Leveraging Conflicting Constraints in Solving Vehicle Routing Problems

N.B. When citing this work, cite the original published paper.

(article starts on next page)

Leveraging Conflicting Constraints in Solving Vehicle Routing Problems ⋆

Sabino Francesco Roselli * **Remco Vader** ** **Martin Fabian** *
Knut Åkesson *

* *Department of Electrical Engineering, Chalmers University of
Technology, Sweden (e-mail:{rsabino, fabian, knut}@chalmers.se).*
** *Department of Mechanical Engineering, Eindhoven University of
Technology, Netherlands (e-mail:r.m.vader@student.tue.nl)*

Abstract: The Conflict-Free Electric Vehicle Routing Problem (CF-EVRP) is a combinatorial optimization problem of designing routes for vehicles to visit customers 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 delivery to the customers, 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 and that solves 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 problems took significant time to solve due to the high number of iterations required to find solutions that satisfy the road segments' capacity constraints. The bottleneck is the *Path Changing Problem*, i.e., the sub-problem of finding a new set of shortest paths to connect a subset of the customers, disregarding previously found shortest paths. This paper presents an improved version of the *PathsChanger* function to solve the *Path Changing Problem* that exploits the *unsatisfiable core*, i.e., information on which constraints conflict, to guide the search for feasible solutions. Experiments show faster convergence to feasible solutions compared to the previous version of *PathsChanger*.

1. INTRODUCTION

We consider scheduling a fleet of mobile robots, in the sequel referred to as Automated Guided Vehicles (AGVs), that pick-up and deliver components to workstations within specified time-windows. The AGVs move on a predefined road network, where each road segment has a maximum number of AGVs it can accommodate at a specific time. The problem is motivated by an industrial need to develop more flexible logistic systems to deliver components just-in-time to an assembly line.

In this scenario, in addition to time-windows in which the components should be delivered, a scheduler needs to consider additional constraints. First, AGVs have a limited operating range and need to recharge their battery when the state-of-charge becomes low. Second, jobs have specific requirements on the AGV eligible to execute them. Finally, the number of AGVs on road segments and workstations are limited to allow low-level trajectory planning problems to be feasible. Thus, we define the *capacity* of the road segments, intersections, and workstations and include *capacity constraints*. A schedule is said to be *conflict-free* if it fulfills the capacity constraints at all times.

The problem of computing conflict-free routes was first introduced in Krishnamurthy et al. (1993) and tackled by means of column generation. In Corréa et al. (2007), conflict-free routing in combination with scheduling of jobs for flexible manufacturing systems is discussed. An ant colony algorithm is applied to the problem of job shop scheduling and conflict free routing of AGVs by Saidi-Mehrabad et al. (2015). In Yuan et al. (2016), a collision-free path planning for multi AGV systems based on the A^* algorithm is presented. Another heuristic approach to solve the conflict-free routing problem with storage allocation is presented by Thanos et al. (2019). In Murakami (2020), a MILP formulation to design conflict-free routes for capacitated vehicles is presented. In Zhong et al. (2020) is presented a hybrid evolutionary algorithm to deal with conflict-free AGV scheduling in automated container terminals, and Chen et al. (2021) handles the problem of conflict-free routing of AGVs by a meta-heuristic improvement strategy based on large neighbourhood search. Hence, conflict-free routing and scheduling has been addressed previously, but to the best of our knowledge, there is no work in the literature that tackles all above mentioned constraints at once. Therefore, Roselli et al. (2021) introduced the *Conflict-Free Electric Vehicle Routing Problem* (CF-EVRP). The CF-EVRP is an extension of the vehicle routing problem (VRP) Dantzig and Ramser (1959), involving the additional constraints. In Roselli et al. (2022) a compositional algorithm, ComSat, for solving the

CF-EVRP is proposed. ComSat breaks down CF-EVRP into sub-problems and iteratively solves these 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 computes routes to serve the customers, and assigns vehicles to the routes attempting to make the execution of the system conflict-free. In a plant there can be several ways to travel from one customer's location to another. Initially, ComSat uses the shortest paths among the customers' locations when designing the routes. However, if a feasible schedule cannot be achieved using the shortest paths, alternative paths have to be found, which is handled by the *Conflict-free Paths Search* (CFPS). CFPS is composed of two main functions; the *PathsChanger* function, that finds alternative sets of paths if the current schedule violates the capacity constraints, and the *CapacityVerifier* function, that checks whether the schedule is conflict-free or not.

Experiments show that when a solution computed using the shortest paths violates the capacity constraints, finding alternative paths using the *PathsChanger* function may require multiple iterations. This does not come unexpected, since the number of possible paths in a graph can be high, and minimizing the cumulative length while looking for alternative paths does not guarantee that the schedule will be conflict-free. In this paper we focus on the CFPS and present improved versions of the *PathsChanger* and *CapacityVerifier* that, in many cases, find feasible solutions faster.

The sub-problems in ComSat are modelled as Satisfiability Modulo Theory (SMT) problems Barrett et al. (2009); De Moura and Bjørner (2011), as SMT solvers have shown to be efficient in solving combinatorial problems Weber et al. (2019).

Moreover, some SMT solvers come with algorithms that allow them to deal with optimization problems Sebastiani and Trentin (2020). Two sub-problems in ComSat, marked by the round boxes in Fig. 1 (see below) are optimization problems.

For the CFPS polynomial time algorithms exist to find paths in graphs, Gross and Yellen (2003). However, modelling the *Path Changing Problem* as an SMT problem is beneficial as it allows 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).

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 *Path Changing Problem*, to increase the chances of finding a feasible schedule.

The contributions in this paper are: (i) exploitation of SMT solvers' *MUC* to extract information about the infeasibility of an SMT formula representing a conflicting schedule for a VRP; (ii) use of such information to find conflict-free schedules; (iii) performance comparison between the unguided and *MUC* guided paths search over a set of CF-EVRP problem instances.

The remainder of the paper is organized as follows. Preliminaries are presented in Section 2. Section 3 presents the mathematical models of the sub-problems that form the CFPS and how it is improved using the *MUC* from the *Capacity Verification Problem*. Proof of soundness and completeness of the procedure is provided in Section 4. In Section 5, the results of the analysis over a set of problem instances are presented. Finally, conclusions are drawn in Section 6.

2. 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 intersections between road segments or customers' locations. A customer 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 serve the customer. 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 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.
- *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 $e \in \mathcal{E}$.
  $d_e \in \mathbb{R}_+$: 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 of time when all jobs have ended, assuming they start at time 0.
  T: the time horizon.
- *Customer*: Entity representing a task to be executed by a vehicle, e.g., a pickup or delivery of material, that needs to be visited exactly once by the vehicle. A customer is always associated with a node where the pickup/delivery operation is executed, and has a time window indicating the earliest and latest time at which it can be visited. Unless explicitly given, the time window is the entire time span $[0, T]$.
  Let $\mathcal{K}$ be the finite set of all customers, and let $l_k, u_k \in \mathbb{R}_+$, $k \in \mathcal{K}$ be the time window's lower (l_k) and upper (u_k) bound for customer k such that $u_k > l_k$.
  Also let $s_k \in \mathbb{R}_+$ and $L_k \in \mathcal{N}$, for $k \in \mathcal{K}$, be the service time and location of customer k, respectively.

- *Route*: an ordered set of unique customers.
$$r_j = \langle k_{j1}, \ldots, k_{jm} \rangle, \ m \le |\mathcal{K}|, \ k_{ji} \in \mathcal{K},$$
$$i = 1, \ldots, m, \ k_{jl} \ne k_{ji} \text{ for } i \ne l.$$
A route can at most include all customers, therefore $m \le |\mathcal{K}|$.

- *Route set*: a set of routes such that each customer belongs to exactly one route, thus guaranteeing that all customers are served.
$$\mathcal{R} = \{r_1, \ldots, r_m\}, \ m \le |\mathcal{K}|$$
A route contains at least one customer, hence $m \le |\mathcal{K}|$.

- *Route start*: the starting time τ_r of route r, computed by the function *Assign*. Γ is the set that contains the *route start* of each route.
$$\Gamma = \{\tau_r \in \mathbb{R} \,|\, r \in \mathcal{R}\}$$

- *Pair Set of route r*: set containing the sequence of customers 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 customers of routes, since each pair of customers in a route is connected by a path.
$$\theta_p = \langle n_1, \ldots, n_m \rangle, \ p \in \mathcal{P}_r, \ m \le |\mathcal{N}|,$$
$$n_i \in \mathcal{N}, \ i = 1, \ldots, m$$

- *Edge sequence*: ordered set of unique edges for a given path θ_p.
$$\delta_p = \langle e_1, \ldots, e_m \rangle, \ p \in \mathcal{P}_r, \ m = |\theta_p| - 1,$$
$$e_i \in \mathcal{E}, \ i = 1, \ldots, m$$

In order to clarify which part of ComSat is analyzed and improved in this work, let us recap briefly how the algorithm works. Fig. 1 shows a simplified flowchart of ComSat that illustrate the concepts of this paper. The first step of ComSat is to design a set of routes $\mathcal{R}$ to serve all the customers; at this point, the shortest path between any two customers is computed using Dijkstra's algorithm Dijkstra (1959).

This optimization problem is handled by the function *Router* and must guarantee that the routes meet specific requirements such as maximum length, specific ordering among the customers and time windows. If this step is infeasible the CF-EVRP instance has no solution and the algorithm terminates. If this step is feasible, the function *Assign* will try to allocate available vehicles to the routes and compute a start time $\tau_r, \ \forall r \in \mathcal{R}$, to the routes. If this step is infeasible then *Router* will try to find different routes, but if it is feasible, the *CapacityVerifier* checks if the current set of routes is conflict-free. More details on the functions *Router* and *Assign* can be found in Roselli et al. (2022).

2.1 The minimal Unsat Core

For infeasible problems, there can be identified a subset of the constraints that conflict, meaning they cannot all simultaneously be satisfied. 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*.
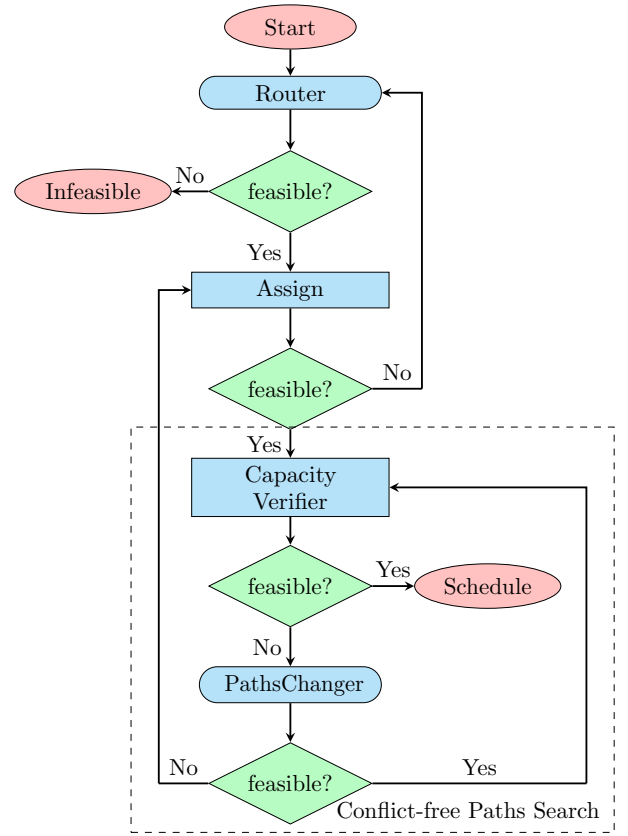

Fig. 1. Flowchart of ComSat.

Formally, given an SMT formula φ and set of conflicting constraints $\mathcal{C} \subseteq \varphi$, $\mathcal{C}$ is a *MUC* of φ if removing any constraint $\mathcal{C}_i \in \mathcal{C}$ makes $\mathcal{C} \backslash \mathcal{C}_i$ no longer infeasible; removing $\mathcal{C}$ 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 though; 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*.

3. 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), 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 model formulation for the *Path Changing Problem* is inspired by Aloul et al. (2006).

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

    $EN(A, n)$: exactly n variables in the set A are true;
    $If(c, o_1, o_2)$: if c is *true* returns o_1, else returns o_2.

We will write $EN_{m \in M}(m, n)$ to denote $EN(\bigcup_{m \in M}\{m\}, n)$ in order to shorten the notation.

3.1 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 of the 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 *PathsChanger*.

Let n_{rpe} be the node visited before edge e of pair p of route r, and let e_{rpn} be the node visited before node n on pair p of route r. Similarly, let n^{rpe} be the node visited after edge e of pair p of route r, and let e^{rpn} be the edge visited after node n on pair p route r. Let p_r^0 be the first pair of route r and n_r^* be its starting node.

Example of Routes, Pairs, Nodes, and Edges
Let $\mathcal{K} = \{k_1, \ldots, k_7\}$ and $\mathcal{N} = \{n_1, \ldots, n_{20}\}$. Let $L_{k_1} = n_1$ and $L_{k_2} = n_7$, and assume two routes designed to serve all customers: $r_1 = \langle k_1, k_2, k_5, k_7 \rangle$, $r_2 = \langle k_3, k_4, k_6 \rangle$.

In order to clarify the notation introduced above, let us analyze r_1. First, the set of pairs for r_1 is defined as $\mathcal{P}_{r_1} = \{\langle k_1, k_2 \rangle, \langle k_2, k_5 \rangle, \langle k_5, k_7 \rangle\}$.
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 $p = \langle k_1, k_2 \rangle$; then for $e = \langle n_1, n_2 \rangle$, $n_{r_1 p e} = n_1$, and $n^{r_1 p e} = n_2$; for $n = n_1$, $e^{r_1 p n} = \langle n_1, n_2 \rangle$, and for $n = n_2$, $e_{r_1 p n} = \langle n_1, n_2 \rangle$.

For each node it must also be specified whether there exists a time window, since some of the nodes are only intersections of road segments in the real plant, while others are actual customers. Let l_{rpn} and u_{rpn} be the earliest and latest arrival time, respectively, at node n of pair p of route r; let s_{rpn} be the service time at node n of pair p of route r. Finally, let $\gamma > 0$ be a small real constant used to prevent *swapping* of vehicles' positions between a node and the previous or following edge.

The *Capacity Verification Problem* decision variables are:

    x_{rpn}: non-negative real variable that models when a vehicle executing route r starts using node n in pair p;
    y_{rpe}: non-negative real variable that models when a vehicle executing route r starts using edge e in pair p;

The model for the *Capacity Verification Problem* is:

$$x_{rp_r^0 n_r^*} \geq \tau_r, \ \forall r \in \mathcal{R} \tag{1}$$

$$y_{rpe} \geq x_{rpn_{rpe}} + s_{rpn_{rpe}}, \ \forall r \in \mathcal{R}, \ p \in \mathcal{P}_r, \ e \in \delta_p \tag{2}$$

$$x_{rpn} = y_{rpe_{rpn}} + d_{e_{rpn}}, \ \forall r \in \mathcal{R}, \ p \in \mathcal{P}_r, \ n \in \theta_p \tag{3}$$

$$x_{rpn} \geq l_{rpn} \wedge x_{rpn} \leq u_{rpn},$$
$$\forall r \in \mathcal{R}, \ p \in \mathcal{P}_r, \ n \in \theta_p \tag{4}$$

$$x_{r_1 p_1 n} \geq y_{r_2 p_2 e^{r_1 p_1 n}} + \gamma \ \vee \ x_{r_2 p_2 n} \geq y_{r_1 p_1 e^{r_2 p_2 n}} + \gamma,$$
$$\forall r_1, r_2 \in \mathcal{R}, \ r_1 \neq r_2, \ p_1 \in \mathcal{P}_{r_1}, \ p_2 \in \mathcal{P}_{r_2}$$
$$n \in \theta_{p1} \cap \theta_{p2}, \ n \notin \mathcal{N}_H \tag{5}$$

$$y_{r_1 p_1 e} \geq y_{r_2 p_2 e} + \gamma \vee y_{r_2 p_2 e} \geq y_{r_1 p_1 e} + \gamma,$$
$$\forall r_1, r_2 \in \mathcal{R}, \ r_1 \neq r_2, \ p_1 \in \mathcal{P}_{r_1}, \ p_2 \in \mathcal{P}_{r_2},$$
$$e \in \delta_{p_1} \cap \delta_{p_2} \tag{6}$$

$$y_{r_1 p_1 e_1} \geq y_{r_2 p_2 e_2} + d_{e_2} \ \vee \ y_{r_2 p_2 e_2} \geq y_{r_1 p_2 e_1} + d_{e_1},$$
$$\forall r_1, r_2 \in \mathcal{R}, \ r_1 \neq r_2, \ p_1 \in \mathcal{P}_{r_1}, \ p_2 \in \mathcal{P}_{r_2},$$
$$e_1 \in \delta_{p_1}, \ e_2 \in \delta_{p_2}, \ e_1 = \bar{e}_2, \ g_{e_1} = g_{e_2} = 1 \tag{7}$$

(1) constrains the start time of a route; (2) and (3) define the precedence among nodes and edges to visit in a route; (4) enforces time windows on the nodes that correspond to the customers; (5) prevents vehicles from using the same node at the same time; (6) and (7) 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 γ 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 *CapacityVerifier* (CV) is defined, that takes a set of routes $\mathcal{R}$, the start times in Γ, and the current set of paths CP as input and returns:

- *CFS*, a list that expresses where each vehicle is at each time; this is empty if the problem is infeasible.
- $\bar{\mathcal{C}}$, the *Unsat Core* relative to constraints (5)-(7) (see Section 3.3); this is empty if the problem is feasible.

3.2 Paths Changing Problem

In the *Paths Changing Problem*, alternative paths are computed to connect the consecutive customers of each route. Finding alternative paths may be necessary when, for a given set of routes $\mathcal{R}$ and starting times Γ, no feasible schedule exists. The *Capacity Verification Problem* may be infeasible due to the current set of paths that connect the customers' locations, therefore a different set may lead to a feasible solution. A route is defined as a sequence of customers, and for any two consecutive customers there is a path (a sequence of edges) connecting them. Therefore, for a route containing $i + 1$ customers 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.

Decision variables used to build the model 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 $i + 1$ customers) 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 optimal solution to the *Path Changing Problem* found at iteration h be

$$CP = \bigcup_{\substack{r \in \mathcal{R} \\ p \in \mathcal{P}_r \\ e \in \mathcal{E}}} \{z_{rpe}^*\},$$

where z_{rpe}^* is the value of z_{rpe} in the current solution; also, let PP be the set containing the optimal solutions found until the $(h-1)$-th iteration. The model is then:

$$\min_{r \in \mathcal{R}, \; p \in \mathcal{P}_r, \; n \in \mathcal{E}} \sum \text{If}(z_{rpe}, d_e, 0) \tag{8}$$

$$w_{rp\xi_p} \wedge w_{ri\pi_p}, \qquad \forall p \in \mathcal{P}_r, \; r \in \mathcal{R} \tag{9}$$

$$\text{EN}_{e \in \mathcal{O}_{\xi_p}}(z_{rpe}, 1), \qquad \forall p \in \mathcal{P}_r, \; r \in \mathcal{R} \tag{10}$$

$$\text{EN}_{e \in \mathcal{I}_{\xi_p}}(z_{rpe}, 1), \qquad \forall p \in \mathcal{P}_r, \; r \in \mathcal{R} \tag{11}$$

$$z_{rpe} \implies \neg z_{rp\bar{e}}, \qquad \forall p \in \mathcal{P}_r, \; r \in \mathcal{R}, \; e \in \mathcal{E} \tag{12}$$

$$\bigwedge_{n \in \mathcal{N}, n \neq \xi_p, n \neq \pi_p} \text{If}(w_{rpn},$$
$$\text{EN}_{e \in \mathcal{O}_n}(z_{rpe}, 1) \wedge \text{EN}_{e \in \mathcal{I}_n}(z_{rpe}, 1),$$
$$\text{EN}_{e \in \mathcal{O}_n}(z_{rpe}, 0) \wedge \text{EN}_{e \in \mathcal{I}_n}(z_{rpe}, 0)),$$
$$\forall p \in \mathcal{P}_r, \; r \in \mathcal{R} \tag{13}$$

$$\bigvee_{z_{rpe} \in CP} \neg z_{rpe}, \qquad \forall CP \in PP \tag{14}$$

The cost function (8) to minimize is the cumulative length of the used edges; (9) guarantees that, for each path of each route, the start and end nodes are used; (10) and (11) make sure that exactly one outgoing (incoming) edge is incident with the start (end) node of a route; (12) makes sure that a path is not allowed to use both an edge and its reverse; (13) 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; finally, (14) rules out all the previously found solutions.

Based on the model described above the function *Paths-Changer* (*PC*) is defined, that takes the previous paths PP as input and returns a new set of paths NP. If the *Paths Changing Problem* is infeasible then $NP = \emptyset$.

Up to this point, unless specified otherwise, the models presented are taken from Roselli et al. (2022).

3.3 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 it searches blindly through the possible paths that connect any two customers, while minimizing the paths' cumulative length. A *conflict-free* solution may involve paths that are quite longer than the current ones though, and the *PC* will have to explore many *shorter* solutions before finding the right one.

Improving the performance of the *PathsChanger* would be beneficial for the overall performance of ComSat, and letting the *MUC* guide the paths changing is such an improvement.

When extracting the *MUC*, it is possible to only 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 (1)-(4). 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 (1)-(7). Assume that φ_0 is infeasible, and let $\bar{\mathcal{C}}_0$ be the subset of a *MUC* retrieved by tracking constraints (5)-(7). 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 (5)-(7), 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 (5)-(7) is defined over two routes r_1 and r_2 and their pairs p_1 and p_2 for a specific node n or edge e; therefore, if the constraint is part of $\bar{\mathcal{C}}$, the routes and pairs that caused the conflict over n or e can be identified. If the conflict was generated by a constraint from (5), then the following constraint is added to (8)-(14):

$$\neg(w_{r_1 p_1 n}) \vee \neg(w_{r_2 p_2 n}). \tag{15}$$

On the other hand, if the conflict was caused by constraint from (6) or (7), the following constraint is added to (8)-(14):

$$\neg(z_{r_1 p_1 e}) \vee \neg(z_{r_2 p_2 e}). \tag{16}$$

Constraints (15) and (16) force at least one of the routes involved in the conflict to avoid the specific node (edge, respectively) when computing a path for the pairs involved in the conflict. 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.

Based on the model described by (8)-(16), 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 the *Path Changing Problem* is infeasible $NP = \emptyset$.

Since for each constraint in $\bar{\mathcal{C}}$ a new constraint is added to the GPC, 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 (15) and (16) in the GPC that may remove feasible solutions.

Fig. 2 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 Γ have already been computed. The shortest paths between any two customers are computed using Dijkstra's algorithm and then set as the current paths CP to travel among customers. Also, CP are added to the list of previous paths PP.
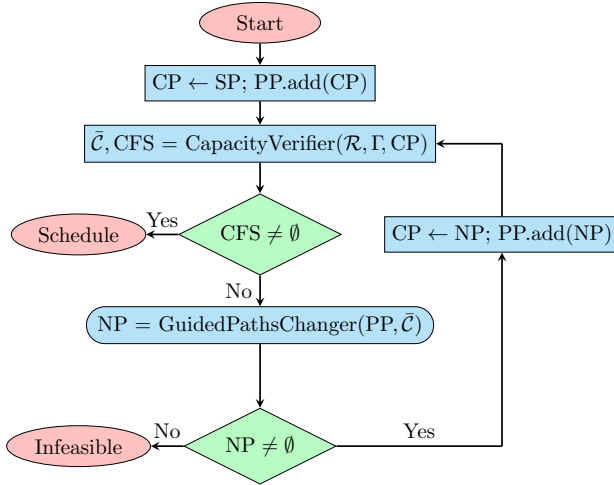

Fig. 2. Flowchart of the MUC-Guided-CFPS.

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 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 $\mathcal{R}$ with the start times Γ.

4. PROOF OF SOUNDNESS AND COMPLETENESS

In this section, proof of soundness and completeness of the *Unsat Core* Guided CFPS is provided. The underlying idea for the proof is the following. There exists a finite number of solutions to the *Path Changing Problem*; the GPC can enumerate at least all feasible solutions to the *Path 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 *Path Changing Problem*; let us 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 *Path 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 this case, even if the GPC is not able to find all feasible solutions $\mathcal{F}$, there is none to find.

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 *Path Changing Problem*, whether it belongs to $\mathcal{F}$ or not, until all solutions are enumerated. However in the GPC there are additional constraints that may remove feasible solutions. In the proof it is shown that such additional constraints only remove infeasible solutions.

Observation 1. The *Path Changing Problem* is a satisfiability problem in propositional logic. The *Capacity Verification Problem* falls into the category of difference logic (a fragment of linear arithmetic). Thus, both problems are decidable.

Observation 2. The *Path Changing Problem* is bounded. In fact, the *Path Changing Problem* involves only a finite number of Boolean variables, so its domain is finite.

Lemma 1. Given a finite, directed, weighted graph, the number of paths that connect two arbitrary nodes is finite.

Proof 1. 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 2. For a given set of routes $\mathcal{R}$ and start times in Γ, repeated calls to the PC function will enumerate all feasible solutions to the *Path Changing Problem*, either belonging to $\mathcal{F}$ or $\mathcal{U}$, before returning infeasible.

Proof 2. Let φ_0 be the conjunction of constraints (9)-(13), 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* 1, 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 3. Using the PC and CV is a sound and complete procedure to solve the CFPS

Proof 3. Because of Observation 1 we know there is a finite number of solutions to the *Path Changing Problem*, and because of Lemma 2 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 4. For a given set of routes $\mathcal{R}$, the GPC is able to find at least all solutions in $\mathcal{F}$.

Proof 4. For each set of current paths CP, $\bar{\mathcal{C}}$ only contains constraints defined by (5), (6), and (7). The constraints in $\bar{\mathcal{C}}$ are iteratively retrieved from *minimal Unsat Core* and therefore represent combinations of nodes and edges in the graph where the conflicts happen. Since each constraint defined by (15) and (16) addresses one constraint from $\bar{\mathcal{C}}$, (15) and (16) only define constraints over nodes or edges that cause conflicts. Hence these constraints only remove solutions of the *Path Changing Problem* that belong to $\mathcal{U}$.

Theorem 1. Using the GPC and CV is a sound and complete procedure to solve the CFPS.

Proof 5. The PC and the GPC are identical, except for constraints (15)-(16), and because of Lemma 4, 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 3), so is the CFPS using the GPC.

5. EXPERIMENTS

In order to evaluate the goodness of the proposed method and its performance against the previous version of the CFPS algorithm, a set of problem instances is designed and used for testing. Both the PC and GPC are embedded in the ComSat algorithm. However, since the goal is to compare the search for alternative paths, problems are designed in such a way that there is only one feasible set of routes $\mathcal{R}$ to serve the customers; also, only the running time for search of conflict-free paths is measured. The algorithms called by ComSat used the SMT solver Z3 4.8.9 (single core mode) to solve the models. All the experiments[1] were performed on an *Intel Core i7 6700K, 4.0 GHZ, 32GB RAM* running *Ubuntu-18.04 LTS*.

Table 1 shows the results of the evaluation of five problem instances of the CF-EVRP solved using ComSat. Each instance was solved twice, once using the PC and once using the GPC; 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 customers in each route. The customers' locations and time windows so that conflicts will arise due to the capacity constraint when the shortest paths are used and a search for alternative paths will be necessary in order to find a conflict-free schedule.

For instances *1* through *4* it took only one iteration of the GPC to find a feasible solution, while the PC required an increasing number of iterations to find a feasible solution, as the instances grew more complicated. The gap in the running time between the GPC and the PC follows the same trend; for instance *1* it only takes 2 iterations to the PC to find a feasible solution, while it takes 24 and 54 iterations to find a solution to instances *2* and *3*. This number drops to 15 iterations for instance *4*. On average, a single iteration of the PC takes less time than an iteration of the GPC, but due to the larger number of iterations

required, the overall running time for the PC is always larger.

Instance *5* is the odd one out, as it only takes one iteration of the PC to find a feasible solution, and, as for the other instances, the running time for the single iteration is shorter.

Results and Discussion

The experiments show that for most of the instances the GPC performed better than PC in terms of running time and number of iterations. To be more specific, one iteration of the GPC is slower than one iteration of the PC, but the number of iterations required by the PC is always higher, and therefore the overall execution time is longer. As the instances become larger, the gap between the running time for one iteration of each method increases too. However, since the number of iterations required for more complex instances grows as well, the GPC shows increasing good performance for harder-to-solve instances. On the other hand, Instance *5* shows a different result, since both the PC and the GPC take only one iteration. As for the other instances, a single iteration of the PC is faster, hence the PC beats the GPC on Instance *5*. We can conclude that for some instances, the PC may be able to quickly find feasible solutions and outperform the GPC. However this is behaviour is highly dependent on the instance and as instances grow larger the chances could grow smaller, as the number of possible paths available increases. Moreover, a detailed analysis of the solutions to the *Path Changing Problem* for each instance[2] confirms that, for the PC, there is no convergence to a feasible solution as the number of iterations increases, since the number of conflicts does not always decrease at the following iteration. On the other hand, the GPC shows a consistent behaviour as it always takes only one iteration to find feasible solutions.

Table 1. Comparison of the PC and GPC over a set of instances of the CF-EVRP. For each instance the number of iterations and the total running time (in seconds) required to find a feasible solution is reported.

Inst.	$|\mathcal{N}|$	$|\mathcal{R}|$	$|\mathcal{K}|$	Iterations		Time	
				PC	GPC	PC	GPC
1	3	2	4	2	1	0.25	**0.16**
2	8	3	6	24	1	8.81	**0.40**
3	5	4	8	54	1	35.92	**1.08**
4	64	4	28	15	1	643.40	**184.60**
5	64	4	28	1	1	**21.20**	128.40

6. CONCLUSIONS

This paper presents an algorithm to search for conflict-free paths for a set of routes to serve customers in a conflict-free electric vehicle routing problem (CF-EVRP). The algorithm exploits the SMT solvers' ability to return a

---

[1] The implementation of the GPC presented in Section 3.3 and the problem instances are available in the *UNSAT_Core* folder at `https://github.com/sabinoroselli/VRP.git`.

[2] Details of the problem instances are discussed in the file *Instances_Results.pdf* in the *UNSAT_Core* folder of the Github repository.

minimal unsat core (*MUC*) when a formula is infeasible, to guide the search for paths. Soundness and completeness of the algorithm are proved, and preliminary experimental data based on a set of generated CF-EVRP problem instances are provided. The experiments show that the new *MUC* based algorithm consistently finds feasible paths taking only one iteration and significantly shorter time than the previous naive method. Future work includes to run extensive computational analyses to strengthen the claims made in this paper, and further development of the *MUC* guided paths search by improving the information extraction from the *MUC*.

REFERENCES

Aloul, F.A., Al Rawi, B., and Aboelaze, M. (2006). Identifying the shortest path in large networks using boolean satisfiability. In *2006 3rd International Conference on Electrical and Electronics Engineering*, 1–4. IEEE.

Barrett, C.W., Sebastiani, R., Seshia, S.A., Tinelli, C., et al. (2009). Satisfiability modulo theories. *Handbook of satisfiability*, 185, 825–885.

Bjørner, N., Phan, A.D., and Fleckenstein, L. (2015). νz-an optimizing SMT solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 194–199. Springer.

Chen, Z., Alonso-Mora, J., Bai, X., Harabor, D.D., and Stuckey, P.J. (2021). Integrated task assignment and path planning for capacitated multi-agent pickup and delivery. *IEEE Robotics and Automation Letters*, 6(3), 5816–5823.

Cimatti, A., Griggio, A., and Sebastiani, R. (2011). Computing small unsatisfiable cores in satisfiability modulo theories. *Journal of Artificial Intelligence Research*, 40, 701–728.

Corréa, A.I., Langevin, A., and Rousseau, L.M. (2007). Scheduling and routing of automated guided vehicles: A hybrid approach. *Computers & operations research*, 34(6), 1688–1707.

Dantzig, G.B. and Ramser, J.H. (1959). The truck dispatching problem. *Management science*, 6(1), 80–91.

De Moura, L. and Bjørner, N. (2011). Satisfiability modulo theories: Introduction and applications. *Commun. ACM*, 54(9), 69–77. doi:10.1145/1995376.1995394. URL http://doi.acm.org/10.1145/1995376.1995394.

Dershowitz, N., Hanna, Z., and Nadel, A. (2006). A scalable algorithm for minimal unsatisfiable core extraction. In *International Conference on Theory and Applications of Satisfiability Testing*, 36–41. Springer.

Dijkstra, E.W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1), 269–271.

Gross, J.L. and Yellen, J. (2003). *Handbook of graph theory*. CRC press.

Huang, J. (2005). MUP: A minimal unsatisfiability prover. In *Proceedings of the ASP-DAC 2005. Asia and South Pacific Design Automation Conference, 2005.*, volume 1, 432–437. IEEE.

Krishnamurthy, N.N., Batta, R., and Karwan, M.H. (1993). Developing conflict-free routes for automated guided vehicles. *Operations Research*, 41(6), 1077–1090.

Kroening, D. and Strichman, O. (2016). *Decision procedures*. Springer.

Murakami, K. (2020). Time-space network model and MILP formulation of the conflict-free routing problem of a capacitated AGV system. *Computers & Industrial Engineering*, 141, 106270.

Nadel, A. (2010). Boosting minimal unsatisfiable core extraction. In *Formal Methods in Computer Aided Design*, 221–229. IEEE.

Nadel, A., Ryvchin, V., and Strichman, O. (2013). Efficient MUS extraction with resolution. In *2013 Formal Methods in Computer-Aided Design*, 197–200. IEEE.

Roselli, S., Fabian, M., and Åkesson, K. (2022). A compositional Algorithm to solve the Conflict-Free Electric Vehicle Routing Problem. *2022 IEEE Transactions on Automation Science and Engineering. Submitted for Publication.* Available on arXiv.org.

Roselli, S.F., Bengtsson, K., and 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)*, 547–552. IEEE.

Roselli, S.F., Fabian, M., and Åkesson, K. (2021). Solving the conflict-free electric vehicle routing problem using SMT solvers. In *2021 29th Mediterranean Conference on Control and Automation (MED)*, 542–547. IEEE.

Saidi-Mehrabad, M., Dehnavi-Arani, S., Evazabadian, F., and Mahmoodian, V. (2015). An ant colony algorithm (ACA) for solving the new integrated model of job shop scheduling and conflict-free routing of AGVs. *Computers & Industrial Engineering*, 86, 2–13.

Sebastiani, R. and Trentin, P. (2020). OptiMathSAT: A tool for optimization modulo theories. *Journal of Automated Reasoning*, 64(3), 423–460.

Selsam, D. and Bjørner, N. (2019). Guiding high-performance SAT solvers with unsat-core predictions. In *International Conference on Theory and Applications of Satisfiability Testing*, 336–353. Springer.

Sinz, C. (2005). Towards an optimal CNF encoding of boolean cardinality constraints. In *International conference on principles and practice of constraint programming*, 827–831. Springer.

Thanos, E., Wauters, T., and Vanden Berghe, G. (2019). Dispatch and conflict-free routing of capacitated vehicles with storage stack allocation. *Journal of the Operational Research Society*, 1–14.

Weber, T., Conchon, S., Déharbe, D., Heizmann, M., Niemetz, A., and Reger, G. (2019). The SMT competition 2015–2018. *Journal on Satisfiability, Boolean Modeling and Computation*, 11(1), 221–259.

Yuan, R., Dong, T., and Li, J. (2016). Research on the collision-free path planning of multi-AGVs system based on improved A* algorithm. *American Journal of Operations Research*, 6(6), 442–449.

Zhong, M., Yang, Y., Dessouky, Y., and Postolache, O. (2020). Multi-AGV scheduling for conflict-free path planning in automated container terminals. *Computers & Industrial Engineering*, 142, 106371.