# The Steiner Multi Cycle Problem with Applications to a Collaborative Truckload Problem

**Vinicius N. G. Pereira**
Institute of Computing, University of Campinas (Unicamp)
Campinas - SP, Brazil
vinicius.pereira@ic.unicamp.br

**Mário César San Felice**
Department of Computing, Federal University of São Carlos (UFSCar)
São Carlos - SP, Brazil
felice@ufscar.br

**Pedro Henrique D. B. Hokama**
Production Engineering Department, Federal University of São Carlos (UFSCar)
São Carlos - SP, Brazil
hokama@ic.unicamp.br

**Eduardo C. Xavier**
Institute of Computing, University of Campinas (Unicamp)
Campinas - SP, Brazil
eduardo@ic.unicamp.br

## Abstract

We introduce a new problem called Steiner Multi Cycle Problem that extends the Steiner Cycle problem in the same way the Steiner Forest extends the Steiner Tree problem. In this problem we are given a complete weighted graph $G = (V, E)$, which respects the triangle inequality, a collection of terminal sets $\{T_1, \ldots, T_k\}$, where for each $a$ in $[k]$ we have a subset $T_a$ of $V$ and these terminal sets are pairwise disjoint. The problem is to find a set of disjoint cycles of minimum cost such that for each $a$ in $[k]$, all vertices of $T_a$ belong to a same cycle. Our main interest is in a restricted case where $|T_a| = 2$, for each $a$ in $[k]$, which models a collaborative less-than-truckload problem with pickup and delivery. In this problem, we have a set of agents where each agent is associated with a set $T_a$ containing a pair of pickup and delivery vertices. This problem arises in the scenario where a company has to periodically exchange goods between two different locations, and different companies can collaborate to create a route that visits all its pairs of locations sharing the total cost of the route. We show that even the restricted problem is NP-Hard, and present some heuristics to solve it. In particular, a constructive heuristic called Refinement Search, which uses geometric properties to determine if agents are close to each other. We performed computational experiments to compare this heuristic to a GRASP based heuristic. The Refinement Search obtained the best solutions in little computational time.

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis, Theory of computation → Routing and network design problems, Applied computing → Transportation

**Keywords and phrases** Steiner Cycle, Routing, Pickup-and-Delivery, Less-than-Truckload

**Digital Object Identifier** 10.4230/LIPIcs.SEA.2018.26

**Funding** This work was supported by CNPq (proc. 154505/2015-3, 425340/2016-3, 304856/2017-7) and FAPESP (proc. 2017/11382-2, 2016/11082-6, 2015/11937-9, 2016/23552-7).





## 1 Introduction

In this paper we present the Steiner Multi Cycle Problem (SMCP) and a restricted version of it, called Restricted Steiner Multi Cycle Problem (R-SMCP), which models a collaborative less-than-truckload problem with pickup and delivery. In the SMCP one is given a complete weighted metric graph $G = (V, E)$ and a collection of pairwise disjoint terminal sets $\{T_1, \ldots, T_k\}$. The problem is to find a set of vertex disjoint cycles of minimum cost such that, for each $a \in [k]$, all vertices of $T_a$ belong to the same cycle in a solution. In the R-SMCP we assume each $T_a$ contains exactly two vertices, a pickup and a delivery, and that $\{T_1, \ldots, T_k\}$ forms a partition of $V$. This problem models a collaborative less-than-truckload problem where companies have pickup and delivery locations, and are willing to collaborate in order to reduce their individual transportation costs, given by a solution where each company creates a separate route containing only their own pickup and delivery vertices. Notice that we do not assume an order between the pickup and delivery vertices, since we consider a recurrent route which a truck will use several times. Throughout the text we refer to each $T_a$ as an agent that has associated with it the pickup and delivery vertices in $T_a$.

The SMCP is a generalization of the Steiner Cycle problem (SCP) in the same way as the Steiner Forest generalizes the Steiner Tree problem. The SCP was introduced by Salazar-González [18], where he analysed the polyhedral structure associated with the problem, introducing two lifting procedures to extend facet-defining inequalities from the travelling salesman polytope. Steinová [19] studied the approximability of the SCP, showing that the general problem in directed graphs does not admit an approximation algorithm with polynomial ratio in the input size, unless P=NP. Moreover, Steinová presents a $\frac{3}{2}$-approximation algorithm for the case in which the input graph is undirected and metric.

As mentioned, the R-SMCP is related to vehicle routing problems, in particular to a collaborative less-than-truckload problem. Literature in vehicle routing problems is vast and considers very different constraints, as can be seen at [20] and [1]. The R-SMCP is similar to the traditional vehicle routing problem with pickup and delivery, for which Parragh et al. [16] presents a survey. The R-SMCP was also inspired by the Lane Covering Problem with less-than-truckload shipments (see Ergun et al. [4, 5]). The objective of this problem is to find a minimum cost set of directed cycles which covers a given set of arcs. Another related problem is the location-routing problem with simultaneous pickup and delivery, presented by Karaoglan et al. [13]. In this problem, one is given a graph with customer vertices and possible depot vertices. Each customer has a pickup and delivery demand to be served by a route that starts at an opened depot. Pickup demands have to be transported to the depot and delivery demands have to be delivered from it.

In the less-than-truckload Problem modeled by R-SMCP, we consider that the demand of each agent is periodically delivered and much smaller than the vehicle capacity, e.g., letters or small deliveries among mail companies, cash or promissory notes among banks.

**Our contributions.** We introduce the R-SMCP, prove that it is NP-hard, and present some algorithms to solve it. In particular, we present an effective heuristic that uses geometric properties to cluster agents, called Refinement Search, and compare it with a proposed GRASP (Greedy Adaptive Randomized Search Procedure) heuristic. Also, we show a 4-approximation to the general problem, SMCP.

The paper is organized as follows. Section 2 presents the formal description of the problems, an ILP formulation and a 4-approximation algorithm for the SMCP. Section 3 presents heuristics we used to solve the Travelling Salesman Problem. Sections 4 and 5 describe the proposed methods to solve R-SMCP. Finally, Section 6 presents the computational results.

## 2 The Steiner Multi Cycle Problem

In this section we present a formal definition for the SMCP and prove that even its restricted version is NP-Hard. We also show an integer linear programming formulation for the problem.

The input for the SMCP is a complete graph $G = (V, E)$, with a metric cost function $c : E \to \mathbb{R}^+$, and a collection of terminals sets $\mathcal{T} = \{T_1, \ldots, T_k\}$, where $T_a \subseteq V$ and $T_a \cap T_b = \varnothing$ for $a$ and $b$ in $[k]$ with $a \neq b$. The problem is to find a set $\mathcal{C}$ of vertex disjoint cycles such that, for each $a \in [k]$, all terminals in $T_a$ belong to a same cycle. The cost of a solution $\mathcal{C}$ is the sum of the edges' costs used in its cycles, i.e., $\sum_{C \in \mathcal{C}} \sum_{e \in C} c_e$ . The goal is to find a minimum cost solution.

In the restricted version of the problem, R-SMCP, the terminals collection $\{T_1, \ldots, T_k\}$ forms a partition of $V$ and each $T_a$ contains exactly two vertices, a pickup and a delivery. In this case, we assume that the vertex set contains $2n$ vertices, $V = \{1, 2, \ldots, 2n\}$ where each $T_a = \{a, a + n\}$ for $a \in [n]$. We define a set of agents $A = \{1, 2, \ldots, n\}$, such that each agent $a \in A$ has a pickup point $p_a$ on vertex $a$ and a delivery point $d_a$ on vertex $a + n$.

Note that, as $G$ is a metric graph, there is always an optimum solution which does not use Steiner vertices. To see this, suppose an optimum solution with a cycle $C = (u_1, u_2, \ldots)$, where some $u_j$ is a Steiner vertex. We can connect vertices $u_{j-1}$ and $u_{j+1}$ directly and remove $u_j$ from $C$, resulting in a solution with cost at most the cost of $C$, due to the triangle inequality. Moreover, this holds to SMCP and R-SMCP.

We can show that the SMCP is NP-Hard since it generalizes the minimum Steiner Cycle problem. In fact, we proved that even R-SMCP is NP-Hard.

▶ **Theorem 1.** *The Restricted Steiner Multi Cycle Problem is NP-Hard.*[1]

### 2.1 An ILP Formulation and its Linear Relaxation

Given an instance $(G, c, T)$ of the SMCP, consider a function $f : 2^V \to \{0, 1\}$ such that for each non-empty set $S \subset V$ we have $f(S) = 1$ if, and only if, for some $T_a \in \mathcal{T}$ we have that $S \cap T_a \neq \varnothing$ and $T_a \nsubseteq S$, i.e, $S$ is a cut that separates terminals in $T_a$. With abuse of notation we use $i \in T$ to denote a vertex that is a terminal. The SMCP can be formulated with the following integer linear program:

$$\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e && && (1)\\
\text{s.t.} \quad & \sum_{e \in \delta(i)} x_e = 2 && \forall i \in \mathcal{T} && (2)\\
& \sum_{e \in \delta(S)} x_e \geqslant 2 f(S) && \varnothing \neq S \subset V && (3)\\
& x_e \in \{0, 1, 2\} && e \in E\,, && (4)
\end{aligned}$$

where $\delta(S)$ denotes the set of edges having exactly one end point in $S$. The variable $x_e$ indicates if an edge is used in the solution, constraint (2) assures that exactly one cycle covers each terminal, constraint (3) assures that vertices belonging to a terminal set $T_a \in \mathcal{T}$ are connected, and constraint (4) allows each edge to be used at most twice, since in the case where $T_a$ has just two vertices, a single cycle between them is a valid solution.

Relaxing the integrality constraints (4) we obtain a linear program useful to find lower bounds for the SMCP and for its restricted version, R-SMCP. Note that the number of constraints (3) is exponential. However, it is possible to solve the relaxed LP in polynomial time by solving the separation problem in polynomial time [9]. We find violated constraints in polynomial time by solving maximum flow problems between each pair of vertices in a same $T_a$, through the use of Gomory-Hu trees [8, 11].

[1] Proof omitted due to space constraints.

## 2.2 A 4-approximation Algorithm for the Steiner Multi Cycle Problem

A 2-approximation algorithm for the Steiner forest problem was presented by Goemans and Williamson [7]. We use this algorithm to obtain a 4-approximation algorithm for the SMCP.

Given an instance $(G, c, \mathcal{T})$ of the SMCP, we use the 2-approximation algorithm of [7] to obtain a Steiner forest $F$ with cost $c(F) \leqslant 2\mathrm{OPT}(G)$, where $\mathrm{OPT}(G)$ is the cost of an optimal solution for the SMCP. Duplicate each edge of $F$, and then find an Eulerian circuit for each component of $F$. The cost of all Eulerian circuits is limited by $2c(F) \leqslant 4\mathrm{OPT}(G)$. Finally, for each component, perform shortcuts in order to transform each Eulerian Circuit into a cycle, obtaining a valid solution $\mathcal{C}$ for the SMCP with cost at most $4\mathrm{OPT}(G)$.

▶ **Proposition 2.** *There is a 4-approximation algorithm for the Steiner Multi Cycle Problem.*

In the remaining of the text we present heuristics for the R-SMCP, since our main interest is in solving the related collaborative less-than-truckload problem with pickup and delivery.

# 3 TSP Heuristics

In this section we present the TSP heuristics that are used by our algorithms. All of them build a cycle cover by first choosing a promising agent partition, and then transforming each part into a cycle, by using a TSP heuristic. A partition of a set $A$ is a collection $\{C_1, \ldots, C_k\}$, where each $C_i \subseteq A$, $\cup_{i=1}^{k} C_i = A$, and for each pair of different parts $C_i$ and $C_j$ we have $C_i \cap C_j = \varnothing$. We call each set $C_i$ a cluster or a part of $A$, for $i \in [k]$.

Given a cluster $C \subseteq V$, we use two heuristics to find a minimum cost TSP of $C$. One is the Nearest Insertion algorithm and the other is the Christofides algorithm. We apply the 2-OPT local search over the solution found by both heuristics. A detailed description of these heuristics can be found in [2]. The Nearest Insertion has a time complexity of $\mathrm{O}(n^2)$, while Christofides has time complexity $\mathrm{O}(n^3 \lg(n))$, where $n$ is the number of vertices. The 2-OPT local search can have an exponential time complexity in the worst case. However, it is usually fast when the initial solution is good.

# 4 Refinement Search Heuristic

In this section we describe a deterministic heuristic, called Refinement Search, for the Restricted Steiner Multi Cycle Problem. We first define a measure of proximity, based on geometric regions, between an agent and any vertex of the graph $G$. This proximity measure is used to establish a neighborhood relation between each pair of agents, which determines if the agents are close to each other. From this, we obtain a partition of the agents and, for each part, we use a TSP heuristic to obtain a cycle that covers all its agents.

## 4.1 Region types

Given an agent $a$ in $A$ and a non-negative real number $r$, we define three types of regions.

**Circular Intersection:** denoted as $R_I(a, r)$, is defined by the intersection of two circles with radius $r$, one with center in $p_a$ and other with center in $d_a$. More precisely, a vertex $v$ is in the circular intersection region $R_I(a, r)$ if, and only if, $c(v, p_a) \leqslant r$ **and** $c(v, d_a) \leqslant r$.

**Circular Union:** denoted as $R_U(a, r)$, is defined by the union of two circles with radius $r$, one with center in $p_a$ and other with center in $d_a$. More precisely, a vertex $v$ is in the circular union region $R_U(a, r)$ if, and only if, $c(v, p_a) \leqslant r$ **or** $c(v, d_a) \leqslant r$.

**Elliptical:** denoted as $R_E(a, r)$, is defined by the ellipse with focal points $p_a$ and $d_a$, and eccentricity $c(p_a, d_a)/2r$. More precisely, a vertex $v$ is in the elliptical region $R_E(a, r)$ if, and only if, $c(v, p_a) + c(v, d_a) \leqslant 2r$.

In the algorithm, we use a factor $\alpha \in \mathbb{R}^+$ to define the radius used by all agents. Given an $\alpha$ value, an agent $a$ will have its regions defined with radius $r_a = \alpha \cdot c(p_a, d_a)$, where $c(p_a, d_a)$ is the edge cost connecting the agent's vertices in $T_a$.

It is not hard to see that, for any type of region, if we increase the radius which defines it, the region can only increase with new vertices being added to it.

▶ **Lemma 3.** *Given an agent $a \in A$ and two radius $r, r' \in \mathbb{R}^+$, with $r \leqslant r'$, we have that $R_x(a, r) \subseteq R_x(a, r')$, for any region type $R_x \in \{R_I, R_E, R_U\}$.*

Given an agent, its circular intersection region is contained in its elliptical region which is contained in its circular union region, as long as these are defined with the same radius.

▶ **Lemma 4.** *Given $a \in A$ and $r \in \mathbb{R}^+$, we have that $R_I(a, r) \subseteq R_E(a, r) \subseteq R_U(a, r)$.*

While the concept of region allows us to relate an agent with a vertex, in order to establish a pairwise relationship among agents, we introduce the concept of neighborhood.

## 4.2 Neighborhood types

Given agents $a$ and $b$ in $A$, with regions $R_a$ and $R_b$, we define two types of neighborhood.
**Total:** agents $a$ and $b$ are total neighbors if $p_b$ and $d_b$ are in $R_a$, or if $p_a$ and $d_a$ are in $R_b$.
**Partial:** agents $a$ and $b$ are partial neighbors if $p_b$ or $d_b$ are in $R_a$, or if $p_a$ or $d_a$ are in $R_b$.
Intuitively, two agents are total neighbors if both vertices of an agent are inside the other's region. Similarly, two agents are partial neighbors if at least one vertex from an agent is inside the other's region. Thus, every total neighbors are partial neighbors. Also, note that the neighborhood relation is symmetric.

Now that the pairwise relationship among agents is defined, we introduce the concept of component in an auxiliary graph to establish a partition of agents.

## 4.3 Auxiliary Graph and Components

Take a set of agents $A$, a collection of regions $\mathcal{R}$, such that each agent $a \in A$ has a region $R_a \in \mathcal{R}$, and a neighborhood type $N_y \in \{N_T, N_P\}$, where $N_T$ stands for the Total neighborhood and $N_P$ for the Partial neighborhood. Consider an auxiliary graph in which there is a vertex for each agent, and there is an edge between two vertices if, and only if, the corresponding agents are neighbors. Notice that the connected components of this auxiliary graph form a partition $\mathcal{P}$ on the set of agents. Thus, from now on we use the words component and part (of the partition) interchangeably to refer to the agents of a same component of this auxiliary graph.

When considering partitions, there is an interesting property called refinement. We say that a partition $\mathcal{P}$ is finer than $\mathcal{P}'$ (or that $\mathcal{P}'$ is coarser than $\mathcal{P}$) if, and only if, every set in $\mathcal{P}$ is uniquely contained by some set of $\mathcal{P}'$. Note that the refinement property is transitive.

The following lemma shows a relationship between regions and refinement.

▶ **Lemma 5** (Region Refinement)**.** *Consider a set of agents $A$, two collections of regions $\mathcal{R}$ and $\mathcal{R}'$, each containing one region $R_a$ for each agent $a \in A$, and a neighborhood type $N_y \in \{N_T, N_P\}$. Suppose that, for each agent $a \in A$, $R_a \subseteq R'_a$, where $R_a \in \mathcal{R}$ and $R'_a \in \mathcal{R}'$. The components obtained using $\mathcal{R}$ and $\mathcal{R}'$, with the same neighborhood type $N_y$, induce partitions $\mathcal{P}$ and $\mathcal{P}'$ such that $\mathcal{P}$ is finer than $\mathcal{P}'$.*

Due to Lemmas 3 and 4, the previous lemma implies that, when all other parameters are constant, if the radius factor $\alpha$ increases, or the region type $R_I$ changes to $R_E$, or $R_E$ changes to $R_U$, then we have a new partition which is coarser than the previous one.

The following lemma shows a relationship between neighborhood types and refinement.

▶ **Lemma 6** (Neighborhood Refinement). *Consider a set of agents $A$, a collection of regions $\mathcal{R}$, and neighborhood types $N_T$ and $N_P$. The components obtained using $\mathcal{R}$, with neighborhood types $N_T$ and $N_P$, induce partitions $\mathcal{P}_T$ and $\mathcal{P}_P$ such that $\mathcal{P}_T$ is finer than $\mathcal{P}_P$.*

## 4.4 Dynamic Programming Algorithm for the Refinement Search

Given an input $(G, c, T)$ of the R-SMCP, consider a partition $\mathcal{P}_{(\alpha,R_x,N_y)}$ induced by radius factor $\alpha \in \mathbb{R}^+$, region type $R_x$ and neighborhood type $N_y$. We denote by $S_{(\alpha,R_x,N_y)}$ the solution for R-SMCP obtained by using a TSP heuristic to construct a cycle on each component of $\mathcal{P}_{(\alpha,R_x,N_y)}$. Moreover, given a component $C \in \mathcal{P}_{(\alpha,R_x,N_y)}$, we denote by $S_{(\alpha,R_x,N_y)}(C)$ the solution of $S_{(\alpha,R_x,N_y)}$ restricted to the agents in $C$.

The idea of the algorithm is to construct partitions of the agents using different types of regions, different values of $\alpha$, and different types of neighborhood. First, lets describe how we choose the different values of $\alpha$. We fix a region type $R_x$ and neighborhood type $N_y$. For each pair of agents $a$ and $b$, we find the smallest factor $\alpha_{ab}$ such that there is an edge between $a$ and $b$ in the auxiliary graph. Notice that, if we use a radius factor $\alpha_{min} = \min_{a,b\in A} \alpha_{ab} - \epsilon$ then $|A|$ components exist in the auxiliary graph. We construct a list of size $O(|A|^2)$ of radius factors containing $\alpha_{min}$ and $\alpha_{ab}$ for each pair of agents $a, b \in A$. A shorter list of radius factors is constructed as follows. Consider the list of factors sorted in increasing order, where the first value $\alpha_{min}$ is inserted in the beginning of the list. For the remaining values of factors, if its use decreases the number of components in the current auxiliary graph, then this $\alpha$ value is included in the list. Thus, the shorter list of radius factors contains $|A|$ values, since we start with $|A|$ components and finish with just a single component.

For each region type $R_x$ and neighborhood type $N_y$ we have a list of radius factors. So, there are at most $6|A|$ radius factors which we save in the list $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \dots]$ arranged in increasing order. A basic idea for a heuristic to solve the R-SMCP is to generate a partition $\mathcal{P}_{(\alpha,R_x,N_y)}$ for each region type $R_x$, neighborhood type $N_y$, and radius factor $\alpha$ in the final list of factors. Then, for each component in $\mathcal{P}_{(\alpha,R_x,N_y)}$ construct a cycle using a TSP heuristic obtaining a solution $S_{(\alpha,R_x,N_y)}$. Thus, we could return the best solution found considering all these $6|A|$ solutions.

However, we can find better solutions using the refinement property, since it allows us to mix solutions of different region types, neighborhood types and radius factors. For example, consider an instance with 10 agents where, if we use $\alpha = 0.5$, region $R_U$, and neighborhood $N_P$, we obtain the partition $\mathcal{P}(0.5, R_U, N_P) = \{\{a_1, a_5, a_6\}, \{a_2, a_3, a_4, a_7\}, \{a_8, a_9, a_{10}\}\}$. From Lemmas 3, 5 and 6, we can obtain finer solutions by changing either the region type, or the neighborhood type, or the value of $\alpha$. Thus, we can obtain the following finer solutions $\mathcal{P}(0.5, R_I, N_P) = \{\{a_1, a_5\}, \{a_6\}, \{a_2, a_3\}, \{a_4, a_7\}, \{a_8, a_9, a_{10}\}\}$, and $\mathcal{P}(0.5, R_U, N_T) = \{\{a_1, a_5, a_6\}, \{a_2, a_3, a_4\}, \{a_7\}, \{a_8, a_9\}, \{a_{10}\}\}$, and $\mathcal{P}(0.4, R_U, N_P) = \{\{a_1\}, \{a_5, a_6\}, \{a_2, a_3\}, \{a_4, a_7\}, \{a_8, a_9, a_{10}\}\}$. Then, when finding the best solution for $\mathcal{P}(0.5, R_U, N_P)$ we could use $\{a_2, a_3\}, \{a_4, a_7\}$ from $\mathcal{P}(0.4, R_U, N_P)$ if the two cycles have a smaller cost than the single cycle for $\{a_2, a_3, a_4, a_7\}$ obtained by $\mathcal{P}(0.5, R_U, N_P)$. This way the best solution could be $\mathcal{P}(0.5, R_U, N_P)^* = \{\{a_1, a_5, a_6\}, \{a_2, a_3\}, \{a_4, a_7\}, \{a_8, a_9, a_{10}\}\}$.

Since the refinement property is transitive, we can recurse further until the finest possible level, checking which parts of a solution should be replaced by smaller parts of finer solutions.

**Algorithm 1:** Refinement Search.

**Input** : $(G, c, T)$ and a vector $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_{6|A|}]$ radius factors in increasing order.
**Output** : A cycle cover $\mathcal{C}$.

1.1 **foreach** $\alpha_i, i \in [1, 2, \ldots, 6|A|]$ **do**
1.2  **foreach** neighbor type $N_y \in (N_T, N_P)$ **do**
1.3  **foreach** region type $R_x \in (R_I, R_E, R_U)$ **do**
1.4  **foreach** $C \in S_{(\alpha_i, R_x, N_y)}$ **do**
1.5  $C_1 \leftarrow S^*_{(\alpha_{i-1}, R_x, N_y)}(C)$
1.6  $C_2 \leftarrow S^*_{(\alpha_i, R_{x-1}, N_y)}(C)$
1.7  $C_3 \leftarrow S^*_{(\alpha_i, R_x, N_{y-1})}(C)$
1.8  $S^*_{(\alpha_i, R_x, N_y)} \leftarrow (S_{(\alpha_i, R_x, N_y)} \backslash C) \cup min\_cost\{C, C_1, C_2, C_3\}$
1.9 **return** $S^*_{(\alpha_{6|A|}, R_P, R_U)}$

Let the region types be organized from finer to coarser, i.e., $(R_I, R_E, R_U)$ and the same for the neighborhood types, i.e., $(N_T, N_P)$. For a set of agents $C$, remember that $S_{(\alpha, R_x, N_y)}(C)$ is the solution obtained from $\mathcal{P}_{(\alpha, R_x, N_y)}$ restricted to the agents in $C$. Formally, we want to find the best solution for all agents which can be described by the recurrence

$$S^*_{(\alpha_j, R_x, N_y)}(A) = \sum_{C \in \mathcal{P}_{(\alpha, R_x, N_y)}} \min \begin{cases} cost(S_{(\alpha_j, R_x, N_y)}(C)) \\ cost(S^*_{(\alpha_j, R_{x-1}, N_y)}(C)) \\ cost(S^*_{(\alpha_j, R_x, N_{y-1})}(C)) \\ cost(S^*_{(\alpha_{j-1}, R_x, N_y)}(C)) \end{cases}$$

where, for each component $C \in \mathcal{P}_{(\alpha, R_x, N_y)}$ we choose the best solution which can be obtained by considering the current solution $S_{(\alpha_j, R_x, N_y)}(C)$ using current radius factor, region type and neighborhood type, and the best solutions which are finer for this set of agents $C$.

The Algorithm 1 shows the pseudocode of a dynamic programming algorithm called Refinement Search. The algorithm constructs solutions in a bottom up fashion, from the base case with the finest $\alpha$, neighborhood type, and region type to the coarsest ones. For each component $C$ in the standard solution $S_{(\alpha, R_x, N_y)}$ of an iteration of loops (1.1 - 1.3), the algorithm checks if in a finer solution, the agents in $C$ were covered with a smaller cost. The best solution for $C$ among the current solution and the finer ones, is set on line 1.8.

## 5 GRASP Heuristic

In this section we present our GRASP based heuristic, which uses the Greedy Randomized Adaptative Algorithm described in Section 5.2 to generate initial solutions. After finding an initial solution, it performs a local search with the algorithm described in Section 5.3. The GRASP heuristic repeats this process until a stopping criteria occurs, which can be a maximum running time or a maximum number of iterations. Then, it returns the best solution found. The meta-heuristic GRASP was introduced by Feo and Resende [6] and it has been successfully applied to several combinatorial optimization problems [17]. More information is available in Resende and Ribeiro [17].

### 5.1 Basic Definitions for the GRASP Heuristic

The GRASP heuristic uses a contracted graph to find a partition of the agents set. This contracted graph is built using a measure of distance between each pair of agents, thus, we propose several ways to determine this measure of distance.

Given an input $(G, c, T)$ of the R-SMCP, each agent $a \in A$ has a pickup vertex $p_a$ and a delivery vertex $d_a$. The contracted graph $G_A = (A, E_A)$ is a complete weighted graph, in which each agent $a \in A$ corresponds to a single vertex and the distance between agents $a$ and $b$, is defined according to one of the following six types of distances:

**Distance 1** is the minimum cost of edges necessary to complete the cycle containing edges $(p_a, d_a)$ and $(p_b, d_b)$. More formally, $\text{dist}_1(a, b) = \min\{c(p_a, p_b) + c(d_a, d_b), c(p_a, d_b) + c(d_a, p_b)\}$. This distance is symmetric and satisfies the triangle inequality.

**Distance 2** is 0 if $a = b$, or the minimum cost of building a cycle that contains the pickup and delivery vertices of agents $a$ and $b$. More formally, for $a \neq b$,

$$\text{dist}_2(a, b) = \min \left\{ \begin{array}{r} \text{dist}_1(a, b) + c(p_a, d_a) + c(p_b, d_b), \\ c(p_a, d_b) + c(d_b, d_a) + c(p_b, d_a) + c(p_a, p_b) \end{array} \right\}$$

This distance is symmetric and satisfies the triangle inequality.

**Distance 3** is 0 if $a = b$, or the minimum cost to connect pickup and delivery vertices of $a$ to the pickup or delivery vertex of $b$. More formally, for $a \neq b$,

$$\text{dist}_3(a, b) = \min \left\{ \begin{array}{l} c(p_a, p_b) + c(d_a, p_b), \\ c(p_a, d_b) + c(d_a, d_b) \end{array} \right\}$$

This distance is not symmetric and does not satisfy the triangle inequality.

**Distance 4** is the symmetric version of $\text{dist}_3$, i.e., the minimum cost between applying $\text{dist}_3$ to $(a, b)$ and to $(b, a)$. More formally, $\text{dist}_4(a, b) = \min\{\text{dist}_3(a, b), \text{dist}_3(b, a)\}$. This distance is symmetric, but does not satisfy the triangle inequality.

**Distance 5** is the minimum cost of edges necessary to connect a vertex from agent $a$ to a vertex from agent $b$. More formally, $\text{dist}_5(a, b) = \min\{c(p_a, p_b), c(p_a, d_b), c(d_a, p_b), c(d_a, d_b)\}$. This distance is symmetric, but does not satisfy the triangle inequality.

**Distance 6** is the cost of a minimum path on the contracted graph using Distance 5 as edges costs. More formally, let $G_{A5}$ be the contracted graph using Distance 5, $\text{dist}_6(a, b) = \min_{P \in \mathcal{P}}\{c(P) | a \in P, b \in P\}$, where $\mathcal{P}$ is the set of all paths in $G_{A5}$ and $c(P)$ is the cost of path $P$. This distance is symmetric and satisfies the triangle inequality.

## 5.2 Greedy Randomized Adaptive Algorithm

The Greedy Randomized Adaptive algorithm, presented in Algorithm 2, creates $n$ different partitions of the agents set, starting with a partition containing just one cluster with all agents. The algorithm iteratively constructs a new partition, increasing by one the number of clusters in it, until a final partition containing $n$ clusters with isolated agents is created. For each partition, the algorithm creates a cycle cover by using the TSP heuristics described in Section 3 to construct a cycle for each cluster in the partition. Among all cycle covers created, the one with minimum cost is returned as solution. The algorithm starts by choosing at random an initial head (line 2.1) and by creating an initial cluster $C_1$ containing all vertices (line 2.3). Then, it computes the cost of the corresponding cycle cover of this initial clustering (line 2.4) and sets this solution as the best one so far. In any iteration of the main loop (lines 2.5 - 2.17), we begin with a partition with clusters $C_1, \ldots, C_{k-1}$ and we want to construct a new partition $C_1, \ldots, C_k$, with one more cluster. Each cluster $C_i$ has a special vertex denominated head, denoted by $h_i$. The algorithm proceeds by choosing a new head $h_k$, which is chosen as the agent that is farthest from its current head (lines 2.6 - 2.7). Then the algorithm re-creates the clusters $C_1, \ldots, C_k$ containing only their respective heads and, for each agent $a$ which is not a head, it is re-assigned to the cluster of minimum distance (lines 2.9–2.14). The distance

**Algorithm 2:** Greedy Randomized Adaptive.

**Input** : $(G, c, A)$, a distance function dist, and a threshold $T$.
**Output** : A cycle cover $\mathcal{C}$.

2.1 $initial\_head \leftarrow$ a random vertice $\in A$
2.2 $initial\_head$ becames the head of cluster $C_1$;
2.3 All agents are assigned to $C_1$;
2.4 $best\_cover \leftarrow$ a cycle cover computed from $C_1$;
2.5 **for** $k \leftarrow 2$ **to** $n$ **do**
2.6 $new\_head \leftarrow$ the agent that is farthest from its closest head;
2.7 $new\_head$ becomes a head of cluster $C_k$;
2.8 Remove all agents that are not a head from their clusters and put them on a list $L$
2.9 **while** there are elements in $L$ **do**
2.10 Let $min\_dist$ and $max\_dist$ be the minimum and maximum distance of any agent in $L$ to any cluster, respectively;
2.11 Create $RCL$ list with the agents that are in a distance less than or equal to $(min\_dist + T(max\_dist - min\_dist))$;
2.12 Randomly choose an agent $a$ in $RCL$;
2.13 Put the $a$ in its closest cluster;
2.14 Remove $a$ from $L$;
2.15 Let $\mathcal{C}$ be a cycle cover computed from clusters $C_1$ to $C_k$;
2.16 **if** cost of $\mathcal{C}$ is less than cost of $best\_cover$ **then**
2.17 $best\_cover \leftarrow \mathcal{C}$
2.18 **return** $best\_cover$

of agent $a$ to a cluster $C$ is defined as $dist(a, C) = \min_{b \in C} \{\text{dist}(a, b)\}$. Since we want a greedy randomized initial solution, instead of attributing each agent to the closest cluster in a deterministic greedy fashion, the heuristic creates a Restricted Candidate List (RCL) with the agents that are close to some cluster. More precisely, the RCL is created as follows, let $min\_dist$ and $max\_dist$ be the minimum and maximum distance, respectively, of any agent to any cluster. Given a threshold $T$, which is a value between 0 and 1, the algorithm constructs the RCL (lines 2.10 - 2.11) as the set containing every agent whose distance to a cluster is at most $(min\_dist + T(max\_dist - min\_dist))$. Then the algorithm chooses, uniformly at random, one of theses agents, and assigns it to its closest cluster (lines 2.12 - 2.13). Notice that the minimum distance of the remaining agents in $L$ has to be updated, since they may now become closer to the cluster where agent $a$ was inserted. The algorithm proceeds until all agents are assigned to some cluster in $\{C_1, \ldots, C_k\}$. Finally, the algorithm obtains a solution for the R-SMCP by running a TSP heuristic for each cluster (line 2.15) and it saves this solution if it is better than the current best one (lines 2.16 - 2.17).

## 5.3 Local Search

The Local Search algorithm begins with a current solution and then generates a set of neighbour solutions. If one of these has a cost reduction compared to the current solution, then the algorithm updates the current solution with the best neighbour solution found. It repeats this process until there is no significant improvement.

The algorithm used to find the best solution among the neighbour solutions is presented in Algorithm 3. The set of neighbor solutions $\mathcal{S}$ initially contains only the current solution (line 3.1). For each agent $a$, the algorithm builds several new solutions as follows. In one of the new solutions agent $a$ is removed from its current cycle and is left alone in an isolated cycle (line 3.3). The other solutions are constructed by removing agent $a$ from its current cycle and inserting it in another cycle (lines 3.5 - 3.6). In the Computational Results section we use a first improvement variant of this algorithm, where it returns the first generated

**Algorithm 3:** Find Best Neighbor Solution

```
Input  : (G, c, A) and an initial solution S'.
Output : A solution.
3.1 Let 𝒮 be a set of solutions initially containing only S'
3.2 foreach a in A do
3.3     S ← isolate(S', a)
3.4     Add S to 𝒮
3.5     foreach cycle C in S' do
3.6         S ← swap_cycle(S', C, a)
3.7         Add S to 𝒮
3.8 return the best solution in 𝒮
```

solution that improves the current one $S'$. The first improvement variant obtained better results since the complete local search has to explore a costly neighbourhood, where for each swap of agent $a$, we need to run the TSP heuristics to rebuild a cycle.

# 6 Computational Results

All computations were performed in a single thread of an Intel Core™i7-4790 processor at 3.60GHz with 16GB of RAM, with Linux. The algorithms were implemented in C++, using the Graph library Lemon [14]. To solve the linear relaxation we used the Gurobi solver [10].

## 6.1 Instances

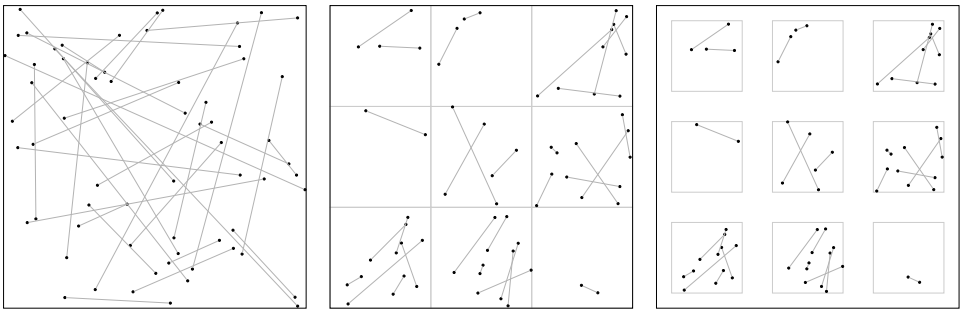
Since, to the best of our knowledge, there is no previous work on the R-SMCP, we have tested our algorithms on two types of instances: type 1 is a set of instances from the multi-commodity one-to-one pickup-and-delivery traveling salesman problem (m-PDTSP), and type 2 is a set of newly random generated instances.

Hernández-Pérez and Salazar-González [12] generated a set of instances to the m-PDTSP. For each instance, they generated $2n-2$ uniformly random points with coordinates from $-500$ to 500, a vertex in position 0 with coordinates $(0,0)$ and a vertex in position $2n-1$ also with coordinates $(0,0)$ (corresponding to Class 3 of [12]). We only take into account the vertex distribution of these instances. For each $i \in \{0,...,n-1\}$, we consider vertex $i$ as a pickup point of an agent and $i+n$ as its corresponding delivery point. The instances have 6, 11 and 16 agents, with a total of 210 instances. This set of instances is the type 1.

We also generated a set of instances having 16, 32, 64, 128 and 256 agents, where vertices corresponds to points distributed in a square of dimensions $100000 \times 100000$. The square is divided in $1 \times 1$, $2 \times 2$, $3 \times 3$, $4 \times 4$ and $5 \times 5$ frames, and each pair of pickup and delivery is in the same frame. The space between frames, which we call a wall, has 0%, 10%, 20%, 30% or 40% of the frame's size. The wall can be seen as a rectangle separating the different frames (see Figure 1). The location of each point is chosen uniformly at random. For each combination of number of agents, number of frames, and wall size, 3 instances were generated with different seeds. Notice that for the instances with division $1 \times 1$ there is no wall. Therefore, we generated a set of 315 instances. This set of instances is the type 2 and it can be found in the Laboratory of Optimization and Combinatorics website [15].

## 6.2 Final Results

We first compared the GRASP algorithm in different versions, each version using a different distance measure between agents (see Section 5.1). Figure 2 shows the performance profiles [3] comparing the different versions of the GRASP algorithm. This graphic is build as follows.



**(a)** 1x1 **(b)** 3x3 **(c)** 3x3 wall 0.3

**Figure 1** Instances randomly generated with 16 agents, in (a) a $1 \times 1$ frame with no wall, in (b) a $3 \times 3$ frame with 0% wall and in (c) a $3 \times 3$ frame with 30% wall.

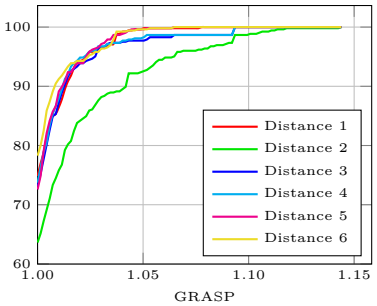
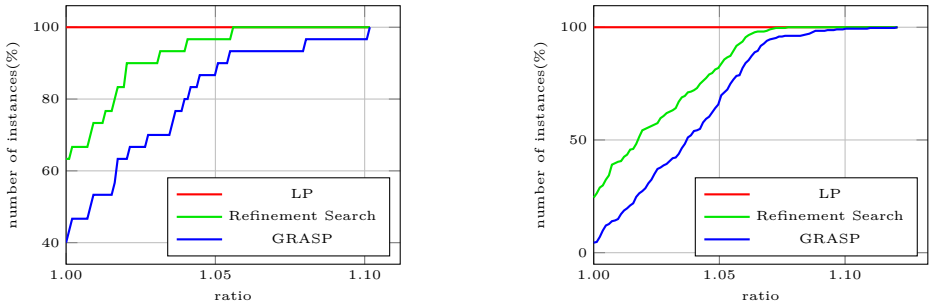



**Figure 2** Comparison between the GRASP algorithm running with different distances. The line that appears closer to the upper-left corner gives the best results, in this case distance 6.

First of all, we execute each version of the algorithm on all instances, obtaining the solution cost of each algorithm version for each instance. The performance profile graphic consists of a curve for each algorithm version. A curve of an algorithm version is constructed in the following manner: for each instance, we compute the ratio between the versions's solution cost and the best solution obtained for that instance among all algorithm versions.

The performance profile of an algorithm version is a plot of the cost ratios ($x$ axes) versus the percentage of instances that this algorithm version could solve within at most that ratio ($y$ axes). From Figure 2 we see that, except for the version using distance 2, all versions solved more or less 75% of the instances with ratio 1, meaning that they found 75% of the best solutions. In fact, the version using distance 6 solved almost 80% of the instances with ratio 1, and 100% of the instances with ratio 1.05, meaning that it found 80% of the best solutions and the other solutions had a cost at most 5% higher than the best solutions found. So we decide to use distance 6 in the GRASP algorithm. As for the threshold $T$ of the adaptive method, defined on Section 5.2, we tested the values of 0.2, 0.4, 0.6, 0.8, and 1, and in general the best results were obtained with $T = 0.6$.

The GRASP algorithm was set to run for $\log_2(n)$ iterations, where $n$ is the number of agents in that instance. Figure 3 presents the performance profiles of the GRASP algorithm and the Refinement Search. We also included the results of the lower bound obtained by solving the LP. For the type 1 instances, the Refinement Search found optimal solutions for approximately 60% of the instances, while GRASP found 40% of optimal solutions, and the Refinement Search found solutions to all instances with cost at most 6% higher than the lower bound given by the LP solution. For the type 2 instances we obtained similar results.




**(a)** Comparison for instances of type 1.

**(b)** Comparison for instances of type 2.

**Figure 3** Comparison between GRASP and Refinement Search heuristics.

**Table 1** Results separated by instance classes.

| | | GAP(%) | | time (s) | |
|---|---|---|---|---|---|
| Classes | # inst | RS | GRASP | RS | GRASP |
| m-PDTSP | 210 | **0.76** | 1.99 | **0.02** | 0.01 |
| 1x1 | 15 | **4.64** | 5.58 | **42.02** | 306.35 |
| 2x2 | 75 | **2.99** | 3.79 | **30.69** | 89.09 |
| 3x3 | 75 | **2.39** | 3.75 | **31.15** | 63.25 |
| 4x4 | 75 | **1.82** | 3.44 | **29.93** | 78.58 |
| 5x5 | 75 | **1.56** | 3.43 | **29.56** | 86.52 |
| W0.0 | 75 | **3.72** | 5.13 | **34.45** | 290.09 |
| W0.1 | 60 | **3.27** | 4.91 | **30.83** | 85.98 |
| W0.2 | 60 | **2.00** | 3.45 | 29.59 | **9.04** |
| W0.3 | 60 | **1.11** | 2.35 | 29.16 | **7.83** |
| W0.4 | 60 | **1.06** | 2.30 | 29.54 | **7.93** |
| rg-016 | 63 | **0.38** | 1.81 | 0.03 | **0.01** |
| rg-032 | 63 | **1.16** | 2.69 | 0.22 | **0.10** |
| rg-064 | 63 | **2.44** | 4.19 | 1.98 | **1.34** |
| rg-128 | 63 | **3.51** | 4.69 | **15.62** | 26.71 |
| rg-256 | 63 | **4.04** | 5.11 | **136.59** | 422.69 |
| total average | | 1.69 | 3.02 | 18.54 | 54.11 |

In Table 1 we present a comparison of the Refinement Search (RS) and GRASP separating the results by the type and properties of the instances. In this table, in each line, we present the average GAP, and time, of the instances of a certain class compared to the lower bound obtained by solving the LP. The line m-PDTSP contains the results of the instances of type 1. For the type 2 instances, we sub-divided it depending on some properties. The "$1 \times 1$" class contains results of the instances with one frame, while the "$2 \times 2$" contains the results of the instances with 4 frames and so on. The "W0.x" class contains the results of the instances with wall separation of $(10 \times x)\%$ of the size of the frame. In the last lines, the "rg-$n$" classes, we separated the instances by the number of agents $n$. The Refinement Search algorithm consistently obtained the lowest gaps and, in general, was also faster than the GRASP algorithm. We observe that the cost of the Refinement Search solution was, on average, 1.29% lower than the cost of the GRASP solution. We conjecture that Refinement Search performs better than GRASP because the latter works with agents as vertices of a modified graph, while the former deals with the original graph with pairs of vertices.

## References


1 Jose Caceres-Cruz, Pol Arias, Daniel Guimarans, Daniel Riera, and Angel A Juan. Rich vehicle routing problem: Survey. *ACM Computing Surveys (CSUR)*, 47(2):32, 2015.

2 William J Cook, WH Cunningham, WR Pulleyblank, and A Schrijver. *Combinatorial optimization*. Springer, 2009.

3 Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

4 Özlem Ergun, Gültekin Kuyzu, and Martin W. P. Savelsbergh. Reducing truckload transportation costs through collaboration. *Transportation Science*, 41(2):206–221, 2007.

5 Özlem Ergun, Gültekin Kuyzu, and Martin W. P. Savelsbergh. Shipper collaboration. *Computers & OR*, 34(6):1551–1560, 2007.

6 Thomas A Feo and Mauricio GC Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, 8(2):67–71, 1989.

7 Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995.

8 Ralph E Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

9 Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

10 Gurobi Optimization, Inc. Gurobi optimizer reference manual, 2016. URL: `http://www.gurobi.com`.

11 Dan Gusfield. Very simple methods for all pairs network flow analysis. *SIAM Journal on Computing*, 19(1):143–155, 1990.

12 Hipólito Hernández-Pérez and Juan-José Salazar-González. The multi-commodity one-to-one pickup-and-delivery traveling salesman problem. *European Journal of Operational Research*, 196(3):987–995, 2009.

13 Ismail Karaoglan, Fulya Altiparmak, Imdat Kara, and Berna Dengiz. The location-routing problem with simultaneous pickup and delivery: Formulations and a heuristic approach. *Omega*, 40(4):465–477, 2012.

14 Lemon. Library for efficient modeling and optimization in networks, 2016. URL: `http://lemon.cs.elte.hu/trac/lemon/`.

15 LOCo - UNICAMP. Laboratory of optimization and combinatorics, 2016. URL: `http://www.loco.ic.unicamp.br`.

16 Sophie N. Parragh, Karl F. Doerner, and Richard F. Hartl. A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, 58(1):21–51, 2008.

17 Mauricio GC Resende and Celso C Ribeiro. *Optimization by GRASP: Greedy Randomized Adaptive Search Procedures*. Springer, 2016.

18 Juan-José Salazar-González. The Steiner cycle polytope. *European Journal of Operational Research*, 147(3):671–679, 2003.

19 Monika Steinová. Approximability of the minimum Steiner cycle problem. *Computing and Informatics*, 29(6+):1349–1357, 2012.

20 Paolo Toth and Daniele Vigo. *Vehicle routing: problems, methods, and applications*, volume 18. Siam, 2014.