Heuristics for the Mixed Swapping Problem

Charles Bordenave
Michel Gendreau
Gilbert Laporte

June 2008

CIRRELT-2008-24

Heuristics for the Mixed 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, to each vertex of a complete directed graph are associated at most two object types representing its supply and demand. It is assumed that for each object type the total supply equals the total demand. A vehicle of unit capacity, starting and ending its route at an arbitrary vertex, is available to carry the objects along the arcs of the graph. The aim is to determine a minimum cost route such that each supply and demand is satisfied. When some of the object types are allowed to be temporarily unloaded at some intermediate vertices before being carried to their final destination, the problem is called the Mixed Swapping Problem. In this paper we describe constructive and improvement heuristics which were successfully applied to randomly generated instances with up to 10,000 vertices, with an average optimality gap not exceeding 1%.

Keywords. Transportation problem, vehicle routing, heuristic.

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 #1322 by the Department of Computer Science and Operations Research of the Université de Montréal.

1 Introduction

Let $G = (V, A)$ be a complete directed graph, where $V = \{1, \ldots, n\}$ is the vertex set and $A = \{(i, j) \mid i \in V, j \in V, i \neq j\}$ the arc set. Without loss of generality, vertex 1 is arbitrarily designated as a *depot*. To each vertex $i \in V$ is associated a pair of unit weight object types (a_i, b_i), where a_i is the type initially located at i (its *supply*), and b_i is the type required by i (its *demand*). The object types belong to a set $O \cup \{0\}$, where $O = \{1, \ldots, m\}$ is the set of real object types, and 0 is an additional *null* object type allowing the vertices to have only a demand or only a supply (or none). A cost matrix (c_{ij}) satisfying the triangular inequality is defined on A. A unit capacity vehicle, starting and ending its route at the depot, is available to carry the objects between the vertices of V. A route segment along which the vehicle carries the null object type (i.e., it is not loaded) is called a *deadheading*. The set O is partitioned into two subsets O_n and O_d, where O_n represents the set of *non-droppable* object types, i.e., objects that must be shipped directly from their origin to their destination, and O_d denotes the set of *droppable* object types, i.e., objects that are allowed to be temporarily dropped at some intermediate vertices on the way to their final destination. The *Mixed Swapping Problem* (MSP) consists of determining a minimum cost route allowing the vehicle to reposition the objects in such a way that all demands are satisfied.

Figures 1 and 2 represent optimal MSP solutions for two different instances defined on the unit square. The object types belonging to O_n are printed in boldface and the object type carried along an arc is shown on the arc. The depot is represented as a square. Note that the first solution of cost 6 does not use any drop because object types 1 and 2 are non-droppable, whereas in the solution of the second instance, the object of type 1 is dropped at the bottom left vertex before being carried to the upper left vertex, yielding a cost of 5.4.

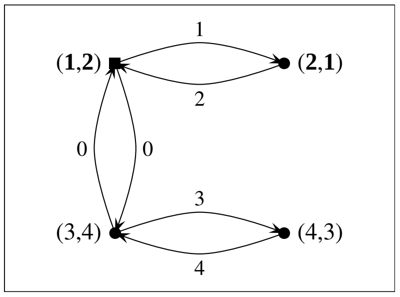

Figure 1: Optimal solution without drop

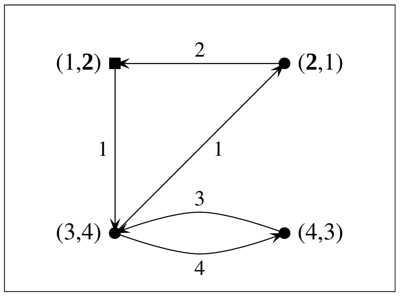

Figure 2: Optimal solution with drop

The MSP was introduced by Anily and Hassin [4], who defined its main terminology and identified interesting structural properties of optimal solutions. These authors showed the problem is NP-hard by reduction to the *Traveling Salesman Problem* (TSP) and designed a 2.5-approximation algorithm for it. Anily et al. [2] have studied the MSP on a line and proved that this particular case can be solved in polynomial time. More recently, Anily et al. [3] have shown that the SP defined on a tree is NP-hard and have provided a 1.5-approximation algorithm for this structure. They have also shown that the case where $m = 2$ can be solved in polynomial time. Bordenave et al. [7, 8] have proposed branch-and-cut algorithms for the non-preemptive and preemptive SP on a general graph. These authors were able to optimally solve instances with up to 200 vertices for the non-preemptive version, and 100 vertices for the preemptive version.

Many known routing problems are special cases of the MSP, like the *Stacker Crane Problem* (SCP) or the *Bipartite Traveling Salesman Problem* (BTSP). In the SCP, a set of arcs to be traversed by the solution is given, and the aim is to determine a minimum cost tour including these arcs. The SCP has been extensively studied. Frederickson et al. [14] have shown that the SCP on a complete graph is NP-hard and have proposed a 1.8-approximation heuristic for it. Atallah and Kosaraju [6] have considered two particular cases of the SCP where vertices are distributed along a line or along a circular shape. They have shown that these problems can be solved in polynomial time. Frederickson and Guan [12] have shown that the preemptive SCP on a tree is polynomial, but the non-preemptive SCP on a tree is NP-hard [13]. They have proposed two algorithms having worst-case performance ratios of 1.5 and 1.25. The SCP is a swapping problem (in general non-preemptive) where there exists exactly one object for each type, which means that the destination of each object is known a priori.

In the BTSP, n is even, half the vertices are black and half are white. The aim is to determine a minimum cost Hamiltonian cycle that does not visit two vertices of the same color in succession. This problem is NP-hard. Chalasani and Motwani [9] have proposed a 2-approximation algorithm for it, based on the intersection of two specific matroids. One can easily show that this problem corresponds to a swapping problem with two object types (for this particular case, the preemptive and non-preemptive swapping problem yield the same optimal solution, which is also an optimal BTSP solution).

Our purpose is to develop heuristics for the MSP, consisting of a constructive phase followed by an improvement phase. The remainder of the paper is organized as follows. The constructive phase of the heuristics is covered in Section 2, while the improvement phase is presented in Section 3. Implementation details are discussed in Section 4. Computational results are reported in Section 5, followed by conclusions in Section 6.

2 Constructive heuristic

The purpose of this section is to describe an algorithm to construct a feasible MSP solution. It is our implementation of Algorithm 3.7 described in [4], except for Step 3.7.4 which cannot be applied if an Eulerian circuit is not available. This step has been replaced with Step 4 of our heuristic. Any feasible MSP solution is characterized by a subset of arcs, the object type carried along each arc, and the order of arc visits. If there is no drop in the solution, then the order of arc visits can be obtained by determining an Eulerian circuit, by means of the end-pairing algorithm [15], for example. The constructive heuristic described by Algorithm 1 consists of four main steps: assignment, patching, matching, and construction of an Eulerian circuit.

Definition 1. *A vertex i with $a_i = b_i$ is called a transshipment vertex.*

Since for every transshipment vertex the demand is already satisfied by its supply, our heuristic ignores all transshipment vertices, except possibly the depot, which is necessarily visited in any feasible solution even if it is a transshipment vertex.

Definition 2. *The set of non-transshipment vertices (in addition to one possible transshipment vertex i if $i = 1$) supplying an object of type k is denoted by A_k, i.e., $A_k = \{i \in V \,|\, a_i = k \text{ and } b_i \neq k\} \cup \{1 \,|\, a_1 = b_1 = k\}$. The set of non-transshipment vertices (in addition to one possible transshipment vertex i if $i = 1$) demanding an object of type k is denoted by B_k, i.e., $B_k = \{i \in V \,|\, b_i = k \text{ and } a_i \neq k\} \cup \{1 \,|\, a_1 = b_1 = k\}$.*

2.1 Assignment solution

By definition the demand and the supply of each vertex must be satisfied. If a solution with no drop is considered, then it consists of a set of *service paths*, i.e., a set of arcs along which the vehicle is repositioning an object from a vertex $i \in A_{a_i}$ to a vertex $j \in B_{b_j}$. Therefore the service paths define a set of assignment arcs. This yields the first step of Algorithm 1, which consists of determining for each object type $k \in O \cup \{0\}$ a minimum assignment in a complete bipartite graph with vertex bipartition $\{A_k, B_k\}$. The assignment problem solution (connecting each supply to a demand), consists of a set of p simple circuits constituting connected components (Figure 3). If there is only one simple circuit, then it constitutes a feasible and optimal solution (see Proposition 1). Otherwise, additional arcs must be added to the current set of arcs in order to construct a feasible solution.

Algorithm 1. Constructive heuristic

```
Input:   $G = (V, A)$ and $(a_i, b_i), \forall\, i \in V$.
Output:  A feasible MSP solution $S$ and the order of arc visits.

Step 1   Assignment
    a)   Determine a minimum assignment problem in a complete bipartite graph
         with vertex bipartition $\{A_k, B_k\}, \forall\, k \in O \cup \{0\}$.
    b)   Superpose all assignment arcs to create the graph $G^0 = (V^0, A^0)$, where
         $V^0 = V$.
    c)   Identify the connected components $\{C_t\}_{t\geq 1}$ of $G^0$.
    d)   If $t = 1$, let $S$ be the simple circuit formed by $A^0$. The order of arc visits
         is trivial since $S$ is a simple circuit. Output $S$ and stop.
Step 2   Patching
    a)   Select a vertex in each $C_t$ and create the undirected graph $G^1 = (V^1, E^1)$,
         where each entry $c^1_{ij}$ of the cost matrix defined on $E^1$ represents the mini-
         mum cost between $C_i$ and $C_j$ (the components containing $i$ and $j$, respec-
         tively).
    b)   Determine a minimum spanning tree in $G^1$. Let $G^2 = (V^2, E^2)$ be the
         resulting tree, where $V^2 = V^1$.
Step 3   Matching
    a)   Identify the odd degree vertices $V^3$ in $G^2$, and create the complete undi-
         rected graph $G^3 = (V^3, E^3)$.
    b)   Determine a minimum perfect matching in $G^3$. Let $G^4 = (V^4, E^4)$ be the
         resulting graph, where $V^4 = V^3$ and $E^4$ is the set of matching edges.
Step 4   Construction of an Eulerian circuit
    a)   Direct the edges of $E^2 \cup E^4$ in such a way that $\delta^+(i) = \delta^-(i), \forall\, i \in$
         $V^2 \cup V^4$. Let $G^5 = (V^5, A^5)$ be the resulting graph.
    b)   Assign object type 0 to each arc of $G^5$.
    c)   Combine $A^0$ and $A^5$ to create the graph $G^6 = (V^6, A^6)$, where $V^6 = V$.
    d)   Determine an Eulerian circuit in $G^6$ to obtain the order of arc visits, and
         output $S = A^6$.
```

Proposition 1. *The assignment solution value provides a lower bound on the optimal MSP solution value.*

Proof. Let z^* be the optimal solution value. Let $\{C_t\}_{t\geq 1}$ be the collection of simple circuits obtained by solving the $m+1$ minimum assignment problems (Step 1c), and denote by $c(U)$ the sum of the arc costs of $U \subseteq A$. In any feasible solution, an object of type k initially at vertex i (with $a_i \neq b_i$ or $i = 1$) is carried to its final destination j ($b_j = k$) either via a single arc (i, j) or via a sequence of drops at intermediate vertices between i and j. In both cases, from the triangular inequality, the total cost of this route segment is greater than or equal to c_{ij}. Therefore, the sum of the cost of each assignment circuit represents a lower bound for the problem, i.e. $z^* \geq \sum_{t\geq 1} c(C_t)$. $\square$

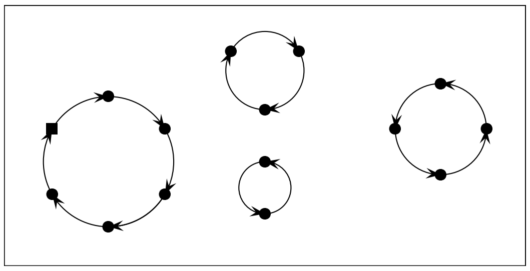

Figure 3: Assignment solution

2.2 Patching solution

After the p connected components in the current solution have been identified, an undirected complete graph whose vertices represent the p components is constructed. The cost of an edge linking two components C_i and C_j can be defined in several ways, for example the minimum arc cost between C_i and C_j. A minimum weight spanning tree is then determined in this graph. The arcs of the current solution and the edges of the spanning tree yield a mixed graph (Figure 4).

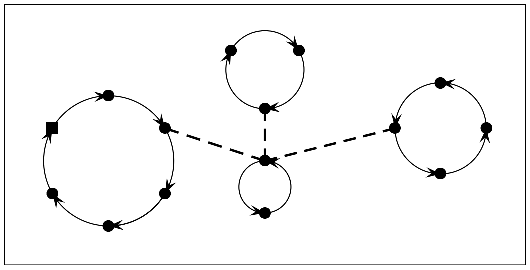

Figure 4: Patching circuits

2.3 Matching solution

Since any tree has at least two leaves, there exist at least two odd degree vertices in the current solution. An undirected complete graph is created on the set of odd degree vertices. It is well known that any graph has an even number of odd degree vertices, and therefore a minimum weight perfect matching can be computed on this graph (Figure 5).

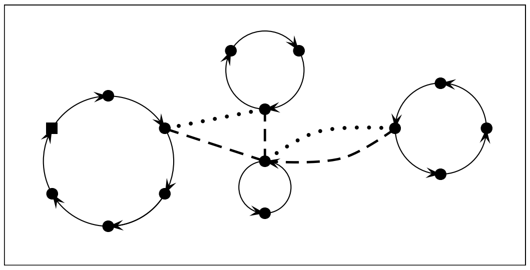

Figure 5: Matching odd degree vertices

2.4 Construction of an Eulerian circuit

Any feasible MSP solution is described by a set of directed arcs where each vertex has the same flow entering it and exiting it. Therefore the patching and matching edges must be directed so that for each vertex the in-degree equals the out-degree. To this end, an Eulerian cycle starting at one of the vertices incident to the patching and matching edges (arbitrarily selected) is determined. The order of edge visits in the Eulerian cycle indicates how the edges can be directed so that the degrees are balanced (Figure 6). Then, the null object is assigned to each of the newly created arcs, which is feasible because the vehicle can always carry the null object from any vertex to any other one.

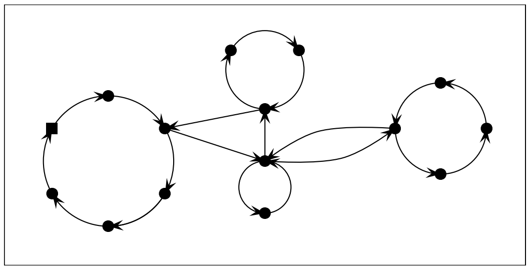

Figure 6: Directing edges

The final step consists of determining the order in which the vehicle must visit the vertices. Indeed, as shown in Figure 7, in which the bold arcs indicate the vehicle circuit, a poor choice of an outgoing arc at some vertices may lead to a partial solution because some non-transshipment vertices will not be visited by the vehicle (the remaining circuit on the right-hand side is isolated). This problem is not mentioned in [4], and the proof of Theorem 3.8 of [4] cannot be based on

Theorem 3.4 as claimed. In addition, in Step 3.7.4 of [4], because the order of arc visits is not specified, it may not be possible to implement the improvement steps suggested in that paper. To avoid this, an Eulerian circuit is determined in the current solution. The sequence of arcs in the circuit indicates the order of arc visits that must be followed by the vehicle in order to visit all non-transshipment vertices (or one possible transshipment vertex if it is the depot). Note that if the cost matrix is asymmetric, then the arcs linking the connected components in the Eulerian circuit may have a cost that is different from that used to compute the minimum spanning tree in Step 2b.

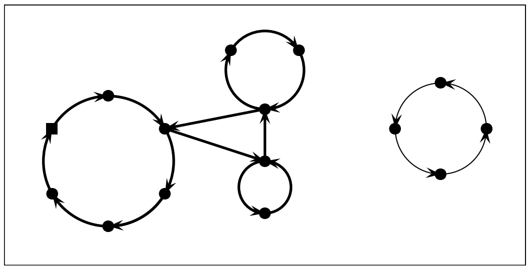
Figure 7: Partial solution with an isolated circuit

Proposition 2. *Algorithm 1 produces a feasible MSP solution with a worst-case ratio of 2.5, and this bound is tight.*

Proof. The proof given [4] applies because the validity of this proposition depends on Steps 1, 2 and 3 only. □

3 Improvement heuristics

The current solution represents a feasible MSP solution. This section described several ways to shorten the solution. These are summarized in Algorithm 2.

Proposition 3. *There exists a feasible solution that does not contain two consecutive arcs associated with the same object type.*

Proof. Follows from the triangular inequality of the cost matrix. □

Algorithm 2. Improvement heuristics

```
Input:   A feasible MSP solution $S$ and the order of arc visits.
Output:  A feasible MSP solution $S'$ of lower or equal cost and the order of arc
         visits.

Step 1   Shortcutting
    a)   Replace each pair of consecutive arcs carrying the same object type by a
         single arc until no such a shortcut can be made, while updating the order
         of arc visits. Let $S'$ be the new solution.
Step 2   Exchanging arcs
    a)   Perform $r$-opt or $r$-$r'$-opt ($r' \neq r$) in $S'$, while updating the order of arc
         visits.
Step 3   Using drops
    a)   Identify simple circuits of deadheadings $\{C_t\}_{t \geq 0}$ in $S'$.
    b)   If $t \geq 1$, let $(u, v)$ be the arc carrying object type $k$, that the vehicle uses
         to reach $C_t$ (for a given $t$), and let $(v, w)$ be the first arc of $C_t$. If $k \in O_d$,
         replace $(u, v)$ and $(v, w)$ with the arc $(u, w)$ carrying $k$. Assign $k$ to every
         other arcs of $C_t$.
    c)   Perform Step 1 in $S'$.
    d)   Output $S'$ and the order of arc visits.
```

3.1 Shortcutting

At this stage of the heuristic, since we have assigned the null object to the patching and matching arcs, shortcutting two consecutive arcs carrying the same type can be possible only with the null object and among incident arcs to vertices that belong to at least two simple circuits in the current solution (Figure 8). The process of shortcutting two arcs must be repeated until no more shortcut of this type can be found, since it may create two new consecutive arcs carrying the same object type.

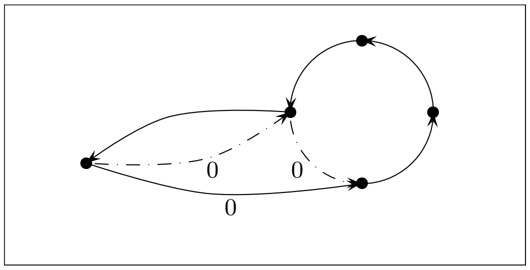

Figure 8: Shortcutting two consecutive arcs carrying the same object type

3.2 Exchanging arcs

A local search method can be used to improve the quality of the solution at the expense of extra computation time. Since the current solution does not contain any drop, an r-opt technique can be applied without worrying about precedence relationships. Three arc exchange techniques have been considered: 3-opt, 4-opt and 3-4-opt, which consists of repeatedly applying 3-opt and 4-opt until no improvement is possible. Similar to what happens for the directed TSP, in the MSP there is only one way, in our 3-opt and 4-opt heuristics, of reconstructing a feasible circuit when three or four arcs have been removed. This can readily be checked by enumeration. For each object type $k \in O \cup \{0\}$, r arcs carrying k are selected, and then interchanged to test whether this improves the quality of the current solution. For example, in 3-opt, three arcs $a_1 = (i_1, j_1)$, $a_2 = (i_2, j_2)$ and $a_3 = (i_3, j_3)$ carrying k are selected. If $c_{i_1 j_2} + c_{i_3 j_1} + c_{i_2 j_3} < c_{i_1 j_1} + c_{i_2 j_2} + c_{i_3 j_3}$, then a_1, a_2 and a_3 are replaced with $a'_1 = (i_1, j_2)$, $a'_2 = (i_3, j_1)$ and $a'_3 = (i_2, j_3)$, yielding a shorter feasible solution. Similarly, in 4-opt, four arcs $a_1 = (i_1, j_1)$, $a_2 = (i_2, j_2)$, $a_3 = (i_3, j_3)$ and $a_4 = (i_4, j_4)$ carrying k are selected. If $c_{i_1 j_3} + c_{i_4 j_2} + c_{i_3 j_1} + c_{i_2 j_4} < c_{i_1 j_1} + c_{i_2 j_2} + c_{i_3 j_3} + c_{i_4 j_4}$, then a_1, a_2, a_3 and a_4 are replaced with $a'_1 = (i_1, j_3)$, $a'_2 = (i_4, j_2)$, $a'_3 = (i_3, j_1)$ and $a'_4 = (i_2, j_4)$, yielding a shorter feasible solution. This process is repeated iteratively until no further improvement can be identified. Figures 9 and 10 illustrate a 3-opt exchange and a 4-opt exchange, respectively.

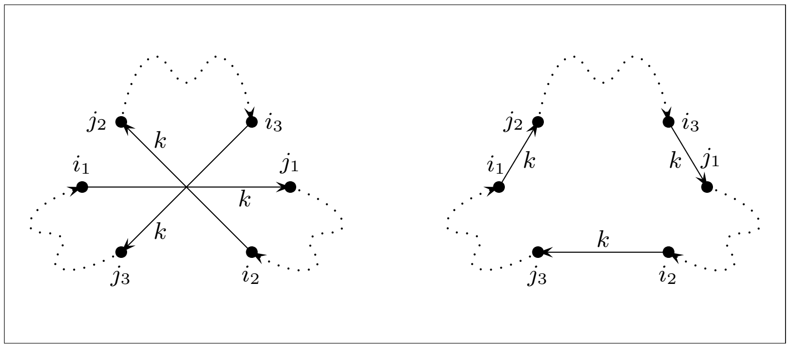

Figure 9: A 3-opt exchange

3.3 Using drops

So far the solution does not contain drops. Suppose there exists a simple circuit C of deadheadings in the current solution. The vehicle reaches C by an arc (u, v) carrying an object of type k, and then travels along the first arc (v, w) of C. If

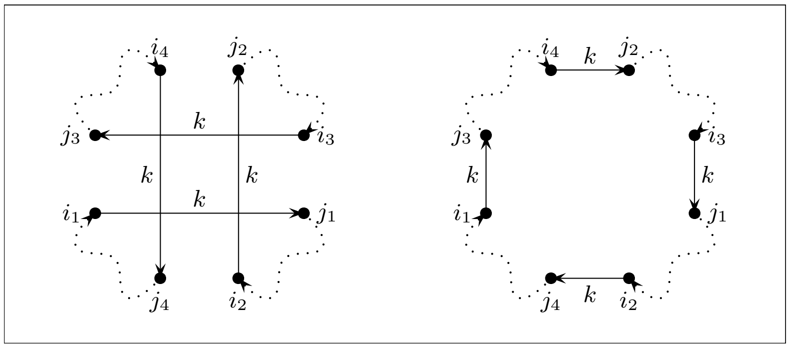

Figure 10: A 4-opt exchange

$k \in O_d$, then arcs (u, v) and (v, w) can be replaced with a new arc (u, w) carrying k. Assigning k to every other arcs in C clearly leads to a new feasible solution no worse than the previous one (Figure 11). After doing this optimization, some consecutive arcs can possibly carry the same object type. Therefore the solution is scanned again to determine whether it can be further shortened by shortcutting two consecutive arcs carrying the same object type, as described in Section 3.1.

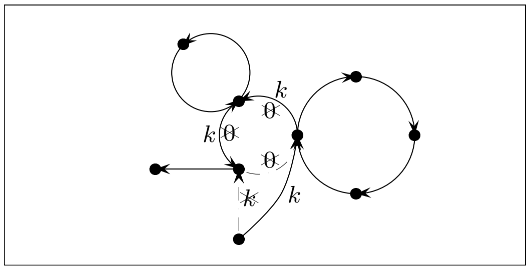

Figure 11: Shortcutting using drop ($k \in O_d$)

4 Implementation

The heuristics just described were implemented in C++. In this section, we discuss the various procedures contained in the heuristic, and their respective time complexity. In what follows, we call *basic* the version of the heuristic that does not incorporate the r-opt and r-r'-opt heuristics, and *full* the version that applies them.

The minimum assignment problems are solved by using the Kuhn-Munkres algorithm ([17], [18]), sometimes referred to as the Hungarian algorithm, which runs in $O(n^3)$ time. This leads to an $O(n^3 m)$ time complexity for the first step of the heuristic. This complexity order dominates all other steps of the basic heuristic. For comparison purposes we also conducted experiments in which CPLEX was used to solve the assignment problems instead of the Kuhn-Munkres algorithm. The advantage of using CPLEX is that it can solve all assignment problems simultaneously. However, memory requirement and running times are higher with CPLEX. For many operations on graphs we used Boost ([1]), which is a publicly available C++ library focussed on data structures and graph algorithms. This package provides many routines that are both fast and easy to use. Retrieving the connected components created by the first step of the heuristic takes $O(n + n\alpha(n))$ time, where α is the inverse of Ackermann's function. Computing the minimum spanning tree over these $p \leq n/2$ components using the Kruskal algorithm ([16]) requires $O(|E| \log |E|)$ time, where E is the edge set of the complete graph defined on the p components. To solve the minimum weight perfect matching we used the *Blossom IV* code developed by Cook and Rohe [10] for Concorde ([5]). This code implements an optimized version of Edmonds algorithm ([11]) running in $O(|\widetilde{V}|^4)$ time, where $\widetilde{V}$ is the set of odd degree vertices ($|\widetilde{V}| \leq p - 1$). For the computation of Eulerian circuits, we used the Hierholzer algorithm ([15]) which runs in linear time. Shortcutting arcs or using drops can also be performed in linear time. The r-opt and r-r'-opt heuristics are pseudo-polynomial. Since this process is time consuming (each step of r-opt requires $O(n^r)$ time and the number of steps can be high), we have applied these improvement steps only on instances containing fewer than 1000 vertices.

5 Computational results

Three sets of instances were generated as follows. Each set contains random geometric instances in which n vertices are located in a 500×500 square according to a uniform discrete distribution. Each vertex is associated with a random supply and demand such that for each object type the total supply equals the total demand. The number $|O_d|$ of droppable object types was randomly selected between 0 and $|O|$. We have tested the heuristic on values of n ranging from 100 to 10,000 and on values of m ranging from 3 to 8. The reported results correspond to the average over these three sets. Tests were performed on an AMD Opteron Dual Core 285 2.6GHz (1 GB RAM was required for solving the large instances).

Tables 1 and 2 report computation times (in seconds) and optimality gaps with respect to the assignment lower bound (in percentage) for different values of n and m, for the basic heuristic. Instances with few object types are more difficult

$n \setminus m$	3	4	5	6	7	8	Average
100	0.0	0.0	0.0	0.0	0.0	0.0	0.0
200	0.2	0.1	0.1	0.1	0.1	0.0	0.1
500	3.8	2.2	1.5	1.1	0.8	0.6	1.7
1000	48.0	23.5	16.0	11.0	8.4	5.8	18.8
2000	434.0	290.4	175.6	130.4	81.7	67.2	196.5
5000	14041.6	7990.2	4940.9	3302.2	2238.6	1534.6	5674.7
10,000	161343.2	101560.0	73437.0	44575.1	31084.9	20461.3	72076.9
Average	25124.4	15695.2	11224.4	6860.0	4773.5	3152.8	

Table 1: Computation times for the basic heuristic (in seconds)

to solve. This can be explained by the fact that the assignment problems, which represent the most time consuming part of the basic heuristic, are harder to solve. Indeed, for a fixed number of vertices, decreasing the number of object types increases the number of possible assignments for a given object.

As shown in Table 2 the optimality gap is remarkably small. It represents on average only 0.9% with respect to the lower bound provided by the assignment solution value. It tends to become smaller as the instance size become larger. The number of object types also influences the size of the optimality gap. If we consider small values of m, the assignment solution is typically formed by a large collection of simple circuits, each containing very few vertices. Since a solution generated by the heuristic is based on the assignment solution, the application of the patching and matching procedures produces solutions for which the optimality gap is larger.

$n \setminus m$	3	4	5	6	7	8	Average
100	2.922	2.338	0.937	1.690	0.671	0.633	1.532
200	2.286	1.923	1.047	1.149	0.470	0.721	1.266
500	2.116	1.317	0.425	0.613	0.638	0.414	0.921
1000	1.855	1.275	0.576	0.689	0.650	0.331	0.896
2000	1.712	0.684	0.820	0.397	0.389	0.267	0.712
5000	1.859	0.767	0.546	0.459	0.381	0.297	0.718
10,000	1.644	0.716	0.632	0.480	0.240	0.200	0.652
Average	2.056	1.289	0.712	0.782	0.491	0.409	

Table 2: Optimality gaps for the basic heuristic with respect to the assignment lower bound (in percentage)

Tables 3 and 4 provide a comparison of the different optimization methods in terms of average computation time and average optimality gap. Since these methods are time consuming we only tested them for relatively small values of n. We

n	Basic	3-opt	4-opt	3-4-opt
100	0.0	0.0	0.1	0.2
200	0.1	0.3	3.3	2.2
500	1.7	7.4	246.7	128.3
1000	18.8	95.7	4596.1	2736.2
Average	5.1	25.8	1211.5	716.7

Table 3: Comparison of computation times for the full heuristic (in seconds)

n	Basic	3-opt	4-opt	3-4-opt
100	1.532	1.251	1.346	1.147
200	1.266	0.922	0.979	0.829
500	0.921	0.643	0.678	0.592
1000	0.896	0.639	0.680	0.585
Average	1.154	0.864	0.921	0.788

Table 4: Comparison of optimality gaps for the full heuristic with respect to the assignment lower bound (in percentage)

can see that 3-opt improves the optimality gap by about 25% with only a relatively small increase in computation time. This algorithm seems to be a good choice for small and medium size instances. The 4-opt heuristic is far less attractive since it generates larger optimality gaps and has very high computation times. The method yielding the smallest optimality gaps was the 3-4-opt heuristic, which combines 3-opt and 4-opt, but with this method running times also increase quickly with n. Performing 4-opt after 3-opt sometimes allows to escape from the current local minimum. On average, the 3-4-opt improvement heuristic yields a gap reduction of about 32% over the basic algorithm. Since 3-opt often improves the solution, there are fewer opportunities for 4-opt to do so, and then the overall process takes in general less time than applying only 4-opt. It should be noted that for each of these optimization techniques, the number of object types drastically affects the computation time. Instances for which m is small take more time since the number of triplets or quadruplets to consider is higher. For example, we noticed that applying 3-opt on a 1000-vertex instance with three object types increases by a factor 10 or more the time needed to solve an instance of the same size but containing eight object types. Finally tests have shown that the proportion of droppable object types has no significant effect on the performance of our heuristics.

6 Conclusions

We have provided a complete description of a constructive and several improvement heuristics for the *Mixed Swapping Problem*. These heuristics were successfully applied to large instances containing up to 10,000 vertices. The average optimality gap is remarkably small, typically less than 1%.

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] Boost C++ libraries, 2001–. http://www.boost.org.

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

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

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

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

[6] 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.

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

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

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

[10] W.J. Cook and A. Rohe. Computing minimum weight perfect matchings. *INFORMS Journal on Computing*, 11:138–148, 1997.

[11] J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17: 449–467, 1965.

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

[13] G.N. Frederickson and D.J. Guan. Nonpreemptive ensemble motion planning on a tree. *Journal of Algorithms*, 15:29–60, 1993.

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

[15] C. Hierholzer. Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechnung zu umfahren. *Mathematische Annalen*, 6:30–32, 1873.

[16] J.B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proceedings of the American Mathematical Society*, volume 7, pages 48–50, 1956.

[17] H.W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quaterly*, 2:83–97, 1955.

[18] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5:32–38, 1957.