed solution approach: it starts from a preliminary variable neighborhood search presented in [9] and improves the definition

of initial solution, neighborhoods, score function and move filtering. Towards further speedup, Section 5 presents a faster solution evaluation procedure integrated with destroy and repair phases, together with parallel implementations of the overall algorithm. In order to provide a bound to assess the result of the proposed heuristic, we have developed a column generation procedure, as described in Section 6. The solution algorithm has been implemented in different variants and computational results on real-world instances and on literature benchmarks are discussed in Section 7. Section 8 concludes the paper with some final remarks.

2 Literature Review

For the static version of MAVRPs, literature proposes both exact and heuristic approaches. Exact methods mostly solve set partitioning formulation by decomposition based on branch-and-price (B&P), able to flexibly deal with several attributes in the pricing problem. Among others, [21] solves a basic pickup and delivery problem with time windows (PDPTW) using valid inequalities to improve the bounds provided by column generation, and a relaxed version of the pricing problem to improve efficiency. The method presented in [2, 3] solves different variants of VRP by combining dual-ascent procedures and a cut-and-column generation algorithm. In [8], different attributes (multiple capacities, hours-of-service regulations, open routes etc.) are combined to cope with a real-world VRP, solved by a column generation approach using a bounded bidirectional dynamic programming algorithm for the pricing problem. A branch-and-price algorithm is devised in [6] for a pickup and delivery problem with multiple depots, heterogeneous vehicles and soft time windows. Concerning heuristic approaches, [15, 17, 26] overview several flexible heuristics able to adapt to the VRP definitions arising in different settings and to handle a variety of objectives and side constraints. Metaheuristic approaches, such as Tabu Search, Genetic Algorithms, Ant Colony Optimization etc., are very popular for solving MAVRPs [7]. The research presented in [27, 28] hybridizes genetic algorithms and local search, and proposes a unified framework for solving a wide range of large-scale vehicle routing problems with time windows, route-duration constraints and further attributes related to client assignment. Towards dynamic settings, two main strategies are proposed

to adapt running routes to include on-line issued requests: reactive policies and anticipatory algorithms. The former consider only the new request and the pending ones already in the routes, the latter exploit, if available, stochastic information on future requests. Among reactive policies, a more general approach is re-optimization, that can be used when the dynamism is low enough and allows continuously or periodically applying fast heuristic for static settings to up-to-date vehicles, requests and network status. Detailed surveys of methods for VRPs in dynamic settings can be found in [11, 16].

3 Problem statement and operational settings

We provide a general statement of the MAVRP under study. It takes several attributes simultaneously into account, in order to comply with the real-life daily operations in small trucking companies offering express transportation services. Attributes are related to positions, vehicles, customer requests and routes. Positions represent the pickup and delivery points and include the depot, corresponding to the main facility of the transportation company. For each ordered pair (i, j) of positions, a distance d_{ij} and a travel time t_{ij} are given. They depend on the road network status and up-to-date information can be retrieved from, e.g., web services or specialized applications like [12, 24].

Vehicles are heterogeneous and characterized by specific loading capacity and operational cost. In particular, denoting by V the set of vehicles, each $v \in V$, defines a constraint on the maximum volume U_v^1 and a maximum weight U_v^2 that can be loaded on it. Since freights are normally trucked on pallets, U_v^1 and U_v^2 provide a reliable estimation of the actual vehicle loading capacity. We remark that, in our problem, capacity relates to on board freight, meaning that deliveries make capacity available again for following pickups. Moreover, vehicle-specific variable and fixed operational costs are provided. The former is a cost per distance unit C_v^U, taking into account several factors among which fuel consumption, vehicle mortgage, maintenance costs etc. The latter represents the fixed cost C_v^F incurred if vehicle v is deployed. Some vehicles are equipped with special loading facilities, such as crane, tail lift, side doors etc.

Customer requests (or orders) correspond to the demands of pickup and delivery services. Orders specify a set of pickup and delivery tasks. In particular, an order may include

more than one pickup or delivery task (multi-pickup / multi-delivery orders). In this case more pickup tasks and/or more delivery tasks are defined, and all the pickup tasks must be executed before all the delivery tasks by the same vehicle. This precedence constraint comes from real-world scenarios where the internal policy of transportation companies requires all the pickups to be on board before any delivery, which better guarantees the overall accomplishment of the order in charge. Each task is defined in terms of position, time window, duration, size of the freight (volume and weight). Time windows can be hard or soft. The former imply that the task must be executed within the time limit, otherwise the solution is not feasible; the latter can be violated at the cost of a penalty proportional to the time window violation, taking into account that early service is not allowed (in case a vehicle arrives early at the task position, it waits until the beginning of the time window). A revenue P_o is associated to each order $o \in O$, O denoting the set of orders, together with a priority. We have three possible priority levels: mandatory orders (subset O^M) must be fulfilled in any feasible solution; urgent orders (O^U) define a penalty in case they are note executed (the penalty for $o \in O^U$ is denoted by L_o,); normal orders (O^N) can be rejected at no penalty. Finally, each order may specify the required loading facilities, if any, affecting the set of vehicles it can be assigned to. We remark that, due to the nature of customer requests demanding for an express delivery service, the tasks' time windows of one order fall in at most two consecutive days. As a consequence, the set of issued orders that are relevant for planning in a given day, hence included in O, are the ones issued either the same day or the day before. The time windows of the related tasks fall in the day before (we call them *yesterday tasks*), in the same day (*today tasks*) or in the next day (*tomorrow tasks*).

A route is defined by a sequence of pickup and/or delivery tasks assigned to a specific vehicle. All the tasks of the same order must be assigned to the same vehicle. In order for a route to be feasible, for each assigned order, all pickups tasks must precede all deliveries in the sequence. The planning horizon is one working day and routes are open: each vehicle starts from the position of the last task executed the day before, in case with pending *yesterday* tasks, it services *today* tasks, and it finishes at the position of the last today task or, in case, of the last pending *tomorrow* task. Moreover, some routes are constrained to reach a prescribed *end-of-day* position, e.g. for maintenance issues. All

routes are subject to hours-of-service regulation: they must include a short break of 45 minutes after 4 hours and 30 minutes driving, and a long break of 9 hours after either 9 hours driving or 13 hours working (driving or attending pickup and delivery tasks).

In addition, further attributes come from preferences aiming at improving the quality of service. In particular we consider preferences on: maximum route duration (e.g. to accommodate specific drivers' requests, or balance drivers' workload in the long term); including subsets of orders in the same route (e.g. different orders of a same customer); favorite route ending position; assignment of orders to specific vehicles (e.g. for truck facilities that may help pickup or delivery tasks, although not necessary); minimum number of vehicles available at the depot at the end of the planning horizon.

Summarizing, we define the Express Pickup and Delivery in freight Trucking problem (EPDT) as follows. Given the orders involving a one-day planning horizon, the initial vehicle status in terms of position and on-board pending *yesterday* tasks, as well as the travel distance and time between positions, determine a set of daily routes to service the today tasks of a subset of selected orders. Each route must be operated by one vehicle and feasible according to constraints described above, namely: vehicle capacity, precedence between pickup and delivery tasks, hard time windows, execution of mandatory orders, loading facilities requirements, hours-of-service regulations, prescribed route ending position (if provided). The objective function is to maximize the profit, taking preference matching into account. The profit is defined as the difference between the total revenue from the orders assigned to routes, and the costs associated to vehicle operations and penalties for missed mandatory orders and soft time windows violation.

The operation management office has to solve EPDT on a daily basis in static settings, in order to assign issued orders and set the initial routes before the deployment of vehicles. In addition, operation managers need support in dynamic settings, to modify routes in operation according to emerging events that impact the status of vehicles, orders or road networks. In this case, up-to-date information is made available by real-time-data sharing tools (e.g., [10, 12, 24]). For example, new orders involving *today* tasks may arrive during the workday: they have to be assigned to the most convenient route, taking into account vehicle current position and all the on-board pending tasks. Similarly, in case of vehicle failures, its orders have to be re-assigned to other vehicles, taking into account

that suitable loading facilities may be required by transshipment. Further dynamic route modifications may be triggered by changes in the road network (deviations, traffic congestion etc.) impacting distances and travel times, or by tasks duration longer (or shorter) than estimated.

From the statement of EPDT, we point out some interesting peculiarities. One is the definition of multi-pickup and multi-delivery orders, representing customer needs that emerge in express freight trucking, and introducing specific constraints in route operations. Moreover, EPDT mixes features from different domains. On the one hand, pickup and delivery positions are spread in a regional area and routes are typical of long-distance intercity less-than-truckload couriers (LTL). On the other hand, orders require a just-in-time service often defining both pickup and delivery as *today* tasks: this, together with open routes, strongly limits, e.g., the impact of consolidation, typical of LTL settings, and introduces features similar to express couriers operations in urban contexts.

4 A two-level local search heuristic

The design of an algorithm to solve EPDT has to fit both the problem statement and the operational context. The problem statement includes several attributes and ask for a flexible approach able to handle them simultaneously. Moreover, concerning dynamic settings, we observe that the frequency of events is relatively small compared to travel times or task duration. This configures relatively-low dynamism scenarios, suitable for reaction policies based on re-optimization, which require an efficient method to solve EPDT under up-to-date information. The algorithm described in this section aims at supporting the operations manager during the initial planning and the real-time route re-optimization.

Due to the inner complexity of the problem and the operational scenarios it supports, a meta-heuristic approach is suitable, and we start from the tabu search approach proposed in [9] for a MAVRP similar to EPDT. As we will detail in the following, the original approach is enhanced by an improved procedure to determine an initial solution and several methodological and implementation devices to gain efficiency in the local search while preserving the quality of the solutions (e.g. generalized and refined neighborhood

definition and filtering, parallel neighborhood evaluation, fast solution evaluation).

Solving EPDT involves, at least conceptually, three decision degrees: (i) assigning orders to vehicles, (ii) for each vehicle, sequencing tasks of the assigned orders, and (iii) determining the best vehicle route through task positions. As in [9], we decompose the problem into two hierarchical levels: the first deals with (i) and determines order-to-vehicle assignments, which provide the input for the second level, dealing with (ii) and (iii), that is, the optimization of vehicle routes. The first level implements a heuristic based on tabu search: it starts from an initial solution (current solution) and it generates a set of neighbor solutions by perturbing the current one; the solutions are evaluated according to a score function and the best neighbor is chosen as the new current solution; the process iterates until a termination condition occurs (e.g. maximum number of iterations). The incumbent solution is updated each time a better current solution is generated. In order to avoid cycling, a tabu list stores information on the last visited solutions, so that they are excluded from eligible neighbors. In the decomposition schema, each neighbor solution represents a first-level decision and determines, for each vehicle, the set of orders assigned to it. In order to evaluate this solution, the second-level decisions are made, since the score function depends on the route run by each vehicle to service assigned orders. Towards vehicle routes optimization, a second-level heuristic based on local search is devised. In other words, the first-level tabu search explores the space of all the possible assignments of orders to vehicles, whereas at the second level the search space covers task sequences and related vehicle routes. The schema sketched above is very general and we need to specify its components, in particular: the score function, the local search procedure that evaluates it, a method to compute an initial order assignment, neighborhood and tabu list.

4.1 The score function

The objective of EPDT is maximizing the profit from daily orders. Given a solution s, let $O(s)$ be the set of orders assigned to a vehicle, and $R(s)$ the set of vehicle routes. Each route $r \in R(s)$, run by a vehicle $v(r)$, defines the sequence of tasks and, as a consequence, their starting time, computed from travel times between positions and task duration.

With reference to the notation defined in Section 3, the profit $G(s)$ is defined as

$$G(s) = \sum_{o \in O(s)} P_o - \sum_{r \in R(s)} [C(r) + W(r)] - \sum_{o \in O^U \setminus O(s)} L_o \quad (1)$$

where

- $C(r)$ is the cost of route r: $C(r)$ includes the fixed deployment cost C_v^F and the variable cost depending on C_v^U and on the length of r up to the last *today* task;
- $W(r)$ is the cost due to penalties for soft time windows violations on r, if any, as derived from task starting times.

Notice that $C(r)$ does not take *tomorrow* tasks into account, since the related cost will be considered by the next-day planning. However, $v(r)$ will execute tomorrow tasks and next-day costs will depend on the final position of $v(r)$. In order to drive the search towards solutions accounting for next-day costs, we introduce in the score function a *prospective route cost*, as a proxy of the variable next-day cost. We define $A(r) = C_v^U \, l^{next}(r)$, where $l^{next}(r)$ is the length of the shortest (according to distances d_{ij}) route starting from the last-served today task and visiting all next-day tasks assigned to $v(r)$.

We recall that, in order for s to be feasible for EPDT, each route in $R(s)$ must satisfy several constraints. In particular, we need $O^M \subseteq O(s)$, that is, all mandatory orders are assigned. For the sake of an effective exploration of the order assignment space, we allow visiting solutions with missing mandatory orders, which is penalized in the score function through a component $B(s) = \sum_{o \in O^M \setminus O(s)} P_o$.

In addition, EPDT defines several preferences, that we model as soft constraint by including them in the score function. In particular, we define the following measures associated to s and $r \in R(s)$:

- $I_D(s)$: it sums up, over all the subsets Q of orders to preferably be in a same route, the number of vehicles servicing Q, minus one;
- $I_E(s)$: number of missing vehicles at the depot with respect to the minimum required;
- $I_F(r)$: number of orders assigned to the preferred vehicle;
- $I_H(r)$: boolean indicator taking value 1 if r has its preferred end-of-day position, 0

otherwise;

- $I_J(r)$: extra-time with respect to the preferred maximum duration of r.

Summarizing, a solution s is evaluated by the following score function:

$$Z_1(s) = G(s) - \sum_{r \in R(s)} w_A A(r) - w_D I_D(s) - w_E I_E(s) + \\ + \sum_{r \in R(s)} \big(w_F I_F(r) + w_H I_H(r) - w_J I_J(r) \big) - M\, B(s) \qquad (2)$$

where w_X, for each indicator I_X, and M are parameters to be calibrated on the field, with M taking a big value to suitably penalize infeasible solutions, and w_X taking smaller non negative values to foster preference matching.

4.2 Solution evaluation and second-level heuristic

Given the order-to-vehicle assignments, most of the indicators taking part in the score function (2) depend on vehicle route, i.e., the sequence of tasks. This asks for solving, for each vehicle, the following *Intra-route optimization* problem, defined as follows: given the orders assigned to a vehicle, determine a feasible route r that minimizes the following function:

$$Z_2(r) = C(r) + W(r) + w_A A(r) - w_F I_F(r) - w_H I_H(r) + w_J I_J(r) + w_E \frac{\bar{N}_0}{|V|} \bar{I}_E(r) \qquad (3)$$

where: $\bar{I}_E(r)$ is a Boolean indicator taking value 1 if r does not end at the depot, 0 otherwise; $\bar{N}_0 = \max\{0, N_0 - \tilde{N}_0\}$, N_0 being the preferred number of vehicles at the depot, and $\tilde{N}_0$ being the number of vehicles with depot as end-of-day position. Notice that all the components of (3) are included in the score function (2) but the last one, which is related to $I_E(s)$: it is intended to penalize solutions with a small number of vehicles ending at the depot taking into account the weight of $I_E(s)$ in $Z_1(s)$. Moreover, the factor $\frac{\bar{N}_0}{|V|}$ increases the relevance of this component of $Z_2(r)$ when the number of vehicles is relatively small with respect to $\bar{N}_0$ (the additional number of vehicles required at the depot besides the one guaranteed by routes' feasibility).

Given a route r, $Z_2(r)$ is evaluated by computing the starting time for each task

according to travel times t_{ij}, task duration, earliest starting time from the time windows, and inserting short and long compulsory breaks when necessary.

Intra-route optimization is a hard problem and, moreover, it has to be performed for each assignment generated by the first-level decisions. We thus propose a second-level heuristic based on local search: it starts from an initial route and generates neighbor solutions, evaluated by (3); the best neighbor is chosen as the new current solution, and the process is iterated until no improving neighbor exists. We now describe its components, taking into account that, as we will detail later, the second-level local search is triggered by:

- the assignment of one new order o to a vehicle route r;
- the removal of one or more orders from r, with, in case, the simultaneous assignment of a new order;
- the assignment to r of a set of orders coming from another route.

Moreover, we assume that the affected route r is feasible, that is, it satisfies all EPDT constraints (precedence, hard time windows, capacity, loading facilities and hours-of-service), but, in case, the one on mandatory orders (which in fact depends on the overall assignment).

In the following, a route r is defined a sequence $t_0, t_1, \ldots, t_n, t_{n+1}$, where t_i is the i-th task, and t_0 and t_{n+1} are the initial and final dummy tasks.

Initial task sequence

In case a set of orders is removed from r, the initial route is the same r without the tasks corresponding to removed orders: this always yields a feasible route, since r is feasible.

If a new order o is assigned to r, we obtain an initial route including o as follows. We recall that o may include one or more pickup and delivery tasks. For each i and j such that $0 \leq i \leq n$, $0 \leq j \leq n$ and $i \leq j$, insert all pickup tasks of o after t_i and all delivery tasks after t_j and evaluate (3), keeping the sequence providing the best value as the initial route. In order to speed-up convergence of the following local search, tasks are inserted after i or j according to a nearest neighbor criterion. Notice that all insertions yielding an infeasible route are discarded so that the procedure ends up with either a feasible route

including the new order o, or no feasible route. The latter case prevents applying further local search.

If more than one new order is assigned to r, coming from a different route $\bar{r}$, a procedure similar to the previous one provides the initial solution. In this case, the pickup (resp. delivery) tasks of all orders are inserted after t_i (resp. t_j), following the sequence they had in $\bar{r}$.

Second-level neighborhoods

We define neighborhoods by the following moves:

- *task insertion*: for each i, j such that $i \neq j$, insert t_i after t_j;
- *task swap*: for each i and j such that $i < j$, swap t_i and t_j.

The moves applied to generate neighbor solutions are different depending on o being fixed or not: we say that o is a *fixed order* if the vehicle it is assigned to cannot be changed. This is the case of orders including *yesterday* tasks: their assignment comes from yesterday planning leaving some pending *today* tasks already on board. Other fixed tasks may derive from special operations manager requirements. If o is fixed, we obtain neighbor solutions from both task insertions and task swaps, otherwise, only task swaps are applied. In order to preserve route feasibility, moves are discarded if the resulting route is not feasible.

Overall outline of the second level heuristic

In Algorithm 1 we provide a high-level outline to summarize the second-level procedure, that we call $L2$. For sake of simplicity, we illustrate the case of a single order insertion o in a route r (the pseudo-code is similar for multiple orders insertion or for single order removal). An optimized route containing all the tasks in o is returned. Variable P (resp. D) represents the set of all pickup (resp. delivery) tasks in o. The procedure determines the initial task sequence is encoded by Steps 3 to 14: it sequentially inserts tasks in P or D after a selected task t_i or t_j already in r according to a nearest neighbor heuristic, and it uses a set R to collect all feasible routes generated, if any, among which the one with better score Z_2 is chosen. Starting from this route, a local search procedure improves the

sequence of all the tasks in r, using the second level neighborhoods described above and Z_2 as score function. If R is empty (see Step 11), the procedure stops and returns no feasible route.

```
1  def L2(r, o):
2      P = getPickups(o), D = getDeliveries(o), R = ∅;
3      foreach (t_i, t_j) ∈ r × r : t_i ≤ t_j do
4          nearest_neighbor(t_i, P);
5          nearest_neighbor(t_j, D);
6          r̄ = concatenate([t_0, ..., t_i], P, [t_{i+1}, ..., t_j], D, [t_{j+1}, ..., t_{|r|}]);
7          if r̄ is feasible then
8              R = R ∪ r̄;
9          end
10     end
11     if R is empty then
12         return Infeasibility;
13     end
14     best = arg max{Z_2(r) : r ∈ R};
15     if o is fixed then
16         best = local_search(best, [task_insertion, task_swap], Z_2);
17     end
18     else
19         best = local_search(best, task_swap, Z_2);
20     end
21     return best;
22 end
```

Algorithm 1: Second-level heuristic

4.3 First-level tabu search

A tabu search procedure explores the space of possible order-to-vehicle assignments: its components are detailed in the following.

Initial order-to-vehicle assignment

A Best Insertion (BI) constructive heuristic provides the initial solution for the tabu search. First, fixed orders are assigned to their vehicle, together with initial and final dummy tasks. Moreover, in order to deal with end-of-day position, a further dummy task is added with suitable time window at the end of the planning day: its position is the one prescribed or preferred for the vehicle, if any, or the depot otherwise; in case of prescribed route ending position, the task is mandatory, normal otherwise. Let $\bar{O}$ be the set of non assigned orders, initially equal to O. For each $o \in \bar{O}$ and each vehicle v, we tentatively assign o to v and compute the score function (2) of the partial solution under construction, selecting the (o, v) pair corresponding to the best score. We remove o from $\bar{O}$ and iterate until $\bar{O}$ is empty.

The procedure is an enhanced version of the BI presented in [9], where orders are sorted by increasing distance from the depot an assigned to the vehicle that optimizes the overall score of the partial solution under construction. The new BI procedure has a greater computational complexity but, according to computational evidence, it better trades off efficiency and quality of the initial solution.

Notice that each tentative assignment triggers the execution of the second level heuristic to evaluate (3) for the route r associated to v. If intra-route optimization does not provide any feasible route, then the assignment is discarded, so that following tentative assignments always start from a feasible route, according to the assumptions of the second-level heuristic. For the sake of tabu search, a dummy vehicle collects all the orders that remain unassigned at the end of the initial heuristic.

Tabu search neighborhoods and move filtering

With the aim of perturbing order-to-vehicle assignments, the first-level tabu search defines moves involving non-fixed orders currently in charge of different routes, including the ones in the dummy vehicle. We recall that fixed orders and dummy tasks cannot be assigned to different vehicles. We consider three neighborhoods.

A first type of moves is the *single order relocation* (1R), where an order is moved from its current route to a new route. Denoting by k the number of non-fixed orders

and by v the number of vehicles, including the dummy one, the size is $O(k \cdot v)$, since the neighborhood contains a solution for each of the k non-fixed orders and each of the $v-1$ alternative vehicles (including the dummy one);

The second neighborhoods originates from *two-orders swaps* (2S), consisting in swapping the assignment of two orders in two different routes. The neighborhood contains a solution for each pair of distinct orders assigned to distinct vehicles, therefore its size is $O(k^2)$.

A third neighborhood is generated by *multiple relocation* moves (mR): they generalize 1R moves as they simultaneously displace up to m orders in a same route r_1 to a different route r_2. We have a neighbor solution for each subset of at most m orders currently on board of the same vehicle and for each of the v alternative vehicle, hence the size of the mR neighborhood is $O(k^m \cdot v)$.

Notice that 1R and 2S corresponds to neighborhoods defined in [9], whereas mR is a generalization of the 2-order-relocation neighborhood considered in the same work. Moreover, we dropped the chain-shift neighborhood presented in [9], after verifying poor performances in fine-grained tests: the algorithm saves a considerable amount of computational effort while preserving the quality of the solutions.

The evaluation of the solutions generated by the above moves involves the second-level heuristic, called twice to determine a new sequence for the two routes affected by each move. The overall neighborhood evaluation may be computationally expensive, in particular for mR, due to the large size. Therefore we propose a granular exploration of mR (which generalizes and refines the one introduced in [9]) by limiting moves to the ones involving most promising order subsets, selected on a distance criterion: the more orders are "close" to each others, the more they are likely to be conveniently accommodated in the new route. More formally, we say that two orders o_1 and o_2 are close to each other if at least a threshold fraction ρ of the tasks of o_1 are within a threshold distance δ from at least one of the tasks of o_2, and vice versa. ρ and δ are parameters to be calibrated. The definition is symmetric and we associate to the route r an undirected graph where each vertex represents a non-fixed order of r, and each edge corresponds to a pair of close orders. Based on this graph, we filter the mR neighborhood by applying moves only to subsets of orders corresponding to cliques of cardinality at most m in the graph.

Exploration strategy

Neighborhoods are sorted by increasing size (1R first, then 2S and mR) and explored according to a variable neighborhood strategy, that is, at each iteration a neighborhood is explored if and only if the previous one does not provide an improving solution. As soon as the best solution of a neighborhood (steepest descent) improves over the current score function, it is chosen as the next solution in the tabu search trajectory. If an improvement is not generated by any of the neighborhoods, the choice of the next solution is either deterministic or stochastic. Deterministic exploration selects the best non-improving solutions. Stochastic exploration chooses the new current solution at random among the best five generated by any explored neighborhood, with probability proportional to the score function (roulette wheel selection). In this case, the exploration can be repeated to obtain hopefully better solutions.

Tabu list and termination criteria

To prevent search space exploration from cycling through the same order-to-vehicle assignments, the tabu list stores the last T moves yielding selected current solutions: a neighbor is declared tabu and temporarily forbidden if it moves at least one order to a vehicle it was assigned in the last T iterations.

The first-level tabu search stops as soon as a maximum number M_1 of iterations without improvement or a maximum total number M_2 of iterations is reached.

Overall outline of the first level heuristic

In Algorithm 2 we provide a pseudo-code of the first-level procedure, denoted as $L1$ and corresponding to the overall approach proposed for EPDT. Given an empty (or even partial) solution s, $L1$ calls procedure BI to obtain a complete initial solution (Step 2). Notice that BI relies on the second level heuristic $L2$ to for evaluating the cost and the feasibility of the tentative routes it considers. Then, Steps 3 and 4 initialize the center and the best solution, together with the total iteration counter k, the non-improving iteration counter l and the tabu list T.

Steps 5 to 35 are the main tabu search loop that explores the EPDT solution space.

At each iteration, all the neighbors of the current center solution are visited by the inner loop (Steps 9 to 24), and a pool of the best 5 solutions, according to Z_1 score function, is maintained (Steps 7 and 22). The inner loop iterates on the neighborhood type N (set to 1R, 2S or mR in the order) and implements the variable neighborhood exploration strategy: each inner iteration generates all the neighbor solutions of type N, taking the tabu list T into account and calling the second level heuristic L_2 to evaluate the cost and check the feasibility of the order-to-vehicle assignments associated to each solution (Step 10).

Each time a neighbor solution is generated with better Z_1 score than the current center one, the inner variable neighborhood loop is broken and the improving solution becomes the center for a new tabu search iteration (Steps 11 to 20), after updating the tabu list (Step 34) and, in case, the best solution found so far (Step 16). On the contrary, if the inner loop generates no improving solution (flag *improved* remains false), the center of the next tabu search iteration is chosen from the pool: the best one in case of deterministic exploration, or the one determined by roulette wheel selection in case od stochastic exploration (Steps 27 to 32). The main loop stops after matching the termination criteria based on the tabu search iteration counters l and k (Step 5).

5 Speeding-up the search

The two-level local search heuristic described above determines the initial daily routes to interactively support the operations manager, and re-optimizes current routes as a reaction to dynamic events. In order to shorten running times, we propose a faster second level heuristic that reduces the overall computational effort, and parallel implementations of the tabu search.

5.1 Fast second-level heuristic and infeasible route fixing

As already observed, the second-level local search heuristic is an as critical as computationally expensive task. With the aim of reducing running times, we speed-up the evaluation of the score function by separately sequencing today and tomorrow tasks. The

```
1  def L1(s, explorationStrategy):
2      s = BI(s, L2);
3      center = best = s;
4      k = l = 0, T = [center];
5      while l < M_1 and k < M_2 do
6          k = k + 1;
7          pool = [ ];
8          improved = false;
9          for N in [1R, 2S, mR] do
10             neighbors = generate_neighbors(center, N, T, L2);
11             n̄ = arg max{Z_1(n) : n ∈ neighbors};
12             if Z_1(n̄) > Z_1(center) then
13                 center = n̄;
14                 l = 0;
15                 improved = true;
16                 if Z_1(n̄) > Z_1(best) then
17                     best = n̄;
18                 end
19                 break
20             end
21             else
22                 pool = take_best_5 (pool ∪ neighbors, Z_1);
23             end
24         end
25         if not improved then
26             l = l + 1;
27             if explorationStrategy is deterministic then
28                 center = arg max{Z_1(n) : n ∈ pool};
29             end
30             else
31                 center = roulette_wheel(pool);
32             end
33         end
34         update_tabu_list(center, T);
35     end
36     return best;
37 end
```

Algorithm 2: First-level heuristic

sequence of tomorrow tasks is relevant in the score function for the evaluation of the prospective tomorrow cost, which, moreover, takes only distance into account. Observing that today tasks precedes tomorrow task in any feasible sequence, we decompose a route r in two sub-sequences including, respectively, today and tomorrow tasks. The second level heuristic limits the local search procedure to today tasks, and, then, a nearest neighbor heuristic sequences tomorrow tasks: starting from the last today task, the next task is the one compliant with pickup and delivery precedence, and minimizing the distance from the current task. This allows reducing the computational effort but has the drawback of yielding possible infeasibilities in the route, e.g. the violation of hard time windows or capacity constraints, although limited to tomorrow tasks. With the aim of taking the computational benefits, we devise the following phases:

1. **Optimize**: we run the first-level tabu search calling the second-level heuristic with fast route sequencing, obtaining a solution s.
2. **Check**: for each $r \in R(s)$, compute tomorrow tasks' starting times and capacity needs and check feasibility. If all routes are feasible, stop and return s. Otherwise, let t_i^r be the first task of r violating some constraints.
3. **Destroy**: for each infeasible route r, remove all the orders containing task t_i^r or any of the following tasks, and assign them to the dummy vehicle.
4. **Repair**: run the first-level tabu search calling the second level heuristic with the usual route sequencing (local search and feasibility check extended to tomorrow tasks).

Notice that the Destroy phase brings feasiblity back, which is preserved by the Repair phase. Therefore the overall procedure ends up with a solution satisfying all the constraints (but, in case, the one on mandatory orders). The proposed approach takes advantage from the relatively small number of tomorrow tasks with respect to the today ones, so that we expect that infeasibilities rarely occur.

5.2 Parallel implementation

To further reduce running times while preserving the quality of solution, we devise two parallel implementations of the first level tabu-search: parallel neighborhood, exploring neighborhoods 1R, 2S and mR in parallel, and the parallel evaluation, where the solutions of each neighborhood are evaluated in parallel.

The *parallel neighborhood* (PN) procedure considers one independent thread for each tabu search neighborhood. Each thread explores one neighborhood and stops as soon as (i) it evaluates all the neighbors or (ii) another thread evaluates all neighbors and finds an improving solution. In the deterministic exploration version, synchronization issues relates to solution improvement notification. In addition, in the stochastic version, we need to synchronize the pool of the best solutions when the last thread terminates.

The *parallel evaluation* (PE) procedure partitions the set of solutions of each neighborhood, and devotes an independent thread to each part. Stopping conditions and synchronization issues are similar to the ones for PN.

Notice that, in deterministic search, PE runs the same steps as the corresponding sequential implementation and, hence, provide the same (unless ties occur) final solution. On the contrary, even in deterministic search, PN may in principle select the best neighbor from a different neighborhood and end up with a different solution.

6 A bound through column generation

In order to assess the performance of the proposed algorithm for EPDT, we introduce an Integer Programming model based on set covering formulation. To this end, we assume that preferences on collecting orders in the same route and minimum number of vehicles with depot as end-of-day position are negligible.

6.1 A set covering formulation

Let $G = (N, A)$ be a complete graph whose nodes N represent tasks and A is the arc set. For each vehicle $v \in V$, N includes the following dummy nodes: α_v and ω_v, corresponding to the initial and final dummy tasks; ω_v^{eod}, if v has a preferred or prescribed end-of-day

position. α_v and ω_v and, in case, ω_v^{eod} are associated to dummy mandatory orders, added to the set O^M.

Given a vehicle $v \in V$, any feasible route for v can be seen as a path in G that starts in α_v and ends in ω_v. Moreover, only v can visit α_v, ω_v and ω_v^{eod}. Let Ω^v be the set of paths in G corresponding to routes feasible for v. We remark that each path $r \in \Omega_v$ is elementary, i.e., r visits each node at most once. For each path $r \in \Omega^v$, we denote its cost by c_r: following the assumption above, we set $w_D = w_E = 0$ and, according to equation (3), we have $c_r = Z_2(r)$. Let y_r be a binary variable equal to 1 if path r is in the solution, 0 otherwise. For each non-mandatory order $o \in O \setminus O^M$, let x_o be a binary variable equal to 1 if and only if o is assigned to no vehicle. EPDT can be formulated as:

$$\min \sum_{v \in V} \sum_{r \in \Omega^v} c_r y_r + \sum_{o \in O \setminus O^M} q_o x_o \tag{4}$$

$$\text{s.t.} \sum_{v \in V} \sum_{r \in \Omega^v} a_{ir} y_r \geq 1 \quad \forall\, i \in N : o(i) \in O^M \tag{5}$$

$$\sum_{v \in V} \sum_{r \in \Omega^v} a_{ir} y_r + x_{o(i)} \geq 1 \quad \forall\, i \in N : o(i) \in O \setminus O^M \tag{6}$$

$$\sum_{v \in V} \sum_{r \in \Omega^v} y_r \leq |V| \tag{7}$$

$$y_r \in \{0,1\} \quad \forall\, r \in \Omega^v, v \in V \tag{8}$$

$$x_o \in \{0,1\}, \quad \forall\, o \in O \setminus O^M \tag{9}$$

where: a_{ir} is a constant equal to 1 if path r includes task node i, 0 otherwise; $o(i)$ is the order i belongs to; q_o is a penalty for non executed orders defined as $q_o = P_o$, if $o \in O^N$, and $q_o = P_o + L_o$, if $o \in O^U$.

Objective function (4) is equivalent to maximizing the score (2), noticing that: (i) by definition $c_r = Z_2(r)$; (ii) by (5) no mandatory order is missing; (iii) the score can be written as

$$\sum_{o \in O} P_o - \sum_{o \in O \setminus O^M} q_o x_o - \sum_{v \in V} \sum_{r \in \Omega^v} c_r y_r \; ; \tag{10}$$

and (iv) the first term in the previous equation is constant.

A solution satisfying (5) visits all tasks belonging to mandatory orders at least once,

whereas, by (6), tasks in non-mandatory orders are not visited if and only if the corresponding order is not assigned. Finally, constraint (7) bounds the number of routes by the number of available vehicles. This constraint, together with (5), also guarantees that each vehicle v runs exactly one route, as shown by the following Lemma.

Lemma 1. *Given a feasible solution* $(\bar{y}, \bar{x})$ *to (5-9),* $\forall\, v \in V, \exists!\, r \in \Omega_v : \bar{y}_r = 1$.

Proof. For each vehicle v, there is at least one dummy task (e.g. α_v) belonging to a mandatory order that can only be included in Ω^v, hence, by (5), at least one variable $y_r, r \in \Omega^v$ has value 1. Moreover, if more than one variable take value one, constraint (7) would be violated. □

Model (6) is a set covering formulation, which is convenient from a computational point of view [4], but allows feasible solutions visiting tasks more than once. To this end, we notice that, in order to include preferences on order assignment and end-of-day position, route costs may include negative terms. We may adopt a set partitioning formulation with (5-6) turned into equalities, however the following theorem shows that solving the set covering formulation is equivalent to solving the set partitioning one.

Theorem 2. *Given an optimal solution to (4-9), it is always possible to derive an equivalent optimal solution with constraints (5-6) tight.*

Proof. Let (y^*, x^*) be an optimal solution and let $i \in N$ be a node satisfying either (5) or (6) strictly. If the solution is such that $y^*_r = 1$ only for paths r with non-negative cost components, we obtain another feasible solution that does not increase the cost (hence optimal) as follows: we remove i from all routes (but one if $o(i) \in O^M$ or $x^*_{o(i)} = 0$), and, if defined, we set $x^*_{o(i)} = 0$ (notice that $q_{o(i)} \geq 0$). Hence, recalling that feasible paths ar elementary, we can restrict to the case when visiting $o(i)$ strictly reduces the cost of two paths r or s such that $y^*_r = y^*_s = 1$ (if more than two variables are equal to one in y^*, we iterate the argument). This can happen if (Case 1) i has a preferred vehicle or (Case 2) i is the final dummy task of a vehicle with preferred end-of-day position. Let $v(r)$ (resp. $v(s)$) be the vehicle associated to r (resp. s): by Lemma 1, $v(r) \neq v(s)$. In Case 1, by definition of EPDT, i has at most one preferred vehicle, say, without loss of generality, $v(r)$, so that removing i from s does not increase c_s and yields a solution with less paths

covering i and non-increased cost. In Case 2, i appears only in either $\Omega_{v(r)}$ or $\Omega_{v(s)}$ which excludes that both r and s visit i. □

An upper bound for EPDT can be obtained by solving the linear relaxation of (4-9), where the domain of y and x variables is extended to real non negative values. By non-negativity of q_o, upper bound constraints $x_o \leq 1$ can be neglected. This can also be done for y variables, by the same argument of Lemma 1. The size of sets Ω^v is very large and we adopt a column generation approach: we start from restricted sets $\bar{\Omega}^v$ and we dynamically add routes up to reaching an optimal solution. Initial sets $\bar{\Omega}^v$, together with variables x, should define an initial feasible solution. To this end: (i) we initialize each Ω^v with the route through tasks of fixed orders assigned to v (as an alternative, routes from the tabu search heuristic can be used), and (ii) we add a further dummy variable y_0, related to a dummy route collecting all remaining tasks but the ones of mandatory orders, with $c_0 = M$, M being a very large constant. Resulting model defines the Restricted Master Problem (RMP). After solving RMP, we exploit dual information to recover a column to be conveniently added, if any. To this end, we consider the dual variables $\pi_i \geq 0$ associated to constraints (5-6) for each node i, and $\mu \leq 0$ associated to (7), and we solve, for each vehicle $v \in V$, the following Pricing Problem:

$$\text{PP}(v): \qquad \bar{c}_v = \min_{r \in \Omega^v} \left\{ c_r - \sum_{i \in N} a_{ir} \pi_i - \mu \right\}. \tag{11}$$

If such vehicles exist that $\bar{c}_v < 0$, then the corresponding y_r are added to the RMP, otherwise we have found the optimal solution to the linear relaxation of (4-9).

6.2 The elementary shortest path problem with multi-pickup and multi-delivery orders

PP(v) is an Elementary Shortest Path Problem with Resource Constraints (ESPPRC) on the subgraph of G induced by the tasks that can be visited by v. ESPPRC can be solved by label correcting algorithms (see, e.g., [14]). We will describe how we adapted the general schema to the attributes of EPDT and, in particular, to multi-pickup and

multi-delivery. We define the cost of an arc $(i, j) \in A$ as

$$c_{ij} = \tilde{w}_A(j)\, C_v^U\, d_{p(i)\,p(j)} - \tilde{w}_F(j, v) - \tilde{w}_H(j, v) - \pi_j - \tilde{\pi}_\alpha(i) \tag{12}$$

where: $\tilde{w}_A(j) = w_A$ if j is a tomorrow task, 1 otherwise; $\tilde{w}_F(i, v) = w_F$ if v is the preferred vehicle of j, 0 otherwise; $\tilde{w}_H(j, v) = w_H$ if j represents the preferred end-of-day position of v, 0 otherwise; $p(i)$ and $p(j)$ are the position of i and j respectively; $\tilde{\pi}_\alpha(i) = \pi_i$ if $i = \alpha_v$, 0 otherwise.

6.2.1 Labels

A label represents a path from α_v as a tuple $L = (i, t^W, t^R, z, l^1, l^2, \mathcal{O}, \mathcal{U})$ where: i is the last node of the path; t^W and t^R are the cumulative working and driving time, respectively; z is the path cost; l^1 and l^2 are the volume and weight on board after visiting i; $\mathcal{O}$ is the set of open orders, i.e., orders with at least one visited pickup node and at least one delivery node not yet executed (orders started but not completed); $\mathcal{U}$ is the set of unreachable nodes (delivery tasks with not-yet-executed pickups or already visited nodes). Each element of L is called *resource*. Notice that the definition of L has been adapted to EPDT and, in particular, $\mathcal{O}$ and $\mathcal{U}$ account for open multi-pickup and multi-delivery orders. In fact, in pickup and delivery contexts, $\mathcal{U}$ usually reports the set of unreachable *orders* or equivalent information (see, e.g., [6, 14]). In fact, if orders include one pickup and one delivery task, the information on unreachable orders, combined with the status reported by $\mathcal{O}$, allows univocally identifying the task (pickup or delivery) that can be visited preserving elementary paths and precedence constraints. This is not the case in EPDT: e.g., once an open order is identified, information on unreachable order is not enough to identify which further pickup or delivery tasks must still be visited. As a consequence, we define $\mathcal{U}$ such that information on unreachable specific tasks is available and can be exploited to, e.g., avoid visiting the same node or, as we will see, define sufficient dominance conditions. We remark that resource $\mathcal{O}$ is redundant (the state of an order can be retrieved from $\mathcal{U}$), and used to speed up dominance checking.

6.2.2 Label extension

A label to node i is extended to a label to node j by adding an arc (i, j) and updating the resources accordingly: t^W and t^R are obtained from travel times $t_{p(i)\,p(j)}$, duration and time window of j and, when needed, short and night breaks; z takes c_{ij} into account, as well as, in case, penalties for soft time window violations in j, and extra time with respect to preferred route duration if $j = \omega_v^{eod}$; we add or subtract the weight and volume of j to l^1 and l^2 respectively, depending on j is a pickup or delivery task (see, e.g., [6]). Notice that only feasible extensions are considered, so that the ones violating hard time windows or capacity constraints are discarded.

Concerning specialized resources for EPDT, $\mathcal{O}$ is initialized to the empty set and: we add $o(j)$ if j is its first pickup task to be visited; we remove $o(j)$ if j is its last visited delivery task. Moreover, we initialize $\mathcal{U}$ with all delivery nodes and we extend it as follows: we add node j (elementary path); if j is the first pickup of $o(j)$ we add ω_v (the path cannot terminate with a new undelivered order on board) and (when defined) ω_v^{eod}, if $o(j)$ has further today tasks (we have today tasks to visit before the end of the day); if j is the last pickup of $o(j)$ we remove all deliveries of $o(j)$ from $\mathcal{U}$ (deliveries become reachable as soon as all pickups are visited); if j is the last delivery node on board, we remove ω_v (all orders have been completely delivered and we can reach the dummy final node); if j is the last today task on board, we remove ω_v^{eod}, if defined (the vehicle can visit the end-of-day dummy task).

6.2.3 Dominance

Given a label L, we denote by $i(L)$ the last node of the path represented by L; $t^W(L)$ the working time in L; $t^R(L)$ the driving time; $z(L)$ the partial path cost; $l^1(L)$ the vehicle load volume; $l^2(L)$ the vehicle load weight; $\mathcal{O}(L)$ the set of open orders in L; $\mathcal{U}(L, o)$ the set of unreachable nodes in L related to task of an order $o \in O$; $\mathcal{U}(L, -)$ the set of unreachable nodes in L with no associated order (e.g. initial and final dummy tasks).

Given two labels L_1 and L_2, L_1 dominates L_2 if all the following conditions hold:

$$i(L_1) = i(L_2) \tag{13}$$

$$z(L_1) \leq z(L_2) \tag{14}$$

$$t^W(L_1) \leq t^W(L_2),\ t^R(L_1) \leq t^R(L_2),\ l^1(L_1) \leq l^1(L_2),\ l^2(L_1) \leq l^2(L_2) \tag{15}$$

$$\mathcal{U}(L_1, -) \subseteq \mathcal{U}(L_2, -) \tag{16}$$

$$\mathcal{U}(L_1, o) \subseteq \mathcal{U}(L_2, o) \quad o \notin \mathcal{O}(L_2) \cup \mathcal{O}(L_2) \tag{17}$$

$$\mathcal{U}(L_1, o) = \mathcal{U}(L_2, o) \quad o \in \mathcal{O}(L_2) \cup \mathcal{O}(L_2). \tag{18}$$

Roughly speaking, dominance states that L_1 has a smaller cost, has consumed less resources and, moreover, L_1 has less unreachable nodes and, hence, at least the same feasible extensions as L_2. Concerning condition (18), we notice that it states equality for orders that are open in at least one label. In fact, in a multi-pickup and multi-delivery scenario, inclusion is not sufficient, as shown by the following example. Assume order o includes three pickup and one delivery tasks and let L_1 represent a path visiting one pickup task, and L_2 include one more pickup; clearly $\mathcal{U}(L_1, o) \subset \mathcal{U}(L_2, o)$, but the feasible extensions of L_2 do not include the task in $\mathcal{U}(L_2, o) \setminus \mathcal{U}(L_1, o)$, whereas feasible extensions of L_1 have to. Therefore, extensions of L_2 are not feasible for L_1, meaning that L_1 does not dominate L_2. If an order is not open in both L_1 and L_2, then it is either completed (all pickups and deliveries are unreachable) or not in charge (all deliveries and no pickups are unreachable), so that inclusion is sufficient, as requested by (17).

7 Computational results

The two-level local search heuristic described in Sections 4 and 5 has been implemented and integrated in a decision support platform in use at Trans-Cel to assist the operations management office during the initial planning and the execution of daily routes [10]. In this section, we present computational results on two sets of instances.

The first benchmark is made of real instances referred to the initial planning of 43 days at Trans-Cel. We test our approach on this benchmark with the aim of comparing

Table 1: Parameters of the two-level local search procedure for EPDT.

	Value	Description
T	4 to 8	tabu list length (depending on the number of non-fixed orders)
M_1	30	maximum number of not improving tabu search iterations
M_2	100	maximum number of tabu-search iterations
ρ	50%	fraction of tasks in the definition of close orders
δ	20 km	distance threshold in the definition of close orders
M	$10 \cdot \sum_{o \in O} P_o$	penalty for missed mandatory orders (infeasible solutions)

different versions of the proposed procedure and assessing its ability to solve EPDT in real settings, measuring its performance in terms of both efficiency and optimality gap. With the second benchmark, including instances from literature on the Pickup and Delivery Problem with Time Windows (PDPTW), we compare the proposed heuristic to the state-of-the art on problems of broad interest.

Algorithms have been coded in C++ (Microsoft compiler) and run on an Intel Core i5-7400 3 GHz CPU with 8 GB RAM. The column generation has been implemented using the SCIP 3.1.1 framework and Soplex as linear programming engine [1]. The parameters of the proposed approach has been calibrated using a set of training real instances, in collaboration with Trans-Cel operations manager: their values are summarized in Table 1. We always consider the mR neighborhood with $m = 2$ (we displace pairs of orders). The parameters calibration have been realized manually by testing the algorithm on a preliminary set of real instances. In particular, the tests have been analyzed along with the operations managers in order to match criteria based on their expertise. Notice that values of parameters w_X, related to all the indicators I_X, depend on the business model of the transportation company and are adjusted on the field according to the relevance of the preference they represent in each specific industrial context.

7.1 Results on real instances

We consider 43 real instances collected from March to December 2017 at Trans-Cel, with up to 55 orders (27.8 on average), 116 tasks (72.2 on average) and 14 vehicles (11.2) on average. Their features are detailed in Table 2, divided into five groups, depending on the tasks number range shown in the first column. The second column gives the group size. Following columns show average and, in parenthesis, minumum and maximum number of

Table 2: Real instances.

Group	Count	Tasks	Orders	Fleet
0-40	9	23.1 (12.0 ; 38.0)	7.7 (3.0 ; 12.0)	7.1 (3.0 ; 14.0)
41-80	9	61.9 (51.0 ; 79.0)	20.7 (14.0 ; 29.0)	12.3 (10.0 ; 14.0)
81-90	10	84.5 (81.0 ; 90.0)	33.3 (30.0 ; 40.0)	12.0 (10.0 ; 13.0)
91-100	9	93.9 (91.0 ; 97.0)	38.3 (34.0 ; 44.0)	12.3 (11.0 ; 14.0)
101-116	6	108.3 (103.0 ; 116.0)	43.7 (36.0 ; 55.0)	12.7 (12.0 ; 13.0)
All	*43*	*72.2 (12.0 ; 116.0)*	*27.8 (3.0 ; 55.0)*	*11.2 (3.0 ; 14.0)*

tasks, orders and vehicles, respectively.

We have tested different versions of the algorithm: Best Insertion heuristic only (BI), tabu search with deterministic neighborhood exploration (DET), tabu search with deterministic exploration and additional three runs of the stochastic exploration (RND). We have also tested the impact of filtered mR neighborhood (+F), parallel neighborhood using three threads, one per neighborhood (+3PN), and parallel evaluation using four threads, the number of CPU cores available in the test environment (+4PE). All the versions are considered with fast second level heuristic and infeasible route fixing since, during our test, the destroy-and-repair phase has been necessary just in three cases.

Table 3 compares the quality of the solutions provided by RND to the bound obtained by the column generation procedure (CG) described in Section 6. Results are aggregated per size group (first column) and we report per-cent values on: number of instances where the CG converges within one hour and, hence, a bound is available; number of instances where the linear programming relaxation from CG has no fractional variables; number of instances RND solves to proven optimality; optimality gap in the remaining cases (when the bound is available), computed as $(B-R)/B$, where R is the value of RND solution and B the upper bound from CG. Notice that the gap measures the profit loss with respect to a baseline (CG in this case). We remark that results in this table do not take preferences on collecting orders in the same route and minimum number of vehicles at the depot at the end of day into account. We observe that the proposed CG converges in most of the real instances (88.4% on average). Moreover, the solution of the LP relaxation is often integer, thus providing the optimal solution to EPDT (under the assumption above) in the 65.1% of cases (almost 78% of smaller and 33% of larger instances are solved). Based on CG bound, we prove the optimality of nine solutions over 43 (21%), in particular for

Table 3: Results on real instances: optimality gap.

Group	CG bound (%)	CG opt (%)	RND opt (%)	RND gap (%)
0-40	100.0	77.8	66.7	0.8 (0.4 ; 1.1)
41-80	77.8	66.7	22.2	1.4 (0.4 ; 3.1)
81-90	90.0	60.0	0.0	0.8 (0.1 ; 1.4)
91-100	100.0	77.8	11.1	0.8 (0.2 ; 2.4)
101-116	66.7	33.3	0.0	0.6 (0.0 ; 0.9)
all	*88.4*	*65.1*	*20.9*	*0.9 (0.0 ; 3.1)*

Table 4: Results on real instances: basic algorithms.

Group	BI vs RND (%)	DET vs RND (%)	BI (s)	DET (s)	RND (s)
0-40	2.8 (0.0 ; 13.3)	0.1 (0.0 ; 0.8)	0.0 (0.0 ; 0.0)	0.3 (0.0 ; 2.4)	1.4 (0.0 ; 9.7)
41-80	5.6 (0.0 ; 14.6)	0.0 (0.0 ; 0.4)	0.1 (0.0 ; 0.3)	3.2 (0.0 ; 8.5)	12.9 (0.0 ; 32.3)
81-90	6.2 (0.2 ; 10.8)	0.7 (0.0 ; 3.3)	0.2 (0.1 ; 0.3)	5.4 (2.4 ; 12.0)	23.7 (14.1 ; 40.1)
91-100	8.8 (3.0 ; 14.1)	0.3 (0.0 ; 1.7)	0.4 (0.2 ; 0.8)	9.0 (3.8 ; 20.1)	40.3 (18.5 ; 79.9)
101-116	8.2 (3.5 ; 14.8)	0.2 (0.0 ; 1.0)	1.1 (0.3 ; 4.3)	19.1 (4.4 ; 56.7)	77.7 (22.9 ; 196.8)
all	*6.2 (0.0 ; 14.8)*	*0.3 (0.0 ; 3.3)*	*0.3 (0.0 ; 4.3)*	*6.5 (0.0 ; 56.7)*	*27.8 (0.0 ; 196.8)*

small instances. In the remaining cases the gap, when available, is fairly small: less than 1% on average and at most 3.1% in the worst case. This attests to the ability of the proposed two-level tabu search of finding optimal or nearly optimal solution for EPDT in real scenarios.

Results on the basic BI, DET and RND algorithms (with again all preferences active) are given in Table 4. We take RND as baseline for the quality of the solutions, since, by definition, it provides better results than BI and DET. For each size group, the first two columns compare BI and DET to RND, providing the gap (average, minimum and maximum within the instance group) computed as $(R - H)/R$, H being the value of the solution of BI or DET. Following columns are devoted to running times in seconds (average, minimum and maximum). Concerning the quality improvement due to different components of the algorithm for EPDT, we notice that the tabu search, either with or without randomized exploration, significantly improves over the initial solution provided by BI, which is about 6% on average far from RND (about 9% and up to 15% for larger instances). Adding randomization provides appreciable benefits in terms of reliability: DET solutions are only 0.3% worse than RND on average, but the gap is 3.3% in the worst case. As we may expect, DET and, in particular, RND pay the better performance with larger running times. On average, DET is 20 times slower than BI and RND is more

Table 5: Results on real instances: per-cent gap vs RND.

Group	DET+F(+4PE)	RND+F	RND+F+4PE	RND+F+3PN
0-40	0.6 (0.0 ; 2.4)	0.0 (0.0 ; 0.0)	0.0 (0.0 ; 0.0)	0.0 (0.0 ; 0.0)
41-80	0.1 (0.0 ; 0.6)	0.1 (0.0 ; 0.6)	0.1 (-0.2 ; 0.6)	0.0 (-0.3 ; 0.6)
81-90	0.7 (0.0 ; 3.3)	0.1 (-0.4 ; 0.8)	-0.2 (-1.2 ; 0.5)	-0.1 (-1.0 ; 0.8)
91-100	0.4 (0.0 ; 1.7)	0.1 (0.0 ; 0.3)	0.1 (-0.6 ; 0.7)	0.2 (-0.8 ; 1.3)
101-116	0.3 (-0.4 ; 1.0)	0.2 (-0.4 ; 1.0)	0.2 (-0.4 ; 1.0)	0.1 (-0.1 ; 0.5)
all	*0.4 (-0.4 ; 3.3)*	*0.1 (-0.4 ; 1.0)*	*0.0 (-1.2 ; 1.0)*	*0.0 (-1.0 ; 1.3)*

than four time slower than DET (from few seconds to half a minute). In the worst case, DET remains under one minute computation, whereas RND may take up to about 200 seconds for larger instances.

All the proposed neighborhoods play a significant role and the number of times they are selected complies with the variable neighborhood strategy: by considering the runs of RND on all the instances, the most selected moves are 1R (77.5% of times), followed by 2S (15.7%) and 2R (6.7%).

Tables 5 and 6 report on how neighborhood filtering and parallel implementations impact on, respectively, solution quality and running times. Table 5 shows the same gap values of the first two columns of Table 4, and compares the different versions of the algorithm to RND, recalling that 4PE has no impact on the solution found by deterministic exploration. For each filtered or parallelized version, the solution quality is almost the same as the corresponding basic version. Only filtering the deterministic version gives an average slightly worse quality (the gap just increases from 0.3% to 0.4%). Random exploration seems not to suffer from restricted mR moves: the average solution quality remains the same and, in five instances, it finds better solutions than RND. In particular, RND+F+4PE or RND+F+3NE have the same average performance, and RND+F+4PE seems slightly better in terms of reliability (quality loss up to 1.0% instead of 1.3%)

The additional components intended for algorithm speedup allow improving efficiency, as shown in Table 6: it reports the running times, in seconds, for the different algorithm versions. Neighborhood filtering is quite effective, since it reduces running times by about 37% on average for both DET and RND. With this improvement, DET+F running times are always below 10 seconds, but for larger instances that may take up do 36 seconds to be solved. Interestingly, RND+F runs always in less then 35 seconds, but for larger

Table 6: Results on real instances: running time (in seconds).

Group	DET+F	DET+F+4PE	RND+F	RND+F+4PE	RND+F+3PN
0-40	0.3 (0.0 ; 1.9)	0.1 (0.0 ; 0.7)	1.2 (0.0 ; 9.2)	0.5 (0.0 ; 2.8)	0.8 (0.0 ; 5.2)
41-80	1.9 (0.0 ; 5.3)	0.7 (0.0 ; 1.8)	9.2 (0.0 ; 32.2)	3.0 (0.0 ; 8.5)	4.5 (0.0 ; 15.6)
81-90	4.0 (1.3 ; 8.2)	1.4 (0.5 ; 2.6)	17.7 (6.1 ; 33.0)	5.7 (2.1 ; 9.6)	8.1 (3.8 ; 16.0)
91-100	4.8 (2.3 ; 9.4)	1.7 (0.8 ; 3.2)	20.4 (10.3 ; 34.4)	7.6 (3.6 ; 13.2)	10.8 (5.6 ; 16.3)
101-116	12.7 (3.3 ; 36.0)	4.0 (1.2 ; 10.4)	48.9 (15.3 ; 113.6)	17.8 (5.8 ; 44.5)	31.8 (6.1 ; 110.3)
all	*4.2 (0.0 ; 36.0)*	*1.4 (0.0 ; 10.4)*	*17.4 (0.0 ; 113.6)*	*6.1 (0.0 ; 44.5)*	*9.7 (0.0 ; 110.3)*

instances, taking almost two minutes in the worst case. We get the most significant speedup from parallel evaluation PE, yielding a further average 65% time reduction by exploiting the four threads currently available on most computers. DET+F+4PE runs always in less than about 10 seconds (just 1.4 on average), and RND+F+4PE takes 6 seconds on average, 17.8 on average for large instances and always less than 45 seconds. The reduction is consistent, but less relevant, for RND+F+3PN, allowing a speedup of 44.5% on average upon RND+F. Moreover, the running time is, in the worst case, close to two minutes. In fact, PN exploits three threads and, moreover, more neighbor solutions are likely to be visited, since parallel neighborhood exploration loses some of the advantages of variable neighborhoods.

Summarizing, RND provides the best average results in terms of solution quality, and the speeding up components (filtering and parallel implementation) preserve the quality of results: in fact, from detailed results, none of these three versions dominates the others. From an efficiency perspective, RND+F+4PE performs better and, more interestingly, is able to limit running time to a few seconds on average and, in any case, to less then 45 seconds. This performance makes RND+F+4PE suitable for both daily route planning in static settings and reaction policies based on re-optimization, due to the relatively low demand dynamism of EPDT. In fact the algorithm, integrated in the operations management office support tool, is triggered by different modules to, e.g., accommodate on-line orders, react to unavailable infrastructures of vehicle failures, estimate costs to assist order negotiation etc. (see [10]), and the response times are compatible with the business model and operational settings of small trucking companies like Trans-Cel.

Table 7: Results on PDPTW instances.

Group	Δvehicle	Δvehicle%	Δdistance%	time (s)
LC1	0.2 (1.0)	2.2 (10.0)	-0.3 (-17.3 ; 19.2)	1000 (652; 1786)
LC2	0.1 (1.0)	3.1 (25.0)	4.8 (0.0 ; 27.3)	4332 (4262; 4441)
LR1	0.7 (2.0)	5.7 (18.2)	3.0 (**-0.3** ; 7.5)	1255 (274; 1983)
LR2	0.6 (1.0)	18.2 (33.3)	7.6 (0.7 ; 17.6)	4269 (4237; 4341)
LCR1	0.6 (2.0)	4.7 (13.3)	2.9 (0.9 ; 4.9)	4425 (4248; 4919)
LCR2	0.6 (1.0)	15.6 (25.0)	6.7 (3.0 ; 11.7)	558 (326; 801)
All	*0.5 (2.0)*	*8.5 (33.3)*	*4.1 (-17.3 ; 27.3)*	2607 (274; 4919)

7.2 Results on literature PDPTW benchmarks

In order to verify how the proposed algorithm solves problems of broad interest, we have considered the Li-and-Lim's benchmark for PDPTW [18], defined on a subsets of EPDT attributes, namely single pickup and single delivery orders, hard time windows and vehicle capacity. Concerning the objective function, PDPTW considers the number of vehicles and the total distance in lexicographic order, which is modeled in EPDT by setting vehicle deployment fixed costs to a big-M constant. Moreover, we set revenues to 0, so that, in fact, we minimize the total travel distance. PDPTW routes are closed and we model them with constraints on the route ending position. We have run the two-level local search version RND+F+4PE with the same parameter settings as for real instances and simplified route evaluation (e.g., we have only today tasks).

The Li-and-Lim's benchmark includes 56 instances adapted from the Solomon's benchmark [23]. They define 100 orders each and are classified in six classes, as listed in the first columun of Table 7, depending on: spatial distribution (orders are clustered in LC, uniformly distributed in LR, mixed in LRC); scheduling horizon (shorter in '1' and longer in '2' instances, with suitable smaller and larger capacities). Table 7 compares the results obtained by our heuristic to the best solution, available from [22] and provided by different works [5, 13, 18, 19, 20]. For each class, we report average and maximum number of additional vehicles needed by our procedure, the related per-cent value, the per-cent additional distance, and the running time in seconds (average, minimum and maximum). The number of additional vehicles is between zero and two (0.5 on average) and the routes are longer by 4.1% on average. Notice that if we restrict to '1'-instances, the average increase in the route length is only 1.9%. In fact, instances of type '1' are more compliant

with EPDT, where the time horizon is short with respect to the distance between tasks. Running times show how the efficiency of our heuristic is strongly related to the number of orders expected in a route, due to the computational complexity of the second-level heuristic for solution evaluation which suffers from routes of consistently large sizes, as is the case for '2'-instances. Therefore, the time of execution of our algorithm differs on average of 1 order of magnitude with respect to other ad-hoc approaches in literature for PDPTW (see, e.g., [18]). Nonetheless, we obtain one new best solution for instance LR106, which slightly improves the one of [18]: we use the same number of vehicles and shorten the total distance by 0.3%.

8 Conclusions

Inspired by the daily operations in a small trucking company, we have addressed the problem of determining an optimal set of routes to service express pickup and delivery requests. We have thus formulated EPDT and proposed a heuristic approach to solve it and support both initial daily route planning and route re-optimization in reaction to real-time modifications of orders, vehicles or road infrastructures. The solution approach takes the specific attributes of EPDT into account, among which, the peculiar definition of multi-pickup and multi-delivery orders, open routes, daily planning horizon involving orders with both one- and two-days span, and other specific constraints and preferences. We have devised a two-level local search heuristic: at the first level a tabu search explores possible order-to-vehicle assignments in a variable neighborhood fashion; the second level uses local search to sequence route tasks and provides the first level with solution evaluation. The search is sped up by a fast solution evaluation procedure, filtering of the most computationally expensive first-level neighborhood (mR), and parallel neighborhood evaluation.

Computational results show the effectiveness of the design choices, in particular in real-world settings. In fact, the algorithm is currently integrated in the support platform used by the operations management office of a small trucking company to deploy initial vehicle routes and to adapt them to changing conditions. In particular, the algorithm in operation uses deterministic and randomized neighborhood exploration, move filtering

and parallel solution evaluation, corresponding to the implementation that better trades off quality of solutions and running times: solving real instances takes a few seconds on average, and fairly less than one minute in the worst case, thus allowing for supporting both static and, thanks to the relatively low demand dynamism, real-time settings. The quality of solutions have been assessed both on the field, through consensus by operations managers, and, more formally, by computing a bound based on an integer linear programming model. It corresponds to a set covering formulation and we solve its linear relaxation by a column generation approach tailored for the specific attributes of EPDT, in particular, multi-pickup and multi-delivery orders, and preferences. To this end, we have adapted the definition of the labels used to solve the ESPPRC for variable generation, and the related dominance rules. We have thus obtained a bound proving the optimality or near-optimality of the solutions proposed by our heuristic. We have also tested the algorithm on a problem of broader interests, namely PDPTW, which is a special case of EPDT. Results show limited gaps with respect to the state of the art for some classes of instances and provide, in one case, a new best solution.

References

[1] T. Achterberg. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[2] R. Baldacci, E. Bartolini, A. Mingozzi, and R. Roberti. An exact solution framework for a broad class of vehicle routing problems. *Computational Management Science*, 7(3):229–268, 2010.

[3] R. Baldacci and A. Mingozzi. A unified exact method for solving different classes of vehicle routing problems. *Mathematical Programming*, 120(2):347–380, 2008.

[4] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[5] R. Bent and P. Van Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research*,

33:875–893, 2006.

[6] A. Bettinelli, A. Ceselli, and G. Righini. A branch-and-price algorithm for the multi-depot heterogeneous-fleet pickup and delivery problem with soft time windows. *Mathematical Programming Computation*, 6:172–197, 2014.

[7] O. Bräysy, M. Gendreau, G. Hasle, A. Lokketangen, and J.Y. Potvin. Metaheuristics for the vehicle routing problem and its extensions: a categorized bibliography. In B. Golden, S. Ranghavan, and E. Wasil, editors, *The vehicle routing problem: latest and new challanges*, pages 143–169. Springer, New York, USA, 2008.

[8] A. Ceselli, G. Righini, and M. Salani. A column generation algorithm for a rich vehicle-routing problem. *Transportation Science*, 43(1):56–69, 2009.

[9] L. De Giovanni, N. Gastaldon, I. Lauriola, and F. Sottovia. A Heuristic for Multi-attribute Vehicle Routing Problems in Express Freight Transportation. In A. Sforza and C. Sterle, editors, *Optimization and Decision Science: Methodologies and Applications ODS 2017. Springer Proceedings in Mathematics & Statistics*, volume 217, pages 161–169. Springer International Publishing, Cham, 2017.

[10] L. De Giovanni, N. Gastaldon, M. Losego, and F. Sottovia. Algorithms for a vehicle routing tool supporting express freight delivery in small trucking companies. *Transportation Research Procedia*, (30):197–206, 2018.

[11] M. Gendreau, C. Gueret, A.L. Medaglia, and V. Pillac. A review of dynamic vehicle routing problems. *EURO Journal on Computational Optimization*, 255(1):1–11, 2013.

[12] Google Maps API. `https://developers.google.com/maps/`, 2018.

[13] G. Hasle and O. Kloster. Industrial vehicle routing. In G. Hasle, K. A. Lie, and E. Quak, editors, *Geometric Modelling, Numerical Simulation, and Optimization: Applied Mathematics at SINTEF*, pages 397–435. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[14] S. Irnich and G. Desaulniers. Shortest path problems with resource constraints. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, pages 33–65. Springer US, Boston, MA, 2005.

[15] M. Khemakhem, R. Lahyani, and F. Semet. Rich vehicle routing problems: from a taxonomy to a definition. *European Journal of Operational Research*, 241(1):1–14, 2015.

[16] C.A. Kontovas, H.N. Psaraftis, and M. Wen. Dynamic vehicle routing problems: three decades and counting. *Networks*, 67(1):3–31, 2017.

[17] G. Laporte, S. Ropke, and T. Vidal. Heuristics for the vehicle routing problem. In P. Toth and D. Vigo, editors, *Vehicle Routing: Problems, Methods, and Applications. MOS-SIAM Series on Optimization*, pages 87–116. 2014.

[18] H. Li and A. Lim. A metaheuristic for the pickup and delivery problem with time windows. In *Proceedings 13th IEEE International Conference on Tools with Artificial Intelligence. ICTAI 2001*, pages 160–167, 2001.

[19] Quintiq. `http://www.quintiq.com/optimization/vrptw-world-records.html`, 2018.

[20] S. Ropke. *Heuristic and exact algorithms for vehicle routing problems*. PhD thesis, Technical University of Denmark, 2006.

[21] S. Ropke and J. F. Cordeau. Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, 43(3):267–286, 2009.

[22] Sintef (transportation optimization portal). `https://www.sintef.no/projectweb/top/pdptw`, 2018.

[23] M.M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constrains. *Operations Research*, 35:254–264, 1987.

[24] Tom Tom Telematics. `https://www.telematics.tomtom.com/`, 2018.

[25] P. Toth and Daniele Vigo. *Vehicle Routing: Problems, Methods, and Applications, Second Edition*. SIAM - Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2014.

[26] T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins. Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research*, 231(1):1–21, 2013.

[27] T. Vidal, T.G. Crainic, M. Gendreau, and C. Prins. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, 40(1):475–489, 2013.

[28] T. Vidal, T.G. Crainic, M. Gendreau, and C. Prins. A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, 234(3):658–673, 2014.