Conflict-free electric vehicle routing problem: an improved compositional algorithm

Citation for the original published paper (version of record):
Roselli, S., Fabian, M., Åkesson, K. (2024). Conflict-free electric vehicle routing problem: an improved compositional algorithm. Discrete Event Dynamic Systems: Theory and Applications, 34(1): 21-51. http://dx.doi.org/10.1007/s10626-023-00388-6

(article starts on next page)

Discrete Event Dynamic Systems
https://doi.org/10.1007/s10626-023-00388-6

Conflict-free electric vehicle routing problem: an improved compositional algorithm

Sabino Francesco Roselli[1] · **Martin Fabian**[1] · **Knut Åkesson**[1]

Received: 2 March 2023 / Accepted: 24 November 2023

Abstract
The Conflict-Free Electric Vehicle Routing Problem (CF-EVRP) is a combinatorial optimization problem of designing routes for vehicles to execute tasks such that a cost function, typically the number of vehicles or the total travelled distance, is minimized. The CF-EVRP involves constraints such as time windows on the tasks' execution, limited operating range of the vehicles, and limited capacity on the number of vehicles that a road segment can simultaneously accommodate. In previous work, the compositional algorithm *ComSat* was introduced to solve the CF-EVRP by breaking it down into sub-problems and iteratively solve them to build an overall solution. Though ComSat showed good performance in general, some problem instances took significant time to solve due to the high number of iterations required to find solutions for two sub-problems, namely the *Routing Problem*, and the *Paths Changing Problem*. This paper addresses the bottlenecks of ComSat and presents new formulations for both sub-problems in order to reduce the number of iterations required to find feasible solutions to the CF-EVRP. Experiments on sets of benchmark instances show the effectiveness of the presented improvements.

Keywords CF-EVRP · Optimization · Scheduling · Autonomous vehicles

1 Introduction

Scheduling of Autonomous Mobile Robots (AMRs) for just-in-time delivery to the main assembly line poses a significant problem. The challenges associated with this issue include the size of the fleet to schedule (De Ryck et al. 2020), as well as ensuring the robots' ability

This article belongs to the Topical Collection: *Special Issue on Applications 2022*
Guest Editors: Jan Komenda, Tomas Masopust, and Spyros Reveliotis

✉ Sabino Francesco Roselli
rsabino@chalmers.se

Martin Fabian
fabian@chalmers.se

Knut Åkesson
knut@chalmers.se

1 Department of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden

Published online: 18 December 2023

to complete their pick-up/delivery tasks within specified time windows (Yao et al. 2020), as well as handling scalability concerns. Moreover, depending on the robots' size and the plant layout's geometrical features, areas may exist within the plant where simultaneous transit of two or more robots leads to conflicts (Pratissoli et al. 2023), i.e., instances where the robots obstruct each other. Lastly, indoor-operating robots are typically powered by batteries, which possess limited operating ranges and require non-negligible charging times (Abderrahim et al. 2020).

A suitable scheduler for such a problem should simultaneously satisfy the following requirements: (i) assign one or more tasks to each robot, ensuring each task is executed within its designated time window; (ii) calculate a path for each robot to carry out its assigned tasks; (iii) schedule the robots along their respective paths to prevent conflicts; and (iv) guarantee that the robots never run out of charge.

Moreover, an industrial plant is a dynamic environment where frequent changes require re-scheduling. Therefore it is necessary that a new schedule can be computed quickly (from seconds to minutes, depending on the specific application). Consequenty, the scheduler must be sufficiently fast to schedule a possibly large fleet in the limited available time.

Let us focus on the requirements from the previous paragraphs one at the time. (i) Assigning all tasks to robots equates to design *Routes* for the robots. Typically, the robots are stationed at one or multiple depots and tasks are assigned to them in a specified order. Therefore the robots leave their depot, travel from one task to the next in their ordered assignment, and return to the depot, thus defining a closed route. (ii) There may be several paths in a plant to travel from one point to another. The choice of one path over another may have consequences in terms of travelled length or possibility of conflicting with other robots. (iii) the road segments, intersections, depots, and workstations are characterized by a *capacity* based on the number of robots that can be simultaneously in these areas. Therefore *capacity constraints* are defined in order to produce conflict-free schedules. (iv) a robot may not have enough charge to serve all customers it is assigned to. Hence, after executing some tasks, it may need to return to its depot and recharge the battery, before it can continue executing the remaining tasks.

In Roselli et al. (2021) we formalized the problem described above and named it the *Conflict-Free Electric Vehicle Routing Problem* (CF-EVRP); we listed all its requirements in detail and formulated a mathematical model for it. Also, we used the Satisfiability Modulo Theory (SMT) (Barrett et al. 2009; De Moura and Bjørner 2011) solver Z3 (De Moura and Bjørner 2008) to solve generated problem instances of the CF-EVRP and found that it was not suitable to schedule large fleets, as it took several minutes to solve instances involving only few vehicles. Therefore, in Roselli et al. (2022) we presented a compositional algorithm, ComSat, for solving the CF-EVRP. ComSat breaks down the CF-EVRP into sub-problems and iteratively solves them to find a feasible solution to the overall problem. Experimental and analytical evaluation shows that ComSat generates high-quality but not necessarily optimal solutions.

Briefly, ComSat (see Fig. 1) solves the *Routing Problem*, i.e., computes routes to execute all the tasks, by calling the *Router* algorithm, and assigns vehicles to the routes (the *Assignment Problem*) by calling the *Assign* algorithm; this part of ComSat is henceforth called *Feasible Routes Search* (FRS). Then, ComSat solves the *Capacity Verification Problem*, i.e., schedules the vehicles along their paths (initially Dijkstra's algorithm (Dijkstra 1959) is used to compute such paths) by calling the *CapacityVerifier* algorithm (*CV*), so that no conflicts arise among them. However, it may be that a feasible schedule cannot be found because, using the current paths, it is impossible to avoid conflicts and meet the time windows; then alternative paths have to be computed. This is handled in the *Conflict-Free Paths Search* (CFPS) where the *Paths-Changer* algorithm (*PC*) solves the *Paths Changing Problem*, i.e., finds alternative

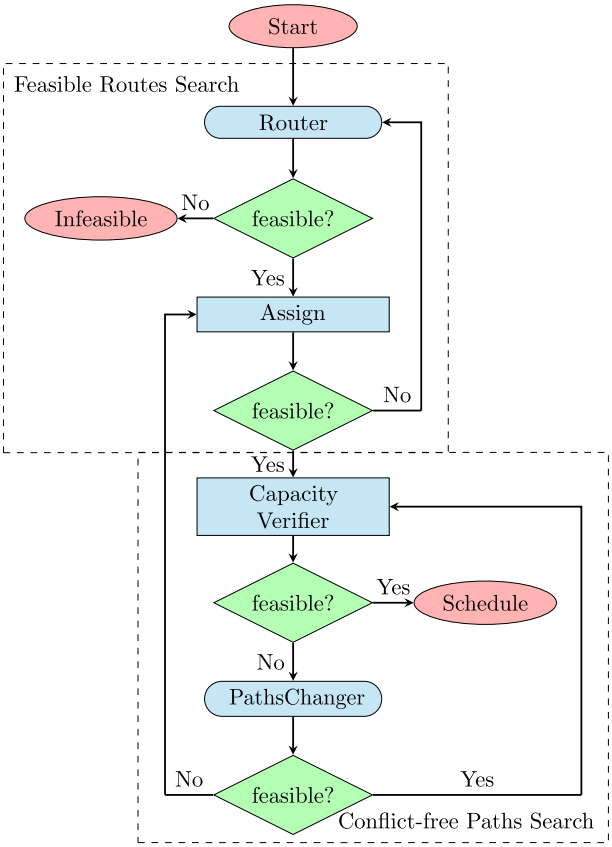


Fig. 1 Simplified flowchart of ComSat

sets of paths if the current schedule has conflicts, and the *CapacityVerifier* finds, if possible, a conflict-free schedule.

Experiments show two bottlenecks in the algorithm, where a large number of iterations may be required in order to find feasible solutions to the CF-EVRP;

- In the FRS, the number of possible routes grows exponentially with the number of tasks, therefore even small problem instances can lead to a large number of possible solutions to the *Routing Problem*. Many of these sets of routes may not satisfy the *Assignment Problem* though, leading to a large number of unnecessary iterations between these two sub-problems.
- When a schedule has conflicts, the *Paths-Changer* algorithm will find an alternative set of paths. Since the *Paths-Changer* has no information about why the previous paths led to conflicts among the vehicles, the new paths may lead to the same conflict, or at least some of them. It may therefore take several iterations to find conflict-free paths.

In this paper we focus on improving the performance of ComSat by ruling out infeasible solutions before they are computed, hence keeping the number of iterations between sub-problems low. In our previous work (Roselli et al. 2022), we tackled the bottleneck in the CFPS by formulating improved versions of the *PC* and *CV*. In this work, we generalize such improvements so that they are applicable to any instance of the CF-EVRP. Moreover,

we present a *tighter* formulation of the *Routing Problem* that prevents the generation of solutions that will be infeasible when solving the *Assignment Problem*. We also present a further improvement where the *Routing* and *Assignment* problems are combined into the *E-Routing Problem*. This approach allows to avoid iterating between the two sub-problems, and experiments show that the computation time required to solve the new sub-problem is not larger than the time required to solve the *Routing Problem* only.

The sub-problems in ComSat are modelled as Mixed Integer Linear Programming (MILP) (Kondili et al. 1993), and SMT problems. For the CFPS, polynomial time algorithms exist to find paths in graphs (Arumugam et al. 2016). However, modelling the *Paths Changing Problem* as an SMT problem is beneficial as it allows us to define problem-specific requirements, such as not returning solutions that are already proven infeasible because they violate the capacity constraints. Moreover, when a problem is infeasible, SMT solvers have the ability to return a *Minimal Unsatisfiable Core* (*MUC*) (Cimatti et al. 2011), i.e., one of the (possibly many) smallest subsets of constraints that make the problem infeasible. The *MUC* can provide useful information about why a problem is infeasible and can therefore be used to guide the search towards a feasible solution (Selsam and Bjørner 2019). Therefore, when dealing with the CF-EVRP, the *MUC* can be extracted when the *Capacity Verification Problem* is infeasible, and used to define additional constraints for the *Paths Changing Problem* to increase the chances of finding a feasible schedule.

The contributions in this paper are: (i) improvements on the FRS; (ii) exploitation of *MUC* to extract information about the infeasibility of an SMT formula representing a conflicting schedule for a VRP and use of such information to find conflict-free schedules; (iii) performance comparison between the different versions of ComSat discussed in the paper over a set of the CF-EVRP problem instances.

The remainder of the paper is organized as follows. Section 2 presents an overview on the topic and introduces existing publications this work is built on. Section 3 provides preliminaries, together with a full description of ComSat. Sections 4 and 5 present the mathematical models of the sub-problems that form the FRS and CFPS, respectively, how they are improved to reduce the number of iterations and the computation time. Proof of soundness and completeness of the procedures is provided in Section 6. In Section 7, the results of the analysis over a set of problem instances are presented. Finally, conclusions are drawn in Section 8.

2 Literature review

The CF-EVRP is an extension of the well-known Vehicle Routing Problem (VRP) (Dantzig and Ramser 1959), a combinatorial optimization problem that involves designing routes for vehicles to visit specific locations, with the objective of minimizing the total length of these routes. Several extensions of the original VRP exist (Kucukoglu et al. 2021) that tackle different aspects of the problem and originate from different real world applications. For instance, the VRP with Time Windows (Desrochers et al. 1992) extends the original problem to include, for each task, an earliest and latest arrival time i.e., a time window in which the task has to be executed. Another extension is the Multi Depot VRP (Lim and Wang 2005) where there can be several depots the vehicles are dispatched from.

For battery powered vehicles, there has been a growing attention to formulating VRPs that account for limited operating range and non negligible charging time. In Schneider et al. (2014) the Electric VRP (E-VRP) is first introduced. In this work, vehicles can return to the depot to fully recharge their batteries (partial charging is tackled in subsequent works Keskin

and Çatay 2016; Cortés-Murcia et al. 2019) after which they may be dispatched again to complete more tasks. The authors present a hybrid heuristic that combines neighbourhood (Hansen and Mladenovi 2005) and tabu (Glover 1989) search to solve problem instances involving up to 500 tasks.

Thus far, the VRPs discussed are characterized by the ability of travelling directly from any task's location to any other, and the movement of one vehicle is unaffected by the presence of other vehicles. These VRPs are suitable for applications such as routing of cars or trucks on public roads, where in most cases vehicles can avoid obstructing each other's path simply by adhering to the traffic rules. Furthermore, vehicles are still widely operated by humans, whose decision-making ability to resolve conflicts can help in those cases where traffic rules alone are insufficient.

On the other hand, a plant layout can be a challenging environment to drive in, due to less space available to maneuver. Also, AMRs typically do not have the same decision-making skills as human drivers. Therefore the problem of computing conflict-free schedules for fleets of AMRs is addressed by several works. There have been attempts to find exact solutions to conflict-free routing problems by means of general purpose solvers. For instance, in Murakami (2020), a MILP formulation to design conflict-free routes for capacitated vehicles is presented. An SMT formulation is introduced in Roselli et al. (2018). The size of such formulation in terms of the number of variables and constraints grows exponentially with the number of vehicles, tasks, and the size of the plant; only very small instances can be solved to optimality within reasonable time.

Alternatively, approximate methods can be used to solve larger problem instances. One early such work (Krishnamurthy et al. 1993) uses column generation to compute conflict-free routes for vehicles on a bidirectional network. An ant colony algorithm is applied to the problem of job shop scheduling and conflict free routing of vehicles by Saidi-Mehrabad et al. (2015); in Yuan et al. (2016), paths are computed for one vehicle at a time using a modified A^* (Hart et al. 1968) to find paths that overlap as little as possible with the paths already computed for other vehicles. This, in combination with collision resolution rules, allow to relieve traffic congestion. A heuristic approach to solve the conflict-free routing problem with storage allocation is presented by Thanos et al. (2019). In Zhong et al. (2020) is presented a hybrid evolutionary algorithm to deal with conflict-free vehicles scheduling in automated container terminals; finally, in Daugherty et al. (2018) the problem of conflict-free routing of vehicles is handled by a heuristic algorithm based on local search.

A typical way to trade optimality for computational speed, is to decompose the problem into sub-problems. For example, in Corréa et al. (2007), the problem is divided into first assigning the tasks to the vehicles, and then route the vehicles through the plant. ComSat (Roselli et al. 2022) is another example of a compositional algorithm to solve this problem. A main difference with Corréa et al. (2007) is that ComSat accounts for the recharge of the vehicles.

In conclusion, the CF-EVRP involves a number of challenges deriving from the different features that characterize it, such as vehicles' limited operating range and non-negligible charging time, time windows for the execution of the tasks, and above all, limited capacity of the road segments of the plant layout. As the CF-EVRP is an extension of the VRP, several studies exist on the topic, and each of the additional features (battery charge, capacity constraints on the routes, time windows, etc.) have been tackled previously. Whatever the objective function (total travelled distance, number of robots, etc.) finding optimal solutions fast is only tractable for small problem instances. Hence several approximate methods have been used to solve larger problem instances fast. In our previous work (Roselli et al. 2018) we claimed the novelty of the CF-EVRP as, to the best of our knowledge and at the time of

publication, no other formulation included all the requirements the CF-EVRP did. We hence developed the compositional algorithm ComSat to quickly solve larger problem instances, and in this work, we present an improved version of it.

3 Preliminaries

In the CF-EVRP the plant layout is represented by a finite, strongly connected, weighted, directed graph, where edges represent road segments and nodes represent either tasks' locations or intersections between road segments. A task is defined by a unique (numerical) identifier, a location, and a time window, i.e., a lower and upper bound that represent the earliest and latest arrival time allowed to execute the task. Edges have two attributes, the first representing the road segment's length, and the second its capacity. The capacity is 2 if two vehicles can simultaneously travel in opposite directions, 1 otherwise.

The following logical operators are used as a shorthand to express cardinality constraints (Sinz 2005) in the SMT sub-problems:

$\bigoplus_{i \in \mathcal{I}}(x_i, n)$: exactly n variables x_i are true $\forall i \in \mathcal{I}$;
$\text{If}(c, o_1, o_2)$: evaluates to o_1 if c is *true*, else o_2.

Moreover, in the MILP sub-problems, the *big-M* method (Trespalacios and Grossmann 2015) is used to model alternatives, hence the need for a big enough integer M.

The following definitions are provided:

- *Node*: a location in the plant. A node can only accommodate one vehicle at a time unless it is a *hub* node that can accommodate an arbitrary number of vehicles.

  $\mathcal{N}$: a finite set of nodes.
  $\mathcal{N}_H \subseteq \mathcal{N}$: the set of hub nodes.

- Depots: nodes at which one or more vehicles start and must return to after completing the assigned tasks. A depot can accommodate an arbitrary number of vehicles at the same time, thus all depots are hubs.

  $\emptyset \subset \mathcal{D} \subseteq \mathcal{N}_H$: the set of depots.
  $\mathcal{D}_\mathcal{S} = \{ s_k \mid k \in \mathcal{D}\}$: the set of dummy tasks representing the start from depot k.
  $\mathcal{D}_\mathcal{F} = \{ f_k \mid k \in \mathcal{D}\}$: the set of dummy tasks representing the arrival at depot k.

  The sets $\mathcal{D}_\mathcal{S}$ and $\mathcal{D}_\mathcal{F}$ are disjoint with each other and with the set of tasks $\mathcal{K}$ (see below).

- *Edge*: a road segment that connects two nodes.

  $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$: the finite set of direct edges.
  $\bar{e}$: the reverse edge of edge $e \in \mathcal{E}$.
  $|e| \in \mathbb{R}^+$: the length of edge $e \in \mathcal{E}$.
  $g_e \in \{1, 2\}$: the capacity of edge $e \in \mathcal{E}$.

- *Time horizon*: a fixed, continuous point in time when all tasks have ended.

  T: the time horizon.

- *Task*: Entity representing a place to be visited exactly once by a vehicle in order to pickup or delivery material. A task is always associated with a node where the pickup/delivery takes place, and has a time window indicating the earliest and latest time at which the node can be visited. Unless explicitly given, the time window is the entire time span $[0, T]$. Moreover, each task has a precedence list that indicates other tasks that must be

executed before the task itself. This so, since the tasks represent pickups and deliveries, hence a vehicle has to deliver the goods after it has picked them up.

$\mathcal{K}$: the finite set of all tasks. Let $\mathcal{K}^+ = \mathcal{K} \cup \mathcal{D}_\mathcal{S} \cup \mathcal{D}_\mathcal{F}$.
$l_k, u_k \in \mathbb{R}^+,\ k \in \mathcal{K}$: the time window's lower (l_k) and upper (u_k) bound for task k such that $u_k > l_k$.
$\tau_k \in \mathbb{R}^+$: the service time of task k.
$L_k \in \mathcal{N}$, for $k \in \mathcal{K}$: location of task k.
$\mathcal{Q}_k \subset \mathcal{K},\ k \in \mathcal{K}$: the set of tasks to execute before task k.

In order to execute a task, the vehicle assigned to it will (unless the task's locations coincides with the vehicle's initial location) visit nodes and edges in the graph other than the node where the task is located.

- Vehicle: an AGV, that is able to move between locations in the plant and perform pickup and delivery operations. Battery-powered vehicles have a limited operating range, travel at constant speed or are stationary, and their energy consumption is linearly proportional to the distance travelled; they have the ability to recharge at the depot they are dispatched from, but not at other depots[1]. Charging, when executed, restores the battery to its full charge capacity; therefore, partial charging is not permitted. This restriction, inspired by Schneider et al. (2014), is necessary to formulate the routing problem of Section 4.2. Finally, we assume that a vehicle's *remaining charge* is directly proportional to the vehicle's *remaining operating range* and, in order to simplify the notation, we only talk about the latter.

  $\mathcal{V}$: the finite set of all vehicles.
  $\mathcal{V}_k \subseteq \mathcal{V},\ \forall k \in \mathcal{K}$: set of vehicles eligible for task k.
  $\psi \in \mathbb{R}^+$: the vehicles' maximum operating range when their battery is fully charged (distance).
  $C \in \mathbb{R}^+$: the charging coefficient (increase of remaining operating range per time unit).
  $\Omega \in \mathbb{R}^+$: vehicle speed (constant) while moving.

- *Route*: a sequence, starting and ending at the same depot, with unique tasks in-between that may have the same depot embedded:

  $R_j = \langle d, k_1, k_2, \ldots, d \ldots, k_{n-1}, k_n, d\rangle$ where $d \in \mathcal{D}, k_i \in \mathcal{K}$
  $k_i \neq k_j$ for $i \neq j$, and $n \leq |\mathcal{K}|$, since a route can at most include all tasks.

- *Route set*: a set of routes such that each task belongs to exactly one route, thus guaranteeing that all tasks are served.

  $\mathcal{R} = \{R_1, \ldots, R_m\},\ m \leq |\mathcal{K}|$

  A route contains at least one task, hence $0 < m \leq |\mathcal{K}|$.
- *Pair Set of route R*: set containing the sequence of tasks of a route $R = \langle k_1, \ldots, k_m\rangle$, grouped as pairs in sequence.

  $\mathcal{P}_R = \{\langle k_1, k_2\rangle, \langle k_2, k_3\rangle, \ldots, \langle k_{m-1}, k_m\rangle\}$

- *Path*: ordered set of unique nodes. It is used to keep track of how vehicles are travelling among tasks of routes, since each pair of tasks in a route is connected by a path.

[1] This restriction was necessary to formulate the previous versions of ComSat, and to make a fair comparison it applies also to the latest version presented in this work. It may, however, be relaxed in future work.

$$\theta_p = \langle n_1, \ldots, n_m \rangle, \; p \in \mathcal{P}_R, \; m \leq |\mathcal{N}|,$$
$$n_i \in \mathcal{N}, \; i = 1, \ldots, m$$

- *Edge sequence*: ordered set of unique edges for a given path θ_p.

$$\delta_p = \langle e_1, \ldots, e_m \rangle, \; p \in \mathcal{P}_R, \; m = |\theta_p| - 1,$$
$$e_i \in \mathcal{E}, \; i = 1, \ldots, m$$

3.1 Recap of ComSat

In order to understand the changes to ComSat introduced in this work, let us recap how it works, and what sub-algorithms it is composed of. The initial step involves computing a set of the (possibly many) shortest paths between each pair of tasks, hereafter referred to as the *current paths CP*. The first time, the computation of the *CP* is performed using Dijkstra's algorithm, hence *CP* will be the shortest paths between the tasks, and are part of the input for the *Router* algorithm. When the *Router* is called, neither the vehicles' availability nor the segment capacities are considered; the goal is simply to design routes to execute tasks within the time windows. At this point, the need for specific paths *CP* is that, through the constant speed Ω, it is possible to infer the arrival time at the tasks' locations in order to guarantee that time windows are respected. If the *Routing* problem is infeasible, the whole problem is infeasible, since the tasks cannot be executed within n their time windows.

If the *Routing Problem* is feasible, *Router* returns a non-empty set of routes $\mathcal{R}$ and for each route $R \in \mathcal{R}$ the latest time Υ_R at which R can start, while still meeting all the time windows of its tasks. $\mathcal{R}$ is stored in the set *PR* so that each time *Router* is called, *PR* can be used to rule it out. In the next step, *Assign* will determine which vehicle is assigned to each route based on the routes' requirements for specific types of vehicles, their latest start time, and the vehicles' operating range and charging rate; also for the *Assignment Problem* there can be more feasible solutions, therefore it is important to store the current assignment $\mathcal{A}$ in the set *PA* to be able to rule it out the next time *Assign* is called. If the *Assignment Problem* is infeasible the algorithm backtracks and solves the *Routing Problem* again, generating new routes that might be assignable; otherwise, ComSat proceeds to the *Capacity Verification Problem*.

At this point, routes have been assigned vehicles to execute them, and start times $\beta_R, \; \forall R \in \mathcal{R}$ have been computed to meet the vehicles' need for charging. Hence it is possible to verify if the execution of the routes is possible without violating the capacity constraints. If that is the case, the overall problem is feasible and ComSat terminates and returns a feasible schedule. On the other hand, if this step is infeasible, ComSat will try to find alternative paths (currently the shortest paths are used as *CP*) for the vehicles to execute the routes by solving the *Paths Changing Problem*.

If the *Paths Changing Problem* is feasible, the new paths found become the current paths *CP* and ComSat backtracks to verify whether a conflict-free schedule exists by solving the *Capacity Verification Problem* again; if not, the *PC* algorithm is called again. Otherwise, if the *Paths Changing Problem* is infeasible, the algorithm backtracks and looks for a new assignment.

Whenever the *Assignment Problem* is infeasible, all possible assignments for the current set of routes $\mathcal{R}$ have been explored. In the same way, whenever the *Paths Changing Problem* is infeasible, all possible paths for the current assignment $\mathcal{A}$ have been explored. Therefore, the set of previous paths *PP* is emptied because these paths are only eligible for the current assignment, and the shortest paths are set as current paths to compute the next assignment.

Note that the complete version of ComSat also involves an additional sub-problem, namely the *Routes Verification Problem*, where it is verified whether the previously computed routes are still feasible when using the newly computed paths. However, discussing this sub-problem is out of the scope of this paper since the sub-problem is very quick to solve and does not constitute a bottleneck; instead, the reader is referred to Roselli et al. (2022) for further details.

3.2 *The minimal unsat core*

For infeasible problems, a subset of the constraints that conflict, i.e., they cannot all simultaneously be satisfied, can be identified. Such a subset is called an *Unsat Core*. An *Unsat Core* with the property that removing any one of the constraints makes the *Unsat Core* feasible, is said to be *minimal*.

Formally, given an SMT formula φ and a set of conflicting constraints $\mathcal{C} \subseteq \varphi$, $\mathcal{C}$ is a *Minimal Unsat Core* (*MUC*) of φ if removing any constraint $\mathcal{C}_i \in \mathcal{C}$ makes $\mathcal{C} \setminus \mathcal{C}_i$ no longer infeasible; removing $\mathcal{C}_i$ removes the particular conflict represented by the *MUC*. Consequently, for an infeasible problem with a *MUC* $\mathcal{C}$, adding to the problem a constraint that prevents all the constraints in $\mathcal{C}$ to be simultaneously active will resolve this particular conflict.

The naïve approach to *MUC* extraction, Dershowitz et al. (2006), successively removes constraints and solves the problem again; if the problem is still infeasible after a constraint has been removed that constraint does not belong to a *MUC*. There exist more efficient approaches; the *MUC* (Huang 2005) algorithm based on efficient manipulation of *Binary Decision Trees* guarantees the extraction of a *minimal Unsat Core*. Nadel (2010) presents an algorithm based on the resolution graph (Kroening and Strichman 2016) for *MUC* extraction. Nadel et al. (2013) improves the resolution based algorithm using *model rotation* and *path strengthening*.

In Section 5 we discuss how to use the *MUC* to extract information about conflicts on when the *Capacity Verification Problem* is infeasible, in order to guide the search for alternative, feasible paths by defining additional constraints for the *Paths Changing Problem*.

4 The feasible routes search

As mentioned in Section 1, experiments showed that for some problem instances *Router* can generate a large number of solutions to the routing problem $\mathcal{R}$ that lead to the *Assignment Problem* being infeasible. In this section we present first a tighter formulation to break the symmetry of different solutions, and then an approach that involves merging the *Routing* and *Assignment* problems, to deal with this bottleneck.

4.1 A tighter formulation to break the symmetry among solutions

The *Routing problem* as formulated in Roselli et al. (2022) does not permit charging during the routes. Therefore, in this section, routes are defined as sequences that starts and end at the same depot, with unique tasks in between and with no depot embedded.

The problem is formulated using 0–1 variables $\sigma_{rk_1k_2}$ ($k_1, k_2 \in \mathcal{K}^+$, $r = 1, \ldots, |\mathcal{K}|$) that represent that a route R includes direct travel from task k_1 to k_2. Let $\varsigma = \{\sigma_{rk_1k_2} \mid k_1, k_2 \in \mathcal{K}^+,\ r = 1, \ldots, |\mathcal{K}|\}$ be the set of all variables $\sigma_{rk_1k_2}$, then the optimal solution to the *Routing Problem* found at iteration h is $\mathcal{R}^* = \{\sigma_{rk_1k_2} \in \varsigma \mid \sigma_{rk_1k_2} = 1\}$; also, let PR^* be the set

containing the optimal solutions $\mathcal{R}^*$ found until the $(h-1)$-th iteration. Finally, note that it is possible to compute $\mathcal{R}$ from $\mathcal{R}^*$ and vice versa. Subsequently, it is possible to compute PR from PR^* and vice versa.

Let us clarify this concept with an example. Assume that we have solved a *Routing Problem* involving four tasks such that $\mathcal{K} = \{k_1, k_2, k_3, k_4\}$ and one depot d such that $\mathcal{D}_\mathcal{S} = \{s_d\}$ and $\mathcal{D}_\mathcal{F} = \{f_d\}$. One extreme case is that all tasks are executed in one route, while another extreme case is that the solution involves four routes, each including only one task. These solutions, and all the others in between, might or might not be feasible, depending on the task locations, their time windows, and other parameters (see Roselli et al. (2022) for further details). Let us assume that a possible solution in terms of the variables $\sigma_{rk_1k_2}$ would be the following:

$$\mathcal{R}^* = \{\sigma_{1s_dk_1},\ \sigma_{1k_1k_2},\ \sigma_{1k_2f_d},\ \sigma_{2s_dk_3},\ \sigma_{2k_3k_4},\ \sigma_{2k_4f_d}\}. \tag{1}$$

From $\mathcal{R}^*$ we can compute $\mathcal{R} = \{\langle s_d, k_1, k_2, f_d\rangle,\ \langle s_d, k_3, k_4, f_d\rangle\}$. Note that the first index of each σ variable represents the *route number*, and is used to keep track of the route the tasks are assigned to in order to compute $\mathcal{R}$ from $\mathcal{R}^*$.

In general, a feasible *Routing Problem* results in a number of the σ variables being set to 1, thus defining a set of routes $\mathcal{R}$. If the *Assignment Problem* fails for this set of routes, the *Router* is called again, with the additional requirement that at least one of the σ variables that were previously set to 1, should now be set to 0. In Roselli et al. (2022) we enforced this requirement using the following constraint:

$$\sum_{\sigma_{rk_1k_2}\in\mathcal{R}^*} \sigma_{rk_1k_2} \leq |\mathcal{R}^*| - 1,\ \forall \mathcal{R}^* \in PR^*. \tag{2}$$

Assuming Eq. 1 is a solution to the *Routing problem*, in the next call of *Router*, constraint Eq. 2 expands to:

$$\sigma_{1s_dk_1} + \sigma_{1k_1k_2} + \sigma_{1k_2f_d} + \sigma_{2s_dk_3} + \sigma_{2k_3k_4} + \sigma_{2k_4f_d} \leq 5.$$

This call of *Router* may then return the solution:

$$\mathcal{R}^* = \{\sigma_{1s_dk_1}, \sigma_{1k_1k_2}, \sigma_{1k_2f_d}, \sigma_{3s_dk_3}, \sigma_{3k_3k_4}, \sigma_{3k_4f_d}\} \tag{3}$$

Solutions Eqs. 1 and 3 are identical except for the route index. Therefore, if one is infeasible, the other will be as well. In order to avoid generating such identical solutions, a constraint is defined that forbids a particular sequence of tasks visited for every route index. For the example above, such a constraint would be $\forall r = 1, \ldots, 4$:

$$(\sigma_{rs_dk_1} + \sigma_{rk_1k_2} + \sigma_{rk_3f_d} \leq 3) \vee (\sigma_{rs_dk_3} + \sigma_{rk_3k_4} + \sigma_{rk_4f_d} \leq 3)$$

In general, a *Routing Problem* solution may involve up to $|\mathcal{K}|$ routes, and at the h-th call of *Router*, there are $h-1$ solutions $\mathcal{R}^*$ stored in PR^*. As mentioned before, $\mathcal{R}^*$ is the set containing the σ variables that evaluated to 1 when solving the *Routing Problem* and it is possible to compute $\mathcal{R}$ from $\mathcal{R}^*$. For $h = 1, \ldots, |PR^*|$, let $\mathcal{R}^*_h$ be the h-th solution to the *Routing Problem*, from which it is possible to compute $\mathcal{R}_h$; let $R \in \mathcal{R}_h$ be a route computed in the h-th call of *Router*, and k_n the task coming after task k_{n-1} in R; then, in order to avoid identical solutions, the following additional decision variables are required:

- $\mathcal{X}_{Rh}$ is a $0-1$ variable that is assigned value 1 if route R of the solution h is forbidden, 0 otherwise.
- ι_{Rhr} is a $0-1$ variable that is assigned value 1 if route index r is forbidden for route R of the solution h, 0 otherwise.

Therefore the constraints to rule out previous solutions can be defined as:

$$\mathcal{X}_{Rh} \leq \iota_{Rhr} \qquad \forall r = 1, \ldots, |\mathcal{K}|, h = 1, \ldots, |PR^*|, \ R \in \mathcal{R}_h \tag{4}$$

$$\sum_{k \in R} \sigma_{rk_{n-1}k_n} \leq |R| - 1 + (1 - \iota_{Rhr})\, M \qquad h = 1, \ldots, |PR^*|, \ R \in \mathcal{R}_h, \ r = 1, \ldots, |\mathcal{K}| \tag{5}$$

$$\sum_{R \in \mathcal{R}} \mathcal{X}_{Rh} \geq 1 \qquad \forall h = 1, \ldots, |PR^*| \tag{6}$$

Constraints Eqs. 4 and 5 enforce the definitions of variables $\mathcal{X}$ and ι, respectively; Eq. 6 states that at least one route in each of the previous solutions is forbidden. For the reminder of the paper, we will call the version of ComSat presented in Roselli et al. (2022) $ComSat_1$, and the version using constraints Eqs. 4-6 $ComSat_2$.

4.2 Merging the routing and assignment problems

While the constraints presented in Section 4.1 can help break the symmetry of the solutions, sets of routes might still be infeasible for other reasons, and having to enumerate them, unnecessarily prolongs ComSat's running time. For instance, in a solution $\mathcal{R}$ there might be two routes R_1 and R_2 whose lengths are 20 and 30 distance units respectively, and whose latest start times Υ are 0 and 15, respectively. Also, R_1 and R_2 can only be executed by vehicle v_1, whose speed Ω is one distance unit per time unit. Clearly, R_1 and R_2 overlap in time, since R_1 will start at time 0 and end at least at time 20 (depending on the service times at the tasks included in R_1), while R_2 cannot start later than time 15. Therefore the *Assignment Problem* will be infeasible for $\mathcal{R}$

Moreover, even if they do not overlap in time, they might still be too close in time for one vehicle to execute them. For instance, in the example above, assume that $\Upsilon_{R_2} = 30$ and that the operating range $\psi = 40$. Even though v_1 can finish R_1 before it needs to start R_2, it does not have enough charge to execute both routes and hence needs to recharge in between them. With the charging coefficient $C = 1$ it will not be able to fully charge before time 30, hence the *Assignment Problem* will be infeasible for $\mathcal{R}$.

The reason for infeasibility discussed in the previous paragraph may be generalized to an arbitrary number of routes and vehicles, e.g., three routes need the same type of vehicles but only two such vehicles are available. It is possible to define constraints that rule out the possibility of generating such routes. However, this would imply the enumeration of a large number of possible solutions, since the possible scenarios that lead to infeasibility are numerous. This might as well turn out to be as time-consuming (or more) as iterating between *Router* and *Assign* until a feasible set of routes is found.

When considering possible causes of infeasibility of the *Routing Problem*, it becomes clear that the lack of knowledge about the vehicles' operating range and the non-negligible time to recharge significantly affect the generated solutions. Therefore, it may be beneficial to extend the *Routing Problem* to include the vehicles' limited operating range, and the possibility to recharge batteries at the charging stations. Such a new problem, henceforth called the *E-Routing Problem*, would at once compute the routes to execute the tasks and assign them to the vehicles. The model we present is an extension of the problem formulation for the E-VRP presented in Schneider et al. (2014).

Because actual vehicles are considered in the *E-Routing Problem*, and recharging is allowed, the decision variables used to formulate a model for the *Routing Problem* in Roselli et al. (2022) have to be adjusted or replaced:

- $\sigma_{vk_1k_2}$: 0–1 variable that is 1 if vehicle v is travelling from the location of task k_1 to the location of task k_2, 0 otherwise.
- γ_k: non-negative real variable that represents the time when task k is served.
- ϵ_{vk}: non-negative real variable that represents the remaining operating range of vehicle v when it is at the location of task k.

Moreover, let $\mathcal{D}_\mathcal{C} = \{ c_k \mid k \in \mathcal{D}\}$ be the set of dummy tasks representing the arrival at a charging station; for each vehicle, its charging station is the station it is dispatched from (and has to go back to). For each depot, a trivial upper bound on the number of dummy tasks (each representing the possibility of recharging once) is the number of tasks $|\mathcal{K}|$. Also, let $\mathcal{K}^\star = \mathcal{K}^+ \cup \mathcal{D}_\mathcal{C}$, and let $\overline{\mathcal{V}}_k = \mathcal{V} \setminus \mathcal{V}_k$. With some abuse of notation, we can define the distance between two arbitrary tasks' locations as $|k_1k_2|$ instead of $|L_{k_1}L_{k_2}|$. Let Δ_k be the set of mutually exclusive tasks for task k (i.e. vehicles eligible for task k are not eligible for any of the task in Δ_k due to requirements on the vehicle type); let Λ_k be the set of permutations of tasks that belong to $\mathcal{Q}_k$, thus, each element $\lambda \in \Lambda_k$ is the list of tasks in $\mathcal{Q}_k$ sorted in a unique way. Λ_k is used to model that pickup tasks can be performed in different orders. Note that not all elements λ result in correct orderings of tasks. However, these invalid orderings will be eliminated out by the other constraints in place. Let $\varrho_{\lambda k}$ be a 0–1 variable that is 1 if the permutation λ of $\mathcal{Q}_k$ is selected, 0 otherwise. Finally, let k_i be the task coming after task k_{i-1} in λ.

Then the formulation of the *E-Routing Problem* is as follows:

$$\min \sum_{k_2\in\mathcal{K}} \sum_{k_1\in\mathcal{D}_\mathcal{S}} \sum_{v\in\mathcal{V}} \sigma_{vk_1k_2} + \sum_{k_1\in\mathcal{K}} \sum_{k_2\in\mathcal{D}_\mathcal{C}} \sum_{v\in\mathcal{V}} \sigma_{vk_1k_2} \tag{7}$$

$$\sigma_{vkk} = 0, \qquad \forall k \in \mathcal{K}^\star,\ v \in \mathcal{V} \tag{8}$$

$$\sigma_{vk_1k_2} = 0, \qquad \forall k_1 \in \mathcal{K},\ k_2 \in \mathcal{D}_\mathcal{S},\ v \in \mathcal{V} \tag{9}$$

$$\sigma_{vk_1k_2} = 0, \qquad \forall k_1 \in \mathcal{D}_\mathcal{F},\ k_2 \in \mathcal{K},\ v \in \mathcal{V} \tag{10}$$

$$\sigma_{vk_1k_2} = 0, \qquad \forall k_1, k_2 \in \mathcal{K}^+,\ v \in \overline{\mathcal{V}}_{k_2} \tag{11}$$

$$\epsilon_{vk} \le \psi, \qquad \forall k \in \mathcal{K}^\star, v \in \mathcal{V} \tag{12}$$

$$\gamma_k \ge \gamma_{k'}, \qquad \forall k \in \mathcal{K},\ k' \in \mathcal{Q}_k \tag{13}$$

$$l_k \le \gamma_k \wedge \gamma_k \le u_k, \qquad \forall k \in \mathcal{K}^+ \tag{14}$$

$$\gamma_{k_2} \ge \gamma_{k_1} + \tau_{k_1} + |k_1k_2|/\Omega - M(1 - \sigma_{vk_1k_2}), \qquad \forall k_1, k_2 \in \mathcal{K}^+, v \in \mathcal{V} \tag{15}$$

$$\gamma_{k_2} \ge \gamma_{k_1} + (\psi - \epsilon_{vk})/C + |k_1k_2|/\Omega - M(1 - \sigma_{vk_1k_2}), \qquad \forall k_1 \in \mathcal{D}_\mathcal{C}, k_2 \in \mathcal{K}^+, v \in \mathcal{V} \tag{16}$$

$$\epsilon_{vk_2} \le \epsilon_{vk_1} - |k_1k_2|/\Omega + M(1 - \sigma_{vk_1k_2}), \qquad \forall k_1 \in \mathcal{K}^+, k_2 \in \mathcal{K}^\star v \in \mathcal{V} \tag{17}$$

$$\epsilon_{vk_2} \le \psi - |k_1k_2|/\Omega + M(1 - \sigma_{vk_1k_2}), \qquad \forall k_1 \in \mathcal{D}_\mathcal{C}, k_2 \in \mathcal{K}^\star, v \in \mathcal{V} \tag{18}$$

$$\sum_{v\in\mathcal{V}}\sum_{k_2\in\mathcal{K}}\sigma_{vk_1k_2}=1, \qquad \forall k_1\in\mathcal{K}^\star, k_1\neq k_2 \tag{19}$$

$$\sum_{v\in\mathcal{V}}\sum_{k_1\in\mathcal{K}^\star}\sigma_{vk_1k_2}\leq 1, \qquad \forall k_2\in\mathcal{D}_\mathcal{C}, k_1\neq k_2 \tag{20}$$

$$\sum_{k_2\in\mathcal{K}}\sigma_{vk_1k_2}\leq 1, \qquad \forall k_1\in\mathcal{D}_\mathcal{S},\ v\in\mathcal{V} \tag{21}$$

$$\sum_{k_1\in\mathcal{K}}\sigma_{vkk_1}=\sum_{k_2\in\mathcal{K}}\sigma_{vk_2k}, \qquad \forall k\in\mathcal{K}^\star, v\in\mathcal{V} \tag{22}$$

$$\sum_{k\in\mathcal{K}^\star}\sigma_{vk_1k}=\sum_{k\in\mathcal{K}^\star}\sigma_{vkk_2}, \qquad \forall k_1\in\mathcal{D}_\mathcal{S},\ k_2\in\mathcal{D}_\mathcal{F},\ v\in\mathcal{V} \tag{23}$$

$$\sum_{v\in\mathcal{V}}\sigma_{vk'_{n-1}k'_n}\geq\varrho_{\lambda k}, \qquad \forall k\in\mathcal{K}, \lambda\in\Lambda_k, k'\in\lambda \tag{24}$$

$$\sum_{\lambda\in\Lambda_k}\varrho_{\lambda k}=1, \qquad \forall k\in\mathcal{K} \tag{25}$$

$$\sum_{\sigma_{vk_1k_2}\in\mathcal{R}^*}\sigma_{vk_1k_2}\leq|\mathcal{R}^*|-1, \qquad \mathcal{R}^*\in PR^*\ \mathsf{var} \tag{26}$$

The cost function to minimize Eq. 7 is the total number of routes (this is done by minimizing the number of direct travels from the depots) plus the number of visits to the charging stations; Eq. 8 forbids to travel from and to the same location; Eqs. 9 and 10 express that a vehicle can never travel to the start, nor travel from the end: *start* and *end* referring to the same depot are physically located at the same node, but they play different roles in the *Routing Problem*, hence two different tasks; Eq. 11 expresses that there cannot be direct travel among mutual exclusive tasks. Equation 12 restricts the remaining operating range to be lower than or equal to the maximum operating range; Eq. 13 guarantees that precedence constraints among tasks are enforced. Equation 14 constrains the earliest and latest arrival time to the tasks locations; Eq. 15 model the case when a vehicle is travelling directly between two tasks' locations and constrain the arrival time at the location of the second task based on its distance from the previous task's location, and the speed Ω. Equation 16 models the case when a vehicle is travelling directly between a charging station and a task $k\in\mathcal{K}^+$. This constraint accounts for the charging time spent at the charging station to restore full battery (the term $(\psi-\epsilon_{vk})/C$), and the time required to travel to the next customer's location; Eqs. 17 and 18 model the decrease of remaining operating range based on the distance travelled. Equation 19 expresses that each task's location must be visited exactly once; Eq. 20 constrains the visit to charging stations, i.e. a vehicle can visit the location of each dummy task at most once; Eq. 21 restricts each vehicle to travel to at most one location from a depot. Since vehicles are only allowed to travel to/from their depot, and task locations can only be visited once, this constraint implies that a vehicle can execute at most one route before returning to the depot it was dispatched from; Eq. 22 guarantees the flow conservation between start and end; Eq. 23 ensures that the number of vehicles leaving the depots equals the number of vehicles returning to them. Since each vehicle can only visits its own depot, each vehicle will return to the depot it started from; Eqs. 24 and 25 state that some tasks have to be executed sequentially, when they represent pick-up and delivery of goods; Eq. 26 allows to rule out

the previously computed sets of routes as a solution. This is necessary as this optimization sub-problem may be called multiple times during the execution of ComSat.

Based on the model described above, the algorithm *E-Router* takes the set of current paths *CP*, the set *PR* (from which PR^* is computed), the set of tasks $\mathcal{K}^\star$ and the set of vehicles $\mathcal{V}$ and returns the current assignment $\mathcal{A}$ containing the routes $\mathcal{R}$ (computed from $\mathcal{R}^*$), the vehicle executing each route $R \in \mathcal{R}$, and the starting time β_R time of the route; if and only if the *E-Routing Problem* is infeasible, $\mathcal{A} = \emptyset$.

If the *Routing* and the *Assignment* problems are merged, ComSat has one less step to execute to find a feasible solution to the CF-EVRP. We call this version of the compositional algorithm S-ComSat. Figure 2 shows how the *E-Router* is now the algorithm that takes care of solving the FRS. If it returns a non-empty $\mathcal{A}$, this solution is checked against the capacity constraints by the *CV*. The CFPS takes place as before and, if there is no set of paths that makes the *Capacity Verification Problem* feasible, then the *E-Router* is called again. S-ComSat terminates either when *CV* returns a feasible schedule or when *E-Router* is infeasible.

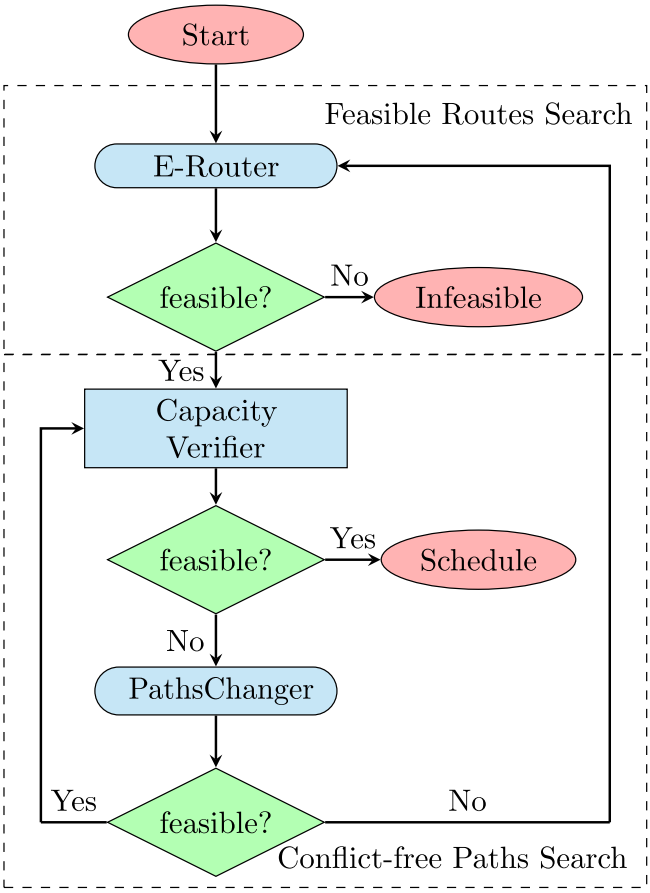


Fig. 2 Flowchart of S-ComSat

5 The conflict-free paths search

In this section the two sub-problems that form the CFPS are presented. The *Capacity Verification Problem* is modelled as a Job Shop Problem (JSP) (Manne 1960), in order to exploit the good performance of the SMT solver Z3 (Bjørner et al. 2015) in dealing with JSPs, as demonstrated in Roselli et al. (2018). The *Paths Changing Problem* formulation is inspired by Aloul et al. (2006).

5.1 The capacity verification problem

The *Capacity Verification Problem* aims to find a feasible schedule for the vehicles, where the routes that the vehicles are assigned to satisfy the capacity constraints on nodes and edges.

In this work the *Capacity Verification Problem*, as defined in Roselli et al. (2022), has been extended to account for pairs as well, since the information about conflicts must be related to a specific pair to define additional constraints in the *PC*. The following example clarifies how the concept of pairs, introduced in Section 3 is used to formulate the *Capacity Verification Problem*.

Example of routes, pairs, nodes, and edges

Let n_e be the node visited before edge e, and let e_n be the edge visited before node n. Similarly, let n^e be the node visited after edge e, and let e^n be the edge visited after node n . Let p_R^0 be the first pair of route R and n_R^* be its starting node.

Let $\mathcal{K} = \{k_1, \ldots, k_7\}$, $\mathcal{N} = \{n_1, \ldots, n_{20}\}$, and a depot d. Let $L_{k_1} = n_1$ and $L_{k_2} = n_7$, and assume two routes are computed to execute all tasks: $R_1 = \langle s_d, k_1, k_2, k_5, k_7, f_d\rangle$, $R_2 = \langle s_d, k_3, k_4, k_6, f_d\rangle$.

In order to clarify the notation introduced above, let us analyze R_1. First, the set of pairs for R_1 is defined as

$\mathcal{P}_{R_1} = \{\langle s_d, k_1\rangle, \langle k_1, k_2\rangle, \langle k_2, k_5\rangle, \langle k_5, k_7\rangle, \langle k_7, f_d\rangle\}$.

Then, let us assume that the *path* and *edge sequence* for pair $\langle k_1, k_2\rangle$ are the following:

$\theta_{\langle k_1,k_2\rangle} = \langle n_1, n_2, n_4, n_5, n_7\rangle$,

$\delta_{\langle k_1,k_2\rangle} = \langle\langle n_1, n_2\rangle, \langle n_2, n_4\rangle, \langle n_4, n_5\rangle, \langle n_5, n_7\rangle\rangle$.

Then $p_{R_1}^0 = \langle k_1, k_2\rangle$ and $n_{R_1}^* = n_1$. Also, let $e = \langle n_1, n_2\rangle$, then $n_e = n_1$, and $n^e = n_2$; for $n = n_1$, $e^n = \langle n_1, n_2\rangle$, and for $n = n_2$, $e_n = \langle n_1, n_2\rangle$.

For a specific pair $p = \langle k_1, k_2\rangle$ of a specific route R, the first and last nodes n_i and n_j are the locations of a task k_1 and k_2 respectively. Then l_{k_1}, the earliest arrival time of task k_1, is equal to l_{n_i} and u_{k_1}, the latest arrival time of task k_2, is equal tou_{n_i}. Similarly, $l_{k_2} = l_{n_j}$, and $u_{k_2} = u_{n_j}$. The same applies also to the service times of tasks k_1 and k_2, i.e., $\tau_{k_1} = \tau_{n_i}$ and $\tau_{k_2} = \tau_{n_j}$. For all the other nodes $n \in p$, $l_n = 0$, $u_n = T$ and $\tau_n = 0$. The same applies to every pair of every route. Finally, let $\mu > 0$ be a small real constant used to prevent *swapping* of vehicles' positions between a node and the previous or following edge.

The decision variables for the SMT model of the *Capacity Verification Problem* are:

x_{Rpn}: non-negative real variable that models the time when a vehicle executing route R starts using node n in pair p;

y_{Rpe}: non-negative real variable that models the time when a vehicle executing route R starts using edge e in pair p.

The model for the *Capacity Verification Problem* is:

$$x_{Rp_R^0 n_R^*} \geq \beta_R, \qquad \forall R \in \mathcal{R} \tag{27}$$

$$y_{Rpe} \geq x_{Rpn_e} + \tau_{n_e}, \qquad \forall R \in \mathcal{R},\ p \in \mathcal{P}_R,\ e \in \delta_p \tag{28}$$

$$x_{Rpn} = y_{Rpe_n} + {}^{|e_n|}/_{\Omega}, \qquad \forall R \in \mathcal{R},\ p \in \mathcal{P}_R,\ n \in \theta_p \tag{29}$$

$$l_n \leq x_{Rpn} \wedge x_{Rpn} \leq u_n, \qquad \forall R \in \mathcal{R},\ p \in \mathcal{P}_R,\ n \in \theta_p \tag{30}$$

$$x_{Rpn} \geq y_{R'p'e^n} + \mu \ \vee\ x_{R'p'n} \geq y_{Rpe^n} + \mu, \qquad \forall R, R' \in \mathcal{R},\ R \neq R',\ p \in \mathcal{P}_R,\ p' \in \mathcal{P}_{R'},\ n \in \theta_p \cap \theta_{p'},\ n \notin \mathcal{N}_H \tag{31}$$

$$y_{Rpe} \geq y_{R'p'e} + \mu \vee y_{R'p'e} \geq y_{Rpe} + \mu, \qquad \forall R, R' \in \mathcal{R},\ R \neq R',\ p \in \mathcal{P}_R,\ p' \in \mathcal{P}_{R'},\ e \in \delta_{p_1} \cap \delta_{p_2} \tag{32}$$

$$y_{Rpe} \geq y_{R'p'e'} + {}^{|e'|}/_{\Omega} \ \vee\ y_{R'p'e'} \geq y_{Rpe} + {}^{|e|}/_{\Omega}, \qquad \forall R, R' \in \mathcal{R},\ R \neq R',\ p \in \mathcal{P}_R,\ p' \in \mathcal{P}_{R'},\ e \in \delta_p,\ e' \in \delta_{p'},\ e = \bar{e}',\ g_e = g_{e'} = 1 \tag{33}$$

Equation 27 constrains the start time of a route; Eqs. 28 and 29 define the precedence among nodes and edges to visit in a route; Eq. 30 enforces time windows on the nodes that correspond to the tasks; Eq. 31 prevents vehicles from using the same node at the same time, μ is a small positive number to prevent swapping; Eqs. 32 and 33 constrain the transit of vehicles over the same edge. If two vehicles are using the same edge from the same node, one has to start at least μ after the other, and if two vehicles are using the same edge from opposite nodes one has to fully transit before the other one can start.

Based on the model described above, the algorithm *CV* is defined, that takes a set of routes $\mathcal{R}$, the start times in $\beta_R, \forall R \in \mathcal{R}$, and the current set of paths *CP* as input and returns:

- *CFS* is a list of triplets where each triplet contains the information of a vehicle (first element), a node (second element) and a time-step (third) element. Vehicles that are unscheduled vehicle have only one triplet, as they do not change location. In contrast, scheduled vehicles generate as many triplets as there are nodes in the paths forming the routes the vehicles travel. In each triplet, the third elements indicates when the vehicle reaches the node. The list is empty if, and only if, the problem is infeasible.
- $\bar{\mathcal{C}}$, the *Unsat Core* relative to constraints Eqs. 31-33 (see Section 5.3); this is empty if the problem is feasible.

5.2 Paths changing problem

In the *Paths Changing Problem*, alternative paths are computed to connect the consecutive tasks of each route. Finding alternative paths may be necessary when, for a given set of routes $\mathcal{R}$ and starting times β, no feasible schedule exists. The *Capacity Verification Problem* may be infeasible due to the current set of paths that connect the tasks' locations, therefore a different set may lead to a feasible solution. A route is defined as a sequence of tasks, and for any two consecutive tasks there is a path (a sequence of edges) connecting them. Therefore, for a route containing $i + 1$ tasks we will have i paths and for each path we can define a start and an end node, ξ_i and π_i, respectively. The sets of outgoing and incoming edges for a certain node n are denoted $\mathcal{O}_n$ and $\mathcal{I}_n$, respectively.

The decision variables used to build the SMT model of the *Paths Changing Problem* are:

w_{Rpn}: Boolean variable that represents whether the pair p of route R is using node n;
z_{Rpe}: Boolean variable that represents whether the pair p of route R is using edge e;

This problem can be split into $R \cdot i$ sub-problems (assuming all routes have $i + 1$ tasks) that find paths for each route separately; simpler and smaller models are faster to solve. Unfortunately it may be necessary to explore different combinations of paths, so to retain the information we have only one model. Therefore, let the set of all variables z be $\mathcal{Z} = \{z_{Rpe} \mid R \in \mathcal{R}, p \in \mathcal{P}_R, e \in \mathcal{E}\}$ and let the optimal solution to the *Path Changing Problem* found at iteration h be $CP^* = \{z \in \mathcal{Z} \mid z \text{ is } True\}$. Finally, let PP^* be the set containing the optimal solutions found until the $(h-1)$-th iteration. Note that it is possible to compute CP from CP^* and vice versa. Subsequently, it is possible to compute PP from PP^* and vice versa. The model is then:

$$\min_{R \in \mathcal{R},\ p \in \mathcal{P}_R,\ e \in \mathcal{E}} \sum \mathrm{If}(z_{Rpe}, |e|, 0) \tag{34}$$

$$w_{Rp\xi_p} \wedge w_{Rp\pi_p}, \qquad \forall p \in \mathcal{P}_R,\ R \in \mathcal{R} \tag{35}$$

$$\bigoplus_{e \in \mathcal{O}_{\xi_p}} (z_{Rpe}, 1), \qquad \forall p \in \mathcal{P}_R,\ R \in \mathcal{R} \tag{36}$$

$$\bigoplus_{e \in \mathcal{I}_{\xi_p}} (z_{Rpe}, 1), \qquad \forall p \in \mathcal{P}_R,\ R \in \mathcal{R} \tag{37}$$

$$z_{Rpe} \implies \neg z_{Rp\bar{e}}, \qquad \forall p \in \mathcal{P}_R,\ R \in \mathcal{R},\ e \in \mathcal{E} \tag{38}$$

$$\bigvee_{z_{Rpe} \in CP} \neg z_{Rpe}, \qquad \forall CP \in PP^* \tag{39}$$

$$\bigwedge_{\substack{n \in \mathcal{N},\\ n \neq \xi_p,\\ n \neq \pi_p}} \mathrm{If}\left(w_{Rpn}, \bigoplus_{e \in \mathcal{O}_n} (z_{Rpe}, 1) \wedge \bigoplus_{e \in \mathcal{I}_n} (z_{Rpe}, 1), \bigoplus_{e \in \mathcal{O}_n} (z_{Rpe}, 0) \wedge \bigoplus_{e \in \mathcal{I}_n} (z_{Rpe}, 0)\right), \qquad \forall p \in \mathcal{P}_R,\ R \in \mathcal{R} \tag{40}$$

The cost function Eq. 34 to minimize is the cumulative length of the used edges; Eq. 35 guarantees that, for each path of each route, the start and end nodes are used; Eqs. 36 and 37 make sure that exactly one outgoing (incoming) edge is incident with the start (end) node of a route; Eq. 38 makes sure that a path is not allowed to use both an edge and its reverse; Eq. 39 rules out all the previously found solutions; finally, Eq. 40 guarantees that if a node (different from the start or end) is selected, exactly one of its outgoing and one of its incoming edges will be used. On the other hand, if a node is not used, none of its incident edges will be used. Based on the model described above the algorithm PC is defined, that takes the previous paths PP (from which PP^* is computed) as input and returns a new set of paths NP. If and only if the *Paths Changing Problem* is infeasible then $NP = \emptyset$.

5.3 Exploiting the *MUC*

Experiments reported in Roselli et al. (2022) show that ComSat performs well for many problem instances, however, for some specific instances ComSat failed to find feasible solutions in reasonable time. Investigations revealed the PC to be the culprit. The reason is that the PC searches blindly through the possible paths that connect any two tasks, while minimizing the

paths' cumulative length. A *conflict-free* solution may involve paths that are longer than the current ones though, and the *PC* may have to explore many *shorter* solutions before finding the longer one. Improving the performance of the *PC* would be beneficial for the overall performance of ComSat, and letting the *MUC* guide the paths changing does indeed improve the search in many cases, even though not all.

When extracting the *MUC*, it is possible to track specific constraints. This feature can be exploited to focus only on the capacity constraints violations. In fact, since time windows and service time are not flexible, it is of little to no use to track constraints represented by Eqs. 27-30. Also, an infeasible formula φ may have multiple *MUCs*; in the CF-EVRP this means that conflicts may arise at different locations in the plant. In order to catch all of them, it is possible to iteratively relax the conflicting constraints from the initial formula and solve it again, until it becomes feasible. The formula will indeed become feasible eventually, since it is based on a feasible solution $\mathcal{R}$ and only the capacity constraints can make it infeasible; in the worst case all such constraints will be removed during the iterations. Note that, since not all constraints are tracked, the set of constraints $\bar{\mathcal{C}}$ returned is not an actual *Unsat Core*, since $\bar{\mathcal{C}}$ would only make the problem infeasible in conjunction with the untracked constraints. Nonetheless, it provides the information about the conflicts needed to guide the search of paths.

Let φ_0 be the conjunction of constraints Eqs. 27-33. Assume that φ_0 is infeasible, and let $\bar{\mathcal{C}}_0$ be the subset of a *MUC* retrieved by tracking constraints Eqs. 31-33. Then let $\varphi_1 = \varphi_0 \setminus \bar{\mathcal{C}}_0$, also infeasible, and let $\bar{\mathcal{C}}_1$ be the subset of a *MUC* retrieved by tracking constraints defined by Eqs. 31-33, not including the ones in $\bar{\mathcal{C}}_0$. In general, the constraints in $\bar{\mathcal{C}}_{i-1}$ can be iteratively relaxed to obtain a new formula φ_i, until a feasible $\varphi_n = \varphi_0 \setminus (\bar{\mathcal{C}}_0 \cup \ldots \cup \bar{\mathcal{C}}_{n-1})$ is found. Then $\bar{\mathcal{C}} = \bar{\mathcal{C}}_0 \cup \ldots \cup \bar{\mathcal{C}}_{n-1}$ contains all the conflicts due to the capacity constraints.

Each constraint represented by Eqs. 31-33 is defined over two routes r_1 and r_2 and their pairs p_1 and p_2 for a specific node/edge; therefore, based on the constraints in $\bar{\mathcal{C}}$, it is possible to identify a sequence of nodes $\theta^c_{p_1} \subseteq \theta_{p_1}$ and/or edges $\delta^c_{p_1} \subseteq \delta_{p_1}$ where the conflict took place. If the conflict was generated by a set of constraints from Eq. 31, then the following constraint is added to Eqs. 34-39:

$$\neg \left(\bigwedge_{n \in \theta_{p_1^c}} w_{r_1 p_1 n} \right) \vee \neg \left(\bigwedge_{n \in \theta_{p_2^c}} w_{r_2 p_2 n} \right). \tag{41}$$

On the other hand, if the conflict was caused by a set of constraints from Eq. 32 or Eq. 33, the following constraint is added to Eqs. 34-39:

$$\neg \left(\bigwedge_{e \in \delta_{p_1^c}} z_{r_1 p_1 e} \right) \vee \neg \left(\bigwedge_{e \in y_{p_2^c}} z_{r_2 p_2 e} \right). \tag{42}$$

Constraints Eqs. 41 and 42 force at least one of the routes involved in the conflict to avoid at least one of the nodes (edges, respectively) that was involved in such conflict when computing new paths. The constraint is formulated so that the choice of the route to change is left to the solver, including the possibility of changing both routes; since the problem is an optimization, the solver will choose the change that leads to the shortest cumulative paths length.

Note that in our previous work (Roselli et al. 2022) constraints Eqs. 41 and 42 were slightly different, forbidding *all* conflicting nodes and edges to be used by both routes. While this would be a stronger restriction and could be more effective to find feasible paths, it can only

work in some situations, e.g., when it is possible to completely change the path from one task to another. For this reason we generalized the constraints so that they work for any instance of the CF-EVRP.

Based on the model described by Eqs. 34-42, the function *MUC-Guided-Paths-Changer* (*GPC*) is defined, that takes the previous paths *PP* and $\bar{\mathcal{C}}$ as input and returns a new set of paths *NP*. If and only if the *Paths Changing Problem* is infeasible $NP = \emptyset$.

Since the constraints added to the *GPC* are based on the constraints in $\bar{\mathcal{C}}$, it is imperative that the *Unsat Core* returned when the *CV* is infeasible is *minimal*. This is so because if the *Unsat Core* is not minimal, it could contain constraints that are not actually causing *capacity conflicts*. These constraints would in turn lead to defining constraints Eqs. 41 and 42 in the *GPC* that may remove feasible solutions.

Figure 3 summarizes the steps required to find a conflict-free schedule *CFS*, if such exists, using the improved paths searching algorithm *GPC*. As mentioned, it is assumed that routes $\mathcal{R}$ and their start times $\beta_r,\ \forall r \in \mathcal{R}$ have already been computed. The shortest paths between any two tasks are computed using Dijkstra's algorithm and then set as the current paths *CP*. Also, *CP* are added to the list of previous paths *PP*.

Then the *CV* will check such routes against the capacity constraints; if this sub-problem has a feasible solution the algorithm terminates and a conflict-free schedule is returned. Otherwise $\bar{\mathcal{C}}$ is extracted as described in the previous paragraph and the *GPC* algorithm is invoked. *GPC* will use the information about previously computed paths *PP* and the information about conflicts from $\bar{\mathcal{C}}$ to compute new paths *NP*, which will be set as the current paths and stored in *PP*. At this point the *CV* is run again using the new paths. The iterations between the two algorithms continue until either the *CV* is feasible, or the *GPC* is infeasible, i.e., there are no feasible, conflict-free paths to execute the routes $\mathcal{R}$.

6 Proof of soundness and completeness

In this section, we provide proof of soundness and completeness of $ComSat_2$ and S-ComSat. Since soundness and completeness of the previous version of ComSat has already been proven

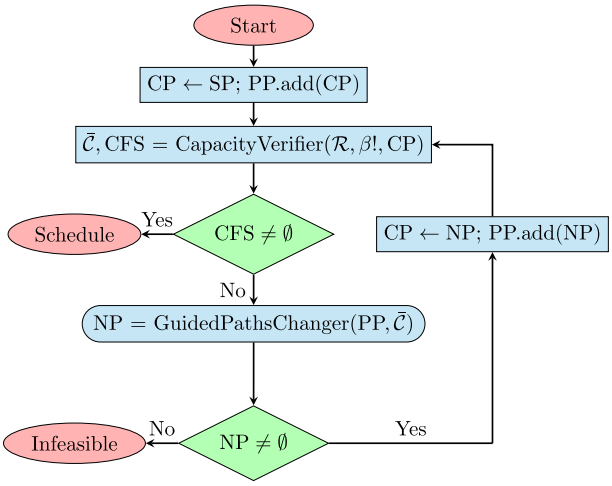


Fig. 3 Flowchart of the *MUC*-Guided-CFPS

in Roselli et al. (2022), soundness and completeness of the new versions can be proven by showing that:

- in the FRS, *E-Router* is able to enumerate all possible assignments that would originate from calling *Router* and *Assign*;
- in the CFPS, *GPC* can enumerate at least as many feasible solutions of the *Paths Changing Problem* as *PC*.

Observation 1 All the problem classes presented in this work are decidable. This is true because they are all combinations of decidable first-order theories and therefore the *Nelson-Oppen theory combination method* (Tinelli and Harandi 1996) applies. In fact the *Routing* and *E-Routing* problems are a combination of linear arithmetic and propositional logic, the *Assignment Problem* and the *Capacity Verification Problem* all fall into the category of difference logic (a fragment of linear arithmetic), and the *Paths Changing Problem* is a propositional logic problem.

Observation 2 The optimization problems solved in ComSat, i.e., the *Routing* and *E-Routing* problems, and the *Paths Changing Problem*, are bounded. The *Routing* and *E-Routing* problems involve a finite number of decision variables that are either integers with a finite domain ($\{0, 1\}$), or non-negative reals and the objective is minimization, with an objective function involving only non-negative coefficients. The *Paths Changing Problem* involves only a finite number of Boolean variables, so the problem domain is finite.

6.1 Soundness and completeness of the FRS

In order to prove soundness and completeness of S-ComSat we are going to show that *E-Router* can compute the same solutions to the FRS as the combination of *Routing* and *Assignment* problems.

Lemma 1 *The FRS has a finite number of feasible solutions.*

Proof By the definition of route given in Section 3, a route contains an ordered set of unique tasks from $\mathcal{K}$. Also, a route must start and end at the same depot, and may include up to $|\mathcal{K}|$ dummy tasks representing charging events at the depots. If we do not consider $\mathcal{D}$, a set of routes $\mathcal{R}$ to execute all tasks in $\mathcal{K}$ is a *partition* of $\mathcal{K}$. The number of partitions of the set $\mathcal{K}$ is the *Bell Number* (Comtet 1974), $B_{|\mathcal{K}|} < \infty$. Given $|\mathcal{D}|$ depots, in the worst case, there may be $2^{|\mathcal{D}|}$ combinations of depots and partitions of $\mathcal{K}$. Also, for each of these combinations, there can be a any number of *extra* visits, between 0 and $|\mathcal{K}|$, to the depots for charging. Finally, for each solution to the *Routing Problem*, there can be as much as $2^{|\mathcal{V}|}$ assignments to vehicles. Therefore, there may be $B_{|\mathcal{K}|} \cdot 2^{|\mathcal{D}|} \cdot |\mathcal{D}| \cdot 2^{|\mathcal{V}|} < \infty$ solutions to a FRS with $|\mathcal{K}|$ tasks, $|\mathcal{V}|$ vehicles, and $|\mathcal{D}|$ depots, assuming that for each depot there can be $|\mathcal{K}|$ charging stations. □

Lemma 2 *Repeated calls to* E-Router *will enumerate all feasible solutions before returning infeasible.*

Proof Let φ_0 be a conjunction of the constraints Eqs. 7-26, and let $\mathcal{A}_0$ be a solution to φ_0. Then, if another solution $\mathcal{A}_1$ for φ_0 exists, it can be found by solving $\varphi_0 \wedge \neg\mathcal{A}_0 = \varphi_1$. In general, the n-th solution can be found by solving $\varphi_0 \wedge \neg\mathcal{A}_0 \wedge \ldots \wedge \neg\mathcal{A}_{n-1} = \varphi_n$. Because of Lemma 1, we know that the number of solutions to φ, i.e., $\mathbb{S}(\varphi) < \infty$ and we enumerate all by solving $\varphi_0, \ldots, \varphi_{\mathbb{S}(\varphi)-1}$. □

Observation 3 Following the same reasoning used in Lemma 2, *Routing* and *Assign* together can enumerate all feasible solutions to the FRS before returning infeasible, since the number of route sets $\mathcal{R}$ to serve a set of tasks $\mathcal{K}$ is finite (Lemma 1) and so is the number of vehicles that can be assigned to them.

One important difference between the *Routing* and *E-Routing* problem is constraint Eq. 21 that allows at most one route to be assigned to a vehicle. This so since in the *E-Routing Problem* vehicles are allowed to go back to the depot to recharge, while in the *Routing Problem* vehicles can be assigned multiple routes and the battery recharge is performed in between routes and handled by *Assign*. On the other hand, in the *Routing Problem* a route's length cannot exceed the operating range of the vehicle, since charging is not included. For instance, let us assume ComSat is solving an instance of the CF-EVRP; it calls *Router* which would compute the routes $R_1 = \langle s_d, k_1, \ldots, k_i, f_d \rangle$, and $R_2 = \langle s_d, k_{i+1}, \ldots, k_j, f_d \rangle$ both only executable by vehicle v, available in the fleet. Now let us assume that the time windows of tasks $k_1, \ldots, k_j$ are all $[0, T]$ and T is large enough so that *Assign* is able to assign vehicle v to both routes. If the same instance of the CF-EVRP was to be solved by S-ComSat, a call to *E-Router* may compute the route $R_1 = \langle s_d, k_1, \ldots, k_i, z_d, k_{i+1}, \ldots, k_j, f_d \rangle$ and directly assign R_1 to vehicle v. Therefore *Router* + *Assign* can compute the same solution that *E-Router* would compute and vice versa. Let us now generalize this concept to any solution.

Lemma 3 Router + Assign *can compute any solution that* E-Router *would compute.*

Proof Trivially, if tasks can be served without having to recharge, *Router* computes the same routes as *E-Router*. Otherwise, an assignment computed by *E-Router* may look like (v, R_1) with $R_1 = \langle s_d, k_1, \ldots, k_i, z_d, k_{i+1}, \ldots, k_j, f_d \rangle$ with a length exceeding the operating range ψ, since charging can be performed at the depot d. The route can be divided into as many routes as charging tasks included in the route, i.e., $R_2 = \langle s_d, k_1, \ldots, k_i, f_d \rangle$, and $R_3 = \langle s_d, k_{i+1}, \ldots, k_j, f_d \rangle$. The resulting routes are disjoint subsets of $\mathcal{K}$ of length shorter than the vehicle's operating range. Hence they can form a solution computed by *Router*. Moreover, since R_1 is computed by *E-Router*, R_2 and R_3 can be assigned to one vehicle, namely v; hence $(v, R_2) \wedge (v, R_3)$ can form a solution computed by *Assign*. □

Lemma 4 E-Router *can compute any solution that* Router + Assign *would compute.*

Proof Trivially, if tasks can be served without having to recharge, *E-Router* computes the same assignment as *Assign* to the routes computed by *Router*. Otherwise, if two routes starting at the same depot d are assigned to the same vehicle v, they look like $R_2 = \langle s_d, k_1, \ldots, k_i, f_d \rangle$, and $R_3 = \langle s_d, k_{i+1}, \ldots, k_j, f_d \rangle$. Since the length of each of these is at most as long as the operating range of vehicle v, they could be merged into a single route $R_1 = \langle s_d, k_1, \ldots, k_i, z_d, k_{i+1}, \ldots, k_j, f_d \rangle$. This can be true for an arbitrary number of routes that are assigned the same vehicle. Moreover, since they were both assigned the same vehicle, it means that v has enough time to recharge its battery in between the execution of the routes. Therefore $(v, R_1) = \langle s_d, k_1, \ldots, k_i, z_d, k_{i+1}, \ldots, k_j, f_d \rangle)$ can be computed by *E-Router*. □

Theorem 1 *S-ComSat in* Fig. 2 *is sound and complete.*

Proof Because of Lemmas 3 and 4, any solution to the FRS computed by one method, can be computed by the other. Thus, since ComSat is sound and complete (Roselli et al. 2022), so is S-ComSat. □

6.2 Soundness and completeness of *ComSat*$_2$

The proof for soundness and completeness of the CFPS builds on the following. There exists a finite number of solutions to the *Paths Changing Problem*; the *GPC* can enumerate at least all feasible solutions to the *Paths Changing Problem*; if a solution that satisfies the *Capacity Constraints* does exist, the *GPC* will eventually find it, otherwise it will declare the problem infeasible.

Let $\mathcal{S}$ be the set of possible solutions to a *Paths Changing Problem*; divide $\mathcal{S}$ into the set of *conflict-free solutions* $\mathcal{F}$ and the set of *conflicting solutions* $\mathcal{U}$. In other words, a solution to the *Paths Changing Problem* from $\mathcal{F}$ will make the *Capacity Verification Problem* feasible, while a solution from $\mathcal{U}$ will not. If the CFPS is infeasible, then $\mathcal{S} = \mathcal{U}$ and $\mathcal{F} = \emptyset$.

In case the CFPS is feasible, though, in order to prove completeness it is necessary to guarantee that at least all feasible solutions $\mathcal{F}$ can be found by *GPC*. This is proven for the *PC*, since each call of the *PC* function will find the next optimal solution to the *Paths Changing Problem*, whether it belongs to $\mathcal{F}$ or not, until all solutions have been enumerated. However, in the *GPC* there are additional constraints that may remove feasible solutions. In the proof below it is shown that such additional constraints only remove infeasible solutions.

Observation 4 Given a finite, directed, weighted graph, the number of paths that connect two arbitrary nodes is finite.

Proof By definition, a path is an ordered set of nodes such that no node appears more than once. If the number of nodes in the graph is finite, there cannot be an infinite number of paths. □

Lemma 5 *For a given set of routes $\mathcal{R}$ and start times in β, repeated calls to the PC function will enumerate all feasible solutions to the* Paths Changing Problem*, either belonging to $\mathcal{F}$ or $\mathcal{U}$, before returning infeasible.*

Proof Let φ_0 be the conjunction of constraints Eqs. 35-40, a relaxation of the *Paths Changing Problem*, and let CP_0 be a solution to φ_0. Then, if another solution CP_1 for φ_0 exists, it can be found by solving $\varphi_0 \wedge \neg CP_0 = \varphi_1$. In general, the n-th solution can be found by solving $\varphi_0 \wedge \neg CP_0 \wedge \ldots \wedge \neg CP_{n-1} = \varphi_n$. Because of Lemma 4, we know that the number of solutions to the *Paths Changing Problem*, $|\mathcal{S}|$, is finite and we can enumerate them all by solving $\varphi_0, \ldots, \varphi_{|\mathcal{S}|-1}$. □

Lemma 6 *Using the PC and CV is a sound and complete procedure to solve the CFPS.*

Proof Because of Observation 4 we know there is a finite number of solutions to the *Paths Changing Problem*, and because of Lemma 5 we know that the *PC* function can enumerate them all. If a solution that belongs to $\mathcal{F}$ exists, the *PC* will find it, otherwise it will return all solutions belonging to $\mathcal{U}$; the *CV* will then check whether they are conflict-free. Therefore, using the *PC* and *CV* in combination will correctly solve the CFPS. □

Lemma 7 *For a given set of routes $\mathcal{R}$, the GPC is able to find at least all solutions in $\mathcal{F}$.*

Proof For each set of current paths *CP*, $\bar{\mathcal{C}}$ only contains constraints defined by Eqs. 31, 32, and 33. The constraints in $\bar{\mathcal{C}}$ are iteratively retrieved from *Unsat Cores* that are *minimal* and thus represent combinations of nodes and edges that are actually involved in conflicts. Since the constraints defined by Eqs. 41 and 42 address the constraints from $\bar{\mathcal{C}}$, Eqs. 41 and 42 only define constraints over nodes or edges that cause conflicts. Hence these constraints only remove solutions of the *Paths Changing Problem* that belong to $\mathcal{U}$. □

Theorem 2 *Using the GPC and CV is a sound and complete procedure to solve the CFPS.*

Proof The *PC* and the *GPC* are identical, except for constraints Eqs. 41-42, and because of Lemma 7, we know that the addition of these constraints only removes solutions from $\mathcal{U}$. Thus, since the CFPS using the *PC* is sound and complete (Lemma 6), so is the CFPS using the *GPC*. □

7 Experiments

To evaluate the efficacy of $ComSat_2$ and S-ComSat, as well as to compare their performance with $ComSat_1$, problem instances are designed and used for testing. The algorithms called by ComSat used the SMT solver Z3 4.11.2 and the MILP solver (Gurobi 2023) 9.1.2 to compute solutions to the sub-problems the CF-EVRP is decomposed into. All the experiments[2] were performed on an *Intel Core i7 6700K, 4.0 GHZ, 32GB RAM* running *Ubuntu-18.04 LTS*.

The first set of experiments is run over one of the benchmark sets of the CF-EVRP presented in Roselli et al. (2022). Instances are characterized by the following parameters:

- *NVK* indicates the number of nodes N in the graph that represents the plant layout, the number of vehicles V, and the number of tasks K;
- *Nodes Connection* refers to the connectivity of the graph. The graph in the instances is grid-like (e.g. for $N = 15$ the graph is a 3×5 grid) and the value of *Node Connection* 100 means that all edges needed to form a grid (in both directions, e.g. from Node *1* to Node *2* and from Node *2* to Node *1*) are present. For a *Node Connection* value of 90, ten percent of the edges are removed from the original graph, and for a *Node Connection* value of 80, twenty percent of edges are removed. In both cases edges are removed so that the graph remains *connected*;
- the time horizon T;
- a *Seed* connected to the random generation of parameters such as tasks location, time windows, charging coefficients and operating range.

In this test, the overall running time for termination is compared for $ComSat_1$, $ComSat_2$, and S-ComSat. In the evaluation, the number of calls of *Router* and *E-Router* is also considered. Repeated calls to *Router* might be necessary for two reasons; either the *Assignment Problem* is infeasible, or the *Capacity Verification Problem* is. Conversly, when repeated calls of *E-Router* are made, it is solely because the previous ones rendered the *Capacity Verification Problem* infeasible. The maximum number of routes is capped at 200; if a solution is not found or infeasibility is not confirmed by the 200th call of *Router* (or *E-Router*), the instance is declared *unknown*. A total of 180 instances is generated combining *NVK* values of 15-3-10 and 25-4-14, *Nodes Connection* values of 100, 90, and 80, and T values of 20, 25, 30, 40, 50, and 60. For each combination of these values, five instances are generated using different seeds. Table 1 shows, for each combination of parameters, calculated over the five instances with different seeds, the number of them that where feasible (or infeasible, respectively), the average solving time in seconds for both the feasible and the infeasible instances, and the average number of *Router* (or *E-Router*) calls for both the feasible and the infeasible instances. It is important to note that the average values for solving time and the number of calls to *Router* (or *E-Router*) are calculated only over the instances that were actually solved, not the *unknown* ones.

[2] The implementation of the models presented in Sections 4 and 5, and the problem instances are available at https://github.com/sabinoroselli/VRP.git.

Table 1 Comparison of $ComSat_1$, $ComSat_2$, and S-ComSat over a set of benchmark instances of the CF-EVRP

	$ComSat_1$			$ComSat_2$			S-ComSat		
T	F/I	Sol. Time	R. Calls	F/I	Sol. Time	R. Calls	F/I	Sol. Time	R. Calls
NVK 15-3-10									
Nodes Connection 0									
20	(1,2)	(0.22,0.08)	(1,1)	(1,4)	(2.33,10.59)	(1,36)	(1,4)	(0.18,0.12)	(1,1)
25	(2,2)	(0.18,0.08)	(1,1)	(2,3)	(0.18,11.78)	(1,34)	(2,3)	(0.16,0.12)	(1,1)
30	(3,2)	(0.55,0.07)	(3,1)	(3,2)	(0.22,0.08)	(1,1)	(3,2)	(0.14,0.10)	(1,1)
40	(3,2)	(0.58,0.07)	(5,1)	(3,2)	(0.30,0.07)	(2,1)	(3,2)	(0.16,0.09)	(1,1)
50	(3,2)	(0.28,0.08)	(1,1)	(3,2)	(0.18,0.08)	(1,1)	(3,2)	(0.15,0.09)	(1,1)
60	(2,2)	(0.22,0.08)	(1,1)	(3,2)	(0.46,0.07)	(3,1)	(3,2)	(0.17,0.09)	(1,1)
Nodes Connection 25									
20	(1,2)	(0.17,0.08)	(1,1)	(1,4)	(0.16,7.19)	(1,30)	(1,4)	(0.14,0.11)	(1,1)
25	(2,2)	(33.15,0.08)	(12,1)	(2,3)	(11.48,5.69)	(6,24)	(2,3)	(0.40,0.11)	(1,1)
30	(2,2)	(0.44,0.07)	(4,1)	(2,3)	(0.20,24.02)	(2,47)	(2,3)	(0.15,0.14)	(1,1)
40	(2,2)	(0.20,0.08)	(2,1)	(3,2)	(0.85,0.07)	(5,1)	(3,2)	(0.16,0.09)	(1,1)
50	(3,2)	(0.44,0.08)	(3,1)	(3,2)	(0.21,0.07)	(1,1)	(3,2)	(0.16,0.09)	(1,1)
60	(2,2)	(0.18,0.07)	(1,1)	(3,2)	(0.29,0.07)	(2,1)	(3,2)	(0.42,0.09)	(1,1)
Nodes Connection 50									
20	(1,2)	(0.71,0.07)	(15,1)	(1,4)	(0.08,2.26)	(1,13)	(1,4)	(0.10,0.10)	(1,1)
25	(0,2)	(-,0.07)	(-,1)	(0,5)	(-,3.89)	(-,15)	(0,5)	(-,0.13)	(-,1)
30	(1,2)	(27.70,0.07)	(31,1)	(1,4)	(0.19,5.27)	(1,23)	(1,4)	(0.14,0.12)	(1,1)
40	(3,2)	(12.72,0.07)	(35,1)	(3,2)	(0.70,0.07)	(1,1)	(3,2)	(2.27,0.09)	(2,1)
50	(3,2)	(0.99,0.08)	(9,1)	(3,2)	(0.27,0.07)	(2,1)	(3,2)	(0.18,0.09)	(1,1)
60	(3,2)	(4.66,0.07)	(35,1)	(3,2)	(0.26,0.07)	(1,1)	(3,2)	(0.16,0.08)	(1,1)

Table 1 continued

T	$ComSat_1$ F/I	$ComSat_1$ Sol. Time	$ComSat_1$ R. Calls	$ComSat_2$ F/I	$ComSat_2$ Sol. Time	$ComSat_2$ R. Calls	S-ComSat F/I	S-ComSat Sol. Time	S-ComSat R. Calls
NVK 25-4-14									
Nodes Connection 0									
20	(0,0)	(-,-)	(-,-)	(0,4)	(-,24.08)	(-,42)	(0,5)	(-,0.42)	(-,1)
25	(1,0)	(0.37,-)	(1,-)	(1,3)	(0.40,28.92)	(1,64)	(1,4)	(0.28,0.49)	(1,1)
30	(1,0)	(1.76,-)	(6,-)	(2,2)	(1.37,297.96)	(2,140)	(2,3)	(0.28,0.99)	(1,1)
40	(3,0)	(35.68,-)	(21,-)	(3,1)	(0.76,23.57)	(1,65)	(3,2)	(0.53,0.74)	(1,1)
50	(3,0)	(18.46,-)	(59,-)	(3,2)	(0.88,222.09)	(2,137)	(3,2)	(0.31,0.39)	(1,1)
60	(5,0)	(7.51,-)	(28,-)	(5,0)	(0.54,-)	(1,-)	(5,0)	(0.30,-)	(1,-)
Nodes Connection 25									
20	(0,1)	(-,0.16)	(-,1)	(0,4)	(-,8.58)	(-,28)	(0,5)	(-,0.22)	(-,1)
25	(0,0)	(-,-)	(-,-)	(0,4)	(-,12.73)	(-,40)	(0,5)	(-,0.40)	(-,1)
30	(1,0)	(0.66,-)	(2,-)	(1,3)	(0.39,92.18)	(1,79)	(1,4)	(38.93,0.41)	(30,1)
40	(2,0)	(0.78,-)	(3,-)	(2,2)	(0.90,99.18)	(1,41)	(2,3)	(0.28,9.52)	(1,13)
50	(2,0)	(13.83,-)	(6,-)	(3,2)	(19.02,156.92)	(9,100)	(3,2)	(22.07,0.27)	(6,1)
60	(5,0)	(6.51,-)	(6,-)	(5,0)	(1.98,-)	(6,-)	(5,0)	(0.31,-)	(1,-)
Nodes Connection 50									
20	(0,1)	(-,0.15)	(-,1)	(0,4)	(-,3.96)	(-,15)	(0,5)	(-,0.19)	(-,1)
25	(0,0)	(-,-)	(-,-)	(0,4)	(-,8.27)	(-,29)	(0,5)	(-,0.29)	(-,1)
30	(1,0)	(0.77,-)	(3,-)	(1,3)	(0.31,27.87)	(1,47)	(1,4)	(0.30,0.42)	(1,1)
40	(3,0)	(1.06,-)	(2,-)	(3,1)	(0.93,5.61)	(1,21)	(3,2)	(0.69,0.29)	(1,1)
50	(2,0)	(5.54,-)	(20,-)	(3,2)	(0.74,75.97)	(2,82)	(3,2)	(20.03,0.24)	(6,1)
60	(5,0)	(5.72,-)	(23,-)	(5,0)	(0.63,-)	(2,-)	(5,0)	(0.34,-)	(1,-)

Instances are classified based on the parameters *NVK*, *Nodes Connection*, and *T*. For each category of instances the number of feasible and infeasible instances (F/I), the corresponding solving time in seconds (feasible left and infeasible right) and the number of calls to the *Router* (feasible left and infeasible right) is reported. The symbol "-" means that no instance for that specific category has been solved

The results show that, for each category of instances, $ComSat_2$ is able to solve more instances than $ComSat_1$. For the infeasible instances, this often leads to a higher number of *R.Calls* and a longer average solving time for $ComSat_2$, since the harder instances are simply not solved by $ComSat_1$ and therefore not counted in. On the other hand, $ComSat_1$ and $ComSat_2$ can solve roughly the same number of feasible instances, but $ComSat_1$ requires a larger number of *R.Calls* and, therefore, a longer solving time. It is also possible to see a correlation between the solving time and the number of *R.Calls* for both $ComSat_1$ and $ComSat_2$. The time required for a single iteration of $ComSat_2$ is generally longer compared to $ComSat_1$, but since the number iterations required is smaller, its overall performance is better.

On the other hand, S-ComSat could solve all instances from all categories except five in less than one second. The categories in which the overall execution took longer than one second are the ones that required more than one call of *E-Router*. Hence, merging the *Routing* and *Assignment* problems into one and solving this problem using *E-Router* is the preferred choice in terms of solving time. Moreover, it is interesting to note that the computation time saved is not only due to a limited number of calls of *E-Router* compared to *Router*. In fact, even when comparing the time required for one call, *E-Router*'s solving time is in the same order of magnitude as *Router*.

Overall, $ComSat_1$ is able to solve 109 instances, with an average time of about 1 second for the feasible instances, 0.1 second for the infeasible ones, and 46 seconds for the unknown ones; on the other hand, $ComSat_2$ is able to solve 168 instances, taking on average about half a second for the feasible instances, 1.78 seconds for the infeasible ones, and about 350 seconds for the unknown ones; finally S-ComSat is able to solve all 180 instances, taking about 0.3 seconds for the feasible ones and 0.1 seconds for the infeasible ones. Table 2 shows the results discussed in this paragraph. Note that the geometric mean is used instead of the arithmetic mean in order to get rid of biases due to outliers.

Aside for the previously described benchmark sets, we have also tested S-ComSat on randomly generated, increasingly larger instances, in order to evaluate its scalability. The largest generated instance had *NVK* equal to 200-30-50. Of course, different instances of the same size may require different solving time, but the solving time of the sub-problems can still provide insights on scalability. Of the three sub-problems, *E-Router* is the least sensitive to the increase in the instance size, while the *CapacityVerifier* and especially the *Paths-Changer* become significantly slower as the problem sizes grow. Already for problems with *NVK* equal to 100-15-25 a *Paths-Changer* call can take over a minute. Above *NVK* 150-20-30 also a single call of *CapacityVerifier* takes on average one minute, and for the largest instances even the solving time of *E-Router* is measured in minutes. Overall, if S-ComSat

Table 2 Comparison of $ComSat_1$, $ComSat_2$, and S-ComSat average solving time (in seconds) and number of instances solved, after sorting the instances according to their feasiblity

	$ComSat_1$		$ComSat_2$		S-ComSat	
	# Instances	Sol. Time	# Instances	Sol. Time	# Instances	Sol. Time
Feasible	71	1.10	77	0.47	77	0.31
Infeasible	38	0.08	91	1.78	103	0.19
Unknown	71	46.55	12	359.23	0	-

does not require several iterations through the sub-problems to find a solution, it can solve instances with hundreds of nodes and tens of vehicles and tasks within a few minutes, at most. When iterations are required, the *Paths-Changer* remains the bottleneck of the algorithm, especially when the problem instance involves a large graph.

In the second set of experiments, the goal is to compare the search for alternative paths, therefore additional instances are designed in such a way that there is only one feasible set of routes $\mathcal{R}$ to execute the tasks and the shortest paths to execute the routes cause infeasibility of the *Capacity Verification Problem*.

Table 3 shows the results of the evaluation of five problem instances of the CF-EVRP solved using $ComSat_1$. Each instance was solved three times, once using the *PC*, once using the *GPC* as defined in Roselli et al. (2022), henceforth called *GPC**, and once using the *GPC* as described in this work; in each case, the number of iterations and the time (in seconds) required to find a feasible solution is reported. The problem instances presented are increasingly hard to solve, in terms of plant size (represented by the number of nodes), number of routes, and number of tasks in each route. The tasks' locations and time windows are set so that conflicts will arise due to the capacity constraints when the shortest paths are used and a search for alternative paths will be necessary in order to find a conflict-free schedule.

Results from Roselli et al. (2022) differ from the results of Table 3 in terms of running time for the *GPC** and both running time and the number of iterations for the *PC*. We believe this to be related to using a different version of Z3 (4.8.9 in Roselli et al. (2022)) since the implementation and computer used in both cases are the same. For this reason, in order to make a fair comparison, we solved all instances again.

When solving instances *1* through *4* *GPC* takes less time and fewer iterations than *PC*. However, for instance *5*, *PC* can find a solution in only five iterations and 69 seconds, while *GPC* takes 23 iterations and 353 seconds. *GPC** is able to solve all instances in only one iteration. Moreover, it seems that the additional constraints in *GPC** slow down the search for a solution to the *Paths Changing Problem*, whereas for *GPC* there is no noticeable difference in the time required for one iteration, compared to *PC*.

These results do not come unexpectedly. *GPC** has strong restrictions on nodes and edges that routes can use since each node/edge involved in a conflict must be avoided by one of the two routes that use it. Hence it may be hard to find a feasible solution and it takes a longer

Table 3 Comparison of the *PC* and *GPC* over a set of instances of the CF-EVRP

Inst.	$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{K}\rvert$	Iterations			Time		
				PC	*GPC**	*GPC*	*PC*	*GPC**	*GPC*
1	3	2	4	2	1	1	0.12	0.08	**0.07**
2	8	3	6	8	1	5	1.18	**0.26**	0.4
3	5	4	8	53	1	25	6.15	**0.53**	2.71
4	64	4	28	12	1	6	177.56	96.06	**92.14**
5	64	4	28	5	1	23	69.05	**68.92**	353.03

For each instance, the number of iterations and the total running time (in seconds) required to find a feasible solution are reported

The bold entries are used to highlight the best results for each category

time to do so. On the other hand, *GPC* does not have such strong restrictions, and the solving time for one iteration is barely affected by the additional constraints when compared to *PC*. In turn, because of the less restrictive constraints, it takes more iterations to find feasible solutions. Instance *5* shows that *GPC* may fail to reduce the number of iterations compared to *PC*.

On the other hand, *GPC* is guaranteed to find a feasible solution, if such exists, for any instance of the CF-EVRP, while *GPC** only works if it is possible to avoid all edges/nodes involved in a conflict. In general, *GPC** might be useful for graphs with high connectivity, where a different path can likely be found; unfortunately, determining beforehand whether a graph is suitable for *GPC** might be a hard problem itself.

8 Conclusion

In this paper, we presented improvements on the compositional algorithm ComSat, first presented in Roselli et al. (2022). The algorithm is designed to solve instances of the CF-EVRP (Roselli et al. 2021), a vehicle routing problem involving capacity constraints on the road segments and vehicles' limited operating range. Our previous work on the topic highlighted bottlenecks in ComSat that, for some problem instances, led to relatively long solving time. These bottlenecks are the *Feasible Routes Search* (FRS), performed through iterations between the algorithms *Router* and *Assign* within ComSat, and the *Conflict-Free Paths Search* (CFPS), performed through iterations between the *PC* and *CV* algorithms.

For the FRS, we introduced a new version of *Router* including a tighter constraint to rule out previous solutions, as well as a new algorithm that deals with *routing* and *assignment*, i.e., FRS, simultaneously. For the CFPS, we presented new versions of *PC* and *CV* that make use of the *Unsat Cores* to guide the search for feasible paths. We proved soundness and completeness of the different versions of ComSat originating from replacing the existing algorithms with the ones presented in this work.

We also compared the performance of the proposed algorithms that form part of ComSat, to the previous ones presented in Roselli et al. (2022). For the FRS, both $ComSat_2$ and S-ComSat outperformed $ComSat_1$, with S-ComSat being the fastest, sometimes by more than one order of magnitude. For the CFPS, the *GPC* needed fewer iterations and less time than the *PC* to find conflict-free paths in four cases out of five. In comparison, the previous version of the *GPC* presented in Roselli et al. (2022) shows a stronger impact on reducing the number of iterations, although it is not always guaranteed to return feasible solutions, if such exist. In conclusion, this method for the CFPS shows potential, but further investigation is needed to assess its actual effectiveness.

Acknowledgements We gratefully acknowledge financial support from Vinnova project CLOUDS (Intelligent algorithms to support Circular soLutions fOr sUstainable proDuction Systems), Chalmers AI Research Centre (CHAIR), ITEA3-projektet AIToC (Artificial Intelligence supported Tool Chain in Manufacturing Engineering), and the Wallenberg AI, Autonomous Systems and Software program (WASP) funded by the Knut and Alice Wallenberg Foundation.

Funding Open access funding provided by Chalmers University of Technology.

Declarations

Conflict of Interests The authors declare to have no conflict of interest.

References

Abderrahim M, Bekrar A, Trentesaux D, Aissani N, Bouamrane K (2020) Manufacturing 4.0 operations scheduling with AGV battery management constraints. Energies 13(18):4948

Aloul FA, Al Rawi B, Aboelaze M (2006) Identifying the shortest path in large networks using boolean satisfiability. In: 2006 3rd international conference on electrical and electronics engineering, pp 1–4

Arumugam S, Brandstädt A, Nishizeki T, Thulasiraman K (2016) Handbook of Graph Theory, Combinatorial Optimization, and Algorithms, vol 34. CRC Press, New York

Barrett CW, Sebastiani R, Seshia SA, Tinelli C et al (2009) Satisfiability modulo theories. Handbook of Satisfiability 185:825–885

Bjørner N, Phan A-D, Fleckenstein L (2015) νZ-an optimizing SMT solver. In: International conference on tools and algorithms for the construction and analysis of systems, pp 194–199

Cimatti A, Griggio A, Sebastiani R (2011) Computing small unsatisfiable cores in satisfiability modulo theories. J Artif Intell Res 40:701–728

Comtet L (1974) Advanced Combinatorics: The Art of Finite and Infinite Expansions. Springer, Dordrecht

Corréa AI, Langevin A, Rousseau L-M (2007) Scheduling and routing of automated guided vehicles: a hybrid approach. Comput Oper Res 34(6):1688–1707

Cortés-Murcia DL, Prodhon C, Afsar HM (2019) The electric vehicle routing problem with time windows, partial recharges and satellite customers. Transport Res Part E: Logistic Transport Rev 130:184–206

Dantzig GB, Ramser JH (1959) The truck dispatching problem. Manage Sci 6(1):80–91

Daugherty G, Reveliotis S, Mohler G (2018) Optimized multiagent routing for a class of guidepath-based transport systems. IEEE Trans Autom Sci Eng 16(1):363–381

De Moura L, Bjørner N (2011) Satisfiability modulo theories: introduction and applications. Commun ACM 54(9):69–77

De Ryck M, Versteyhe M, Debrouwere F (2020) Automated guided vehicle systems, state-of-the-art control algorithms and techniques. J Manuf Syst 54:152–173

De Moura L, Bjørner N (2008) Z3: an efficient SMT solver. In: International conference on tools and algorithms for the construction and analysis of systems, Springer, Berlin, Heidelberg, pp 337–340

Dershowitz N, Hanna Z, Nadel A (2006) A scalable algorithm for minimal unsatisfiable core extraction. In: International conference on theory and applications of satisfiability testing, pp 36–41

Desrochers M, Desrosiers J, Solomon M (1992) A new optimization algorithm for the vehicle routing problem with time windows. Oper Res 40(2):342–354

Dijkstra EW (1959) A note on two problems in connexion with graphs. Numer Math 1(1):269–271

Glover F (1989) Tabu search—part I. ORSA J Comput 1(3):190–206

Gurobi Optimization, LLC (2023) Gurobi Optimizer Reference Manual. https://www.gurobi.com

Hansen P, Mladenović N (2005) In: Burke EK, Kendall G (eds) Variable Neighborhood Search, pp 211–238. Springer, Boston, MA

Hart PE, Nilsson NJ, Raphael B (1968) A formal basis for the heuristic determination of minimum cost paths. IEEE Trans Syst Sci Cybern 4(2):100–107

Huang J (2005) MUP: a minimal unsatisfiability prover. In: Proceedings of the ASP-DAC 2005. Asia and South Pacific design automation conference, 2005, vol 1, pp 432–437

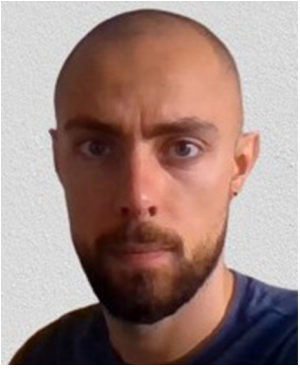
Keskin M, Çatay B (2016) Partial recharge strategies for the electric vehicle routing problem with time windows. Transport Res part C: Emerg Technol 65:111–127

Kondili E, Pantelides CC, Sargent RW (1993) A general algorithm for short-term scheduling of batch operations—I. MILP formulation. Computers & Chemical Engineering 17(2):211–227

Krishnamurthy NN, Batta R, Karwan MH (1993) Developing conflict-free routes for automated guided vehicles. Oper Res 41(6):1077–1090

Kroening D, Strichman O (2016) Decision procedures-An Algorithmic Point of View. Springer, Heidelberg

Kucukoglu I, Dewil R, Cattrysse D (2021) The electric vehicle routing problem and its variations: a literature review. Comput Industrial Eng 161:107650
Lim A, Wang F (2005) Multi-depot vehicle routing problem: a one-stage approach. IEEE Trans Autom Sci Eng 2(4):397–402
Manne AS (1960) On the job-shop scheduling problem. Oper Res 8(2):219–223
Murakami K (2020) Time-space network model and MILP formulation of the conflict-free routing problem of a capacitated AGV system. Comput Ind Eng 141:106270
Nadel A (2010) Boosting minimal unsatisfiable core extraction. In: Formal methods in computer aided design, pp 221–229
Nadel A, Ryvchin V, Strichman O (2013) Efficient MUS extraction with resolution. In: 2013 formal methods in computer-aided design, pp 197–200
Pratissoli F, Brugioni R, Battilani N, Sabattini L (2023) Hierarchical traffic management of multi-AGV systems with deadlock prevention applied to industrial environments. IEEE Trans Autom Sci Eng:1–15
Roselli SF, Bengtsson K, Åkesson K (2018) SMT solvers for job-shop scheduling problems: models comparison and performance evaluation. In: 2018 IEEE 14th international conference on automation science and engineering (CASE), pp 547–552
Roselli SF, Fabian M, Åkesson K (2021) Solving the conflict-free electric vehicle routing problem using SMT solvers. In: 2021 29th mediterranean conference on control and automation (MED), pp 542–547
Roselli SF, Vader R, Fabian M, Åkesson K (2022) Leveraging conflicting constraints in solving vehicle routing problems. IFAC-PapersOnLine 55(28):22–29
Roselli SF, Götvall P-L, Fabian M, Åkesson K (2022) A compositional algorithm for the conflict-free electric vehicle routing problem. IEEE Trans Autom Sci Eng 19(3):1405–1421
Saidi-Mehrabad M, Dehnavi-Arani S, Evazabadian F, Mahmoodian V (2015) An ant colony algorithm (ACA) for solving the new integrated model of job shop scheduling and conflict-free routing of AGVs. Comput Ind Eng 86:2–13
Schneider M, Stenger A, Goeke D (2014) The electric vehicle-routing problem with time windows and recharging stations. Transp Sci 48(4):500–520
Selsam D, Bjørner N (2019) Guiding high-performance SAT solvers with unsat-core predictions. In: International conference on theory and applications of satisfiability testing, pp 336–353
Sinz C (2005) Towards an optimal CNF encoding of boolean cardinality constraints. In: International conference on principles and practice of constraint programming, pp 827–831
Thanos E, Wauters T, Vanden Berghe G (2019) Dispatch and conflict-free routing of capacitated vehicles with storage stack allocation. J Oper Res Soc:1–14
Tinelli C, Harandi M (1996) A new correctness proof of the Nelson-Oppen combination procedure. In: Baader F, Schulz KU (eds) Frontiers of Combining Systems. Springer, Dordrecht, pp 103–119
Trespalacios F, Grossmann IE (2015) Improved big-M reformulation for generalized disjunctive programs. Comput Chem Eng 76:98–103
Yao F, Alkan B, Ahmad B, Harrison R (2020) Improving just-in-time delivery performance of IoT-enabled flexible manufacturing systems with agv based material transportation. Sensors 20(21):6333
Yuan R, Dong T, Li J (2016) Research on the collision-free path planning of multi-AGVs system based on improved A* algorithm. Amer J Oper Res 6(6):442–449
Zhong M, Yang Y, Dessouky Y, Postolache O (2020) Multi-AGV scheduling for conflict-free path planning in automated container terminals. Comput Ind Eng 142:106371



Sabino Francesco Roselli is a PostDoc within the Automation group, at the Department of Electrical Engineering of Chalmers University of Technology, Gothenburg, Sweden. He holds a M.Sc. in Production Engineering from Politecnico di Bari, Italy, and a PhD in Automation from Chalmers University of Technology. His research interests include scheduling and large scale optimization within the context of production systems, specifically fleets of mobile robots.

Martin Fabian is Full Professor in Automation and Head of the Automation Research group at the Department of Electrical Engineering, Chalmers University of Technology. His research interests include formal methods for automation systems in a broad sense, spanning the fields of Control Engineering and Computer Science. He has authored more than 200 publications, and is co-developer of the formal methods tool Supremica, which implements several state-of-the-art algorithms for supervisory control synthesis.

Knut Åkesson is Professor in the Department of Electrical Engineering at Chalmers University of Technology, Gothenburg, Sweden. His main research is in using rigorous methods for analysis of cyber-physical systems. Åkesson holds a M.Sc. in Computer Science and Technology from Lund Institute of Technology, Sweden, and PhD in Control Engineering from Chalmers University of Technology, Gothenburg, Sweden.