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 Non-Preemptive Swapping Problem

Charles Bordenave
Michel Gendreau
Gilbert Laporte

June 2008

CIRRELT-2008-22

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 Non-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), we are given a complete graph, a set of object types and a vehicle of unit capacity. An initial state specifies the object type currently located at each vertex (at most one type per vertex). A final state describes where these object types must be repositioned. In general there exist several identical objects for a given object type, yielding multiple possible destinations for each object. The SP consists of finding a shortest vehicle route starting and ending at an arbitrary vertex, in such a way that each object is repositioned in its final state. This article exhibits some structural properties of optimal solutions and proposes a branch-and-cut algorithm based on a 0-1 formulation of the problem. Computational results on random instances containing up to 200 vertices and eight object types are reported.

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

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 #1320 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 *Swapping Problem* (SP) is defined on a complete directed graph $G = (V, A)$, 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 of m object types $O = \{1, \ldots, m\}$, also referred to as products, 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 transport a *null object* denoted by 0. The version of the SP considered in this paper is called *non-preemptive* because objects are *non-droppable*, i.e., they cannot be dropped at temporary locations along the vehicle route. Applications of the SP arise in robotics ([3]) and in printed circuit board assembly ([4]).

The SP was introduced by Anily and Hassin [2] who proved it to be 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.

A problem closely related to the non-preemptive 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. The SCP is a special case of the asymmetric TSP and also corresponds to a special case of the non-preemptive SP in which there exists only one object for each type. Frederickson et al. [8] analyzed the SCP on a complete graph and proposed a 1.8-approximation heuristic, based on matching and on minimum spanning trees. Atallah and Kosaraju [3] proved that the non-preemptive SCP can be solved in $O(m+nf(n))$ time on a line and in $O(m + n \log n)$ time on a circle, where $f(n)$ is the inverse of Ackermann's function.

Another related problem is the *Bipartite TSP* (BTSP), a TSP in which the number of vertices is even, half white, and half black. The problem consists of determining a shortest Hamiltonian tour in which no two vertices of the same color are consecutive. Chalasani and Motwani [5] have proposed a 2-approximation algorithm for the undirected version of this problem. Ghiani et al. [9] have considered a generalization of the BTSP in which the numbers of white and black vertices may differ, and the number of white vertices between two consecutive black vertices

does not exceed a prespecified value. The authors have developed a branch-and-cut algorithm for this NP-hard problem. The BTSP corresponds to a special case of the non-preemptive SP in which there exist only two object types (referred to as the *simple* SP in [2]).

Finally, we mention the *One-Commodity Pickup-and-Delivery TSP* investigated by Hernández-Pérez and Salazar González [12, 13]. In this problem each vertex has a non-negative supply or demand of a single product. The problem is to determine a shortest Hamiltonian tour for a capacitated vehicle in such a way that all requests are satisfied. This problem is NP-hard and was solved by branch-and-cut for up to 100 vertices.

Our aim is to develop, for the first time, an exact branch-and-cut algorithm for the non-preemptive SP on a general graph. The remainder of the paper is organized as follows. In Section 2 we prove some structural properties of optimal solutions. An integer linear programming formulation is presented in Section 3. Section 4 contains a description of the branch-and-cut algorithm. Computational results are presented in Section 5, followed by conclusions in Section 6.

2 Structural properties of optimal solutions

In addition to the notation already introduced, we define the binary coefficients α_{ik} and β_{ik} equal to 1 if and only if $a_i = k$ and $b_i = k$, respectively. Multiple optimal solutions may exist for a given instance, especially when the triangular inequality holds with equality. As in [2] we are interested in solutions having minimum number of arcs because they induce some interesting structural properties.

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

Lemma 1. *In any optimal solution every visited vertex i is incident to exactly one incoming arc carrying object type b_i.*

Proof. Consider an optimal solution S. Suppose there exists a vertex i for which $r \geq 2$ incoming arcs (u_t, i), $t \in \{1, ..., r\}$ carry object type b_i. Then there must exist $r-1$ arcs (i, v_t), $t \in \{1, ..., r-1\}$ exiting i with object type b_i (one object is left at i to satisfy its demand). This means that a non-negative number of b_i objects are temporarily unloaded at i and carried later to another vertex. Since the problem is non-preemptive such drops are not allowed, thus contradicting the optimality of S. □

Lemma 2. *In any optimal solution every visited vertex i is incident to exactly one outgoing arc carrying object type a_i.*

Proof. Similar to the proof of Lemma 1 by symmetry. □

Lemma 3. *In any optimal solution every vertex i satisfying one of the conditions 1) $a_i \neq 0$ and $b_i = 0$, 2) $a_i = 0$ and $b_i \neq 0$, 3) $a_i = b_i = 0$ and $i = 1$, has degree two.*

Proof. Consider a vertex i such that $a_i \neq 0$ and $b_i = 0$. The vehicle must visit i to satisfy its demand. By Lemma 1 there exists exactly one incoming arc carrying object type $b_i = 0$ (i.e., a deadheading). By Lemma 2, there exists exactly one outgoing arc carrying object type a_i. Since every object type is non-droppable, no other object can enter i and no other object can exit i. Hence i is a degree-two vertex. The second condition where $a_i = 0$ and $b_i \neq 0$ follows by symmetry. For the third condition, we can also apply Lemmas 1 and 2 on i (i.e., the depot) because it is necessarily a visited vertex. Then there exist exactly one incoming arc carrying object type $b_i = 0$ and exactly one outgoing arc carrying $a_i = 0$. Since objects are non-droppable, no other types of arc can be adjacent to i. □

Lemma 4. *In any optimal solution every vertex is incident to at most one incoming and one outgoing deadheading.*

Proof. Consider an optimal solution S and consider a vertex i visited by S. Two cases are possible: 1) Case 1: $a_i \neq 0$ and $b_i = 0$, $a_i = 0$ and $b_i \neq 0$, $a_i = b_i = 0$ and $i = 1$. By Lemma 3, i has degree two and therefore i is incident to at most one incoming and one outgoing deadheading. 2) Case 2: $a_i \neq 0$ and $b_i \neq 0$. Suppose there exist $r \geq 2$ incoming deadheadings (u_t, i), $t \in \{1, ..., r\}$. Since i does not require nor supply object type 0, there exist $r \geq 2$ outgoing deadheadings (i, v_t), $t \in \{1, ..., r\}$. By Lemmas 1 and 2, i is incident to exactly one incoming arc carrying b_i and exactly one outgoing arc carrying a_i, then $r + 1$ arcs enter and exit i. Since $r \geq 2$ there exists at least one pair of deadheadings (u_p, i) and (i, v_q), for some p and $q \in \{1, ..., r\}$, that are traversed consecutively in S and then can be replaced with a single arc (u_p, v_q). This yields a new feasible solution S' that is no worse than S and containing fewer arcs, thus contradicting the optimality of S. □

Lemma 5. *An optimal solution can possibly pass through a vertex i such that $a_i = b_i$ and different from the depot if and only if $a_i \neq 0$ (or equivalently $b_i \neq 0$). Furthermore if the solution visits such a vertex i, then i is incident to two incoming arcs (one carrying object type b_i and one deadheading) and two outgoing arcs (one carrying object type a_i and one deadheading).*

Proof. We start with the first part of the lemma.
$\Rightarrow$ Suppose there exists a vertex $i \in V \setminus \{1\}$ with $a_i = b_i = 0$ visited in an

optimal solution S. By Lemmas 1 and 2, there exist exactly one incoming arc (u, i) carrying $b_i = 0$ (i.e., a deadheading) and exactly one outgoing arc (i, v) carrying $a_i = 0$ (i.e., a deadheading). Replacing these two arcs with a unique deadheading (u, v) yields a new feasible solution S' no worse than S and containing fewer arcs, thus contradicting the optimality of S. Therefore if the vehicle can possibly visit such i, then a_i must be non-zero.
$\Leftarrow$ Consider the instance shown in Figure 1 containing four vertices located at the corners of the unit square and an additional vertex at the center.

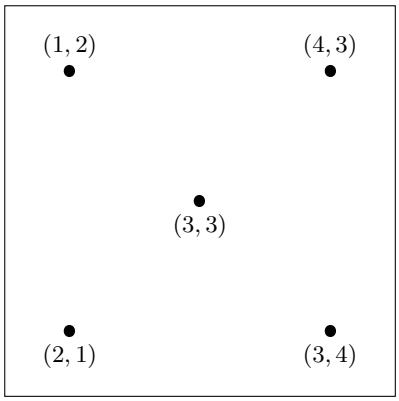

Figure 1: Instance with a vertex i such that $a_i = b_i$

Object types 1, 2 and 4 have only one source and one destination, and therefore the vehicle must carry them from their origin to their unique destination with no other choice, whereas there exist two alternatives for object type 3 which are shown in Figures 2 and 3 (one passing through the center point). To construct the corresponding solutions from these two alternatives we need to patch the two connected components together using a pair of deadheadings (Figures 4 and 5). The total cost of the first solution is 6 whereas the second one has a smaller cost equal to $3 + 2\sqrt{2}$.

To prove the second part of the lemma, consider an optimal solution S visiting a vertex $i \neq 1$ with $a_i = b_i$ and $a_i \neq 0$. Suppose there exists only one arc entering i, a deadheading (u, i). Since $b_i \neq 0$ there exists a deadheading (i, v). Since $a_i = b_i$ we can construct a new feasible solution S' by simply replacing these two arcs with a single deadheading (u, v). Again from triangular inequality S' is no worse than S and contains fewer arcs, thus contradicting the optimality of S. Suppose now that this single incoming arc (u, i) carries b_i. The vehicle must exit i with a_i, following an arc (i, v). Since $a_i = b_i$, we can again replace these two arcs with a unique arc (u, v) carrying b_i, thus contradicting the optimality of S.

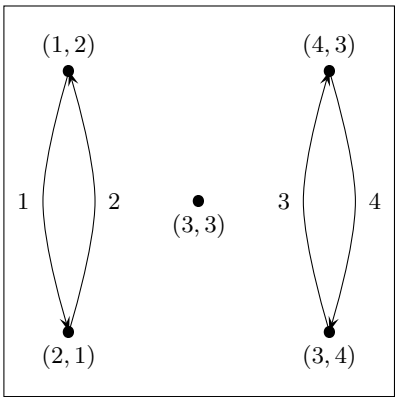

Figure 2: Alternative 1

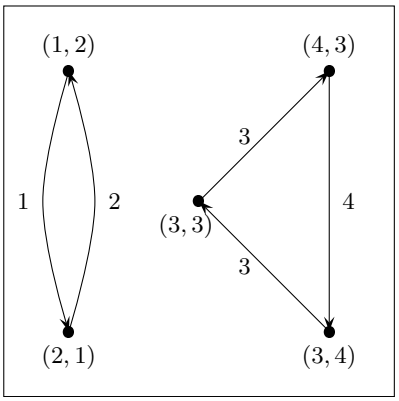

Figure 3: Alternative 2

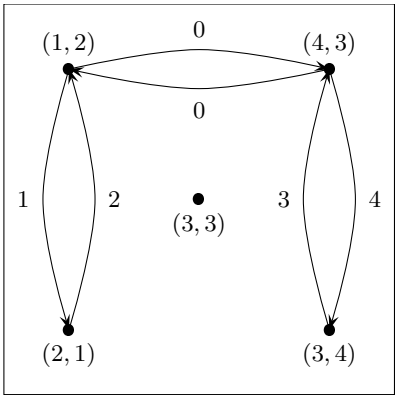

Figure 4: Solution 1

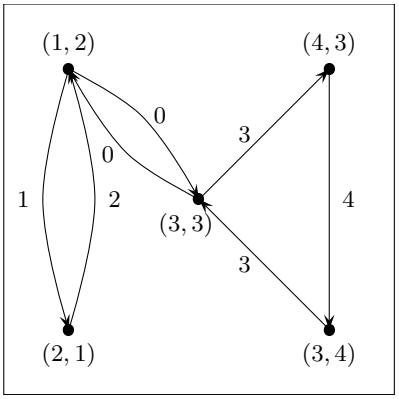

Figure 5: Solution 2

From Lemmas 1 and 4, there exist two incoming arcs (an arc carrying b_i and an incoming deadheading) and two outgoing arcs (an arc carrying a_i and an outgoing deadheading). □

Lemma 5 contradicts an observation by Anily and Hassin [2] which incorrectly states that there always exists an optimal solution for the non-preemptive SP (called pure no-drop case in their paper) that does not visit vertices for which the demand equals the supply.

Theorem 1. *In any optimal solution each vertex has degree zero, two or four.*

Proof. Lemma 5 implies that an optimal solution may or may not visit a vertex i with $a_i = b_i$. Therefore such a vertex may have a zero degree in an optimal solution. Every vertex i with $a_i \neq b_i$ must be visited by the vehicle in order to satisfy its

demand and supply. By Lemmas 1 and 2, every visited vertex is incident to exactly one incoming arc carrying b_i and exactly one outgoing arc carrying a_i, yielding a vertex of degree two. By Lemma 4, the same vertex can possibly be incident to at most one additional pair of arcs (incoming and outgoing deadheadings) so that it would become a vertex of degree four. □

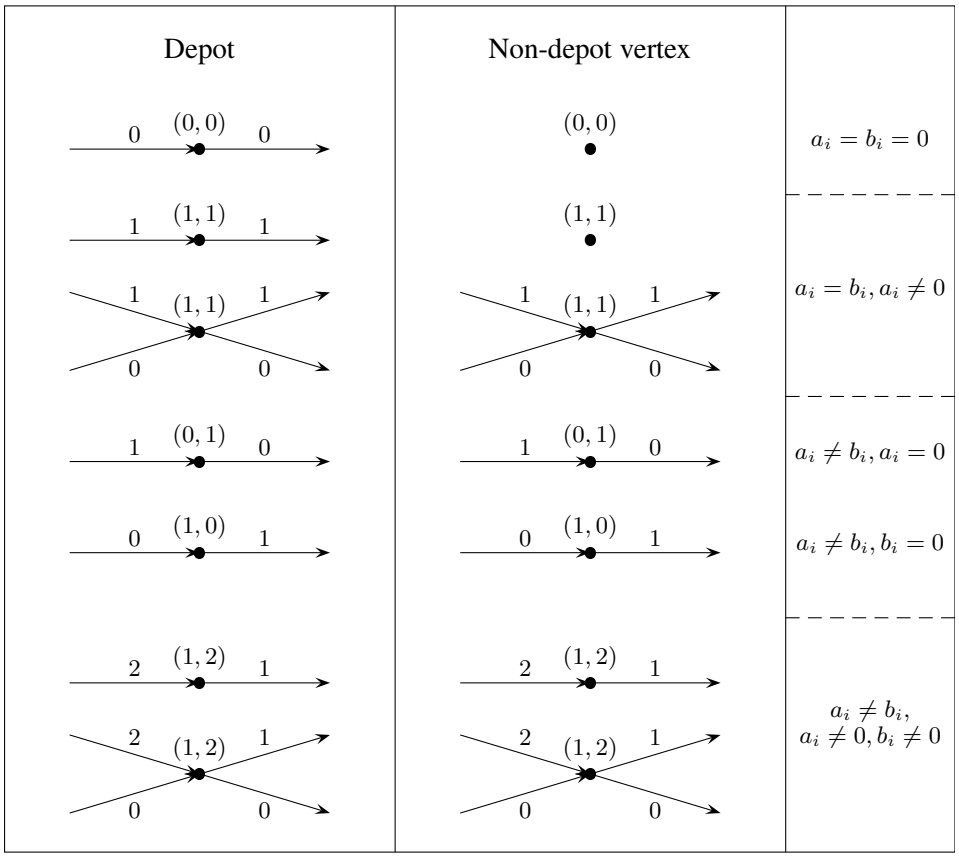

Figure 6: All possible configurations for a vertex in an optimal solution.

If vertices having the same demand and supply are disallowed (which would force the vehicle to visit each vertex at least once), then the problem can be viewed as a particular case of the *Graphical Traveling Salesman Problem* (GTSP) studied by Cornuéjols et al. [7], a variant of the TSP in which the degree constraints allow each vertex to be visited more than once. If we denote by $GTSP(n)$ and $SP(n)$

the polyhedra associated with the GTSP and this special case of non-preemptive SP respectively, then $SP(n) \subset GTSP(n)$. Therefore every known valid inequality for the GTSP is also valid for this special case of non-preemptive SP.

Definition 2. *A triplet (i, j, k), $(i, j) \in A$, $k \in O \cup \{0\}$, is called discardable if the transport of object type k from vertex i to vertex j will never be part of any optimal solution. Denote by $\mathcal{N}$ the subset of non-discardable triplets.*

Proposition 1. *A triplet (i, j, k), $(i, j) \in A$, $k \in O \cup \{0\}$ is discardable if one of the following conditions is satisfied: 1) $a_i = b_i = 0$, $i \neq 1$, 2) $a_j = b_j = 0$, $j \neq 1$, 3) $k \neq a_i$, $k \neq 0$, 4) $k \neq b_j$, $k \neq 0$, 5) $k \neq b_i$, $a_i = 0$, $b_i \neq 0$, 6) $k \neq a_i$, $a_i \neq 0$, $b_i = 0$.*

Proof. The first two conditions are direct consequences of Lemma 5. The third condition comes from the definition of the problem. Indeed, since object types are non-droppable, the vehicle is not allowed to carry an object of type k from i to j if k is not initially located at i, except possibly for the null object (i.e., (i, j) would be a deadheading). The fourth condition is the symmetric case of the third condition. If $a_i = 0$ and $b_i \neq 0$, then by Lemma 3, i has degree two in any optimal solution and then, by Lemma 1, the only object entering i is b_i. By symmetry, from Lemmas 2 and 3, if $a_i \neq 0$ and $b_i = 0$ then there exists only one arc exiting i, and this arc carries a_i. This yields the last two conditions. □

To conclude this section, we summarize in Figure 6 all possible configurations for a vertex in an optimal solution (in terms of incoming and outgoing arcs). Numerical values have been added for the sake of clarity.

3 Mathematical model

The non-preemptive SP can be formulated as an integer linear program. Any feasible solution is a subset of arcs where each one is associated with the object type carried along that arc. For all triplets $(i, j, k) \in \mathcal{N}$, let x_{ij}^k be a binary variable equal to 1 if and only if an object of type k is carried from i to j. The formulation is as follows.

$$\text{minimize} \quad \sum_{(i,j,k)\in N} c_{ij} x_{ij}^k \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} x_{ij}^{a_i} = 1 \qquad , \forall\, i \in V \text{ s.t. } a_i \neq b_i \text{ and } a_i = 0 \tag{2}$$

$$\sum_{j=1}^{n} x_{ji}^{b_i} = 1 \qquad , \forall\, i \in V \text{ s.t. } a_i \neq b_i \text{ and } b_i = 0 \tag{3}$$

$$\sum_{j=1\,|\,b_j=a_i}^{n} x_{ij}^{a_i} = 1 \qquad , \forall\, i \in V \text{ s.t. } a_i \neq b_i \text{ and } a_i \neq 0 \tag{4}$$

$$\sum_{j=1\,|\,a_j=b_i}^{n} x_{ji}^{b_i} = 1 \qquad , \forall\, i \in V \text{ s.t. } a_i \neq b_i \text{ and } b_i \neq 0 \tag{5}$$

$$\sum_{j=1\,|\,b_j=a_i}^{n} x_{ij}^{a_i} + \sum_{j=1}^{n} x_{ij}^{0} \geq 1 \qquad , a_i = b_i \text{ and } i = 1 \tag{6}$$

$$\sum_{j=1\,|\,a_j=b_i}^{n} x_{ji}^{b_i} + \sum_{j=1}^{n} x_{ji}^{0} \geq 1 \qquad , a_i = b_i \text{ and } i = 1 \tag{7}$$

$$\sum_{j=1}^{n} (x_{ij}^k - x_{ji}^k) = \alpha_{ik} - \beta_{ik} \qquad , \forall\, i \in V, \forall k \in O \cup \{0\} \tag{8}$$

$$\sum_{j=1}^{n} (x_{ji}^0 - x_{ji}^{b_i}) = 0 \qquad , \forall\, i \in V \text{ s.t. } a_i = b_i \text{ and } i \neq 1 \tag{9}$$

$$\sum_{k=0}^{m} \sum_{i\in U} \sum_{j\notin U} x_{ij}^k \geq 1 \qquad , \forall\, U \in \mathcal{U} \tag{10}$$

$$x_{ij}^k \in \{0,1\} \qquad , \forall\, (i,j,k) \in \mathcal{N}, \tag{11}$$

where $\mathcal{U}$ is a family of vertex sets to be defined in Section 3.3, and $\mathcal{N}$ is the set of possibly optimal arcs (see Definition 2). We now explain the various constraints used in the formulation.

3.1 Supply and demand constraints

If i is a vertex for which $a_i \neq b_i$, then an arc carrying a_i must leave i. There are two cases: either the supply of the given vertex is null and therefore the vehicle can carry this object to any other vertex, or the vertex has a non-zero supply and then

the supplied object type must be carried to a vertex requiring that type. In both cases, by Lemma 1, there exists exactly one outgoing arc carrying object type a_i (constraints 2 and 4). If a given vertex has the same supply and demand, then it is not certain that the vertex will be part of an optimal solution, except if the vertex is the depot (see Lemma 5). In this case, the vehicle can possibly load the supplied object and carry it to a vertex requiring that object type, and it may also leave the vertex with a deadheading (see Lemmas 2 and 4) (constraints 6). The constraints 3, 5 and 7 associated with the demand follow by symmetry.

3.2 Flow conservation constraints

Constraints 8 represent the flow conservation constraints. They are similar to the constraints proposed in [2] and take care of the conservation of objects at the vertices. Relaxing the integrality requirement may yield solutions satisfying constraints 8 but violating Lemma 5. Such a fractional solution is depicted in Figure 7. The numbers on the arcs show the object type carried and the associated x_{ij}^k values in boldface. Therefore we consider constraints 9 which ensure that if a deadheading enters a vertex $i \neq 1$ with $a_i = b_i$, then an arc carrying the demand b_i also enters i with the same flow.

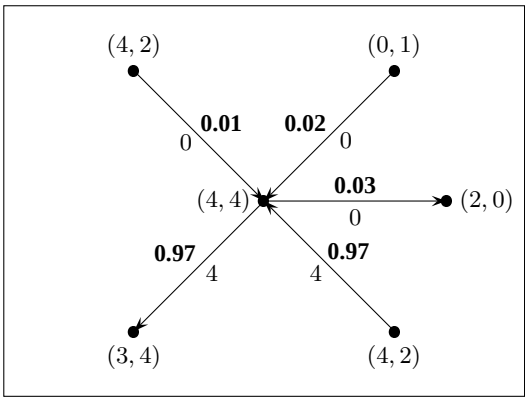

Figure 7: Part of solution satisfying constraint 8 but violating constraint 9

3.3 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 non-preeemptive SP the

vertices $i \in V \setminus \{1\}$ with $a_i = b_i$ may or may not be part of an optimal solution. Therefore standard SEC must be slightly modified. Actually, subtour elimination constraints 10 are the same as in the directed TSP but the subset of vertices to which the constraint can be applied must satisfy additional requirements.

Definition 3. *A subset $U \subset V$ 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 2. *$U \in \mathcal{U}$ if and only if it satisfies one of the two conditions 1) $1 \in U$ and $\exists\, i \in V \setminus U$ such that $a_i \neq b_i$, or 2) $1 \notin U$ and $\exists\, i \in U$ such that $a_i \neq b_i$.*

Proof. $\Rightarrow$ Let $U \in \mathcal{U}$. The vehicle must cross the border of U (in both directions) at least once. Therefore there must exist a vertex in U and a vertex in $V \setminus U$ visited in any optimal solution. Lemma 5 shows that not all vertices are necessarily part of the solution but by definition any solution passes through vertices i with $a_i \neq b_i$ (to satisfy their supply and demand) and also, by definition, any solution must visit the depot. These are the two types of vertices that are guarantied to be visited. If the depot is in U, then there exists a vertex i such that $a_i \neq b_i$ that is not in U. Otherwise, if the depot is not in U, then there exists a vertex i in U such $a_i \neq b_i$.
$\Leftarrow$ Let $U \subset V$. Suppose that the depot belongs to U and that there exists a vertex $i \in V \setminus U$ such that $a_i \neq b_i$. By definition, every feasible solution must visit the depot and also the vertex i to satisfy its supply and demand. Therefore the solution must contain an arc from U to $V \setminus U$, which is an SEC on U. Replacing U by its complementary $V \setminus U$ gives the second condition. □

3.4 Comb inequalities

Comb inequalities are not part of the basic formulation but they will be dynamically incorporated into the model during the branch-and-cut algorithm. These inequalities were first identified by Chvátal [6] and then generalized by Grötschel and Padberg [10]. A comb is defined by a *handle* $H \subset V$, and an odd number $t \geq 3$ of *teeth* $T_i \subset V$ $(i = 1, \ldots, t)$ such that (see Figure 8):

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

For the TSP the comb inequality is the following constraint:

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

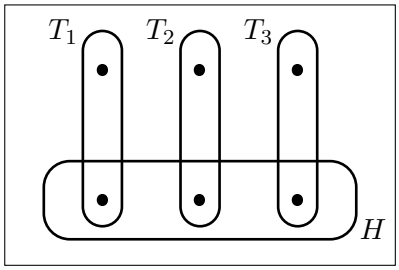

Figure 8: Minimal comb configuration

where the notation $\delta(S)$ 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 non-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 for the non-preemptive SP. We have to take into account the fact that some vertices are not necessarily required to be part of the solution (i.e., vertices i with $a_i = b_i$). The subsets H and T_i must satisfy an additional requirement which is introduced in the next definition.

Definition 4. *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 3. *A comb (H, T_i) is comb-compatible if the following two conditions are satisfied for each T_i: 1) $1 \in T_i \setminus H$ or $\exists\, i \in T_i \setminus H$ such that $a_i \neq b_i$, 2) $1 \in T_i \cap H$ or $\exists\, i \in T_i \cap H$ such that $a_i \neq b_i$.*

Proof. The proof is similar to that of Proposition 2. If a vertex has the same demand and supply and is not the depot, the vehicle can possibly skip it in an optimal solution. Such a vertex is somehow unconstrained as its supply can satisfy its own demand. On the other hand, the depot and vertices i with $a_i \neq b_i$ must be visited at least once in any solution. Since the minimal comb configuration (Figure 8) consists of exactly one vertex in each $T_i \setminus H$ and $T_i \cap H$, having 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. □

4 Branch-and-cut algorithm

We have developed a branch-and-cut algorithm for the non-preemptive SP. Initially, the subtour elimination constraints 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, it becomes the new incumbent. Otherwise, branching is performed by creating two child nodes and adding them to the node set of the branch-and-cut tree which is referred to as *NODEPOOL* (see the following pseudo-code of *NodeTreatment*).

Procedure 1 *NodeTreatment*
Input: *LP*, *NODEPOOL*

```
1:  detectedCut = true
2:  while detectedCut do
3:     while FINDSUBTOUR do
4:        detectedCut = true
5:        ADDSUBTOUR
6:        SOLVE
7:     end while
8:     if FINDCOMBS then
9:        detectedCut = true
10:       ADDCOMBS
11:       SOLVE
12:    else
13:       detectedCut = false
14:    end if
15: end while
16: if ISINT then
17:    incumbent ← x*
18: else
19:    create leftChild
20:    create rightChild
21:    NODEPOOL ← NODEPOOL ∪ {leftChild}
22:    NODEPOOL ← NODEPOOL ∪ {rightChild}
23: end if
```

To separate subtour elimination constraints we use the Hao-Orlin algorithm ([11]) 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, we check for SEC-compatibility as defined in Section 3.3. If the cutset U is SEC-compatible, then the corresponding SEC constraint is added to the LP which is solved again.

We have used two different heuristics to separate comb inequalities. The first one comes from the publicly available package CVRPSEP developed by Lysgaard ([15]) for the *Capacitated Vehicle Routing Problem* ([14, 16]), and the second one is our implementation of the heuristic proposed by Naddef and Thienel [17]. The two heuristics are executed subsequently and each new cut is stored (in our implementation we limit the number of stored cuts to 30 for a given input LP). Cuts for which comb-compatibility is satisfied (see Proposition 3) are added to the LP which is solved again.

We have applied the standard branching on variables. Given the current fractional solution x^*, we select the variable closest to 0.5. The lexicographic order is used to break ties. Then we generate two child nodes (*leftChild* and *rightChild*, see the pseudo-code of *NodeTreatment*) by fixing the value of that variable to either 0 or 1. These two new nodes are added to the pool (*NODEPOOL*) which maintains the nodes of the branch-and-cut tree as a priority stack. In our computation the best-bound search strategy was used to explore the pool. We have tested several branching rules (branching on variable with various selection rules, branching on cutset) but the standard branching rule has, in general, produced the best results.

5 Computational results

The branch-and-cut algorithm 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 in 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. Vertices $i \neq 1$ with $a_i = b_i = 0$ were not generated since it has been proved in Section 2 that they are not visited in an optimal solution. We have considered instances with $50 \leq n \leq 200$ vertices (with an increment of 10) and $3 \leq m \leq 8$ object types. For each pair (n, m) three different instances were generated, yielding a total of 96 instances. Computational

results given in this paper represent the average results over these three sets of instances. The running time limit was set to 5400 seconds.

Our results are summarized in Table 1 where we report the average number of cuts added to the relaxation, the average number of nodes in the branch-and-cut tree and the average running time in seconds. All instances have been solved to optimality, and 31% were solved at the root node. In general, instances with few object types (typically 3 or 4) were the hardest to solve. This can be explained by the fact that the number of potential destinations (for each object) increases as the number of object types decreases.

	Cuts		B&C Nodes		Seconds	
$n \setminus m$	3,4,5	6,7,8	3,4,5	6,7,8	3,4,5	6,7,8
50, 60, 70, 80	59	8	6	1	7	1
90, 100, 110, 120	158	95	8	6	79	79
130, 140, 150, 160	212	112	11	17	239	204
170, 180, 190, 200	378	205	12	7	955	580

Table 1: Summary of computational results on random instances

Table 2 gives the average relative gap at the root which is the relative difference between the value of the optimal solution and the value obtained at the first node of the branch-and-cut tree. As we can see the gap was extremely small.

$n \setminus m$	3	4	5	6	7	8
50, 60, 70, 80	0.0033	0.0011	0.0008	0.0000	0.0000	0.0000
90, 100, 110, 120	0.0006	0.0008	0.0001	0.0002	0.0001	0.0004
130, 140, 150, 160	0.0011	0.0003	0.0002	0.0006	0.0004	0.0002
170, 180, 190, 200	0.0003	0.0002	0.0002	0.0000	0.0004	0.0004

Table 2: Relative gap at the root of the branch-and-cut tree

Tables 3, 4, 5 and 6 show the results of our branch-and-cut algorithm on all instances (for each pair (n, m) the reported results correspond to the average over three different instances). The column headings are as follows. **Subtours** is the number of subtour elimination constraints added to the relaxation, **Combs** is the number of comb inequalities added to the relaxation, **Gap** is the relative gap between the solution value obtained at the root node of the search tree and the optimal solution value, **Nodes** is the total number of nodes in the branch-and-cut tree, and **Seconds** is the running time in seconds.

6 Conclusions

We have proposed the first ever exact algorithm for the non-preemptive Swapping Problem on a complete graph. We have elaborated a mathematical model based on typical arc-routing variables as well as a branch-and-cut algorithm to solve it. Computational tests show our algorithm can solve reasonably large instances to optimality 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 and R. Hassin. The swapping problem. *Networks*, 22:419–433, 1992.

[3] 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.

[4] M.O. Ball and M.J. Magazine. Sequencing of insertions in printed circuit board assembly. *Operations Research*, 36:192–201, 1988.

[5] P. Chalasani and R. Motwani. Approximating capacitated routing and delivery problems. *SIAM Journal on Computing*, 28:2133–2149, 1999.

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

[7] G. Cornuéjols, J. Fonlupt, and D. Naddef. The traveling salesman problem on a graph and some related integer polyhedra. *Mathematical Programming*, 33:1–27, 1985.

[8] G.N. Frederickson, M.S. Hecht, and C.E. Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7:178–193, 1978.

[9] G. Ghiani, G. Laporte, and F. Semet. The black and white traveling salesman problem. *Operations Research*, 54:366–378, 2006.

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

[11] 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.

[12] 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.

[13] 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.

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

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

[16] 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.

[17] 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.

(n, m)	Subtours	Combs	Gap	Nodes	Seconds
(50,3)	60	23	0.0125	7	4
(50,4)	24	0	0.0000	1	0
(50,5)	28	4	0.0032	7	2
(50,6)	5	0	0.0000	3	0
(50,7)	3	0	0.0000	1	0
(50,8)	7	0	0.0000	1	0
(60,3)	40	0	0.0000	1	2
(60,4)	31	4	0.0038	3	2
(60,5)	21	0	0.0001	5	1
(60,6)	1	0	0.0000	1	0
(60,7)	5	0	0.0000	1	0
(60,8)	1	0	0.0000	1	0
(70,3)	32	73	0.0008	9	12
(70,4)	43	33	0.0005	37	19
(70,5)	39	0	0.0000	1	4
(70,6)	15	0	0.0000	1	1
(70,7)	11	0	0.0000	1	1
(70,8)	1	0	0.0000	1	0
(80,3)	62	46	0.0000	3	14
(80,4)	70	58	0.0002	3	24
(80,5)	14	2	0.0000	1	2
(80,6)	31	0	0.0000	1	5
(80,7)	17	0	0.0000	1	3
(80,8)	1	0	0.0000	1	0

Table 3: Detailed computational results on random instances ($50 \leq n \leq 80$)

(n, m)	Subtours	Combs	Gap	Nodes	Seconds
(90,3)	215	22	0.0001	7	75
(90,4)	195	24	0.0001	3	87
(90,5)	153	42	0.0049	19	84
(90,6)	7	0	0.0000	5	2
(90,7)	22	0	0.0003	9	9
(90,8)	139	20	0.0015	3	78
(100,3)	164	36	0.0012	7	70
(100,4)	34	52	0.0018	13	33
(100,5)	84	0	0.0001	5	34
(100,6)	20	0	0.0000	1	6
(100,7)	33	4	0.0001	11	21
(100,8)	160	0	0.0000	1	132
(110,3)	42	36	0.0000	7	21
(110,4)	252	71	0.0013	11	233
(110,5)	13	0	0.0000	1	4
(110,6)	182	91	0.0003	9	242
(110,7)	20	0	0.0000	1	7
(110,8)	31	0	0.0003	5	16
(120,3)	194	168	0.0010	23	269
(120,4)	66	0	0.0000	1	25
(120,5)	29	4	0.0000	3	12
(120,6)	251	12	0.0007	29	339
(120,7)	51	0	0.0000	1	28
(120,8)	100	0	0.0000	3	73

Table 4: Detailed computational results on random instances ($90 \leq n \leq 120$)

(n, m)	Subtours	Combs	Gap	Nodes	Seconds
(130,3)	119	151	0.0013	33	274
(130,4)	51	0	0.0000	3	20
(130,5)	89	80	0.0003	19	129
(130,6)	62	90	0.0024	25	187
(130,7)	35	4	0.0003	3	26
(130,8)	69	55	0.0001	15	148
(140,3)	158	33	0.0018	3	121
(140,4)	154	184	0.0002	7	266
(140,5)	46	0	0.0000	1	31
(140,6)	31	5	0.0000	3	23
(140,7)	48	2	0.0000	3	43
(140,8)	41	33	0.0001	7	81
(150,3)	459	47	0.0003	11	882
(150,4)	162	42	0.0010	9	211
(150,5)	189	38	0.0006	27	346
(150,6)	79	0	0.0000	1	67
(150,7)	206	55	0.0013	29	576
(150,8)	59	0	0.0000	1	59
(160,3)	179	164	0.0012	19	418
(160,4)	119	51	0.0000	3	143
(160,5)	34	0	0.0000	1	23
(160,6)	155	42	0.0000	3	260
(160,7)	57	0	0.0002	3	64
(160,8)	78	133	0.0006	111	909

Table 5: Detailed computational results on random instances ($130 \leq n \leq 160$)

(n, m)	Subtours	Combs	Gap	Nodes	Seconds
(170,3)	569	101	0.0001	5	1595
(170,4)	200	44	0.0006	15	354
(170,5)	315	50	0.0002	9	816
(170,6)	56	0	0.0000	1	57
(170,7)	235	189	0.0006	35	1467
(170,8)	116	137	0.0009	13	631
(180,3)	420	162	0.0005	11	1106
(180,4)	356	1	0.0002	3	646
(180,5)	53	71	0.0003	9	145
(180,6)	166	122	0.0001	7	487
(180,7)	291	54	0.0010	9	1097
(180,8)	45	0	0.0000	1	56
(190,3)	111	50	0.0002	5	180
(190,4)	133	69	0.0001	21	312
(190,5)	237	7	0.0001	5	589
(190,6)	99	0	0.0000	1	135
(190,7)	49	1	0.0000	1	69
(190,8)	31	7	0.0001	9	74
(200,3)	538	257	0.0006	47	3251
(200,4)	424	14	0.0000	3	1555
(200,5)	292	59	0.0001	7	911
(200,6)	247	0	0.0000	1	800
(200,7)	296	5	0.0000	3	886
(200,8)	309	10	0.0005	3	1207

Table 6: Detailed computational results on random instances ($170 \leq n \leq 200$)