MASARYK
UNIVERSITY

FACULTY OF INFORMATICS

Vehicle routing with transfers

Master's Thesis

VÁCLAV SOBOTKA

Brno, Spring 2022

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Vehicle routing with transfers

Master's Thesis

VÁCLAV SOBOTKA

Advisor: doc. Mgr. Hana Rudová PhD.

Brno, Spring 2022


FACULTAS ARTIS INFORMATICÆ
UNIVERSITAS MASARYKIANA

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Václav Sobotka

Advisor: doc. Mgr. Hana Rudová PhD.

Acknowledgements

I would like to thank to my partner Lucie Tesařová for patience and kind calming words when needed the most. All her support and care made me eventually finish this work.
I owe many thanks to my advisor doc. Hana Rudová PhD. for guiding my efforts into the right directions. Her well-aimed comments, advice and restrictions on the number of pages per chapter pushed the thesis further than I originally expected.
Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA CZ LM2018140) supported by the Ministry of Education, Youth and Sports of the Czech Republic.

Abstract

The thesis addresses a variant of vehicle routing problem allowing for transfers. The introduction of transfers makes it possible for vehicles to exchange request loads at designated transfer points. This option provides more flexible means of transportation which can be exploited in order to reduce the traveled distances and costs. A novel approach capable of utilizing the possibility of transfers on very large instances is proposed, implemented and experimentally evaluated. The experiments were conducted on both real and synthetic instances. A real-world dataset targeting an application of large-scale freight transportation was provided by the company Wereldo. Instance generator built upon OpenStreetMaps geographical data was implemented and used to generate the synthetic instances. The obtained experimental results demonstrate that the implemented solver is capable of finding substantial savings by introducing transfers, especially in case of the real-world instances.

Keywords

Vehicle routing problem, artificial intelligence, optimization, transportation, logistics, Wereldo.com, transfers

Contents

1 Introduction

With today's scale, freight transportation presents a broad field of study for research in operations. The vehicle routing problem and its numerous variants can be used to formalize real-world transportation problems. Thanks to decades-long research in the area, effective algorithms for solving such problems exist, being it exact methods or heuristic approaches.

The grave importance of employing algorithms in vehicle routing lies in the fact that computed solutions of the problem are considerably better than results obtained by humans. Good quality solutions potentially eliminate unnecessary traffic and reduce the overall transportation costs. Since costs associated with transportation are significant, savings even in the order of units of percents are important.

The problem approached in this thesis was provided by the company Wereldo, a member of the Association of Industrial Partners of the Faculty of Informatics. It builds upon a solver for the pickup and delivery problem with time windows from the master thesis of Vojtěch Sassmann [1] developed for the same company. The aim of this work is to explore a generalized version of the pickup and delivery problem allowing for request transfers between vehicles at designated locations. The goal is to exploit this generalization so that better solutions can be found. The setup with large scale instances is of crucial interest, since the targeted application requires vehicle routing of a fleet counting few hundreds of vehicles with more than 1,000 transportation requests to be served.

The core of the problem at hand appears in the literature mostly under the name pickup and delivery problem with transfers. Papers presenting algorithms for solving the pickup and delivery problem with transfers exist for both exact and heuristic methods, none of them, however, targets the problem in such a large scale. Specifically, the usual instance size present in the literature is below 100 requests with several works running experiments on instances with up to 300 requests. The only notable exception is the work of Petersen and Ropke [2], where the target instances consisted of around 1,000 requests. The observed limitation is clearly tied with the newly added dimension of

transfers. Consequently, the traditional search methods suffer from quick expansion of the search space.

Inspite the fact that the possibility of transfers opens a new space for improvements in solution quality, this potential seems to be utilized almost exclusively in the context of relatively small problems. The goal of this work is to design an approach capable of exploiting the possibility of transfers while allowing for solving very large instances. The proposed method is assessed on a real problem and a set of generated instances. The real problem was provided by the company Wereldo. In order to get the latter set of instances, custom instance generator based on real-world geographies sourced from OpenStreetMaps was implemented.

The rest of the thesis is organized as follows. Chapter 2 provides the necessary background regarding the vehicle routing problem and its variants and introduces the pickup and delivery problem with transfers. Chapter 3 covers the proper formalization of the application problem provided by Wereldo. Chapter 4 summarizes the state-of-the-art present in the literature. Chapter 5 introduces the implemented approach and comments on other considered solver designs. Chapter 6 presents the instance generator based on the OpenStreetMaps. Chapter 7 describes the instances used in the experiments, discusses the parameter tuning, provides further insights into the behavior of the algorithm and reports results on the proposed experiments. Chapter 8 concludes the work, summarizes its contributions and gives directions for further research.

2 Problem overview

The pickup and delivery problem with transfers (PDPT) is a member of the vehicle routing problem (VRP) family of problems. The VRP was first defined in 1959 by Dantzig and Ramser [3]. Since then, VRP and its numerous variants have been a subject to active research. Even nowadays new variants of VRP emerge as novel challenges in logistics need to be tackled.

The original VRP can be seen as a direct generalization of the traveling salesman problem (TSP). In TSP, the task is to find a route through a defined set of vertices on a fully connected graph so that the route ends in its origin and the overall traveled distance is minimized. In VRP, the goal is the same, but more than one route is generally constructed (more salesmen understood as vehicles can be used to serve the requests). Since TSP is NP-complete and any TSP can be viewed as a VRP, VRP is an NP-complete problem (as well as any of its more general variants).

Next, a brief overview of the VRP family is provided and the concept of transfers in PDPT is properly described. Finally, the specific real-world problem provided by Wereldo and its main characteristics are presented.

2.1 Vehicle routing problems

Two principal branches can be tracked in the VRP family – static and dynamic problems. In the static problems, all of the requested visits and conditions are known upfront and do not change over the course of optimization. In the case of dynamic problems, this information is revealed over time – new requests may appear, existing requests may get canceled and various conditions may be updated (e.g. request load size, travel times…). In this section, only the static branch of VRP is considered so that the researched problem can be characterized properly. Interested reader may consider reading [4] for a systematic classification of VRP variants in general and [5] for a survey concentrating on the dynamic VRP variants.

Naturally, a rich pallet of additional characteristics is often considered. Examples of such characteristics are limits on vehicle capacities,

introduction of time windows in which locations must be visited or assumption of more than one vehicle depot in the problem. In fact, such constraints and features are a must in case of real-world applications simply because solutions ignoring these properties would be of no practical use. A VRP variant of special interest for this work is the pickup and delivery problem (PDP), since our application is a generalized variant of PDP.

In PDP, the notion of transportation request is understood in a more generic[1] way than in plain VRP. Whereas VRP specifies the transportation requests as single locations to be visited, a request in PDP is represented by a pair of locations denoted as *pickup* and *delivery*. In order to service a request, a vehicle must first visit the pickup where the cargo is loaded. Secondly, the same vehicle must eventually visit the delivery of the request where the cargo is unloaded.

2.2 Pickup and delivery problem with transfers

PDPT is a direct generalization of PDP. The key difference of PDPT is that it allows for multiple vehicles to cooperate on servicing a single request. This cooperation takes the form of request load exchanges referred to as *transfers*. Throughout the journey of a request load between its pickup and delivery, zero or potentially multiple[2] transfers may take place.

Shall a request be serviced by more than one vehicle, the respective load must be transferred between vehicles somewhere. Generally, this transfer must take place in one of the designated *transfer points*. The set of transfer points is typically very limited.

Naturally, transfers have inherent operational aspects. The transfer itself requires synchronization of some kind between the two cooperating vehicles. The exact form of the synchronization is dependent

1. The pickup and delivery modelling approach directly generalizes the plain VRP. In case of plain VRP, either all pickup or all delivery request locations are implicitly assumed to be in the depot.
2. The reviewed literature usually limits to at most one transfer. The reasons are purely practical since the search space expansion would be even faster in case of more transfers. In this work, the possibility to transfer a request twice was allowed in a limited manner.

on particular application context, however two main synchronization requirements are dominant.

The first and more common synchronization scenario allows for temporary storage of the load at the transfer point. Consequently, it is only required that the first vehicle completely delivers the load to the transfer point before the second vehicle starts picking it up[3]. This transfer mode is suitable for applications such as freight or parcel transportation assuming that warehousing or depot facilities with proper capacities and equipment are available.

The second transfer mode requires a complete synchronization of the cooperating vehicles. More precisely, it is expected that the vehicles meet at the transfer point and the handover of the load is performed from one vehicle to the other with no option of intermediate storage. This scenario is more common in applications targeting passenger transportation.

In conclusion, transfers in PDPT are performed with the aid of transfer points through which two cooperating vehicles exchange the load relevant to the request being transferred. Depending on the application, this exchange may benefit from temporary storage at the transfer point or may require full synchronization of the vehicles.

2.3 Real-world problem

The core of the real-world problem provided by the company Wereldo matches the PDPT as introduced in the previous section. The targeted application is freight transportation roughly covering territory of a single mid-sized European country with a vehicle fleet counting slightly more than 300 vehicles. A dataset of requests from one week period was provided.

The first group of characteristics is related to time aspects. Firstly, both the pickup and delivery in each request are required to be visited within the given time windows with hard boundaries. Secondly, both the pickup and delivery actions are expected to take some time. This time requirement is referred to as *service time*. Lastly, the routes in

3. It is worth mentioning that these two vehicles may be actually one and the same vehicle. This matches the scenario in which a vehicle visits the transfer point, temporarily unloads some request load and returns later to pick it up.

the solution are subject to a constraint limiting the total duration of the route.

Secondly, limited vehicle capacities are assumed. The vehicles are limited in the maximum weight and number of pallets they can transport at once. Conversely, the transportation requests must specify the weight and number of pallets in the load.

Another important feature of the dataset is the heterogeneous nature of the vehicle fleet. The types of the vehicles range from vans to large trucks. Regarding particular properties, the vehicles differ in their capacity limits, locations of their depots and costs per covered distance unit. The last property directly affects the objective function as the optimization minimizes the distances covered by the vehicles weighted by their respective prices per distance unit.

Apart from the mentioned route duration limit, vehicle routes are also limited in the number of stops. Importantly, multiple actions at one physical location count as a single stop towards the limit. Moreover, the service times are treated in a similar manner. Specifically, the service time is counted solely for the last action during a stop as the service time is meant to account for the complete loading and unloading of a vehicle at a location.

Regarding the transfer points, the dataset contains four logistical facilities at strategic locations. Based on the discussions with Wereldo, the transfer points allow for intermediate storage, are assumed to operate non-stop, no additional costs must be incurred for realized transfers and both the cooperating vehicles (separately) should be subject to standard service time during the transfer. Notably, the transfer points physically coincide with depots of the vehicle fleet.

Lastly, it is important to mention the number of requests as one of the most prominent properties of the problem. A typical mid-week day in the dataset counts roughly 1,200 transportation requests. This number is high even in the context of PDP without transfers. In case of PDPT, this present an order of magnitude larger instances than those considered large in the literature. Consequently, the problem size is of special importance for this work as it gravely impacts the solution approaches which may be taken into consideration.

3 Formal model

The modelling approach in this work is based on [6]. First, the underlying graph is explained. Then, the constraints of the model are discussed. The notation is summarized in Tables 3.1 and 3.3.

3.1 Underlying graph

The goal of the optimization is to construct routes servicing all given requests while minimizing the objective without violating any constraints. Naturally, the graph network on which the routes are situated is necessary. First, the graph nodes and their logic are discussed. Second, the set of all reasonable edges is presented.

Table 3.1: Graph-related notation.

	Fundamental sets
R	Set of transportation requests
K	Set of vehicles
T	Set of transfer points
	Individual nodes
r^+	Pickup node of request $r \in R$
r^-	Delivery node of request $r \in R$
k^+	Origin depot node of vehicle $k \in K$
k^-	Destination depot node of vehicle $k \in K$
$s(m)$	Start node of transfer $m \in T$
$f(m)$	Finish node of transfer $m \in T$
	Node types
N^+	$\{ r^+ \mid r \in R \}$, i.e., set of pickup nodes
N^-	$\{ r^- \mid r \in R \}$, i.e., set of delivery nodes
K^+	$\{ k^+ \mid k \in K \}$, i.e., set of origin depot nodes
K^-	$\{ k^- \mid k \in K \}$, i.e., set of destination depot nodes
$s(T)$	$\{ s(m) \mid m \in T \}$, i.e., set of transfer point start nodes
$f(T)$	$\{ f(m) \mid m \in T \}$, i.e., set of transfer point finish nodes
N	$N^+ \cup N^-$, i.e., set of all request nodes

3.1.1 Graph nodes

The first important property of the graph nodes is their relationship with physical locations. One physical location may play multiple roles in the problem. For example, a large logistical facility may serve as pickup location for many requests. On the level of the graph, however, each request gets its own pickup node. In general, one physical location is mapped to potentially many graph nodes, one node for each distinct role. Moreover, there is a zero distance among such nodes.

The second important aspect is the heterogeneous character of the node set. It consists of several disjoint subsets playing different roles in the modelling. Specifically, there are six different types of nodes in the graph.

The first two types are related to requests. The set N^+ are the *pickup nodes*, one per each request. Symmetrically, N^- is the set of *delivery nodes*. The set created by their union $N = N^+ \cup N^-$ is referred to as *request nodes*.

The next two node types are vehicle depots. The set K^+ contains a node modelling the *origin depot* for each present vehicle. In analogy, the set K^- consists of nodes for *destination depots*.

Finally, the last two node types implement the transfer points in the model. Each transfer point is represented by two nodes in the graph. These nodes are referred to as *start node* and *finish node* of the transfer point. In case of request nodes, it is always clear whether the node is associated with loading or unloading action. In contrast, transfer points must necessarily support both types of actions. The transfer point abstraction logically separates unloading actions at the start node from the loading actions at the finish node. Despite a transfer point is represented as two distinct nodes, further constraints of the model tie the nodes together into one logical unit. On the level of the graph, visit to a transfer point is implemented as a visit to the respective start node and immediate traversal of the edge to the finish node. It is never possible to visit the start or finish node separately without visiting its counterpart.

In summary, the set of nodes in the network graph G is defined as in the following equation.

$$V = N^+ \cup N^- \cup K^+ \cup K^- \cup s(T) \cup f(T)$$

3.1.2 Graph edges

It makes sense to restrict the set of edges to a reasonable subset of $V \times V$ only. First, some edges may never appear in a feasible solution. Second, excluding some edges may address problems in the model without the need of additional constraints.

As discussed, six different types of nodes are distinguished. Naturally, ordered pairs of node types can be seen as types of oriented edges. Since many edge types are redundant or bypass the intended traversal rules in the network, edges of such properties should be eliminated[1]. The inclusion or exclusion of particular edge types is summarized in Table 3.2.

Table 3.2: Overview of valid edge types.

Rows correspond to type of the source node, column to target node of the edge.

	N^+	N^-	K^+	K^-	$s(T)$	$f(T)$
N^+	✓	✓	×	×	✓	×
N^-	✓	✓	×	✓	✓	×
K^+	✓	×	×	✓	✓	×
K^-	×	×	×	×	×	×
$s(T)$	×	×	×	×	×	✓
$f(T)$	✓	✓	×	✓	✓	×

Apart from edge types, there are more criteria based on which edges can be removed. First, reflexive edges should be dismissed completely since they play no valuable role in the network. Second, the number of edges between node types $s(T)$ and $f(T)$ can be substantially reduced. In fact, the internal logic of transfer points dictates that the visit of start node must be directly followed by a visit to the corresponding finish node. Consequently, there is no need to assume edges between start and finish node of two different transfer points. In the opposite way, edges from finish node to their respective start node are completely redundant. Lastly, edges starting in $i \in K^+$ and ending in $j \in K^-$ should be limited to the set $\{(k^+, k^-) | k \in K\}$.

1. For example, any edge connecting origin depot node with request delivery node is of no use. This is due to the fact that the vehicle must visit the respective pickup node before visiting the delivery node.

In conclusion, the reasonable set of edges $E \subset V \times V$ of the network graph G is formally defined by the following equation.

$$E = E' \setminus E^X$$

Where E' is the set of edges summarized in Table 3.2 and E^X is the set of edges to be eliminated defined as follows.

$$\begin{aligned} E^X = \quad & \{(i,i) \mid i \in V\} \quad \cup \\ & \{(s(m), f(n)) \mid m, n \in T \text{ s.t. } m \neq n\} \quad \cup \\ & \{(f(m), s(m)) \mid m \in T\} \quad \cup \\ & \{(k^+, l^-) \mid k, l \in K \text{ s.t. } k \neq l\} \end{aligned}$$

3.2 Model constraints and objective

The decision variables in the model are centered around three concepts. First, the variables x_{ij}^k are binary indicators of whether the vehicle $k \in K$ traversed the edge $(i,j) \in E$. Secondly, the variables z_j^{kr} similarly indicate whether the load of the request $r \in R$ was loaded in the vehicle $k \in K$ in the node $j \in V$. Lastly, it is necessary to track time in order to ensure proper precedence between actions. This is done by the variables a_i^k holding the arrival time of the vehicle $k \in K$ to the node $i \in V$.

Table 3.3: Constraints notation.

Network graph	
G	(V, E), i.e., the network graph as defined in Section 3.1
$V^-(i)$	$\{ j \mid (j,i) \in E \}$, i.e., predecessors of the node $i \in V$
$V^+(i)$	$\{ j \mid (i,j) \in E \}$, i.e., successors of the node $i \in V$
E^T	$E \setminus \{(s(m), f(m)) \mid m \in T\}$
t_{ij}	Time necessary to traverse $(i,j) \in E$
d_{ij}	Physical distance between the nodes $i, j \in V$

Vehicle properties

W_k	Weight limit of the vehicle $k \in K$
P_k	Pallets count limit of the vehicle $k \in K$
C_k	Price per distance unit of the vehicle $k \in K$
D_k	Maximum duration of the route of the vehicle $k \in K$
S_k	Maximum number of physical stops in the route associated with the vehicle $k \in K$

Request properties

w_r	Weight of the load of the request $r \in R$
p_r	Number of pallets of the request $r \in R$

Graph node properties

l_i	Lower bound of the time window of the node $i \in N$
u_i	Upper bound of the time window of the node $i \in N$
o_i	Service time of the node $i \in N \cup s(T) \cup f(T) \cup K^+$

Decision variables

x_{ij}^k	Binary variable signaling whether the vehicle $k \in K$ uses the edge $(i, j) \in E$
z_i^{kr}	Binary variable signaling whether the vehicle $k \in K$ carries the request $r \in R$ while in the node $i \in V$
a_i^k	Arrival time of the vehicle $k \in K$ at node $i \in V$

3.2.1 Vehicle routes

Route of the vehicle $k \in K$ is a simple path between k^+ and k^- in G^2. The first high-level requirement on a valid solution is that the subgraph induced by $x_{ij}^k = 1$ for each individual vehicle $k \in K$ is a route. Together, the Constraints 3.1 to 3.6 secure that the assignment of variables x_{ij}^k is consistent with this requirement.

The combination of Constraints 3.1 with 3.2 establishes that the node k^+ is necessarily the start of route of the vehicle $k \in K$ and that

2. The depot nodes k^+ and k^- are assumed to be one physical location as in the literature [6]. Without this assumption, the model would need to prevent counting costs for unused vehicles.

the route contains at least one edge. Constraint 3.2 allows each vehicle to start from at most one origin depot node, not more. Constraint 3.1 ensures that the chosen depot is actually the origin depot associated with the vehicle $k \in K$.

$$\sum_{j \in V^+(k^+)} x^k_{k^+ j} = 1 \qquad \forall k \in K \tag{3.1}$$

$$\sum_{i \in K^+} \sum_{j \in V^+(i)} x^k_{ij} \leq 1 \qquad \forall k \in K \tag{3.2}$$

Constraint 3.3 ensures flow conservation in the request, start and finish nodes. For each edge taken by a vehicle into a pickup or delivery node, the vehicle must also traverse an edge leaving the node. In case of transfer points, the construction of E guarantees, that there is only one outgoing edge from each start node and, analogically, single incoming edge into each finish node. This is the edge connecting the start and finish nodes of a particular transfer point. Thus, the start and finish node of a transfer point are visited in an immediate sequence or not at all.

$$\sum_{j \in V^+(i)} x^k_{ij} \; - \sum_{j \in V^-(i)} x^k_{ji} = 0 \quad \forall k \in K, \; \forall i \in N \cup s(T) \cup f(T) \tag{3.3}$$

Another important consequence is that a route can end solely in a node $i \in K^-$. Since Constraint 3.3 prevents entering any node $i \in N \cup s(T) \cup f(T)$ without leaving it and there are no edges incoming into the nodes $i \in K^+$, the only remaining possibility is to end the route in a node $i \in K^-$.

The role of Constraint 3.4 is to force the vehicle $k \in K$ to end its route in its designated destination depot $k^- \in K^-$ rather than in an arbitrary node $i \in K^-$. After this constraint is added, the sub-graph induced by each vehicle must contain edges forming a valid route.

$$\sum_{i \in V^-(k^-)} x^k_{ik^-} = 1 \quad \forall k \in K \tag{3.4}$$

Constraint 3.5 establishes the pickups and deliveries of loads at the request nodes. First, it secures that each request node is entered by a unique vehicle. Due to Constraint 3.3, the unique entering vehicle must also leave the node. Moreover, there is no other vehicle leaving the node since it would need to enter it first.

$$\sum_{k \in K} \sum_{j \in V^+(i)} x_{ij}^k = 1 \quad \forall i \in N \tag{3.5}$$

At this point, all request nodes are visited by some vehicle and the sub-graph induced by each vehicle contains a valid route. The sub-graph may, however, include additional cycles detached from the depot-to-depot simple path. Fortunately, this issue is solved[3] once node arrival times are tracked within routes. Constraint 3.6 expresses the general requirement on arrival times of two subsequent nodes in a route.

$$\begin{aligned} & x_{ij}^k = 1 \implies a_i^k + t_{ij} + o_i \cdot \operatorname{sgn}(d_{ij}) \leq a_j^k \\ & \forall k \in K,\ \forall (i, j) \in E \end{aligned} \tag{3.6}$$

A route imposes natural ordering on the arrival times of the nodes visited by the vehicle. Moreover, the edge traversal potentially takes some time and the vehicle may need additional time during node visits for loading and unloading. The travel time is accounted for by the addition of the t_{ij} term. The service time for the origin node is counted solely in the case that a non-zero distance was traversed. This conditional behavior is achieved by multiplying the service time by the signum function applied on the traveled distance.

It should be noted that the implementation of route-related time precedence in the form of single Constraint 3.6 was chosen for its

3. To be completely precise, the natural time precedence constraints within routes eliminate only cycles of positive duration. Cycles with zero duration may be still present. In order to solve the problem with cycles completely, it is possible to force the value of t_{ij} to be at least some small positive ε. This technicality is intentionally left from the model presentation.

succinctness. Despite it is very suitable for presentation of the idea, it introduces redundant decision variables[4] and requires some additional technicalities[5, 6].

3.2.2 Request load flows

So far, the only guarantee regarding request servicing is that all the pickup and delivery nodes have a unique vehicle which visits them. There is, however, no implicit nor explicit association between request loads and vehicles. In order to ensure that the request loads eventually reach their delivery nodes in some sensible manner, tracking of request-load flows is needed.

Constraint 3.7 establishes that vehicles must not have any request load on board while in their origin or destination depot nodes.

$$z^{kr}_{k^+} = z^{kr}_{k^-} = 0 \quad \forall k \in K,\ \forall r \in R \tag{3.7}$$

Constraints 3.8 and 3.9 state that pickup and delivery nodes serve as source and sink of request-load flow of their respective request.

$$x^{k}_{r^+ j} = 1 \implies z^{kr}_{j} = 1 \quad \forall k \in K,\ \forall r \in R,\ \forall j \in V^+(r^+) \tag{3.8}$$

$$x^{k}_{r^- j} = 1 \implies z^{kr}_{j} = 0 \quad \forall k \in K,\ \forall r \in R,\ \forall j \in V^+(r^-) \tag{3.9}$$

---

4. For a node $i \in V$, the variables a^k_i must be set for all vehicles, regardless of whether they visit the node or not. From the correctness point of view, there is no problem, since the non-visiting vehicles can simply choose arbitrary value inside the node's time window. From the practical point of view, however, there is a large number of redundant decision variables. It is alternatively possible to track only single variable a_i for $i \in N$ at the expense of having to express the same precedence constraint for many different types of edges.
5. It is necessary to assume zero service time for origin depot nodes so that the constraint works properly for edges starting from origin depots.
6. In case the last node in route physically coincides with the destination depot, the last service time is not counted towards $a^k_{k^-}$. This may lead to violation of the maximum duration constraint. The problem can be solved by a separate constraint or by manipulating the distance of edges into K^- by adding some very small ε.

The role of Constraint 3.10 is to ensure request-load flow conservation where needed. For a request $r \in R$ and a vehicle $k \in K$, if the request load enters a node in the vehicle, it must also leave it in the same vehicle. This must hold with two exceptions. First, the nodes r^+ and r^- are excluded as they serve as source and sink of the request-load flow. Secondly, the flow conservation is not required between the start and finish nodes of a transfer point. The rules for flow inside transfer points will be established by Constraint 3.12.

$$\begin{aligned} &x_{ij}^k = 1 \implies z_i^{kr} = z_j^{kr} \\ &\forall k \in K,\ \forall r \in R,\ \forall (i,j) \in E^T \text{ s.t. } i \notin \{r^+, r^-\} \end{aligned} \tag{3.10}$$

Constraint 3.11 prevents the existence of the flow where not wanted. It forbids vehicles to carry any request at nodes which were not visited by the vehicle. Depots of the vehicle are omitted since there is no edge to $k^+ \in K^+$ and exactly one edge into $k^- \in K^-$ is required.

$$\begin{aligned} &\sum_{i \in V^-(j)} x_{ij}^k = 0 \implies \sum_{r \in R} z_j^{kr} \leq 0 \\ &\forall k \in K,\ \forall j \in V \setminus \{k^+, k^-\} \end{aligned} \tag{3.11}$$

Next, it is necessary to specify the flow behavior in the transfer point start and finish nodes. Constraint 3.12 establishes that it is not possible for a request load to end its journey in the transfer point. If request-load flow enters the start node in any vehicle, it must eventually leave through the finish node, possibly in a different vehicle. In a sense, 3.12 is transfer-aware flow conservation constraint complementing Constraint 3.10.

$$\sum_{k \in K} z_{s(m)}^{kr} - \sum_{k \in K} z_{f(m)}^{kr} = 0 \quad \forall m \in T,\ \forall r \in R \tag{3.12}$$

Since each pickup and delivery node must be visited, the request-load flow is necessarily created. The vehicle visiting the pickup node

of a request starts hosting the request-load flow with only two possibilities to dismiss it. The first option is to visit the delivery node of the request. The second is to visit the start node of some transfer point. Due to Constraint 3.12, this is essentially equivalent to moving the pickup of the request to the finish node of the transfer point. Since Constraint 3.7 requires vehicles to arrive empty at the destination depot, the only true sink of the request-load flow is the delivery node of the request.

The constraints thus already ensure that there are subsequent route segments (of potentially different vehicles) connecting the pickup and delivery node of every request. The last bit that remains is expressed in Constraint 3.13. It ties together the time tracking for any pair of distinct vehicles transferring load from one to another. Naturally, it is required that the second vehicle cannot arrive for loading before the first vehicle arrived and successfully unloaded the request load.

$$\begin{aligned} &z^{kr}_{s(m)} + z^{vr}_{f(m)} = 2 \implies a^{k}_{s(m)} + o_{s(m)} \leq a^{v}_{f(m)} \\ &\forall k, v \in K \text{ s.t. } k \neq v,\ \forall m \in T,\ \forall r \in R \end{aligned} \tag{3.13}$$

3.2.3 Additional constraints

Constraints 3.14 and 3.15 secure that vehicles cannot exceed their limits on maximum weight and number of pallets on board.

$$\sum_{r \in R} w_r z^{kr}_i \leq W_k \qquad \forall k \in K,\ \forall i \in V \tag{3.14}$$

$$\sum_{r \in R} p_r z^{kr}_i \leq P_k \qquad \forall k \in K,\ \forall i \in V \tag{3.15}$$

Next, Constraint 3.16 requires that arrival times to all request nodes must belong into the time window associated with the given node[7].

$$l_i \leq a^{k}_i \leq u_i \qquad \forall k \in K,\ \forall i \in N \tag{3.16}$$

7. The need to quantify this constraint for all vehicles is a consequence of the succinct time precedence modelling from Constraint 3.6.

Lastly, Constraints 3.17 and 3.18 implement the limits on route duration and maximum number of stops within a route. The implementation of 3.18 uses the signum function conditioning previously used for the service times.

$$a^k_{k^-} - a^k_{k^+} \leq D_k \qquad \forall k \in K \tag{3.17}$$

$$\sum_{(i,j)\in E} x^k_{ij} \cdot \operatorname{sgn}(d_{ij}) \leq S_k \qquad \forall k \in K \tag{3.18}$$

3.2.4 Objective

The minimized objective is a weighted sum of the vehicle route distances where weights are the prices per distance unit of the respective vehicles.

$$\sum_{k\in K} C_k \sum_{(i,j)\in E} d_{ij} \cdot x^k_{ij} \tag{3.19}$$

4 State of the art

This chapter summarizes the state-of-the-art of the works on PDPT present in the literature. Sections 4.1 to 4.4 systematically review encountered solution approaches. Important observations from the review are discussed in Section 4.5. Additionally, features of particular reviewed works are summarized in Table 4.1.

Among the reviewed works, four general streams of approaches can be identified. The first stream is formed by exact methods [6, 7]. The second direction is represented by local search based multi-phase heuristics [8, 9, 10]. The third and largest group of methods revolves around the large neighbourhood search [2, 11] or its adaptive variant [12, 13, 14]. Lastly, population-based methods appear as well [15, 11].

Note, that these four categories are not completely disjoint as some of the works assess more than one approach. Thus, few of the reviewed works are discussed in more places in this chapter.

4.1 Exact methods

One of the most prominent exact approaches for hard combinatorial optimization is the *integer linear programming* (ILP) [16]. In ILP framework, the target problem is expressed in the form of a model which consists of integer[1] variables and a set of linear constraints. Sophisticated implementations of solvers such as Gurobi or CPLEX allow for solving complex problems solely by providing a model to the solver. Apart from serving as an input for solvers, model formulations are useful as a mean of formal description of problems as in Chapter 3.

The paper of Cortes et. al. [6] is the pioneering work on purely exact methods. The main contribution of this work is the first formalization of PDPT in the literature. The problem is formalized in the form of ILP model. The authors propose a custom solution based on branch-and-cut method utilizing the Bender's decomposition technique. The custom solution is then compared to generic branch&bound approach

1. In case only some variables are integers, the term *mixed integer linear programming* (MILP) is used. Solving general ILP and MILP tasks is an NP-complete problem.

Table 4.1: Overview of PDPT properties in the reviewed literature.

Work	*Capacities*	*Time windows*	*Heterogen. fleet*	*Max no. of TPs*	*TP storage limit*	*Multi-transfers*	*Max. requests*	*Solution method*	*Objective*
Cortes et. al., 2010 [6]	✓	✓	×	1	×	×	6	Exact (ILP), custom impl.	Total time
Rais et. al., 2014 [7]	✓	✓	✓	–	×	✓[a]	7	Exact (ILP), Gurobi	Distance
Laporte and Minic, 2006 [8]	×	✓	×	5	×	×	100	Multi-phase iter. heurisitic	Distance
Godart et. al., 2019 [9]	✓	✓[b]	✓	6	✓[c]	×	150	Multi-phase iter. heurisitic	Distance+tardiness[d]
Fu and Chow, 2021 [10]	✓	×	×	–	✓[e]	×	300	Multi-phase iter. heurisitic	Weighted sum[f]
Petersen and Ropke, 2011 [2]	✓	✓	×	1	×[g]	×	982	Apriori transfers + LNS	Weighted sum[h]
Qu and Bard, 2012 [13]	✓	✓	×	1	×	×	25	GRASP+ALNS	No. vehicles+distance[i]
Masson et. al., 2013 [14]	✓	✓	×	33	×	×	193	ALNS	Distance
Sampaio et. al., 2020 [12]	×	✓	×	5	×	×	100	ALNS	Distance
Danloup et. al., 2018 [11]	✓	✓	×	5	×	×	100	LNS, GA	Distance
Peng et. al., 2019 [15]	✓	✓	×	2	×	×	50[j]	Hybrid particle swarm	Profit+distance[k]

a. This is the only work in which multiple transfers of one request were reportedly evaluated. In all other works, either only one transfer points is used, it is explicitly stated that only single transfer is possible or the described procedure cannot produce multiple transfers.
b. The time windows have soft upper bounds. Violations of the upper bounds are accounted for in the objective function.
c. Storage possible, but explicitly limited.
d. Pareto optimization of two objectives.
e. Storage not possible, full vehicle synchronization is required.
f. The sum consists of distance, vehicle transfer time, customer waiting time and total travel time components.
g. Transfers are subject to additional costs accounted for in the objective function.
h. The sum consists of distance, initial and up-time costs for vehicles, costs for applied transfers and road toll components.
i. Objectives are optimized lexicographically, number of vehicles is primary.
j. The largest clearly reported instance counted 26 requests.
k. Objectives are optimized lexicographically, profit is primary.

with favourable results. This comparison alongside with model validation is done on a set of generated instances.

Interestingly, the evaluated instances were very small and the speed of growth in the required time is striking. The largest evaluated instance counted only 6 requests, 2 vehicles and 1 transfer point. Solving this instance required 1,800 seconds for the branch&bound method and 120 seconds for the custom solution. In comparison with the instance containing one request less, both the discussed times were roughly 6 times larger. Clearly, the results indicate that introducing the transfers leads to a severe expansion of the search space even with one transfer point only.

In the work of Rais et. al. [7] an alternative ILP model for PDPT is provided. Their aim is to provide as concise core of the model as possible and discuss VRP variants covered by the model with only minor adjustments needed. The model is evaluated on instances derived from the Li and Lim benchmark [17] using the Gurobi solver. The number of requests in the evaluated instances was at most 7, yet a variable number of vehicles was allowed and any node in the graph could serve as a transfer node resulting in more complex instances than in [6]. Notably, this is the only reviewed work arguably taking the possibility of *multi-transfers*[2] into account. The authors optimize the distance weighted by the vehicle cost and report savings of less than 7 %, instances are solved to optimality. Regarding the runtimes, the maximum reported time exceeded 18,000 seconds for one of the largest instances. Interestingly, the minimum time for a same sized instance was only 18 seconds indicating strong dependence of instance properties and complexity.

It is worth mentioning that several other works [10, 15] concentrating on heuristics provide ILP formulations and validate their ILP models by solving several very simple tasks by the means of ILP solver.

4.2 Multi-phase heuristics

The heuristic approaches in this section are characteristic with their clear division into multiple phases. In such methods, the initial phase typically creates one or potentially many solutions, often by some

2. On its way from pickup to delivery, a request may be transferred more than once.

form of greedy construction. In the later phases, the outcomes of the first phase are iterativelly improved including the opportunity for transfers.

Laporte and Minic [8] were the first to address PDPT. Their proposed algorithm employs a two-phase heuristic. In the first phase, several different permutations on requests are assumed and for each permutation, a solution is constructed by successively inserting requests in the given order. The cheapest insertion including the possibility of transfers is always taken. The second phase takes the best solution obtained in the initial phase and attempts to improve it in multiple iterations. One improvement iteration successively unassigns each request in the solution and then attempts to reinsert it back in the cheapest way possible (with the possibility of transfers).

The evaluation part of the work targets the conditions upon which the transfers are favorable. The authors conducted a set of experiments on randomly generated instances of 100 requests with various properties and provide quite a broad overview of interactions of various instance properties with transfers. Their conclusions are that depending on the instance properties, improvements in traveled distance[3] based on the possibility of transfers can range from almost none up to 40 % in extreme cases.

Among the key factors identified as important for transfer usefulness were the service times upon transfer, geographical clustering and relative positions of the pickup and delivery locations, transfer point placement and time window lengths. Benefits obtained for no or small service times are reduced significantly if the service times are increased. The geographical clustering of locations is reported to play a vital role for the usefulness of transfers. Instances without location clusters were identified to benefit from transfers only marginally and more smaller clusters seemed to be more beneficial than a smaller number of larger clusters. The most favourable transfer point and cluster configuration relied on a transfer point in the center of the service area and location clusters nearby the centers of the area bor-

3. The generated instances use Manhattan distance.

ders[4]. Lastly, rather longer than shorter time windows were found favourable for transfers.

The work of Godart et. al. [9] concentrates on Pareto optimization of two objectives. In comparison to other works, the properties of the addressed problem are quite specific. Transfer points allow for a limited storage only, the authors are interested in a heterogeneous vehicle fleet, and the time windows have a soft upper bound. Because of the soft upper bounds of the time windows, one of the optimized criteria is minimization of the overall tardiness (soft boundary violations) with the second criterion being the overall traveled distance.

They address the problem hierarchically with a three-phase heuristic. In the first phase, requests are assigned to vehicles based on three different heuristics providing a wide Pareto-front of solutions. The second phase takes the vehicle-requests assignments in each solution and solves the routing problem for the given assignment. The proposed heuristic employs the idea that nearest neighbours are likely to be served after each other in good solutions. The third phase aims to incorporate transfers into the solutions. The proposed heuristic identifies regions covered by each vehicle route in the solution. If these regions of two vehicles intersect and the intersection contains a transfer point, transfers between the vehicles are considered and applied greedily. This procedure is applied iterativelly until no further improvement is possible.

The reported effects of transfers are a reduction in the total traveled distance by up to 30 % as well as benefits in reduced tardiness criterion. Not surprisingly, the best improvements in the total distance were obtained at the expense of notably larger tardiness and vice versa. Regarding the runtimes, the total time is clearly dominated by the third phase addressing transfers. The largest assessed instance counted 150 requests, 12 vehicles and 4 transfer points and the time needed for the third phase exceeded 2,300 seconds.

Fu and Chow [10] propose a two-phase heuristic. The first phase of the heuristic sequentially and greedily inserts requests into the solution without the possibility of transfers. The second phase improves

4. Since the Manhattan distance is assumed, this configuration allows for transfers with marginal or even zero detours. Specifically, a request with pickup and delivery locations anywhere on the horizontal and vertical mid-axes of the service area can be transferred via the central point of the area with no additional detour.

the obtained solution by considering transfers in a way very similar to the third heuristic phase in [9]. Each transfer point keeps track of all vehicles passing within a parameterized search distance. Any pair of vehicles passing in proximity of the same transfer point is considered for a transfer. Then transfers from these candidates are chosen greedily.

The *enroute microtransit*[5] problem targeted in their work has several distinct characteristics. The first are the absence of explicit time window constraints and the orientation towards passenger transportation. The time aspect is accounted for in the form of passenger comfort component of the objective function. Next consequence of the passenger transportation application is the requirement of full vehicle synchronization during transfers. The last notable features are the assumed grid network graph infrastructure and the possibility to perform a transfer at any node of this grid.

The work also provides the most detailed scaling tests among the reviewed works. The authors test three different sizes of the grid and scale the number of requests and vehicles up to 300 and 100, respectively. The largest evaluated instance consisted of the maximum number of vehicles and requests and only used the medium-sized network with total computation time exceeding 7,000 seconds. The runtime growth was steeper on the largest network and the last reported instance counted 210 request and 70 vehicles with computation time exceeding 6,500 seconds.

4.3 Methods based on large neighbourhood search

The large neighbourhood search (LNS) [18] and its adaptive variant (ALNS) [19] are iterative metaheuristics working with a single solution. In each iteration, a relatively large portion of the current solution is destroyed so that it can be repaired into a new solution. Depending on the quality of the acquired solution, it is either accepted or rejected in the search[6]. LNS based implementations typically provide multiple

5. Microtransit transportation is a conceptual compromise between public and taxi-like transportation of passengers.

6. This decision rule is referred to as *acceptance criterion*. The acceptance criterion is typically dependent mainly on the quality of the assessed solution, but other factors

problem-dependent strategies for destroying and repairing the solution referred to as *operators*. The additional feature of ALNS is that it keeps track of performance of the individual operators and projects this information into the probabilities of choosing the particular operators.

Masson et. al. [14] target an application of regular transportation service for people with disabilities within a city. Notably, they evaluated real-world instances which are quite large both in the number of requests and transfer points when compared to other works. The largest evaluated instance counted 193 requests and 5 transfer points. Results on this instance yielded the best benefits of transfers of almost 10 % in the total covered distance. Unfortunately, the reported runtime exceeds 10 hours (as opposed to 750 seconds without transfers). Generally, the reported runtimes after transfer inclusion are roughly one order of magnitude larger than without transfers.

The solution method generalizes ALNS for PDP. Two new removal operators and three insertion operators targeting transfers are adapted based on the existing ALNS operators and previously proposed heuristics. The first removal operator removes more requests transferred via the same transfer point in order to enable the whole group to be navigated through a different transfer point. The second transfer-oriented removal operator removes requests clustered geographically. The idea is that requests with clustered deliveries or clustered pickups could be all transferred via the same transfer point while being picked up or delivered by one vehicle.

The first insertion operator *best insertion with transfer* is in principle based on the insertion scheme proposed by Laporte and Minic [8]. The next *transfer first* operator can be seen as a group variant of the previous one. Instead of assuming the requests to be inserted sequentially, the group is inserted at once with transfers enforced (if possible). Then, some of the transfers are possibly cancelled if this leads to improvement of the solution quality. The last insertion operator called *regret insertion with transfer* uses transfers for requests which would be more expensive to deliver directly without the transfer.

---

may be included as well. As an example, the simulated annealing criterion takes randomness and the progress of the search into account as well. Interested reader may refer to [20] for a survey of acceptance criteria for ALNS.

Conceptually identical approach was taken by Sampaio et. al. [12]. The authors build on the work of Masson et. al. [14] and extend it by novel operators. First, the removal operator *transfer-based request removal* randomly removes requests based on their history of transfers while preferring requests which were transferred only few times over requests transferred frequently. Secondly, the idea of redundant transfer point visits is introduced in the form of an insertion operator. The point of the operator is to provide opportunity for simpler future transfers. Such unnecessary transfer point visits are kept in the solution for a given number of iterations and are removed afterwards, if no transfer takes advantage of this opportunity.

Notable feature of the work is the targeted application. The authors concentrate on the area of crowd-shipping transportation and find the transfers very favourable in the context of typical instance characteristics. In the crowd-shipping environment, the majority of the vehicle fleet consists of individually contracted drivers usually capable of providing their services for relatively short periods of time only. The evaluation concentrates on randomly generated instances with different lengths of driver shifts and different distances between the pickup and delivery locations in a request. The reported benefits of transfers range from no benefits up to 50 % in the total traveled distance. The results suggest that short driver shifts with long distance requests benefit from transfers the most.

Qu and Bard [13] propose a method based on the GRASP metaheuristic and utilize ALNS in order to improve the generated solutions. First, a pool of solutions is generated by sequentially inserting requests into routes with the possibility of transfers. The insertions are performed randomly with probabilities proportional to their costs. A subset of the generated pool is passed to the second phase and the solutions are improved by ALNS. The operators present in the ALNS allow for transfers and are conceptually comparable to already mentioned operators.

Notably, the authors propose a way to expand very small instances with known global best solution value so that the global best solution value is known even for the expanded instance. The GRASP solver is evaluated on a set of instances with 25 requests and 1 transfer point obtained in this way with results very close to the optimal values.

Danloup et. al. [11] aim to compare large neighbourhood search and genetic algorithm approaches. Their LNS implementation uses a relatively simple set of standard operators adjusted to account for transfers. The LNS implementation is, however, reported to be outperformed by the genetic algorithm approach. Despite the LNS in this work seems to serve as a referential point for the newly assessed genetic algorithm mainly, the authors propose a more efficient implementation of one of the insertion operators from [13].

Among the reviewed works, Petersen and Ropke [2] are the only who address some variant of problem with transfers in scale comparable to our problem. The instances are based on an industrial application for Alex Andersen Ølund, a major Danish transporter of flowers. Typical instances cover daily transportation performed by a fleet of 170 trucks serving between 500 to 1,000 requests with one transfer point available. Interestingly, the transfers seem to be treated only marginally by the authors and the evaluation of their benefits is missing.

The distinct feature of the work is the completely different approach to transfers when compared to other works. The decision whether a request will be delivered directly or will be transferred is done apriori based on a fairly simple heuristic. Generally, requests with a large time interval between the latest pickup and earliest delivery are favored to be transferred as well as requests with very long distances between pickup and delivery (if not causing too large detour when transferred). Requests identified suitable for transfer are split into two requests. Then, a parallelized implementation of LNS unaware of any transfers is started. Clearly, this approach completely bypasses the steep growth in required computation time observed in other methods. On the other hand, the proposed heuristic rule is a severe simplification of the transfer dimension of the problem.

4.4 Population-based methods

Population-based metaheuristics [21] operate over a population of solutions rather than working with a single solution only. Importantly, the mechanisms responsible for exploring new and potentially better solutions often rely on multiple existing solutions or even on the

whole population. The first prominent branch of these metaheuristics is inspired by evolutionary processes. Representative method from this group are the genetic algorithms (GA). In GA, new solutions are obtained by the means of combination of two solutions referred to as reproduction and by local changes of a single solution denoted as mutations. The second branch is inspired by swarm intelligence observed in insect colonies, flocks of birds etc. The solutions in the swarm move in the search space based on relatively simple rules dependent on interactions with other members of the swarm.

Peng et. al. [15] address the selective variant of PDPT with a hybrid variant of the Particle Swarm Optimization (PSO) metaheuristic. In the selective variant of PDPT, delivering a request is not mandatory. Instead, requests are associated with profits for servicing them. The authors lexicographically optimize the total achieved profit and the total traveled distance (in this respective order).

PSO relies on a population of solutions referred to as particles encoded as vectors of numbers. During the optimization, the movement of the particles in the vector space is guided by the global best position among all particles and the best position found by the given particle. The authors hybridize the idea of the plain PSO metaheuristic by applying local search during particle decoding phase.

The work seems to primarily assess the potential of hybrid PSO with accent on the multi-objective character of the problem. Unfortunately, the largest clearly reported result is on a randomly generated instance with only 28 requests, 10 vehicles and 1 transfer point. In summary, the choice of hybrid PSO does not seem to bring substantial benefits in the context of transfers when compared to other reviewed methods.

Danloup et. al. [11] are the first to address PDPT by the means of GA metaheuristic. The aim of their work is to implement a solver for PDPT based on GA and compare the results with LNS based approach. They assess both implementations on data from [8] and report improvements over the results found in the earlier literature. Moreover, the GA implementation qualitatively outperforms the LNS.

The key part of the work is the concept of coding of solutions in the population. The choice of the authors is a highly indirect coding of the solutions resulting in a pair of number vectors. One vector holds a number between 0 and 1 per request representing the priority of the

request insertion during solution reconstruction. The second vector holds a bit indicator for each request defining the mode of insertion during solution reconstruction. The first option is to choose the best insertion with enforced transfer while the second insertion variant takes the best insertion regardless of whether transfer is applied. On one hand, this coding allows for a straightforward and efficient implementation of traditional crossover and mutation operators. On the other hand, evaluation of solutions requires a costly reconstruction process from the indirect coding. Means of caching common patterns in solutions are implemented in order to reduce the time needed for the evaluations.

New solutions are produced by the means of standard one-point and two-point crossovers and several mutation operators. The standard mutations include random value replacement for the priorities, bit flipping for the insertion modes and exchange of two priority values. Also, two interesting mutations specific to the problem are introduced. The first is said to be inspired by LNS as it chooses a larger number of priority values and replaces them randomly. The second novel operator aims to reduce the number of used vehicles. It increases the priority of the least prioritized requests as these are likely to require additional vehicle to be served.

Regarding the population management, a population of a fixed size (20 solutions) was used. After new solutions are created, duplicated solutions are eliminated and the new generation is chosen based on the elitism criterion. Solutions with better quality are more likely to be chosen for reproduction. In one iteration, two new children are produced by the means of either one or two point crossovers and randomly chosen mutation is applied to some solution in the new population.

4.5 Discussion

Three points with great importance to our problem are to be made from the literature review. The first regards the potential benefits arising from transfers and their relationship to the properties of the targeted problem. Secondly, the literature suggests that transfers add substan-

tially to the problem complexity. Lastly, the benchmark datasets for PDPT should be mentioned.

Regarding the first point, variability of results across the reviewed works and the detailed insights provided by Laporte and Minic [8] show strong relationship between the benefits obtainable by the means of transfers and the properties of the target problem. Based on the factors identified in the literature, our problem seems to demonstrate properties quite favourable for transfers[7].

With the introduction of transfers, practical complexity of the problem is increased drastically. This fact is clearly indicated by the reported computation times of exact method and the sizes of the largest instances solved to optimality. This phenomenon, however, appears even in the case of heuristic approaches. While exact methods are limited to less than 10 requests, the breaking point for heuristic search algorithms comes with lower hundreds of requests. The time requirements clearly evince faster than linear growth with respect to the number of requests. Consequently, even very careful implementations of the reviewed methods cannot be expected to yield any reasonable results on our instances with more than 1,000 requests.

The general approach from Petersen and Ropke [2] seems to present a viable direction under these circumstances. Clearly, its main weakness is its excessive simplicity. On the other hand, it bypasses the main issue of our application. As discussed, the properties of the target problem are quite favourable for transfers and preliminary experiments indicated that splitting requests apriori may result in benefits. Thus, generalizing this approach to more transfer points, characteristics and elaborating more on the transfer decision making seems to be an interesting research direction.

On the last point, datasets potentially usable as benchmarks were identified. Upon contacting the authors, we managed to obtain datasets from the works of Laporte and Minic [8] and Qu and Bard [13]. Unfortunately, the character of these instances is very far from the problem at hand, mainly due to low number of requests and unrealistic geographical distributions of locations and transfer points. Since the

7. Generally, the distances between pickups and deliveries tend to be relatively long. Secondly, the transfer points are real logistical facilities placed at strategic locations. Lastly, the constructed routes are limited in their maximum duration (yet the limit is not particularly strict).

number of instances derived from the data provided by Wereldo is very limited, the decision was to implement a generator capable of producing instances of reasonably similar character as in the original data.

5 PDPT solver

Conceptually, it is possible to address the transfers in three ways. First and dominantly used approach is to search the transfer options together with the route construction [14, 6, 7, 11, 12, 13, 15]. Secondly, it is possible to solve the problem without transfers and then improve the routes by adding transfers afterwards [9, 10]. Lastly, the requests to be transferred may be decided apriori. The transferred requests get divided into two or potentially more requests with non-overlapping time windows and the resulting PDP instance may be solved with a solver unaware of transfers [2].

As outlined in Section 4.5, the proposed method follows the third direction. Instead of deciding the transfers in a single shot, however, the idea is to rather evaluate larger number of instances with different transfer setups. These evaluations start at relatively short runs of the PDP solver and get prolonged as weak-performing instances get filtered out with gradual strictness. In a sense, the described procedure can be seen as a coarse search over the space of instances. The short PDP solver runs then serve as estimates of instance quality used for comparison of the assessed instances.

The outlined implementation of the PDPT solver will be introduced in three parts. First, the overall schema of the PDPT search will be described in detail in Section 5.1. Then, the key components of the system will be covered. Section 5.2 targets the PDP solver component adapted from [1]. The procedure responsible for generating the initial front of instances with transfers is introduced in Section 5.3. Lastly, Section 5.4 provides a discussion of ideas and concepts which ultimately led to the presented form of the PDPT search procedure.

Since the thesis was prepared in cooperation with the company Wereldo, the source codes are not publicly available as a part of this thesis.

5.1 PDPT search schema

Starting with the input instance, the PDPT solver first generates a front of derived instances with a variable portion of requests transferred. Then, the initial instance front is being evaluated and filtered in two

subsequent phases. Finally, deeper search is performed on few of the best performing instances. A diagram of the proposed PDPT search procedure is shown in Figure 5.1.

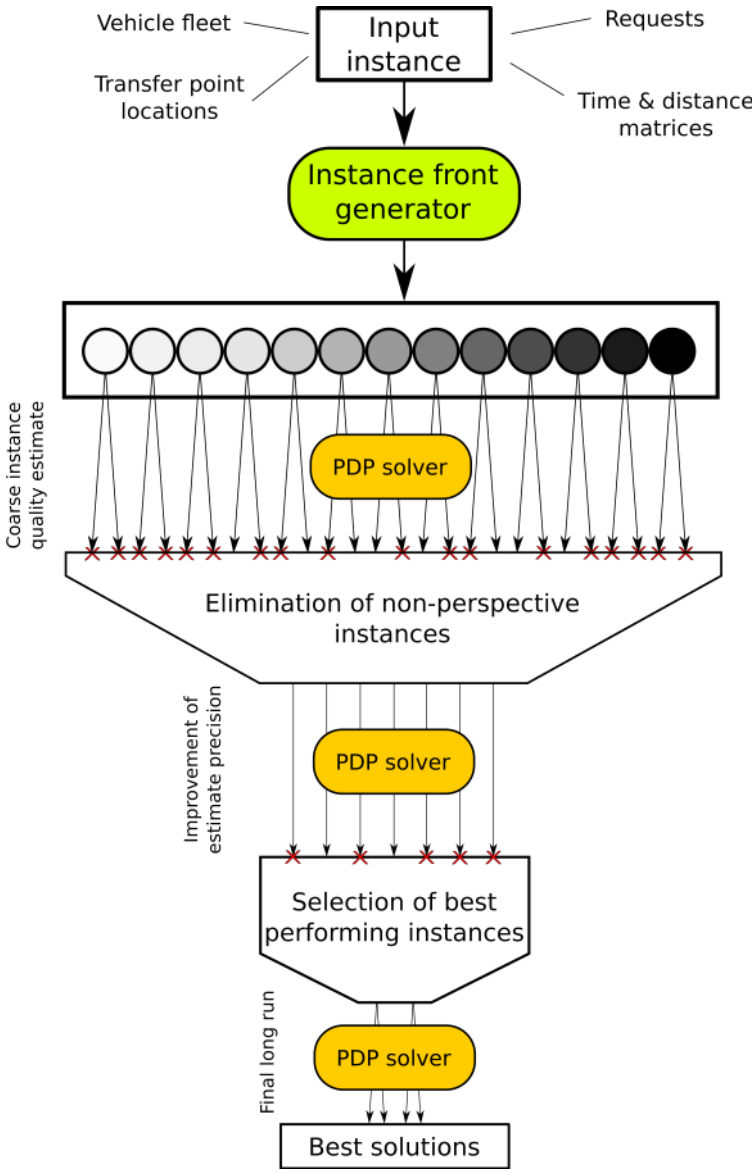

Figure 5.1: High-level schema of the PDPT search procedure.

The high-level shape of the search can be divided into three important phases. The *initial phase* takes the instances provided by the instance front generator component and evaluates the instances with short PDP solver runs. The *middle phase* prolongs the search on promising instances which survived the filtering after the initial phase. Lastly, a very limited number of instances surviving the filtering after the middle phase advances into the *final phase* which eventually returns a set of high-quality solutions after longer search. In case of all three phases, the evaluation of each instance may be duplicated into multiple PDP solver runs. This measure improves the stability of results and is subject to parameterization of the PDPT solver.

The goal of the initial phase is to rule out large portion of the weak-performing instances without spending unnecessary computational resources on their evaluation. For the purpose of this pruning, instance quality estimates based on several hundreds of PDP solver iterations were found sufficient in the target application. In order to minimize the risk of eliminating instances which are still improving rapidly, the subsequent filtering is relatively benevolent.

During the middle phase, the instances considered promising enough get additional iterations of the PDP search. The best runs of all instances surviving the first filtering are restarted from their best solutions and undergo longer PDP search than in the initial phase. The role of the middle phase is to provide enough iterations for the PDP search to reach a stable best solution. Based on the experience with solution quality evolution in the PDP search, improvements start to appear scarcely after few thousands of iterations and the solutions acquired at this point are very close to qualities reached after full long runs of the solver. Thus, the quality estimates at the end of the middle phase are significantly more precise than the initial estimates and allow for reliable selection of the best performing instances into the final phase.

The final phase of the solver solely improves the few surviving instances by restarting the PDP search from their best acquired solution. Since the phase considers a very small amount of instances worth spending resources on, the PDP search should be prolonged by several thousands of iterations. After the final phase finishes, the transfer setups present in the best PDP instances with their best found solutions form the output of the PDPT solver.

Important advantage of the proposed scheme is its embarrassingly parallel nature. All the PDP solver runs within one phase are completely independent of each other. The PDPT solver is designed to utilize 8 CPU cores in the default setup, but the execution time and stability of the results can be easily improved by providing additional resources. The only point to keep in mind are the values of parameters responsible for instance front size, filtering and duplication. The choice of these parameters should reflect the number of available CPUs. In ideal case, all the available CPUs may be utilized almost completely during the whole course of the search.

5.2 PDP solver component

In principle, the PDP solver serves two purposes simultaneously. First, it is used to acquire short evaluations resulting in upper bound estimates of the instance solution quality. These estimates are then used to compare instances during filtration steps. Second, the sequence of PDP solver runs on an instance surviving both filtering steps principally forms one longer PDP optimization of the given instance.

One of the notable features of the PDPT solver design is its independence on the implementation of the PDP solver component. The PDP solver is generally treated as a black-box component with a simple well-defined interface. As an input, the PDP solver component must be able to take the input instance, optionally an initial solution and the number of iterations of the requested search. The output of the component is the best solution encountered during the requested search.

In order to obtain the PDP component for the PDPT solver, the solver from [1] underwent refactoring and was extended to provide the described functional interface. The interface implementation was decided to communicate via files, partly in order to keep the record of the whole PDPT search procedure progress. Apart from the ability to start the search from a given initial solution, the implemented adjustments required minor implementation efforts. It is worth noting that any reasonable PDP solver implementation can be adapted to provide the described interface with minimum insights into the internal logic of the PDP search.

5.3 Instance front generator component

The proposed PDPT search scheme is capable of sorting out the best transfer setups from the given instance front and provide high-quality solutions to the selected instances. The role of the front generator component is to reasonably cover the space of transfer-aware instances for the search. Clearly, the cost to obtain a quality estimate for an instance allows to consider a very limited number of transfer setups only.

The key idea used for the initial front generation is to consider transferring requests on the level of groups rather than individually. First, the effects of transferring a single request cannot be usually distinguished since the results of the PDP solver are subject to non-trivial variance[1]. Second, the consolidation of request loads in the transfer points forms an important part of the transfer benefits. Discovering the possibilities to serve multiple requests transferred via the same transfer point in one vehicle is easier when new transfers are introduced in larger groups. Lastly, considering request transfers per individual requests is not tractable. Even the number of potentially transferable requests exceeds the number of instances that can be realistically evaluated during the initial phase of the search.

Second important idea is that similar transfers should be applied all together, or not at all. Thus, the goal is to aim for groups of transfers homogeneous in their properties. Natural way to obtain such groups is to employ a clustering algorithm. Pseudo-code of the procedure used to obtain the initial instance front discussed throughout this section is shown in Figure 5.2.

5.3.1 Split schemes

The outlined clustering is performed on the level of *split schemes* rather than requests. Split scheme is a pair of request and transfer point[2]

1. Moreover, the variance of results is significantly higher for shorter runs used to obtain the quality estimates in the proposed scheme.
2. The text gives a description for split schemes with single transfer point only. The implementation, however, allows for double transfers for requests with both pickup and delivery not coinciding with any transfer point. All the described concepts and measures can be extended to multi-transfers in a straightforward manner.

representing a possible way to transfer the given request. The broadest possible set of split schemes is obtained as a Cartesian product of the set of requests and the set of transfer points. Not all such split schemes are, however, feasible or even remotely sensible. This includes split schemes which cannot be applied due to time constraints, reflexive split schemes transferring the request via its own pickup or delivery location or split schemes inducing detour clearly not worth further consideration. The first preparation step before the clustering is to filter the broadest set of split schemes and keep only potentially useful split schemes. The filtration is done based on predefined thresholds set to very benevolent values[3].

1: **procedure** INSTANCEFRONT(R, T)
2: $S := R \times T$
3: remove clearly irrelevant schemes from S
4: $C :=$ CLUSTERING(S)
5: $C_{scores} :=$ SCORE(C)
6: eliminate same-request schemes within clusters (keep best)
7: sort clusters in C by C_{scores} (ascending)
8: $I := \emptyset$
8: **for** i **in** $0 \ldots C$.length $- 1$
9: $I := I \cup$ CLUSTERSTOREQUESTS(R, $C[0,..,i]$)
10: **return** I

Figure 5.2: Pseudo-code of the instance front generator procedure.

In order to cluster the split schemes, it is necessary to identify their properties relevant to the desirability of their application. From our experience and obtained insights outlined in the next paragraph, total number of 7 split scheme properties was selected. First 3 factors are *weight*, *volume* and *pickup-to-delivery distance*. These factors are independent of the transfer point used by the split scheme. The remaining 4 factors are *absolute* and *relative detour*, *slack* and *uniformity* of the split scheme.

3. These threshold values are not subject to tuning as they only serve to eliminate split schemes which are clearly out of question.

Regarding weight and volume, transferring smaller requests is generally more beneficial as it allows to consolidate larger number of requests at the transfer point into one vehicle. On the other hand, consolidation of larger requests may become problematic due to capacity constraints. Next, the benefits of transfers were observed to increase with the growing pickup-to-delivery distances. The detour factors present direct quantification of potential cost penalties induced by applying the split scheme. Formulas for absolute and relative detour are shown in Equations 5.1 and 5.2, respectively. The slack of the split scheme is the time available between earliest pickup and latest delivery reduced by the travel times and service times necessary at the pickup, transfer point and delivery. Lastly, the uniformity factor reflects then fairness of pickup-transfer-point-delivery distance distribution between the created requests. Fairness of the distance distribution was identified to positively affect the usefulness of split schemes. The formula for uniformity measure is in Equation 5.3. *P-TP* is the distance between pickup and transfer point of the split scheme. *TP-D* is the distance between transfer point of the split scheme and delivery. *P-D* represents the distance between pickup and delivery of request in the split scheme. Lastly, *F* is the fair half of the tour from pickup to delivery via the split scheme transfer point.

$$\text{absolute detour} = \text{P-TP} + \text{TP-D} - \text{P-D} \tag{5.1}$$

$$\text{relative detour} = \frac{\text{P-TP} + \text{TP-D}}{\text{P-D}} \tag{5.2}$$

$$\text{F} = \frac{\text{P-TP} + \text{TP-D}}{2}$$

$$\text{uniformity} = \frac{|\text{P-TP} - \text{F}| + |\text{TP-D} - \text{F}|}{\text{P-TP} + \text{TP-D}} \tag{5.3}$$

5.3.2 Clustering and instance generation

Based on the properties discussed in previous subsection, it is now possible to cluster all the reasonable split schemes. Before the clustering takes place, all the 7 dimensions are standardized. Second, each di-

mension is stretched or shrunk by its associated weight. Then, the split schemes are clustered with Euclidean distance serving as the metric. The dimension weights W and the number of clusters $C_{\#}$ produced at this point are parameters of the PDPT solver. Pseudo-code of the clustering procedure is in Figure 5.3.

```
1: procedure Clustering(S)
2:     standardize clustering properties in schemes from S
3:     transform properties in S by dimension weights W
4:     // Standard clustering algorithm, e.g. k-means or
5:     // hierarchical agglomerative clustering
6:     return ClusterSchemes(S, C#)   //   Euclidean metric
```

Figure 5.3: Pseudo-code of the clustering procedure.

Next step is to score the created clusters. The cluster scores will serve to measure how desirable is the application of a typical split scheme from the given cluster. First, a centroid is calculated as a representative of each cluster. Then, the cluster centroids are assigned ranks in each of the 7 clustering dimensions. The final score of each cluster is given as a weighted sum of the 7 centroid ranks weighted by the dimension weights used during clustering. The procedure computing the cluster scores is in Figure 5.4.

```
1: procedure Score(C)
2:     rank cluster centroids from C in each property separately
3:     return sum of ranks weighted by W (per centroid)
```

Figure 5.4: Pseudo-code of the cluster scoring procedure.

So far, a split scheme cluster may contain multiple schemes of one request. In case the cluster is to be applied, it is necessary to resolve such duplicates. In order to address this situation, schemes sharing same cluster and request are scored identically as the cluster centroids

in the previous step. Then, only the best scheme per cluster and request is kept and the remaining schemes are discarded. As a result, the clusters may be applied separately without any ambiguity.

In order to generate the instance front, the clusters are sorted in the order of their desirability based on the calculated scores. Then, instances are generated by selecting the best N clusters as active for N from one up to the number of clusters. Thus, the number of instances in the front matches the parameter setting the number of clusters.

One particular instance in the front is created based on a set of active clusters and their scores. The produced instance contains each of the original requests either as a direct copy or in the form of two requests created by an applied transfer. The first case appears if the request in question does not have any scheme in any of the active clusters. In the opposite case, one or potentially more clusters containing a scheme of the request exist. If there are multiple such clusters, scheme from the best scored cluster is preferred. The original request is then divided into two requests based on the preferred scheme. The time windows at the transfer point location are set so that the available slack is distributed fairly between the two requests. The construction of a new instance from a set of active clusters is illustrated in Figure 5.5.

At this point, it is also necessary to track the mapping between the original requests and requests in the created instance. In case this information is not tracked, a solution of the created PDP instance could not be transformed into a transfer-aware transportation plan.

```
1: procedure ClustersToRequests(R, C_active)
2:     R' := ∅
3:     for r in R
4:         if r not in any scheme in any cluster in C_active
5:             R' := R' ∪ {r}
6:             continue
7:         find best scored cluster c containing scheme s of r
8:         r_1, r_2 := s.GetSplitRequests()
9:         R' := R' ∪ { r_1, r_2 }
10 :   return R'
```

Figure 5.5: Pseudo-code of the instance creation procedure.

Overall, the selected approach aims at two targets. First, the clustering produces a set of transfer features based on relevant properties rather than on fixed sized groups. This allows to generate the instances with the structure of the available transfer options in mind. Second, the scale of transfer application percentage is effectively covered by the cluster accumulation strategy used to produce the instances. Notably, the scale is covered unevenly due to the different-sized clusters. This, however, present an advantage rather than a drawback as the coverage density is concentrated in important areas of the scale. Ultimately, the mechanism generating the instance front is completely deterministic. Attempts to form instances by including the clusters more or less randomly were made, but they always produced lower-quality and less stable results. The whole generative process starting from the original requests and ending with a front of instances is illustrated graphically in Figure 5.6.

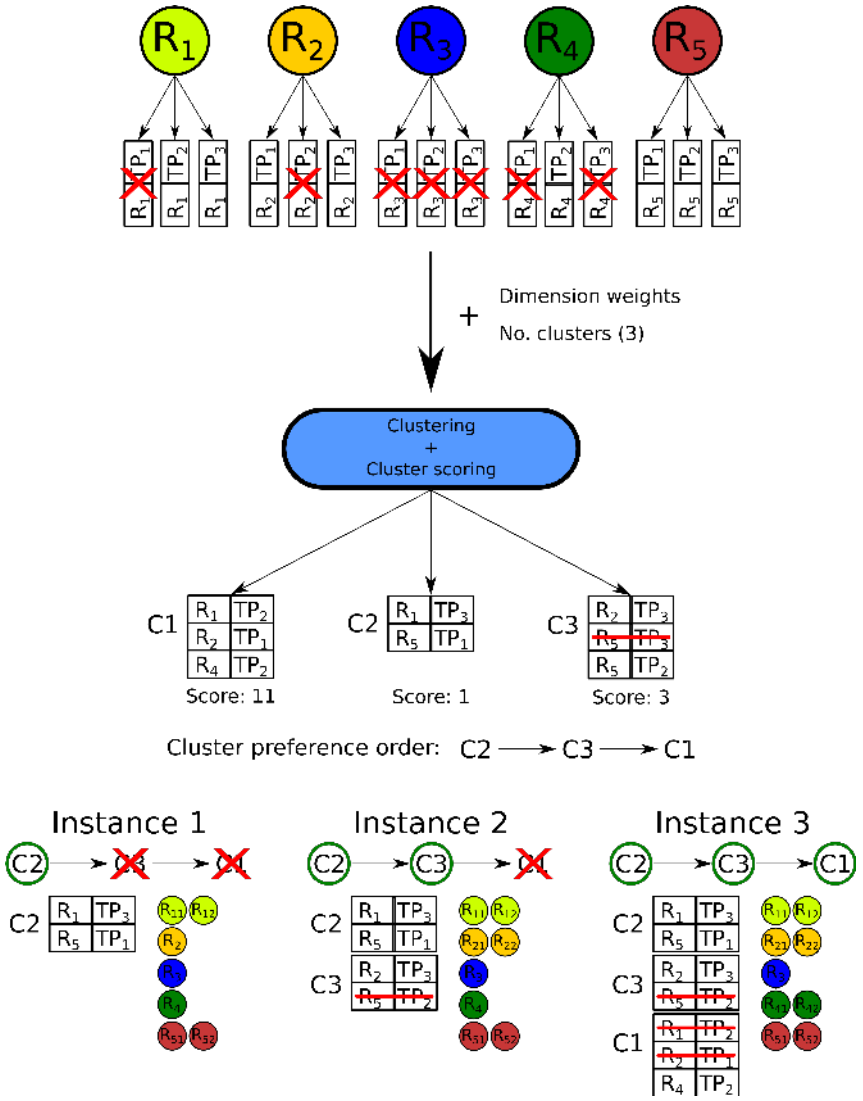

Figure 5.6: Instance front generator schema.
Split schemes are eliminated on three levels. First, schemes with clearly undesirable properties are filtered out (very large detour, negative or very small slack, reflexive schemes). Second, schemes may be eliminated within a cluster. If one cluster contains multiple schemes of one requests, only the best scheme is kept and others are discarded. Lastly, schemes may be eliminated during instance construction. If multiple schemes of one request from different clusters shall be applied, scheme from the best scored cluster is selected. Note the example of request R_3 which cannot be transferred in any way. During instance generation, it is treated the same as requests with no split schemes in the selected clusters.

5.4 Prior solver designs

The presented PDPT solver framework is not the only concept tested during the course of this work. Four alternative approaches that ultimately led to the chosen design are presented as they provide valuable insights into the limits of methods deciding the transfers apriori.

The first implemented approach attempted to identify requests suitable for transfers in one shot. The implementation relied on rule-based decisions driven by a set of thresholds. The request properties used in favor of transfers were mainly the pickup-to-delivery distance together with load volume. Moreover, information about clusters of pickup and delivery locations were used in attempt to take the relative positions of request locations into account. Thresholds on detour and slack, on the other hand, were used to disallow potential transfers. Interestingly, the attempts to exploit the location clusters in order to force request consolidation proved to play marginal effect on the performance. Arguably, the biggest disadvantage of this approach are the strict orthogonal threshold-based decision boundaries.

Second attempted method concentrated on the tradeoff between additional flexibility and drawbacks introduced with added transfers. The idea was to introduce an easy to calculate measure correlated with the best achievable instance quality. Then, instances with various transfer setups could be compared quickly and the costly PDP search would be executed only on a handful of instances with the best scores. Unfortunately, it proved to be unrealistic to design reasonable measures with the required properties.

Next method aimed to analyse the typical traffic patterns occurring in solutions of the target instance. The instance was solved several times without transfers for a lower number of iterations. Then, a traffic heatmap was assembled based on these solutions. The heatmaps captured the intensity of traffic between relatively small regions in a grid-based system. The idea was to transfer requests so that they may copy the patterns of typically created traffic flows. Despite the method attempts to base the transfer decisions on more than just static information about the requests, this relatively sophisticated approach did not prove to be much useful.

The last of the four attempts is relatively close to the final approach. During the design of the heatmaps, it turned out that the obtained ben-

efits are far more dependent on the overall percentage of transferred requests rather than on a careful choice of the requests to be transferred. In order to exploit this observation, one best split scheme was decided for each request greedily based on the relative detour criterion. Before this step, the full set of split schemes was filtered similarly as described in Section 5.3.1. Then, the requests were sorted based on the relative detour of their chosen split scheme and instances with transfers were derived by cumulatively adding fixed sized groups of split schemes in the given order. Around 10 instances were produced in this way. The instances were then solved in parallel for 1,000 iterations in 8 duplicates each. The high duplication factor often allowed for finding high-quality solutions even on the short 1,000 iteration runs. This method sometimes managed to find solutions of comparable quality to the final implementation, it, however, suffered two problems. First, the method wasted unnecessary computation resources on instances not worth consideration, mainly due to the high duplication factors. Second, the coverage of the transfer percentage scale was rather sparse which often led to missing important points due to the insufficient coverage.

Across all of the attempted approaches, several ideas proved to be very useful. First, the overall percentage of transfers in the instance is far more important than very careful selection of requests to be transferred. Second, the best way to assess quality of transfer decision is to simply run PDP solver on the instance. It is, however, crucial to invest the computational resources carefully. Lastly, thresholding is useful for elimination of unreasonable split schemes, but critical decisions based on thresholds are too rigid and do not allow for interactions between the thresholded properties.

6 OpenStreetMaps instance generator

The goal of the generator is to respect reasonable assumptions on geographic distribution of locations and transfer points and allow for complete control over distributions from which request properties are generated. Based on these requirements, it was a natural choice to build the generator upon real geographical datasets. One of the widely recognized and open collections of geographical data is the project OpenStreetMaps (OSM)[1]. The generator implementation is available as a part of the thesis in the IS MU.

The process of instance generation has two major phases. First, the downloaded OSM package for a region (typically one country) is preprocessed into an intermediate JSON file. Second, the generator uses the preprocessed JSON and provided configuration to produce PDPT instances. The path from the OSM package to the generated PDPT instances is presented step-by-step with emphasis on the transfer point placement as the most interesting part of the process.

6.1 Location extraction

The main purpose of using the OSM datasets is to obtain realistic locations for pickups, deliveries and transfer points with similar properties to our real-world problem. The goal of the preprocessing is to extract and clean candidate locations for these purposes. The first step is to download OSM package for the required region in the *.shp.zip* format. OSM packages for particular countries are regularly exported and maintained by the company Geofabrik based in Germany. These packages are available for download online.

In the original problem, the request pickups are very often located in large logistical facilities which serve as the transfer points. The rest of the pickup locations is rather scattered across the region. Based on this observation, industrial zones were extracted from the OSM package as the image of the scattered locations. Together with the

1. © OpenStreetMap contributors. The OSM datasets are available under the Open Data Commons Open Database License [22].

transfer points, these locations serve as the candidates for pickups in the generated instances.

The vast majority of delivery locations in our problem is situated to larger towns or cities. In order to mimic this characteristics, locations of supermarkets were extracted from the OSM package. Similarly to pickups, these locations together with transfer points form the set of candidates for delivery locations in the generated instances.

Next, the extracted locations are filtered based on the local outlier factor [23] technique for outlier detection. The main purpose of this step is to prevent unwanted artifacts arising from inclusion of detached territories such as distant islands. Consequently, the pickup-delivery location pairs taken from the filtered candidate sets should be enclosed in a reasonably compact area.

6.2 Transfer point placement

In contrast to pickup and delivery locations, there is very few transfer points and their placement within the region is far from arbitrary. In fact, the choice of location for a large logistical facility serving as a transfer point is an important strategical decision. Clearly, the optimal placement is dependent on distribution of serviced locations within the region and on the placement of other similar facilities.

The outlined problem is a well-known NP-hard optimization problem of operational research called *facility location problem* (FLP) [24]. The goal in FLP is to find optimal placement for a given number of facilities so that the set of given customer locations is covered in the best possible way. In order to allocate the transfer points at sensible locations, the preprocessing phase of the generator solves a variant of FLP. The time demanded for this step is the main reason for the separation of preprocessing and instance generation.

Calculating FLP for all the extracted locations would be intractable. Since approximate solution is sufficient for the given purpose, the locations are first clustered to a fixed number of clusters. The clusters are replaced by centroids and weighted by the number of members. Then, a weighted variant of FLP (WFLP) is solved with the centroids serving as both set of customer and facility candidate nodes. WFLP is solved repeatedly resulting in precalculated positions of transfer points for several setups with different number of transfer points.

6.3 Weighted facility location problem model

The WFLP in the generator preprocessing phase was formalized using ILP and solved by the SCIP solver [25].

A solution to the problem places exactly k facilities at some of the candidate locations and pairs each customer location with one of these facilities. The facility placement and customer-facility assignment should minimize the sum of distances of customers to their assigned facility weighted by the importance of particular customers.

As expressed in Constraint 6.2, each customer is associated with exactly one facility. This is the facility candidate node from which the customer shall be serviced. From the customer's perspective, such facility should be as close as possible. Constraint 6.3 ensures that a customer may be assigned to a facility candidate node solely if the candidate node was selected to host a facility. Lastly, Constraint 6.4 secures that exactly k facility candidates will be selected.

$$\text{minimize} \quad \sum_{i \in C} \sum_{j \in F} d_{ij} \cdot x_{ij} \cdot w_i \tag{6.1}$$

$$\text{subject to} \quad \sum_{j \in F} x_{ij} = 1 \qquad \forall i \in C \tag{6.2}$$

$$x_{ij} \leq y_j \qquad \forall i \in C, \forall j \in F \tag{6.3}$$

$$\sum_{j \in F} y_j = k \tag{6.4}$$

Table 6.1: FLP model notation.

C	Set of the customer nodes
F	Set of the facility candidate nodes
d_{ij}	Distance between the nodes $i, j \in C \cup F$
w_i	Weight of the customer location $i \in C$
x_{ij}	Decision binary variable expressing that the customer node $i \in C$ is served from the facility in the node $j \in F$
y_j	Decision binary variable expressing that a facility is placed in the node $j \in F$

6.4 Instance generation process

The instance generation takes the extracted locations and selects one of the precalculated transfer point configurations based on the required number of transfer points. Together with the provided configurable distributions[2], requests, vehicle fleet and time and distance matrices are generated.

Regarding the vehicles, a heterogeneous fleet is generated. There are four vehicle types in the fleet ranging from vans to large trucks. Their properties (capacity limits, price per kilometer) reflect the experience provided to us by Wereldo. These four vehicle prototypes are duplicated ample number of times into each vehicle depot physically coinciding with the transfer points.

The request generation first splits the total required number of requests into four groups based on a provided distribution. These groups reflect the *logistical context* of a request, i.e., whether the locations of its pickup and delivery coincide with some transfer point (serving as a depot). The requests from these groups are generated separately in the appropriate numbers.

Properties of requests such as time windows[3], number of pallets or weight[4] are drawn from configurable distributions. Service time is subject to global setting shared among all requests and locations including transfer points. The generative process includes a check so that only deliverable requests may be generated.

Finally, a Haversine[5] distance matrix is calculated based on the longitude-latitude coordinates of the locations. The time matrix is then derived based on a constant configurable vehicle travel speed. The final output of the generator are two JSON files. The first is the PDPT instance while the second describes longitude-latitude coordinates of the particular locations used in the instance.

---

2. The full set of configurable parameters with examples is documented in the project README file. Multiple convenience configurations are prepared as well.
3. Time windows are generated only for locations not coinciding with transfer points, transfer points are assumed to operate non-stop. Options and distributions are assumed separately for pickup and delivery locations.
4. Request weight is obtained by generating the number of pallets in the request first and then generating weights of the individual pallets from a distribution.
5. Shortest distance between two points accounting for the curvature of Earth.

7 Experiments

The purpose of this chapter is to assess the proposed method. In Section 7.1, properties of the instances used for experiments are presented. In Section 7.2, the choice of parameter setup is justified. In Section 7.3, insights into the behavior of the algorithm are discussed. Lastly, Section 7.4 reports numerical results of the conducted experiments.

7.1 Data

The proposed implementation was experimentally assessed on two sets of instances. First, the data provided by Wereldo were preprocessed and separated into 5 single-day instances with around 1,200 requests each. Second, the instance generator introduced in Chapter 6 was used to create 12 new instances based on the characteristics extracted from the real-world dataset. Different numbers of requests and transfer points were used on two separate geographies as listed in Table 7.1. Since the thesis was prepared in cooperation with the company Wereldo, only the generated instances are available as a part of the thesis in the IS MU.

Table 7.1: Summary of generated instances.

Instance name	Geography	No. TPs	No. requests
bulgaria_500_1	Bulgaria	3	500
bulgaria_500_2	Bulgaria	4	500
bulgaria_500_3	Bulgaria	5	500
bulgaria_1000_1	Bulgaria	3	1,000
bulgaria_1000_2	Bulgaria	4	1,000
bulgaria_1000_3	Bulgaria	5	1,000
czechia_500_1	Czechia	2	500
czechia_500_2	Czechia	3	500
czechia_500_3	Czechia	4	500
czechia_1000_1	Czechia	2	1,000
czechia_1000_2	Czechia	3	1,000
czechia_1000_3	Czechia	4	1,000

The global numerical characteristics of the real-world dataset are summarized in Table 7.2. Note that despite the formal model allows for different service times and route limits per location and vehicle, these features have always one value with no exceptions.

Table 7.2: Global characteristics of the dataset.

Feature	Value
Total no. requests	6,539
No. requests single day	cca 1,200
Service time	15 minutes
Maximum route duration	11 hours
Maximum no. stops in route	15
Average vehicle speed	19 m/s
No. transfer points	4
No. vehicles	323

The key characteristic of the dataset is the distribution of requests among different logistical contexts. Slightly less than 85 % of the requests are delivery-only[1] and around 10 % of requests do not pick up from nor deliver to the transfer point locations at all (pure pickup-delivery context). The remaining portion of requests is represented by pickup-only requests (delivery to transfer point), transportation between two transfer points appears rather marginally.

Next distinct feature of the data is the number of pallets and its relationship to the logistical contexts. The distribution of pallet counts in the dominant delivery-only context is shown in Figure 7.1. In the pure pickup-delivery context, the relative trends are the same, but around 30 % of these requests are full-truck loads. Pickup-only requests are, on the other hand, clearly dominated by full-truck loads.

1. Delivery-only requests have their pickup in one of the transfer points (depots).

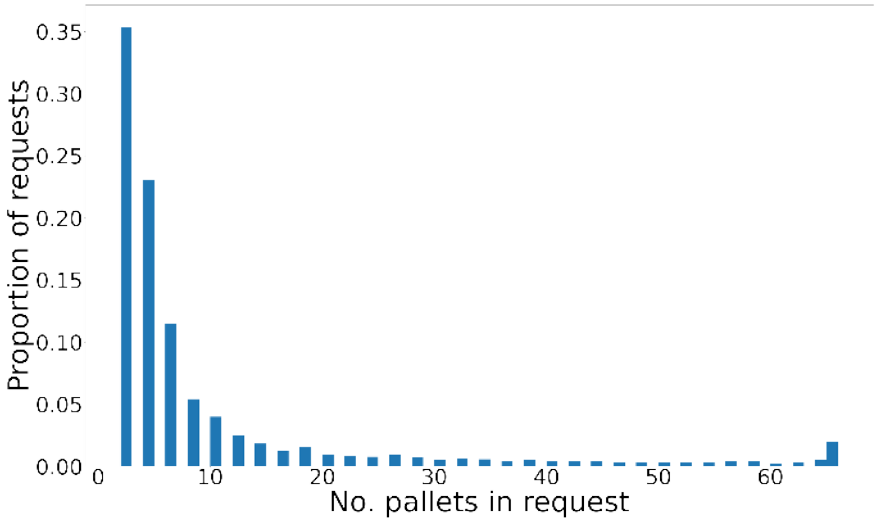

Figure 7.1: Distribution of no. of pallets in delivery-only context.

Regarding the time windows, strong regularities were observed. The transfer point locations operate non-stop for both pickups and deliveries. Pickup time windows outside transfer points start regularly at one day-hour and last for 13 hours. Delivery time windows outside transfer points dominantly start at two different day-hours 6 hours apart, but also delivery windows starting at different times appear. The most variable time-related aspect is the length of the delivery time windows ranging from less than an hour up to 12 and even 24 hour windows.

Lastly, the vehicle fleet is heterogeneous both in capacities as well as in operational costs. Regarding capacities, a significant portion of the fleet are large trucks capable of accommodating 66 or 72 pallets (when double-stacked) and up to 24 tons of load. Operational costs mainly vary with the vehicle type.

7.2 Parameter settings

Two major groups of parameters can be identified in the implemented method. Parameters in the first group generally affect the extent of the search procedure. This includes the initial front size given by the number of clusters, number of iterations in different search phases, run replication factors in different phases and strictness of filtering in between subsequent phases. The second parameter group is formed by the dimension weights used during the clustering and cluster scoring. Part of the parameters was set based on empirical experience with the rest being set by an automated tuning procedure.

The replication factors and numbers of iterations were set empirically. The choices reflect the experience with the tradeoff between the time consumed by the PDP solver runs and variances in the instance cost estimates. Generally, running the PDP solver for more iterations and in more duplicates is always beneficial for the quality and stability of the instance cost estimate. Unfortunately, the price for the better estimate is a significant amount of additional computation time in both cases. Consequently, it is advisable not to invest computation time if there is no need for further estimate precision or stability.

The choice of 500 iterations for the initial phase proved to be enough to distinguish significant cost differences between instances[2]. In order to support the stability in the initial phase, the 500 iteration PDP solver runs are executed in 2 duplicates. During the intermediate phase of the search, the goal is to prolong the search on the promising instances. Experience with full 25,000 iteration runs on various instances showed that solutions after 2,000 iterations are relatively close to the the value obtained after the full run[2]. Thus, the intermediate phase adds 1,500 more iterations on solutions obtained from the initial 500 iteration runs. Lastly, the final search prolongs few best solutions by 5,000 more iterations in 2 duplicates. As verified experimentally, prolonging the final search phase would not lead to any significant improvements[2]. The duplication, on the other hand, proved to be useful in longer runs.

2. Further discussion and visualizations are presented in Section 7.3. Specifically, the qualities of results at the phase borders are illustrated in Figures 7.3 and 7.4. The effects of final phase prolongation is shown in Figure 7.4.

The last part of parameters chosen empirically are the filters in between the search phases. Reduction of the initial front of instances to 16 best only aims to eliminate a large portion of instances from consideration while being benevolent enough to compensate for low-quality estimates on good instances. Based on the experiments from Section 7.4, the filtration at this point could be potentially slightly more strict without significant impacts on the quality of the final solution. Second, the choice to keep 4 best instances into the final search phase aims at two targets. First, it is to diversify the final search and potentially obtain multiple high-quality solutions with different transfer setups[3]. Secondly, the choice of 4 instances in 2 duplicates serves to keep the expected number of 8 processors busy. Alternatively, 2 instances in 4 duplicates could be considered to provide more stable results at the expense of the solution diversity. Values of the parameters affecting the search extent are summarized in Table 7.3.

Table 7.3: Solver parameters – search extent.

Parameter	Value
Number of clusters	40
Init. phase iterations	500
Init. phase replication	2
Filter init.	16
Mid. phase iterations	1,500
Mid. phase replication	1
Filter mid.	4
Final phase iterations	5,000
Final phase replication	2

Second major part of the parameter setting revolves around the choice of appropriate combination of the number of clusters and dimension weighting. The maximum number of clusters directly affects the size and granularity of the generated instance front. The dimension weights are used to first transform the split scheme space before

3. In practice, the solution with the best objective is not necessarily the best from the user perspective. Thus, producing multiple high-quality solutions with reasonable diversity is definitely advisable.

clustering and then for scoring of the created clusters. The weighted dimensions are the 7 split scheme properties identified to be relevant for split scheme applicability as discussed in Section 5.3.1. The dimension weights were tuned separately for values 20, 30, 40 and 50 of the parameter configuring the number of clusters. For each of the four cluster counts, Bayesian optimization [26] was employed in order to identify the best dimension weights.

For the purpose of automated tuning, two additional instances were generated. The tuning instances are available as a part of the thesis in the IS MU. Both instances were based on the Czechia geography and included 3 transfer points. A shorter run of the Bayesian optimization was applied on the smaller instance with 200 requests in order to provide reasonable starting points for the main tuning. Then, the same optimization procedure was initialized with the obtained starting points and executed on the larger instance of 500 requests for 35 iterations. This step was done separately for all the four cluster counts. The distributions of costs encountered during the course of tuning are reported separately for each of the four cluster count values in Figure 7.2. A complete overview of the parameter configurations evaluated during the automated tuning and the respective acquired costs are available in tabular form in Appendix B.

The best two dimension weightings were obtained with 40 clusters and shared the same pattern. First, the tuning resulted in elimination of the *slack*, *detour relative* and *weight* dimensions. In case of slack, the amount of available spare time seems to play minor to no role given the minimum slack of around one hour is guaranteed as in the implementation. In case of *detour relative* and *weight*, these dimensions seem to be rendered unimportant due to their strong correlation with their more favored counterpart dimensions *detour absolute* and *volume*. Second, the tuning reveals that the primary dimensions for the splitting decision are *detour absolute* and *pickup-delivery distance*. The measure of *uniformity* and *volume* then play rather secondary role in the clustering and scoring. The best acquired dimension weights used in the final experiments are summarized in Table 7.4.

Table 7.4: Solver parameters – dimension weighting.
The available range of values was set from 0.2 to 2.0. Notably, the obtained configuration often prefers the border values of this range.

Parameter	Value
Detour absolute	2.0
Detour relative	0.2
Pickup-delivery distance	2.0
Slack	0.2
Uniformity	1.28
Volume	0.91
Weight	0.2

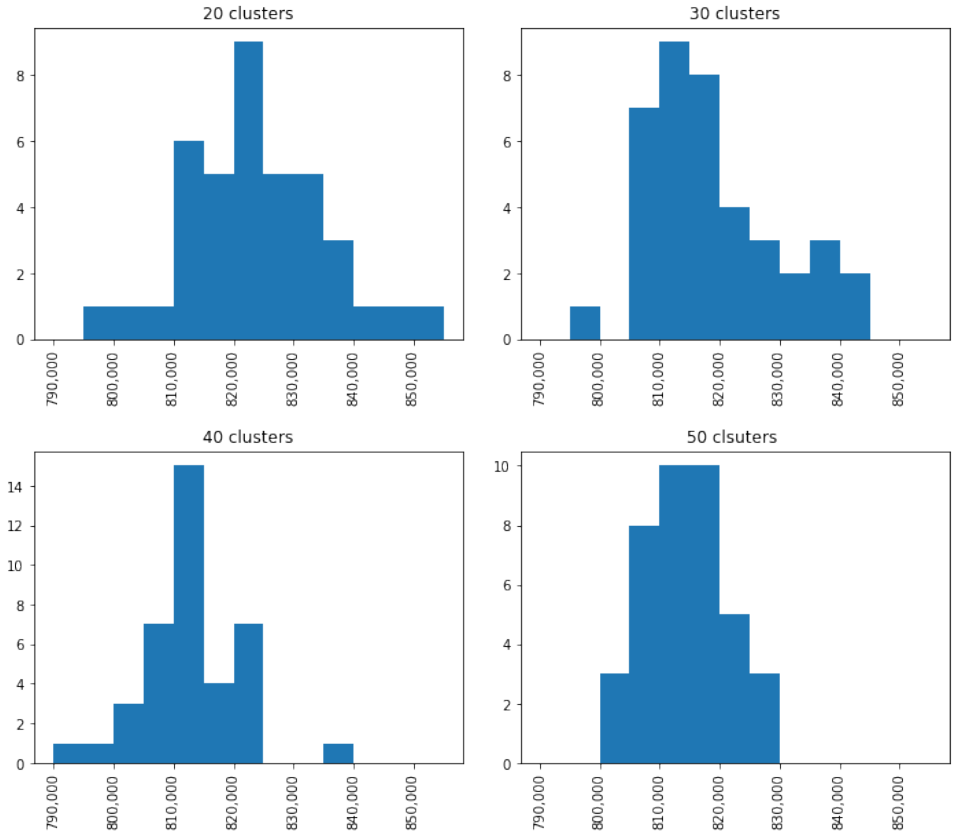

Figure 7.2: Distributions of costs achieved during tuning.

Histogram bins size is 5,000 cost units. Y-axis shows the number of runs belonging to the given bin. The best cost 792,243 was achieved with 40 clusters.

7.3 Solver run insights

In order to demonstrate reasonable convergence of the PDPT solver and give insights into the search progress, plots of runs on a representative instance are presented. The instance *czechia_1000_2* was chosen as representative as it achieved average transfer benefits. As a demonstration of typical behavior of the PDPT solver, an average run of the selected instance is plotted in Figure 7.3[4].

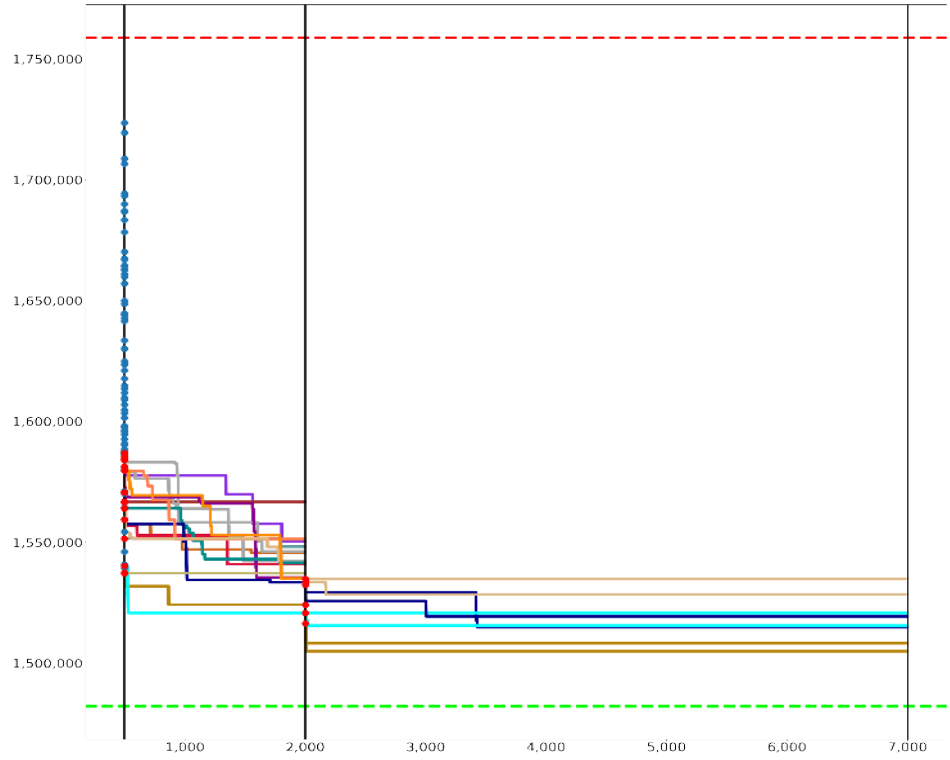

Figure 7.3: Search progress on average run of *czechia_1000_2*.

The X-axis represents the number of iterations in the PDP search, the Y-axis is the best achieved cost so far. Vertical lines at 500, 2,000 and 7,000 iterations are the borders of search phases. Points at the 500 iteration border represent results of runs from the initial phase. Red points are runs accepted to the next phase. Red and lime dashed lines represent best achieved baseline and PDPT solution achieved over the 10 runs on the given instance.

4. Note that the group of red points at the end of the initial phase also contains few blue points. This is due to the fact that each instance is executed in two duplicates, but a fixed number of instances rather than runs is accepted into the next phase.

As discussed in Section 7.2, the choice of 5,000 iterations for the final phase of the search is sufficient. In order to support this claim experimentally, the best runs were taken for each instance. Then, the final phase of the search was restarted with 23,000 iterations instead of the usual 5,000 iterations. Neither of the prolonged runs achieved improvement over the best result on the given instance. Consequently, terminating the PDP search after 7,000 iterations is not to be seen as premature. An example of the best run prolongation on the representative instance is shown in Figure 7.4.

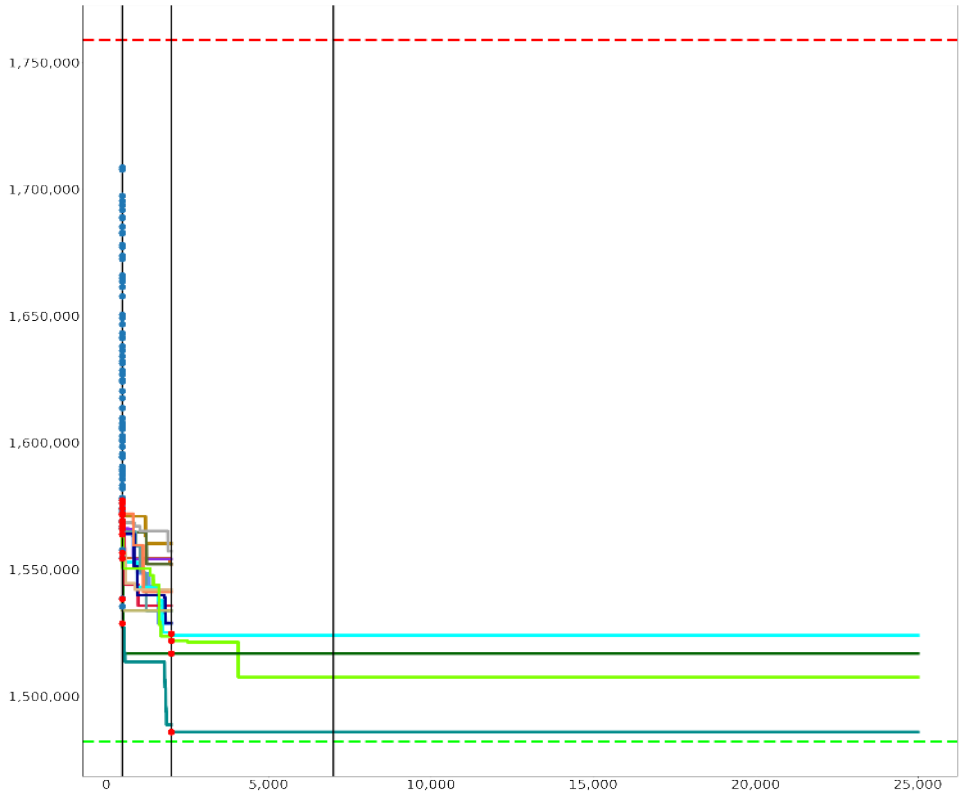

Figure 7.4: Search progress on a prolonged best run of *czechia_1000_2*.

As it can be seen in Figures 7.3 and 7.4, improvements are still very common in the phase between 500 an 2,000 iterations. Especially for instances with higher percentage of transfers, it often takes more than the initial 500 iterations to settle on a relatively stable best solution. The middle phase provides enough additional iterations to compensate for this fact. With quite stable instance quality estimates at the end of the middle phase, the final filtering may compare the instances with reasonable reliability.

In contrast to the dynamic middle phase, improvements occur scarcely during the final search. Generally, the usefulness of the final phase and its potential prolongation was observed to be higher on runs with lower quality results as it allows to find important patterns missed in the earlier stages of the search.

The overall convergence of the search scheme towards high-quality results relies on two factors. First, it is the instance front reasonably covering the transfer options that is provided to the scheme. Since the search scheme itself does not interact with the transfers at all, any transfer-related patterns to be assessed must be provided as a part of the front. Second, the search scheme in principle solely evaluates all of the provided instances and selects the best encountered solutions. It, however, prunes evaluations of instances which are unlikely to achieve high-quality results.

In case it is necessary to further improve the stability in the result quality, higher replication factors and prolongation of the first two search phases are recommended rather than a very long final search. First, opting for a very long final phase results in a considerable increase of the execution time of the PDPT solver. Second, the best acquired solutions were observed to be findable within lower thousands of iterations[5]. Consequently, more short runs of the PDP solver usually present a cheaper way to hit the key pattern in the search space rather than one run traversing the space long enough to eventually return to what it missed during the early iterations.

The relationship of the costs and percentage of transfers in the assessed instances presents an alternative point of view on the behavior

5. Especially instances with more transferred requests were observed to behave in this way. The baseline runs with no transfers, however, tend to find important improvements in the late stages of the search regularly.

of the PDPT solver. In order to demonstrate this dependence, 10 full runs of the PDPT solver were taken and all solutions across all the 3 solver phases were plotted together in Figure 7.5.

Figure 7.5: Costs and transfer percentage on instance *czechia_500_3*.

The X-axis describes the total number of requests in instance. The Y-axis is the cost obtained in the given run. The blue, green and red points represent instance runs from initial, middle and final phase, respectively. The lime dashed line is the cost of the best baseline run without transfers.

First notable pattern is that the best solutions are concentrated around similar percentage of transfers. Moreover, this transfer percentage serves as a breaking point of the transfer usefulness. While adding more transfers generally improves costs up to the given transfer percentage, additional transfers beyond the breaking point are harmful. The width of the breaking region differs across instances.

Second important observation relates to the coverage of transfer percentage scale. The evaluated instances range from almost zero transfers up to the maximum possible transfer percentage. The coverage of the interval is, however, clearly not uniform. The gaps in the direction of the X-axis are a direct consequence of the clustering procedure generating the initial population of instances. The larger gaps correspond to larger split scheme clusters being included into the next instance. The uneven coverage grows in importance with the instance size as it allows to address large bulks of very similar transfers together while more diverse groups of transfers may be inspected in more detail.

7.4 Evaluation

The purpose of the proposed method is to obtain cost benefits in comparison with solutions not allowing for transfers. In order to evaluate these benefits, a set of experiments was conducted on the instances presented in Section 7.1. As a baseline, each instance was solved 10 times without the possibility of transfers. These runs were set to last 25,000 iterations each while respecting the original parameter settings from [1]. Next, the implemented PDPT solver was used to solve the same instances, again 10 times each. The PDPT solver parameters were set as described in Section 7.2. The experiments were run on virtualized infrastructure[6] under CentOS 8 and the PDPT solver was implemented in Python 3.9[7].

The main focus is on the comparison of costs obtained with and without transfers on particular instances as costs present the optimized objective. Additionally, similar comparisons of overall traveled distance and number of utilized vehicles are reported in order to provide more detailed insight into the results. The comparisons are shown in Figures 7.6, 7.7 and 7.8. Lastly, overviews of the runtimes and transfer percentages are presented in Table 7.5.

6. It is important to interpret the reported runtimes with this fact in mind as the real performance of the infrastructure varies over time.
7. Python was chosen as it allows for convenient rapid prototyping. Since the performance-critical PDP search is offloaded to a dedicated external component, the implementation language of the PDPT framework plays negligible role in the overall performance.

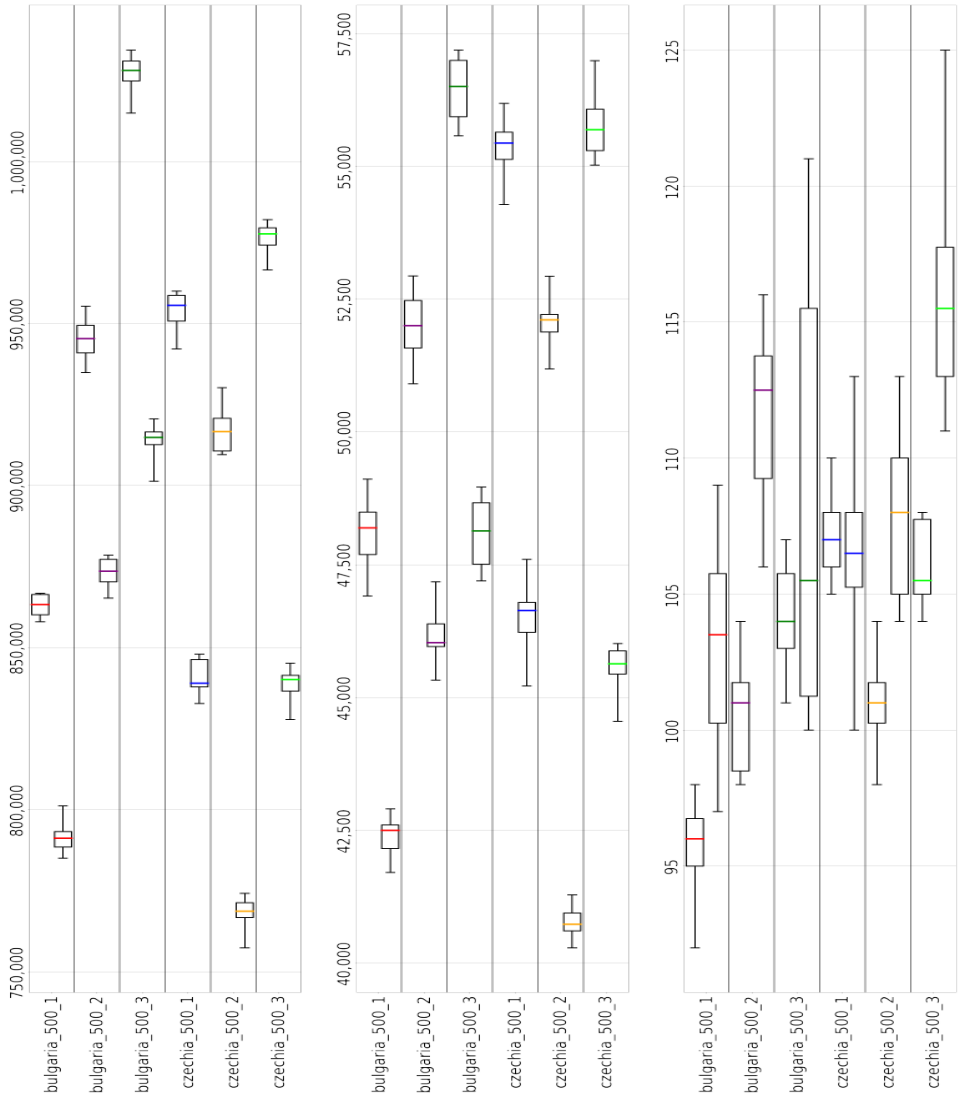

Figure 7.6: Costs, distances and vehicles on 500 request instances.

Y-axis in the three subplots represent costs, distances and number of used vehicles.

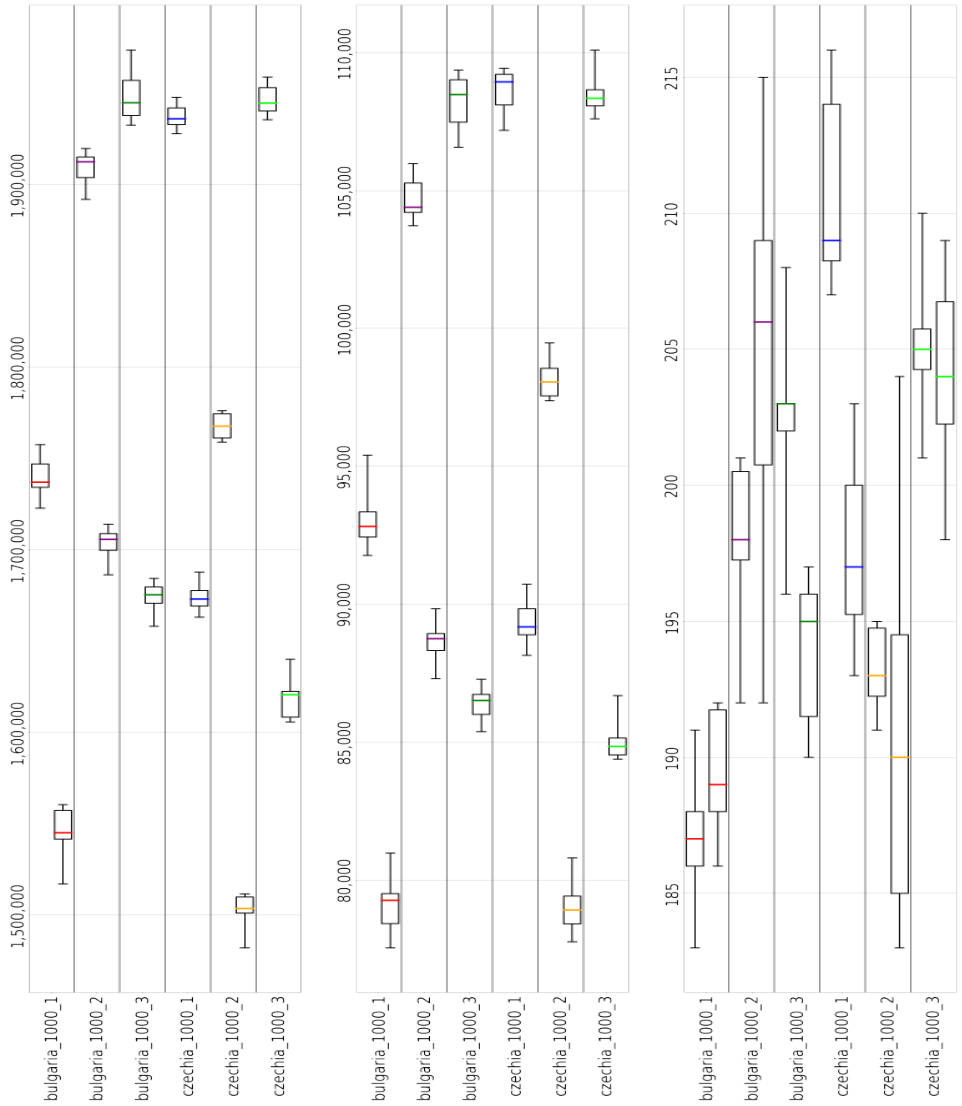

Figure 7.7: Costs, distances and vehicles on 1,000 request instances.

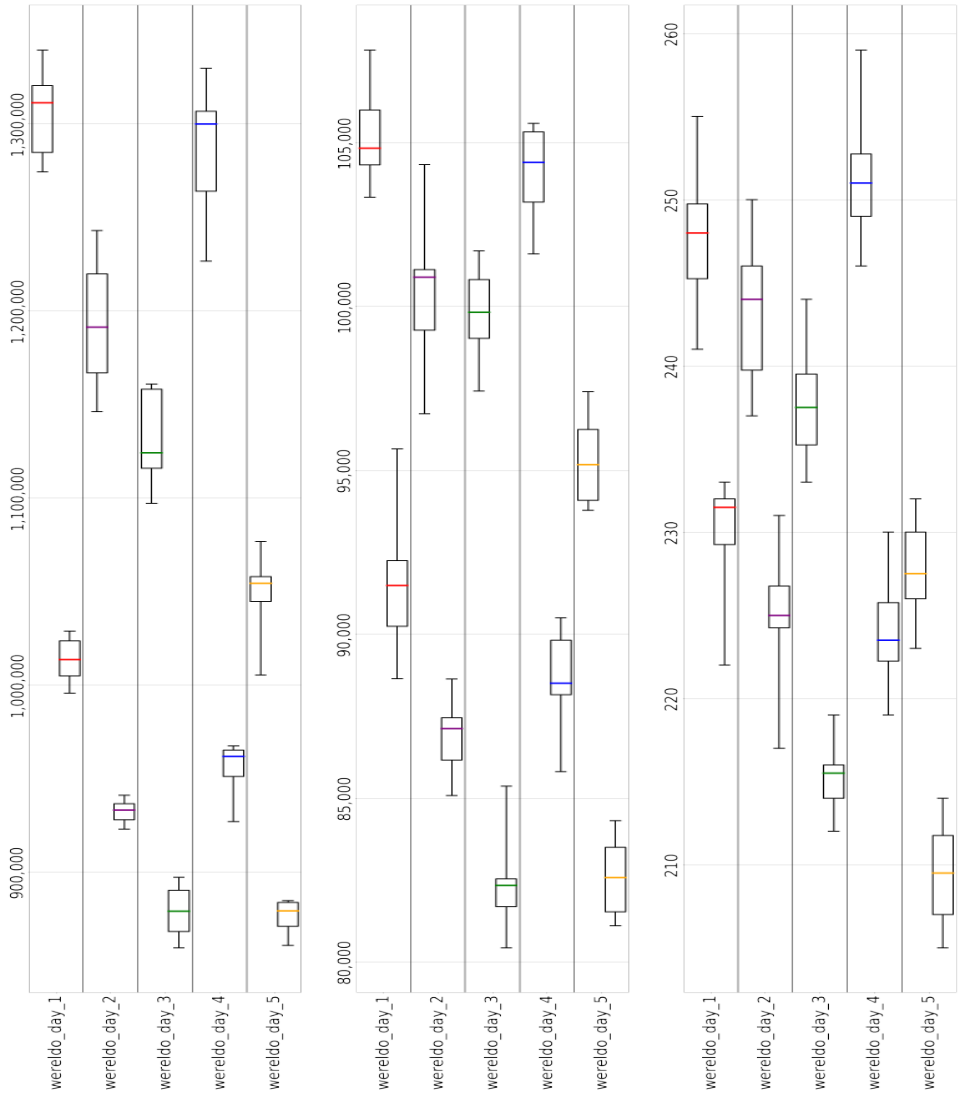

Figure 7.8: Costs, distances and vehicles on Wereldo instances.

Across all the 17 instances, introduction of transfers resulted in measurable benefits in costs. The obtained profits range from 7.5 % up to 24.5 % with average savings of around 15 %[8, 9]. Clearly, the strongest factor influencing the transfer benefits is the geography of the instances.

The weakest results were obtained on the Bulgaria geography. Instances *bulgaria_500_1* and *bulgaria_500_2* were the only one to yield transfer profits lower than 10 %. Benefits for the remaining instances were around 11 % with the exception of *bulgaria_1000_3* with more than 14 % in savings. The instances generated with the Czechia geography evince benefits around the overall average ranging from 11.5 % up to 17 %. Lastly, the most prominent benefits were obtained on the real-world instances from Wereldo. With the exception of the instance *wereldo_day_5* with only 14 % savings, profits of around 20 % were achieved. The overall best profit of 24.4 % was obtained on the instance *wereldo_day_4*.

Further factors observed to play role in the transfer benefits are the instance size and number of transfers. Among the generated instances sharing the same geography, instances with more transfer points and larger number of requests tend to benefit more from the transfers. In case of the transfer points, the key point seems to be the right level of saturation of the serviced area by the transfer point facilities. Regarding the instance size, the larger instances seem to fit the coarse nature of the proposed search method better than the smaller instances. As the instance size grows, the importance of transferring one particular request diminishes while the importance of the general percentage of the applied transfers becomes more prominent.

Speaking of the variability in the baseline and PDPT results, both are comparably stable on the generated instances. In case of the real-

8. The profits are calculated in two variants. The *best-to-best* profits compare the best baseline run with the best PDPT run while *average-to-average* profits compare the average of all baseline runs with the average of all PDPT runs. Reported values are the best-to-best comparisons, but the average-to-average values do not differ much.

9. The percentages are calculated as $100 \cdot (1 - \frac{pdpt}{baseline})$ where *pdpt* is the representative cost for PDPT solver runs and *baseline* is the representative cost of the baseline runs.

world instances, however, the baseline results vary significantly more than the their PDPT counterparts.

Similarly to costs, the introduction of transfers reduced the total covered distances significantly for all 17 instances. The reduction in distance ranges from 11 % up to slightly less than 22 %. When compared to costs, the percentual benefits are larger for distances on the generated instances, real instances have higher cost benefits than distance benefits. The difference between distance and cost benefits seems to be relatively stable for the generated instances, but varies more on the real data. Regarding the variability in baseline and PDPT results, both are comparably stable. In case of the real instances, however, the results vary more than in the generated data for both baseline and PDPT.

In case of the number of vehicles used in the solutions, the situation is different than with costs or distances. First, the number of used vehicles varies significantly more in the PDPT runs when compared to the baselines. Within one instance, the difference was observed to be up to plus minus 10 vehicles from the average. Second, the PDPT solutions often require more vehicles than the baselines. There are, however, instances on which the situation is the opposite or the number of vehicles are comparable between PDPT and baselines. Notably, the real instances are exception to the two observations. The numbers of vehicles used in the PDPT solutions are comparably stable to the baselines. More importantly, the PDPT solutions on real instances always resulted in substantial reduction in the number of used vehicles.

A summary of the runtimes and percentages of transferred requests is in Table 7.5. The values are averages over all baseline and PDPT runs on the given instance, respectively. Runtimes for PDPT runs were obtained as a sum of the computation times for all PDP tasks performed during the search (across all CPUs) as the instance front generation consumes negligible amount of time.

Speaking of the runtimes, the PDPT solver requires between 4 to 5.5 times more computational capacity than a single 25,000 iteration run of the PDP solver. The PDPT solver is, however, intended to run on multiple CPUs and allows for their full utilization for the vast majority of the search. Consequently, the PDPT solver in the default configuration with 8 CPUs finishes up to 2 times faster than a single full run of the PDP solver.

Table 7.5: Avg. runtimes and percentages of transferred requests.

Instance name	Baseline $t(s)$	Transfers $t(s)$	Transfer %
bulgaria_1000_1	2,608	12,460	21.6
bulgaria_1000_2	2,748	14,788	29.9
bulgaria_1000_3	2,925	15,646	27.8
bulgaria_500_1	1,368	5,751	18.6
bulgaria_500_2	1,492	7,209	33.4
bulgaria_500_3	1,542	7,650	28.6
czechia_1000_1	2,379	10,488	9.6
czechia_1000_2	2,659	13,214	23.9
czechia_1000_3	2,788	14,396	28.0
czechia_500_1	1,175	4,788	10.8
czechia_500_2	1,395	6,165	29.9
czechia_500_3	1,436	7,081	33.5
wereldo_day_1	3,259	14,827	19.3
wereldo_day_2	2,828	15,535	21.3
wereldo_day_3	3,201	16,723	13.2
wereldo_day_4	3,501	14,337	7.6
wereldo_day_5	3,270	14,527	13.3

In case of the transferred request percentages, neither a notable pattern nor a relationship with the obtained profits was observed. The percentage in which transfers are applied in the best transfer-aware solutions strongly varies across the assessed instances. The proportion of transferred requests ranges from 7.5 % up to 33.5 % with no clear association with the transfer benefits.

A detailed version of the results presented in this chapter is available in a tabular form as a part of Appendix B. Secondly, complete tabular results from the experiment runs are available as a part of the thesis in the IS MU.

8 Conclusion

The thesis targets the pickup and delivery problem with transfers in context of very large freight transportation problems. A formal integer programming model of the specific application was formulated. The state-of-the-art approaches for PDPT were systematically explored and summarized. A method inspired by the work of Petersen and Ropke [2] addressing the application at hand was proposed and implemented. Due to the lack of realistic benchmark instances relevant to the target application, an instance generator building upon real geographical data from OpenStreetMaps was implemented. The implemented solver was experimentally evaluated on the set of real-world instances provided by Wereldo together with synthetic instances generated by the implemented generator. The conducted experiments confirmed that the proposed approach is capable of finding significant cost benefits arising from the introduction of transfers. The achieved cost benefits ranged from 7.5 % up to 24 % with average savings around 15 %. Importantly, the strongest results were achieved on the real-world and very large instances.

8.1 Contributions

Apart from Petersen and Ropke [2], no literature assessing the potential of apriori decided transfers is available. In contrast to their work, the possibility of shifting the approach from one-shot decision towards a search in the space of instances is explored. The thesis provides a comparison of costs with and without enabled transfers. To the best of our knowledge, this comparison is missing in the aforementioned paper making this work the only one quantifying the benefits acquired by deciding the transfers apriori.

Second, the implemented method allowed for finding substantial savings on instances of sizes far beyond the scope of traditional approaches found in the literature. Together with the favorable experimental results especially on the real-world instances, the chosen direction is clearly a good fit for the target application.

Apart from performing well, the proposed method is easy to implement provided that a reasonable PDP solver implementation is

available. With a PDP solver at hand, it is arguably more convenient to use it as a component in modular system rather than reimplement it in order to support transfers. Moreover, the design is highly parallel in its nature and allows for improvements in execution times and result stability by providing additional CPUs. Altogether, the method is very practical from all implementation, maintenance and operational points of view.

Lastly, the implemented instance generator together with the generated instances is a part of the thesis attachments for further use. The core idea to build the synthetic instances on real geographical data may be useful for more realistic benchmarking in general. Moreover, the implementation demonstrates that extracting valuable geographical information from publicly available data is trivial.

8.2 Future works

First direction of research would be elaborating more on the limits of the proposed method. Despite it clearly performs well on the target application, the typical instances are quite specific in their characteristics. Especially the distribution of requests among different logistical contexts and the strictness of time constraints could play major role. Also, the experiments revealed that the method obtained higher savings on larger instances. Exploring the relationship between instance sizes and transfer benefits could provide valuable point of view on the weaknesses and strengths of the method.

Second research direction is related to the possible extension of the instance space search based on genetic algorithms. The proposed approach uses clusters of split schemes as transfer features which are either applied or not. Such clusters may be directly used as binary genes and standard cross-over and mutation operators may be applied. In order to provide more diverse pool of genes, clusterings based on different dimension weightings could be used.

Lastly, a fusion of traditional methods with the proposed approach may be considered. Specifically, applying iterative heuristics similar to Godart et. al. [9] or Fu and Chow [10] on solutions obtained after the full execution of the PDPT solver could lead to further improvements. Since the solutions obtained from the PDPT solver already contain

very good setups of transfers, the iterative heuristic would not need to discover a vast majority of the transfer-related improvements. Its role would then be to solely identify straightforward savings missed by the relatively coarse PDPT solver search.

Bibliography

1. SASSMANN, Vojtěch. *Vehicle Routing with Metaheuristics*. 2020. Master thesis. Faculty of Informatics, Masaryk University, Brno.
2. PETERSEN, Hanne L.; ROPKE, Stefan. The Pickup and Delivery Problem with Cross-Docking Opportunity. In: *Computational Logistics*. 2011, pp. 101–113.
3. DANTZIG, G. B.; RAMSER, J. H. The Truck Dispatching Problem. *Management Science*. 1959, vol. 6, no. 1, pp. 80–91.
4. VAN NIEUWENHUYSE, Inneke; BRAEKERS, Kris; RAMAEKERS, Katrien. The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*. 2016, vol. 99, pp. 300–313.
5. OJEDA RIOS, Brenner Humberto; AMORIM, Pedro; XAVIER, Eduardo C.; MIYAZAWA, Flávio K.; CURCIO, Eduardo; SANTOS, Maria João. Recent dynamic vehicle routing problems: A survey. *Computers & Industrial Engineering*. 2021, vol. 160.
6. CORTÉS, Cristián E.; MATAMALA, Martín; CONTARDO, Claudio. The pickup and delivery problem with transfers: Formulation and a branch-and-cut solution method. *European Journal of Operational Research*. 2010, vol. 200, no. 3, pp. 711–724.
7. RAIS, A.; ALVELOS, F.; CARVALHO, M. S. New mixed integer-programming model for the pickup-and-delivery problem with transshipment. *European Journal of Operational Research*. 2014, vol. 235, pp. 530–539.
8. MITROVIC-MINIC, Snezana; LAPORTE, Gilbert. The pickup and delivery problem with time windows and transshipment. *Information Systems and Operational Research*. 2006, vol. 44, no. 3, pp. 217–227.
9. MANIER, Herve; GODART, Alexis; BLOCH, Christelle; MANIER, Marie-Ange. A greedy based algorithm for a bi-objective Pickup and Delivery Problem with Transfers. In: *Conference on Systems, Man and Cybernetics (SMC)*. 2019, pp. 3229–3234.

10. FU, Zhexi; CHOW, Joseph Y. J. *The pickup and delivery problem with synchronized en-route transfers for microtransit planning*. 2021. Available from arXiv: 2107.08218 [math.OC].

11. DANLOUP, N.; ALLAOUI, H.; GONCALVES, G. A comparison of two meta-heuristics for the pickup and delivery problem with transshipment. *Computers and Operations Research*. 2018, vol. 100, pp. 155–171.

12. SAMPAIO, Afonso; SAVELSBERGH, Martin; VEELENTURF, Lucas P.; VAN WOENSEL, Tom. Delivery systems with crowd-sourced drivers: A pickup and delivery problem with transfers. *Networks*. 2021, vol. 76, no. 2, pp. 232–255.

13. QU, Yuan; BARD, Jonathan F. A GRASP with adaptive large neighborhood search for pickup and delivery problems with transshipment. *Computers and Operations Research*. 2012, vol. 39, pp. 2439–2456.

14. MASSON, Renaud; LEHUÉDÉ, Fabien; PÉTON, Olivier. An Adaptive Large Neighborhood Search for the Pickup and Delivery Problem with Transfers. *Transportation Science*. 2013, vol. 47, no. 3, pp. 344–355.

15. PENG, Zhihao; AL CHAMI, Zaher; MANIER, Hervé; MANIER, Marie-Ange. A hybrid particle swarm optimization for the selective pickup and delivery problem with transfers. *Engineering Applications of Artificial Intelligence*. 2019, vol. 85, pp. 99–111.

16. NEMHAUSER, George; WOLSEY, Laurence. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.

17. LI; LIM. [N.d.]. https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/, last accessed on 22/05/04.

18. AHUJA, Ravindra K.; ERGUN, Özlem; ORLIN, James B.; PUNNEN, Abraham P. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*. 2002, vol. 123, no. 1, pp. 75–102.

19. ROPKE, Stefan; PISINGER, David. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation science*. 2006, vol. 40, no. 4, pp. 455–472.

20. SANTINI, Alberto; ROPKE, Stefan; HVATTUM, Lars Magnus. A comparison of acceptance criteria for the adaptive large neighbourhood search metaheuristic. *Journal of Heuristics*. 2018, vol. 24, pp. 783–815.

21. TALBI, El-Ghazali. *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009.

22. *Open Data Commons Open Database License (ODbL)*. [N.d.]. `https://opendatacommons.org/licenses/odbl/`, last accessed on 22/05/04.

23. BREUNIG, Markus M.; KRIEGEL, Hans-Peter; NG, Raymond T.; SANDER, Jörg. LOF: Identifying Density-Based Local Outliers. In: *International Conference on Management of Data*. 2000.

24. DREZNER, Zvi; HAMACHER, Horst W. (eds.). *Facility location. Applications and theory*. Berlin: Springer, 2002.

25. GAMRATH, Gerald; ANDERSON, Daniel; BESTUZHEVA, Ksenia; CHEN, Wei-Kun; EIFLER, Leon; GASSE, Maxime; GEMANDER, Patrick; GLEIXNER, Ambros; GOTTWALD, Leona; HALBIG, Katrin; HENDEL, Gregor; HOJNY, Christopher; KOCH, Thorsten; LE BODIC, Pierre; MAHER, Stephen J.; MATTER, Frederic; MILTENBERGER, Matthias; MÜHMER, Erik; MÜLLER, Benjamin; PFETSCH, Marc E.; SCHLÖSSER, Franziska; SERRANO, Felipe; SHINANO, Yuji; TAWFIK, Christine; VIGERSKE, Stefan; WEGSCHEIDER, Fabian; WENINGER, Dieter; WITZIG, Jakob. *The SCIP Optimization Suite 7.0*. 2020-03. Technical Report. Optimization Online. `http://www.optimization-online.org/DB_HTML/2020/03/7705.html`, last accessed on 22/05/04.

26. NOGUEIRA, Fernando. *Bayesian Optimization: Open source constrained global optimization tool for Python*. 2014–. `https://github.com/fmfn/BayesianOptimization`, last accessed on 22/05/04.

A Appendix

Files submitted into the master thesis archive in the Information System of Masaryk University:

- `thesis.pdf`: Text of the thesis in PDF format
- `osm_generator.zip`: Source codes of the OpenStreetMaps instance generator
- `osm_instances.zip`: Synthetic instances used for experiments and parameter tuning
- `experiments_tabular_results.zip`: Full results of the experiments from Chapter 7
- `illustrations.zip`: Important illustrations from the text and additional plots

B Appendix

This appendix contains a detailed overview of the results presented in Chapter 7 in tabular form. Additionally, an overview of the automated parameter tuning is reported here. The tuning summaries list attempted dimension weight configurations with their respective acquired costs in chronological order of appearance.

- Table B.1: major numerical characteristics and cost profits
- Table B.2: distance profits
- Table B.3: comparison of number of utilized vehicles
- Table B.4: tuning results (20 clusters)
- Table B.5: tuning results (30 clusters)
- Table B.6: tuning results (40 clusters)
- Table B.7: tuning results (50 clusters)

Table B.1: Results of experiments from Chapter 7 (costs).

Best and average costs over the 10 runs are reported for both baseline and PDPT solutions. Best-to-best and average-to-average profits are presented. Percents of transferred requests in best runs and average runs are provided. Finally, the absolute difference between the best and worst solution over the 10 runs is reported.

Instance	*Baseline best*	*Transfers best*	*Best-to-best profits %*	*Baseline avg.*	*Transfers avg.*	*Avg.-to-avg. profits %*	*Split requests best %*	*Split requests avg. %*	*Worst-best diff. baseline*	*Worst-best diff. transfers*
bulgaria_1000_1	1,722,632.2	1,517,092.3	11.9	1,739,676.3	1,545,115.4	11.2	26.6	21.6	34,844.6	43,379.1
bulgaria_1000_2	1,891,845.0	1,686,236.9	10.9	1,909,013.1	1,703,741.8	10.8	35.7	29.9	27,846.0	27,723.1
bulgaria_1000_3	1,932,437.6	1,658,031.3	14.2	1,947,614.2	1,673,729.1	14.1	28.4	27.8	41,106.0	26,215.7
bulgaria_500_1	857,977.1	785,093.4	8.5	862,954.1	791,230.2	8.3	16.8	18.6	8,820.8	16,119.7
bulgaria_500_2	934,884.1	865,261.1	7.4	945,366.1	873,214.8	7.6	36.8	33.4	20,417.8	13,252.2
bulgaria_500_3	1,014,903.0	901,379.4	11.2	1,027,199.6	914,158.1	11.0	18.6	28.6	19,464.2	19,166.2
czechia_1000_1	1,927,777.2	1,663,020.3	13.7	1,937,206.5	1,673,853.1	13.6	9.9	9.6	19,826.5	24,727.5
czechia_1000_2	1,758,858.5	1,482,080.2	15.7	1,767,642.3	1,502,647.6	15.0	25.4	23.9	17,163.1	29,534.1
czechia_1000_3	1,935,350.0	1,605,761.9	17.0	1,946,155.8	1,618,709.6	16.8	27.0	28.0	23,404.7	34,211.9
czechia_500_1	942,164.0	832,740.0	11.6	953,820.2	840,690.1	11.9	8.6	10.8	17,809.0	15,290.3
czechia_500_2	909,585.6	757,431.5	16.7	916,941.7	768,057.7	16.2	31.4	29.9	20,614.0	16,758.1
czechia_500_3	966,542.3	827,813.3	14.4	976,442.6	838,694.1	14.1	33.0	33.5	15,466.9	17,365.5
wereldo_day_1	1,274,091.9	995,539.1	21.9	1,306,293.4	1,013,648.6	22.4	17.6	19.3	65,105.2	33,227.8
wereldo_day_2	1,146,082.9	922,947.0	19.5	1,193,459.0	932,435.8	21.9	32.4	21.3	96,642.4	18,070.0
wereldo_day_3	1,096,989.1	859,586.5	21.6	1,132,561.6	879,019.1	22.4	16.3	13.2	63,705.5	37,583.6
wereldo_day_4	1,226,417.7	927,051.4	24.4	1,284,818.9	955,694.8	25.6	9.4	7.6	103,017.5	40,357.8
wereldo_day_5	1,005,203.5	860,797.0	14.4	1,048,671.9	876,585.0	16.4	20.3	13.3	71,469.6	23,911.3
Min	857,977.1	757,431.5	7.4	862,954.1	768,057.7	7.6	8.6	7.6	8,820.8	13,252.2
Avg	1,300,095.5	1,105,849.7	14.6	1,319,932.9	1,120,515.7	14.8	22.4	21.0	37,530.3	25,008.1
Max	1,935,350.0	1,686,236.9	24.4	1,947,614.2	1,703,741.8	25.6	36.8	33.5	103,017.5	43,379.1

Table B.2: Results of experiments from Chapter 7 (distances).

Best and average total distances over the 10 runs are reported for both baseline and PDPT solutions. Best-to-best and average-to-average profits are presented (best and average based on costs). The absolute difference of the worst and best achieved distance is reported.

Instance	*Baseline best*	*Transfers best*	*Best-to-best profits %*	*Baseline avg.*	*Transfers avg.*	*Avg.-to-avg. profits %*	*Worst-best diff. baseline*	*Worst-best diff. transfers*
bulgaria_1000_1	93,198.6	77,542.5	16.8	93,049.8	79,099.3	15.0	3,637.8	3,435.5
bulgaria_1000_2	103,725.1	87,301.6	15.8	104,671.6	88,673.4	15.3	2,246.4	2,535.0
bulgaria_1000_3	106,866.9	85,623.7	19.9	108,210.2	86,403.2	20.2	2,797.2	1,906.0
bulgaria_500_1	46,912.8	41,706.9	11.1	48,087.1	42,388.3	11.9	2,198.9	1,200.7
bulgaria_500_2	51,675.9	45,518.1	11.9	52,007.9	46,114.8	11.3	2,028.6	1,847.8
bulgaria_500_3	55,922.6	47,194.3	15.6	56,432.2	48,116.2	14.7	1,615.1	1,767.6
czechia_1000_1	109,020.7	89,152.3	18.2	108,626.8	89,373.1	17.7	2,247.9	2,578.9
czechia_1000_2	98,494.4	77,988.3	20.8	98,181.7	78,991.1	19.5	2,094.0	3,034.7
czechia_1000_3	108,039.7	84,381.4	21.9	108,449.4	85,010.2	21.6	2,496.2	2,306.2
czechia_500_1	55,552.3	45,690.7	17.8	55,376.8	46,495.6	16.0	1,906.8	2,384.5
czechia_500_2	51,185.3	40,291.6	21.3	52,078.1	40,771.1	21.7	1,741.9	994.0
czechia_500_3	55,018.9	44,994.6	18.2	55,789.7	45,541.3	18.4	1,965.8	1,465.4
wereldo_day_1	103,326.3	88,643.7	14.2	105,088.3	91,493.8	12.9	4,492.3	7,009.1
wereldo_day_2	99,156.7	86,535.1	12.7	100,423.7	86,801.3	13.6	7,600.7	3,561.7
wereldo_day_3	99,274.8	81,787.8	17.6	99,782.1	82,322.2	17.5	4,273.3	4,931.0
wereldo_day_4	103,416.5	86,268.4	16.6	104,079.6	88,558.8	14.9	3,976.4	4,685.0
wereldo_day_5	94,132.8	81,195.4	13.7	95,250.6	82,542.7	13.3	3,617.2	3,202.1
Min	46,912.8	40,291.6	11.1	48,087.1	40,771.1	11.3	1,615.1	994.0
Avg	82,324.1	68,450.4	16.4	82,981.8	69,414.9	15.9	2,919.5	2,768.8
Max	109,020.7	89,152.3	21.9	108,626.8	91,493.8	21.7	7,600.7	7,009.1

Table B.3: Results of experiments from Chapter 7 (vehicles).
Best and average number of utilized vehicles over the 10 runs are reported for both baseline and PDPT solutions. Best-to-best and average-to-average comparisons are presented as the difference of baseline and PDPT solutions (best and average based on costs). The absolute difference of the highest and lowest number of vehicles in instance is reported.

Instance	*Baseline best*	*Transfers best*	*Best-to-best diff.*	*Baseline avg.*	*Transfers avg.*	*Avg.-to-avg. diff.*	*Worst-best diff. baseline*	*Worst-best diff. transfers*
bulgaria_1000_1	188	192	-4	187	190	-3	8	6
bulgaria_1000_2	198	206	-8	198	205	-8	9	23
bulgaria_1000_3	200	195	5	202	194	8	12	7
bulgaria_500_1	96	100	-4	96	103	-8	6	12
bulgaria_500_2	98	113	-15	100	112	-11	6	10
bulgaria_500_3	104	100	4	104	108	-4	6	21
czechia_1000_1	209	193	16	211	198	13	9	10
czechia_1000_2	193	193	0	193	191	2	4	21
czechia_1000_3	205	203	2	205	204	1	9	11
czechia_500_1	107	100	7	107	107	0	5	13
czechia_500_2	101	108	-7	101	108	-7	6	9
czechia_500_3	105	111	-6	106	116	-10	4	14
wereldo_day_1	241	232	9	248	230	18	14	11
wereldo_day_2	239	231	8	244	225	19	13	14
wereldo_day_3	240	215	25	238	215	22	11	7
wereldo_day_4	252	224	28	252	224	28	13	11
wereldo_day_5	232	209	23	228	209	18	9	9
Min	96	100	-15	96	103	-11	4	6
Avg	172	168	4	173	169	4	8	12
Max	252	232	28	252	230	28	14	23

Table B.4: Tuning with cluster limit 20.

Cost	*Detour abs.*	*Detour rel.*	*PD distance*	*Slack*	*Uniformity*	*Volume*	*Weight*
818,330.0	1.500	1.400	0.400	0.400	1.400	1.700	0.600
830,513.2	0.500	2.000	0.750	1.400	0.500	1.500	0.400
812,182.9	0.951	1.497	0.200	0.744	0.464	0.366	0.535
812,656.3	0.822	0.914	1.170	0.955	1.433	0.568	1.781
813,597.0	0.603	1.940	1.162	1.547	1.698	0.754	1.198
798,586.8	**0.913**	**1.235**	**0.254**	**0.826**	**0.342**	**0.239**	**0.624**
823,647.5	1.937	0.319	0.294	1.301	1.242	1.749	0.925
828,328.5	0.898	0.978	1.216	0.913	1.172	0.734	1.713
813,903.6	0.995	1.116	1.251	1.711	0.205	1.672	0.400
821,481.0	1.490	1.373	0.388	0.357	1.379	1.697	0.563
821,279.7	1.865	1.709	0.214	1.810	0.210	1.116	1.000
825,413.8	0.840	1.351	0.300	0.843	0.305	0.634	0.749
818,119.7	0.945	1.294	0.224	0.794	0.394	0.200	0.561
809,114.1	1.038	0.624	0.546	0.374	1.870	0.972	1.273
836,894.1	0.570	1.880	1.168	1.590	1.741	0.626	1.094
818,984.2	1.690	1.265	1.305	1.189	1.276	1.349	0.822
830,334.5	0.447	1.988	0.741	1.418	0.428	1.499	0.342
830,694.5	0.952	1.299	1.209	1.755	0.415	1.686	0.493
812,806.4	0.234	1.753	0.969	0.552	1.426	0.945	0.884
845,236.9	0.627	1.057	0.708	1.906	1.823	1.759	1.666
830,348.3	1.017	1.768	1.005	0.353	0.323	1.511	1.476
811,180.7	0.747	1.554	0.253	0.721	0.366	0.312	0.538
831,169.2	0.793	0.889	1.151	0.970	1.536	0.502	1.807
840,719.9	1.936	0.366	1.404	1.680	1.404	1.173	1.762
819,985.8	1.753	1.282	1.388	0.934	1.435	1.376	0.752
837,917.3	0.250	1.019	1.440	0.544	1.409	0.814	1.677
835,160.2	0.711	0.994	0.631	1.527	1.085	1.525	0.952
829,633.6	1.388	1.558	1.035	0.649	1.561	0.702	1.117
821,751.0	0.678	1.546	0.222	0.904	0.274	0.568	0.528
850,907.4	0.446	0.336	0.680	1.890	1.195	1.114	1.437
829,026.7	0.748	1.048	0.323	0.713	0.284	0.287	0.441
815,190.9	0.566	1.609	0.417	0.590	0.414	0.258	0.446
821,277.7	1.570	1.392	0.517	1.678	1.351	0.468	1.905
804,541.3	1.337	1.693	1.565	1.726	0.872	1.147	0.503
824,608.2	1.012	1.014	1.273	1.680	0.200	1.663	0.347
827,589.8	1.714	1.258	1.366	1.081	1.375	1.373	0.809
823,714.8	0.242	1.619	0.918	0.657	1.515	1.166	0.832
822,810.8	0.564	1.665	0.492	1.128	0.478	1.143	0.672
824,278.2	0.255	0.970	1.770	1.397	1.144	0.416	0.367

Table B.5: Tuning with cluster limit 30.

Cost	*Detour abs.*	*Detour rel.*	*PD distance*	*Slack*	*Uniformity*	*Volume*	*Weight*
815,715.0	1.500	1.400	0.400	0.400	1.400	1.700	0.600
816,656.3	0.500	2.000	0.750	1.400	0.500	1.500	0.400
815,860.0	0.951	1.497	0.200	0.744	0.464	0.366	0.535
825,869.6	0.822	0.914	1.170	0.955	1.433	0.568	1.781
825,161.6	1.150	1.560	0.326	0.669	0.861	1.164	0.476
831,706.8	1.577	0.397	1.140	1.583	1.437	0.439	0.563
822,137.7	1.937	0.319	0.294	1.301	1.242	1.749	0.925
816,909.5	0.782	1.483	1.064	0.248	1.429	1.603	1.953
824,250.8	0.995	1.116	1.251	1.711	0.205	1.672	0.400
823,294.0	1.490	1.373	0.388	0.357	1.379	1.697	0.563
814,957.3	1.865	1.709	0.214	1.810	0.210	1.116	1.000
811,831.1	1.967	0.557	1.873	0.948	0.928	0.229	1.605
811,086.1	1.332	1.692	1.321	0.794	0.729	0.364	0.225
832,396.5	1.038	0.624	0.546	0.374	1.870	0.972	1.273
798,600.1	**1.687**	**1.599**	**1.491**	**1.552**	**1.746**	**0.212**	**1.473**
810,195.0	1.690	1.265	1.305	1.189	1.276	1.349	0.822
816,919.9	0.447	1.988	0.741	1.418	0.428	1.499	0.342
819,338.1	1.280	0.665	1.731	1.158	0.807	1.768	1.293
813,161.5	0.234	1.753	0.969	0.552	1.426	0.945	0.884
837,230.6	0.627	1.057	0.708	1.906	1.823	1.759	1.666
816,435.0	1.017	1.768	1.005	0.353	0.323	1.511	1.476
823,453.9	0.672	0.955	0.793	0.469	1.898	0.351	1.153
806,203.7	1.220	1.245	1.112	1.326	0.224	0.669	1.317
841,812.6	1.936	0.366	1.404	1.680	1.404	1.173	1.762
811,472.6	1.242	1.232	1.108	1.267	0.326	0.678	1.367
837,113.0	0.250	1.019	1.440	0.544	1.409	0.814	1.677
840,458.6	0.711	0.994	0.631	1.527	1.085	1.525	0.952
807,097.6	1.388	1.558	1.035	0.649	1.561	0.702	1.117
807,359.8	1.450	1.506	1.088	0.756	1.506	0.822	1.060
806,608.2	1.651	1.592	1.429	1.427	1.713	0.297	1.413
815,878.7	1.483	1.579	1.150	0.870	1.589	0.617	1.169
814,210.6	1.429	1.639	0.934	0.598	1.561	0.928	1.025
812,554.4	1.570	1.392	0.517	1.678	1.351	0.468	1.905
809,448.0	1.337	1.693	1.565	1.726	0.872	1.147	0.503
835,341.4	0.274	0.341	1.662	1.114	1.528	0.485	0.714
827,346.0	1.104	1.336	1.978	1.506	1.336	1.736	1.929
808,095.0	1.769	1.529	0.566	0.920	0.579	0.661	0.950
813,944.2	0.564	1.665	0.492	1.128	0.478	1.143	0.672
806,004.3	1.548	1.522	0.456	0.596	1.496	1.717	0.768

Table B.6: Tuning with cluster limit 40.

Cost	*Detour abs.*	*Detour rel.*	*PD distance*	*Slack*	*Uniformity*	*Volume*	*Weight*
810,737.6	1.500	1.400	0.400	0.400	1.400	1.700	0.600
824,431.6	0.500	2.000	0.750	1.400	0.500	1.500	0.400
824,982.9	0.951	1.497	0.200	0.744	0.464	0.366	0.535
824,615.8	0.822	0.914	1.170	0.955	1.433	0.568	1.781
811,882.4	1.516	1.483	0.427	0.524	1.506	1.604	0.554
808,860.0	2.000	0.693	0.200	0.200	1.820	2.000	0.298
817,876.5	2.000	1.719	0.200	0.200	2.000	2.000	1.917
812,495.4	2.000	0.551	1.564	0.200	1.232	2.000	0.200
835,489.4	0.557	0.200	0.200	0.200	2.000	2.000	0.200
822,469.3	2.000	1.046	0.343	0.200	0.832	2.000	0.200
817,129.9	1.951	1.014	0.703	0.237	1.855	1.869	0.637
812,080.6	1.705	1.140	0.200	0.472	1.794	2.000	0.200
809,008.5	2.000	0.572	0.200	0.576	1.825	1.498	0.200
813,736.2	2.000	0.373	0.200	0.906	1.796	2.000	0.521
808,278.2	2.000	0.200	0.717	0.200	1.642	1.408	0.200
802,207.2	2.000	0.200	0.200	0.200	1.945	0.760	0.200
813,471.8	2.000	0.200	0.783	0.200	2.000	0.291	0.200
810,522.9	1.844	0.280	0.242	0.780	1.844	0.429	0.302
801,871.0	2.000	0.200	2.000	0.200	0.693	0.990	0.200
820,094.1	2.000	0.200	2.000	0.776	0.200	1.124	0.200
792,243.1	**2.000**	**0.200**	**2.000**	**0.200**	**1.281**	**0.911**	**0.200**
805,212.3	2.000	0.200	2.000	0.200	1.246	0.358	0.200
801,988.9	2.000	0.200	2.000	0.200	1.738	1.208	0.200
805,186.5	1.510	0.200	2.000	0.200	1.227	1.022	0.200
806,858.8	1.918	0.621	1.803	0.235	1.661	0.980	0.394
815,941.3	1.823	0.202	1.623	0.225	1.340	1.163	0.615
812,488.7	1.990	1.833	1.967	0.285	1.449	0.452	0.212
813,869.7	1.890	0.243	1.853	0.540	1.958	0.448	0.252
816,737.5	2.000	0.200	2.000	2.000	2.000	2.000	0.200
811,872.0	1.775	1.152	0.244	0.260	1.661	0.595	0.242
822,183.3	2.000	2.000	2.000	2.000	0.200	2.000	2.000
812,928.3	1.961	0.388	1.805	0.307	1.069	0.784	0.318
814,558.1	1.570	1.392	0.517	1.678	1.351	0.468	1.905
820,975.9	1.337	1.693	1.565	1.726	0.872	1.147	0.503
811,910.9	1.436	1.506	0.320	0.524	1.338	1.381	0.681
810,516.4	1.660	1.336	0.530	0.423	1.345	1.772	0.430
814,775.3	1.484	1.700	0.402	0.463	1.417	1.292	0.530
809,045.2	0.564	1.665	0.492	1.128	0.478	1.143	0.672
797,928.8	2.000	0.200	2.000	0.200	1.525	0.825	0.200

Table B.7: Tuning with cluster limit 50.

Cost	*Detour abs.*	*Detour rel.*	*PD distance*	*Slack*	*Uniformity*	*Volume*	*Weight*
810,608.6	1.500	1.400	0.400	0.400	1.400	1.700	0.600
816,980.8	0.500	2.000	0.750	1.400	0.500	1.500	0.400
820,746.9	0.951	1.497	0.200	0.744	0.464	0.366	0.535
812,193.0	0.822	0.914	1.170	0.955	1.433	0.568	1.781
807,749.6	1.544	1.229	0.510	0.274	1.471	1.873	0.759
815,894.2	1.699	0.824	0.862	0.200	1.970	2.000	1.289
817,876.4	1.937	0.319	0.294	1.301	1.242	1.749	0.925
807,749.9	1.708	1.476	0.232	0.345	1.545	1.664	0.724
823,344.9	0.995	1.116	1.251	1.711	0.205	1.672	0.400
806,692.8	1.937	1.533	0.248	0.200	1.285	2.000	1.118
806,420.0	2.000	2.000	0.305	0.200	2.000	2.000	0.829
817,712.9	1.174	2.000	0.200	0.200	1.822	2.000	1.385
812,078.5	2.000	1.378	0.200	0.200	1.900	2.000	0.276
812,525.4	2.000	1.837	0.813	0.200	1.506	1.823	0.840
813,664.5	1.808	1.123	0.324	0.272	0.966	1.579	1.334
802,087.6	1.690	1.265	1.305	1.189	1.276	1.349	0.822
829,658.9	0.447	1.988	0.741	1.418	0.428	1.499	0.342
816,703.3	1.460	1.057	0.416	0.220	1.492	1.828	0.584
812,620.2	0.234	1.753	0.969	0.552	1.426	0.945	0.884
822,982.5	0.807	0.919	1.311	1.021	1.551	0.627	1.758
826,470.5	1.017	1.768	1.005	0.353	0.323	1.511	1.476
802,619.6	1.535	1.430	0.558	0.449	1.280	1.991	0.758
809,832.8	1.574	1.316	0.517	0.471	1.264	1.696	0.659
818,963.7	1.936	0.366	1.404	1.680	1.404	1.173	1.762
817,514.1	1.975	1.808	0.804	0.272	1.467	1.702	0.795
815,853.5	0.250	1.019	1.440	0.544	1.409	0.814	1.677
819,827.5	0.711	0.994	0.631	1.527	1.085	1.525	0.952
805,688.0	1.388	1.558	1.035	0.649	1.561	0.702	1.117
810,908.4	1.568	1.616	0.377	0.456	1.203	1.707	0.522
812,619.3	1.564	1.184	0.483	0.383	1.037	1.830	0.675
805,781.3	1.783	1.191	1.180	1.136	1.134	1.414	0.684
814,884.0	1.473	1.531	0.421	0.439	1.190	1.872	0.844
822,929.5	1.570	1.392	0.517	1.678	1.351	0.468	1.905
819,325.5	1.337	1.693	1.565	1.726	0.872	1.147	0.503
807,182.2	1.493	1.515	0.646	0.392	1.270	1.923	0.679
813,505.2	1.729	1.233	1.252	1.166	1.216	1.377	0.763
801,551.7	**1.536**	**1.560**	**1.052**	**0.920**	**1.579**	**0.802**	**1.062**
824,693.2	0.564	1.665	0.492	1.128	0.478	1.143	0.672
827,952.6	0.255	0.970	1.770	1.397	1.144	0.416	0.367