The Merchant Subtour Problem*

Bram Verweij and Karen Aardal

October 25, 2000

Abstract

We consider the problem of a travelling merchant that makes money by buying commodities where they are cheap and selling them in other places where he can make a profit. The merchant ships commodities of his own choice in a van of fixed capacity. Given the prices of all the commodities in all of the places, and the cost of driving from one place to another, the problem the merchant faces each day is to select a subset of the cities that he can visit in a day, and that allows him to maximise the profit he makes. We call this problem the *Merchant Subtour Problem*. The MSP models the pricing problem of a rather complex pickup and delivery problem that was given to us by the Dutch logistics company Van Gend & Loos.

We show that a special case of the MSP has a totally unimodular constraint matrix. This knowledge enables us to develop a tabu-search algorithm for finding good feasible solutions to the MSP, and a branch-and-price-and-cut algorithm for solving the MSP to optimality. The relaxations solved in each node of the branch-and-bound tree are strengthened by lifted knapsack inequalities, lifted cycle inequalities and mod-k cuts. We present computational results on data sets derived from our main instance of the Van Gend & Loos pickup and delivery problem.

1 Introduction

A *merchant* is a person who travels around in a vehicle of fixed capacity, and makes a living by buying goods at places where they are cheap and selling them at places where he can make a profit. A *merchant subtour* is a directed closed walk of a merchant starting and ending at a given place, together with a description of the load of the vehicle along each traversed arc. We assume the merchant has to pay a cost that is linear in the distance he travels. This paper addresses the problem of finding a minimum total cost merchant subtour where the total cost of a merchant subtour equals the travel cost minus the profit made by trading along the way. We refer to the problem as the Merchant Subtour Problem (MSP).

Our interest in the MSP was raised because it models the pricing problem of a certain complex pickup and delivery problem occurring at the Dutch logistics company Van Gend & Loos, as reported on by Verweij [36]. Based on the work presented in this paper we are able to find approximate solutions of provable quality for small to medium size instances of this complicated pickup and delivery problem. A similar motivation is given by Bauer, Linderoth and Savelsbergh [8] for studying a certain circuit problem. In addition, the MSP is related to several other problems that are reported on in the literature. Also, its nice structure makes it an interesting problem to study in its own right.

*This research was partially supported by EC - Fifth Framework Programme contract IST-1999-14186 (Project ALCOM-FT: Algorithms and Complexity — Future Technologies).

1.1 Problem Definition

Here we formalise the MSP. We are given a directed graph $G = (V, A)$, where the set of nodes V corresponds to a set of *distribution centres*, and where $A = \{(u, v) \mid u, v \in V\}$ is a set of directed arcs (including self-loops). Furthermore we are given a set of *commodities* K, vectors $\boldsymbol{d} \in \mathbb{N}^K$, $\boldsymbol{t} \in \mathbb{N}^A$, $\boldsymbol{c} = (\boldsymbol{c}_A, \boldsymbol{c}_K) \in \mathbb{N}^A \times \mathbb{Z}^K$, and numbers $D, T \in \mathbb{N}$. With every commodity $k \in K$, a unique ordered pair $(u, v) \in V \times V$ is associated, $u \neq v$. Here, u is the *source* and v is the *destination* of the commodity. We will sometimes use $(u, v) \in K$ to denote a commodity with source u and destination v (it should be clear from the context whether an ordered pair of nodes represents an arc or a commodity). Further, $\boldsymbol{d}$ is the *demand* vector, i.e., for each $k \in K$ the maximum amount of commodity k that can be shipped is d_k. The *capacity* of our vehicle is D. Note that it is not required that the merchant ships all demand. The travel times are given in the vector $\boldsymbol{t}$, i.e., for each $a \in A$, t_a is the time it takes to traverse arc a. The maximum amount of time we are allowed to use in total is T, and $\boldsymbol{c}$ is the cost vector, i.e., for each $a \in A$ the cost of traversing a is c_a. For each $k \in K$ the profit made by shipping k is $-c_k$ per unit. We assume that for all self-loops $a = (v, v) \in A$, $c_a = t_a = 0$ and that $-\boldsymbol{c}_K \geq \boldsymbol{0}$. Finally, there is a special node $s \in V$, called the *depot*, that is the starting location of our merchant. We will make one assumption:

Assumption 1.1. *The subtour traversed by the merchant is a directed cycle.*

We can use a network transformation called node duplication (see Ahuja, Magnanti, and Orlin [2]) and slightly generalise the demands in such a way that they are associated with sets of nodes to allow for the situation in which every node of the original problem can be visited a number of times. In the presence of a finite time limit and strictly positive travel times, this is sufficient to allow any possible closed walk to appear as part of a solution. However, we will see that the MSP is $\mathcal{NP}$-hard. This means that from a computational point of view node duplication is not very attractive, as it blows up the dimensions of the problem. Assumption 1.1 does allow us to make the following definition:

Definition 1.1. A *merchant subtour* is a tuple $(C, \boldsymbol{\ell})$, where C is a directed cycle in G that contains s, and where $\boldsymbol{\ell} \in \mathbb{N}^K$.

Let $(C, \boldsymbol{\ell})$ be a merchant subtour. The directed cycle C indicates the subtour traversed by the merchant, and $\boldsymbol{\ell}$ indicates the load shipped by the merchant. We can write $C = (s = v_0, a_1, v_1, \ldots, a_n, v_n = s)$ where $a_i = (v_{i-1}, v_i) \in A$. We denote the nodes (arcs) of a directed cycle C by $V(C)$ (and $A(C)$, respectively), so with C as above $V(C) = \{v_1, \ldots, v_n\}$ and $A(C) = \{a_1, \ldots, a_n\}$. Figure 1 depicts a merchant subtour.

Definition 1.2. A merchant subtour $(C, \boldsymbol{\ell})$ with C as above is *feasible* if

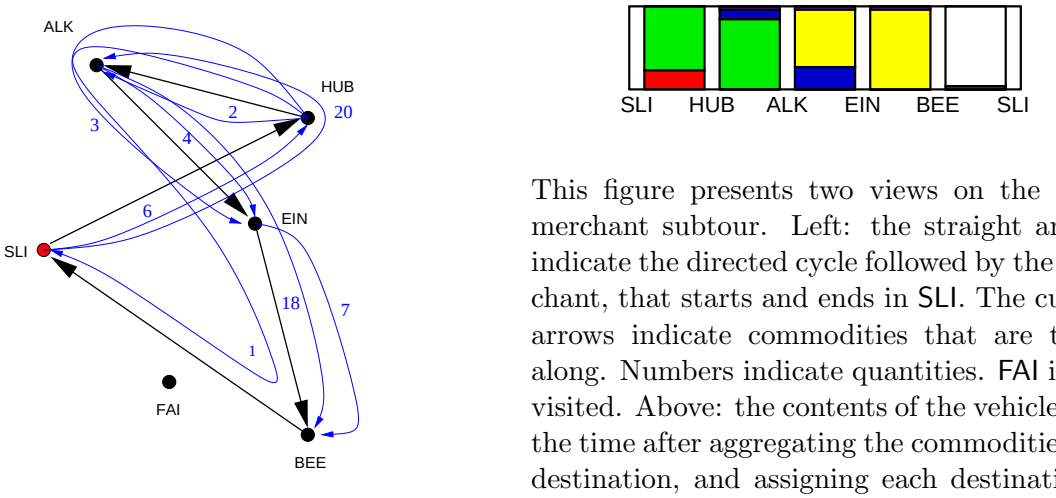
(i) for each $k \in K$ with $\ell_k > 0$ we have $k = (v_i, v_j)$ for some $v_i, v_j \in V(C)$ with $i < j$,

(ii) for each arc $(v_{i-1}, v_i) \in A(C)$ we have that

$$\boldsymbol{\ell}(\{(v_{j_1}, v_{j_2}) \in K \mid j_1 \in \{1, \ldots, i-1\}, j_2 \in \{i, \ldots, n\}\}) \leq D,$$

(iii) $\boldsymbol{\ell} \leq \boldsymbol{d}$, and

(iv) $\boldsymbol{t}(A(C)) \leq T$.



This figure presents two views on the same merchant subtour. Left: the straight arrows indicate the directed cycle followed by the merchant, that starts and ends in SLI. The curved arrows indicate commodities that are taken along. Numbers indicate quantities. FAI is not visited. Above: the contents of the vehicle over the time after aggregating the commodities per destination, and assigning each destination a different shade.

Figure 1: A Merchant Subtour

Here (i) enforces that commodities are only moved in the direction of the cycle, (ii) enforces that the total amount of commodities that traverses any arc a in the directed cycle C is at most the capacity of our vehicle, (iii) enforces that we do not exceed the demand, and (iv) enforces that the maximum travel time is not exceeded.

1.2 Related Literature

This paper is a based upon parts of Verweij's Ph.D. thesis [36, Chapter 6]. Since a merchant subtour is a tour through a directed graph, the MSP to the asymmetric travelling salesman problem reported on, among others, by Grötschel and Padberg [18] and by Balas and Fischetti [6]. Moreover, a merchant subtour does not have to visit all nodes of the graph, which makes it related to the prize collecting travelling salesman problem, reported on by Balas [4, 5], and the weighted girth problem, reported on by Bauer [7] and by Bauer, Linderoth, and Savelsbergh [8]. The MSP differs from the latter two problems in that we have one given node (namely, the depot) that has to be included in the subtour, and in that the cost of a tour is not a linear function of the arcs in the tour, but also incorporates a term that depends on the demand that is shipped.

2 Formulations and Complexity

2.1 The Loading Problem

Let

$$C = (s = v_0, (v_0, v_1), v_1, \ldots, (v_{n-1}, v_n), v_n = s) \tag{1}$$

be a directed cycle in G. In this section we discuss the MSP restricted to the case that the graph $G = (V, A)$ is the directed cycle C, i.e., $V = V(C)$ and $A = A(C)$. In this case, we know the route of the vehicle, and only have to decide on its load. We call such

a restricted MSP a *loading problem*. To avoid confusion, we will refer to the original MSP where G can be an arbitrary directed graph as the *unrestricted* MSP. We will introduce an integer programming formulation that describes the loading problem, and prove that its constraint matrix is totally unimodular (see e.g. Schrijver [34]). This result is fundamental for the algorithms in the remainder of this paper, as it allows us to generalise local search neighbourhoods for the travelling salesman problem to the unrestricted MSP, and it allows us to give a good integer programming formulation for the unrestricted MSP.

For $(v_i, v_j) \in K$, let $\mathcal{P}^C_{(v_i,v_j)}$ be the set of all paths from v_i to v_j in G that do not have s as internal node. Due to the numbering of the nodes induced by C, and due to the paths being simple, we have that

$$\mathcal{P}^C_{(v_i,v_j)} = \begin{cases} \{(v_i, (v_i, v_{i+1}), v_{i+1}, \ldots, (v_{j-1}, v_j), v_j)\} & \text{if } i < j, \\ \varnothing, & \text{otherwise.} \end{cases}$$

Let $\mathcal{P}^C = \bigcup_{k \in K} \mathcal{P}^C_k$ be the set of all paths that connect the source and the sink of some commodity ($\mathcal{P}^C$ contains all paths with demand).

We use the decision variables $\boldsymbol{f} \in \mathbb{R}^{\mathcal{P}^C}$, where for each path $P \in \mathcal{P}^C$ with P a path from u to v, f_P indicates the amount of commodity (u, v) we ship. The loading problem can now be stated as the following integer programming problem:

$$\begin{aligned}
z^*_{\text{load}}(C) = \min \quad & \boldsymbol{c}_A(A(C)) + \sum_{(u \rightsquigarrow v) \in \mathcal{P}^C} (\boldsymbol{c}_K)_{(u,v)} f_{(u \rightsquigarrow v)} && && \text{(2a)} \\
\text{subject to} \quad & \boldsymbol{f}(\{P \in \mathcal{P}^C \mid A(P) \ni a\}) \leq D && \forall a \in A, && \text{(2b)} \\
& f_{(u \rightsquigarrow v)} \leq d_{(u,v)} && \forall (u \rightsquigarrow v) \in \mathcal{P}^C, && \text{(2c)} \\
& -\boldsymbol{f} \leq \boldsymbol{0}, \text{ integer.} && && \text{(2d)}
\end{aligned}$$

Here, inequalities (2b) enforce that the capacity of the vehicle is not exceeded, and inequalities (2c) enforce that we do not exceed the demand. Observe that if $\boldsymbol{f}$ is a feasible solution to model (2), then $(C, \boldsymbol{\ell})$ with $\boldsymbol{\ell}_{(u,v)} = f_{(u \rightsquigarrow v)}$ if $(u \rightsquigarrow v) \in \mathcal{P}^C$ and $\boldsymbol{\ell}_{(u,v)} = 0$ otherwise is a feasible merchant subtour of the same value.

Lemma 2.1. *The constraint matrix of model (2) is totally unimodular.*

Proof. Let $M \in \{0,1\}^{\mathcal{P}^C \times A}$ be the transpose of the constraint matrix of inequalities (2b), i.e.,

$$M = \left(\chi^{A(P)} \mid P \in \mathcal{P}^C\right)^T.$$

We claim that M is totally unimodular. From this claim the lemma follows because with M, also the matrix

$$\begin{pmatrix} M^T \\ I \\ -I \end{pmatrix}$$

is totally unimodular, which is the constraint matrix of model (2).

It remains to prove our claim. Order the arcs in A in the order in which they occur on the directed cycle (1), i.e., such that $a_i = (v_{i-1}, v_i)$ and $s = v_0 = v_{|A|}$. Note that for any path $P \in \mathcal{P}^C$ the arcs in $A(P)$ occur consecutively in the ordering of A. Hence M is an interval matrix as defined by Nemhauser and Wolsey [28, Part III, Section 2, Definition 2.2].

It follows directly that M is totally unimodular (see e.g. Nemhauser and Wolsey [28, Part III, Section 2, Corollary 2.10], or Schrijver [34, Section 19.2, Example 7]).

□

As a consequence, any basic solution to the LP relaxation of model (2) will be integral. Because LP relaxations can be solved in polynomial time [17], and the dimensions of model (2) are polynomial in $|V|$, it follows directly from Lemma 2.1 that we can solve the loading problem in polynomial time. Moreover, if we have an (optimal) fractional solution $\boldsymbol{f}$ to model (2), we can write it as a convex combination of (optimal) integer solutions to model (2). This implies that within the context of the unrestricted MSP, we do not need to enforce integrality conditions on the flow of commodities, even when these are required to be integral. Indeed, given a merchant subtour that ships commodities in fractional amounts, we can always find a merchant subtour with the same value that is integral by solving the loading problem on the induced directed cycle of the merchant subtour. After relaxing the integrality conditions on the load shipped, we may assume that the vehicle capacity is $D = 1$, and that all demand is expressed in (fractions of) vehicle loads.

Assumption 2.2. *The vehicle capacity D equals 1.*

This assumption is non-restrictive, as we can always scale an MSP instance by multiplying the demand vector $\boldsymbol{d}$ with a factor $1/D$ and multiplying the cost vector $\boldsymbol{c}_K$ by a factor D without changing the value of the optimal solution. Note that for such a scaled problem the demand vector then is an element from $\frac{1}{D}\mathbb{N}^K$ instead of from $\mathbb{N}^K$, and that feasible merchant subtours are of the form $(C, \boldsymbol{\ell})$ with $\boldsymbol{\ell} \in \frac{1}{D}\mathbb{N}^K$ (we also "scale" Definition 1.1). In the remainder of this paper, we assume we are working with a scaled problem unless mentioned otherwise.

2.2 An Integer Programming Formulation for the MSP

In this section we introduce an integer programming formulation for the MSP. For each commodity $(u, v) \in K$, let $\mathcal{P}_{(u,v)}$ denote the set of all possible paths in G from u to v that do not have s as an internal node, and let $\mathcal{P}$ denote the set of all such paths in G connecting the source and destination of some commodity (i.e., $\mathcal{P} = \bigcup_{k \in K} \mathcal{P}_k$). We use the decision variables $\boldsymbol{x} \in \{0,1\}^A$, $\boldsymbol{f} \in \mathbb{R}_+^{\mathcal{P}}$, and $\boldsymbol{z} \in \mathbb{R}_+^K$, where $x_a = 1$ when we use arc a and $x_a = 0$ otherwise. For a path $P = (u \rightsquigarrow v) \in \mathcal{P}$ the variable $\boldsymbol{f}_P$ indicates the amount of commodity (u, v) we ship via path P, and z_k indicates the total amount of commodity k that we ship.

The following formulation is obtained by extending the formulation of the prize collecting travelling salesman problem by Balas [4] with the $\boldsymbol{f}$- and $\boldsymbol{z}$-variables to model the loading

dimension of our problem:

$$\begin{aligned}
\min \quad & z_{\mathrm{MSP}}(\boldsymbol{x}, \boldsymbol{f}, \boldsymbol{z}) = \boldsymbol{c}_A^T \boldsymbol{x} + \boldsymbol{c}_K^T \boldsymbol{z} && && \text{(3a)}\\
\text{subject to} \quad & \boldsymbol{x}(\delta^-(v)) + x_{(v,v)} = 1 && \forall v \in V, && \text{(3b)}\\
& \boldsymbol{x}(\delta^+(v)) + x_{(v,v)} = 1 && \forall v \in V, && \text{(3c)}\\
& \boldsymbol{x}(A(S) \setminus \{(v,v)\}) \leq |S| - 1 && \forall v \in S \subseteq V \setminus \{s\} && \text{(3d)}\\
& \boldsymbol{t}^T \boldsymbol{x} \leq T, && && \text{(3e)}\\
& \boldsymbol{f}(\mathcal{P}_k) = z_k && \forall k \in K, && \text{(3f)}\\
& \boldsymbol{f}(\{P \in \mathcal{P} \mid A(P) \ni a\}) \leq x_a && \forall a \in A, && \text{(3g)}\\
& \boldsymbol{z} \leq \boldsymbol{d}, && && \text{(3h)}\\
& \boldsymbol{x} \in \{0,1\}^A, \boldsymbol{f} \in \mathbb{R}_+^{\mathcal{P}}, \boldsymbol{z} \in \mathbb{R}_+^K. && && \text{(3i)}
\end{aligned}$$

Here, equalities (3b) and (3c) are called the *in-* and *out-degree constraints*, respectively. The in- and out-degree constraints enforce that for each node $v \in V$ we either use the self-loop (v, v), or one arc entering and one arc leaving v. Inequalities (3d) are the *subtour elimination constraints*. They make solutions containing a subtour that does not include node s infeasible. Inequality (3e) enforces that we do not exceed the time limit. Equalities (3f) link the $\boldsymbol{z}$-variables with the $\boldsymbol{f}$-variables. Inequalities (3g) are called the *capacity* constraints, and enforce that we only allow flow of commodities on arcs that are in the subtour as indicated by the $\boldsymbol{x}$-variables. Finally, inequalities (3h) enforce that we do not exceed the demand. Note that the $\boldsymbol{z}$-variables are redundant and can be eliminated from the model using equality (3f).

The following theorem shows that (3) indeed models the MSP correctly.

Theorem 2.3. *Let $(\boldsymbol{x}, \boldsymbol{f}, \boldsymbol{z})$ be a basic feasible solution to the LP relaxation of (3) with $\boldsymbol{x} \in \{0,1\}^A$. Then $D\boldsymbol{f} \in \mathbb{N}^{\mathcal{P}}$ for all $D \in \mathbb{N}$ such that $D\boldsymbol{d} \in \mathbb{N}^K$.*

Proof. Choose $D \in \mathbb{N}$ such that $D\boldsymbol{d} \in \mathbb{N}^K$. By substituting (3h) in (3f) we find that $\boldsymbol{f}$ satisfies

$$\begin{aligned}
\boldsymbol{f}(\{P \in \mathcal{P} \mid A(P) \ni a\}) &\leq x_a && \forall a \in A && \text{(4a)}\\
\boldsymbol{f}(\mathcal{P}_k) &\leq d_k && \forall k \in K, && \text{(4b)}\\
\boldsymbol{f} &\geq \boldsymbol{0} && && \text{(4c)}
\end{aligned}$$

Since $\boldsymbol{x} \in \{0,1\}^A$ and $\boldsymbol{x}$ satisfies (3c)–(3d), the set of arcs $A(C) = \mathrm{supp}(\boldsymbol{x})$ induces a directed cycle C in G. It follows that for all paths $P \in \mathcal{P}$, $f_P > 0$ implies $A(P) \subseteq A(C)$, and $A(P) \not\subseteq A(C)$ implies $f_P = 0$. Hence we have $\boldsymbol{f}_{\mathcal{P} \setminus \mathcal{P}^C} = \boldsymbol{0}$, and

$$\begin{aligned}
\boldsymbol{f}(\{P \in \mathcal{P}^C \mid A(P) \ni a\}) &\leq 1 && \forall a \in A(C), && \text{(5a)}\\
\boldsymbol{f}(\mathcal{P}_k^C) &\leq d_k && \forall k \in K, && \text{(5b)}\\
\boldsymbol{f}_{\mathcal{P}^C} &\geq \boldsymbol{0}, && && \text{(5c)}
\end{aligned}$$

where $\mathcal{P}^C, \mathcal{P}_k^C$ are as in Section 2.1 (note that $\mathcal{P}_k^C$ is a singleton set). But then, $\boldsymbol{f}' = D\boldsymbol{f}_{\mathcal{P}^C}$ satisfies

$$\begin{aligned}
\boldsymbol{f}'(\{P \in \mathcal{P}^C \mid A(P) \ni a\}) &\leq D && \forall a \in A(C), && \text{(6a)}\\
f'_{(u \rightsquigarrow v)} &\leq D d_{(u,v)} && \forall (u \rightsquigarrow v)\mathcal{P}^C && \text{(6b)}\\
\boldsymbol{f}' &\geq \boldsymbol{0}. && && \text{(6c)}
\end{aligned}$$

Because $(\boldsymbol{x}, \boldsymbol{f}, \boldsymbol{z})$ is a basic feasible solution to (3), $\boldsymbol{f}'$ is a basic feasible solution to (6). By choice of D and Lemma 2.1 we find that $\boldsymbol{f}'$ is integral. Hence $D\boldsymbol{f}$ is integral. □

Proposition 2.4. *Let $\boldsymbol{x}$ be defined as $x_a = 1$ if $a = (v, v)$ for some $v \in V$, and $x_a = 0$ otherwise. Then, $(\boldsymbol{x}, \boldsymbol{0}, \boldsymbol{0})$ is a feasible solution and $z_{MSP}(\boldsymbol{x}, \boldsymbol{0}, \boldsymbol{0}) = 0$.*

Proof. Recall that $c_{(v,v)} = t_{(v,v)} = 0$ for all $v \in V$. The proof is directly from model (3). □

Proposition 2.4 states that it is feasible for the merchant to stay at the depot. In this situation, the merchant does not have any cost, but neither does he make a profit, so the objective function value is 0. So, if $(\boldsymbol{x}^*, \boldsymbol{f}^*, \boldsymbol{z}^*)$ is the optimal merchant subtour, then $z_{\text{MSP}}(\boldsymbol{x}^*, \boldsymbol{f}^*, \boldsymbol{z}^*) \leq 0$.

2.3 The Computational Complexity of the MSP

In this section we show that the MSP is $\mathcal{NP}$-hard by a reduction from the travelling salesman problem. We first show that for suitably chosen $\boldsymbol{c}_K$, the optimal solution of the MSP maximises the volume of the demand shipped. Using this, given a TSP instance, we define an MSP instance in which the optimal solution contains an optimal travelling salesman tour.

Lemma 2.5. *Let $\bar{z}$ be an upper bound on the travel cost $\boldsymbol{c}_A^T\boldsymbol{x}$ of any subtour $\boldsymbol{x}$. For $k \in K$, let $c_k = -(\bar{z} + 1)$. Then, the optimal solution to the MSP maximises $\sum_{k \in K} z_k = \boldsymbol{z}(K)$.*

Proof. The proof is by contradiction. Let $(\boldsymbol{x}^*, \boldsymbol{f}^*, \boldsymbol{z}^*)$ be an optimal solution to the MSP, and suppose there exists a solution $(\boldsymbol{x}', \boldsymbol{f}', \boldsymbol{z}')$ with $\boldsymbol{z}'(K) > \boldsymbol{z}^*(K)$. Then:

$$\begin{aligned}
z_{\text{MSP}}(\boldsymbol{x}', \boldsymbol{f}', \boldsymbol{z}') - z_{\text{MSP}}(\boldsymbol{x}^*, \boldsymbol{f}^*, \boldsymbol{z}^*) &= \boldsymbol{c}_A^T\boldsymbol{x}' + \boldsymbol{c}_K^T\boldsymbol{z}' - (\boldsymbol{c}_A^T\boldsymbol{x}^* + \boldsymbol{c}_K^T\boldsymbol{z}^*) \\
&\leq \bar{z} + \boldsymbol{c}_K^T\boldsymbol{z}' - (\boldsymbol{c}_A^T\boldsymbol{x}^* + \boldsymbol{c}_K^T\boldsymbol{z}^*) \\
&\leq \bar{z} + \boldsymbol{c}_K^T\boldsymbol{z}' - \boldsymbol{c}_K^T\boldsymbol{z}^* \\
&= \bar{z} + \sum_{k \in K} c_k z'_k - \sum_{k \in K} c_k z^*_k \\
&= \bar{z} - (\bar{z} + 1)(\boldsymbol{z}'(K) - \boldsymbol{z}^*(K)) \\
&< 0.
\end{aligned}$$

Hence $z_{\text{MSP}}(\boldsymbol{x}', \boldsymbol{f}', \boldsymbol{z}') < z_{\text{MSP}}(\boldsymbol{x}^*, \boldsymbol{f}^*, \boldsymbol{z}^*)$. This contradicts with the optimality of the solution $(\boldsymbol{x}^*, \boldsymbol{f}^*, \boldsymbol{z}^*)$, so $(\boldsymbol{x}', \boldsymbol{f}', \boldsymbol{z}')$ does not exist. It follows that $(\boldsymbol{x}^*, \boldsymbol{f}^*, \boldsymbol{z}^*)$ maximises $\boldsymbol{z}(K)$. □

Lemma 2.5 gives us a way to let the merchant visit all cities that occur as the source or the sink of some commodity. Given $G = (V, A)$ and $\boldsymbol{c}_A$, we will use this to construct $K, \boldsymbol{c}_K, \boldsymbol{d}, D, \boldsymbol{t}$, and T, such that the optimal merchant subtour to this MSP instance induces an optimal travelling salesman tour in G (with respect to $\boldsymbol{c}_A$).

Theorem 2.6. *The merchant subtour problem is $\mathcal{NP}$-hard.*

Proof. The proof is by a reduction from the general asymmetric travelling salesman problem, which is known to be $\mathcal{NP}$-hard (see Johnson and Papadimitriou [24]). Let $G = (V, A)$ be a complete directed graph (including self-loops), and let $\boldsymbol{c}_A$ be the cost of the arcs in A (with $c_{(v,v)} = 0$ for all $v \in V$). The tuple $(G, \boldsymbol{c}_A)$ defines an instance of the asymmetric travelling salesman problem.

We define an instance of the MSP as follows. Take G and $\boldsymbol{c}_A$ as above. Choose any node $s \in V$ arbitrarily as depot. Let $K = \{(s, v) \mid v \in V \setminus \{s\}\}$ and for $k \in K$, let $d_k = 1$. So, we have a demand of one unit from the depot to each other node in the graph. Let the vehicle capacity be $D = \boldsymbol{d}(K) = |V| - 1$, the total demand. Define the driving times $\boldsymbol{t} = \boldsymbol{0}$, and let $T = 0$ be the maximum allowed driving time. It follows from the choice of $\boldsymbol{t}$ and T that every subtour in G satisfies the time constraint. Finally, let $\bar{z} = |V| \max_{a \in A} c_a$, and for all $k \in K$ choose $c_k = -(\bar{z} + 1)$ as in Lemma 2.5.

We will show the existence of a solution to this MSP instance that ships all demand. Let $\{v_1, \ldots, v_{|V|}\}$ be an ordering of V such that $s = v_1$. Let

$$C = (v_1, (v_1, v_2), v_2, (v_2, v_3), \ldots, v_{|V|-1}, (v_{|V|-1}, v_{|V|}), v_{|V|}, (v_{|V|}, v_1), v_1)$$

be a tour in G visiting all nodes. Then the vector $(\chi^{A(C)}, \chi^{\mathcal{P}^C}, \chi^K)$, with $\mathcal{P}^C$ defined as in Section 2.1, is a feasible solution to the MSP that ships all demand.

Observe that any merchant subtour consists of at most $|V|$ arcs, so $\bar{z}$ is an upper bound on the travel cost of any merchant subtour. It follows from Lemma 2.5 and the choice of D that the optimal solution to the MSP instance ships all demand (i.e., $z_k^* = 1$ for all $k \in K$). If we choose C in such a way that it follows an optimal tour in G, then

$$(\boldsymbol{x}^*, \boldsymbol{f}^*, \boldsymbol{z}^*) = (\chi^{A(C)}, \chi^{\mathcal{P}^C}, \chi^K)$$

must be an optimal solution to the MSP instance since it ships all demand and minimises $\boldsymbol{c}_A^T \boldsymbol{x}$.

Now let $(\boldsymbol{x}^*, \boldsymbol{f}^*, \boldsymbol{z}^*)$ be any optimal solution to the MSP instance defined above. Because it is optimal it must ship all demand. This implies that the subtour induced by $\text{supp}(\boldsymbol{x}^*)$ visits all nodes of G ($x^*_{(v,v)} = 0$ for all $v \in V$), so $\text{supp}(\boldsymbol{x}^*)$ induces a tour in G. From the optimality of $(\boldsymbol{x}^*, \boldsymbol{f}^*, \boldsymbol{z}^*)$ it follows that $(\boldsymbol{x}^*, \boldsymbol{f}^*, \boldsymbol{z}^*)$ minimises

$$z_{\text{MSP}}(\boldsymbol{x}^*, \boldsymbol{f}^*, \boldsymbol{z}^*) = \boldsymbol{c}_A^T \boldsymbol{x}^* + \boldsymbol{c}_K^T \boldsymbol{z}^* = \boldsymbol{c}_A^T \boldsymbol{x}^* - (\bar{z} + 1)|K|.$$

Because $-(\bar{z} + 1)|K|$ does not depend on $(\boldsymbol{x}^*, \boldsymbol{f}^*, \boldsymbol{z}^*)$, we conclude that $\text{supp}(\boldsymbol{x}^*)$ induces a tour in G of minimum cost, which is the optimal solution to the asymmetric travelling salesman problem instance. □

Resuming, in this section we have introduced the MSP. We have shown that we can solve a special case, called the loading problem, in polynomial time. We have also shown that unless $\mathcal{P} = \mathcal{NP}$, we cannot expect to find polynomial time algorithms that solve the unrestricted MSP to optimality. The remainder of this paper is organised as follows. In Section 3 we give an algorithm that produces feasible solutions to the MSP and runs in pseudo-polynomial time. Section 4 presents an exponential time algorithm that solves the MSP to optimality. We have performed extensive computational experiments to study the behaviour of the algorithms. These experiments are reported on in Section 5.

3 A Tabu Search Heuristic

In this section we present a tabu search algorithm for the MSP. We follow the ideas presented by Herz, Taillard, and De Werra [22]. Our tabu search algorithm works with a relaxation of model (3) that is presented in detail in Section 3.1. The algorithm works in iterations and

maintains the current solution X, the best feasible solution X^* to model (3), and the best feasible solution $\tilde{X}^*$ to the relaxation. Denote the current solution in iteration i by X^i. The initial solution X^1 is given in Section 3.2. In iteration i, the algorithm computes the best solution X' in a neighbourhood of X^i, and moves from X^i to X' by setting $X^{i+1} = X'$, even if X^i is better than X' in terms of objective function value of the relaxation. To avoid cycling of the algorithm, the algorithm maintains a set of forbidden neighbourhood transitions, called *tabu moves*. The tabu moves are stored in a list that is called the *tabu list*. Tabu moves are excluded in the neighbourhood search except if they lead to a solution X' that improves over either X^* or $\tilde{X}^*$ or both. When a move is made tabu, it stays tabu for a fixed number κ_1 of iterations. The algorithm enters a so-called *intensification* and *diversification* phase when after a fixed number κ_2 of iterations no change of $(X^*, \tilde{X}^*)$ has occurred. This is described in Section 3.5. If, after another κ_2 iterations still no change of $(X^*, \tilde{X}^*)$ has occurred it terminates, in which case it returns X^*. In our implementation $(\kappa_1, \kappa_2) = (10, 50)$.

We use the following neighbourhoods: the *node insertion* neighbourhood, the *node deletion* neighbourhood, and the *arc exchange* neighbourhood. The node insertion and deletion neighbourhoods are described in Section 3.3. The node insertion neighbourhood is used in the nearest insertion algorithm by Rosenkrantz, Stearns, and Lewis II [33]. Its use as a local search neighbourhood to go from one feasible solution to another is new, as is the node deletion neighbourhood. These neighbourhoods arise because in the MSP, we are only interested in a subtour; there is no hard constraint that we visit all nodes as in the travelling salesman problem and the vehicle routing problem. There is a relation with the relocation neighbourhood for the vehicle routing problem as described by Kindervater and Savelsbergh [25]. The arc exchange neighbourhood, described in Section 3.4, is a directed version of the traditional 2-Opt neighbourhood for the travelling salesman problem [23, 25].

We will see that we can optimise over all these neighbourhoods in polynomial time. Since each move that improves a best known solution does improve it by at least one, and since we only allow a constant number of consecutive non-improving iterations, the tabu search algorithm runs in pseudo-polynomial time.

3.1 Relaxation of MSP for Tabu Search

Recall from Section 2.1 that we can solve the MSP when the graph G is a directed cycle in polynomial time by solving model (2). When we are given a directed cycle X in G with $s \in V(X)$, we can construct a solution $(X, \boldsymbol{\ell})$ to the MSP by solving a loading problem to determine an optimal choice of $\boldsymbol{\ell}$. The function

$$z^*_{\text{load}} : \mathcal{C} \to \mathbb{R} : C \mapsto z^*_{\text{load}}(C),$$

defined by model (2), which maps each directed cycle $C \in \mathcal{C}$ to the optimal value of the loading problem defined by C, can be used to determine the value of $(X, \boldsymbol{\ell})$.

The tabu search algorithm works on the following formulation:

$$\begin{aligned} \min \quad & z_{\text{TS}}(X) = z^*_{\text{load}}(X) + \rho(\boldsymbol{t}(X) - T)^+ && \text{(7a)} \\ \text{subject to} \quad & X \text{ is a directed cycle in } G \text{ with } s \in V(C), && \text{(7b)} \end{aligned}$$

where ρ is a self-adjusting penalty coefficient, and $(\boldsymbol{t}(C) - T)^+$ is the violation of the constraint on the total time (3e). Model (7) is a combinatorial formulation of a relaxation of the projection of model (3) on the $\boldsymbol{x}$-variables. Note that for all feasible solutions $(\boldsymbol{x}, \boldsymbol{f}, \boldsymbol{z})$

we have that $z_{\mathrm{MSP}}(\boldsymbol{x}, \boldsymbol{f}, \boldsymbol{z}) = z_{\mathrm{TS}}(X)$, where X is the directed cycle that contains s with $A(X) = \mathrm{supp}(\boldsymbol{x}) \setminus \{(v, v) \mid v \in V\}$.

In iteration i the tabu search algorithm algorithm searches a neighbourhood of X^i, where each neighbour is evaluated by solving its associated LP of model (2). This yields X^{i+1}. Note that X^{i+1} might improve over $\tilde{X}^*$, and if X^{i+1} satisfies part (iv) of Definition 1.2 it might also yield a new X^*.

The penalty coefficient ρ is treated similarly to the way that is used by Gendreau, Laporte and Séguin [15]. It is initially set equal to $|V|$ and is multiplied every ten iterations by $2^{\lfloor((\alpha+4)/5)\rfloor-1}$, where α is the number of infeasible solutions among the last ten solutions. This way the penalty of exceeding the time constraint is multiplied by two if no solution satisfied the time constraint in the last ten iterations, and divided by two if all those solutions satisfied the time constraint. Intermediate cases yield lesser changes of ρ. The intuition here is to find the right balance between feasible and infeasible solutions.

3.2 Initial Solution

Initially, the solution is $X = (s, (s, s), s)$ as in Proposition 2.4. As this solution is feasible, it also implies that we start with $X^* = \tilde{X}^* = (s, (s, s), s)$.

3.3 Node Insertion and Deletion Neighbourhoods

Suppose we are given a directed cycle $X = (s = v_0, a_1, v_1, \ldots, a_k, v_k = s)$. For each node $w \in V \setminus V(X)$, let $X_{w,i}$ be the directed cycle that contains s obtained from X by inserting node w directly after node v_i, i.e.,

$$X_{w,i} = (s = v_0, \ldots, v_i, (v_i, w), w, (w, v_{i+1}), v_{i+1}, \ldots, v_k = s),$$

and let $i(w)$ be the index minimising the increase in travel cost over all possibilities of $i \in 0, \ldots, k-1$:

$$i(w) = \arg \min_{i=0,\ldots,k-1} \boldsymbol{c}_A^T(\chi^{A(X_{w,i})} - \chi^{A(X)}).$$

Since we have to reformulate and solve the linear program representing the loading problem for each evaluation of the value of a neighbour, we restrict ourself to inserting w after node $v_{i(w)}$. We will call $X_{w,i(w)}$ the *node insertion neighbour* of X that *adds* w.

Definition 3.1. The *node insertion neighbourhood* of a directed cycle X as above is defined as the set

$$\mathcal{N}_1(X) = \{X_{w,i(w)} \mid w \in V \setminus V(X)\}.$$

We can optimise over the node insertion neighbourhood $\mathcal{N}_1(X)$ by solving $|V \setminus V(X)|$ loading problems.

Now, for $i \in 1, \ldots, k-1$ let X_i be the directed cycle obtained from X by removing node v_i and connecting v_{i-1} to v_{i+1}, i.e.,

$$X_i = (v_0, a_1, \ldots, v_{i-1}, (v_{i-1}, v_{i+1}), v_{i+1}, \ldots, a_k, v_k).$$

We call X_i the *node deletion neighbour* of X that removes v_i.

Definition 3.2. The *node deletion neighbourhood* of a directed cycle X as above is defined as the set

$$\mathcal{N}_2(X) = \{X_i \mid i \in 1, \ldots, k-1\}.$$

We can optimise over the node deletion neighbourhood $\mathcal{N}_2(X)$ by solving $|V(X)| - 1$ loading problems, one for each node in $V(X) \setminus \{s\}$.

It remains to specify the tabu mechanism regarding the node insertion and deletion neighbourhoods. If we move to a neighbour X' of X that adds w, and X' does not improve over X in terms of the objective function, then the node deletion neighbour that removes w is made a tabu move. If X' is the node deletion neighbour of X removing node w, and X' does not improve over X in terms of the objective function, then the node insertion neighbour that adds w becomes tabu.

3.4 Arc Exchange Neighbourhood

Suppose we are given a directed cycle $X = (s = v_0, a_1, v_1, \ldots, a_k, v_k = s)$. The arc exchange neighbourhood of X is a set of directed cycles X' that can be obtained from X by replacing two arcs by two other arcs, and reversing the direction of a part of the cycle. For $i, j \in 1, \ldots, k-1$ with $i < j$, let $X_{i,j}$ denote the directed cycle obtained from X by exchanging arcs (v_{i-1}, v_i) and (v_j, v_{j+1}) for the arcs (v_{i-1}, v_j) and (v_i, v_{j+1}), respectively, and reversing the direction of X on the segment from v_i to v_j, i.e.,

$$X_{i,j} = \\ (v_0, a_1, \ldots, v_{i-1}, (v_{i-1}, v_j), v_j, a_j^{-1}, \ldots, a_{i+1}^{-1}, v_i, (v_i, v_{j+1}), v_{j+1}, \ldots, a_k, v_k).$$

We say that $X_{i,j}$ is obtained from X by *reversing* the segment from v_i to v_j.

Definition 3.3. The *arc exchange neighbourhood* of a directed cycle X as above is defined as the set

$$\mathcal{N}_3(X) = \{X_{i,j} \mid i \in 1, \ldots, k-2, j \in i+1, \ldots, k-1\}.$$

We can optimise over the arc exchange neighbourhood by solving $(|V(X)|-1)(|V(X)|-2)$ loading problems.

The tabu mechanism with respect to the arc exchange neighbourhood is defined as follows. If we move to a neighbour X' that is obtained from X by reversing the segment from v_i to v_j for some indices i, j, and if X' does not improve over X in terms of the objective function, then the arc exchange neighbour that reverses the segment from v_j to v_i becomes tabu.

3.5 Intensification and Diversification

To continue the search process, we set our current solution to X^*, and modify the objective function (7a) to incorporate a penalty term for frequently used arcs. This modified objective function is of the form

$$\tilde{z}_{\mathrm{TS}}(X) = z^*_{\mathrm{load}}(X) + \rho(\boldsymbol{t}(X) - T)^+ + \boldsymbol{\sigma}^T \chi^X,$$

where in iteration i, $\boldsymbol{\sigma} \in \mathbb{R}^A_+$ is defined by $\sigma_a = p_a/i$, and $p_a \in \mathbb{R}^A_+$ is a weight that is determined by previous solutions in which arc a took part, i.e.,

$$p_a = - \sum_{\substack{j \in \{1,\ldots,i-1\} \\ a \in A(X^j)}} \frac{z_{\mathrm{TS}}(X^j)}{|A(X^j)|}.$$

The intuition behind this scheme is that an arc that did not occur in any of the previous solutions will be charged its original cost, whereas an arc that was present in all previous solutions will be charged its part in the average solution value.

As soon as we find an improvement of $(X^*, \tilde{X}^*)$ in terms of the objective function (7a), we proceed as normal. If this improvement does not occur within κ_2 iterations we give up and terminate the search.

4 A Branch-and-Price-and-Cut Algorithm

In this section we describe how to solve the MSP to optimality using a branch-and-price-and-cut algorithm. Section 4.1 describes the relaxations that will be solved in each node of the branch-and-bound tree. The relaxations are solved by column generation, for which the pricing algorithm is described in Section 4.3. Moreover, we strengthen the relaxations by adding valid inequalities described in Section 4.4. Note that the out- and in-degree constraints (3b) and (3c) are GUB constraints. This enables to use strengthened criteria for setting variables based on reduced cost. These criteria are discussed in Section 4.2. We try to set additional variables by logical implications to speed up the computation in each node of the branch-and-bound tree; the logical implications are described in Section 4.5. Section 4.6 presents the branching scheme we employ to subdivide the solution space.

4.1 LP Master Problems of the MSP

Here we describe the relaxations that will be solved in each iteration of the branch-and-price-and-cut algorithm. Focus on some iteration. The LP relaxation is uniquely determined by bounds $\boldsymbol{l}, \boldsymbol{u} \in \{0,1\}^A$ on the $\boldsymbol{x}$-variables. These bounds are incorporated in the linear programming relaxation of model (3), which leads to the following LP master problems:

$$
\begin{aligned}
\min \quad & z^{\boldsymbol{l},\boldsymbol{u}}_{\mathrm{MSP}}(\boldsymbol{x}, \boldsymbol{f}, \boldsymbol{z}) = \boldsymbol{c}_A^T \boldsymbol{x} + \boldsymbol{c}_K^T \boldsymbol{z} && && \text{(8a)}\\
\text{subject to} \quad & \boldsymbol{x}(\delta^-(v)) + x_{(v,v)} = 1 && \forall v \in V, && \text{(8b)}\\
& \boldsymbol{x}(\delta^+(v)) + x_{(v,v)} = 1 && \forall v \in V, && \text{(8c)}\\
& \boldsymbol{x}(A(S) \setminus \{(v,v)\}) \leq |S| - 1 && \forall v \in S \subseteq V \setminus \{s\} && \text{(8d)}\\
& \boldsymbol{t}^T \boldsymbol{x} \leq T, && && \text{(8e)}\\
& \boldsymbol{f}(\mathcal{P}_k) = z_k && \forall k \in K, && \text{(8f)}\\
& \boldsymbol{f}(\{P \in \mathcal{P} \mid A(P) \ni a\}) \leq x_a && \forall a \in A, && \text{(8g)}\\
& \boldsymbol{z} \leq \boldsymbol{d}, && && \text{(8h)}\\
& \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}, && && \text{(8i)}\\
& \boldsymbol{x} \in \mathbb{R}^A_+, \boldsymbol{f} \in \mathbb{R}^{\mathcal{P}}_+, \boldsymbol{z} \in \mathbb{R}^K_+. && && \text{(8j)}
\end{aligned}
$$

In the root node, we take $\boldsymbol{l} = \boldsymbol{0}$ and $\boldsymbol{u} = \boldsymbol{1}$. The subtour elimination constraints (8d), as well as some other classes of valid inequalities for model (3), are added to the formulation when they are reported by our separation algorithms. Apart from the exact separation algorithm for finding violated subtour elimination inequalities, we use a heuristic that looks for violated lifted cycle inequalities, and we use our procedure to find maximally violated mod-k inequalities. These are described in more detail in Section 4.4.

Associate dual variables $\boldsymbol{\pi}$ and $\boldsymbol{\gamma}$ to the constraints (8f) and (8g), respectively. Choose $(v, w) \in K$ and let

$$P = (v = v_0, a_1, v_2, \ldots, a_k, v_k = w)$$

be a path from v to w in G with $u_a = 1$ for all $a \in A(P)$. The reduced cost of the variable f_P is $-\pi_{(v,w)} - \boldsymbol{\gamma}(A(P))$. Moreover any feasible solution to the dual linear programming problem of model (8) satisfies $\boldsymbol{\gamma} \leq \mathbf{0}$.

In each node of the branch-and-bound tree, we solve a restricted version of model (8) with all the $\boldsymbol{x}$- and $\boldsymbol{z}$-variables, and a subset of the $\boldsymbol{f}$-variables. This gives us a primal solution $(\boldsymbol{x}^*, \boldsymbol{f}^*, \boldsymbol{z}^*)$, together with an optimal solution to the dual linear program of the restricted model (8). We check whether all $\boldsymbol{f}$-variables of model (8) that can assume strict positive values satisfy their reduced cost optimality criteria using the pricing algorithm described in Section 4.3. When there exist $\boldsymbol{f}$-variables that violate the reduced cost optimality criteria, we add them and re-optimise. Otherwise, we have a proof that $(\boldsymbol{x}^*, \boldsymbol{f}^*, \boldsymbol{z}^*)$ is an optimal solution to model (8) in the current node.

4.2 Reduced Cost Variable Setting and GUB Constraints

The aim of this section is to state the criteria we use for setting variables based on reduced cost. For this purpose, consider the problem

$$\begin{aligned} \min \quad & z(\boldsymbol{x}) = \boldsymbol{c}^T \boldsymbol{x} && \text{(9a)} \\ \text{subject to} \quad & A\boldsymbol{x} = \boldsymbol{b}, && \text{(9b)} \\ & \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}, && \text{(9c)} \end{aligned}$$

where $\boldsymbol{c}, \boldsymbol{l}, \boldsymbol{u} \in \mathbb{Z}^n$, $\boldsymbol{b} \in \mathbb{Z}^m$, and $A \in \mathbb{Z}^{m \times n}$. Denote the set of all variable indices by N, i.e., $N = \{1, 2, \ldots, n\}$.

We will first restate the traditional arguments for modifying lower and upper bounds based on reduced cost criteria. Suppose that problem (9) is feasible and let $(\boldsymbol{x}^{\text{LP}}, \boldsymbol{\pi})$ be an optimal primal-dual pair to it, where $\boldsymbol{\pi} = \boldsymbol{c}_B^T A_B^{-1}$ for some basis $B \subseteq N$. The reduced cost with respect to $\boldsymbol{\pi}$ is defined as $\boldsymbol{c}^{\boldsymbol{\pi}} = \boldsymbol{c} - (\boldsymbol{\pi}^T A)^T$. Denote $z(\boldsymbol{x}^{\text{LP}})$ by z^{LP}. Define the sets of variable indices L and U as $L = \{j \in N \mid c_j^{\boldsymbol{\pi}} > 0\}$ and $U = \{j \in N \mid c_j^{\boldsymbol{\pi}} < 0\}$. By LP optimality we know that $\boldsymbol{x}_L^{\text{LP}} = \boldsymbol{l}$ and $\boldsymbol{x}_U^{\text{LP}} = \boldsymbol{u}$. The traditional arguments for modifying lower and upper bounds based on reduced cost criteria are captured in the following proposition:

Proposition 4.1. (see e.g. Wolsey [40, Exercise 7.8.7]) *Given $z^* \in \mathbb{Z}$, let $\tilde{\boldsymbol{l}}, \tilde{\boldsymbol{u}} \in \mathbb{Z}^n$ be defined as*

$$\tilde{l}_j = \begin{cases} \max(l_j, u_j + \lceil (z^* - z^{LP} - 1)/c_j^{\boldsymbol{\pi}} \rceil), & \text{if } j \in U, \\ l_j, & \text{otherwise,} \end{cases} \tag{10}$$

$$\tilde{u}_j = \begin{cases} \min(u_j, l_j + \lfloor (z^* - z^{LP} - 1)/c_j^{\boldsymbol{\pi}} \rfloor), & \text{if } j \in L, \\ u_j, & \text{otherwise.} \end{cases} \tag{11}$$

If $\boldsymbol{x}^{IP} \in \mathbb{Z}^n$ satisfies (9b)–(9c) and $z(\boldsymbol{x}^{IP}) < z^*$ then $\tilde{\boldsymbol{l}} \leq \boldsymbol{x}^{IP} \leq \tilde{\boldsymbol{u}}$.

Typically, z^* in the above proposition is taken to be the value of the best known integer feasible solution to problem (9).

We proceed by strengthening the above proposition for the special case in which some of the constraints are GUB constraints. A GUB constraint is a constraint of the form $\boldsymbol{x}(J') = 1$, where $\boldsymbol{x}_{J'} \geq \mathbf{0}$, and where $J' \subseteq N$. Suppose without loss of generality that the first m' constraints are GUB constraints, and let $M' = \{1, 2, \ldots, m'\}$ denote its set of row indices. Also, suppose without loss of generality that the set $N' = \bigcup_{i \in M'} \text{supp}(a_i) = \{1, 2, \ldots, n'\}$ is the set of variable indices involved in one or more GUB constraints. So we have $a_i \in \{0,1\}^n$ and $\text{supp}(a_i) \subseteq N'$ for all $i \in M'$, $\boldsymbol{l}_{N'} = \mathbf{0}$, and $\boldsymbol{u}_{N'} = \mathbf{1}$. For $j \in N'$ let the set $N'(j)$ of *adjacent variable indices* be defined as

$$N'(j) = \{j' \in N' \setminus \{j\} \mid \{j, j'\} \subseteq \text{supp}(a_i) \text{ for some } i \in M'\}.$$

Variable indices j and j' are adjacent if x_j and $x_{j'}$ occur in one or more common GUB constraints. Our improved arguments for strengthening variable bounds based on reduced cost criteria are:

Proposition 4.2. (Verweij [36, Proposition 3.2]) *Given $z^* \in \mathbb{Z}$, let $\tilde{\boldsymbol{c}}^{\boldsymbol{\pi}} \in \mathbb{Q}^n$, $\tilde{\boldsymbol{l}}, \tilde{\boldsymbol{u}} \in \mathbb{Z}^n$ be defined as*

$$\tilde{c}_j^{\boldsymbol{\pi}} = \begin{cases} c_j^{\boldsymbol{\pi}} - \boldsymbol{c}^{\boldsymbol{\pi}}(N'(j) \cap U), & \text{if } j \in L \cap N', \\ \max\{-\tilde{c}_{j'}^{\boldsymbol{\pi}} \mid j' \in N'(j)\}, & \text{if } j \in U \cap N', \\ c_j^{\boldsymbol{\pi}}, & \text{otherwise,} \end{cases} \tag{12}$$

$$\tilde{l}_j = \begin{cases} \max(l_j, u_j + \lceil (z^* - z^{LP} - 1)/\tilde{c}_j^{\boldsymbol{\pi}} \rceil), & \text{if } j \in U, \\ l_j, & \text{otherwise,} \end{cases} \tag{13}$$

$$\tilde{u}_j = \begin{cases} \min(u_j, l_j + \lfloor (z^* - z^{LP} - 1)/\tilde{c}_j^{\boldsymbol{\pi}} \rfloor), & \text{if } j \in L, \\ u_j, & \text{otherwise.} \end{cases} \tag{14}$$

If $\boldsymbol{x}^{IP} \in \mathbb{Z}^n$ satisfies (9b)–(9c) and $z(\boldsymbol{x}^{IP}) < z^*$ then $\tilde{\boldsymbol{l}} \leq \boldsymbol{x}^{IP} \leq \tilde{\boldsymbol{u}}$.

The vector $\tilde{\boldsymbol{c}}^{\boldsymbol{\pi}}_{N \setminus N'}$ equals the vector $\boldsymbol{c}^{\boldsymbol{\pi}}_{N \setminus N'}$, so Proposition 4.2 coincides with Proposition 4.1 for the $\boldsymbol{x}_{N \setminus N'}$-variables. Let us study the per-unit change in objective function value with respect to a variable x_j with $j \in N'$. If $j \in L \cap N'$, setting x_j to one forces $\boldsymbol{x}_{N'(j) \cap U}$ to $\mathbf{0}$. This implies that the change in objective function value is at least $\tilde{c}_j^{\boldsymbol{\pi}}$. If $j \in U \cap N'$, setting x_j to zero forces at least one $x_{j'}$ with $j' \in N'(j)$ to one. This leads to a change in objective function value of at least $\tilde{c}_j^{\boldsymbol{\pi}} = -\tilde{c}_{j'}^{\boldsymbol{\pi}}$ for some properly chosen $j' \in N'(j)$.

Note that feasible solutions $(\boldsymbol{x}, \boldsymbol{f}, \boldsymbol{z})$ to model (3) satisfy $z_{\text{MSP}}(\boldsymbol{x}, \boldsymbol{f}, \boldsymbol{z}) \in \mathbb{Z}$. We apply Proposition 4.2 to the $\boldsymbol{x}$-variables in model (8). Here, the in- and out-degree constraints are the GUB constraints that are used in the definition of $\tilde{\boldsymbol{c}}$.

4.3 The Pricing Problem

Focus on any node of the branch-and-bound tree, and assume we are given vectors $\boldsymbol{\pi} \in \mathbb{R}^K$, and $\boldsymbol{\gamma} \in \mathbb{R}^A$ that are part of an optimal solution to the dual linear program of the restricted model (8) in that node. A path $P = (v \rightsquigarrow w) \in \mathcal{P}$ violates its reduced cost criteria if $-\pi_{(v,w)} - \boldsymbol{\gamma}(A(P)) < 0$, and allows positive flow if for all $a \in A(P)$ we have $u_a = 1$. Recall from the definition of $\mathcal{P}$ that no path in $\mathcal{P}$ has s as an internal node. Let $\tilde{G} = (\tilde{V}, \tilde{A})$ be the graph obtained from G by eliminating all arcs with $u_a = 0$, duplicating s to s' and redirecting all arcs that enter s so that they enter s' (the self-loop (s, s) becomes the arc (s, s')), and

removing all remaining self-loops in $\tilde{A}$. Define a cost vector $\tilde{\boldsymbol{\gamma}} \in \mathbb{R}^{\tilde{A}}$ by setting $\tilde{\gamma}_{(v,w)} = -\gamma_{(v,w)}$ for all $(v,w) \in \tilde{A}$ with $w \neq s'$, and $\tilde{\gamma}_{(v,s')} = -\gamma_{(v,s)}$ for all $(v,s') \in \tilde{A}$.

Lemma 4.3. *Let an upper bound vector $\boldsymbol{u} \in \{0,1\}^A$ and vectors $\boldsymbol{\pi} \in \mathbb{R}^K$ and $\boldsymbol{\gamma} \in \mathbb{R}^A$ be given, and let $\tilde{G}$, $\tilde{\boldsymbol{\gamma}}$ be as described above. Then, there exists a path $P = (v \rightsquigarrow w) \in \mathcal{P}$ with $-\pi_{(v,w)} - \boldsymbol{\gamma}(A(P)) < 0$ and $\boldsymbol{u}_{A(P)} = \mathbf{1}$ if and only if there exists a path $\tilde{P}$ in $\tilde{G}$ with $\tilde{\boldsymbol{\gamma}}(A(P)) < \pi_{(v,w)}$.*

Proof. We first show that there is a one-to-one correspondence between the paths in $\mathcal{P}$ and the paths in $\tilde{G}$. Choose $P = (v = v_0, a_1, v_1, \ldots, a_k, v_k = w) \in \mathcal{P}$ with $\boldsymbol{u}_{A(P)} = \mathbf{1}$ arbitrarily. Observe that node s can occur only as an end node of P. If $s \notin V(P)$ or if $u = s$, then take $\tilde{P} = P$. If $s = v$, then take $\tilde{P} = (v_0, a_1, v_1, \ldots, v_{k-1}, (v_{k-1}, s'), s')$. By construction of $\tilde{G}$, we have that $V(\tilde{P}) \subseteq \tilde{V}$ and $A(\tilde{P}) \subseteq \tilde{A}$. So, $\tilde{P}$ is a path in $\tilde{G}$. Using the reverse argument, we can show that for any path $\tilde{P}$ in $\tilde{G}$, there exists a path $P \in \mathcal{P}$ with $\boldsymbol{u}_{A(P)} = \mathbf{1}$.

Now, let $P = (v \rightsquigarrow w)$ be a path in $\mathcal{P}$ with $-\pi_{(v,w)} - \boldsymbol{\gamma}(A(P)) < 0$, and let $\tilde{P}$ be its corresponding path in $\tilde{G}$. By construction of $\tilde{\boldsymbol{\gamma}}$, we have that $\tilde{\boldsymbol{\gamma}}(A(\tilde{P})) = -\boldsymbol{\gamma}(A(P))$. Substituting this in $-\pi_{(v,w)} - \boldsymbol{\gamma}(A(P)) < 0$ and rewriting proves the lemma. □

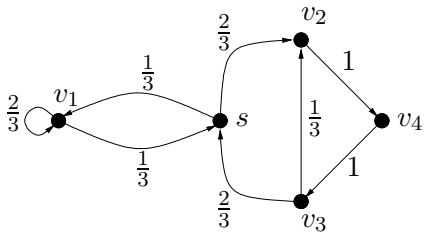
Recall that $\mathcal{P} = \bigcup_{k \in K} \mathcal{P}_k$. Focus on any commodity $k = (v,w) \in K$. By Lemma 4.3, there exists a path $P \in \mathcal{P}$ with $\boldsymbol{u}_{A(P)} = \mathbf{1}$ such that f_P has negative reduced cost if and only if there exists a path $\tilde{P}$ in $\tilde{G}$ from v to w, or to s' if $w = s$, with $\tilde{\boldsymbol{\gamma}}(A(\tilde{P})) < \pi_k$. When $\boldsymbol{\gamma}$ is part of a feasible solution to the dual linear program to the restricted model (8), then $\boldsymbol{\gamma} \leq \mathbf{0}$. It follows that $\tilde{\boldsymbol{\gamma}} \geq \mathbf{0}$. We can use Dijkstra's algorithm [13, 2] on $\tilde{G}$ with arc lengths $\tilde{\boldsymbol{\gamma}}$ to compute a shortest path tree that is directed towards a root node $w \in V$. Using this tree we can solve the pricing problem for all commodities with destination w. So the pricing problem can be solved by running Dijkstra's algorithm $|V|$ times, once for each possible destination. As a consequence, the pricing problem can be solved in $O(|V|(|A| + |V| \log |V|))$ time.

4.4 Valid Inequalities for the MSP

In this section we describe the classes of valid inequalities we use to find violated valid inequalities in each node of the branch-and-bound tree. When adding valid inequalities, we have to take care that we do not alter the reduced cost of the $\boldsymbol{f}$-variables, as this would render our pricing algorithm invalid. Therefore, we have to restrict ourself to inequalities that have zero coefficients for all $\boldsymbol{f}$-variables. As there are no constraints linking the $\boldsymbol{z}$-variables to the $\boldsymbol{x}$-variables directly, we look for violated valid inequalities in the $\boldsymbol{x}$-variables only. So, our goal here is to obtain integer values of the $\boldsymbol{x}$-variables. To reach our goal we try to get a good description of the convex hull of incidence vectors of subtours containing s. This is done by adding the valid inequalities that are described below. Second, we will branch on the $\boldsymbol{x}$-variables, which is the subject of Section 4.6.

4.4.1 Subtour Elimination and Directed Cut Inequalities

Part of the MSP is to compute a subtour including node s. Hence, any solution that contains a subtour that does not contain s is infeasible. The subtour elimination constraints (3d) are incorporated in model (3) to exclude such infeasible solutions. A subtour elimination constraint is indexed by a tuple (v,S) with $v \in S \subseteq V \setminus \{s\}$. By subtracting the out-degree



The global minimum cut is realised by the arc set $\{(s, v_1)\}$, has value $\frac{1}{3}$, and does not correspond to violated subtour elimination constraints. All $s-t$ cuts with $t \in \{v_2, v_3, v_4\}$ have value $\frac{2}{3}$. All subtour elimination constraints on $\{v_2, v_3, v_4\}$ are violated by $\frac{1}{3}$.

Figure 2: Global versus $s-t$ Minimum Cut example

equalities (3b) from the subtour elimination constraint induced by (v, S) for all nodes $u \in S$ and multiplying by minus one we obtain the *directed cut constraints* (see also Balas [4])

$$\boldsymbol{x}(\delta^+(S)) + x_{(v,v)} \geq 1. \tag{15}$$

Similarly, by using the in-degree equalities (3c) we can derive the inequalities

$$\boldsymbol{x}(\delta^-(S)) + x_{(v,v)} \geq 1. \tag{16}$$

By a similar argument, the violation (or the slack) of these inequalities is equal, i.e.,

$$\boldsymbol{x}(A(S) \setminus \{(v,v)\}) - |S| + 1 = 1 - \boldsymbol{x}(\delta^+(S)) - x_{(v,v)} = 1 - \boldsymbol{x}(\delta^-(S)) - x_{(v,v)}.$$

Given a fractional solution $(\boldsymbol{x}^*, \boldsymbol{f}^*, \boldsymbol{z}^*)$ to model (8), the *support graph* $G(\boldsymbol{x}^*)$ is defined as the directed graph

$$G(\boldsymbol{x}^*) = (V, \text{supp}(\boldsymbol{x}^*)),$$

with arc capacities $\boldsymbol{x}^*$. For each $t \in V \setminus \{s\}$, if the maximum $s-t$ flow in $G(\boldsymbol{x}^*)$ is strictly less than the value $1 - x_{(t,t)}$, then all $s-t$ minimum cuts define a directed cut constraint that is violated by $\boldsymbol{x}^*$. It follows that we can prove that a solution satisfies all directed cut constraints, or we can find a violated directed cut constraint by computing $|V|-1$ maximum flows. This can be done in $O(|V|^4)$ time.

When $\boldsymbol{x}^*$ is a fractional solution to model (8), the $s-t$ minimum cut of $G(\boldsymbol{x}^*)$ can have any value between zero and one. It can occur that there are violated subtour elimination constraints, but none of these correspond to a global minimum cut in $G(\boldsymbol{x}^*)$. An example of this is given in Figure 2. Padberg and Rinaldi [30] show that the identification of violated subtour elimination constraints for the travelling salesman problem can be done using Gomory and Hu's algorithm [16], for which the best known implementation is Hao and Orlin's algorithm [21]. Gomory and Hu's algorithm only guarantees to report all global minimum cuts. Since we need to compute all the $s-t$ minimum cuts for each $t \in V \setminus \{s\}$, this implies that we cannot use Gomory and Hu's algorithm, nor can we take advantage of clever data structures as presented by Fleischer [14] that are also designed to represent only the set of all global minimum cuts.

4.4.2 Lifted Cycle Inequalities

Given a cycle C in G, a *cycle inequality* has the form

$$\boldsymbol{x}(A(C)) \leq |V(C)| - 1,$$

and states that from all the arcs in the cycle we have to omit at least one in a solution satisfying the inequality. In the MSP, part of the object we are looking for is a directed

cycle that includes s. Hence, a cycle inequality obtained from a cycle C in G is valid for the MSP if and only if $s \notin V(C)$. Cycle inequalities are not very strong, in general they can be strengthened by lifting. For the asymmetric travelling salesman problem, this was reported on by Balas and Fischetti [6]. One obvious way to strengthen these inequalities (for the MSP) is to add all $\boldsymbol{x}$-variables of the arcs in $A(V(C)) \setminus (A(C) \cup \{(v,v)\})$ with a coefficient equal to one, for some $v \in V(C)$. This results in the subtour elimination constraint on $(v, V(C))$.

For one special class of lifted cycle inequalities Balas and Fischetti gave a heuristic that separates them and runs in polynomial time. This class consists of inequalities that are obtained from subtour elimination constraints. In the remainder of this section, we show that this particular class, and its separation algorithm, can be generalised to the MSP.

Proposition 4.4. *Suppose we are given a set $S \subset V \setminus \{s\}$ and $v \in S$. For $q > 1$, let $\{(u_1, w_1), \ldots, (u_q, w_q)\} \subseteq A(S)$ be a collection of distinct arcs and let $\{v_1, \ldots, v_q\} \subseteq V \setminus (S \cup \{s\})$ be a collection of distinct nodes. Let $U = \{u_1, \ldots, u_q\}$, $W = \{w_1, \ldots, w_q\}$, and let $A_q = \bigcup_{i=1}^{q} \{(u_i, v_i), (v_i, w_i), (u_i, w_i)\}$.*

(i) If $v \notin U \cup W$, then the following inequality is valid for model (3):

$$\boldsymbol{x}(A(S) \setminus \{(v,v)\}) + \boldsymbol{x}(A_q) \leq |S| + q - 1 \tag{17}$$

(ii) If $v \in U \cup W$, then the following inequality is valid for model (3):

$$\boldsymbol{x}(A(S)) + \boldsymbol{x}(A_q) \leq |S| + q - 1.$$

Proof. We will first prove (i). Suppose that $v \notin U \cup W$. The proof is by induction on q. For the case that $q = 1$, we add the following inequalities:

$$\boldsymbol{x}(A(S) \setminus \{(v,v)\}) \leq |S| - 1, \tag{18}$$

$$\boldsymbol{x}(A(S \cup \{v_1\}) \setminus \{(v_1, v_1)\}) \leq |S|, \text{ and} \tag{19}$$

$$2x_{(u_1,w_1)} + x_{(u_1,u_1)} + x_{(w_1,w_1)} + x_{(u_1,v_1)} + x_{(v_1,w_1)} \leq 2. \tag{20}$$

This results in the inequality

$$\begin{aligned} &2(\boldsymbol{x}(A(S) \setminus \{(v,v)\}) \\ &\quad + x_{(u_1,w_1)} + x_{(u_1,v_1)} + x_{(v_1,w_1)}) + x_{(u_1,u_1)} + x_{(w_1,w_1)} \\ &\quad + \boldsymbol{x}(A(S \cup \{v_1\}) \setminus ((A(S) \setminus \{(v,v)\}) \cup \{(u_1,v_1),(v_1,w_1)\})) \leq 2|S| + 1. \end{aligned} \tag{21}$$

Inequality (18) is a subtour elimination constraint on the node set S that is valid for model (3) since $s \notin S$. Inequality (19) is a subtour elimination constraint on the node set $S \cup \{v_1\}$ that is valid for model (3) since $s \notin S \cup \{v_1\}$. Inequality (20) is valid for model (3) because it follows from the degree constraints and the non-negativity constraints on $\boldsymbol{x}$. It follows that inequality (21) is valid for the MSP. After dividing both sides of (21) by 2 and rounding the coefficients down, we obtain the valid inequality

$$\boldsymbol{x}(A(S) \setminus \{(v,v)\}) + x_{(u_1,w_1)} + x_{(u_1,v_1)} + x_{(v_1,w_1)} \leq |S|.$$

This is the special case of inequality (17) for $q = 1$.

For the induction step let $q = t > 1$, and assume that inequality (17) is valid for $q = t - 1$. Adding the inequalities

$$\boldsymbol{x}(A(S) \setminus \{(v,v)\}) + \boldsymbol{x}(A_{q-1}) \leq |S| + q - 2, \tag{22}$$

$$\boldsymbol{x}(A(S \cup \{v_q\}) \setminus \{(v_q,v_q)\}) + \boldsymbol{x}(A_{q-1}) \leq |S| + q - 1, \text{ and} \tag{23}$$

$$2x_{(u_q,w_q)} + x_{(u_q,u_q)} + x_{(w_q,w_q)} + x_{(u_q,v_q)} + x_{(v_q,w_q)} \leq 2, \tag{24}$$

yields inequality (17) after dividing by two and rounding down. Because the inequalities (22) and (23) are valid for model (3) by the induction hypothesis, we conclude that inequality (17) is valid for model (3). This completes the proof of (i).

To prove (ii), observe that if $v \in U \cup W$, then we can assume without loss of generality that $v = u_1$ or $v = w_1$. In this case the same derivation we used to prove (i) yields the proof of (ii). □

Suppose we are given a fractional solution $(\boldsymbol{x}^*, \cdot, \cdot)$ to model (8). Let $v \in S \subseteq V \setminus \{s\}$ and suppose $\boldsymbol{x}^*(A(S) \setminus \{(v,v)\}) = |S| - 1$. The following is a straightforward adaption of the separation procedure proposed by Balas and Fischetti. Given v, S, and $\boldsymbol{x}^*$ we search for a suitable set of arcs $A_q \subseteq A(S)$ and the associated set of nodes $\{v_1, \ldots v_q\} \subseteq V \setminus (S \cup \{s\})$ in a greedy fashion by trying for each arc $(u,w) \in A(S)$ with $v \notin \{u,w\}$ (or with $v \in \{u,w\}$) to identify a node $v' \in V \setminus (S \cup \{s\})$ such that $x_{(u,w)} + x_{(u,v')} + x_{(v',w)} > 1$ (or $x_{(u,w)} + x_{(u,v')} + x_{(v',w)} + x_{(v,v)} > 1$, respectively). Because we start from a tight subtour elimination constraint, this term contributes $x_{(u,w)} + x_{(u,v')} + x_{(v',w)} - 1$ to the violation of the resulting inequality (or $x_{(u,w)} + x_{(u,v')} + x_{(v',w)} + x_{(v,v)} - 1$ in the case that $v \in \{u,w\}$). Therefore, if $A_q \neq \varnothing$, the procedure produces a violated inequality. Moreover, the procedure can be implemented to run in $O(|V|^3)$ time.

4.4.3 Maximally Violated Mod-k Inequalities

We use the separation algorithm described by Caprara, Fischetti, and Letchford [9, 10] to find maximally violated mod-k inequalities (see also Verweij [36]). The algorithm has as input a set of valid inequalities that are satisfied with equality (*tight*) by the current fractional point, and a value of k. Caprara *et al.* show that it suffices to use only values of k that are prime. In our implementation we run the routine for $k = 2, 3, 5, 7, 11, 13, 17, 19, 23, 29$, which are the 10 smallest prime numbers. Moreover, in order to maximise the opportunity to find maximally violated mod-k cuts, we include as many linearly independent constraints that are satisfied with equality as possible. We include all tight directed cut constraints. These can be derived from the computation of the $s - v$ minimum cuts that was done in order to look for violated directed cut constraints. Next to this, we include all tight constraints that are in the current formulation of model (8). These can include non-negativity constraints, and upper bounds of value one on components of $\boldsymbol{x}$ (note that upper bound constraints of value zero can occur but these are only locally valid and therefore excluded), degree constraints (3b), (3c) (which are tight by definition), the knapsack constraint (3e), lifted cycle inequalities (17), lifted cover inequalities, and finally mod-k cuts themselves.

4.4.4 Lifted Knapsack Cover Inequalities

Constraint (3e) states that we have to pack arcs $a \in A$ that take positive time to complete (namely, t_a) into a working day of length T. Hence, solutions satisfying (3e) can be interpreted

knapsack sets (see Padberg [29], Balas [3], Wolsey [38], Hammer, Johnson, and Peled [20], Crowder, Johnson and Padberg [12], Weismantel [37], and Wolsey [40]). Hence, lifted cover inequalities derived from (3e) are valid for the MSP. We use a separation algorithm as described by Van Roy and Wolsey [35] and Gu, Nemhauser and Savelsbergh [19] to identify violated lifted cover inequalities.

4.4.5 Combined Separation Algorithms

The separation algorithms described above are combined as follows. Suppose we are given a fractional solution $(\boldsymbol{x}^*, \cdot, \cdot)$ to model (8). We start by constructing the support graph $G(\boldsymbol{x}^*)$. We build a set F of constraints that are satisfied with equality. Initially, F contains all degree constraints, and all non-negativity constraints and upper bounds of one on components of $\boldsymbol{x}$ that are tight in $\boldsymbol{x}^*$. In order to find directed cut constraints that are satisfied with equality or violated, for each node $t \in V \setminus \{s\}$ we compute a maximum $s-t$ flow in $G(\boldsymbol{x}^*)$. Focus on the $s-t$ maximum flow for any $t \in V \setminus \{s\}$, and let ξ be its value.

If $\xi < 1 - x^*_{(t,t)}$, then there exist directed cuts separating s from t that correspond to violated directed cut constraints. The *closure* of a set of nodes S in a directed graph is the set of all nodes that occur as the end node of a directed path that starts in S. We compute the smallest set S with $s \in S'$ such that $\delta^+(S)$ is an $s-t$ separating cut; S is the closure of $\{s\}$ in the residual graph of the maximum $s-t$ flow. Observe that $\delta^+(S) = \delta^-(V \setminus S)$, and $s \notin V \setminus S$. Hence the inequality $\boldsymbol{x}(\delta^-(V \setminus S)) + x_{(t,t)} \geq 1$ is a directed cut constraint that is violated by $\boldsymbol{x}^*$ and valid for model (3). So, we add $\boldsymbol{x}(\delta^-(V \setminus S)) + x_{(t,t)} \geq 1$ to our formulation.

In the case that $\xi \geq 1 - x^*_{(t,t)}$, all $s-t$ and $t-s$ directed cut constraints are satisfied. As each minimum directed cut that separates s from t corresponds to a tight $s-t$ cut constraint, we compute all $s-t$ minimum cuts and add them to F. Picard and Queranne [32] showed that all $s-t$ minimum cuts appear as closures in the residual graph of the maximum flow that contain s but not t. To enumerate all $s-t$ minimum cuts we first compute all strongly connected components in the residual network of the $s-t$ maximum flow and contract these into single nodes to obtain the *strongly connected component* graph of the residual network. The strongly connected components of the residual network can be computed in $O(|V|+|A|)$ time [11]. Let s' (t') be the strongly connected component that contains s (t', respectively). Note that $s' \neq t'$ because in the residual network of a maximum $s-t$ flow there does not exist any path from s to t. The strongly connected component graph is a directed acyclic graph from which all closures that contain s' but not t' can be enumerated in $O(|V|^3)$ time (see Fleischer [14]). From these closures, we can derive the set of all closures of the residual network that contain s but not t by expanding each node of the strongly connected component graph. Hence, enumerating all tight $s-t$ cuts can be done in $O(|V|^3)$ time.

We proceed to compute lifted cycle inequalities. To do this, we first construct all sets $S \subset V \setminus \{s\}$ with $t \in S$ such that (t, S) induces a tight subtour elimination constraint, i.e., $\boldsymbol{x}^*(A(S) \setminus \{(t,t)\}) = |S| - 1$, and use these as input for the lifted cycle inequality separation routine described above. We construct all such sets S by computing a $t-s$ maximum flow in $G(\boldsymbol{x}^*)$, and computing all $t-s$ minimum cuts as above. If S_1 is a $s-t$ minimum cut and S_2 is a $t-s$ minimum cut, then the set $S = (V \setminus S_1) \cap S_2$ induces a tight subtour elimination constraint on (t, S). We do this for all combinations of S_1 and S_2, each of them producing a different set S.

Next, we run the separation routine for mod-k cuts starting from the cuts in F. Finally,

we run the lifted cover inequality separation on $\boldsymbol{x}^*$.

4.5 Logical Implications

In this section we present the logical implications we use to set variables in each node of the branch-and-bound tree. These implications are all on the bounds of the $\boldsymbol{x}$-variables, and force components of $\boldsymbol{x}$ either to 0 or to 1. Note that this is compatible with model (8). These implications not only help to speed up the solution of the restricted linear programs, but also make the graph $\tilde{G}$ used in the pricing problem of Section 4.3 sparser. This helps in reducing the number of iterations needed for the column generation process to terminate in each node. The implications used are the following:

Proposition 4.5. *Choose any node $v \in V$ arbitrarily, and let $\boldsymbol{x}$ be a feasible solution to model (8). Take arc $a \in \delta^+(v) \cup \{(v,v)\}$ (or $a \in \delta^-(v) \cup \{(v,v)\}$), and let $A' = (\delta^+(v) \cup \{(v,v)\}) \setminus \{a\}$ (or $A' = (\delta^-(v) \cup \{(v,v)\}) \setminus \{a\}$, respectively). Then, $l_a = 1$ implies $\boldsymbol{x}_{A'} = \boldsymbol{0}$, and $\boldsymbol{u}_{A'} = \boldsymbol{0}$ implies $x_a = 1$.*

Proof. Directly from the degree constraints, the non-negativity constraints, and the upper bound constraints on $\boldsymbol{x}$. □

So, if $x_{(u,v)}$ is set to one, then x_a can be set to zero for all arcs $a \in (\delta^+(u) \cup \{(u,u)\}) \setminus \{(u,v)\}$ and $a \in (\delta^-(v) \cup \{(v,v)\}) \setminus \{(u,v)\}$. If x_a is set to zero for all arcs $a \in (\delta^+(u) \cup \{(u,u)\}) \setminus \{(u,v)\}$ or for all arcs $a \in (\delta^-(v) \cup \{(v,v)\}) \setminus \{(u,v)\}$, then $x_{(u,v)}$ can be set to one. These implications can propagate through the graph. We use the following algorithm to compute as many implications as possible, starting from an initial set of variable settings. Suppose we are give sets $A_0, A_1 \subset A$ of arcs with $\boldsymbol{u}_{A_0} = \boldsymbol{0}$ and $\boldsymbol{l}_{A_1} = \boldsymbol{1}$ such that the bounds $\boldsymbol{l}$, $\boldsymbol{u}$ define a feasible instance of model (8). For each arc $(u,v) \in A_1$ we compute the set of arcs A' that have $\boldsymbol{u}_{A'} = \boldsymbol{1}$ and can be set to zero by Proposition 4.5. We set $\boldsymbol{u}_{A'} := \boldsymbol{0}$, add A' to A_0, and remove (u,v) from A_1. This way, we proceed until A_1 is empty. Next, we process the arcs in A_0. Let arc $(u,v) \in A_0$ and let A' be the set of arcs that have $\boldsymbol{l}_{A'} = \boldsymbol{0}$ and can be set to one by Proposition 4.5. We remove (u,v) from A_0. If $A' = \varnothing$, we proceed with the next arc. Otherwise, we set $\boldsymbol{l}_{A'} := \boldsymbol{1}$ and set $A_1 := A'$. This gives us two new sets A_0, A_1, for which we start again from the beginning.

The correctness of this procedure follows from Proposition 4.5. Because every arc is set to 0 or 1 only once, the procedure can be implemented to run in $O(|V| + |A|)$ time.

4.6 Enumeration and Branching

There are two aspects of the branch-and-price-and-cut algorithm that need to be specified to complete its description, namely, how we select the next open problem and how we choose to branch on fractional solutions. We use the best-first strategy in selecting the next open problem, i.e., we choose to process the open problem for which the value of the LP relaxation in its parent node is as low as possible.

There are several possible ways to branch on a fractional solution. One way is branching on a fractional $\boldsymbol{x}$-variable, another is using a GUB constraint to derive a partitioning of the feasible region. We compute a so-called *pseudo-cost* for each possible branch, and then use these to decide on how to partition the feasible region. Details of our branching scheme can be found in Verweij [36]. Using pseudo-costs dates back to the nineteen seventies and has most recently been reported on by Linderoth and Savelsbergh [27].

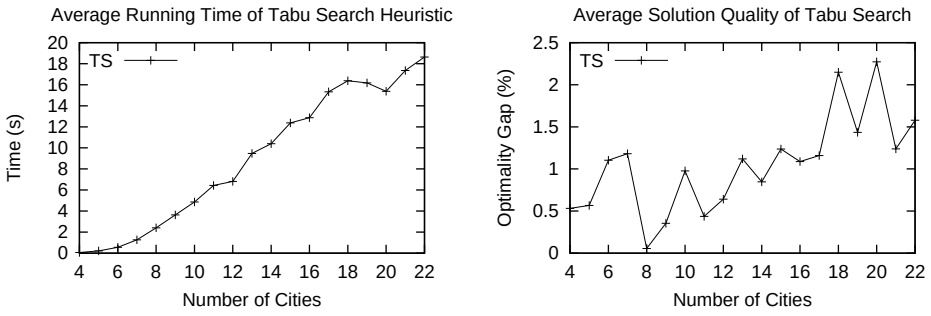


Figure 3: Average Running Time and Solution Quality of Tabu Search for the MSP

Observe that whatever branching decision we make, in each branch the LP relaxation differs only in the upper and lower bounds of the $\boldsymbol{x}$-variables. This implies that the LP relaxations in each node of the branch-and-bound tree are indeed of the form of model (8).

5 Computational Results

In this section, we present computational results of the algorithms described in this paper. Since we are interested in the MSP because it arises as sub-problem in the context of the Van Gend & Loos problem (see Verweij [36, Chapter 7]) we used our main instance of the Van Gend & Loos problem to generate random instances for the MSP. This was done as follows. The instance of the Van Gend & Loos problem consists of 27 cities, a corresponding distance matrix, and a fixed demand of commodities. For each $n = 4, 5, \ldots, 22$, we generated 50 MSP instances on n cities by selecting 50 distinct subsets of n cities at random, with equal probability. This defines the node set V for each randomly generated problem instance, and we always take $A = V \times V$ as the set of arcs. The arc cost $\boldsymbol{c}_A$ and the demand vector are taken as in the Van Gend & Loos problem instance. For each node set V generated this way, we select one node $s \in V$ to be the depot (again with uniform probability). The set K is taken to be the set of all pairs in $V \times V$ for which the Van Gend & Loos problem instance has positive demand. We generate the profits $\boldsymbol{c}_K$ at random such that the expected profit of each commodity equals the distance from its source to its destination, in absolute value, as follows. For $k = (u, v) \in K$, let X be a random variable drawn from a geometric distribution with success probability $p = 1/c_a$, where $a = (u, v) \in A$ is the arc between the source of commodity k and the destination of commodity k. We set $c_k := -X$. The entire test set generated this way consists of 950 instances of the MSP.

The solution quality and running times of the tabu search algorithm of Section 3 are depicted in Figure 3. The running time and branch-and-bound tree sizes of the branch, price and cut algorithm presented in Section 4 are depicted in Figure 4. The dimensions of the integer program and the constraint pool sizes are presented in Figure 5, together with data on the performance of the valid inequalities. The gaps reported in Figure 5 are computed as follows. Consider any run of the branch-and-bound algorithm, and any class of cutting planes. Let z_1 be the value of the optimal solution to model (8) in the root node without the subtour elimination constraints (the *2-matching relaxation*), let z_2 be the value of the 2-matching relaxation (also in the root node) together with all inequalities of the class of

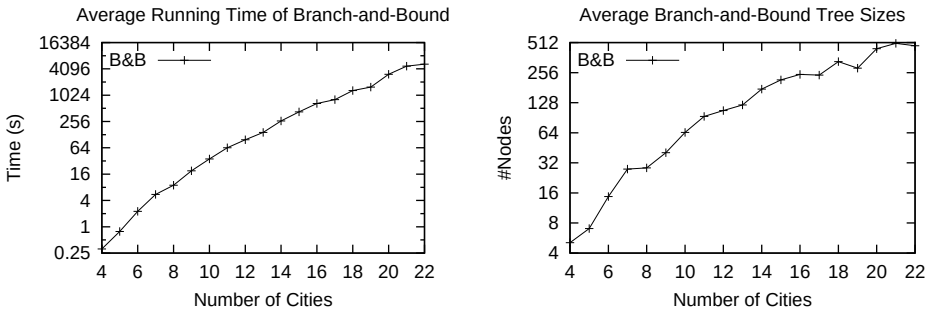


Figure 4: Average Branch-and-Bound Running Time and Tree Sizes

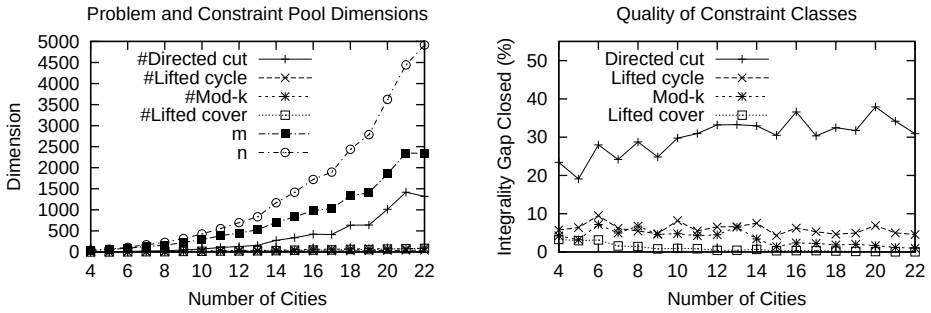


Figure 5: Problem Statistics and Constraint Performance

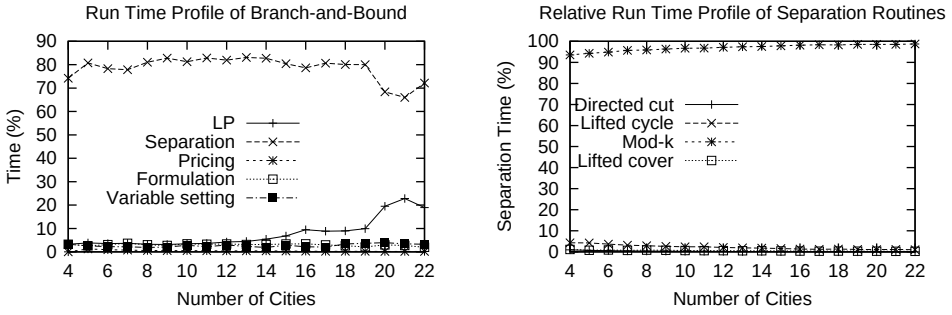


Figure 6: Run Time Profiling Data

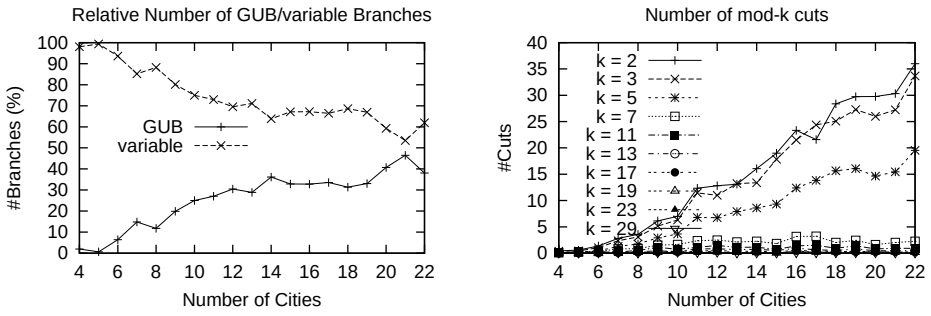


Figure 7: Branching Decisions and Mod-k Statistics

valid inequalities of interest found in this particular run of the algorithm, and let z^* be the value of the optimal integer solution. We measure the gap closed by the class of inequalities by the value $(z_2 - z_1)/(z^* - z_1) * 100\%$. Run time profiles can be found in Figure 6. Finally, the mix of variable against GUB branches and the distribution of the mod-k cuts over the different values of k are depicted in Figure 7.

From Figure 3 it is clear the solutions produced by the tabu search algorithm are of a very high quality. For all problem sizes the reported solutions were within 3% of optimality on average. The running time of the tabu search algorithm behaves equally well.

Our branch-and-price-and-cut algorithm solves problem instances with up to 22 cities within 2 hours of computing time on average. The average branch-and-bound tree sizes on these runs stay just under 512 nodes (see Figure 4).

From Figure 5 it is clear that the lifted cycle inequalities contribute significantly to the quality of our LP formulations. The same holds for the mod-k inequalities. The lifted cover inequalities appear to be less important. It would be interesting to see whether the so-called *lifted GUB cover inequalities* by Wolsey [39], that can also be used for the MSP, perform better and if so, by how much.

Also from Figure 5, and from Figure 6, we can see that solving the LP relaxations becomes a substantial part of the running time when the number of variables increases. Applying a primal-dual algorithm (see for example [31]) should allow us to solve the linear programs more efficiently.

The running time within the separation algorithms is used mainly for mod-k separation. This may be due to our policy to include as many tight cuts as possible. Caprara *et al.* [10] show that for the travelling salesman problem such a policy is not necessary. Their ideas can also be applied to the MSP.

Finally, Figure 7 shows that the our GUB branching scheme produces a constant fraction of all branches that are actually performed, and is more competitive for larger values of $|V|$. We conclude that we are able to solve the MSP to optimality for real-life instances with up to 22 cities within reasonable time, and we expect that we can solve even larger problems by taking the above considerations into account.

References

[1] E. Aarts and J.K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons Ltd, Chichester, 1997.

[2] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1993.

[3] E. Balas. Facets of the knapsack polytope. *Mathematical Programming*, 8:146–164, 1975.

[4] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19:621–636, 1989.

[5] E. Balas. The prize collecting traveling salesman problem: II. Polyhedral results. *Networks*, 25:199–216, 1995.

[6] E. Balas and M. Fischetti. Lifted cycle inequalities for the asymmetric traveling salesman problem. *Mathematics of Operations Research*, 24(2):273–292, 1999.

[7] P. Bauer. The circuit polytope: Facets. *Mathematics of Operations Research*, 22(1):110–145, 1997.

[8] P. Bauer, J.T. Linderoth, and M.W.P. Savelsbergh. A branch and cut approach to the cardinality constrained circuit problem. Technical Report TLI/LEC-98-04, Georgia Institute of Technology, 1998.

[9] A. Caprara, M. Fischetti, and A. Letchford. On the separation of maximally violated mod-k cuts. In G. Cornuéjols, R.E. Burkard, and G.J. Woeginger, editors, *Integer Programming and Combinatorial Optimization, Proceedings of the 7th International IPCO Conference*, volume 1610 of *Lecture Notes in Computer Science*, pages 87–98. Springer-Verlag, Berlin, 1999.

[10] A. Caprara, M. Fischetti, and A. Letchford. On the separation of maximally violated mod-k cuts. *Mathematical Programming*, 87:37–56, 2000.

[11] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 1990.

[12] H. Crowder, E.L. Johnson, and M.W. Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31:803–834, 1983.

[13] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[14] L. Fleischer. Building chain and cactus representations of all minimum cuts from Hao-Orlin in the same asymptotic run time. *Journal of Algorithms*, 33:51–72, 1999.

[15] M. Gendreau, G. Laporte, and R. Séguin. A tabu search heuristic for the vehicle routing problem with stochastic demands and customers. *Operations Research*, 44(3):469–477, 1996.

[16] R.E. Gomory and T.C. Hu. Multi-terminal network flows. *SIAM Journal on Applied Mathematics*, 9:551–570, 1961.

[17] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, Berlin, 1988.

[18] M. Grötschel and M.W. Padberg. Polyhedral theory. In Lawler et al. [26], pages 251–305.

[19] Z.G. Gu, G.L. Nemhauser, and M.W.P. Savelsbergh. Lifted cover inequalities for 0-1 integer programs: Computation. *INFORMS Journal on Computing*, 10(4):427–437, 1998.

[20] P.L. Hammer, E.L. Johnson, and U.N. Peled. Facets of regular 0-1 polytopes. *Mathematical Programming*, 8:179–206, 1975.

[21] J. Hao and J.B. Orlin. A faster algorithm for finding the minimum cut in a graph. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 165–174, 1992.

[22] A. Hertz, E. Taillard, and D. de Werra. Tabu search. In Aarts and Lenstra [1], pages 121–136.

[23] D.S. Johnson and L.A. McGeoch. The traveling salesman problem: a case study. In Aarts and Lenstra [1], pages 215–310.

[24] D.S. Johnson and C.H. Papadimitriou. Computational complexity. In Lawler et al. [26], pages 37–85.

[25] G.A.P. Kindervater and M.W.P. Savelsbergh. Vehicle routing: handling edge exchanges. In Aarts and Lenstra [1], pages 337–360.

[26] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors. *The Traveling Salesman Problem*. John Wiley & Sons Ltd, Chichester, 1985.

[27] J.T. Linderoth and M.W.P. Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11(2), 1999.

[28] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley and Sons, New York, 1988.

[29] M.W. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5:199–215, 1973.

[30] M.W. Padberg and G. Rinaldi. Facet identification for the symmetric traveling salesman polytope. *Mathematical Programming*, 47:219–257, 1990.

[31] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization, Algorithms and Complexity*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

[32] J.C. Picard and M. Queranne. On the structure of all minimum cuts in networks. *Mathematical Programming Study*, 13:8–16, 1980.

[33] D.J. Rosenkranz, R.E. Stearns, and P.M. Lewis II. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6:563–581, 1977.

[34] A. Schrijver. *Theory of linear and integer programming.* John Wiley & Sons Ltd, Chichester, 1986.

[35] T.J. Van Roy and L.A. Wolsey. Solving mixed integer programming problems using automatic reformulation. *Operations Research*, 35(1):45–57, 1987.

[36] A.M. Verweij. *Selected Applications of Integer Programming: A Computational Study.* PhD thesis, Department of Computer Science, Utrecht University, Utrecht, 2000.

[37] R. Weismantel. On the 0/1 knapsack polytope. *Mathematical Programming*, 77:49–68, 1997.

[38] L.A. Wolsey. Faces for a linear inequality in 0-1 variables. *Mathematical Programming*, 8:165–178, 1975.

[39] L.A. Wolsey. Valid inequalities for mixed integer programs with generalised and variable upper bound constraints. *Discrete Applied Mathematics*, 25:251–261, 1990.

[40] L.A. Wolsey. *Integer Programming.* John Wiley and Sons, New York, 1998.