# A BRANCH-CUT-AND-PRICE ALGORITHM FOR THE TRAVELLING SALESMAN WITH DRONE

MARK BIEREMA
454657


Erasmus School of Economics
Erasmus University Rotterdam

November 5, 2018

"Bit by bit, byte by byte."

master's thesis econometrics

supervisor:
dr. P.C. Bouman

co-assessor:
dr. T.A.B Dollevoet

# A BRANCH-CUT-AND-PRICE ALGORITHM FOR THE TRAVELLING SALESMAN WITH DRONE

MARK BIEREMA

**Abstract:** The collaboration between a truck and drone with the joint objective to satisfy customer demand under minimisation of the execution time is studied. Leading exact methods approach the problem in the fashion of a set covering problem but have to cope with a large solution space. A framework is introduced that refines the solution space, improving the performance of existing methods. Furthermore, a branch-cut-and-price algorithm is proposed and a branching rule is introduced that is specific for the problem. Despite, the pricing procedure did not yield any computational advantage over the branch-and-cut benchmark. The computational study showed promising results as problems with 34 customers were solved to optimality, whereas the best-known exact methods could handle up to 15 customers under the same conditions. Furthermore, a local search routine within the methods was considered and achieved reductions in the costs of up to 6% for a problem with 70 customers compared to a respectable heuristic (Agatz et al., 2016).

# 1 INTRODUCTION

Drone delivery started as a futuristic concept and is now receiving rising interest in the industry of logistics. Especially in the parcel delivery industry companies are trying to innovate through the use of drones. Ideas range from standalone delivery using drones based from a central warehouse, or even based from a large airship, to assisting delivery vehicles with drones. Nowadays, the development is also expanding to the logistics of large cargo (Shivdas, 2018). As the technology becomes available, research is needed to make the best operational decisions.

The setting in this paper is that of a collaborative delivery network involving a single truck and drone. As usual, the problem involves a delivery truck that has to visit locations to deliver parcels. However, in addition, the truck is accompanied by a drone that is installed on top of the truck. This drone can ascent from the truck to deliver packages to destinations and returns to the truck after doing so. The drone and truck work together and in parallel, which possibly reduces costs and improves service. In this paper, an exact method is developed that optimises the problem.

In the literature the problem is often referred to as the TSP-D, the travelling salesman problem with drone. Although some papers have introduced exact algorithms, these can only solve small problems. Besides, branch-cut-and-price has become an established method in the literature, yet remains to be unconsidered for the TSP-D.

This paper aims to set a new standard for the current exact approaches by developing a branch-cut-and-price algorithm. For this purpose, TSP-D related concepts from existing literature will be adapted and refined. Computational experiments are conducted on a computer cluster to assess the performance of the method and its performance relative to alternatives. Finally, it is studied whether the method acts useful for large instances.

The remainder of this paper is structured as follows. Section 2 discusses literature related to the TSP-D problem and applications of methods relevant to this paper. In Section 3 the problem is formally introduced and provided as a mathematical problem. Besides, existing concepts are discussed and refinements are made to the framework of these concepts. Next, Section 4 discusses the components of the algorithm. These include two methods for the pricing procedure in the algorithm, and a discussion on dynamic subtour elimination. In Section 5 the experimental framework is introduced and the performance of the algorithm is assessed. Finally, concluding remarks are provided in Section 6.

# 2 LITERATURE REVIEW

The literature exhibits different names for problems involving the collaborative use of trucks and drones, that are essentially the same. Nevertheless, each of these papers provides a good overview of related research. This section positions the research of this

paper in the literature and provides an overview of research conducted on the TSP-D and related problems. Likewise, the branch-cut-and-price method will be addressed.

The TSP-D problem, as defined in this paper, first appeared in Agatz et al. (2016) and was also studied in the follow-up (Bouman et al., 2017). Notable is that the papers handle the problem using the convenient concept of operations, describing a collection of several truck and drone actions. In the initial paper the authors approached to solve the TSP-D by developing heuristics. An exact method using integer programming was also studied, but shortly after, the leading dynamic programming approach was developed. This paper adopts and refines the framework of operations and intends to solve instances beyond what current methods can handle.

A problem almost identical to the TSP-D is the flying sidekick travelling salesman problem (FSTSP), first studied in Murray and Chu (2015). The problem differs as it does not allow for a rendezvous at the same location as where the drone took off priorly. Murray and Chu (2015) together with Ha et al. (2015) provided integer programming formulations and developed heuristics to solve the problem. Furthermore, Ferrandez et al. (2016) studied a genetic algorithm and Ponza (2016) used simulated annealing to obtain solutions.

Numerous papers study the problem in which solely drones may complete deliveries. In Mathew et al. (2015), the problem is called the heterogeneous delivery problem. Besides, Othman et al. (2017) study the last-stretch delivery problem under the added assumption that the truck route is predetermined. In Carlsson and Song (2017), the horsefly problem is studied, which assumes no restricted launch and return locations for the drone. Garone et al. (2010) study carrier-vehicle systems for military applications. Although the problem emerged from a different environment, many similarities with the foregoing problems are present.

Also, Ha et al. (2018) consider minimising the operational costs together with costs incurred for waiting times instead of the usual total delivery time. At last, the paper of Dorling et al. (2017) is mentioned, which studies an energy consumption constraint that relates flight times and payload weight. This may become interesting for more realistic applications of the TSP-D.

A natural extension is to consider a single truck with multiple drones. This has been studied in Ferrandez et al. (2016) and was also part of Carlsson and Song (2017). Furthermore, the VRPD, a setting with multiple vehicles and multiple drones dedicated to specific vehicles, is studied by Wang et al. (2016) together with Poikonen et al. (2017). The papers contribute by researching the added value of including drones in contrast to operating trucks exclusively. Next, Daknama and Kraus (2017) include the possibility for drones to switch between trucks, and attempts to solve the problem with a heuristic.

Applications of branch-cut-and-price have started to appear rapidly in the last decade. The method combines branch-and-bound, cutting planes and column generation to solve integer programs. Previously, the combination of branch-and-bound and cut-

ting planes led to advancements in solving the travelling salesman problem (Padberg and Rinaldi, 1991). Some time following, the combination of branch-and-bound and column generation was studied (Barnhart et al., 1998) to solve integer programs with many decision variables. In turn, the combination of all three methods shows to be effective for problems such as the capacitated vehicle routing problem (Fukasawa et al., 2006). Despite its wide applicability, a drawback of the method is that it is often considered too sophisticated as it requires tailoring to the problem at hand.

## 3 PROBLEM FORMULATION

In this section, a formal problem formulation is provided. Besides, the concept of operations is introduced together with a theoretical framework. In addition, an integer program is provided that solves the problem. Thereby, the concept of operations and the integer program stem from Agatz et al. (2016) and are further built upon.

### 3.1 *Formal Problem Description*

The TSP-D is a routing problem that can be represented on a graph $G = (V, E)$ with a collection of nodes $V$ connected by a complete set $E$ of edges. The members of $V$, say $v_0$ and $v_1, ..., v_n$, represent the depot and $n$ customer locations, respectively. Throughout this paper, the customer locations are simply referred to as locations, whereas the depot may be involved when speaking of nodes. Furthermore, the truck and drone are assumed to be heterogeneous and are, respectively, subject to costs $c_{uw}$ and $c^d_{uw}$ incurred for using edge $(u, w)$ to travel between nodes $u$ and $w$.

The problem requires to visit all customer locations by constructing a closed walk in the graph $G$, that starts and ends at $v_0$, for both the truck and the drone. From now on, such walks will be referred to as a tours. In other words, each customer location is at least either on the tour of the truck or drone. The objective is to do this with minimal costs. Like other studies, this paper will focus on minimising the total time to complete the longest tour, implying that $c_{uw}$ and $c^d_{uw}$ represent travelling times. For writing convenience throughout the paper, travel times are often referred to as costs as well. In addition, the research in this paper is limited to the assumption that the triangle inequality is respected by $c_{uw}$ and $c^d_{uw}$, respectively. In words, the fastest way to reach node $w$ from node $u$ is by utilising the edge $(u, w)$.

Two further restrictions are imposed on the solution. Firstly, a rendezvous between the truck and drone must occur at nodes in the graph $V$. As a result, a node is visited by either the truck, the drone or both. Such nodes will be referred to as truck nodes, drone nodes and combined nodes, respectively. Secondly, a drone can only deliver one parcel at a time before it has to rendezvous with the truck again.

As a side note, Agatz et al. (2016) observed that, unlike most routing minimisation problems, it could be optimal to visit nodes more than once as this allows the truck

and drone to interact, but that it remains optimal to visit a node once that appears as a truck node or drone node. As the tours start at the depot, it is reasonable to assume that the depot cannot be a truck node or drone node.

## 3.2 *Operation Framework*

Define an operation as a smaller segment in the tour consisting of the collection of one or two combined nodes, a non-negative number of truck nodes and at most one drone node. In this way, the truck and drone tour can be represented by a set of operations. Operations with no drone node will be called simple operations in this paper. Building upon this concept, the following framework is introduced that elaborates on previous research.

First some notation regarding operations is introduced, to be used throughout the rest of this paper. Conveniently, an operation $o$ can be represented by a tuple. Consider $O'$ to be the collection of all possible operations that satisfy the definition of an operation. Accordingly, for $o \in O'$, let

$$o \equiv \quad < u_o, w_o, d_o, T_o, V_o, c_o > . \tag{1}$$

The tuple describes an operation by a start node $u_o$, end node $w_o$, drone node $d_o$, sequence of truck nodes $T_o$, set of nodes covered $V_o$ and operational costs $c_o$. Here, $V_o = \{u_o, w_o, d_o\} \cup T_o$, and $c_o$ corresponds to the largest among the costs of the drone path and truck path. For convenience, denote $t_v$ for the travel time of the truck from start node $u_o$ to some node $v \in T_o \cup \{w_o\}$.

The number of possible operations is large and some may be more desirable than others. For most methods, it is desired to identify and select those operations that have the most potential to produce a good solution. In the context of exact methods, a sufficient subset is desired, being a set $O \subseteq O'$ that can still provide an optimal solution to the TSP-D. In the following discussion a framework intended for exact methods is developed that introduces properties, describing different classes of operations. Before doing so, consider the notion of equivalence.

**Definition 1** (Equivalence)**.** Let sequences of $n$ operations $\{o_{11}, ..., o_{1n}\} \in O'$ and $m$ operations $\{o_{21}, ..., o_{2m}\} \in O'$ be given. The sequences of operations are equivalent if the start nodes and end nodes of the sequences are equal and the sequences cover the same set of nodes. In other words, $u_{o_{11}} = u_{o_{21}}$, $w_{o_{1n}} = w_{o_{2m}}$ and $\bigcup_{i=1}^{n} V_{o_{1i}} = \bigcup_{i=1}^{m} V_{o_{2i}}$.

Definition 1 implies that replacing a sequence of operations by an equivalent sequence in a solution also creates a solution to the TSP-D. Equivalence will be useful in the context of two single operations or a single operation and a sequence of operations. Namely, an operation that can be represented in an alternative way that is as at least as good can be left out of consideration.

A subset $O^e \subseteq O'$ is called an equivalent subset of $O'$ if for all operations $o \in O'$ there is an operation or sequence of operations from $O^e$ that is equivalent to $o$. A special case of this is $O_c$, defining an equivalent subset of minimal costs. Here, $O_c$ satisfies that its members are equivalent or part of a sequence equivalent to some $o \in O'$ for which no other equivalent operation or sequence of operations exists with smaller costs. Nevertheless, equivalence relations may still exist among the individual members of $O_c$ as two operations can be equally good. Removing such relations defines another special set $O_s$, an equivalent subset of minimal size. The defining property of $O_s$ is that removing any of its members results in $O_s$ not being an equivalent subset any longer. As a final remark, $O_c$ and $O_s$ are not unique sets but arbitrary sets that respect the definition.

Proving that some subset $O$ is sufficient is straightforward provided that $O$ is equivalent. A number of properties will be discussed that are based on equivalence. Firstly, two properties will be introduced that concern equivalence between operations. The property of efficiency was already considered by Bouman et al. (2017).

**Definition 2** (Efficiency)**.** An operation $o$ is called efficient if the trucks nodes $T_o$ are covered by the truck along one of the shortest paths starting at $u_o$ and ending at $w_o$.

Efficiency assesses an operation based on the quality of its truck path. A sufficient subset can be constructed by taking all efficient operations from $O'$. However, equivalence relations concerning two operations may still be present even if all shortest paths are unique. The following property aims to deal with this.

**Definition 3** (Effectiveness)**.** An operation $o \in O'$ is called effective if the partition of the truck nodes and drone nodes, and the sequence of truck nodes are optimal in the sense that the largest amongst the costs of the truck path and drone path is minimised for fixed combined nodes.

The property of effectiveness accounts for the possibility that it is better to interchange the node assigned to the drone with one of the truck nodes. Note that it does not always imply efficiency as the truck path need not be the shortest if the drone path is longer.

Next, a property that acts upon equivalence between a single operation and a sequence of operations is introduced. This is useful as it enables large operations to be disregarded and represented by an equivalent sequence of smaller operations instead.

**Definition 4** (Compactness)**.** An operation $o \in O'$, with drone node $d_o$, is called compact if no equivalent sequence of operations $o_1, ...o_n \in O \setminus \{o\}$, with all but one simple operations, exists with the same drone node and the same truck path that has total costs not exceeding $c_o$.

In other words, it is not possible for the drone to take a different path without increasing the costs. This implies that compactness filters out operations that require

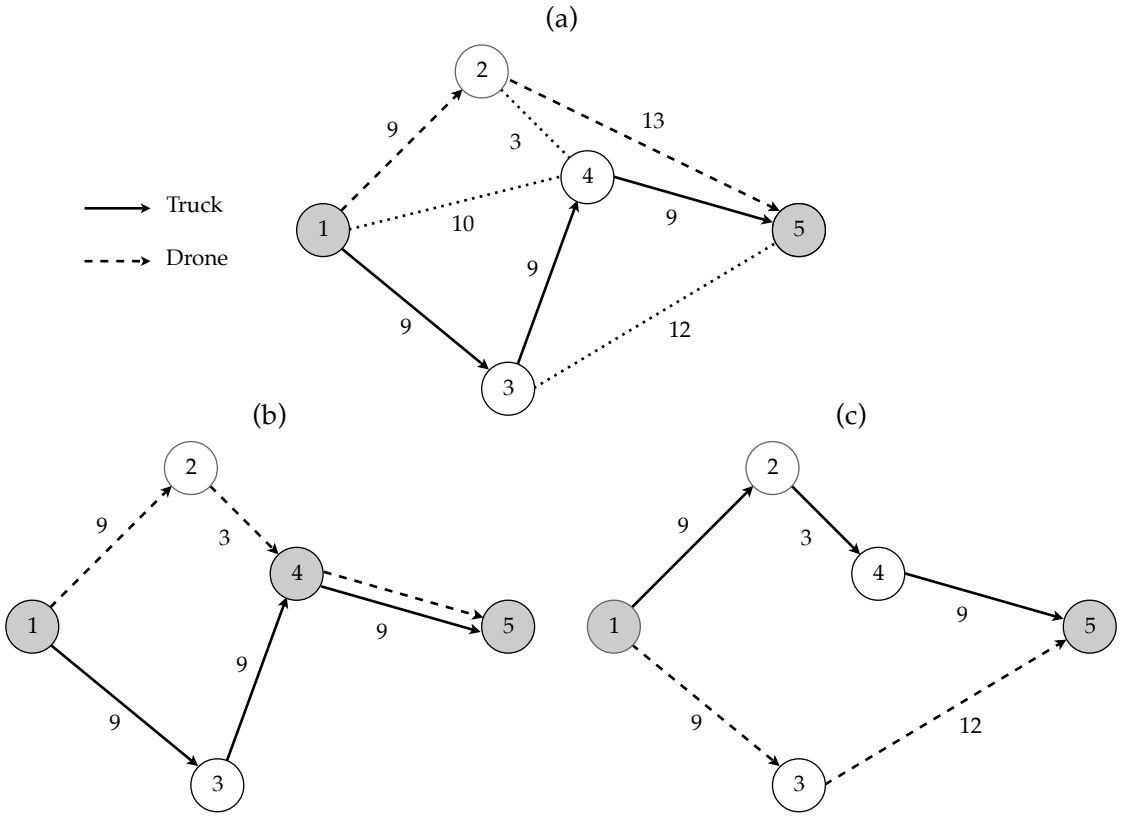


Figure 1: Illustration of compactness and effectiveness. (a) presents a graph with edges that can be traversed in both directions for the stated costs, by both the drone and the truck. On top of that, (a) depicts an efficient operation. (b) shows an equivalent alternative of (a) that is compact. (c) shows an equivalent alternative of (a) that is effective. Combined nodes are shaded.

the drone to take off sooner or postpone the rendezvous between the truck and drone without benefit. Consequently, the implicit conditions that it is not possible for the drone to depart later or return sooner will be referred to as compactness with respect to the drone departure and return, respectively.

Examples of compactness and effectiveness are shown in Figure 1. Here, the operation that (a) depicts, with costs of $27 = \max\{27, 21\}$, is efficient as the path $1 \to 3 \to 4 \to 5$, of the truck, is shorter than the alternative $1 \to 4 \to 3 \to 5$. However, the operation is not compact nor effective. The former is shown by (b), here the difference is that the drone returns to the truck already at 4 instead of 5. Therefore, by the definition of an operation, now two operations are depicted. Since the total costs of the two operations are also 27, it follows that in (a) the rendezvous is postponed, but without benefit. The latter claim is shown by (c). By interchanging the drone node with one of the truck nodes, the costs can be decreased to 21. These are the lowest possible for

any operation that starts at 1, ends at 5, and covers the nodes $\{2, 3, 4\}$, which implies that the operation in (c) is effective.

With the properties that have been discussed thus far, the following proposition can be formulated.

**Proposition 1** (Equivalent subset of minimal costs)**.** The operations in $O'$ that are effective and compact constitute an equivalent subset $O_c \subseteq O'$ of minimal costs.

*Proof.* Note that both effectiveness and compactness are properties that are based on equivalence. As a result, the combination describes an equivalent subset. To show that this set is a valid set $O_c$ of minimal costs, it is proven that every equivalence of minimal costs to some $o \in O'$ can be described by operations that are effective and compact.

By contradiction, assume that $\exists o \in O'$ for which no equivalence of minimal costs can be constituted by operations that are effective and compact. This implies that there exists an equivalence of minimal costs with operations from $O'$, that has strictly lower costs than the cheapest alternative present in $O_c$. As this equivalence cannot be found in $O_c$, one of the operations is not effective or compact. However, by their definitions, it is then possible to replace the operation in $O'$ with an equivalent operation or sequence of operations without incurring larger costs. This contradicts the above assumptions, which completes the proof. □

Obtaining an equivalent subset of minimal costs is desirable to deal with all of the possible operations more easily. However, it is not always preferred to construct a sufficient subset that is equivalent of minimal costs. In practice, it is better to deduce such set from some other sufficient subset through explicitly checking for violations to properties. The most valuable result is that of compactness with respect to the drone arrival as it is can be applied as a resource constraint. This can be done relatively easily while it can significantly reduce the size of sufficient subsets.

To conclude, one final property is introduced alongside the equivalence relations. From a different perspective, even in the equivalent subset of minimal size $O_s$, some operations can be left out. To be more specific, it could occur that the travel distance for the drone is large, which could make it possible for the truck to visit additional locations within the same period of time. Such inefficiencies caused by imbalances between the drone and truck route can be dealt with through the following property.

**Definition 5** (Balancedness)**.** An operation $o \in O$, with drone node $d_o$, is called balanced if there is no node $v \in V \setminus V_o$ that can be inserted into $T_o$, meaning that al locations currently in $T_o$ are visited in the same relative order, without increasing $c_o$.

In the worst case, imposing balancedness would rule out a particular optimal solution. However, a different solution with the same objective value can still be found by replacing non-balanced operations with balanced operations. As a result, the solution will visit some nodes multiple times.

### 3.3 *Mixed Integer Programming Formulation*

To solve the problem, essentially it has to be decided which operations to execute. A binary programming formulation for this problem is presented in Formulation 1 that has similarities with the set covering problem and the travelling salesman problem. The formulation improves on a number of aspects on the original formulation in Agatz et al. (2016).

In essence, the formulation decides upon which nodes in $V$ are designated as combined nodes and constructs a tour through these nodes. This is done by considering a sufficient subset $O \subseteq O'$ of operations. Binary decision variables $x_o$ are introduced that equal one if operation $o$ is chosen, and zero otherwise, together with auxiliary binary decision variables $y_v$ that equal one if node $v$ is used as an end node amongst the operations for which $x_o$ equals one. The costs for operation $o$ are denoted by $c_o$. Besides, $O(v)$, $O^-(v)$, $O^+(v)$ and $O^+(S)$ define subsets of $O$. Here, $O(v)$ is the set of operations that contain node $v$, and $O^-(v)$ and $O^+(v)$, are sets of operations that have $v$ as start and end node, respectively. Likewise, for a given $S \subset V$, the operations with their end node in $S$ and start node outside $S$ constitute $O^+(S)$. Lastly, each constraint is followed by the notation of a dual variable corresponding to that constraint which will be used in Section 4.

$$\text{Minimise} \quad \sum_{o \in O} c_o x_o \tag{2}$$

Subject to

$$\begin{aligned}
\textstyle\sum_{o \in O(v) \setminus O^+(v)} x_o &\geqslant 1, && \forall v \in V && (3) && \lambda_v \\
\textstyle\sum_{o \in O^+(v)} x_o - \sum_{o \in O^-(v)} x_o &= 0, && \forall v \in V && (4) && \mu_v \\
n \cdot y_v - \textstyle\sum_{o \in O^+(v)} x_o &\geqslant 0, && \forall v \in V && (5) && \eta_v \\
\textstyle\sum_{o \in O^+(S)} x_o - y_v &\geqslant 0, \quad \forall S \subseteq V \setminus \{v_0\}, && \forall v \in S && (6) && \eta_{vS} \\
\\
y_{v_0} &= 1, && \forall o \in O && (7) \\
x_o &\in \{0,1\}, && \forall o \in O && (8) \\
y_v &\in \{0,1\}, && \forall i \in V && (9)
\end{aligned}$$

Formulation 1: An improved formulation for the TSP-D problem.

The objective in (2) minimises the total time to serve all customers, which equals the sum of the execution times of the separate operations that are chosen. The constraints (3) and (4) require that the chosen operations form a tour. By constraints (3), every

customer is served at least once. Constraints (4) make sure that an operation that ends at a node is followed by an operation that starts at this node. An important remark is that the expression in (3) excludes the end node. This prevents the constraints to be loose and possibly improves the bound obtained from the linear relaxation. Next, the constraints in (5) are auxiliary and identify the combined nodes in the solution by setting the value of $y_v$. As a result, constraints (6) use these values to eliminate possible sub-tours in the solution. Essentially, these require a strongly connected graph by stating that for every subset $S$ of locations that consist of a combined node, an operation is chosen that starts outside $S$ and ends in $S$. In contrast to the original formulation, the set $S$ can be equal to $V \setminus \{v_0\}$. This ensures that the formulation behaves properly if the truck decides to stay at the depot. Lastly, (7) sets $y_{v_0}$ to 1 for computational efficiency, and (8)-(9) require binary decisions to be made.

Formulation 1 introduces the TSP-D problem in its elementary form. For more realistic applications of the problem, considerations such as a maximum flight time for the drone, restrictions on nodes that the drone and truck can visit, or sparseness of the network may be relevant. Most of these can easily be included by properly defining a sufficient subset $O \subseteq O'$.

# 4 METHODOLOGY

In this section the branch-cut-and-price algorithm will be introduced. Branch-cut-and-price is a generic approach that applies column generation and cutting planes during branch-and-bound. The approach can deal with problems that have a large number of redundant decision variables and constraints. This makes it a plausible approach for solving Formulation 1, whose number of decision variables $x_o$ and subtour elimination constraints increase exponentially in the number of locations. In the upcoming subsections an outline of the algorithm is provided, and its components are elaborated. At last, a heuristic approach is proposed that is based on branch-cut-and-price.

## 4.1 *Outline of Branch-Cut-and-Price*

Branch-cut-and-price avoids solving the entirety of Formulation 1 by solving a so-called restricted master problem, which abbreviates to RMP. This problem is restricted in the sense that only a subset of the decision variables and constraints from the original formulation are included. Consequently, the usual branch-and-bound scheme is applied to the RMP, except the scheme is extended with procedures to achieve that the quality of the solutions for the linear relaxations of the RMP and original formulation are equal. This is done by dynamically adding needful rows and columns, which represent variables and constraints, respectively, to the RMP. Cutting planes and column generation procedures, respectively, deal with dynamically including constraints and decision variables.

Cutting planes, sometimes called constraint generation, is a procedure that searches for constraints, in the original formulation, that are violated by the solution to the RMP. For this purpose, an optimisation problem called the separation problem is solved. The objective value of the separation problem indicates whether or not violated constraints exist. Furthermore, possible violated constraints are derived from the feasible solutions to the separation problem. As a side note, cutting planes is often used to tighten the feasible region of a problem while not changing the integer feasible region. However, in this paper cutting planes exclusively handles constraints that describe the feasible region of the problem.

Column generation operates like cutting planes and is used to dynamically include decision variables to the RMP. A so-called pricing problem is solved to find decision variables that have potential to improve the quality of the solution value, when their values are set to a non-zero value. The process is often referred to as pricing out variables. If no more variables can be priced out it is established that the RMP solution is optimal for the entire sufficient set $O$ under the constraints that are present in the RMP.

In the context of Formulation 1, the RMP is obtained by replacing the binary restrictions of (8) and (9) by linear restrictions between 0 and 1. Cutting planes is used for the exponentially growing number of subtour elimination constraints in (6). Furthermore, a proper sufficient subset can significantly reduce the number of $x_o$ variables but can still make the problem intractable as the number of locations increases. Therefore, it is convenient to apply column generation to the $x_o$ variables while $y_v$ can be included beforehand.

## 4.2 *Separation Problem*

A mixed integer programming formulation will be provided for the separation problem. The formulation models the difference between the left-hand and right-hand side of constraints in (6) to determine whether a violated constraint exists. Thereby, a constraint is characterised by a set $S \subset V \setminus \{v_0\}$ and a specific element $v \in S$.

Let a zero subscript correspond to the depot from now on and consider $i, j \in V$. The parameters in the model are determined by the solution values of $y_i$ and $x_o$ in the RMP. The chosen operations in the solution are accumulated based on their start node $i$ and end node $j$ to $\bar{x}_{ij} = \sum_{o \in O^-(i) \cap O^+(j)} x_o$. In addition, the outcomes of $y_i$ are represented by the parameters $\bar{y}_i$.

Introduce decision variables $\Phi_i$ that indicate whether or not $i \in S$ for some node $i$. Furthermore, let $\phi_{ij}$ be auxiliary decision variables, whose values depend on $\Phi_i$ and $\Phi_j$, which equal one if the event that $i \notin S$ together with $j \in S$ occurs, and zero otherwise. Finally, consider decision variables $\iota_i$ that achieve to set a constraint to active or inactive, and $y_{max}$ that corresponds to the maximum value of $\bar{y}_i$ among the $i \in S$. In Formulation 2 the mixed integer formulation is provided.

$$\text{Minimise} \quad \sum_{i \in V} \sum_{j \in V} \bar{x}_{ij} \phi_{ij} - y_{max} \tag{1}$$

Subject to

$$\begin{aligned}
\phi_{ij} &\geqslant \Phi_j - \Phi_i, && \forall i, j \in V && (2)\\
y_{max} &\leqslant \bar{y}_i \Phi_i + \iota_i, && \forall i \in V && (3)\\
\textstyle\sum_{i \in V} \iota_i &\leqslant n - 1 && && (4)\\
\\
\Phi_0 &= 0 && && (5)\\
\Phi_i, \iota_i &\in \{0, 1\}, && \forall i \in V && (6)\\
\phi_{ij} &\geqslant 0, && \forall i, j \in V && (7)\\
y_{max} &\geqslant 0 && && (8)
\end{aligned}$$

Formulation 2: The separation problem that searches for violated subtour elimination constraints.

The objective (1) finds the most violated constraint in (6) by creating a set S through the variables $\Phi_i$. Here, the summations in the objective accumulate all operations that start outside the set S and end inside S, which represents $\sum_{o \in O^+(S)} x_o$ from (6). The appropriate value of $\phi_{ij}$, corresponding to the event that $i \notin S$ together with $j \in S$, is guaranteed through constraints (2).

On the other hand, the severity of a violation also depends on the particular constraint corresponding to the set S. The variable $y_{max}$ is set through (3) and (4), which implicitly maximise the value of $\bar{y}_i$ amongst $i$ selected for S. Constraint (4) implies that one of the constraints in (3) must be active. Therefore, $y_{max}$ will be set to the largest possible $\bar{y}_i$. Furthermore, the depot may not be part of S as stated by (5), and (6)-(8) describe possible values for the decision variables.

A violated constraint is found if the objective is negative. If the optimal solution has a negative objective value, the decision variables $\Phi_i$ and the node $i \in S$ implied by $\iota_i$ depict the most violated constraint. Additionally, multiple violated constraints can be found by inspecting all feasible solutions that were found during the process of solving the mixed integer program.

### 4.3 *Pricing Problem Formulation*

The pricing problem contributes to the optimisation of the RMP over all operations in the sufficient subset O. Pricing out new decision variables can be done through

duality. Recall the dual variable definitions in Formulation 1. Given a solution to the RMP, dual values of constraints can be computed. Next, these can be used to determine the reduced costs of an operation $o \in O$, which is an indication of the effect of increasing $x_o$ on the objective value. In a minimisation problem negative reduced costs imply that the objective value potentially improves if the corresponding decision variable is added to the RMP.

Determining the reduced costs forms the basis of the pricing problem. To express the reduced costs of some $o \in O$ it is convenient to decompose it into a direct and indirect effect component. The respective components correspond to $c_o$, the cost of an operation, and the effect that arises from the dual values. The latter may be interpreted as the effect that is caused by changing the current solution to adopt a new operation in the solution.

To structure the pricing problem, the dual values are aggregated into effects. By the structure of Formulation 1, the coefficients that represent a decision variable $x_o$ in the RMP depend on whether the operation is part of $O(v)$, $O^-(v)$, $O^+(v)$ or $O^+(S)$, for $v \in V$ and $S \subseteq V \setminus \{v_0\}$. Stated differently, it depends on the nodes that $o$ covers, but also on the particular start node $u_o$ and end node $w_o$. Moreover, as implied by $O^+(S)$, it depends on the start and end node simultaneously. Denote these effects by $\pi_v$, $\pi_v^-$, $\pi_v^+$ and $\pi_{vv'}^{\pm}$, for $v, v' \in V$, which define, respectively, the effect of including $v$ as a truck or drone node, choosing $v$ as the start node, choosing $v$ as the end node, and simultaneously choosing $v$ and $v'$ as the start and end node, respectively. In the following expressions the effects are quantified.

$$\pi_v = \lambda_v \tag{11}$$

$$\pi_v^- = \lambda_v - \mu_v \tag{12}$$

$$\pi_v^+ = \mu_v - \eta_v \tag{13}$$

$$\pi_{vv'}^{\pm} = \sum\nolimits_{\{S \subseteq V \setminus \{v_0\} : v' \in S, v \notin S\}} \sum\nolimits_{i \in S} \eta_{iS} - \mathbb{1}_{\{v = v'\}} \lambda_v \tag{14}$$

Now, consider the reduced costs $r_o$ of an operation $o \in O$. It follows that the reduced costs depend on the direct effect and an indirect effect consisting of the four components discussed before. Given some operation $o = < u, w, d, T, V, c >$, the reduced cost is defined as

$$r_o = c - \Big( \pi_u^- + \pi_w^+ + \pi_{uw}^{\pm} + \pi_d + \sum_{v \in T} \pi_v \Big). \tag{15}$$

The pricing problem is that of finding operations $o \in O$ for which $r_o$ is negative. In essence, it is a problem that finds a path in the graph $G$ for the drone and truck. For column generation it suffices to find any operation with negative $r_o$ to advance. However, to guarantee that a solution is optimal with respect to the entire sufficient subset $O$ it must be proven that no operations exist with $r_o$ below zero. The apparent

way to do so is by solving an optimisation problem which has the additional benefit that the operation with the most potential is found.

It should be noted that optimisation approaches for the pricing problem are mostly extensions of elementary shortest path problems as the partial effect of $r_o$ ascribed to individual arcs in G may assume negative values. This makes the problem hard to solve because so-called negative cycles have to be dealt with. A potential way to solve the problem more easily is by reducing the feasible region, for example, by introducing resource constraints. In the context of operations, this makes the formulation of a proper sufficient subset O an important aspect of developing pricing algorithms.

Desirable properties for a pricing algorithm are that it is fast, exact and flexible. It is common that column generation takes a fair number of iterations to find an optimal solution. Therefore, it can be beneficial if the pricing algorithm provides multiple operations with negative reduced costs so that potentially fewer iterations are needed. Besides, it may be taken into account that finding o for which $r_o$ is minimal is not required until the last iteration.

The pricing algorithm has a considerable influence on the overall performance of the branch-cut-and-price algorithm. Therefore, two approaches are proposed in this paper. The first uses mixed integer programming and the second is a labelling algorithm. For a background reading on elementary shortest path problems and its solutions, such as labelling algorithms, the reader can turn to Irnich and Desaulniers (2005).

### 4.3.1 *Mixed Integer Programming Formulation*

The first method uses mixed integer programming. Formulation 3 elaborates this approach and aims to find the operation with minimal reduced costs. The formulation is presented in its most extensive form to account for all possible operations in $O'$ such as simple operations. In practice it would be better to include all simple operations beforehand as it will reduce the difficulty of solving Formulation 3. Besides, techniques are introduced later that shift the focus of the formulation towards a smaller sufficient subset of operations.

Formulation 3, makes a decision on variables $z_{ij}$, $d_i$ and $T_i$ which represent nodes. Here, $z_{ij}$ corresponds to the joint decision of the start node $i$ and end node $j$, and $d_i$ and $T_i$, respectively, describe whether the drone and truck visit node $i$. In addition, decision variables $a_{ij}$ and $a^d_{ij}$ indicate whether the truck or drone uses arc $(i,j)$, respectively, and help to ensure that proper paths are constructed. Finally, $t_i$ captures the arrival time of the truck at node $i$, whereas $M$ is a constant that is set sufficiently large to ensure that the formulation behaves as desired. More details on the value of $M$ are provided later.

The objective (16) minimises the reduced costs specified in (15). Together, (17) and (18) capture the direct component as the maximum between the costs of the drone

$$\text{Minimise} \quad c_o - \sum_{i \in V} \pi_i (T_i + d_i) - \sum_{i \in V} \sum_{j \in V} (\pi_i^- + \pi_j^+ + \pi_{ij}^{\pm}) z_{ij} \tag{16}$$

Subject to

$$\begin{aligned}
c_o &\geqslant \textstyle\sum_{i \in V} \sum_{j \in V} c_{ij}^d a_{ij}^d - M(1 - \sum_{j \in V} d_j) && (17)\\
c_o &\geqslant \textstyle\sum_{i \in V} \sum_{j \in V} c_{ij} a_{ij} && (18)\\
\textstyle\sum_{i \in V} \sum_{j \in V} z_{ij} &= 1 && (19)\\
\textstyle\sum_{i \in V} d_i &\leqslant 1 && (20)\\
\textstyle\sum_{j \in V} a_{ij}^d &= \textstyle\sum_{j \in V} z_{ij} + d_i, \qquad \forall i \in V && (21)\\
\textstyle\sum_{j \in V} a_{ji}^d &= \textstyle\sum_{j \in V} z_{ji} + d_i, \qquad \forall i \in V && (22)\\
T_i &= \textstyle\sum_{j \in V} a_{ji} - \sum_{j \in V} z_{ji}, \qquad \forall i \in V && (23)\\
T_i + d_i &\leqslant 1 - \textstyle\sum_{j \in V} z_{ji} - \sum_{j \in V \setminus \{i\}} z_{ij}, \qquad \forall i \in V && (24)\\
\textstyle\sum_{j \in V} a_{ji} - \sum_{j \in V} a_{ij} &= \textstyle\sum_{j \in V} z_{ji} - \sum_{j \in V} z_{ij}, \qquad \forall i \in V && (25)\\
z_{ii} &\geqslant a_{ii}, \qquad \forall i \in V && (26)\\
t_i + c_{ij} &\leqslant t_j + M(1 + z_{ii} - a_{ij}), \qquad \forall i, j \in V && (27)\\
\textstyle\sum_{i \in V} T_i &\leqslant n \textstyle\sum_{i \in V} d_i, && (28)\\
d_0, T_0 &= 0 && (29)\\
a_{ii}^d &= 0, \qquad \forall i \in V && (30)\\
a_{ij}, a_{ij}^d, z_{ij} &\in \{0, 1\}, \qquad \forall i, j \in V && (31)\\
T_i, d_i, t_i &\geqslant 0, \qquad \forall i \in V && (32)
\end{aligned}$$

Formulation 3: The pricing problem formulation.

path and truck path. Besides, in (17) the term with $M$ in it accounts for the special case where no drone node is chosen, which makes the drone path irrelevant.

Next, constraints (19)-(22) describe the actions of the drone. The first two ensure that a single start and end node, and at most one drone node is chosen. Furthermore, (21) and (22) construct a path along the start node, drone node and end node, and if no drone node is chosen the path consists of a single arc.

The next set of constraints, (23)-(26), account for the actions of the truck. A truck node is identified by constraints (23). Next, (24) states that a chosen node can be

designated as a start node, end node, drone node or a truck node only once. The path of the truck is described by (25). Here, for node $i \in V$, the right-hand side equals -1 if $i$ is the start node, 1 if $i$ is the end node and 0 otherwise. Therefore, the number of arrivals and departures in each node must be equal except for the start and end node. In these cases, only a departure and arrival occurs at the start and end node, respectively. Besides, (26) ensures that the truck may only stay at a node if it is both the start and end of the operation. Lastly, (27) capture the arrival time of the truck at each node. They also act as subtour elimination constraints and allow the truck route to be either a path or a tour.

For the final constraint recall that $n$ represents the number of locations in the problem. (28) enforces the operation to be simple if no drone node is chosen. Furthermore, (29)-(32) describe the range of the decision variables. Here, note that the binary restrictions of $T_i$ and $d_i$ are relaxed as the variables are artificial to the problem. Furthermore, the depot cannot be a drone or truck node and the drone cannot stay at the same node.

Turning to the value of $M$, the value of the most expensive operation in the problem suffices for $M$ in order to allow constraints involving $M$ to be non-restrictive to the problem when desired. In the implementation the maximum of twice the distance of $\max_{i,j\in V} c^d_{ij}$ and $n$ times $\max_{i,j\in V} c_{ij}$ is used.

To improve the pricing algorithm enhancements are made to Formulation 3. As a start, all simple operations can be provided beforehand to the RMP. This removes the need to account for simple operations. The implications for the model are that the second term in constraint (17) and the entirety of constraint (28) can be removed. In addition, the inequality in (20) becomes and equality. Next, constraints are introduced so that the possible solutions constitute a smaller sufficient subset. More specifically, compactness and balancedness are imposed. The benefits of this are twofold. Firstly, there is a potential that Formulation 3 can be solved more quickly. On top of that, the lower bound obtained from the RMP may increase which is beneficial for branch-and-bound.

Before turning to compactness and balancedness, the artificial variable $W^+$ is introduced. $W^+$ defines the time the truck has to wait for the drone at the end of the operation. Besides, it will be required that $t_i$ is set to the earliest arrival time of the truck at a node $i \in V$. In Formulation 3, this was not necessarily the case.

$$W^+ \geqslant \sum_{i\in V}\sum_{j\in V} c^d_{ij} a^d_{ij} - \sum_{i\in V}\sum_{j\in V} c_{ij} a_{ij} \tag{33}$$

$$W^+ \geqslant 0 \tag{34}$$

$$t_i \leqslant \sum_{i\in V}\sum_{j\in V} c_{ij} a_{ij}, \qquad \forall i \in V \tag{35}$$

With the addition of constraints (33)-(35) the values of $W^+$ and $t_i$ are uniquely defined. For $W^+$, this can be seen by the fact that the objective stated by (16) implicitly minimises the value for $W^+$.

In (36)-(38) the constraints for compactness and balancedness are provided. Here, $\epsilon$ is a small constant that is used to achieve the strict inequality of the constraints. Therefore, $\epsilon$ can be set to the highest precision amongst $c_{ij}$ and $c^d_{ij}$. For example, $\epsilon$ equals 0.1 if all costs are reported with at most 1 decimal place.

$$t_i + W^+ + \epsilon \leqslant c^d_{ji} + \textstyle\sum_{k \in V} c^d_{kj} a^d_{kj} + M(2 - T_i - d_j), \quad \forall i, j \in V \tag{36}$$

$$c_o - t_i + \epsilon \leqslant c^d_{ij} + \textstyle\sum_{k \in V} c^d_{jk} a^d_{jk} + M(2 - T_i - d_j), \quad \forall i, j \in V \tag{37}$$

$$W^+ + \epsilon \leqslant c_{ik} + c_{kj} - c_{ij} + M(1 + T_k - a_{ij}), \quad \forall k \in V \setminus \{v_0\} \quad \forall i, j \in V \tag{38}$$

The above constraints (36) and (37) impose compactness. The former collection states that the waiting time of the truck must increase if the drone would return to the truck earlier, at some truck node $i$. In turn, this would imply the total costs to increase. Also, by the latter constraints, the total costs must increase if the drone were to depart at a truck node $i$ instead. In both cases the last terms, that include $M$, require the constraint to hold only if $i$ is a truck node and $j$ is a drone node. Next, (38) requires operations to be balanced. The constraints state that the costs of the operation increase if an additional node is inserted on the path of the truck. Similarly, here the terms with $M$ require the expression to hold only if the truck uses the arc $(i, j)$ and $k$ is a node that is not covered by the operation. Besides, the value of $M$ introduced before remains sufficient with the addition of the foregoing constraints.

As only a small number of the constraints in (36)-(38) are relevant, the best performance is achieved by including these constraints dynamically. As the number of compactness constraints is manageable, they are implemented as lazy constraints, meaning constraints are created but only added to the problem if they are violated. However, the number of balancedness constraints can grow unnecessarily large while the non-negativity of $\pi_v$ implies that the constraints are only required for $k$ for which $\pi_k$ is zero. Therefore, it is wiser to create such constraints on the fly by explicitly inspecting a non-balanced solution.

Two final remarks are discussed. Firstly, it may occur that pricing out certain operations must be avoided due to branching. Provided that this does not occur too often, a quick fix is to reject the solution. This option is available in commercial integer programming software packages such as CPLEX. Secondly, in column generation often multiple columns are added instead of only the best column. The most obvious way to accomplish this is to consider all feasible solutions obtained during the process of solving Formulation 3. All feasible solutions with negative reduced costs that are found can be added to the RMP or a selection can be made. Alternatively, the problem can be split up by setting $d_i$, $z_{ij}$ or both beforehand to find more solutions. Unfortunately, such a good performing approach could not be established through preliminary experiments.

### 4.3.2 *Labelling Algorithm*

The second method adopts labelling techniques for elementary shortest path problems with resource constraints. A label-correcting algorithm will be presented that is based on finding the shortest path from a provided start node to all other nodes in the network. In essence, this path is related to the truck and is subject to restrictions that depend on some pre-decided drone node. Thereby, the algorithm considers every possible combination of the start and drone node. However, the labelling algorithm does not search for simple operations which should be taken into account by the overall algorithm.

The key feature of the technique is that paths are represented by a chain of labels and extended through dynamic programming. Due to possible negative arc costs, the number of nodes on the path can grow large. In an attempt to increase computational efficiency, compactness with respect to the drone return is imposed in the form of resource constraints. Denote a label by $L$ and let $P_L$ be the path that $L$ represents which is found by going back along the chain of preceding labels. The notation for a label is as follows.

$$L \equiv (j, u_L, d_L, R_L, t_L, r_L) \tag{39}$$

Each label corresponds to a node $j \in V$. This is the node where the path $P_L$ ends. Furthermore, $u_L$ and $d_L$, respectively, represent the start node and drone node that are set during the creation of the initial labels. For this reason, they will be called the prefix of $L$. The resource consumption of the label is stated by $R_L$ and $t_L$. Here, $R_L$ is the set of nodes that $L$ covers and $t_L$ are the costs of the truck path $P_L$. Finally, $r_L$ is an indication of the reduced costs. Until completion, this value accounts for the start node, drone node and truck path. Only when the end node is chosen explicitly the drone path and end node are accounted for.

New labels are created by extending existing labels. If no more labels can be extended the operation with the smallest reduced costs can be found among the completed labels. Optionally, labelling techniques may rule out redundant labels while searching for the optimal solution. Such algorithms are in the class of label-correcting algorithms which typically use a dominance rule to establish that a label is redundant. For the pricing problem, a dominance rule based on efficiency is introduced in Definition 6.

**Definition 6.** Consider two labels $L_1 = (j, u, d, R, t_1, r_1)$ and $L_2 = (j, u, d, R, t_2, r_2)$ defined on the same node with identical prefixes. Then, $L_2$ is dominated by $L_1$ if $r_1 \leqslant r_2$.

In words, a label $L_2$ is dominated if a label, say $L_1$, with reduced costs $r_1$ below $r_2$ exists that has the same start, end and drone node, and covers the same nodes as $L_2$. The above conditions suggest that $L_1$ can always be extended in the same way as

$L_2$, hence $L_2$ can never produce a better solution than $L_1$ and is redundant. Another implication of the conditions is that the indirect component of the indicated reduced costs must be equal. As a consequence, differences between $r_1$ and $r_2$ must be due to differing direct components. In particular, for the considered labels $t_1 < t_2$. Since these represent the costs for the truck path, it follows that Definition 6 eliminates inefficient truck paths.

Aside from dominance, a lower bound on the possible reduced costs may be computed to discard labels that appear to be unable to produce an operation with negative reduced costs. A possible way to do so involves estimating for how much longer a label can be extended and overestimating the reduced costs for successive labels. In turn, the former requires a measure of expandability to be chosen, such as travel time or the number of truck nodes on the truck path. Both of the aforementioned measures were considered but it was decided to go with the latter as the estimated bounds turned out to be better.

Consider a label $L = (j, u, d, R, t, r)$ and let $\bar{r}$ denote an estimated lower bound on the reduced costs for labels that have $L$ in their chain of preceding labels. Next, let $\rho$ be an estimate on the maximum possible number of nodes in a compact operation, not less than the true value. The details on determining $\rho$ are provided in Appendix A for it being too comprehensive for the main text. Finally, denote $c_{min} = \min_{i,j \in V, j \neq 0}\{c_{ij} - \pi_j, 0\}$, which can be interpreted as the value of the best arc in the pricing problem, and denote $\omega_u = \min_{j,w \in V, j \neq 0}\{c_{jw} - \pi^+_w - \pi^\pm_{uw}\}$, which represents the value of the best possible change in reduced costs for the explicit choice of the last arc conditional on the start node. Now, a possible lower bound is given by

$$\bar{r} = r + c_{min}(\rho - |R|) + \omega_u. \tag{40}$$

The expression assumes that labels are extended using the most profitable arc and completed by the best last arc conditional on the start node. In the multiplication with the number of nodes, implicitly 1 is subtracted as one node is reserved for the end node and 1 is added to correct for the fact that $R$ contains the drone node. Besides, $c_{min}$ is at least zero as the truck path does not have to be extended by truck nodes if this is not beneficial. Note that the possible waiting time for the truck can be ignored as this overestimates the reduced costs.

Pseudocode for the label-correcting algorithm is provided in Algorithm 1. Here, $\Lambda_v$ is the set of labels that are defined on node $v \in V$ and $\Lambda^*$ is the set of completed labels with negative reduced costs. Furthermore, $H$ and $W$ are local values that represent, respectively, the nodes that a label can be extended to and the waiting time for the truck if a rendezvous with the drone occurs at the next node.

The algorithm starts with initialising data structures and creating initial labels for all possible combinations of the start node and drone node. Next, the queued labels are processed sequentially. First, each considered label is subject to a completion procedure that ends the truck path with every possible end node. Here, the reduced

**Algorithm 1** Pseudocode for the label-correcting algorithm.

**Input:** Graph $(V, E)$, with arc costs $c^d_{ij}$ for the drone and modified arc costs $c'_{ij} = c_{ij} - \pi_j$ with associated resource consumption $c_{ij}$ for the truck.
**Output:** Operation with minimal reduced costs.

**procedure** INITIALIZATION
    $Q$ = new queue of labels;
    $(P^*, d^*, r^*)$ = best operation with corresponding reduced costs;
    $\Lambda^*$ = new set of completed labels;
    **for all** $u \in V$ **do**
        $\Lambda_u$ = new node of labels;
        **for all** $d \in V$ **do**
            $L = (u, u, d, \{u, d\}, 0, -\pi^-_u - \pi_d)$;
            $\Lambda_u = \Lambda_u \cup \{L\}$;
            add $L$ to $Q$;

**while** $Q \neq \emptyset$ **do**
    take $L = (j, u, d, R, t, r)$ representing path $P$ in front of the queue;
    $H$ = new set of nodes;
    **procedure** LABEL COMPLETION
        **for all** $w \in V \setminus R \cup \{u\}$ **do**
            $W = c^d_{ud} + c^d_{dw} - t - c_{jw}$;
            $r' = r + c_{jw} - \pi^+_w - \pi^{\pm}_{uw} + \max(W, 0)$;
            **if** $W > 0$ and $w \neq v_0$ **then**
                $H = H \cup \{w\}$;
            **if** $r' < 0$ and the operation is compact and balanced **then**
                $L' = (w, u, d, R \cup \{w\}, t + c_{jw}, r')$;
                $\Lambda^* = \Lambda^* \cup L'$;
                **if** $r' < r^*$ **then**
                    $(P^*, d^*, r^*) = (P \cup \{w\}, d, r')$;
    **procedure** LABEL EXTENDING
        **for all** $v \in H$ **do**
            $L' = (v, u, d, R \cup \{v\}, t + c_{jv}, r + c'_{jv})$;
            estimate $\bar{r}$ for $L'$;
            check for dominance relations between $L'$ and labels in $\Lambda_v$;
            **if** $\bar{r} > 0$ or $L'$ is dominated **then**
                continue;
            delete labels dominated by $L'$;
            $\Lambda_v = \Lambda_v \cup \{L'\}$;
            add $L'$ to $Q$;
**return** $(P^*, d^*, r^*)$;

costs $r'$ are corrected if the truck incurs waiting time $W$ at the end node. Besides, labels that can still be extended, so that compactness with respect to the drone return is maintained, are marked through lines 19 and 20. The result is a completed label, which is kept if the associated operation has negative reduced costs. Operations that violate compactness or balancedness are filtered out, which is reasonable as explicit checks can be done quickly so that it is unlikely to worsen the overall algorithm. Whenever a label is kept and improves upon the best-found solution, this solution is updated.

Next, a label extending procedure is followed by the completion procedure. Here, the path of the truck is extended only if the drone cannot arrive prior to the truck at the next node. In addition, the new label must show a potential to produce an operation with negative reduced costs and cannot be a dominated label. If the label is qualified, it will be queued and possibly be further extended later. Moreover, all labels that are dominated by the new label are removed.

These two procedures repeat until all labels have been processed after which the operation with minimum reduced costs can be identified. Subsequently, all operations with negative reduced costs can be added to the RMP or a selection among these operations can be made.

## 4.4 *Branch-and-Bound*

Up to this point, it has been discussed how the linear relaxation of Formulation 1 can be solved by cutting planes and column generation. The next step is to apply branch-and-bound to work towards an integer feasible solution. The outline of branch-and-bound is briefly discussed followed by the specific strategies that are proposed for the problem at hand.

Branch-and-bound gradually partitions and cuts downs the feasible region in such a way that the integer feasible region is preserved. Branching operates on a tree structure of nodes where each node represents a restricted version of the problem in the preceding node. To expand the tree a node selection strategy is needed to choose a node in the tree to branch on, together with a branching strategy that describes how to create new node problems. Furthermore, the method keeps track of an upper and a lower bound to explore the tree more efficiently and to gain an understanding of the quality of a feasible solution. In the context of Formulation 1, an upper bound is provided by the best feasible solution and the lower bound corresponds to the least lower bound among the leaf node problems, which are the nodes that have not yet been branched.

The node selection strategy that is considered is the best-bound strategy. This strategy selects the node with the smallest lower bound to branch on, ensuring that nodes are never explored unnecessarily after updating the upper bound. Besides, depth-first-search is a strategy that branches on the longest branch in the tree and is often used to put emphasise towards finding a feasible solution to the problem. However, for the

TSP-D problem this would not be effective as the available heuristics already perform relatively well.

The branching strategy that is most common is branching on fractional variables. In Formulation 1 the decision variables are binary so that two branches are created that restrict the variable to zero or one. Branching strategies that choose the fractional variable closest to either zero, a half and one will be considered. In addition, a strong branching extension to these rules is considered that selects multiple fractional variables, creating tentative branches. After solving the branches, only the tentative branch that provides the best lower bound is kept. Despite the fact that more problems have to be solved, strong branching can significantly reduce the size of the tree.

In branching strategies special attention has to be paid to the underlying problem. Formulation 1 exhibits a large number of $x_o$ variables. As a result, branching on a $x_o$ is very inefficient as fixing $x_o$ to zero hardly restricts the problem, whereas requiring $x_o$ to be one is very restrictive. Branching on $y_v$ seems to be plausible as it would, respectively, require and forbid a node to be a combined node. Still, binary valued $y_v$ do not guarantee binary decisions on $x_o$, therefore, it can only be achieved by branching on fractional $y_v$ with greater priority than $x_o$. Preliminary experiments confirm that this works well as integer solutions can be found relatively fast while branching on $x_o$ is rarely needed.

Since the strategy performs so well an alternative strategy that branches on arcs will not be considered. Intuitively, this method is expected to not outperform branching on combined nodes as the branching decisions are well balanced for the latter strategy. Besides, the proposed branching strategy requires almost no synergy with the pricing problem in the context of the branch-cut-and-price algorithm. Restrictions on $y_v$ are fully handled by the formulation while carrying over the restriction for $x_o$ to the pricing problem requires little effort as explained in the previous subsection.

## 4.5 *Additional Procedures and Considerations*

The discussion of the branch-cut-and-price algorithm is finalised with a discussion of some important steps involved. These include finding an initial solution, the creation of initial operations, maintaining a candidate pool of operations, column and row management, and a local search procedure that attempts to find feasible solutions outside the node problems. In the flow chart in Figure 2 a detailed overview of the complete branch-cut-and-price algorithm is provided. Here, the main components have already been discussed and the other aspects will be further explained in the remainder of this subsection.

First, consider the initialisation steps in Figure 2. An *initial solution* that provides a strong upper bound for branch-and-bound significantly cuts computation times. In Agatz et al. (2016) route first-cluster second heuristics were introduced that have

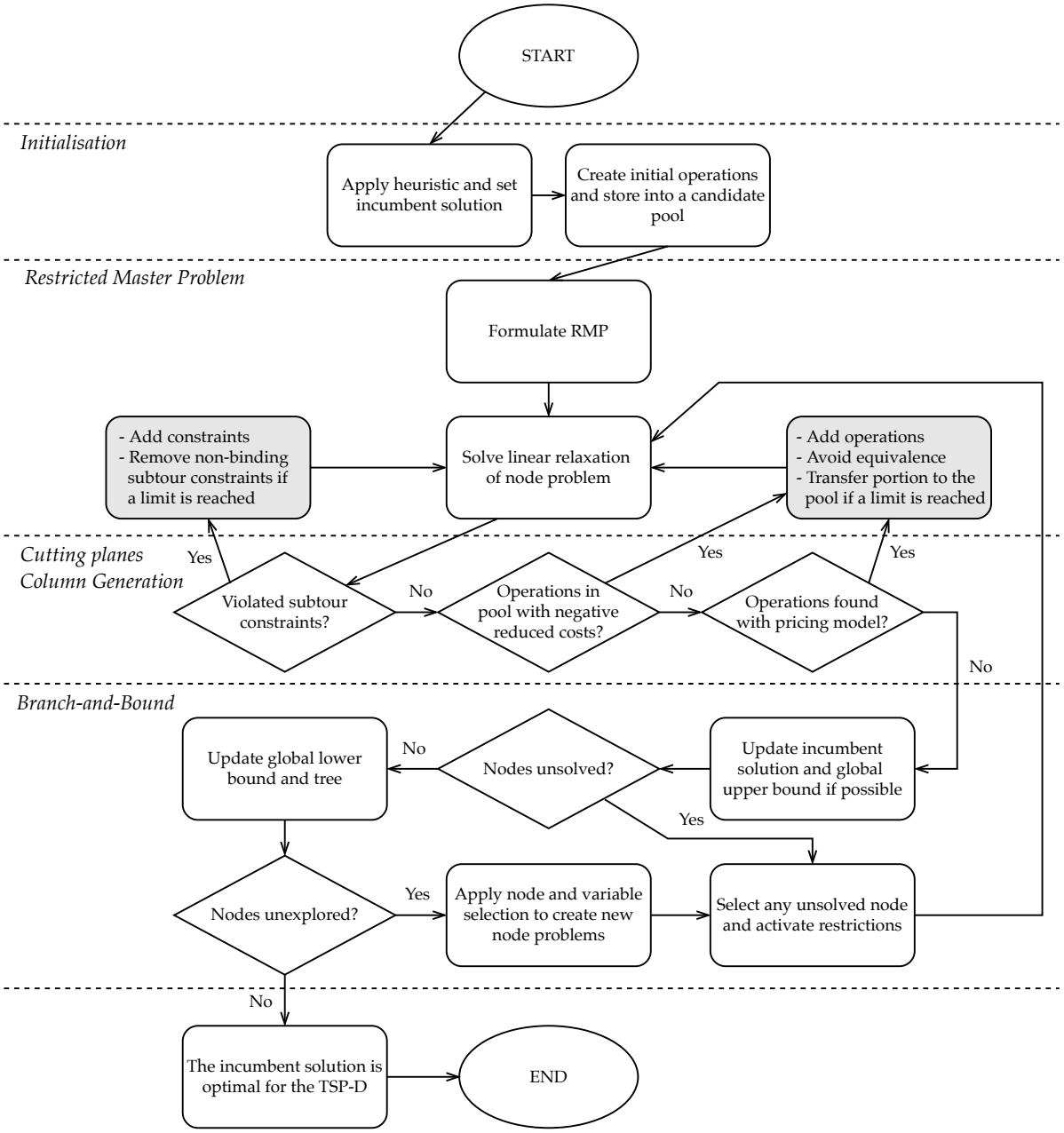


Figure 2: Flowchart of the in-depth functioning of the exact branch-cut-and-price algorithm.

shown to be fast and perform well. The TSP-ep-all configuration showed to produce the best results and will be used in this paper for obtaining initial solutions.

In addition, initial operations can be provided to the problem. Smaller operations, being relatively cheap, are expected to be useful and can be created relatively easily. Though, it would be inefficient to simply add such operations to the RMP as many operations will still turn out to be redundant. A better approach would be to maintain a *candidate pool* of operations that is checked prior to invoking the pricing algorithm. Thereby, the reduced costs of all operations in the pool can be checked in the order of less than a second. In this way, the possible bottlenecks occur from external factors such as the computational effort that is required to create initial operations and memory resources.

Sufficient subsets of initial operations are created through an adjusted version of Algorithm 1 that essentially removes the elements involving reduced costs. Note that in Bouman et al. (2017) sufficient subsets were created in a truck first-drone second fashion, while here a *drone first-truck second labelling* approach is used. Afterwards, equivalence relations are ruled out which results in an almost equivalent subset of minimal costs. This is because sometimes an ineffective operation cannot be ruled out as the equivalent operation with smaller costs is in violation with compactness and has already been discarded. In order to create only a part of the sufficient subset a restriction can be introduced for the number of labels created or the number of truck nodes in an operation.

Next, Figure 2 shows the order of execution for column and constraint generation. It is decided to apply cutting planes prior to column generation as it is plausible to only generate new columns on solutions that are feasible with respect to the subtour elimination constraints. Otherwise, there is a chance that column generation leads into an infeasible direction, resulting in redundant computations. Moreover, preliminary results show that constraint generation is less intensive than column generation.

In the shaded boxes in Figure 2 *memory management* procedures are described. If the algorithm runs for a long time the model can grow large while some rows and columns that were added will rarely be used for future node problems. The box at the left describes this for the subtour elimination constraints. If a specified limit is exceeded all constraints that are not binding are simply removed from the problem. Similarly, for the box at the right, it is decided to transfer operations with large reduced costs to the pool. As a side note, the specific limits are not of great importance as cutting planes and searching in the candidate pool are relatively fast procedures.

As the number of locations grows, it becomes more difficult to find feasible solutions during branch-and-bound. A *local search* procedure is proposed that solves a subproblem periodically as a TSP-D problem under a restricted set of operations with branch-and-cut. Here, the number of selected operations should preferably be large while still being able to solve the subproblems fast. Selecting operations can be based on a measure of quality such as the recent occurrence in the solution of a node problem.

Consequently, a separate set of operations is maintained for the local search routine and the TSP-D subproblem is solved every time significant alterations have been made to this set. As a side note, the procedure is not displayed in Figure 2 because local search will only be considered outside exact settings.

Finally, the use of branch-cut-and-price is motivated for large problems. Problems become intractable for mainly two reasons: the branch-and-bound tree becomes too large and the pricing problem becomes significantly hard to solve. Nonetheless, branch-cut-and-price can be used as an optimiser to look for good quality solutions by restricting the solution space, while a trade-off has to be made between enhanced computational efficiency and potential quality of the solution.

# 5 RESULTS

The goal of this section is to assess the performance of branch-cut-and-price and to benchmark it against branch-and-cut. Mostly, attention will be paid to solving problems optimally but experiments with an emphasis on feasibility are also considered. An outline of the experiments together with the parametrisation choices for the models will follow now. Thereafter, the results are shown and discussed.

## 5.1 *Experimental Framework*

Broadly speaking, three exact methods are considered which all operate on a sufficient subset of balanced and compact operations. The first is the branch-cut-and-price algorithm (BCP) for which two different pricing algorithms were discussed. Second, a pooled branch-and-cut (PBC) variant is considered that creates the entire sufficient subset beforehand but preserves memory by storing the operations in a candidate pool instead of adding them directly to the RMP. Finally, the third method is a full branch-and-cut (FBC) approach and uses a commercial branch-and-cut solver for the entirety of Formulation 1. The idea behind the three methods is that BCP can be benchmarked against PBC to assess the value of column generation. In addition, comparing PBC and FBC allows assessing the efficiency of the branch-and-cut implementation.

Before turning to the main experiments, sufficient subsets are to be created for several instances to study the value of the properties introduced in Section 3. This includes a comparison between two possible approaches to creating initial operations: the truck first-drone second (TFDS) procedure from Bouman et al. (2017) and the truck first-drone second (DFTS) approach adopted from Algorithm 1 in this paper. Furthermore, small experiments will be conducted to compare various implementations of the main methods. For BCP these are related to the branching rule, the pricing algorithm and pool of initial operations, while for PBC only the branching rule needs to be considered and for FBC the commercial solver fully controls the implementation.

At last, experiments are conducted for large instances in a heuristic setting. The exact methods are extended with the local search routine (LS), discussed at the end of Section 4.5, and the solution space is restricted by restricting the creation of initial operations through the number of labels created in the DFTS labelling method. Besides, LS keeps track of a set of operations that were most recently part of the solution to a node problem and solves a subproblem for these operations every time this set has changed sufficiently. Different limits on the size of this set are considered together with different values for the threshold that triggers the routine.

Instances of the problem are created randomly by allocating locations on a plane uniformly. Consequently, travel costs are determined by Euclidean distances. To study the sufficient subsets and the computational complexity of the problem, various travel speeds of the drone relative to the truck are considered. For the main experiments, however, the drone travel speed is assumed to be twice that of the truck as it provides a realistic setting for the TSP-D. For each problem size, 10 instances are created and solved by each of the methods. Besides, separate instances are used to tune the implementation for the methods which allows for a fair comparison between the final implementations. It should also be noted that results are shown for the number of locations which excludes the depot.

## 5.2 *Technical Details*

Preliminary results have shown that a sensible limit for the number of rows and columns in the RMP is 10,000 which is based on the observation that the limits are not reached often. Furthermore, resetting the rows and columns is inexpensive compared to the increased efficiency for solving node problems. For the column management procedure it is reasonable to keep a small portion of the operations in the RMP during a reset which is decided to be 10% of the limit.

During column generation all operations that have been found by the pricing models are added to the RMP, up to a limit of 100 operations per iteration to prevent a large number of operations with small negative reduced costs to be added. Besides, there is a chance that a node problem is irregular with a solve time that is well above average. To make the algorithm more robust a time limit of 10 minutes is given to node problems. Instances that often require more than 10 minutes per node problem are unlikely to be solved within a reasonable time anyway because of the number of node problems. Furthermore, the same time limit applies to the local search routine. Lastly, recall that the cutting planes procedure tries to find a violated constraint through a set $S \subseteq V \setminus \{v_0\}$ and a specific element $v \in S$. For efficiency, the constraints corresponding to all $v \in S$ are checked and the violated constraints are added to the RMP.

For the main experiments the initialisation procedures must be done within 3 hours, else the algorithm is terminated as computational resources are likely to be insufficient, and for the remainder of the algorithm a time limit of 12 hours is used. Besides, a

maximum of 5 minutes is provided to estimate the truck node bound $\rho$ as discussed in Appendix A which both pricing algorithms can exploit to narrow down the set of possible solutions.

The framework is implemented in Java 8 and uses CPLEX 12.6.3 as a linear programming solver and as a black box branch-and-cut solver for FCP. Experiments are conducted on nodes of the Lisa computer cluster of SURFsara. Every experiment is run on a single core @ 2.6GHz (Intel® Xeon® E5-2650 v2) and is provided 4GB of RAM.

## 5.3 *Experimental Results*

This subsection is structured as follows. First, the construction of sufficient subsets is considered and the computational complexity of the problem is explored. In addition, the performance of the estimator for a bound on the number of truck nodes amongst operations in a sufficient subset is examined. Next, the exact algorithms are compared. Finally, the exact algorithms are extended by a local search routine and the performance on large problems is discussed.

### 5.3.1 *Sufficient Subsets*

Sufficient subsets are constructed through TFDS introduced in Bouman et al. (2017) and DFTS proposed in this paper. For both approaches, results are collected for instances ranging from 5 to 35 locations and a drone travel speed relative to the truck, of 1:1, 2:1 and 3:1.

It turns out that DFTS consistently outperforms TFDS in terms of the time required to construct the sufficient subset and the size of this set, thus favouring the newly proposed approach. Generally speaking, the size of the sufficient subset grows exponentially in the number of locations, however, for TFDS this occurs at a larger rate. The difference in performance can be explained by the fact that with DFTS the operations in the sufficient subset are compact and balanced, whereas TFDS merely guarantees for efficient operations. Selected results from the experiments are provided in Table 1, showing only results for instances for which the maximum run-time among 10 instances did not exceed 1 hour. The largest number of locations for which this condition holds are reported in the upper part of the table.

For the purpose of illustration, consider a relative travel speed of 2:1 and 20 locations in the table. The first part of the table shows that DFTS managed instances up to 27 locations, while TFDS was not able to go beyond 20 locations. In addition, the difference in performance can be deduced from the second part in the table by recalling that DFTS constructs compact and balanced operations. Consequently, the sufficient subset created by DFTS is 1.1% of the size created by TFDS, averaged over 10 instances. This large reduction is mostly due to compactness, seeing that only 1.5% of the operations were compact for TFDS. A plausible explanation is that compactness

Table 1: Results for the creation of an entire sufficient subset derived from 10 instances, showing results for the largest instances that TFDS handled within 1 hour.

| | **Relative drone travel speed:** | | |
|---|---|---|---|
| | 1:1 | 2:1 | 3:1 |
| ***Locations in the largest instance managed*** | | | |
| TFDS - Bouman et al. (2017) | 15 | 20 | 26 |
| DFTS - this paper | 17 | 27 | 35 |
| ***Sufficient subset characteristics of TFDS**†*** | | | |
| Operations created | 2,467,762 | 8,556,474 | 8,318,534 |
| Compact operations | 19.9% | 1.5% | 1.3% |
| Balanced operations | 87.2% | 99.6% | 99.7% |
| Compact and balanced operations | 7.1% | 1.1% | 1.0% |
| ***Largest number of truck nodes observed**††*** | | | |
| TFDS - Bouman et al. (2017) | 12.6 | 9.5 | 7.8 |
| DFTS - this paper | 10.5 | 7.4 | 6.6 |

* Statistics are related to the largest instances managed by TFDS (first row of results) and averaged over 10 instances.
† For example, assuming relative speed 1:1 and averaged over ten instances with 15 locations, TFDS created 2,467,762 operations of which 87.2% were balanced.
†† For example, assuming relative speed 2:1 and averaged over ten instances with 20 locations, the largest operation had 9.5 truck nodes for TFDS opposed to 7.4 for DFTS.

significantly limits the size of operations, which is supported by the bottom part of the table showing that the largest operation in DFTS had on average 1.9 fewer truck nodes than in TFDS. Lastly, note that 99.6% of the operations were balanced, which is rather large, however, the reduction from 1.5% to 1.1% that is possible through balancedness is still significant. Overall, the contribution of compactness seems to increase as the drone speed increases, while the opposite holds true for balancedness.

Figure 3 shows the average number of operations created by DFTS. The plot shows that the size of the sufficient subset increases exponentially in the number of locations, and at a lower degree as the drone travels faster. Particularly, the results for a travel speed of 2:1 are promising as entire sufficient subsets can be created for relatively large instances. For example, with 20 locations the sufficient subsets consisted of 94,369 operations on average. It is likely that such problems can be solved by just branch-and-cut which would not have been possible before.

Besides, the bound on the number of truck nodes $\rho$, which is defined in Section 4.3.2 and elaborated in Appendix A, is estimated and compared to the true value. The largest maximum error averaged over 10 similar instances was 1.2 among the various instances for which results were available. This justifies that the estimation procedure in Appendix A is a suitable approach to obtain a good estimate within a short period of time, being five minutes.

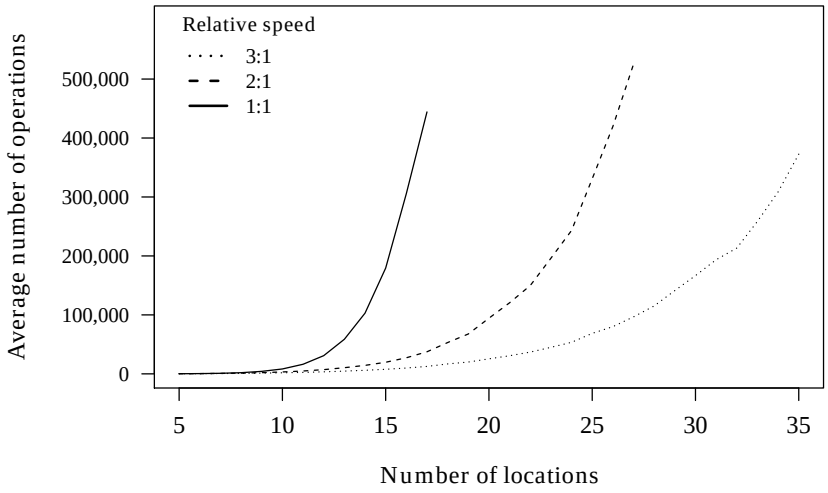


Figure 3: Number of operations in the sufficient subset created by DFTS.

### 5.3.2 *Optimal Setting*

In the remainder of the experiments the various algorithms will be studied. Here attention is restricted to a relative drone travel speed of 2:1. First the branching rules will be compared. This is done through PBC as the branching dynamics are mostly independent of solving individual node problems. In the first stage of the experiment, fractional rules, thus considering a single candidate, that choose the variables closest to zero, a half and one are compared. In the second stage, the best performing fractional rule is extended into a strong branching rule and studied for several numbers of branching candidates. In Table 2 the average solve times for instances with 10, 15 and 20 customer locations are shown for both stages at once.

Table 2: Results, displaying solve times averaged over 10 instances for PBC, for the two stages of branching experiments, exploring the fractional rule and the strong branching rule extended from the best fractional rule, respectively.

| | **Stage 1 (fractional) Fraction:** | | | **Stage 2 (strong on fraction 1) Candidates:** | | | |
|---|---|---|---|---|---|---|---|
| **Locations** | ***0*** | ***1/2*** | ***1*** | ***2*** | ***3*** | ***4*** | ***5*** |
| 10 | 6 | 4 | 4 | 4 | 5 | 6 | 7 |
| 15 | 43 | 41 | 43 | 33 | 26 | 26 | 26 |
| 20 | 1:18:19 | 14:47 | 12:29 | 5:35 | 1:52 | 1:51 | 1:42 |

The left part of Table 2 shows that the performance of the fractional branching rules are similar for the smaller instances. For the instances with 20 locations, however, the closest to 1/2 and 1 rules clearly outperform the closest to 0 rule. In turn, the closest to 1 rule performs slightly better, therefore, it will be considered for the strong branching extension.

The right part of the table shows that the solve time decreases as a result of strong branching. It appears that beyond choosing up to 3 branching candidates only slight efficiency gains are realised. Therefore, it is decided to go with the strong branching rule with 3 candidates in combination with the closest to 1 fractional rule.

Secondly, the two pricing algorithms and different pools of initial operations are compared for BCP. Operation pools are created for all operations with up to 0, 1 or 2 truck nodes so that the initialisation time is kept small. The performance measure is the average time to solve a node problem. This is computed as the total time, including initialisation time, divided by the number of node problems solved. This also allows for studying larger instances which can take a long time to solve. Experiments are conducted for instances ranging from 10 to 35 locations in increments of 5 and are subject to a time limit of one hour, excluding the initialisation time.

In Table 3 the average times on node problems for the labelling approach with different pools of operations are shown. The mixed integer programming approach is omitted as it was consistently outperformed by the labelling approach. As an example, the latter succeeded in solving all instances with 15 and 20 locations, whereas a 15 location instance was rarely solved with the mixed integer programming approach.

Table 3: Average time, computed as the total time including initialisation time divided by the number of node problems solved, and size of the operation pool for the labelling algorithm used for BCP, under different truck node limit restrictions on the initial operations, averaged over 10 instances.

| | **Truck node limit:** | | | | | |
|---|---|---|---|---|---|---|
| | ***0*** | | ***1*** | | ***2*** | |
| **Locations** | time | pool | time | pool | time | pool |
| 10 | 0 | 769 | 0 | 2,117 | 0 | 2,909 |
| 15 | 0 | 2,307 | 0 | 9,994 | 0 | 16,317 |
| 20 | 1 | 5,114 | 1 | 29,514 | 1 | 60,359 |
| 25 | 5 | 9,643 | 5 | 67,190 | 4 | 162,728 |
| 30 | 46 | 16,281 | 38 | 134,096 | 33 | 380,917 |
| 35 | 5:22 | 25,450 | 4:39 | 242,673 | 4:30 | 796,650 |

According to the reported average times in the table, adding operations with more truck nodes to the initial set of operations is beneficial as the average times are the smallest for a truck node limit of 2, despite the extra time spent on constructing the sufficient subset. In addition, the average size of the initial set of operations provides

a sense for the trade-off that exists between increasing memory requirements versus realising greater efficiency. For example, with 30 locations the average time can be decreased from 38 to 33 seconds, but the initial pool of operations almost triples in size. Besides, it is worth noting that solving instances beyond 30 locations seems to become challenging for branch-cut-and-price as the average solve time for node problems starts to increase rapidly.

Altogether, it is decided to configure BCP with the labelling algorithm and an initial operation pool with all operations having no more than 2 truck nodes. Larger pools may require substantial memory, while only slight performance increases are expected.

Now that the implementation is finalised the three methods can be compared. Results will be provided for instances ranging from 20 up to 34 locations in increments of 2. Experiments that run out of memory or take longer than 3 hours during the initialisation phase are excluded. Results on the performance of the methods are provided in Table 4.

Table 4: Comparison of the exact methods measured on 10 instances showing, respectively, the number of instances solved to optimality, the number of completed solves and the average solve time of completed solves.

| | **Method:** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | ***BCP*** | | | ***PBC*** | | | ***FBC*** | | |
| **Locations** | opt. | tot. | time* | opt. | tot. | time* | opt. | tot. | time* |
| 20 | 10 | 10 | 27:17 | 10 | 10 | 02:43 | 10 | 10 | 2:33 |
| 22 | 9 | 10 | 1:49:01 | 10 | 10 | 13:41 | 10 | 10 | 5:27 |
| 24 | 8 | 10 | 3:20:46 | 10 | 10 | 1:06:15 | 10 | 10 | 12:46 |
| 26 | 2 | 10 | 10:32:22 | 5 | 10 | 7:26:50 | 7 | 7 | 45:31 |
| 28 | 1 | 10 | 11:37:57 | 5 | 10 | 7:36:18 | 8 | 8 | 1:17:58 |
| 30 | 0 | 10 | 12:10:47 | 2 | 9 | 10:41:17 | 3 | 3 | 1:33:23 |
| 32 | 0 | 10 | 12:16:20 | 0 | 8 | 12:41:17 | 6 | 6 | 5:03:21 |
| 34 | 0 | 10 | 12:25:00 | 2 | 8 | 11:13:21 | 2 | 2 | 5:04:08 |

* Averages exclude instances that failed to complete due to an out of memory error or exceedance of the initialisation time limit.

First, in Table 4 it is observed that PBC and FBC ran into problems while solving some of the instances. By inspection of the experiments it turns out that an exceedance of the initialisation time limit was the cause for the issues for PBC, whereas for FBC out of memory errors frequently occurred as well. Nevertheless, all methods perform relatively well as optimal solutions were found for a reasonable number of instances. The fact that all methods perform fairly well suggests that the decision to branch on combined nodes accounts largely to the success of the methods. Further inspection of the results also shows that among all instances solved by PBC only two experiments

required an operation to be branched on. In other words, branching on operations appears on a rare occasion.

Second of all, the average solving times are reported in Table 4. For the instances with 22, 24 and 26 locations a comparison of the methods shows that PBC significantly outperforms BCP. This suggests that it never occurs that the pricing algorithm beats a construction of the entire sufficient subset, however, it should be noted that this result could have been different in absence of the properties based on equivalence that were introduced. Recalling from Table 1, beyond 20 locations it becomes increasingly difficult to create a sufficient subset in absence of the properties based on equivalence. In this case BCP could be more efficient than PBC. On the other hand, FBC, which uses a black box branch-and-cut solver, in turn significantly outperforms PBC in terms of computation time.

In Table 5 additional results are provided which allows for a better comparison between the methods. More specifically, the solution quality for larger problems can be compared. The statistics are based on the problems that were solved successfully, as described in Table 4.

Table 5: Statistics for the quality of the final solution, deduced from successful solves* and showing, respectively, the average and maximum percentage gap between the final solution and the best obtained bound and the average percentage savings of the final solution relative to the initial solution.

| | **Method:** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | ***BCP*** | | | ***PBC*** | | | ***FBC*** | | |
| | % gap | | % save | % gap | | % save | % gap | | % save |
| **Locations** | avg. | max. | avg. | avg. | max. | avg. | avg. | max. | avg. |
| 20 | 0 | 0 | 1.9 | 0 | 0 | 1.9 | 0 | 0 | 1.9 |
| 22 | 5 | 4.7 | 1.5 | 0 | 0 | 1.6 | 0 | 0 | 1.6 |
| 24 | 2.2 | 21.0 | 1.3 | 0 | 0 | 1.8 | 0 | 0 | 1.8 |
| 26 | 4.9 | 24.4 | 0.9 | 2.0 | 14.7 | 1.6 | 0 | 0 | 2.2 |
| 28 | 7.3 | 25.3 | 0.8 | 3.5 | 16.2 | 1.8 | 0 | 0 | 2.7 |
| 30 | 11.2 | 33.4 | 0.2 | 4.1 | 11.9 | 1.7 | 0 | 0 | 4.4 |
| 32 | 16.0 | 37.2 | 0.0 | 7.4 | 11.8 | 1.5 | 0 | 0 | 3.1 |
| 34 | 12.8 | 38.1 | 0.0 | 5.5 | 12.0 | 0.2 | 0 | 0 | 1.5 |

* Averages and maxima exclude instances that failed to complete due to an out of memory error or exceedance of the initialisation time limit.

Table 5 shows averages and maxima for the percentage gap between the final solution and the best obtained bound. The most remarkable result is that FBC found optimal solutions to all instances for which it did not run into issues. In addition, the relative savings in costs of the final solution over the initial solution from the TSP-ep-

all heuristic are shown. The results suggest that this heuristic from Agatz et al. (2016) performs fairly well as the improvements are rather small.

For the comparison between the methods, consider Table 4 and Table 5 jointly. It is clear that BCP is consistently outperformed by the branch-and-cut methods PBC and FBC. Not only are the optimality gaps relatively large but the savings on initial solutions with 32 and 34 locations show that BCP did not succeed to find a better solution. On the other hand, the ranking of the methods PBC and FBC is less apparent when considering the solution quality. It appears that, based on the number of optimal solutions found, FBC performs better on instances that are tractable for the method, whereas PBC is the more robust choice according to the fact that issues occurred less frequently.

From another point of view, the results suggest that PBC can be improved to match the performance of FBC without compromising the robustness. It remains unclear how this can be achieved but it is probable that a noticeable difference can be made with a generally more efficient branch-and-bound procedure or by including an effective family of valid cuts.

### 5.3.3 *Local Search Extension*

Finally, the methods are extended with local search to solve instances with 30, 50 and 70 locations but first a grid search is performed to decide on different parameters. In the initialisation step maxima of 2, 5 and 10 million created labels are considered. For the local search routine a maximum set of 500, 1000 or 2000 operations is maintained and the threshold on the number of updates for this set that triggers the routine is considered to be 5%, 10% or 20% of its maximum size. Under these settings the performance of PBC-LS and FBC-LS are studied by solving instances with 70 locations for one hour, excluding initialisation time.

For the sake of brevity, the results of the grid search will not be discussed in detail. A remarkable result is that the potential memory issues for the FBC-LS method appear to be problematic as more than half of the experiments ran into memory issues, therefore, the method will be omitted from future experiments. For PBC-LS the improvement over the initial solution, averaged over 10 instances, ranged from 0.8% to 2.2% while 22 out of 27 configurations showed a small improvement below 1.5% averaged over 10 instances. The best performance was obtained with 5 million labels, a maximum subproblem size of 500 operations and a trigger set at 20% of this maximum. Moreover, the configuration seems to be plausible as the average time to solve a node subproblem was only 33 seconds while the number of initial operations was on average below 1.7 million.

Results corresponding to the above configuration for the PBC-LS method are provided in Table 6. In addition, the results of PBC for the instances with 30 locations are provided for comparison.

Table 6: Percentage saving of the final solution over the initial solution for PBC-LS considering different instance sizes and PBC for instances with 30 locations in parenthesis for comparison.

| | **Instance:** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Locations** | ***1*** | ***2*** | ***3*** | ***4*** | ***5*** | ***6*** | ***7*** | ***8*** | ***9*** | ***10*** |
| 30 | 3.8 | 4.9 | 4.7 | 3.1 | 0.6 | 0 | 2.7 | 0.4† | 4.6 | 0.8 |
| | (0) | (4.9) | (0.6) | (3.1) | (0.6) | (0) | (2.1) | (0) | (4.4) | (0) |
| 50 | 4.6† | 2.5 | 1† | 3.2 | 1.3 | 3.7 | 1.2 | 1.9 | 1.1 | 2.8 |
| 70 | 2.9† | 4.4† | 3.2† | 2.6 | 6.0 | 1.4† | 3† | 0.2 | 2.1† | 4.7 |

† The experiment terminated early due to running out of memory.

Comparing the final solution of PBC-LS and PBC for the instances with 30 locations provides evidence for the effectiveness of the local search procedure as PBC-LS never found a worse solution. Furthermore, significant improvements were found over the initial TSP-ep-all solution for the instances with 50 and 70 locations which are on average 2.3% and 3.1%, respectively. This suggests that PBC-LS is a reliable method to find solutions of good quality. On the other hand, a number of experiments ran unexpectedly into memory issues. A potential explanation for this could be that the number of initial operations is too large as on average 1,298,280 and 1,673,486 initial operations were created for the instances with 50 and 70 locations. Consequently, the observed memory issues indicate that more consideration has to be put into choosing the limits for the number of rows and columns in the RMP.

Experiments with BCP-LS were also conducted but the method remains to be inferior. For the instances with 30 locations the effort put into generating additional operations could not be justified by the results and the pricing algorithm hardly found any additional operations with negative reduced costs for instances with 50 and 70 locations.

## 6 CONCLUSION

Methods have been studied that aim to find an optimal solution to the travelling salesman problem with drone minimising the execution time. A framework was introduced for the concept of operations introduced in Agatz et al. (2016) that could be used to refine the solution space without compromising the potential solution value. In addition, the binary programming formulation from the same paper was improved, and a branch-cut-and-price algorithm was proposed to deal with larger problems. Moreover, a mixed integer programming formulation and a labelling algorithm were used to solve the pricing problem.

Due to the contributions of this paper a branch-and-cut method was able to solve problems with up to 35 nodes. Despite, the column generation framework did not

appear to add any value to solving the problem in an exact nor heuristic way. Further research has to be conducted to study whether this framework can be implemented more effectively. Branch-and-cut shows to be suited for finding high quality solutions to large problems for which heuristics leave room for improvement. Experiments have shown that the operational costs decreased by several percentage points for large instances with 50 and 70 nodes, excluding the depot.

The main conclusion that branch-cut-and-price is ineffective has to be interpreted with care as multiple improvements were proposed alongside in this paper. It is probable that a different conclusion would have been drawn in the absence of some of these. In addition, a commercial branch-and-cut solver appeared to be more successful in solving instances of reasonable size. This provides the promising perspective that the methods can still be significantly improved.

For future research that wants to improve the methods introduced in this paper the following research directions are proposed. Firstly, it may be studied whether bi-directionality can be applied to improve the pricing algorithm. Next, including effective families of cuts seems to be a promising way to make branch-and-bound more efficient. Lastly, the methods may be parametrised more carefully to allow for a more efficient and robust implementation.

## REFERENCES

Agatz, N., P. Bouman, and M. Schmidt (2016). Optimization approaches for the traveling salesman problem with drone. *ERIM report series research in management Erasmus Research Institute of Management*, urn:issn:1566–5283.

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

Bouman, P., N. Agatz, and M. Schmidt (2017). Dynamic programming approaches for the traveling salesman problem with drone. *ERIM report series research in management Erasmus Research Institute of Management*, urn:issn:1566–5283.

Carlsson, J.G. and S. Song (2017). Coordinated logistics with a truck and a drone. *Management Science*.

Daknama, R. and E. Kraus (2017). Vehicle routing with drones. *CoRR abs/1705.06431*.

Dorling, K., J. Heinrichs, G.G. Messier, and S. Magierowski (2017). Vehicle routing problems for drone delivery. *Systems, Man, and Cybernetics: Systems, IEEE Transactions on 47*(1), 70–85.

Ferrandez, S.M., T. Harbison, T. Webwer, R. Sturges, and R. Rich (2016). Optimization of a truck-drone in tandem delivery network using k-means and genetic algorithm. *Journal of Industrial Engineering and Management 9*(2).

Fukasawa, R., H. Longo, J. Lysgaard, M. P. de Aragão, M. Reis, E. Uchoa, and R.F. Werneck (2006). Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming 106*(3), 491–511.

Garone, E., R. Naldi, A. Casavola, and E. Frazzoli (2010). Cooperative mission planning for a class of carrier-vehicle systems. pp. 1354–1359. IEEE Publishing.

Ha, Q.M., Y. Deville, D. Pham, and M.H. Hà (2015). Heuristic methods for the traveling salesman problem with drone.

Ha, Q.M., Y. Deville, Q. Dung Pham, and M.H. Hà (2018). On the min-cost traveling salesman problem with drone. *Transportation Research Part C: Emerging Technologies 86*, 597–621.

Irnich, S. and G. Desaulniers (2005). *Shortest Path Problems with Resource Constraints*, pp. 33–65. Boston, MA: Springer US.

Mathew, N., S.L. Smith, and S.L. Waslander (2015). Planning paths for package delivery in heterogeneous multirobot teams. *Automation Science and Engineering, IEEE Transactions on 12*(4), 1298–1308.

Murray, C.C. and A.G. Chu (2015). The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. *Transportation Research Part C 54*, 86–109.

Othman, bin M.S., A. Shurbevski, Y. Karuno, and N. Nagamochi (2017). Routing of carrier-vehicle systems with dedicated last-stretch delivery vehicle and fixed carrier route. *Journal of Information Processing 25*, 655–666.

Padberg, M. and G. Rinaldi (1991). A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review 33*(1), 60–100.

Poikonen, S., X. Wang, and B. Golden (2017). The vehicle routing problem with drones: Extended models and connections. *Networks 70*(1).

Ponza, A. (2016). Optimization of drone-assisted parcel delivery.

Shivdas, S. (2018). Boeing unveils prototype for unmanned electric cargo air vehicle. `https://uk.reuters.com/article/us-boeing-electric-cav/boeing-unveils-prototype-for-unmanned-electric-cargo-air-vehicle-idUKKBN1EZ2EF`. Accessed February 10, 2018.

Wang, X., S. Poikonen, and B. Golden (2016). The vehicle routing problem with drones: several worst-case results. *Optimization Letters 11*(4).

# APPENDIX A

Here, details are provided on the estimation of $\rho$, the maximum possible number of nodes on the truck path among operations that are compact with respect to the drone return. The estimate is used in the labelling algorithm in Section 4 to discard labels early on. A mixed integer programming formulation is provided that approximates $\rho$ and is intended to be fast. More accurate estimates can be found by including additional constraints but this comes at the expensive of slower convergence towards the optimal solution.

As before, let $a_{ij}$ be binary decisions for the truck arcs and $z_{ij}$ for the joint decision on the start and end node. In addition, let $\tau_i$ be auxiliary variables for the subtour elimination constraints. Furthermore, consider the parameters $c_{ij}^{max} = \max_{d \in V \setminus \{v_0\}} \{c_{id}^{d} + c_{dj}^{d}\}$ which represent the maximum possible travel time for the drone given the start node $i$ and end node $j$. In Formulation 4 an integer programming formulation is provided. In essence, it loosely requires operations to be compact with respect to the drone return.

$$\text{Maximise} \quad \rho = 2 + \sum_{i \in V} \sum_{j \in V} a_{ij} \tag{49}$$

Subject to

$$\begin{aligned}
\textstyle\sum_{j \in V} a_{ij} &\leqslant 1, && \forall i \in V && (50)\\
\textstyle\sum_{i \in V} \sum_{j \in V} z_{ij} &\leqslant 1 && && (51)\\
\textstyle\sum_{j \in V} a_{j0} + \sum_{j \in V} a_{0j} &\leqslant 1 + z_{00} && && (52)\\
\textstyle\sum_{j \in V} a_{ji} - \sum_{j \in V} a_{ij} &= \textstyle\sum_{j \in V} z_{ji} - \sum_{j \in V} z_{ij}, && \forall i \in V && (53)\\
\tau_i - \tau_j + n a_{ij} &\leqslant n - 1, && \forall i, j \in V && (54)\\
\textstyle\sum_{i \in V} \sum_{j \in V} c_{ij} a_{ij} &\leqslant \textstyle\sum_{i \in V} \sum_{j \in V} c_{ij}^{max} z_{ij} && && (55)\\
\\
a_{ii} &= 0, && \forall i \in V && (56)\\
a_{ij}, z_{ij} &\in \{0, 1\}, && \forall i, j \in V && (57)\\
\tau_i &\geqslant 0, && \forall i \in V && (58)
\end{aligned}$$

Formulation 4: Estimation of the maximum number of nodes on the truck path.

Beginning with the constraints, (50)-(53) represent the decisions on the nodes. Here, (50) describes that a node may be visited only once, (51) states that a start and end node can only be chosen once, and (52) states that the depot can only be a start or

end node. Next, constraints (53) construct a path covering the selected nodes and (54) ensure that the path is connected. Constraint (55) is a relaxation of the condition that the operation is compact with respect to the drone return. Instead of requiring a corresponding drone node to be chosen explicitly, it is assumed that the selected drone node provides the longest possible travel time for the drone.

The objective (49) maximises the number of arcs on the truck path. By (55), compactness with respect to the drone return is guaranteed to be violated if this path is extended with an additional arc, therefore, one is added to the objective value. Moreover, this number is corrected by adding one again to obtain the number of nodes, which shows that $\rho$ in (49) provides an upper bound on the maximum number of nodes on the truck path. Besides, the linear relaxation of Formulation 4 also provides an upper bound which may be rounded down. Therefore, an estimate is always available and solving Formulation 4 can be terminated if the solution converges too slowly.