Les Cahiers du GERAD ISSN: 0711–2440

Decomposition strategies for solving the spring sweeping routing problem

A. Lamghari, O. Foutlane, J. F. Audy

G–2025–10

January 2025

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2025
– Bibliothèque et Archives Canada, 2025

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2025
– Library and Archives Canada, 2025

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Decomposition strategies for solving the spring sweeping routing problem

Amina Lamghari [a, b]

Omar Foutlane [a, b]

Jean-François Audy [a]

[a] *Department of Management, Université du Québec à Trois-Rivières, Trois-Rivières (Qc), Canada, G8Z 4M3*

[b] *GERAD, Montréal (Qc), Canada, H3T 1J4*

amina.lamghari@uqtr.ca
omar.foutlane@uqtr.ca
jean-francois.audy@uqtr.ca

January 2025
Les Cahiers du GERAD
G–2025–10

Abstract : This paper addresses the problem of efficiently routing vehicles for spring sweeping operations in countries that spread sand and gravel on roads in winter. Given a road network, one sweeping fleet, and several depots, we seek to route the fleet over the network so as to minimize the time needed to sweep all required road segments and reduce the distance covered by deadheading trips. The schedule is subject to service time window constraints for highways, time constraints of the crew, and visits to depots along the route. The little work that has been carried out so far to solve this practical problem treat it as an arc routing problem. Instead, we transform the arc routing problem into an equivalent node routing problem and propose three decomposition-based approaches to solve the latter. Tests are conducted on randomly generated instances as well as on a real-world case in Victoriaville, Québec, Canada.

Keywords : OR in service industries, street sweeping problem, vehicle routing, decomposition strategies, matheuristics

Acknowledgements: This work was supported by the Canadian Natural Sciences and Engineering Research Council (grant number RGPIN–2019–07172). This support is gratefully acknowledged. We thank our industry partner *Arseno Balayage* for their support and for providing the data used in the computational analysis.

1 Introduction

In the interest of driving safety, road authorities try to maintain good traction when roads are covered by ice and snow, which make surfaces treacherously slippery. To improve traction, abrasives, i.e., sand and gravel, are spread on roads and highways. While this practice makes the roadways safer in winter, loose abrasives negatively affect traction on dry surfaces. Removing leftover abrasives from roads and highways in the spring is thus necessary both for safety and for reducing airborne dust and the amount of abrasive material that enters water courses and groundwater.

The work of sweeping abrasives is done by private contractors who are awarded contracts to sweep and dispose of leftover material on specified roads and highways in a given contract area. In the sweeping process that we consider, a contractor uses a fleet that consists of two sweepers, two dump trucks, and a safety vehicle that follows the sweepers and trucks to warn vehicles approaching from behind. The first sweeper is the principal sweeper, and it continuously loads a dump truck while sweeping. When the truck is full, it goes to the assigned dump site to offload the swept abrasive material, and the second truck takes its place. The secondary, or finishing sweeper, picks up what the first sweeper missed, but it does not fill as quickly as the principal sweeper. The presence of two trucks ensures that the primary sweeper is never idle, and hence the fleet operates in a continuous process. The spring sweeping season is only a few weeks long, and so having an efficient vehicle routing plan is of utmost importance. Because of the small profit margins involved, a major concern of every contractor is to devise a plan that minimizes the completion time and thereby the costs.

In devising the routing plan, two main operational constraints must be respected. Since a crew cannot work 24 hours per day, work must be done in shifts. Shift changes must occur at pre-designated depots, which are typically work sites where equipment can be stored and serviced. A crew starts at a depot, services a single route, and returns to a depot within the shift time limit (8-10 hours) to hand over the fleet to another crew. The other main constraint is that roads designated as highways can only be swept at night, according to law, so as not to excessively hinder heavier daytime traffic from moving efficiently. Secondary roads and streets can be swept during the day or during the night. It is worth noting that a contract (a sector) consists of non-contiguous areas because of jurisdictional boundaries and geography, among other things. This means that deadheading trips are necessary, i.e., the sweeping fleet must traverse some road segments without sweeping them.

The *spring sweeping routing problem* (SSRP) consists of finding a route to be performed by the sweeping fleet, such that all road segments of a given contract are swept, all operational constraints are satisfied, and the total cost of the operation is minimized. The total cost is defined as the sum of fixed and variable costs. The variable costs depend on the distance the sweeping fleet must travel, including the deadheading trips, while the fixed costs are associated with the cost of equipment, salaries, and renting depots. Minimizing the time it takes to complete the sweeping tasks is equivalent to minimizing the total costs.

The goal of this paper is to develop a decision-support tool to assist spring sweeping companies in making routing decisions to reduce their operating costs and improve profitability, while meeting contractual obligations. To do so, we propose a mathematical formulation of the SSRP described above and three decomposition-based strategies to solve it. Such methodologies are capable of solving large-scale instances of practical relevance and should be able to produce good solutions in a reasonable amount of time.

Various real-world routing problems have similarities with the SSRP, including the street sweeping problem and the vehicle routing problem for snow plowing and salt spreading operations. The latter is known to be difficult and site specific due to the range of factors and the various operational constraints that influence how and when winter road maintenance operations are carried out (Perrier et al., 2007b). The problem has been extensively studied in the literature. For an overview of the problem variants and the optimization techniques developed for solving such problems, the reader is referred to the exhaustive surveys in Perrier et al. (2007a,b). The literature on the street sweeping problem, which

consists of routing and scheduling street sweepers, is, on the other hand, relatively sparse. The problem can be modelled as a directed rural postman problem with additional constraints (Eiselt et al., 1995). In what follows, we describe the constraints considered in the literature and outline the methods developed to solve the proposed models.

In urban areas, streets are often one-way and can only be swept at specific times due to parking regulations. This constraint was considered by Bodin and Kursh (1978, 1979), who developed an algorithm for designing balanced workload sweeper routes in New York City and Washington DC. In Eglese and Murdock (1991), the authors considered rural areas where parking regulations are not a concern, and all roads to be swept are two-way. The constraints accounted for in this case are capacity constraints: each sweeper has a bin with a limited capacity, necessitating visits to tip-off sites to empty the bin when full, potentially several times a day. The problem is to construct routes starting and ending at the depot such that each required road is swept, and to determine which tip-off sites to use to empty each sweeper in order to minimize deadheads. The authors developed a constructive heuristic to produce a feasible solution and designed a computerized system to assist planners in constructing and evaluating routing plans. The decision support system has been implemented in Lancashire, England.

While Bodin and Kursh (1978, 1979) and Eglese and Murdock (1991) considered either one- or two-way streets, Blazquez et al. (2012) considered both types. Parking restrictions were disregarded, similar to the study by Eglese and Murdock (1991). By assuming that there is only one sweeper with sufficient capacity to sweep all required street segments without needing to empty the bin, the authors showed that the problem reduces to the asymmetric travelling salesman problem. A *nearest neighborhood* procedure, where each street segment is iteratively added to the solution, was developed to solve the problem. This procedure was tested on real data from Santiago, Chile. Cerrone et al. (2014) addressed the integration of two problems: scheduling parking restrictions and designing sweeper routes. The objective was to minimize the distance travelled, assuming that there is only one sweeper servicing all required streets over two days. An arc formulation of the integrated problem was proposed and solved using a genetic algorithm. Tests were performed on instances involving up to 882 arcs representing the street segments that require sweeping.

Recent models have incorporated more characteristics encountered in practice. For instance, Yu et al. (2019) focused on minimizing the time for service completion considering real-time traffic data in the city of Zhengzou, China. An arc formulation of the problem was developed and solved using Gurobi. Yurtseven and Gokce (2019) considered the case of electric-powered sweepers in Izmir, Turkey. Given a heterogenous fleet of sweepers with different capacities and battery levels, the problem considered was to design a set of routes such that each street segment is swept while minimizing energy consumption required for the sweeping, travelling, and disposal operations. The problem was formulated as an arc routing problem and solved using CPLEX. More recently, Kraiem et al. (2025) investigated the special case of spring sweeping on highways and provincial roads (the SSRP), which is also the focus of this paper. The authors proposed an arc routing formulation for the SSRP, which was solved using CPLEX. It was concluded that the problem is computationally intractable. Within a two-hour time limit, CPLEX was able to solve only small instances with up to 24 road segments.

The above literature review shows that most research on the street sweeping problem has focused on urban areas. The main exceptions we are aware of are the problem studied by Eglese and Murdock (1991), who discussed the case of rural areas and the more recent problem studied by Kraiem et al. (2025), who considered a context similar to that addressed in this paper, i.e., spring sweeping of highways and provincial roads. Most optimization models for the street sweeping problem are arc routing formulations since the sweeping activity involves the traversal and service of arcs of the road network. To solve the proposed models, most studies focused on constructive heuristics or used off-the-shelf solvers.

Our paper takes a novel approach by proposing a new node routing formulation for the SSRP. Node routing is a more suitable and effective modeling approach to handle the SSRP than arc routing because, on highways and provincial roads, spring sweeping takes place on road segments scattered

over a wide area and not on a continuous collection of roads, as is common in urban areas. Our contributions also consist of developing three solution approaches based on decomposition to tackle the problem. The first one, *Infrastructure-based decomposition* (IBD), takes advantage of the structure of the problem and builds sub-problems by partitioning the roads into highways and other roads. The second one, *Proximity-based decomposition* (PBD), first groups roads within a short distance of each other and then creates the sub-problems accordingly. The last one, *Shift-only decomposition* (SOD) does not decompose the set of roads, but rather the set of shifts and thus can be seen as a time-decomposition approach. We conduct computational experiments using a real-world case in Victoriaville, Québec and randomly generated instances of various sizes. We analyze the effects of the number of highway segments and the number of depots on the performance of the solution approaches. Our experiments and analysis indicate that the SSRP is a computationally challenging problem. Out of the three proposed decomposition approaches, we found only one, the PBD, to be efficient on randomly generated large instances of practical interest, obtaining near-optimal solutions within a reasonable amount of time. When applied to the real-world instance, this PBD method is able to achieve significant cost savings compared to the existing manual solution.

The remainder of the paper is divided into four sections. Section 2 gives a formal description of the problem under study and presents a mathematical formulation of it. Section 3 describes the solution methodology we have developed. Section 4 gives the computational results. This is followed by conclusions in Section 5.

2 Problem description and formulation

In this section, we first introduce a formal description of the SSRP addressed in this paper and some notation and terminology that will be used throughout the paper. We then present the proposed node routing formulation.

2.1 Problem definition

We are given a set of tasks $\mathcal{N}$, each corresponding to a road segment that must be swept by an available sweeping fleet, and a set of shifts $\mathcal{S}$ over which the tasks can be performed. With each task $i \in \mathcal{N}$ is associated a positive duration δ_i that indicates how long it will take to perform the task, and with each shift $s \in \mathcal{S}$ is associated a time limit $\overline{U}^s$, which dictates the number of tasks that can be completed during the shift. We assume, without loss of generality, that $\delta_i \leq \overline{U}^s$ for all $i \in \mathcal{N}$ and $s \in \mathcal{S}$, which means that any given task can be completed within a single shift.

The shifts are specified either as day or night shifts. We denote the day shifts by $\mathcal{S}_d$ and the night shifts by $\mathcal{S}_n$ ($\mathcal{S} = \mathcal{S}_d \cup \mathcal{S}_n$ and $\mathcal{S}_d \cap \mathcal{S}_n = \emptyset$). The tasks are partitioned into two main classes: $\mathcal{H}$ and $\mathcal{O}$ representing highways and other roads, respectively ($\mathcal{N} = \mathcal{H} \cup \mathcal{O}$ and $\mathcal{H} \cap \mathcal{O} = \emptyset$). Tasks in $\mathcal{H}$ must be performed during night shifts to minimize traffic disruptions, whereas those in $\mathcal{O}$ can be performed during either day or night shifts.

Each shift $s \in \mathcal{S}$ incurs fixed and variable costs. Fixed costs include fleet costs and costs of renting depots, which serve as hubs where equipment is maintained and crews are changed at the end of each shift. The depot where a shift ends is therefore the same depot where the next shift begins. We denote the set of depots by $\mathcal{D}$. Variable costs include the costs of fuel and are a function of the mileage. Usually, road segments to be swept (tasks) are not connected, so moving from one task to another or from a depot to a task might require deadheading along some road segments. Variable costs are thus proportional to the distance traveled.

The goal of the SSRP is to assign tasks to shifts so as to minimize the global cost and satisfy the constraints described above. It can be modelled as a routing problem that seeks to determine a set of routes, each associated with a shift, in conjunction with the depots at which each route starts and ends, such that the routes are connected and cover all tasks within appropriate periods, the duration of each

route does not exceed the shift length, and the global cost is minimized. Recall that the global cost depends on the global distance traveled (variable costs) and the number of shifts used (fixed costs). Therefore, minimizing the global cost is equivalent to minimizing the completion time.

In Kraiem et al. (2025), the authors formulate the SSRP as an arc routing problem, a natural formulation because demand for sweeping is located on arcs in a road network. The model involves six types of variables: shift variables (binary), deadheading variables (binary), service variables (binary), time variables (continuous), and end-shift variables (continuous). Shift variables indicate whether a shift is used or not. Deadheading and service variables are four-index variables indicating if an arc is swept or traversed in a given shift and the position of the route in which it appears. Time variables are also four-index variables. They specify the starting time of service or traversal. Finally, end-shift variables are defined for every shift and represent the completion time of the shift. The drawback of this arc routing approach is that it results in a very large model, with $|\mathcal{S}|+2|\mathcal{A}||\mathcal{S}||\mathcal{K}|$ binary variables and $|\mathcal{S}|+|\mathcal{A}||\mathcal{S}||\mathcal{K}|$ continuous variables ($|\mathcal{A}|$ and $|\mathcal{K}|$ being the number of arcs and possible positions, respectively). As pointed out in Kraiem et al. (2025), this model is intractable even for small instances. In the next section, we present an equivalent node routing formulation, which is more compact and has fewer binary variables. To transform the arc routing problem into a node routing problem, we basically adapt standard techniques described in Baldacci and Maniezzo (2006), Longo et al. (2006), and Tagmouti et al. (2007), among others.

2.2 Problem formulation

We consider a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{A}$ is the set of arcs. $\mathcal{V}$ is partitioned into three subsets $\mathcal{D}$, $\mathcal{N}$, $\{0, e\} \subset \mathcal{V}$: $\mathcal{D}$ is the set of depots, $\mathcal{N}$ is the set of tasks, and 0 and e are dummy start and end terminals used to model the start and end of a shift, respectively. We assume that there is a complete interconnection between the vertices of $\mathcal{N}$, that the start terminal 0 is connected to all the depot nodes, that all depot nodes are connected to all tasks nodes, and that all depot nodes are connected to the end terminal, e. That is, there are no arcs from the start terminal to tasks ($\mathcal{A} \cap (\{0\} \times \mathcal{N}) = \emptyset$) or from tasks to the end terminal ($\mathcal{A} \cap (\mathcal{N} \times \{e\}) = \emptyset$) or between depots ($\mathcal{A} \cap (\mathcal{D} \times \mathcal{D}) = \emptyset$). Through this paper, we write $\mathcal{A} = \mathcal{A}_{0D} \cup \mathcal{A}_R \cup \mathcal{A}_{De}$ where $\mathcal{A}_{0D} = \{(0, d) : d \in \mathcal{D}\}$, $\mathcal{A}_R = \{(i, j) : i \in \mathcal{D}, j \in \mathcal{N}\} \cup \{(i, j) : i, j \in \mathcal{N}, i \neq j\} \cup \{(i, j) : i \in \mathcal{N}, j \in \mathcal{D}\}$, and $\mathcal{A}_{De} = \{(d, e) : d \in \mathcal{D}\}$. Figure 1 shows a small graph example with $\mathcal{D} = \{d_1, d_2\}$ and $\mathcal{N} = \{i, j, k\}$.

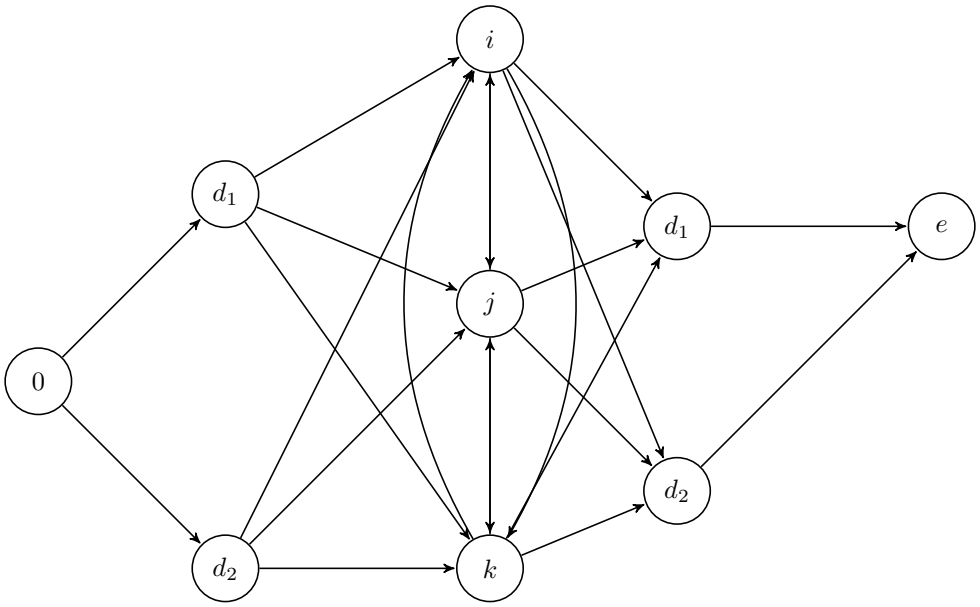

Figure 1: A small graph example with two depots and three tasks.

A cost c_{ij} and a travel time τ_{ij} are associated with each arc $(i,j) \in \mathcal{A}$. For arcs in $\mathcal{A}_{0D}$, the cost represents the fixed cost that has to be paid if a shift is used in the schedule, and the travel time is set equal to zero. For arcs in $\mathcal{A}_{De}$, both the cost and the travel time are set equal to zero. Finally, for the other arcs (i,j) in subset $\mathcal{A}_R$, the cost and travel time are those of the shortest path between i and j. With each vertex $i \in \mathcal{V}$ is associated a service time σ_i. For $i \in \mathcal{N}$, this is the time needed to perform task i, and therefore we have $\sigma_i = \delta_i$. For any other vertex in $\mathcal{D} \cup \{0, e\}$ the service time is set equal to zero.

Two types of decision variables are used in the mathematical model: binary variables $x^s_{ij} \in \{0,1\}$ equal to 1 if and only if vertex $j \in \mathcal{V}$ is visited after vertex $i \in \mathcal{V}$ in shift $s \in \mathcal{S}$, and continuous variables $y^s_i \geq 0$ indicating when the sweeping fleet arrives at location i. The latter are only well-defined when vertex i is actually visited in shift s.

Using the notation above, the SSRP can be modeled as follows:

$$(P) \qquad \min \sum_{s\in\mathcal{S}} \sum_{(i,j)\in\mathcal{A}} c_{ij} x^s_{ij} \tag{1}$$

$$s.t. \qquad \sum_{s\in\mathcal{S}} \sum_{i\in\mathcal{D}\cup\mathcal{N}} x^s_{ij} = 1 \qquad \forall j \in \mathcal{O} \tag{2}$$

$$\sum_{s\in\mathcal{S}_n} \sum_{i\in\mathcal{D}\cup\mathcal{N}} x^s_{ij} = 1 \qquad \forall j \in \mathcal{H} \tag{3}$$

$$\sum_{d\in\mathcal{D}} x^s_{0d} \leq 1 \qquad \forall s \in \mathcal{S} \tag{4}$$

$$\sum_{d\in\mathcal{D}} x^s_{0d} = \sum_{d\in\mathcal{D}} x^s_{de} \qquad \forall s \in \mathcal{S} \tag{5}$$

$$x^{s+1}_{0d} \leq x^s_{de} \qquad \forall d \in \mathcal{D}, s \in \mathcal{S} \tag{6}$$

$$\sum_{i:(i,k)\in\mathcal{A}_{0D}\cup\mathcal{A}_R} x^s_{ik} = \sum_{j:(k,j)\in\mathcal{A}_R\cup\mathcal{A}_{De}} x^s_{kj} \qquad \forall k \in \mathcal{D}\cup\mathcal{N}, s \in \mathcal{S} \tag{7}$$

$$y^s_0 = 0 \qquad \forall s \in \mathcal{S} \tag{8}$$

$$y^s_i \leq \overline{U}^s \sum_{(i,j)\in\mathcal{A}_R} x^s_{ij} \qquad \forall i \in \mathcal{D}\cup\mathcal{N}, s \in \mathcal{S} \tag{9}$$

$$y^s_e \leq \overline{U}^s \qquad \forall s \in \mathcal{S} \tag{10}$$

$$y^s_i + \sigma_i + \tau_{ij} - y^s_j \leq (\overline{U}^s + \sigma_i + \tau_{ij})(1 - x^s_{ij}) \qquad \forall (i,j) \in \mathcal{A}, s \in \mathcal{S} \tag{11}$$

$$x^s_{ij} = 0 \qquad \forall (i,j) \in \mathcal{A}, s \in \mathcal{S} : \sigma_i + \tau_{ij} > \overline{U}^s \tag{12}$$

$$x^s_{ij} \in \{0,1\} \qquad \forall (i,j) \in \mathcal{A}, s \in \mathcal{S} \tag{13}$$

$$y^s_i \geq 0 \qquad \forall i \in \mathcal{V}, s \in \mathcal{S}. \tag{14}$$

The objective function (1) minimizes the sum of fixed costs associated with the shifts and travel costs. Constraints (2) and (3) guarantee that each task is completed exactly once in an appropriate shift. Constraints (4) and (5) state that a shift is used only if the sweeping fleet leaves the start terminal to go to a depot, in which case it must enter the end terminal from a depot at the end of the shift. Constraints (6) capture the interdependence between shifts. They impose that the depot where a shift ends is the same depot where the next shift begins. Together with constraints (7) this ensures that consecutive paths are formed for each shift. Constraints (8)–(11) ensure that the y variables track the time elapsed from the beginning of a shift and guarantee that the maximum length of any shift is never exceeded. They also eliminate subtours. Constraints (12)–(14) define the domains of the decision variables and fix some of them according to temporal considerations.

The SSRP is an extension of the vehicle routing problem (VRP) (if $|\mathcal{D}| = 1$ and $\mathcal{H} = \emptyset$, the problem can be viewed as a VRP by considering each of the $\mathcal{S}$ shifts to be a vehicle). Because only small instances of the VRP can be solved exactly, it is clear that one cannot solve the SSRP with formulation (1)–(14), even though, according to our preliminary tests, this formulation is computationally more efficient than its arc routing counterpart proposed in Kraiem et al. (2025). Decomposing the problem into a set of smaller sub-problems appears a methodological avenue worth taking. In the next section, we describe the three decomposition strategies that we propose.

3 Solution methodology

Advances in the sophistication and power of MIP solvers in recent years have made it possible to solve difficult combinatorial optimization problems such as the one addressed in this paper. Yet, there are still limits to the sizes that these solvers can handle in a reasonable amount of time. Supplying an integer feasible solution to the solver as a warm start can significantly speed up the solution process (Bertsimas and Weismantel, 2005). Not only does the warm start solution provide an initial upper bound on the optimal solution, allowing more efficient pruning of the search tree, but it also serves as a starting point for local search heuristics (Bertsimas and Dunn, 2019). The effectiveness of this approach depends on the quality of the warm start. Hence, before solving, it is useful to quickly and heuristically find a good feasible solution. In our case, this can be done by taking advantage of the tractability of instances with just a few tasks. We start by decomposing the problem into a series of sub-problems, each associated with a subset of tasks and/or a subset of shifts. These sub-problems are solved separately, and then their solutions are merged to obtain a feasible solution of the SSRP. Because this solution will be improved, solving the sub-problems to proven optimality is not necessary.

To summarize, we use a two-phase procedure. In the first phase, a decomposition-based heuristic is used to build an initial solution, while in the second phase the so-obtained solution is refined. It is supplied to a state-of-the-art MIP solver as a warm start for the solution process. The solution procedure is summarized in Algorithm 1. The approaches proposed to decompose the problem (Line 3 of Algorithm 1) are discussed in detail in sections 3.1–3.3.

Algorithm 1 Solution procedure

Input: an instance of the SSRP; optimality tolerance `Tol%`; maximum CPU time `Tmax`.
Output: solution x^{best}.
1: Set $x^{init} \leftarrow \emptyset$ and $x^{best} \leftarrow \emptyset$ (initial and best feasible solution)
2: **Phase I: Generating a warm start (initial) solution**
3: Decompose the instance into K smaller sub-problems, SP^k
4: $k \leftarrow 1$
5: **while** $k \leq K$ and $time <$ `Tmax` **do**
6: Solve SP^k to get a partial route x^k within `Tol%` of optimality
7: $x^{init} \leftarrow x^{init} \cup x^k$
8: $k \leftarrow k + 1$
9: **end while**
10: **Phase II: Improving the solution**
11: **if** $time <$ `Tmax` **then**
12: $x^{best} \leftarrow x^{init}$
13: Solve formulation (1)–(14) using x^{init} as a warm start for the solution process
14: Update x^{best}
15: **end if**
16: **return** x^{best}.

3.1 Infrastructure-based decomposition

Recall that tasks are differentiated according to whether they are highways or other roads ($\mathcal{N} = \mathcal{H} \cup \mathcal{O}$ and $\mathcal{H} \cap \mathcal{O} = \emptyset$) and that shifts are either day or night shifts ($\mathcal{S} = \mathcal{S}_d \cup \mathcal{S}_n$ and $\mathcal{S}_d \cap \mathcal{S}_n = \emptyset$). Recall also that tasks in $\mathcal{H}$ must be performed during night shifts $\mathcal{S}_n$, whereas those in $\mathcal{O}$ can be performed during either day or night shifts. A natural idea is therefore to decompose the problem into two sub-problems: one associated with the pair ($\mathcal{H}$, $\mathcal{S}_n$) and one associated with the pair ($\mathcal{O}$, $\mathcal{S}_n \cup \mathcal{S}_d$). Both sub-problems have the same definition and the same structure as the original problem (P), but they consist of a limited number of variables in contrast to (P), which consists of all variables.

Since tasks in $\mathcal{H}$ have more restrictions than tasks in $\mathcal{O}$, it makes sense to start by solving the sub-problem associated with the pair ($\mathcal{H}$, $\mathcal{S}_n$). Referring to Algorithm 1, this sub-problem is denoted by SP^1. We solve SP^1 to the predefined optimality tolerance `Tol%` and extract the binary variables that take value 1 in the solution. The set of such variables is denoted by $\mathcal{B}^1$. Recall that, by construction, each route visiting vertices $0, d, v_1, v_2, \ldots, v_h, d', e$, where $d, d' \in \mathcal{D}$ and $v_1, v_2, \ldots, v_h \in \mathcal{N}$, corresponds to a feasible shift that starts at depot d, consecutively performs tasks $v_1, v_2, \ldots, v_h$, and ends at depot

d' (see constraints (4)–(7)). Let $\overline{\mathcal{B}^1}$ be the subset of $\mathcal{B}^1$ containing only variables related to arcs whose head and tail are tasks and let $\underline{\mathcal{B}^1}$ denote its complement. We fix variables in $\overline{\mathcal{B}^1}$ to their current value (one), leave the variables in $\underline{\mathcal{B}^1}$ free, and solve the original problem (P) (referring to Algorithm 1, fixing the variables corresponds to generating the partial route x^1, and the problem solved after the variable fixing corresponds to the second sub-problem, SP^2). Leaving variables in $\underline{\mathcal{B}^1}$ free can be seen as "breaking" the links from and to the depots to avoid locking short shifts. That way, further tasks can be added to shifts in $\mathcal{S}_n$ when solving SP^2, which should produce an initial solution better than the one that would be produced if variables in $\underline{\mathcal{B}^1}$ were also fixed. Recall that each shift incurs a fixed cost, so the objective is to minimize the number of shifts (see the objective function (1)).

Obviously, the heuristic described above does not help in solving instances where there are no highways. Also, it might not be successful at solving large scale practical instances with highways since the two sub-problems could be large and very time consuming to solve. In the next section, we present an alternative decomposition strategy that avoids these shortcomings.

3.2 Proximity-based decomposition

We extend the idea of the *infrastructure-based decomposition* by considering finer partitions of the tasks and the shifts. Let $\{\mathcal{N}^1,\dots,\mathcal{N}^K\}$ be a partition of $\mathcal{N}$. One can also partition the shifts into $\{\mathcal{S}^1,\dots,\mathcal{S}^{K'}\}$, with $K' \geq K$. We start by solving for routes that consider only tasks and shifts in subsets $\mathcal{N}^1$ and $\mathcal{S}^1$. This problem could be infeasible if the number of shifts in $\mathcal{S}^1$ is insufficient to cover all tasks in $\mathcal{N}^1$. We address this by simply extending the subset of shifts to include the next subset, $\mathcal{S}^2$, and solve again. This is repeated until feasible routes are obtained (because the number of shifts is not fixed *a priori*, feasibility is guaranteed after a finite number of iterations). We then break the links from and to depots, fix the resulting partial routes, include tasks and shifts from the next subsets, and solve again. As stated earlier, breaking the links from and to depots is desirable because if we decide not to do so, we commit ourselves to locking shifts, which precludes adding more tasks to them. The cycle of fixing partial routes, adding tasks, progressively adding shifts, and solving repeats until all tasks are assigned and thus we have solved the original problem. A more formal description of the procedure is given in Algorithm 2. Note that the sub-problem SP^k solved at each iteration is a restriction of the original problem since the former contains only a subset of the variables of the latter. Therefore, we refer to it as $P(\bigcup_{p=1}^{k} \mathcal{N}^p, \bigcup_{p=1}^{q} \mathcal{S}^p, \overline{\mathcal{B}^1})$, where $\overline{\mathcal{B}^1}$ denotes the set of fixed x^s_{ij}.

Algorithm 2 Proximity-based decomposition

Input: a partition of the set of tasks $\{\mathcal{N}^1,\dots,\mathcal{N}^K\}$; a partition of the set of shifts $\{\mathcal{S}^1,\dots,\mathcal{S}^{K'}\}$.
Output: a feasible solution $(\hat{x},\hat{y})$ to (P).

Set $(\hat{x},\hat{y}) \leftarrow \emptyset$ and $\overline{\mathcal{B}^1} \leftarrow \emptyset$
$k \leftarrow 1$
$q \leftarrow 1$
while $k \leq K$ **do**
  $SP^k \leftarrow P(\bigcup_{p=1}^{k} \mathcal{N}^p, \bigcup_{p=1}^{q} \mathcal{S}^p, \overline{\mathcal{B}^1})$
  Solve SP^k
  if SP^k is infeasible **then**
    $q \leftarrow q+1$
  else
    $(\hat{x},\hat{y}) \leftarrow Sol(SP^k)$
    Update $\overline{\mathcal{B}^1} = \{x^s_{ij} : i,j \in \bigcup_{p=1}^{k} \mathcal{N}^p,\ s \in \bigcup_{p=1}^{q} \mathcal{S}^p,\ \hat{x}^s_{ij} = 1\}$
    $k \leftarrow k+1$
    $q \leftarrow q+1$
  end if
end while
return $(\hat{x},\hat{y})$.

The effectiveness of the heuristic described above strongly depends on how astutely the partitions are chosen. Let us first consider the set of shifts, $\mathcal{S}$. Because the number of shifts is not fixed but

the main goal is rather to complete all tasks with the fewest possible number of shifts, including two consecutive shifts in each subset of $\mathcal{S}$ is an appealing strategy, since it will allow us to introduce more shifts only when needed. Further, it will lead to small sub-problems, reducing the computational burden. So, in our implementation, $\mathcal{S}^1$ is composed of shifts 1 and 2, $\mathcal{S}^2$ is composed of shifts 3 and 4, and so on.

Let us now consider the set of tasks, $\mathcal{N}$. We break this set down by dividing tasks into two geographically proximate groupings. We continue dividing each set of tasks in two until each group of tasks can be completed within a shift period, and not two, in order to accommodate deadheads and traverses between tasks and from and to the depot. To do so, we recursively call an algorithm akin to the linear-time mincut algorithm proposed in Fiduccia and Mattheyses (1982) for the graph partitioning problem.

The Fiduccia-Mattheyses (FM) graph partitioning algorithm is based on the popular Kernighan-Lin algorithm (Kernighan and Lin, 1970). It can roughly be summarized as an iterative procedure that, starting from an initial partition $(\mathcal{N}^1, \mathcal{N}^2)$, progressively improves that partition by moving one vertex from its current group to the other. The objective is to find a partition that minimizes the cost of the cut, defined as $C(\mathcal{N}^1, \mathcal{N}^2) = \sum_{(i,j)\in(\mathcal{N}^1,\mathcal{N}^2)} c(i,j)$, where $c(i,j)$ represents the cost associated with edge (i,j). The partition must also satisfy a balance criterion that requires the sizes of $\mathcal{N}^1$ and $\mathcal{N}^2$ to be within a certain factor of each other. Because our purpose is to divide the tasks into geographically proximate groupings, we define $c(i,j) = M - d_{ij}$, where M is a large value and d_{ij} is the length of the shortest path leading from i to j. Also, because the partition we wish to find needs to be balanced in terms of workload, instead of using the cardinality of $\mathcal{N}^1$ and $\mathcal{N}^2$ as a balance criterion, we use the sum of service time. More precisely, any subset $T \subseteq \mathcal{N}$ is evaluated using the function $W(T) = \sum_{i\in T} \sigma_i$. Dividing tasks terminates when, for each subset $\mathcal{N}^k$, $W(\mathcal{N}^k) \leq \overline{U}_{min}$, where $\overline{U}_{min}$ is the length of the shortest shift ($\overline{U}_{min} = \min_{s\in\mathcal{S}} \overline{U}_s$). This means that we impose that the total duration of service in each group of tasks is no larger than the duration of any shift period.

As stated above, a partition of the tasks into K groups such that each group of tasks can be completed within a single shift period, and not two (although each subset of shifts consists of two shifts), allows us to accommodate deadheads and traverses between tasks and from and to the depots. It also increases the likelihood that other tasks can be inserted in subsequent iterations, thereby alleviating the negative consequence of decomposition (missing good solutions).

3.3 Shift-only decomposition

In this section, we present a third decomposition strategy based on the *fix-and-relax heuristic*, also known as *the sliding time window heuristic* (Dillenberger et al., 1994; Escudero and Salmeron, 2005; Brown et al., 2001; Pochet and Wolsey, 2006; Lamghari and Dimitrakopoulos, 2016). Unlike the strategies presented in Sections 3.1 and 3.2, which partition both the set of tasks and the set of shifts, here, only the set of shifts is partitioned. We do this because partitioning tasks might not achieve the best performance in terms of solution quality, for decisions made in the early stages of the algorithm limit the available possibilities in later stages. Considering all tasks allows us to adopt a less myopic and more flexible strategy. While this may result in longer computational times, it has the potential to yield better solutions.

Each iteration of the heuristic consists of two main steps. In the first step, we solve a sub-problem to obtain routes for a small number of shifts, ν. Variables x^s_{ij} associated with shifts $s > \nu$ are not projected out, as in Section 3.2. They are retained, but with their integrality relaxed. This guarantees that a feasible solution can be obtained and that the sub-problem to be solved is of tractable size. In the second step of the algorithm, the routes obtained are fixed and the next ν shifts are considered. The process continues until all tasks are assigned to appropriate shifts. A pseudo-code of the procedure is given in Algorithm 3. The set of shifts $\mathcal{S}$ is partitioned into three disjoint subsets $\mathcal{S}^1$, $\mathcal{S}^2$, and $\mathcal{S}^3$ (Lines 2–4). The sub-problem obtained from the original problem after fixing variables x^s_{ij} associated with $s \in \mathcal{S}^1$ and relaxing the integrality of those associated with $s \in \mathcal{S}^3$ is denoted $\overline{P}(\mathcal{S}^1, \mathcal{S}^2, \mathcal{S}^3)$. Once a sub-problem is solved (Line 8), the partition is updated (Lines 10–13).

Algorithm 3 Shift-only decomposition

Input: ν, the number of shifts to consider at each iteration.
Output: a feasible solution $(\hat{x}, \hat{y})$ to (P).
Set $(\hat{x}, \hat{y}) \leftarrow \emptyset$
$\mathcal{S}^1 \leftarrow \emptyset$
$\mathcal{S}^2 \leftarrow \{1, \ldots, \nu\}$
$\mathcal{S}^3 \leftarrow \mathcal{S} \setminus \{\mathcal{S}^1 \cup \mathcal{S}^2\}$
$k \leftarrow 1$
repeat
  $SP^k \leftarrow \overline{P}(\mathcal{S}^1, \mathcal{S}^2, \mathcal{S}^3)$
  Solve SP^k
  $(\hat{x}, \hat{y}) \leftarrow Sol(SP^k)$
  $\mathcal{S}^1 \leftarrow \mathcal{S}^1 \cup \mathcal{S}^2$
  $k \leftarrow k + 1$
  $\mathcal{S}^2 \leftarrow \{(k-1)\nu + 1, \ldots, k\nu\}$
  $\mathcal{S}^3 \leftarrow \mathcal{S} \setminus \{\mathcal{S}^1 \cup \mathcal{S}^2\}$
until all tasks are assigned
return $(\hat{x}, \hat{y})$.

3.4 Relationship of the proposed decomposition strategies to other known methodologies

We now describe similarities and differences among the three heuristics described in Sections 3.1–3.3 and other well-known methodologies.

The basic idea of the three heuristics is to reduce the complexity of the problem by decomposing it into smaller and easier-to-solve sub-problems, a common strategy that allows the search to focus on a small region of the search space. The three heuristics alternate between fixing a part of the solution to its current value and optimizing the remaining part, and as such, they are closely related to *projection strategies* (Geoffrion, 1970a,b) and to *the cyclic coordinate descent methods* (Luenberger, 1984) used in *the Gauss-Seidel* and *the Jacobi* procedures to optimize a function of several variables.

We also note that there is a close relationship between the heuristic described in Section 3.2 and the large neighborhood search (LNS) heuristic proposed by Shaw (Shaw, 1998).[1] Breaking links from and to depots is essentially the destroy step in LNS, while solving $SP^k := P(\bigcup_{p=1}^{k} \mathcal{N}^p, \bigcup_{p=1}^{q} \mathcal{S}^p, \overline{\mathcal{B}^1})$ corresponds to the repair step. The degree of destruction gradually increases (as the search progresses, more shifts are considered and therefore larger parts of the solution are destroyed. The sub-problem solved adjusts routes associated with those shifts, in the best possible way). That said, our heuristic does not purely follow the concepts of LNS. While LNS includes an acceptance criterion, we omit this criterion because, in our case, the repair step necessarily returns an improved solution, with less violations of constraints (2) and (3). Another main difference lies in the variables considered in the repair step. LNS attempts to improve the current solution by changing only the values of the variables selected at the destroy step, whereas the proposed heuristic considers additional variables and gradually extends the search to previously unexplored regions of the search space.

Because the three heuristics combine various algorithmic ideas and involve mathematical programming models, they can be seen as matheuristics (Gunther et al., 2019).

4 Computational results

To assess the tractability of model (1)–(14) and compare the three decomposition strategies described in Section 3, we performed computational experiments on several classes of both random and real-world

[1]Note that the proposed heuristic is basically a construction heuristic, while LNS is an improvement algorithm. However, constructing a solution can be viewed as improving a solution where violations of constraints (2) and (3) are allowed.

test instances. These instances are described in Section 4.1, followed by an outline of implementation details in Section 4.2 and a summary of computational results in Sections 4.3–4.6.

4.1 Generation of instances

Since there has been little study of the SSRP in the literature, no benchmark instances exist. We therefore randomly generated a set of 48 instances that strike a balance between realism and ease of generation. These instances vary according to size and complexity. The size of an instance is defined by the number of tasks ($|\mathcal{N}| = |\mathcal{H}| + |\mathcal{O}|$), while the percentage of tasks associated with highways ($\%H = \frac{|\mathcal{H}|}{|\mathcal{N}|} \times 100$) and the number of depots ($|\mathcal{D}|$) determine the complexity of an instance.

We use the instance *DI-NEARP-n422-Q2k-TP.dat* of Vidal (2017) as a basis to generate our 48 instances. *DI-NEARP-n422-Q2k-TP* contains 710 nodes and one depot, whose Euclidean coordinates are specified. We joined series of North-South and East-West nodes to make two highways. The other nodes in each of the four quadrants established after making the highways were linked to constitute other roads. The service time and travel time of each road segment (whether highway or non-highway) were calculated first by computing the Euclidean distance between the extremities that define the road segment. Then, the service time and travel time were obtained by multiplying this distance by the sweeping speed and the travel speed, respectively, and rounding the result to the nearest integer. Both speeds were provided by our industry partner, *Arseno Balayage*, a company that specializes in spring sweeping.

We generated instances with sizes ranging between 40 and 180 tasks. In doing so, we made sure that not all road segments (tasks) in any instance are connected, giving us instances that are close to reality. These instances are divided into two classes: those with a small number of highway segments and those with a relatively large number of highways. In the former group, tasks associated with highway segments make up an average of 25% of all tasks, and in the latter group, they consist of 30% of all tasks on average. According to discussions with our partner, we can consider these two percentages realistic.

For both classes, we considered two scenarios for each instance: in the first scenario, there is a single depot, which corresponds to the depot in *DI-NEARP-n422-Q2k-TP*, while in the second there are multiple depots (2 or 3 additional depots, depending on the size of the instance). These additional depots are randomly chosen, but such that each of them is close to a cluster of road segments to be swept, as is the case in reality. Finally, in all instances, the duration of each shift ($\overline{U}^s$) was set to 10 hours. The fixed cost was set to \$4,500, while the variable costs were set to 1/3 and 5/3 dollar per minute for deadheading and sweeping, respectively. These two values represent an estimate of the fuel cost per km for the sweeping fleet and are based on the simulation study of Ben Daya et al. (2024). Recall that the fleet is composed of two sweepers, two dump trucks, and one pickup truck used as a safety vehicle.

In addition to the instances described above, we also consider an instance based on real data from Victoriaville, a municipality in the Centre-du-Québec region of Québec, Canada. The contract area, shown in Figure 2, consists of 20.5 km of highways and 144.5 km of other roads, amounting to 46 tasks of type $\mathcal{H}$ and 128 tasks of type $\mathcal{O}$, represented by circles in the figure. There are 4 depots, represented by black squares in the figure. The data collected from *Google Maps* and from our partner, who has been awarded this contract many times in the past, were used to compute the service and travel times. For the shift duration, fixed cost, and variable costs, we used values similar to those in the random test instances.

Table 1 summarizes the characteristics of the 48 randomly generated instances and those of the real test instance.

Table 1: Characteristics of the instances.

Dataset	Instance	# Vertices $\lvert\mathcal{H}\rvert$	$\lvert\mathcal{O}\rvert$	$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{D}\rvert$
	A1	10	30	40	2
	A2	12	38	50	2
A	A3	15	45	60	2
Number of instances: 13	A4	18	52	70	2
Percentage of tasks associated with highways: $\%H = 25\%$	A5	22	68	90	3
Number of depots: Multiple	A6	25	75	100	3
	A7	28	82	110	3
	A8	30	90	120	3
	A9	33	97	130	4
	A10	36	104	140	4
	A11	39	111	150	4
	A12	42	118	160	4
	A13*	46	128	174	4
	B1	15	35	50	2
	B2	18	42	60	2
B	B3	21	49	70	2
Number of instances: 12	B4	24	56	80	2
Percentage of tasks associated with highways: $\%H = 30\%$	B5	28	67	95	3
Number of depots: Multiple	B6	32	73	105	3
	B7	34	81	115	3
	B8	38	87	125	3
	B9	39	91	130	4
	B10	42	103	145	4
	B11	48	112	160	4
	B12	54	126	180	4
	C1	10	30	40	1
	C2	12	38	50	1
C	C3	15	45	60	1
Number of instances: 12	C4	18	52	70	1
Percentage of tasks associated with highways: $\%H = 25\%$	C5	22	68	90	1
Number of depots: Single	C6	25	75	100	1
	C7	28	82	110	1
	C8	30	90	120	1
	C9	33	97	130	1
	C10	36	104	140	1
	C11	39	111	150	1
	C12	42	118	160	1
	D1	15	35	50	1
	D2	18	42	60	1
D	D3	21	49	70	1
Number of instances: 12	D4	24	56	80	1
Percentage of tasks associated with highways: $\%H = 30\%$	D5	28	67	95	1
Number of depots: Single	D6	32	73	105	1
	D7	34	81	115	1
	D8	38	87	125	1
	D9	39	91	130	1
	D10	42	103	145	1
	D11	48	112	160	1
	D12	54	126	180	1

(*): Real instance corresponding to Victoriaville, QC, Canada.

4.2 Implementation details and performance measures

The algorithms were coded in Python, and the experiments were carried out on an Intel Core i7-8700 with a 3.2 GHz CPU and 64 GB of RAM running under Linux. We used CPLEX 22.1.1 to solve the sub-problems and to refine the initial solution at the second stage of the solution procedure (c.f. Algorithm 1). The time limit, `Tmax`, was set to 3600 seconds (one hour) for instances with up to 80

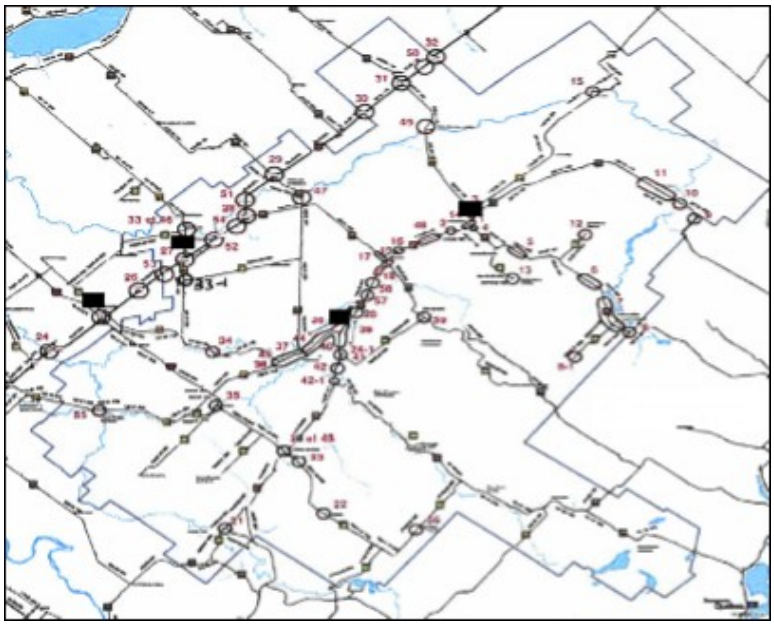

Figure 2: Sweeping contract area - Victoriaville, Québec, Canada.

tasks ($|\mathcal{N}| \leq 80$), to 7200 seconds (two hours) for instances with 90 to 125 tasks, and to 10,800 seconds (three hours) for instances with more than 125 tasks ($|\mathcal{N}| \geq 130$). When solving sub-problems (Phase I), CPLEX was executed with an optimality tolerance (`Tol%`) of 5% and a time limit of 800 seconds. If no solution is reached after 800 seconds, the time limit is increased by 200 seconds. This is repeated as long as the maximum CPU time, `Tmax`, is not reached. For the *Shift-only decomposition*, an extra parameter is required: ν, the number of shifts to consider at each iteration of the first stage of the solution procedure. To calibrate this parameter, some preliminary experiments were completed with the largest instances with more than 130 tasks. We considered four values for ν: 1, 2, 3, and 4. The best results were obtained with $\nu = 2$. Hence, we completed the rest of the tests using this value.

In the next sections, we compare the following methods:

- No decomposition (ND in the following) where we solve the monolithic formulation (1)–(14) using CPLEX 22.1.1 with its default settings and time limits of 3600 seconds, 7200 seconds and 10,800 seconds depending on the size of the instance, as explained above.
- Decomposition methods: IBD, PBD and SOD, corresponding to the solution procedure presented in Algorithm 1 with the *Infrastructure-based decomposition*, the *Proximity-based decomposition* and the *Shift-only decomposition* described in Sections 3.1, 3.2 and 3.3, respectively.

To assess the performance of the methods, we use the following measures:

- The initial optimality gap, in percentage. Since the optimal solutions are not known, we overestimate this measure by the ratio (Laporte and Toth, 2022): *Initial gap* $= \frac{Z_X^{init} - LB^*}{LB^*} \times 100$, where Z_X^{init} is the objective function value of the warm start (initial) solution produced by method X (IBD, PBD or SOD) and LB^* is the best known lower bound. Note that this measure does not apply to ND.
- The CPU time, in seconds, required to complete phase I. This measure also applies only to the three decomposition methods, IBD, PBD, and SOD.
- The final gap, in percentage, measured as *Final gap* $= \frac{Z_X^* - LB^*}{LB^*} \times 100$, where LB^* is as defined above and Z_X^* is the value of the best feasible solution found by method X, either ND, IBD, PBD or SOD.

- In addition to the final gap, we also report *Difference* $= \frac{Z^*_{PBD} - Z^*_X}{Z^*_{PBD}} \times 100$, where X can be either ND, IBD or SOD. This measure allows us to compare these three methods to PBD, which was found to give the best results in preliminary tests.

The four measures are computed for each instance. Because we want to compare the behavior of the methods with respect to problem size, and we also want to see which of them work best on each class of instances, we chose to provide detailed results in Sections 4.3–4.5. These detailed results are summarized in Section 4.6, where arithmetic averages over each set of instances are computed.

4.3 Results on the first set of benchmark instances (A instances)

Our first set of benchmark instances includes 13 instances with multiple depots ($|\mathcal{D}| = 2$, 3 or 4). The number of tasks $|\mathcal{N}|$ ranges between 40 and 174, of which highway segments constitute approximately 25%. Table 2 presents the results obtained by the four methods for these instances. For IBD, PBD and SOD, *Initial gap* and *Time* are shown, while *Final gap* and *Difference* are given, respectively, for all four methods and for ND, IBD and SOD. A dash ("-") indicates that no feasible solution was obtained within the time limit. In addition, the table also shows, in column LB^*, the value of the best known lower bound used to compute the gaps.

Table 2: Performance of the four methods for the A instances.

	$\lvert\mathcal{N}\rvert, \lvert\mathcal{D}\rvert$	LB^*	*Initial gap (%)*			*Time (sec.)*			*Final gap (%)*				*Difference (%)*		
			IBD	PBD	SOD	IBD	PBD	SOD	ND	IBD	PBD	SOD	ND	IBD	SOD
A1	40, 2	9,192.00	1.49	1.22	10.04	3	1	5	0.65	0.65	0.67	0.11	0.02	0.02	0.56
A2	50, 2	9,267.00	2.60	2.35	5.13	4	2	45	1.36	1.36	1.36	0.49	0.00	0.00	0.86
A3	60, 2	9,318.85	5.08	3.63	4.55	9	2	70	1.62	1.60	1.64	0.65	0.02	0.04	0.97
A4	70, 2	13,987.00	3.31	1.87	3.28	54	4	540	0.97	1.29	1.06	2.19	0.08	-0.23	-1.12
A5	90, 3	14,072.55	-	3.65	35.75	-	8	1239	-	-	0.77	34.67	-	-	-33.64
A6	100, 3	14,144.75	-	2.58	38.44	-	9	1211	-	-	1.54	34.45	-	-	-32.41
A7	110, 3	18,753.63	-	2.83	51.86	-	19	2475	-	-	2.11	50.53	-	-	-47.42
A8	120, 3	18,744.00	-	2.89	50.82	-	23	2566	-	-	1.90	50.15	-	-	-47.35
A9	130, 4	18,667.84	-	2.98	52.78	-	122	2495	-	-	1.54	52.29	-	-	-49.98
A10	140, 4	18,747.42	-	2.98	51.88	-	124	2510	-	-	1.30	51.34	-	-	-49.41
A11	150, 4	18,817.67	-	2.99	27.09	-	131	2467	-	-	1.71	26.68	-	-	-24.54
A12	160, 4	18,816.79	-	3.18	50.82	-	260	2439	-	-	1.97	50.22	-	-	-47.31
A13	174, 4	23,798.20	-	2.59	68.09	-	180	2475	-	-	1.53	67.52	-	-	-65.00

Time limit = 3600 seconds for instances A1-A4, 7200 seconds for A5-A8, and 10800 seconds for A9-A13. A13 is the real instance corresponding to Victoriaville, QC, Canada.

As one might expect, large instances of practical interest cannot be solved in a reasonable amount of time without decomposing the problem. Thus ND cannot solve instances with more than 70 tasks within the time limit. Of the three decomposition methods, PBD would rank first, SOD second, and IBD last. The performance of IBD is comparable to that of ND. On the other hand, SOD and PBD are able to solve all 13 instances. The performance of PBD is superior, however, as it is consistent regardless of the number of tasks, with an average optimality gap of 1.47% (column *Final gap*). The performance of SOD significantly deteriorates as the number of tasks increases, with an average optimality gap of 32.41%. The advantage of PBD over IBD and SOD is that the sub-problems are considerably smaller, as explained in Section 3, which results in a significant reduction of the time needed to complete the first phase and generate the warm start (initial) solution, as can be seen by examining the values of *Time*. Consequently, more time remains for the second phase to improve this solution. Additionally, PBD generates better initial solutions than SOD and IBD (column *Initial gap*). This could in part be explained by the fact that PBD assigns clusters of geographically close tasks to the same shift. Turning to the impact of the size of the problem on the time required to complete the first phase, Table 2 shows that the rate of increase is higher for SOD than it is for PBD. This is not surprising, given that the size of the sub-problems that PBD solves is quite similar irrespective of the number of

tasks, while it is proportional to the number of tasks in SOD. For the largest random instance A12, PBD takes less than 5 minutes, while SOD takes almost 10 times longer.

When we consider that our goal is to find a way for our industry partner to reduce costs, it is noteworthy that the solution provided by PBD for the real instance A13 consists of 5 shifts, as opposed to 15.5 shifts in the solution generated manually by our industry partner. Given the fixed costs, the economic gains realized by implementing a PBD solution in this case can be near $50,000, a significant amount for a small company. Moreover, the company would be able to bid on more contracts, as the time saved would make their equipment available for more work. Recall that the sweeping season is very short, so time saved is an extremely important consideration. PBD thus provides an efficient vehicle routing plan that minimizes completion time and costs and is of benefit to our industry partner.

4.4 Results on the second set of benchmark instances (B instances)

The second set of benchmark instances is used to analyze the performance of the four methods when the percentage of highways is increased. It contains 12 instances of various sizes with multiple depots and with highway segments constituting around 30% of all tasks, as opposed to 25% in the set A considered in the previous section. While the difference between 25% and 30% seems relatively small, values beyond 30% would not be realistic, according to our industry partner.

Table 3: Performance of the four methods for the B instances.

	$\lvert\mathcal{N}\rvert, \lvert\mathcal{D}\rvert$	LB^*	*Initial gap (%)*			*Time (sec.)*			*Final gap (%)*				*Difference (%)*		
			IBD	PBD	SOD	IBD	PBD	SOD	ND	IBD	PBD	SOD	ND	IBD	SOD
B1	50, 2	9,258.42	4.39	2.68	3.02	6	2	51	1.53	1.53	1.59	0.65	0.06	0.06	0.93
B2	60, 2	9,417.17	2.53	2.76	99.54	675	3	1349	2.12	1.26	1.29	0.05	-0.82	0.03	1.22
B3	70, 2	14,015.10	2.92	3.02	35.24	12	6	1324	34.91	1.70	1.34	34.27	-33.12	-0.36	-32.49
B4	80, 2	13,995.04	4.27	2.07	34.48	16	8	1220	-	2.98	1.16	32.98	-	-1.80	-31.46
B5	95, 3	18,605.51	-	2.68	2.98	-	13	1207	-	-	1.68	1.21	-	-	0.46
B6	105, 3	18,779.99	-	3.52	51.35	-	14	1454	-	-	1.59	49.72	-	-	-47.37
B7	115, 3	18,768.34	-	3.00	52.56	-	18	1562	-	-	1.85	51.08	-	-	-48.34
B8	125, 3	18,745.14	-	2.97	51.88	-	22	2464	-	-	1.76	50.75	-	-	-48.14
B9	130, 4	18,713.76	-	2.67	3.24	-	44	1429	-	-	1.81	1.85	-	-	-0.03
B10	145, 4	18,772.76	-	3.60	52.47	-	154	2528	-	-	2.51	51.14	-	-	-47.44
B11	160, 4	18,786.92	-	4.11	51.93	-	160	2501	-	-	2.17	50.28	-	-	-47.09
B12	180, 4	23,447.58	-	3.10	24.86	-	175	2750	-	-	1.99	23.97	-	-	-21.54

Time limit = 3600 seconds for instances B1-B4, 7200 seconds for B5-B8, and 10800 seconds for B9-B12.

The numerical results are summarized in Table 3. We see that IBD performs very well on small instances. In all but one case (B4), a feasible solution within less than 2% of optimality is obtained. However, IBD fails to solve larger instances in the maximum time allowed. As explained above, this is a direct result of the large size of the sub-problems to be solved in the first phase of the solution procedure. Overall, when there are more highway segments, IBD outperforms ND, which is contrary to the instances when the number of highways was smaller. The average gaps for IBD and ND are 1.50% and 12.85%, respectively, as opposed to 1.23% and 1.15% (considering only instances where both methods are able to produce a feasible solution within the time limit).

The other two methods, PBD and SOD, were able to solve all instances for this benchmark set before the time limit was reached (similar to the set A), confirming their ability to tackle large instances within a reasonable amount of time. The two methods perform the same for one instance (B9), SOD performs slightly better for 3 instances (the two smallest instances and B5), and PBD performs better for the remaining 8 instances. Despite the apparent similarities between the two methods, the range of optimality gaps obtained demonstrate that PBD is much more robust than SOD, with optimality gaps ranging between 1.16% and 2.51% when PBD is used, and optimality gaps between 0.05% and 51.14% when SOD is employed. In fact, the optimality gaps for instances other than B1, B2, B5, and B9 are very high for SOD, ranging from 23.97% to 51.14%. These large values are due to the fixed

costs associated with each shift. For example, PBD and SOD report a final gap of 2.51% and 51.14%, respectively, for B10. This is not surprising, considering that in the solution found by PBD, 4 shifts are used, whereas in that provided by SOD, 6 shifts are necessary to sweep all segments.

The results above seem to indicate that increasing the percentage of highways makes the problem less complicated for SOD, which performs better on average for set B than for set A (average final gap of 29% versus 32.41%). On the other hand, PBD's performance slightly degrades (1.73% versus 1.47%). With respect to the time required to complete the first phase, there is no notable difference between set A and set B for both methods.

4.5 The impact of the number of depots (C and D instances)

We now analyze how the results are affected when only one depot is considered. Instances in sets C and D, considered in this section, were derived from those in sets A and B by considering the single depot in *DI-NEARP-n422-Q2k-TP* (Vidal, 2017). This single depot serves all the area under consideration, unlike cases with multiple depots, where each depot was close to a cluster of road segments to be swept.

Table 4 summarizes the results of the experiments. Two key observations can be made from this table. First, the performance of all methods deteriorates when one depot is considered compared to the case with multiple depots. This can be partially explained by the fact that the best lower bound values are quite similar to those obtained in the case of multiple depots (see column LB^*). The upper

Table 4: Performance of the four methods for the C and D instances.

	$\|\mathcal{N}\|, \|\mathcal{D}\|$	LB^*	*Initial gap (%)*			*Time (sec.)*			*Final gap (%)*				*Difference (%)*		
			IBD	PBD	SOD	IBD	PBD	SOD	ND	IBD	PBD	SOD	ND	IBD	SOD
C1	40, 1	9,198.48	2.76	3.77	8.88	3	1	1	0.88	0.88	0.88	0.30	0.00	0.00	0.57
C2	50, 1	9,256.50	3.43	4.15	5.13	4	2	5	2.08	2.08	2.07	1.23	-0.01	-0.01	0.82
C3	60, 1	9,315.25	5.73	3.97	4.91	8	2	179	-	2.28	2.54	1.37	-	0.25	1.14
C4	70, 1	13,991.43	-	2.82	34.32	-	4	607	-	-	1.17	0.73	-	-	0.43
C5	90, 1	13,955.22	-	3.65	40.21	-	47	1231	-	-	2.27	36.44	-	-	-33.41
C6	100, 1	14,008.25	-	2.47	40.94	-	30	1235	-	-	2.01	36.44	-	-	-33.75
C7	110, 1	18,595.77	-	3.31	30.31	-	51	2440	-	-	3.02	29.63	-	-	-25.83
C8	120, 1	18,757.33	-	3.48	51.71	-	1226	3450	-	-	2.81	51.22	-	-	-47.09
C9	130, 1	18,671.23	-	4.11	3.98	-	254	1658	-	-	3.22	3.33	-	-	-0.11
C10	140, 1	18,746.01	-	3.75	28.16	-	370	2444	-	-	3.05	27.43	-	-	-23.66
C11	150, 1	18,813.60	-	3.62	53.36	-	176	2502	-	-	2.91	52.89	-	-	-48.52
C12	160, 1	18,814.33	-	3.38	52.57	-	829	2550	-	-	3.19	52.22	-	-	-47.52
D1	50, 1	9,255.00	4.77	4.50	3.12	19	2	660	2.56	2.68	2.76	1.69	0.19	0.07	1.04
D2	60, 1	9,321.86	53.20	4.15	102.79	1225	15	1290	101.67	3.42	3.35	99.86	-95.13	-0.07	-93.39
D3	70, 1	13,889.32	5.27	5.06	37.08	16	6	2400	-	3.82	3.24	36.28	-	-0.56	-32.00
D4	80, 1	13,998.90	4.77	3.07	38.87	1374	7	2200	-	2.96	2.67	34.02	-	-0.29	-30.54
D5	95, 1	18,603.99	-	4.27	59.33	-	84	1212	-	-	2.95	2.30	-	-	0.63
D6	105, 1	18,780.76	-	3.86	58.65	-	1426	1223	-	-	3.27	52.47	-	-	-47.64
D7	115, 1	18,766.68	-	3.63	62.20	-	212	1479	-	-	3.50	51.86	-	-	-46.72
D8	125, 1	18,743.58	-	3.89	54.94	-	1450	2464	-	-	3.52	53.28	-	-	-48.07
D9	130, 1	18,714.33	-	3.44	28.88	-	54	2441	-	-	2.97	28.12	-	-	-24.42
D10	145, 1	18,771.75	-	4.19	109.04	-	934	3510	-	-	2.78	107.08	-	-	-101.47
D11	160, 1	18,798.92	-	3.98	28.17	-	158	2449	-	-	3.01	27.61	-	-	-23.88
D12	180, 1	23,452.96	-	3.61	25.34	-	371	2519	-	-	3.11	24.28	-	-	-20.54

Time limit = 3600 seconds for instances C1-C4 and D1-D4, 7200 seconds for C5-C8 and D5-D8, and 10800 seconds for C9-C12 and D9-D12.

bound values, however, become larger because distances to the depot are longer when one depot is considered, which results in more deadhead time and thus higher costs. This does not favorably affect the gaps, and it is not surprising to see their values increase. We also note that, irrespective of the

method, the time required to complete the first phase is higher than when there is more than one depot.

Second, the results of the tests clearly exhibit the expected pattern: small instances with up to 50 tasks are solved to near optimality by all four methods and thus do not yield insights into the performance of the decomposition approaches. We see, though, that as the size of the problem increases, decomposition has a positive effect on the performance as long as the sub-problems to be solved are not large and the warm start is a good quality solution, as in PBD. ND performs very poorly, as it finds a feasible solution for only 4 instances out of 24. Again, IBD is effective only for small instances with up to 80 tasks, while SOD was able to solve all instances but with an average gap of 33.84%. PBD solved all instances in sets C and D within the time limit, with an average gap of 2.76%, which reinforces its superiority over the other three methods.

4.6 Summary of results

Table 5 shows comparative statistics for the four methods by set of instances. Under *Solved*, we report the percentage of instances that each method was able to solve within the time limit. The second group of columns, *Best solution*, gives the percentage of instances for which each method can find the best solution. The columns under the heading *Average gap* provide the obtained average final gaps over i) the instances that ND was able to solve (Restricted) and ii) over all instances (Overall).

Table 5: Summary of the results.

Set	*Solved (%)*				*Best solution (%)*				*Average gap (%)*					
									Restricted				Overall	
	ND	IBD	SOD	PBD	ND	IBD	SOD	PBD	ND	IBD	SOD	PBD	SOD	PBD
A	30.77	30.77	100.00	100.00	7.69	0.00	23.08	69.23	1.15	1.23	0.86	1.18	32.41	1.47
B	25.00	33.33	100.00	100.00	0.00	0.00	25.00	75.00	12.85	1.50	11.66	1.41	29.00	1.73
C	16.67	25.00	100.00	100.00	0.00	0.00	33.33	66.67	1.48	1.48	0.77	1.47	24.44	2.43
D	16.67	33.33	100.00	100.00	0.00	0.00	16.67	83.33	52.11	3.05	50.78	3.05	43.24	3.09

A: multiple depots, $\%H = 25\%$; B: multiple depots, $\%H = 30\%$; C: single depot, $\%H = 25\%$; D: single depot, $\%H = 30\%$.

The numerical results indicate that among the four methods considered in this paper, ND performs the worst as it is able to solve only 11 small instances out of all 49 instances tested (22.45%) within the time limit. Considering these 11 instances, the average gap of the solutions provided by ND is 16.90% as opposed to 1.81% for IBD, 16.02% for SOD, and 1.78% for PBD. Clearly, the results indicate that it pays off to decompose the problem, as the three decomposition methods find better solutions, except for one instance in set A.

IBD was able to solve the 15 smallest instances out of 49 (30.61%). For these instances, it finds good quality solutions, comparable to or better than those produced by ND. The comparison between IBD and SOD shows that SOD is able to obtain better solutions for instances with a small percentage of highways. However, for the instances with a larger percentage of highways, IBD typically gives better solutions than SOD.

SOD and PBD are distinguished by their ability to solve all 49 instances. The comparison between SOD and PBD shows that the performance of SOD is slightly better than that of PBD when the instances are of small size, but for larger instances, PBD is significantly better except for a handful of instances (4 out of 37). The results also indicate that the percentage of highways and the number of depots has almost no impact on the performance of PBD. PBD always provides high quality solutions with a final gap generally below 3%, independent of the particularities of a given instance or its size.

The look-ahead features that SOD includes (considering all tasks and not only a subset of tasks at each iteration of the first phase as PBD does) are beneficial and apparent on small instances. However,

as the problem size increases, the advantages of the look-ahead features are offset by the large size of the sub-problems. Although somewhat smaller than the original problem, these sub-problems remain quite large, making the first phase time consuming. PBD performs better with larger problems most likely due to the small size of the sub-problems and the geographical proximate groupings strategy that allow a good feasible warm start (initial) solution to be generated quickly. This strategy leads to a more effective solution procedure, as supported by the excellent results obtained for all tested instances, including the real test instance, A13.

It is worth noting that the quality of the initial solution provided by PBD is very good, with an average gap of 3.31%, which is affected neither by the number of depots nor by the percentage of highways. The experiments also showed that very little time, up to a few minutes, is actually spent generating this solution. Therefore, the *proximity-based decomposition* (phase I of PBD) is a useful decision-support tool that can be employed by sweeping companies in contexts where solutions must be obtained quickly and/or when they want to test various scenarios of travel and service times in order to construct and evaluate routing plans before bidding on a contract. Phase II can be incorporated in contexts where a precise measure of deviation from optimality is required.

5 Conclusions

In this paper, we have considered the practical problem of generating efficient routing plans that minimize costs for spring sweeping of road abrasives (the SSRP). We have proposed a node formulation that is more compact than the arc formulation previously proposed in the literature, and we have developed three decomposition procedures to solve the problem. The performance of these procedures was evaluated through computational testing on a set containing 49 random and real-life instances. The three procedures were compared to each other and to a general-purpose solver (CPLEX), which was used to solve the SSRP directly as an MILP model.

The computational study indicates that CPLEX can only solve instances with a small number of tasks. Of our three proposed decomposition strategies, the *proximity-based decomposition* strategy has proven to be the most efficient for addressing large real-world instances. Such instances cannot be solved directly as MILP models and are also difficult for the *infrastructure-based* and *shift-only decomposition* approaches.

Future directions for research include the incorporation into the model of some additional constraints encountered in practice, such as *sweeping hierarchy* constraints that impose that high-priority roadways must be cleaned first. Another possible avenue for research is to extend the model to incorporate uncertainty in travel and service times, which vary depending on the state of the roads, the quantity of abrasives spread in winter, and the weather conditions when sweeping occurs. Last but not least, we have observed that the lower bound progresses very slowly. Thus, developing strategies to obtain better lower bounds is a promising research area worth pursueing.

References

Baldacci, R., Maniezzo, V., 2006. Exact methods based on node routing formulations for undirected arc routing problems. Networks 47, 52–60.

Ben Daya, B., Audy, J.F., Lamghari, A., 2024. Real-world data simulation comparing GHG emissions and operational performance of two sweeping systems. Logistics 8, 120.

Bertsimas, D., Dunn, J., 2019. Machine learning under a modern optimization lens. Dynamic Ideas Press.

Bertsimas, D., Weismantel, R., 2005. Optimization over integers. Dynamic Ideas Press.

Blazquez, C.A., Beghelli, A., Meneses, V.P., 2012. A novel methodology for determining low-cost fine particulate matter sweeping routes. Journal of the Air & Waste Management Association 62, 242–251.

Bodin, L., Kursh, S., 1978. A computer-assisted system for the routing and scheduling of street sweepers. Operations Research 26, 525–537.

Bodin, L., Kursh, S., 1979. A detailed description of a computer system for the routing and scheduling of street sweepers. Computers & Operations Research 6, 181–198.

Brown, G., Keegan, J., Vigus, B., Wood, K., 2001. The kellogg company optimizes production, inventory, and distribution. Interfaces 31, 1–15.

Cerrone, C., Dussault, B., Golden, B., 2014. Multi-period street scheduling and sweeping. International Journal of Metaheuristics 3, 21–58.

Dillenberger, C., Escudero, L., Wollensak, A., Zhang, W., 1994. On practical ressource allocation for production planning and scheduling with period overlapping setups. European Journal of Operational Research 75, 275–286.

Eglese, R., Murdock, H., 1991. Routeing road sweepers in a rural area. The Journal of the Operational Research Society 42, 281–288.

Eiselt, H.A., Gendreau, M., Laporte, G., 1995. Arc routing problems, part ii: the rural postman problem. Operations Research 43, 399–414.

Escudero, L., Salmeron, J., 2005. On a fix-and-relax framework for a class of project scheduling problems. Annals of Operations Research 140, 163–188.

Fiduccia, C.M., Mattheyses, R.M., 1982. A linear-time heuristic for improving network partitions, in: Proceedings of the 19th Design Automation Conference, pp. 175–181.

Geoffrion, A., 1970a. Elements of large-scale mathematical programming. part i: Concepts. Management Science 16, 652–675.

Geoffrion, A., 1970b. Elements of large-scale mathematical programming. part ii: Synthesis of algorithms and bibliography. Management Science 16, 676–691.

Gunther, R., Puchinger, J., Blum, C., 2019. Metaheuristics hybrids, in: Handbook of Metaheuristics, Third Edition, pp. 385–417.

Kernighan, B.W., Lin, S., 1970. An efficient heuristic procedure for partitioning graphs. Bell System Technical Journal 49, 291–307.

Kraiem, A., Audy, J.F., Lamghari, A., 2025. Mixed integer linear programming model for a multi-depot arc routing problem with different arc types and flexible assignment of end depot. Transportation Research Procedia 82, 1109–1119.

Lamghari, A., Dimitrakopoulos, R., 2016. Progressive hedging applied as a metaheuristic to schedule production in open-pit mines accounting for reserve uncertainty. European Journal of Operational Research 253, 843–855.

Laporte, G., Toth, P., 2022. A gap in scientific reporting. 4OR - A Quarterly Journal of Operations Research 20, 169–171.

Longo, H., de Aragão, M., Uchoa, E., 2006. Solving capacitated arc routing problems using a transformation to the cvrp. Computers & Operations Research 33, 1823–1837.

Luenberger, D.G., 1984. Linear and Nonlinear Programming, Second Edition. Addison-Wesley.

Perrier, N., Langevin, A., Campbell, J.F., 2007a. A survey of models and algorithms for winter road maintenance. part iii: Vehicle routing and depot location for spreading. Computers & Operations Research 34, 211–257.

Perrier, N., Langevin, A., Campbell, J.F., 2007b. A survey of models and algorithms for winter road maintenance. part iv: Vehicle routing and fleet sizing for plowing and snow disposal. Computers & Operations Research 34, 258–294.

Pochet, Y., Wolsey, L., 2006. Production planning by mixed integer programming. Springer, New York, NY.

Shaw, P., 1998. Using constraint programming and local search methods to solve vehicle routing problems, in: Lecture Notes in Computer Science, pp. 417–431.

Tagmouti, M., Gendreau, M., Potvin, J.Y., 2007. Arc routing problems with time-dependent service costs. European Journal of Operational Research 181, 30–39.

Vidal, T., 2017. Node, edge, arc routing and turn penalties: multiple problems-one neighborhood extension. Operations Research 65, 992–1010.

Yu, B., Xin, J., D'Ariano, A., 2019. Route planning of a sanitation vehicle in a dynamic traffic environment, in: 2019 Chinese Automation Congress (CAC), pp. 2898–2903.

Yurtseven, C., Gokce, M.A., 2019. A novel arc-routing problem of electric powered street sweepers with time windows and intermediate stops. IFAC-PapersOnLine 52, 2308–2313.