CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

A Branch-and-Cut Algorithm for the Preemptive Swapping Problem

Charles Bordenave
Michel Gendreau
Gilbert Laporte

June 2008

CIRRELT-2008-23

Bureaux de Montréal :
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince, local 2642
Québec (Québec)
Canada G1K 7P4
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

A Branch-and-Cut Algorithm for the Preemptive Swapping Problem

Charles Bordenave[1,2,*] , Michel Gendreau[1,2] , Gilbert Laporte[1,3]

1. Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)
2. Department of Computer Science and Operations Research, Université de Montréal, P.O. Box 6128, Station Centre-ville, Montréal, Canada H3C 3J7
3. Canada Research Chair in Distribution Management, HEC Montréal, 3000 Côte-Sainte-Catherine, Montréal, Canada H3T 2A7

Abstract. In the Swapping Problem (SP), every vertex of a complete graph may supply and demand an object of a known type. A vehicle of unit capacity starting and ending its tour at an arbitrary vertex is available for carrying objects of given types between vertices. The SP consists of determining a minimum cost route that allows the vehicle to satisfy every supply and demand. This article investigates the preemptive version of the SP in which the objects are allowed to be dropped at temporary locations along the route. The problem is modeled as a mixed integer linear program which is solved by branch-and-cut. Computational results on random geometric instances containing up to 100 vertices and eight object types are reported.

Keywords. Swapping Problem, vehicle routing, branch-and-cut, precedence relationships.

Acknowledgements. This research was partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under grants 338816-05 and 39682-05.This support is gratefully acknowledged.

---

* Corresponding author: Charles.Bordenave@cirrelt.ca

This document is also published as Publication #1321 by the Department of Computer Science and Operations Research of the Université de Montréal.

Dépôt légal – Bibliothèque et Archives nationales du Québec,
Bibliothèque et Archives Canada, 2008

1 Introduction

The purpose of this paper is to describe a branch-and-cut algorithm for the *Swapping Problem* (SP) defined as follows. Let $G = (V, A)$ be a complete directed graph, where $V = \{1, \ldots, n\}$ is the vertex set and $A = \{(i, j) \mid i \in V, j \in V, i \neq j\}$ is the arc set. Without loss of generality, vertex 1 is arbitrarily designated as a *depot*. A cost matrix (c_{ij}) satisfying the triangular inequality is defined on A. We consider a set $O = \{1, \ldots, m\}$ of m *object types* located at the vertices. With vertex i is associated a pair (a_i, b_i) of object types corresponding to its supply and its demand. Initially, the supply object is located at the vertex. Each object has a unit weight and appears the same number of times as a supply object and as a demand object. In the SP, the aim is to carry the objects using a unit capacity vehicle, in such a way that all vertices receive their demand object and the total cost is minimized. The vehicle can perform empty trips (called *deadheading*), in which case it is assumed to carry a *null object* denoted by 0. The version of the SP considered in this paper is called *preemptive*, meaning that the objects are *droppable*, i.e., they can be dropped at temporary locations along the route before being moved to their final destination.

The SP was introduced by Anily and Hassin [3] who proved it is NP-hard by reduction to the *Traveling Salesman Problem* (TSP), and derived interesting structural properties of optimal solutions. They also developed a 2.5-approximation algorithm based on matching and patching methods. The case of a linear graph was analyzed by Anily et al. [1] and was shown to be solvable in $O(n^2)$ time. The same authors ([2]) proved that the preemptive SP on a tree is NP-hard by reduction to the *Steiner Tree Problem* on a bipartite graph, but the case where $m = 2$ can be solved in polynomial time. The authors developed a 1.5-approximation heuristic for the case where $m \geq 3$.

A well known problem closely related to the SP is the *Stacker Crane Problem* (SCP) in which each vertex has a supply or a demand, but not both, and each object appears only once as a supply and as a demand. This NP-hard problem is a special case of the asymmetric TSP and also corresponds to a special case of the SP in which there exists only one object for each type. Atallah and Kosaraju [9] proved that the preemptive SCP on a line and on a circle can be solved in $O(m + n)$ time. Frederickson and Guan [17] showed that the preemptive SCP on a tree can also be solved in polynomial time, and proposed two exact algorithms of order $O(m + n \log n)$ and $O(m + qn)$ (where $q \leq \min\{m, n\}$). Table 1 summarizes known complexity results for the preemptive versions of the SCP and the SP.

Another related problem is the *One-commodity Pickup-and-Delivery TSP* (m-PDTSP when m commodities are considered) investigated by Hernández-Pérez and Salazar González [21, 22]. In the 1-PDSTP, each vertex of a complete graph

Graph structure	Preemptive SCP	Preemptive SP
General	NP-hard[1]	NP-hard [3]
Tree	Polynomial [17]	NP-hard [2]
Circle	Polynomial [9]	?
Line	Polynomial [9]	Polynomial [1]

[1]By reduction to the TSP

Table 1: Complexity results for the preemptive SCP and the preemptive SP

is associated with a non-negative demand or supply of a single product. The problem consists of determining a shortest Hamiltonian tour for a capacitated vehicle, starting and ending its route at an arbitrary vertex, in such a way that every request is satisfied. The 1-PDTSP and the SP belong to the class of the *many-to-many* routing problems ([11]), in which multiple supplies can serve multiple demands. The authors have proposed a branch-and-cut algorithm that combines standard TSP constraints with Benders cuts, clique cuts and multistar inequalities (see [23] and [25]). Optimal solutions on instances containing up to 100 vertices were reported.

Finally, routing problems with precedence constraints are also related to the preemptive SP. These include the *Sequential Ordering Problem* (SOP) introduced by Escudero [16]. The SOP is defined on a complete directed graph $G = (V, A)$, where the vertex set V represents jobs to be processed on a single machine, and the arc set A represents sequencing of the jobs. A cost matrix representing processing and setup times is defined on A. Given an additional precedence graph that specifies sequencing relationships between the different jobs, the SOP consists of determining a minimum cost Hamiltonian path in which no precedence constraints are violated. Ascheuer et al. [6, 7], and Ascheuer [5] have proposed branch-and-cut algorithms that make use of various types of valid inequalities for solving this NP-hard problem. It should be noted that in the preemptive SP, unlike in the SOP, the precedence relationships are not known a priori, but they are induced by the solution.

Our aim is to develop, for the first time, an exact branch-and-cut algorithm for the preemptive SP on a general graph. The remainder of the paper is organized as follows. In Section 2 we prove some properties of optimal solutions and present our approach to handle preemption. Sections 3 and 4 cover the mathematical model and the branch-and-cut algorithm. Computational results are presented in Section 5. A description of how the algorithm can be modified to solve the general or *mixed* SP is given in Section 6, followed by conclusions in Section 7.

2 Properties of optimal solutions

After recalling an important result presented in [3] and exhibiting some structural properties of optimal solutions, we introduce our technique for the handling of precedence relationships induced by droppable objects. We use the same definition of optimality as in [3].

Definition 1. *A solution is called optimal if it has minimum cardinality among all solutions that minimize the objective function.*

2.1 Structural properties

Theorem 1 (Anily and Hassin [3]). *There exists an optimal solution in which every vertex $i \in V$ satisfies the following properties: 1) i is incident to at most one incoming arc carrying an object of type b_i, one outgoing arc carrying an object of type a_i, and at most one additional pair of deadheadings (one entering i, the other leaving it); 2) if there is a drop at i, then i is different from the depot and the drop is associated with the first entry in i and the last exit from it.*

Theorem 1 is the backbone of our model. It implies that there exists an optimal solution in which every vertex is visited at most three times. It also indicates the possible order in which the vehicle traverses the arcs incident to a given vertex. In the remainder of this section we will use this theorem to describe all possible configurations that can be part of an optimal solution.

Proposition 1. *In an optimal solution the depot can be possibly visited twice if and only if $a_1 \neq 0$ and $b_1 \neq 0$.*

Proof. Consider an optimal solution S.
$\Rightarrow$ Suppose the depot is visited twice (i.e., it is incident to four arcs), and $a_1 = 0$ or $b_1 = 0$ (or both). By Theorem 1, there exists at most one incoming arc carrying b_1 and one outgoing arc carrying a_1 (actually since the depot must necessarily be visited at least once in any solution, "at most once" means "exactly once"). Therefore no additional null object can be carried to or from the depot. Furthermore, by Theorem 1, there is no drop at the depot. Then the depot is incident to two arcs, thus contradicting our assumption.
$\Leftarrow$ Suppose $a_1 \neq 0$ and $b_1 \neq 0$ like in the instance shown in Figure 1 (the depot is represented as a square dot). On such a small instance it is easy to check by enumeration that the optimal solution is the one depicted in Figure 2. The depot is incident to four arcs: a pair associated with the carrying of its demand and supply and a pair of deadheadings (incoming and outgoing). □

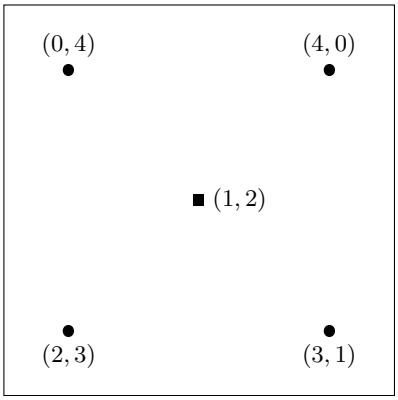

Figure 1: Instance with $a_1 \neq 0$ and $b_1 \neq 0$

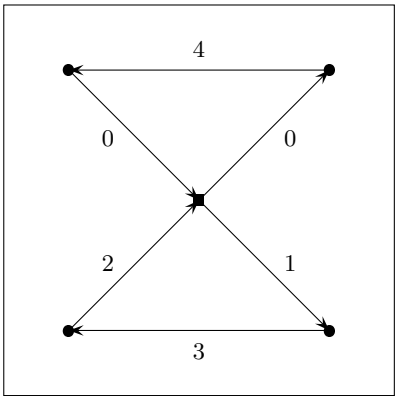

Figure 2: Optimal solution

Definition 2. *A vertex i with $a_i = b_i$ is called a transshipment vertex.*

Proposition 2. *In an optimal solution no transshipment vertex different from the depot can be visited exactly once.*

Proof. Consider an optimal solution S. Suppose there exists a transshipment vertex $i \in V \setminus \{1\}$ that is visited exactly once. Then there exists an incoming arc (u, i) carrying object type b_i and an outgoing arc (i, v) carrying a_i. By Definition 2, $a_i = b_i$. Replacing these two arcs with a unique arc (u, v) carrying $a_i = b_i$, yields a new feasible solution S' which is no worse than S and contains fewer arcs, thus contradicting the optimality of S. The proposition does not hold for the depot because by definition the depot belongs to all feasible solutions and therefore cannot be skipped by the vehicle (the contraction described above cannot be applied). □

Proposition 3. *If object type k is dropped at vertex i, then $k \neq a_i$ and $k \neq b_i$.*

Proof. By Theorem 1, there exists at most one incoming arc carrying b_i and one outgoing arc carrying a_i. Therefore the dropped object, if any, must be different from a_i and b_i. □

Proposition 4. *There exists an optimal solution that does not contain two consecutive arcs associated with the same object type.*

Proof. Follows from the triangular inequality of the cost matrix and Definition 1. □

The structural properties of optimal solutions are summarized in Figure 3 which depicts all possible configurations for a vertex, in terms of incoming and outgoing arcs, in an optimal solution. Numerical values have been added for the sake of clarity. When an object of type k is used, it is assumed to be different from the null object, from the supply, and from the demand (i.e., $k \neq 0$, $k \neq a_i$ and $k \neq b_i$).

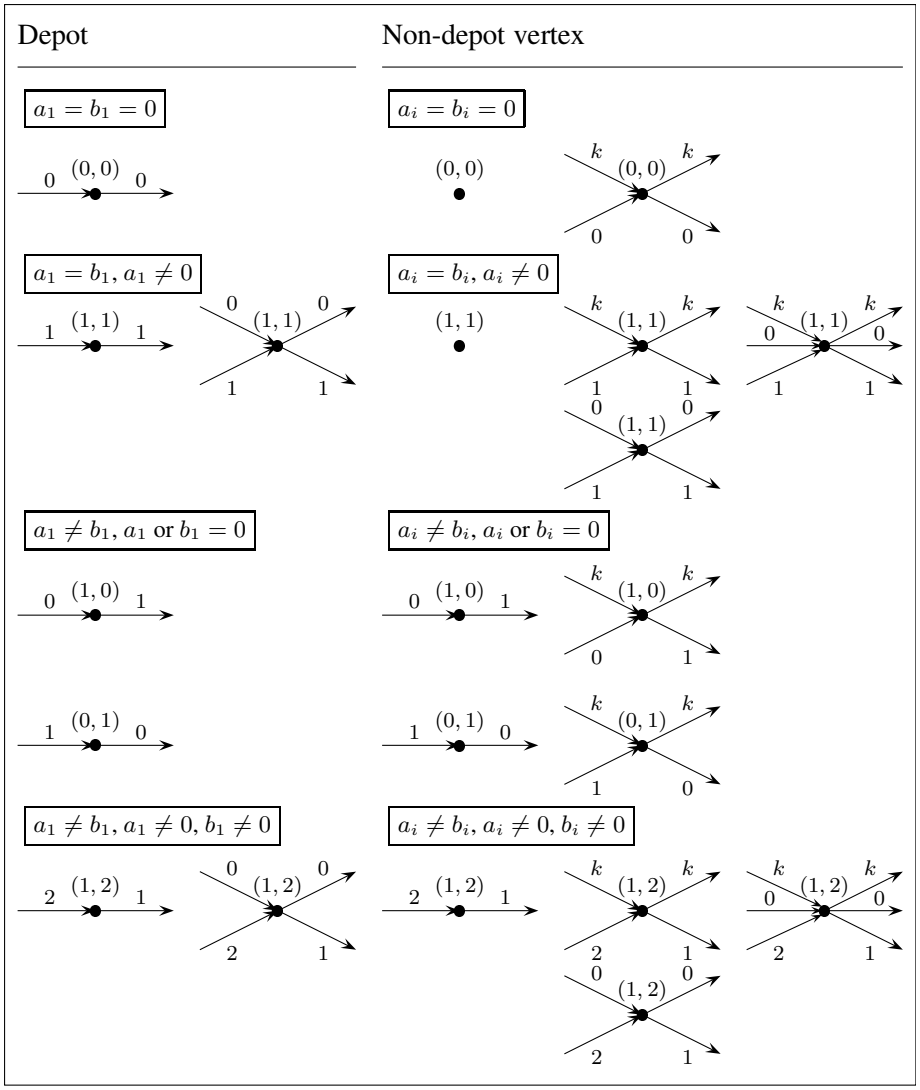

Figure 3: All possible configurations for a vertex in an optimal solution

Proposition 5. *The choice of the depot may influence the value of the optimal solution.*

Proof. Consider the instance and solution shown in Figure 4 in which the depot is located at the center of the unit square. On such a small instance it is easy to check by enumeration that the solution is optimal and has a cost of 6.4. Now if we choose the upper-left vertex as the depot, the optimal solution has a cost of 5.4 (see Figure 5). □

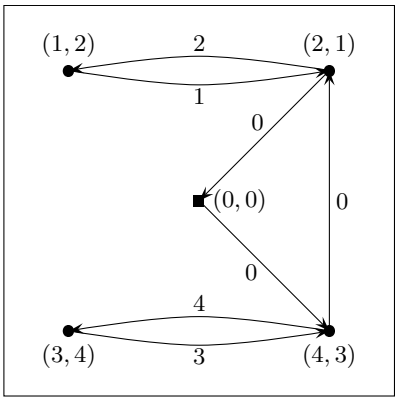

Figure 4: Optimal solution of cost 6.4

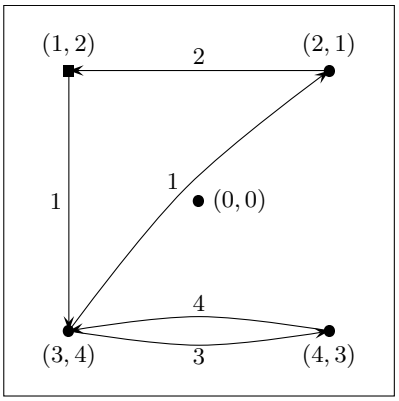

Figure 5: Optimal solution of cost 5.4

2.2 Handling preemption

Handling preemption generates a significant level of difficulty to the formulation of the SP since it is necessary to keep information on the portion of the vehicle route prior to visiting a vertex. This stems from the fact that the vehicle cannot reload an object at a vertex if the object has not been previously dropped at that vertex. We must somehow keep track of the vertices already visited. To this end, we split each vertex into three vertices i, $i+n$ and $i+2n$ such that $a_i = a_{i+n} = a_{i+2n}$, $b_i = b_{i+n} = b_{i+2n}$ and $c_{i,i+n} = c_{i,i+2n} = c_{i+n,i+2n} = 0$ (see Figures 6 and 7). This triplication is justified by Theorem 1 which implies that the vehicle may visit the same vertex at most three times in an optimal solution. In this representation, a vertex i is used for a first visit, whereas $i+n$ and $i+2n$ represent a second and third visit to i respectively. This precedence relationship is denoted by the symbol $\prec$. For example $i \prec i+n$ means that vertex i must be visited prior $i+n$ in any feasible solution.

In what follows we use the sets $V_1 = \{1, \ldots, n\}$, $V_2 = \{n+1, \ldots, 2n\}$ and $V_3 = \{2n+1, \ldots, 3n\}$ to distinguish between the triplicates. The full set of

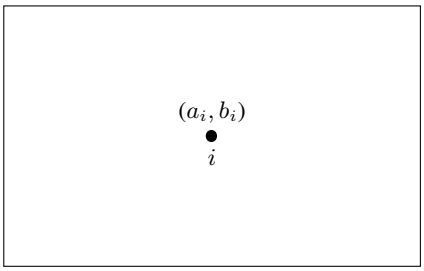

Figure 6: Original vertex

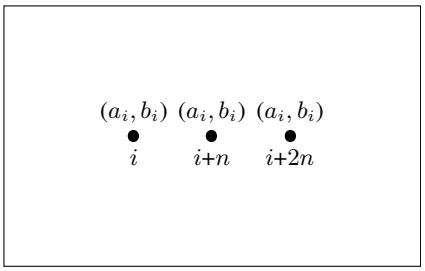

Figure 7: Vertex triplication (copies)

vertices is denoted by $\mathcal{V} = V_1 \cup V_2 \cup V_3$. To avoid confusion between the vertices, a vertex of V is called the *original* vertex and the triplicates are referred to as its *copies*. We also consider an additional object, noted -1, that can only be carried between two consecutive copies of the same vertex (i.e., from i to $i+n$ and from $i+n$ to $i+2n$). More precisely, if the vehicle visits an original vertex three times, then after triplication object -1 is not used among the arcs incident to its copies. Otherwise, -1 is carried from i to $i+n$ and from $i+n$ to $i+2n$ when the original vertex is visited only once, and from $i+n$ to $i+2n$ when the original vertex is visited twice. This representation will allow us to introduce some constraints to handle preemption, i.e., to force the vehicle to always visit the first copy before the second copy, and the second copy before the third one.

2.3 Discardable arcs

From Theorem 1 many arcs associated with an object type can be discarded because there exists an optimal solution that does not contain them. Let $\mathcal{O}$ be the set containing all object types that can be carried between two vertices, i.e., $\mathcal{O} = O \cup \{-1, 0\}$.

Definition 3. *A triplet (i, j, k), $i \in \mathcal{V}$, $j \in \mathcal{V}$, $j \neq i$, $k \in \mathcal{O}$, is called discardable if the carrying of object type k from vertex i to vertex j cannot be part of an optimal solution. Denote by $\mathcal{N}$ the subset of non-discardable triplets.*

We begin by enumerating some discardable triplets that concern every vertex (Propositions 6 and 7), and then analyze particular cases based on the supply and the demand object. All cases are in a way or another implied by Theorem 1 and the vertex triplication described in Section 2.2.

Proposition 6. *If $i \in V_1$, then the following triplets are discardable: 1) $(j \in \mathcal{V}, i, -1)$; 2) $(i, j \in \mathcal{V} \setminus \{i+n\}, -1)$; 3) $(i+n, j \in \mathcal{V} \setminus \{i+2n\}$; 4) $(i+2n, j \in$*

$\mathcal{V}, -1$); *5)* $(i+n, i, k \in \mathcal{O})$; *6)* $(i+2n, i, k \in \mathcal{O})$; *7)* $(i+2n, i+n, k \in \mathcal{O})$; *8)* $(i, i+n, k \in \mathcal{O} \setminus \{-1\})$; *9)* $(i+n, i+2n, k \in \mathcal{O} \setminus \{-1\})$; *10)* $(i, i+2n, k \in \mathcal{O})$.

Proof. 1) to 4): by definition object -1 can only be carried between two consecutive copies of the same vertex, i.e., from i to $i+n$ and from $i+n$ to $i+2n$; 5): by definition $i \prec i+n$; 6): by definition $i \prec i+2n$; 7): by definition $i+n \prec i+2n$; 8) and 9): by definition object type -1 is the only available object that can be carried between two consecutive copies of the same vertex; 10): i and $i+2n$ cannot be connected by any arc since $i \prec i+n \prec i+2n$. □

Proposition 7. *If $i \in V_1$ and $a_i \neq b_i$, then the following triplets are discardable: 1)* $(j \in \mathcal{V}, i, a_i)$; *2)* $(j \in \mathcal{V}, i+n, a_i)$; *3)* $(j \in \mathcal{V}, i+2n, a_i)$; *4)* $(i+n, j \in \mathcal{V}, b_i)$; *5)* $(i+2n, j \in \mathcal{V}, b_i)$; *6)* $(i, j \in \mathcal{V}, b_i)$.

Proof. 1) to 5): by Theorem 1 the only objects that can be carried to an original vertex are the null object, the demand object or a dropped object, therefore when $a_i \neq b_i$ there exists an optimal solution in which the supply object is never carried to any of the copies of that original vertex; 6): by Theorem 1 again, there exists an optimal solution in which the vehicle can only carry from an original vertex the null object, the supply object or an object that has been dropped previously; therefore the demand object never exits any of the copies of that original vertex. □

Now consider the depot and its copies. There are two cases depending on the supply and the demand object (Propositions 8 and 9).

Proposition 8. *If $a_1 = 0$ or $b_1 = 0$ or both, then the following triplets are discardable: 1)* $(j \in \mathcal{V}, 1, k \in \mathcal{O} \setminus \{b_1\})$; *2)* $(1, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{-1\})$; *3)* $(j \in \mathcal{V} \setminus \{1\}, 1+n, k \in \mathcal{O})$; *4)* $(1+n, j \in \mathcal{V} \setminus \{1+2n\}, k \in \mathcal{O})$; *5)* $(j \in \mathcal{V}, 1+2n, k \in \mathcal{O} \setminus \{-1\})$; *6)* $(1+2n, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{a_1\})$.

Proof. 1): if a_i or b_i is the null object, then by Theorem 1 no additional deadheading can be incident to the original depot, and there is no drop at the depot. Then there exists an optimal solution in which the original depot is visited only once. Therefore the demand object must be carried to the first copy; 2) and 3): by definition, when the original vertex is visited only once, object -1 must be carried from i to $i+n$; 4) and 5): by definition, when the original vertex is visited only once, object -1 must be carried from $i+n$ to $i+2n$; 6): since there is no null object nor a dropped object entering the original vertex, the only available object at the last copy is the supplied object. □

Proposition 9. *If $a_1 = b_1$ and $a_1 \neq 0$, or if $a_1 \neq b_1$, then the following triplets are discardable: 1)* $(j \in \mathcal{V}, 1, k \in \mathcal{O} \setminus \{0, b_1\})$; *2)* $(1, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{-1, 0, a_1\})$;

3) $(j \in \mathcal{V}, 1+n, k \in \mathcal{O} \setminus \{-1, 0, b_1\})$*; 4)* $(1+n, j \in \mathcal{V} \setminus \{1+2n\}, k \in \mathcal{O})$*; 5)* $(j \in \mathcal{V}, 1+2n, k \in \mathcal{O} \setminus \{-1\})$*; 6)* $(1+2n, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{0, a_1\})$.

Proof. By Theorem 1 there is no drop at the original depot. Therefore, there exists an optimal solution in which the original depot is visited at most twice, where the two visits case corresponds to an additional pair of deadheadings (incoming and outgoing). Then we deduce that: 1): only a deadheading or an arc carrying the demand object can enter the first copy; 2): object -1 is carried from 1 to $1+n$ when the original vertex is visited only once, otherwise the two possibilities are a deadheading or an arc carrying the supply object; 3): similar to 2) but consider the demand instead of the supply because it is an incoming arc; 4) and 5): since the original vertex is visited at most twice, by definition $1+n$ and $1+2n$ are connected by an arc carrying object -1; 6): since there is no drop at the original depot, the only available objects at the last copy are the null object or the supply object. □

We now enumerate the discardable triplets related to a non-depot vertex. There are four cases (Propositions 10, 11, 12 and 13).

Proposition 10. *If* $a_i = b_i = 0$*,* $i \in V_1 \setminus \{1\}$*, then the following triplets are discardable: 1)* $(j \in \mathcal{V}, i, 0)$*; 2)* $(i, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{0\})$*; 3)* $(j \in \mathcal{V}, i+n, k \in \mathcal{O} \setminus \{0\})$*; 4)* $(i+n, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{-1\})$*; 5)* $(j \in \mathcal{V}, i+2n, k \in \mathcal{O} \setminus \{-1\})$*; 6)* $(i+2n, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{0\})$.

Proof. By Theorem 1 and Proposition 2 there exists an optimal solution in which an original vertex is either visited twice or not visited at all.
1): if the vehicle carries the null object to the first copy, then there will be two consecutive deadheadings because $a_i = b_i = 0$ (see Proposition 4); 2): the null object (i.e., the supply object here) is the only available type at the first copy; 3): similar to 2) but consider the supply instead of the demand (they are actually the same here) since it is an incoming arc; 4) and 5): since the original vertex is visited at most twice, by definition the last two copies are connected with an arc carrying object -1; 6): since the original vertex cannot be visited only once, the supply object (i.e., the null object) cannot be carried from the last copy. Only a dropped object, if any, can be available at that copy. □

Proposition 11. *If* $a_i = 0$ *or* $b_i = 0$ *(but not both),* $i \in V_1 \setminus \{1\}$*, then the following triplets are discardable: 1)* $(i, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{-1, a_i\})$*; 2)* $(j \in \mathcal{V}, i+n, k \in \mathcal{O} \setminus \{-1, b_i\})$*; 3)* $(i+n, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{-1\})$*; 4)* $(j \in \mathcal{V}, i+2n, k \in \mathcal{O} \setminus \{-1\})$.

Proof. 1): the null object cannot be carried from the first copy because this would imply two consecutive deadheadings since a_i or b_i is 0 (see Proposition 4); 2): similar to 1) but consider the demand instead of the supply; 3) and 4): since $a_i = 0$

or $b_i = 0$, there are no additional deadheading among the incident arcs. Therefore, by Theorem 1, the original vertex is visited at most twice, and then by definition, the last two copies are connected by an arc carrying object -1. □

Proposition 12. *If $a_i = b_i$ and $a_i \neq 0$, $i \in V_1 \setminus \{1\}$, then the following triplets are discardable: 1) $(i, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{0, a_i\})$; 2) $(j \in \mathcal{V}, i+n, k \in \mathcal{O} \setminus \{0, b_i\})$; 3) $(i+n, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{0, a_i\})$; 4) $(j \in \mathcal{V}, i+2n, k \in \mathcal{O} \setminus \{-1, 0, b_i\})$.*

Proof. 1): by Theorem 1, if there is a drop at a vertex, then the dropped object is reloaded during the last exit (i.e., at the third copy). Therefore the only available objects at the first copy are the null object and the supply object; 2): similar to 1) but consider the demand instead of the supply since it is an incoming arc; 3) similar to 1), the only available object at $i+n$ are 0 or a_i since a dropped object, if any, can only be reloaded at $i + 2n$; 4): by Theorem 1 and by definition of the triplication, a drop can only occur at the first copy. Then any arc entering the last copy must carry -1, 0 or the demand object. □

Proposition 13. *If $a_i \neq b_i$ and $a_i \neq 0$, $i \in V_1 \setminus \{1\}$, then the following triplets are discardable: 1) $(i, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{-1, 0, a_i\})$; 2) $(i+n, j \in \mathcal{V}, k \in \mathcal{O} \setminus \{-1, 0, a_i\})$; 3) $(j \in \mathcal{V}, i+n, k \in \mathcal{O} \setminus \{-1, 0, b_i\})$; 4) $(j \in \mathcal{V}, i+2n, k \in \mathcal{O} \setminus \{-1, 0, b_i\})$.*

Proof. 1) and 2): by Theorem 1 a dropped object is reloaded during the last exit, i.e., at the last copy. Therefore only -1, 0 or a_i can be carried from i; 3): similar to 1) but consider the demand instead of the supply object; 4): similar to 3) for the third copy. □

3 Mathematical model

This section presents a mixed integer linear program for the preemptive SP. The allowable triplet configurations described in Section 2 are handled through the variables and the constraints. To each triplet $(i, j, k) \in \mathcal{N}$, is associated a binary variable x_{ij}^k equal to 1 if and only if an object of type k is carried from i to j, and to each vertex $i \in \mathcal{V}$ is associated a real variable u_i indicating the position of i in the route ($u_0 = 0$ and $1 \leq u_i \leq 3n-1$, $i = 2, \ldots, 3n$). The u_i variables are similar to those introduced by Miller, Tucker and Zemlin (MTZ) in their TSP formulation ([26]). The possible optimal configurations for a vertex depend on its supply and its demand. The constraints relative to each case are presented separately. The idea is to restrict the search for a solution to the subspace of solutions that satisfy optimal structural properties. Every variable representing a discardable triplet must be interpreted as zero.

3.1 Depot vertex with $a_1 = 0$ or $b_1 = 0$

By Theorem 1, objects cannot be dropped at the depot, and for any original vertex the demand (supply) object is carried only once to (from) that vertex. Since $a_1 = 0$ or $b_1 = 0$ and because of Proposition 4, the depot is visited only once (constraints 1 and 4). Therefore after triplication the three copies are connected by two arcs carrying object -1 as shown in Figure 8 (constraints 2 and 3). The corresponding constraints are

$$\sum_{j \in \mathcal{V}} x_{j1}^{b_1} = 1 \quad (1)$$

$$x_{1,1+n}^{-1} = 1 \quad (2)$$

$$x_{1+n,1+2n}^{-1} = 1 \quad (3)$$

$$\sum_{j \in \mathcal{V}} x_{i+2n,j}^{a_1} = 1. \quad (4)$$

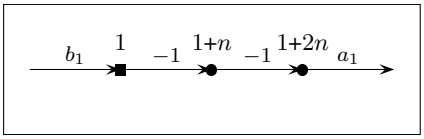

Figure 8: Configuration for the depot with $a_1 = 0$ or $b_1 = 0$

3.2 Depot vertex with $a_1 \neq 0$ and $b_1 \neq 0$

Since neither a_1 nor b_1 is the null object, a pair of deadheadings (incoming and outgoing) can possibly be used among the arcs incident to the three copies of the depot. By Proposition 4, there exist only two ways of visiting the original depot twice. This leads to the three possible configurations depicted in Figure 9. Constraints 5 and 6 ensure that the demand and the supply are satisfied. Since the original depot is visited at most twice, its second and third copies are connected by an arc carrying object -1 (constraint 7). Constraint 8 takes care of the conservation of the null object, which must be reloaded at the third copy if it has been dropped at the first copy (this is similar to the drop of a real object). The corresponding constraints are

$$\sum_{j \in \mathcal{V}} \left(x^{b_1}_{j1} + x^{b_1}_{j,1+n} \right) = 1 \tag{5}$$

$$\sum_{j \in \mathcal{V}} \left(x^{a_1}_{1j} + x^{a_1}_{1+2n,j} \right) = 1 \tag{6}$$

$$x^{-1}_{1+n,1+2n} = 1 \tag{7}$$

$$\sum_{j \in \mathcal{V}} \left(x^{0}_{j1} - x^{0}_{1+2n,j} \right) = 0. \tag{8}$$

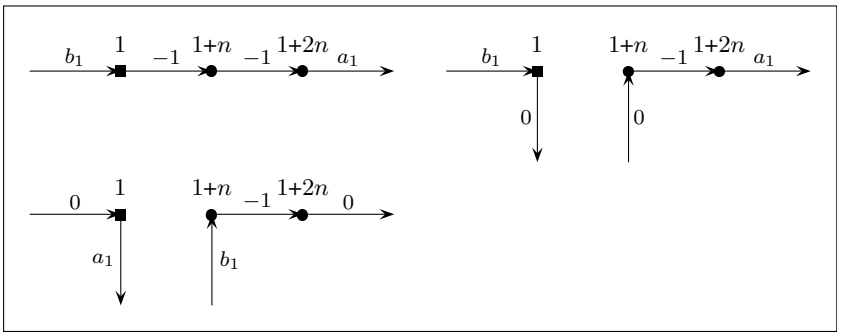

Figure 9: Configurations for the depot with $a_1 \neq 0$ and $b_1 \neq 0$

3.3 Non-depot vertex with $a_i = b_i = 0$

Since $a_i = b_i$, the original vertex is a transshipment vertex and the vehicle can possibly skip it during its tour (the demand is already satisfied by the supply). Since both a_i and b_i are the null object, no additional deadheading can be incident to the three copies of the vertex, except the two arcs that carry the supply and the demand themselves, which can be carried from the first copy and to the second copy, respectively. Furthermore, by Theorem 1, if there is a drop at vertex i, it occurs during the first entry and the reloading takes place during the last exit, yielding constraints 9 and 11. Since the original vertex is visited at most twice, the second and the third copies must be connected by an arc carrying object -1, which only occurs when the second copy has received its demand (constraints 10). Figure 10 represents the two possible configurations. Note that the vehicle cannot visit the original vertex only once because it is a transshipment vertex (see Proposition 2). We consider the following set of constraints: $\forall\, i \in \left\{ j \in V_1 \setminus \{1\} \,\middle|\, a_j = b_j = 0 \right\}$,

$$\sum_{k\in\mathcal{O}}\sum_{j\in\mathcal{V}} x_{ji}^{k} - \sum_{j\in\mathcal{V}} x_{ij}^{a_i} = 0 \tag{9}$$

$$\sum_{j\in\mathcal{V}} x_{j,i+n}^{b_i} - x_{i+n,i+2n}^{-1} = 0 \tag{10}$$

$$\sum_{j\in\mathcal{V}} \left(x_{ji}^{k} - x_{i+2n,j}^{k} \right) = 0 \qquad (k = 1, \ldots, m). \tag{11}$$

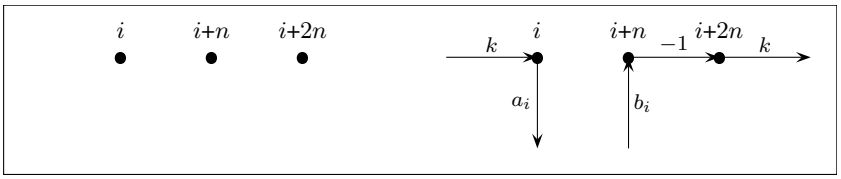

Figure 10: Configurations for a non-depot vertex with $a_i = b_i = 0$

3.4 Non-depot vertex with $a_i \neq b_i$, $a_i = 0$ or $b_i = 0$

The supply of vertex i differs from its demand, so the vehicle must necessarily carry b_i (a_i) to (from) the original vertex, which is visited at most twice because there is no additional deadheading among the incident arcs (since a_i or b_i is the null object). The demand can be carried to either the first copy (when the original vertex is visited only once) or to the second copy (when there is a drop) (constraints 12), and the supply can exit the first copy or the third copy (constraints 13). If the vehicle carries the demand to the first copy, then the original vertex is visited only once and the first two copies must be connected by an arc carrying object -1 (constraints 14). Furthermore, since the original vertex cannot be visited more than twice, then the last two copies are also connected by an arc carrying object -1 (constraints 15). By Theorem 1, if there is a drop it must occur during the first entry and the reloading occurs during the last exit (constraints 16). The two possible configurations are shown in Figure 11. We consider the following set of constraints: $\forall\, i \in \left\{ j \in V_1 \setminus \{1\} \,\middle|\, a_j = 0 \text{ or } b_j = 0 \right\}$,

$$\sum_{j\in\mathcal{V}}\left(x_{ji}^{b_i}+x_{j,i+n}^{b_i}\right) = 1 \tag{12}$$

$$\sum_{j\in\mathcal{V}}\left(x_{ij}^{a_i}+x_{i+2n,j}^{a_i}\right) = 1 \tag{13}$$

$$\sum_{j\in\mathcal{V}} x_{ji}^{b_i} - x_{i,i+n}^{-1} = 0 \tag{14}$$

$$x_{i+n,i+2n}^{-1} = 1 \tag{15}$$

$$\sum_{j\in\mathcal{V}}\left(x_{ji}^{k}-x_{i+2n,j}^{k}\right) = 0 \qquad (k=1,\ldots,m). \tag{16}$$

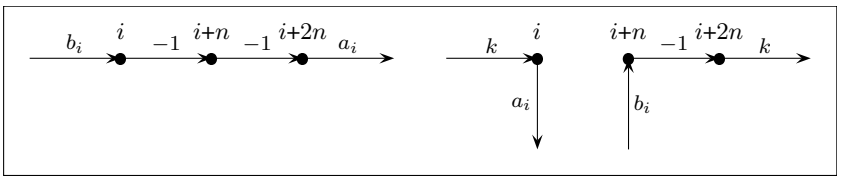

Figure 11: Configurations for a non-depot vertex with $a_i = 0$ or $b_i = 0$

3.5 Non-depot vertex with $a_i = b_i$ and $a_i \neq 0$

Here we handle a more complicated case because the vehicle may visit the original vertex three times due to the fact that the supply and demand object differ from the null object. This situation induces additional deadheading. Also note that since $a_i = b_i$ (i.e., i is a transshipment vertex), the original vertex is not necessarily visited in the solution. When the vehicle visits the original vertex, then there exist three alternatives if it visits the original vertex twice, and only two if it visits the original vertex three times (see Figure 12). Again these alternatives come from the fact that there exists an optimal solution that does not contain two consecutive arcs carrying the same object type (Proposition 4). Constraints 17 ensure that any drop occurs at the first copy of the vertex, and the reloading occurs at third copy (Theorem 1). Constraints 18, 19 and 20 ensure that the supply and the demand are satisfied and take care of the conservation of additional deadheading. We consider the following constraints: $\forall\, i \in \left\{ j \in V_1 \setminus \{1\} \,\middle|\, a_j = b_j, a_j \neq 0 \right\}$,

$$\sum_{j\in\mathcal{V}}\left(x^k_{ji} - x^k_{i+2n,j}\right) = 0 \qquad (k = 0,\ldots,m) \tag{17}$$

$$\sum_{j\in\mathcal{V}}\left(x^0_{ij} - x^0_{j,i+n}\right) = 0 \tag{18}$$

$$\sum_{j\in\mathcal{V}}\left(x^{a_i}_{ij} - x^{b_i}_{j,i+n}\right) = 0 \tag{19}$$

$$\sum_{j\in\mathcal{V}}\left(x^{a_i}_{i+n,j} - x^{b_i}_{j,i+2n}\right) = 0. \tag{20}$$

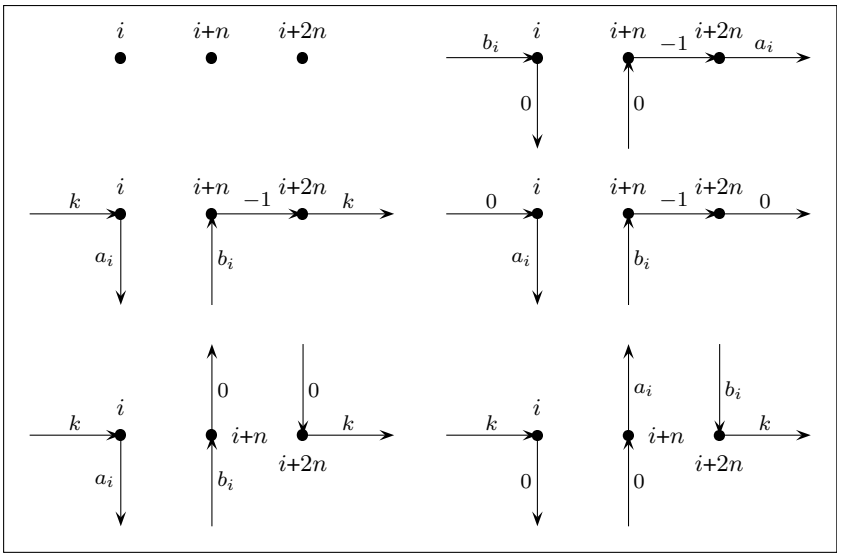

Figure 12: Configurations for a non-depot vertex with $a_i = b_i$ and $a_i \neq 0$

3.6 Non-depot vertex with $a_i \neq b_i$ and $a_i \neq 0, b_i \neq 0$

This case is the most common: it corresponds to the situation where the demand is different from the supply, and both are non-zero. There exists an additional possible configuration with respect to the case described in Section 3.5 because $a_i \neq b_i$. Indeed, when the original vertex is visited three times, there may exist a route in which the vehicle carries the demand to the second copy and immediately loads

the supply. This could not happen in the case of Section 3.5 because both the demand and the supply were identical. As a consequence the demand (supply) object can be carried to (from) any of the three copies. This yields the seven possible optimal configurations represented in Figure 13. We consider the following set of constraints: $\forall\, i \in \left\{ j \in V_1 \setminus \{1\} \,\middle|\, a_j \neq b_j, a_j \neq 0, b_j \neq 0 \right\}$,

$$\sum_{j \in \mathcal{V}} \left(x_{ji}^{b_i} + x_{j,i+n}^{b_i} + x_{j,i+2n}^{b_i} \right) = 1 \tag{21}$$

$$\sum_{j \in \mathcal{V}} \left(x_{ij}^{a_i} + x_{i+n,j}^{a_i} + x_{i+2n,j}^{a_i} \right) = 1 \tag{22}$$

$$\sum_{j \in \mathcal{V}} \left(x_{ji}^{b_i} - x_{i+2n,j}^{a_i} \right) = 0 \tag{23}$$

$$\sum_{j \in \mathcal{V}} x_{ji}^{b_i} - x_{i+n,i+2n}^{-1} \leq 0 \tag{24}$$

$$\sum_{j \in \mathcal{V}} \left(x_{ji}^{b_i} - x_{ij}^{0} \right) - x_{i,i+n}^{-1} \leq 0 \tag{25}$$

$$\sum_{j \in \mathcal{V}} \left(x_{ij}^{a_i} - x_{j,i+n}^{b_i} \right) \leq 0 \tag{26}$$

$$\sum_{j \in \mathcal{V}} x_{j,i+n}^{b_i} - \sum_{j \in \mathcal{V}} \left(x_{ij}^{a_i} + x_{i+n,j}^{a_i} \right) \leq 0 \tag{27}$$

$$\sum_{j \in \mathcal{V}} x_{i+n,j}^{a_i} - \sum_{j \in \mathcal{V}} \left(x_{j,i+2n}^{0} + x_{j,i+2n}^{b_i} \right) \leq 0 \tag{28}$$

$$\sum_{j \in \mathcal{V}} \left(x_{ji}^{k} - x_{i+2n,j}^{k} \right) \leq 0 \qquad (k = 0, \ldots, m). \tag{29}$$

Constraints 21 and 22 ensure that the demand is satisfied and the supply is carried to another vertex. By Theorem 1, if there is a drop, it occurs at vertex $i \in V_1$ and the dropped object is reloaded at vertex $i + 2n$. Therefore, if the vehicle carries b_i to vertex i, then there is no drop at i and the only available object at vertex $i + 2n$ is the supply object (constraints 23). If there is no drop, then the original vertex must be visited at most twice (Theorem 1) and, by definition, vertices $i + n$ and $i + 2n$ are connected by an arc carrying -1 (constraints 24). If the demand b_i is carried to vertex i, then there are two ways of exiting i: carrying the null object or carrying object -1 to vertex $i + n$ (constraints 25). When a_i is carried from vertex i, then b_i must be carried to vertex $i + n$, otherwise there would be two consecutive arcs carrying the same object type which is non-optimal

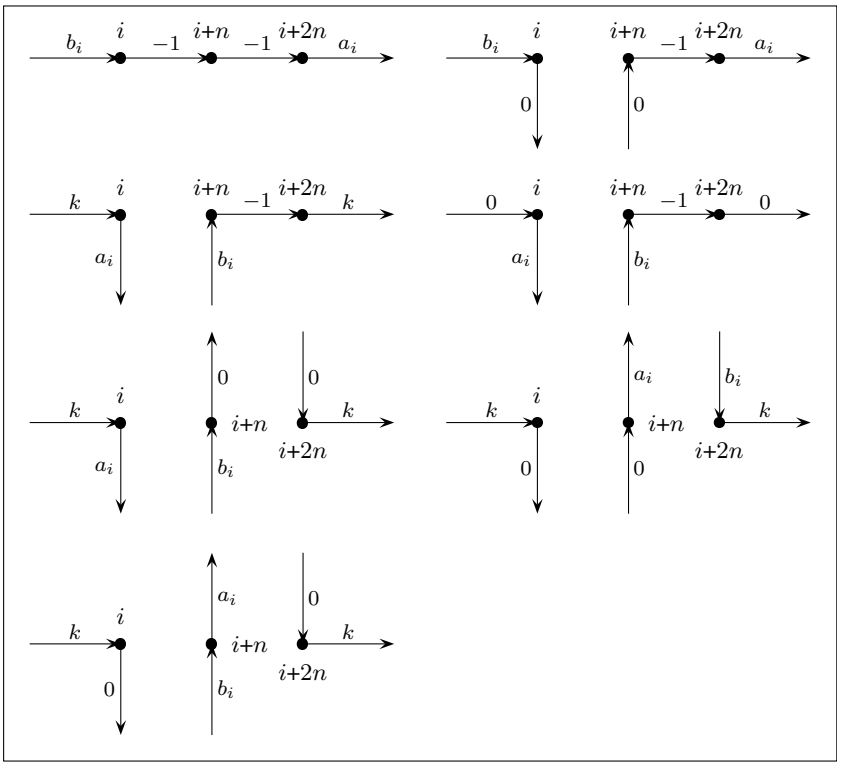

Figure 13: Configurations for a non-depot vertex with $a_i \neq b_i$ and $a_i \neq 0, b_i \neq 0$

by Proposition 4 (constraints 26). Constraints 27 are similar to 26 but apply to vertex $i + n$. If the vehicle carries b_i to vertex $i + n$, then in order to avoid two consecutive deadheadings, object a_i must exit from i or $i + n$. If the vehicle exits from $i + n$ with object a_i, then this implies that the original vertex is visited three times. Therefore an object has been dropped at the first copy and must be reloaded at the last copy. Again to avoid two consecutive arcs carrying the same object type (see Proposition 4), the vehicle must carry either 0 or b_i to vertex $i + 2n$ (constraints 28). Finally constraints 29 ensure that any object dropped at vertex i is reloaded at vertex $i + 2n$ (Theorem 1).

3.7 Degree constraints

The three copies of the depot and every non-transshipment vertex have an in-degree and an out-degree equal to one. Therefore we consider the following degree constraints: $\forall\, i \in \mathcal{V} \setminus \big\{ j \in \mathcal{V} \setminus \{1, 1 + n, 1 + 2n\} \,\big|\, a_j \neq b_j \big\}$,

$$\sum_{k\in\mathcal{O}}\sum_{j\in\mathcal{V}} x^k_{ji} = 1 \tag{30}$$

$$\sum_{k\in\mathcal{O}}\sum_{j\in\mathcal{V}} x^k_{ij} = 1. \tag{31}$$

For transshipment vertices (which are not necessarily part of an optimal solution) we consider the general conservation constraints which ensure that the in-degree is equal to the out-degree (this is also true for non-transshipment vertices but it is implied by constraints 30 and 31): $\forall\, i \in \left\{ j \in \mathcal{V} \,\middle|\, a_j = b_j \right\}$,

$$\sum_{k\in\mathcal{O}}\sum_{j\in\mathcal{V}} \left(x^k_{ji} - x^k_{ij} \right) = 0. \tag{32}$$

If one copy of a transshipment vertex is visited, then the other copies must be visited as well (again this is also true for non-transshipment vertices but it is implied by constraints 30 and 31). This can be expressed by the following constraints: $\forall\, i \in \left\{ j \in V_1 \,\middle|\, a_j = b_j \right\}$,

$$\sum_{k\in\mathcal{O}}\sum_{j\in\mathcal{V}} \left(x^k_{ji} - x^k_{j,i+n} \right) = 0 \tag{33}$$

$$\sum_{k\in\mathcal{O}}\sum_{j\in\mathcal{V}} \left(x^k_{j,i+n} - x^k_{j,i+2n} \right) = 0. \tag{34}$$

3.8 Subtour elimination constraints

Standard subtour elimination constraints (SEC) for the directed TSP state that in any optimal solution the vehicle must leave (and equivalently enter) every proper subset of vertices at least once. As we have seen, in the preeemptive SP the non-depot vertices whose supply and demand are the same may or may not be part of an optimal solution. Therefore standard SEC must be slightly modified. Actually, subtour elimination constraints 35 are the same as in the directed TSP, but the subset of vertices to which they apply must satisfy additional requirements.

Definition 4. *A subset $U \subset \mathcal{V}$ is called SEC-compatible if it is valid to impose an SEC on U. Denote by $\mathcal{U}$ the family of SEC-compatible subsets.*

Proposition 14. *$U \in \mathcal{U}$ if and only if it satisfies one of the two conditions: 1) at least one copy of the depot is in U, and at least one copy of the depot is not in U*

or there exists at least one non-transshipment vertex not in U; 2) at least one copy of the depot is not in U, and at least one copy of the depot is in U or there exists at least one non-transshipment vertex in U.

Proof. $\Rightarrow$ Let $U \subset \mathcal{V}$. The vertices that are visited in all solutions are the three copies of the depot and the non-transshipment vertices. By Definition 4, it is feasible to impose an SEC on U, which means that any feasible circuit must exit U at least once. This implies that the vehicle must visit at least one vertex in U and at least another vertex not in U. There are two cases. In the first one $\exists\, i \in \{1, 1+n, 1+2n\}$ in U and $\exists\, j \in \{1, 1+n, 1+2n\}$ not in U ($i \neq j$). In the second one $\exists\, i \in \{1, 1+n, 1+2n\}$ in U and $\exists\, j \in \mathcal{V} \setminus \{1, 1+n, 1+2n\}$ such that $a_j \neq b_j$. This gives us the first condition of the proposition. The second condition follows by considering the complement of U.
$\Leftarrow$ Suppose $\exists\, i \in \{1, 1+n, 1+2n\}$ in U and $\exists\, j \in \{1, 1+n, 1+2n\}$ not in U. Since any feasible solution visits the three copies of the depot, this implies that the vehicle must exit U at least once, which corresponds to an SEC on U. Similarly, suppose $\exists\, i \in \{1, 1+n, 1+2n\}$ in U and $\exists\, j \in \mathcal{V} \setminus \{1, 1+n, 1+2n\}$ such that $a_j \neq b_j$. Since the three copies of the depot are visited by any solution and every non-transshipment vertex must also be visited (in order to satisfy its demand and its supply), the vehicle must exit U at least once, which corresponds to an SEC on U. The second condition of the proposition follows by considering the complement of U. □

Subtour elimination constraints for the preemptive SP can be written as follows: $\forall\, U \in \mathcal{U}$,

$$\sum_{k \in \mathcal{O}} \sum_{i \in U} \sum_{j \notin U} x_{ij}^k \quad \geq \quad 1. \tag{35}$$

3.9 u-precedence constraints

The u_i variable indicates the position of vertex i in the tour. Since $i \prec i+n$ and $i+n \prec i+2n$, every feasible solution must satisfy the following constraints: $\forall\, i \in V_1$,

$$u_{i+n} - u_i \quad \geq \quad 1 \tag{36}$$

$$u_{i+2n} - u_{i+n} \quad \geq \quad 1. \tag{37}$$

3.10 MTZ constraints

Miller et al. [26] introduced the so-called MTZ constraints for the TSP (on n vertices):

$$u_i - u_j + (n-1)x_{ij} \quad \leq \quad n-2 \qquad (i, j = 2, \ldots, n).$$

Desrochers and Laporte [15] have later shown that these constraints can be lifted as follows:

$$u_i - u_j + (n-1)x_{ij} + (n-3)x_{ji} \quad \leq \quad n-2 \qquad (i,j = 2,\ldots,n).$$

In our context we must consider $3n$ vertices and the object types. The lifted MTZ constraints become: $\forall\, i,j \in \mathcal{V} \setminus \{1\}$,

$$u_i - u_j + (3n-1)\sum_{k\in\mathcal{O}} x_{ij}^k + (3n-3)\sum_{k\in\mathcal{O}} x_{ji}^k \quad \leq \quad 3n-2. \qquad (38)$$

These constraints mean that if the vehicle travels from i to j, then u_j should be greater or equal than $u_i + 1$. They are efficient only if they are combined with the u-precedence constraints: they link the u_i to the x_{ij}^k , but not necessarily sequence the u_i variables in a feasible way at an integer solution. On the other hand, the u-precedence constraints impose precedence relationships but are not sufficient to guarantee a feasible vehicle route. However, the presence of these two sets of constraints in the model ensures that any integer solution satisfies all precedence relationships.

3.11 Comb inequalities

Comb inequalities were first identified by Chvátal [13] and then generalized by Grötschel and Padberg [19]. A comb is defined by a subset $H \subset V$, called the *handle*, and an odd number $t \geq 3$ of *teeth* $T_i \subset V$ $(i = 1,\ldots,t)$ such that (see Figure 14):

$$\begin{aligned} H \cap T_i &\neq \emptyset\,, \quad (i = 1,\ldots,t) \\ T_i \setminus H &\neq \emptyset\,, \quad (i = 1,\ldots,t) \\ T_i \cap T_j &\neq \emptyset\,, \quad (1 \leq i < j \leq t). \end{aligned}$$

For the TSP the comb inequality is expressed as:

$$x(\delta(H)) + \sum_{i=1}^{t} x(\delta(T_i)) \geq 3t+1,$$

where $\delta(S)$ is the set of edges having only one endpoint in S, and $x(S)$ represents the sum of the values of the edges having their two endpoints in S.

In the preemptive SP the graph is directed and in a solution each arc is associated with an object type. Simply replacing each arc (i,j) by an edge $e = (i,j)$ with value $x_e = \sum_{k=0}^{m}(x_{ij}^k + x_{ji}^k)$ is not sufficient to define a valid comb inequality. It is necessary to also take into account the fact that some vertices are not required in the solution (i.e., transshipment vertices). The subsets H and T_i must satisfy an additional requirement which is introduced in the next definition.

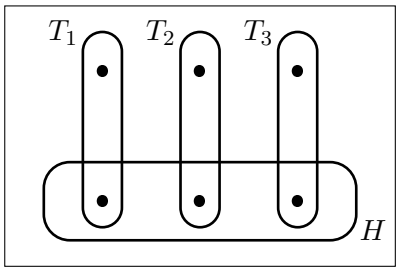

Figure 14: Minimal comb configuration

Definition 5. *A subset $H \subset V$ and an odd number $t \geq 3$ of subsets $T_i \subset V$ $(i = 1, \ldots, t)$ such that 1) $H \cap T_i \neq \emptyset$, $(i = 1, \ldots, t)$, 2) $T_i \setminus H \neq \emptyset$, $(i = 1, \ldots, t)$ and 3) $T_i \cap T_j \neq \emptyset$, $(1 \leq i < j \leq t)$, is said to be comb-compatible if it is valid to impose a comb inequality from the handle H and the teeth T_i.*

Proposition 15. *A comb (H, T_i) is comb-compatible if the following two conditions are satisfied for each T_i: 1) at least one copy of the depot is in $T_i \setminus H$ or there exists at least one non-transshipment vertex in $T_i \setminus H$; 2) at least one copy of the depot is in $T_i \cap H$ or there exists at least one non-transshipment vertex in $T_i \cap H$.*

Proof. The proof is similar to that of Proposition 14. If a vertex has the same demand and supply (i.e., a transshipment vertex) and is not a copy of the depot, the vehicle can possibly skip it in an optimal solution. Such a vertex is somehow unconstrained since its supply can satisfy its own demand. On the other hand each copy of the depot and the vertices i with $a_i \neq b_i$ must necessarily be visited by any feasible solution. Since the minimal comb configuration (see Figure 14) consists of exactly one vertex in each $T_i \setminus H$ and $T_i \cap H$, having one copy of the depot or a vertex i with $a_i \neq b_i$ in each of these subsets forces the vehicle to cross each border an appropriate number of times to satisfy the standard comb inequality. □

3.12 π-inequalities

Dropping an object at a vertex induces precedence relationships in the vehicle route. Therefore, cuts developed for other problems such as the SOP and the *asymmetric TSP with precedence constraints* (PCATSP) (see Ascheuer et al. [8], Balas et al. [10]) can be used to strengthen our formulation.

Definition 6. *For all $j \in \mathcal{V} \setminus \{1\}$, let $\pi(j)$ be the subset of vertices, excluding the depot, that must be visited before j in any feasible solution, i.e., $\pi(j) = \{i \in \mathcal{V} \setminus \{1\} \mid i \prec j\}$. The elements of $\pi(j)$ are called the predecessors of j.*

Definition 7. *Given $S \subset \mathcal{V}$, $\pi(S)$ is the set of its predecessors, i.e., $\pi(S) = \{i \in \pi(j) \mid j \in S\}$.*

The π-inequalities were introduced by Balas et al. [10] and can be expressed as follows for the PCATSP defined on a graph $G = (V, A)$ with vertex 1 as a depot: $\forall S \subset V \setminus \{1\}$,

$$x\left(S \setminus \pi(S) : \bar{S} \setminus \pi(S)\right) \quad \geq \quad 1.$$

These constraints mean that in any feasible solution there must be a path from any vertex $j \in S$ to vertex 1 that does not pass through $\pi(j)$. For the preemptive SP, due to the presence of transshipment vertices, an additional requirement must be satisfied by the subsets S to which the constraints apply.

Definition 8. *A subset $S \subset \mathcal{V} \setminus \{1\}$ is called π-compatible if it is valid to impose a π-inequality on S. Denote by Π the family of π-compatible subsets.*

Proposition 16. *A subset $S \subset \mathcal{V} \setminus \{1\} \in \Pi$ if there exists at least one non-transshipment vertex in $S \setminus \pi(S)$.*

Proof. Let $S \subset \mathcal{V}\setminus\{1\}$. Suppose that $S \in \Pi$ and $a_j = b_j, \forall j \in S\setminus\pi(S)$. Since $S\setminus \pi(S)$ contains only transshipment vertices, the inequality $x\left(S \setminus \pi(S) : \bar{S} \setminus \pi(S)\right) \geq 1$ is clearly not valid because we cannot force the vehicle to exit $S \setminus \pi(S)$ since the demand of each vertex in $S \setminus \pi(S)$ is already satisfied by its own supply. This contradicts our assumption that $S \in \Pi$. □

The π-inequalities for the preemptive SP become: $\forall\, S \in \Pi$,

$$x\left(S \setminus \pi(S) : \bar{S} \setminus \pi(S)\right) \quad \geq \quad 1. \tag{39}$$

3.13 σ-inequalities

The σ-inequalities are similar to the π-inequalities, the role of the predecessors is simply interchanged with the successors.

Definition 9. *For all $i \in \mathcal{V} \setminus \{1\}$, let $\sigma(i)$ be the subset of vertices, excluding the depot, that must be visited after i in any feasible solution, i.e., $\sigma(i) = \{j \in \mathcal{V} \setminus \{1\} \mid j \succ i\}$. The elements of $\sigma(i)$ are called the successors of i.*

Definition 10. *Given $S \subset \mathcal{V}$, $\sigma(S)$ is the set of its successors, i.e., $\sigma(S) = \{j \in \sigma(i) \mid i \in S\}$.*

The σ-inequalities were introduced by Balas et al. [10] and can be expressed as follows for the PCATSP defined on a graph $G = (V, A)$ with vertex 1 as a depot: $\forall S \subset V \setminus \{1\}$,

$$x\left(\bar{S} \setminus \sigma(S) : S \setminus \sigma(S)\right) \quad \geq \quad 1.$$

These constraints mean that in any feasible solution there must be a path from vertex 1 to any vertex $i \in S$ that does not pass through $\sigma(i)$. For the preemptive SP, due to the presence of transshipment vertices, an additional requirement must be satisfied by the subsets S to which the constraints apply.

Definition 11. *A subset $S \subset \mathcal{V} \setminus \{1\}$ is called σ-compatible if it is valid to impose a σ-inequality on S. Denote by Σ the family of σ-compatible subsets.*

Proposition 17. *A subset $S \subset \mathcal{V} \setminus \{1\} \in \Sigma$ if there exists at least one non-transshipment vertex in $\bar{S} \setminus \sigma(S)$.*

Proof. The proof is similar by symmetry to that of Proposition 16. □

The σ-inequalities for the preemptive SP become: $\forall\, S \in \Sigma$,

$$x\left(\bar{S} \setminus \sigma(S) : S \setminus \sigma(S)\right) \quad \geq \quad 1. \tag{40}$$

Note that in our problem the π- and σ-inequalities do not consider transshipment vertices because these are not necessarily part of an optimal solution. However, there also exist precedence relationships for transshipment vertices; if the vehicle visits such vertices these precedence relationships must be satisfied as well. That is the reason why we have introduced the MTZ constraints 38 (combined with the u-precedence constraints 36 and 37). Although they are much weaker than the π- and σ-inequalities because they may be inefficient at fractional solution, they apply to all vertices, even those that are not visited in the solution.

3.14 Generalized order constraints

The generalized order constraints (GOC) were introduced by Ruland [28] for the *Pickup and Delivery Problem*. Similar constraints were proposed by Balas et al. [10] under the name of *precedence cycle breaking inequalities*. These constraints are defined as follows. Let $S_1, \ldots, S_t \subset \mathcal{V} \setminus \{1\}$ be $t \geq 2$ disjoint subsets such that $S_i \cap \pi(S_{i+1}) \neq \emptyset, \forall\, i = 1, \ldots, t$, where $S_{t+1} = S_1$. The GOC is given by the inequality

$$\sum_{i=1}^{t} x(S_i) \quad \leq \quad \sum_{i=1}^{t} |S_i| - t - 1.$$

The GOC can be viewed as a tightened version of the SEC. Cordeau [14] has shown that they can be lifted by adding extra arcs to their left-hand side. Figure 15 illustrates a simple GOC configuration with $t = 3$ and $|S_i| = 2$ $(i = 1, 2, 3)$.

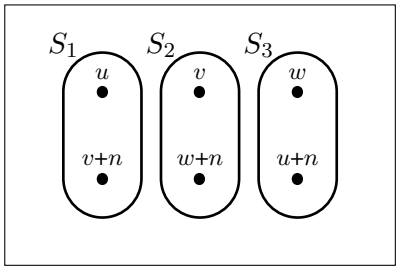

Figure 15: Example of GOC configuration

3.15 Formulation

Our model for the preemptive SP is as follows:

$$\text{minimize} \quad \sum_{(i,j,k)\in\mathcal{N}} c_{ij}x_{ij}^k$$

subject to	constraints (1)–(4)	if a_1 or $b_1 = 0$
	constraints (5)–(8)	if $a_1 \neq b_1$ and $a_1 \neq 0$
	constraints (9)–(11)	for all $i \in V_1 \setminus \{1\} \mid a_i = b_i = 0$
	constraints (12)–(16)	for all $i \in V_1 \setminus \{1\} \mid a_i$ or $b_i = 0$
	constraints (17)–(20)	for all $i \in V_1 \setminus \{1\} \mid a_i = b_i$ and $a_i \neq 0$
	constraints (21)–(29)	for all $i \in V_1 \setminus \{1\} \mid a_i \neq b_i$, $a_i \neq 0$ and $b_i \neq 0$
	constraints (30)–(31)	for all $i \in \mathcal{V} \mid a_i \neq b_i$
	constraints (32)–(34)	for all $i \in \mathcal{V}$
	constraint (35)	for all $U \in \mathcal{U}$
	constraints (36)–(37)	for all $i \in V_1$
	constraint (38)	for all $i, j \in \mathcal{V} \setminus \{1\}$
	$u_1 = 0$	
	$1 \leq u_i \leq 3n-1$	for all $i \in \mathcal{V} \setminus \{1\}$
	$x_{ij}^k \in \{0,1\}$	for all $(i,j,k) \in \mathcal{N}$.

4 Branch-and-cut algorithm

We have developed a branch-and-cut algorithm for the preemptive SP. Initially the subtour elimination constraints 35, the MTZ constraints 38 and the integrality constraints are relaxed. We denote the current linear program by LP.

Whenever the relaxation is solved, an attempt is made to detect some violated inequalities of a certain type. If some are detected, they are added to the current relaxation which is solved again. The process continues until no more violated inequalities can be identified. At this point, if the optimal LP solution satisfies the integrality constraints and all precedence relationships are satisfied, then the current solution becomes the new incumbent. Otherwise, branching is performed (see the following pseudo-code of *NodeTreatment*).

Procedure 1 *NodeTreatment*

Input: LP

1: solve LP
2: **if** violated subtour found **then**
3: separate
4: **goto** 1
5: **else if** violated combs found **then**
6: separate
7: **goto** 1
8: **else if** violated π-inequalities found **then**
9: separate
10: **goto** 1
11: **else if** violated σ-inequalities found **then**
12: separate
13: **goto** 1
14: **else if** x^* is integer **then**
15: **if** violated precedence found **then**
16: separate by adding lifted MTZ for each 1-arc
17: **goto** 1
18: **else**
19: incumbent $\leftarrow x^*$
20: **end if**
21: **else**
22: branch
23: **end if**

4.1 Separation of inequalities

We now explain the various separation procedures and the branching rules used in our algorithm.

4.1.1 Exact separation of subtour elimination constraints

To separate the SEC we use the Hao-Orlin algorithm ([20]) to compute a global minimum cut in the support graph of the current solution. If the value of the minimum cut is strictly less than 1, then we check for SEC-compatibility (see Proposition 14). When the cutset is SEC-compatible, the corresponding constraint is added to the LP which is solved again.

4.1.2 Heuristic separation of comb inequalities

Comb inequalities are normally applied to the symmetric TSP. However, adjustments must be made for the preemptive SP because the underlying graph G is directed. A solution is determined by a list of arcs, each associated with an object type. There is no known polynomial algorithm to separate general combs and this problem is most likely NP-hard. Therefore some heuristic method must be considered. In our implementation we use two different heuristics that both take as input a list of edges with their corresponding value (obtained when the LP is solved). We first transform each arc (i, j) into an edge $e = (i, j)$ with value $x_e = \sum_{k \in \mathcal{O}} (x^k_{ij} + x^k_{ji})$ and apply a comb violation heuristic: if the search is successful, then we check for comb-compatibility (see Proposition 15). When comb-compatibility is satisfied we add the corresponding comb inequality to the LP. The first heuristic comes from the publicly available package CVRPSEP developed by Lysgaard (see [24]) for the *Capacitated Vehicle Routing Problem* ([23, 25]), and the second one is our implementation of the heuristic described by Naddef and Thienel [27]. For the latter the idea is to detect a candidate for the handle first and then try to find an appropriate set of teeth by means of a growing sets technique. The two heuristics are executed subsequently and only distinct cuts are kept.

4.1.3 Exact separation of π-inequalities

To detect a violated π-inequality, we apply the exact procedure described in [10]. For every vertex $i \in V_1 \setminus \{1\}$ with $a_i \neq b_i$, we construct a new graph G'_{i+n} from the current support graph, having the same vertex set and the same arc set, except that we remove $\pi(i + n)$ (i.e., vertex i), and its incident arcs. Next, we try to send one unit of flow in G'_{i+n} from $i + n$ to the depot (see Figure 16). We actually compute a minimum s-t cut in G'_{i+n}, where $s = i + n$ and $t = 1$. If the value

of the minimum cut is strictly less than 1, then a violated π-inequality has been detected and the identified cutset is used to build the constraint. We proceed in a similar way for each vertex $i+n \in V_2 \setminus \{1+n\}$ with $a_{i+n} \neq b_{i+n}$, constructing G'_{i+2n} (this time we delete i, $i+n$ and their incident arcs) and determining the value of a maximum flow from $i+2n$ to the depot. All violated inequalities are stored and are added to the LP at the end of the procedure.

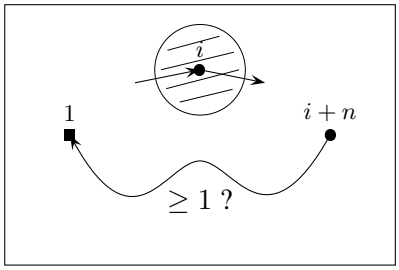

Figure 16: Illustration of the π-inequality separation procedure

4.1.4 Exact separation of σ-inequalities

The detection procedure for σ-inequality violations is similar to the one applied to π-inequalities. For each vertex $i \in V_1 \setminus \{1\}$ with $a_i \neq b_i$, we first construct a graph G'_i from the current support graph with the same vertex set and the same arc set except that we remove $i+n$, $i+2n$ and their incident arcs. Then we determine a minimum s-t cut in G'_i, where $s = 1$ and $t = i$. If the value of the minimum cut is strictly less than 1, we have found a violated σ-inequality. The same process is executed for each vertex $i+n \in V_2 \setminus \{1+n\}$ with $a_{i+n} \neq b_{i+n}$, creating the graph G'_{i+n} (by copying the support graph and deleting $i+n$ and its incident arcs), computing a minimum s-t cut, where $s = 1$ and $t = i+n$. Again if the value of the minimum cut is strictly less than 1, a violated σ-inequality has been identified and is stored. Figure 17 illustrates the main idea of the procedure.

For the calculation of the successive minimum s-t cuts we use the routine *CCcut_mincut_st* from Concorde (see [4]). It is a very fast implementation of the push-relabel flow algorithm described by Goldberg and Tarjan [18].

4.1.5 Precedence for transshipment vertices

Because we decided not to include the MTZ constraints in the LP and the π- and σ-inequalities do not consider transshipment vertices, there may exist violated precedence relationships among these vertices. Therefore at the end of the separation

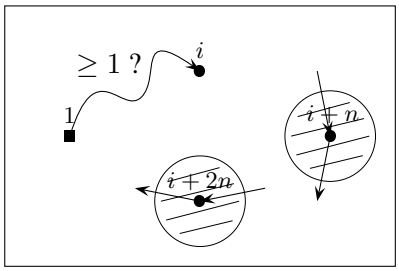

Figure 17: Illustration of the σ-inequality separation procedure

phase, if the solution is on integer circuit we check whether the precedence conditions are satisfied. For this, we follow the route starting at the depot and as soon as a violated precedence relationship is detected, for instance a vertex $i + n \in V_2$ being visited before $i \in V_1$, the process stops and a lifted MTZ constraint is added for each arc having value 1 in the current solution (see the pseudo-code of *Node-Treatment*).

4.1.6 Generalized order constraints

Separating GOC constraints with an exact method can be very time consuming even for the simplest GOC with $t = 2$ and $|S_1| = |S_2| = 2$. For this case Ruland [28] (see also Ruland and Rodin [29]) has proposed an exact separation procedure for the PDP that requires solving $O(n^2)$ maximum flow problems. In our context there are $3n$ vertices and two different precedence relationships to consider ($i \prec i + n$ and $i + n \prec i + 2n$). For the size of the instances we aim to solve ($20 \leq n \leq 100$), this involves an unacceptable number of maximum flow calculations (each separation routine may be called hundreds of times during the branch-and-cut algorithm). We have therefore opted not to incorporate the separation of this class of inequalities in our algorithm.

4.2 Branching

Three different branching rules were implemented. When the current solution is still fractional after the separation process, two child nodes are generated and added to the pool of branch-and-cut tree nodes. The *best-bound first* strategy is used to explore the pool.

4.2.1 Branching on a variable

Given the current fractional solution x^*, we select the variable closest to 0.5 and generate two child nodes by setting the value of that variable to either 0 or 1. In case of ties we choose the variable for which the associated arc is the longest. This is the most common rule applied in similar contexts. It is easy to implement and usually yields good results even if the branch-and-cut tree is not balanced. Indeed, setting a variable to 0 has much less consequence than setting it to 1 because the number of variables having value 1 in a feasible solution is much smaller than the number of variables having value 0. This standard branching is noted *rule A*.

We have implemented a second method for branching on variables, called *rule B*. This branching rule is slightly more flexible than rule A as it considers a wider range of variables. It consists of the following three steps:

1. store all fractional variables in a set C and determine the value that is the closest to 0.5 (noted μ),
2. remove from C all variables whose value is below $(1-\epsilon)\,\mu$ or above $(1+\epsilon)\,\mu$ (in our implementation we chose $\epsilon = 0.2$),
3. select among the remaining candidates the variable with the longest corresponding arc.

4.2.2 Branching on a constraint

We have also implemented a procedure for branching on a constraint. We have applied it to the SEC, but the same idea could work on other constraints such as comb inequalities. Every feasible route traverses any subset of vertices an even number of times. By traversing we mean crossing the border in the two possible directions. When we consider directed arcs, the vehicle exits and enters any proper subset that contains at least one non-transshipment vertex at most once or at least twice. If we can identify a subset $U \subset \mathcal{V}$ for which the value of the cutset $\delta^+(U)$ is close to 1.5 (where $\delta^+(U)$ represents the set of arcs with the tail in U and the head not in U), we can generate two children by enforcing $x(\delta^+(U)) \leq 1$ for the left child and $x(\delta^+(U)) \geq 2$ for the right child. In general this approach creates a branch-and-cut tree that is more balanced than when branching is performed on variables. Determining subsets on which to branch is done heuristically. We have implemented a routine that attempts to detect a subset for which the absolute difference from 1.5 is less than 0.1 (i.e., $1.4 \leq x(\delta^+(U)) \leq 1.6$). We call this branching method *rule C*. When the routine fails to find such a subset in the current fractional solution, then rule A is used.

5 Computational results

The branch-and-cut algorithm just described was coded in C++ and integrated in a branch-and-bound framework called *OOBB*, which stands for Object-Oriented Tools for Parallel Branch-and-Bound, currently under development at the CIRRELT in Montreal. Our code uses the sequential mode. As for the LP solver we used ILOG CPLEX 10.1. Tests were performed on an AMD Opteron Dual Core 285 2.6GHz running Linux.

To generate the instances, vertices were randomly distributed in the 500×500 square according to a discrete distribution. We have associated to the vertices a random supply and a random demand within $\{0, \ldots, m\}$ such that each object type was requested and supplied at least once. We tested with $m = 3, \ldots, 8$ object types and $n = 20, \ldots, 100$ vertices with an increment of 10 (for conciseness and without losing too much information an increment of 20 is considered in our tables of results). For each pair (n, m) three different instances were generated leading to a total of 162 random instances.

Table 2 shows the average computation time in seconds for different values of n and m. As we can see the running time was not affected by increasing or decreasing the number of object types considered in the instance, the number of vertices seems to be the most influent parameter.

$n \setminus m$	3	4	5	6	7	8	Seconds
20	4	3	1	10	2	2	4
40	56	350	21	15	105	19	94
60	93	435	200	498	4903	447	1096
80	873	436	2932	237	1287	1925	1282
100	4609	795	2159	1699	1557	2561	2230
Seconds	1127	404	1063	492	1571	991	

Table 2: Average computation times

Table 3 shows how the execution time was distributed during the process. The separation phase is clearly the most time-consuming step. The column heading "Other" includes the time needed to create the model, the branching procedure and miscellaneous routines inside the code. Reported values represent the average values over the 162 instances we have considered.

Table 4 shows the average number of drops used in the optimal solution, which corresponds to the number of times the vehicle has unloaded an object (different from the null object) at a vertex and reloaded it later during its tour. This number

Solving LPs	Separating Cuts	Other
38.95%	59.88%	1.17%

Table 3: Distribution of execution time

does not seem to be influenced by the number of object types in the instance and represented approximately 5% of the total number of vertices.

$n \setminus m$	3	4	5	6	7	8	Average
20	1	0	1	2	2	1	1.2
40	2	3	2	2	3	3	2.5
60	3	3	3	3	4	3	3.2
80	3	5	4	3	4	6	4.2
100	5	3	4	5	5	4	4.3
Average	2.8	2.8	2.8	3.0	3.6	3.4	

Table 4: Number of drops in an optimal solution

Tables 5 and 6 show the results of our algorithm using branching rule B. The column headings are as follows. **SEC** is the number of subtour elimination constraints added to the relaxation; **Combs** is the number of comb inequalities added to the relaxation; $\boldsymbol{\pi}$ is the number of π-inequalities added to the relaxation; $\boldsymbol{\sigma}$ is the number of σ-inequalities added to the relaxation; **MTZ** is the number of times infeasible integer solutions were found and separated by adding some lifted MTZ constraints; **Gap** is the relative deviation between the solution value $\underline{z}$ obtained at the root node and the optimal solution value z^*, i.e., $(z^* - \underline{z})/z^*$; **Nodes** is the total number of nodes in the branch-and-cut tree; **Seconds** is the running time in seconds.

The π- and σ-inequalities were very useful during the resolution. Not incorporating them in the separation phase generally increases the number of branch-and-cut nodes by a factor 3 or more, and the execution time also increases. Branching rule B, which examines a pool of candidate variables, was the most efficient of the three tested rules. Branching on constraints (rule C) occasionally reduced the number of branch-and-cut nodes but at the expense of extra computation time. All results presented in this section were obtained with branching rule B.

On this set of instances we did not find any violated precedence between transshipment vertices, which is why the MTZ column has only zero entries. This can be explained by two facts. The first is that the algorithm checks for violated prece-

dence among transshipment vertices only when the current solution is integer and integer solutions rarely make use of transshipment vertices. Another reason is that the number of transshipment vertices is in general relatively small compared to the total number of vertices. However, every solution must satisfy these relationships in order to be feasible, so they have to be checked. On some other sets of instances we sometimes found violated precedences among transshipment vertices, but only rarely.

(n, m)	SEC	Combs	π	σ	MTZ	Gap	Nodes	Seconds
(20,3)	7	1	24	3	0	0.00227	3	4
(20,4)	8	1	4	0	0	0	2	3
(20,5)	5	0	1	0	0	0	1	1
(20,6)	7	5	5	2	0	0.0001	6	10
(20,7)	5	0	2	0	0	0	1	2
(20,8)	6	0	1	0	0	0	1	2
(40,3)	37	7	13	0	0	0.00003	4	56
(40,4)	40	49	98	17	0	0.00037	22	350
(40,5)	11	0	2	1	0	0	1	21
(40,6)	4	0	2	0	0	0	1	15
(40,7)	9	6	6	4	0	0.0001	6	105
(40,8)	11	1	0	0	0	0	1	19
(60,3)	36	23	33	0	0	0	1	93
(60,4)	103	46	51	1	0	0	1	435
(60,5)	47	2	23	1	0	0	2	200
(60,6)	38	11	17	7	0	0.00003	8	498
(60,7)	32	81	129	34	0	0.0001	53	4903
(60,8)	14	19	17	7	0	0	5	447

Table 5: Detailed computational results on random instances ($20 \leq n \leq 60$)

The optimality gap at the root of the search tree is remarkably small. This yields a limited search tree exploration (less than eight nodes on average) to reach the optimal solution or to prove that the current feasible solution was optimal.

6 Note on the mixed case

The model and the branch-and-cut algorithm presented in Sections 3 and 4 for the preemptive SP can easily be modified to handle the *mixed* case where the set of

(n, m)	SEC	Combs	π	σ	MTZ	Gap	Nodes	Seconds
(80,3)	137	49	156	2	0	0.00007	2	873
(80,4)	89	3	13	0	0	0	1	436
(80,5)	138	36	64	7	0	0.00033	11	2932
(80,6)	22	1	49	0	0	0	2	237
(80,7)	33	36	71	8	0	0.00003	8	1287
(80,8)	90	41	62	7	0	0	4	1925
(100,3)	355	40	105	1	0	0	3	4609
(100,4)	51	5	39	2	0	0.00003	2	795
(100,5)	46	13	18	5	0	0	11	2159
(100,6)	66	37	19	3	0	0	4	1699
(100,7)	46	4	12	6	0	0	3	1557
(100,8)	74	11	19	10	0	0	5	2561

Table 6: Detailed computational results on random instances ($80 \leq n \leq 100$)

object types O is partitioned into a set O_d of droppable objects and a set O_n of non-droppable objects. To handle this situation, we only need to consider additional discardable triplets. In addition to the discardable triplets presented in Section 2.3 the following cases must now be considered.

Proposition 18. *If $k \in O_n$ and $b_i \neq k$, $i \in \mathcal{V}$, then the triplets $(j \in \mathcal{V}, i, k)$ are discardable.*

Proof. By definition any object type $k \in O_n$ is non-droppable so the vehicle cannot carry k to vertex i if the latter does not demand an object of that type. □

Proposition 19. *If $a_i \in O_n$ and $b_j \neq a_i$, $i, j \in \mathcal{V}$, then the triplets $(i, j \in \mathcal{V}, a_i)$ are discardable.*

Proof. This is the symmetric case of Proposition 18. By definition, if $a_i \in O_n$ the vehicle is not allowed to carry this object type to a vertex that does not demand an object of that type. □

By setting to zero the variables associated with these triplets we ensure that the vehicle will never drop a non-droppable object. The model and the branch-and-cut algorithm presented in Sections 3 and 4 remain unchanged. With this approach we optimally solved instances with up to 100 vertices in running times similar to those reported in Table 5.

7 Conclusions

We have presented the first ever exact algorithm for the preemptive Swapping Problem on a general graph. This version of the SP turns out to be much harder to solve than the non-preemptive case studied in [12]. We have designed a mathematical model based on the structural properties of optimal solutions and elaborated a branch-and-cut algorithm to solve it. To handle the precedence relationships induced by the drop of an object at a vertex, we have used a splitting technique that allowed us to incorporate the π- and σ-inequalities, the lifted MTZ and the u-precedence constraints. Computational results show that the value of the relaxation at the root of the search tree is very close to the optimal solution value. This enables us to solve reasonably large instances within acceptable computation times.

Acknowledgements This research was partially supported by the Canadian Natural Sciences and Engineering Research Council under grants 338816-05 and 39682-05. This support is gratefully acknowledged.

References

[1] S. Anily, M. Gendreau, and G. Laporte. The swapping problem on a line. *SIAM Journal on Computing*, 29:327–335, 1999.

[2] S. Anily, M. Gendreau, and G. Laporte. The preemptive swapping problem on a tree. Submitted for publication, 2006.

[3] S. Anily and R. Hassin. The swapping problem. *Networks*, 22:419–433, 1992.

[4] D.L. Applegate, R.E. Bixby, V. Chvátal, and W.J. Cook. Concorde TSP solver, 1995–. `http://www.tsp.gatech.edu/concorde.html`.

[5] N. Ascheuer. *Hamiltonian Path Problems in the On-line Optimization of Flexible Manufacturing Systems*. PhD thesis, Technische Universität Berlin, Germany, 1995.

[6] N. Ascheuer, L.F. Escudero, M. Grötschel, and M. Stoer. On identifying in polynomial time violated subtour elimination and precedence forcing constraints for the sequential ordering problem. In *Proceedings of the First Integer Programming and Combinatorial Optimization Conference*, pages 19–28. Waterloo, Canada, 1990.

[7] N. Ascheuer, L.F. Escudero, M. Grötschel, and M. Stoer. A cutting plane approach to the sequential ordering problem (with applications to job scheduling in manufacturing). *SIAM Journal on Optimization*, 3:25–42, 1993.

[8] N. Ascheuer, M. Jünger, and G. Reinelt. A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints. *Computational Optimization and Applications*, 17:61–84, 2000.

[9] M.J. Atallah and S.R. Kosaraju. Efficient solutions to some transportation problems with applications to minimizing robot arm travel. *SIAM Journal on Computing*, 17:849–869, 1988.

[10] E. Balas, M. Fischetti, and W.R. Pulleyblank. The precedence-constrained asymmetric traveling salesman polytope. *Mathematical Programming*, 68: 241–265, 1995.

[11] G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte. Static pickup and delivery problems: A classification scheme and survey. *TOP*, 15:1–31, 2007.

[12] C. Bordenave, M. Gendreau, and G. Laporte. A branch-and-cut algorithm for the non-preemptive swapping problem. Submitted for publication, 2008.

[13] V. Chvátal. Edmonds polytopes and weakly hamiltonian graphs. *Mathematical Programming*, 5:29–40, 1973.

[14] J.-F. Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54:573–586, 2006.

[15] M. Desrochers and G. Laporte. Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints. *Operations Research Letters*, 10:27–36, 1991.

[16] L.F. Escudero. An inexact algorithm for the sequential ordering problem. *European Journal of Operational Research*, 37:236–249, 1988.

[17] G.N. Frederickson and D.J. Guan. Preemptive ensemble motion planning on a tree. *SIAM Journal on Computing*, 21:1130–1152, 1992.

[18] A. Goldberg and R. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35:921–940, 1988.

[19] M. Grötschel and M.W. Padberg. On the symmetric traveling salesman problem I: Inequalities. *Mathematical Programming*, 16:265–280, 1979.

[20] J. Hao and J.B. Orlin. A faster algorithm for finding the minimum cut in a graph. In *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms*. Orlando, Florida, 1992.

[21] H. Hernández-Pérez and J.J. Salazar González. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics*, 145:126–139, 2004.

[22] H. Hernández-Pérez and J.J. Salazar González. The one-commodity pickup-and-delivery traveling salesman problem: Inequalities and algorithms. *Networks*, 50:258–272, 2007.

[23] A.N. Letchford, R.W. Eglese, and J. Lysgaard. Multistars, partial multistars and the capacitated vehicle routing problem. *Mathematical Programming*, 94:21–40, 2002.

[24] J. Lysgaard. CVRPSEP package, 2004–. http://www.hha.dk/~lys/.

[25] J. Lysgaard, A.N. Letchford, and R.W. Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100:423–445, 2004.

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

[27] D. Naddef and S. Thienel. Efficient separation routines for the symmetric traveling salesman problem I: General tools and comb separation. *Mathematical Programming*, 92:237–255, 2002.

[28] K.S. Ruland. *Polyhedral Solution to the Pickup and Delivery Problem*. PhD thesis, Sever Institute of Washington University, St. Louis, Missouri, 1994.

[29] K.S. Ruland and E.Y. Rodin. The pickup and delivery problem: Faces and branch-and-cut algorithm. *Computers and Mathematics with Applications*, 33:1–13, 1997.