An SMT Based Compositional Algorithm to Solve a Conflict-Free Electric Vehicle Routing Problem

Sabino Francesco Roselli¹ and Martin Fabian¹ and Knut Åkesson¹

Abstract— The Vehicle Routing Problem (VRP) is the combinatorial optimization problem of designing routes for vehicles to visit customers in such a fashion that a cost function, typically the number of vehicles, or the total travelled distance is minimized. The problem finds applications in industrial scenarios, for example where Automated Guided Vehicles run through the plant to deliver components from the warehouse. This specific problem, henceforth called the Electric Conflict-Free Vehicle Routing Problem (CF-EVRP), involves constraints such as limited operating range of the vehicles, time windows on the delivery to the customers, and limited capacity on the number of vehicles the road segments can accommodate at the same time. Such a complex system results in a large model that cannot easily be solved to optimality in reasonable time. We therefore developed a compositional algorithm that breaks down the problem into smaller and simpler sub-problems and provides sub-optimal, feasible solutions to the original problem. The algorithm exploits the strengths of SMT solvers, which proved in our previous work to be an efficient approach to deal with scheduling problems. Compared to a monolithic model for the CF-EVRP, written in the SMT standard language and solving a state-of-the-art SMT solver the compositional algorithm was found to be significantly faster.

I. INTRODUCTION

The use of Automated Guided Vehicles (AGVs) for just-in-time deliveries is becoming common in modern manufacturing facilities [1]. Adapting this solution, rather than storing all the components by the assembly line, makes the environment more worker-friendly and using AGVs instead of fixed transportation belts (or similar) makes it more flexible [2].

A system like this requires a Scheduler that guarantees that the deadlines for the delivery of components are met, but also that AGVs do not create queues by the workstations by arriving too early. Also, AGVs are battery-powered so their operating range is limited. Therefore they need to recharge occasionally, and this must be taken into account when designing the schedule. Finally, though AGVs are usually equipped with low-level controllers to avoid dangerous conditions, they may not be able to avoid deadlocks, i.e. getting stuck due to circular waiting between them. Hence the scheduler must ensure such situations are avoided.

This type of problem can be modelled as a Vehicle Routing Problem (VRP) [3], the combinatorial optimization problem of designing routes for vehicles to visit customers, such that a cost function is optimized. There exist extensions of the VRP that involve additional constraints, such as time windows for the customers’ service (VRP with time windows, or VRPTW [4]), limited operating range of the vehicles and possibility to recharge at the charging stations (Electric VRP, or E-VRP [5]), and limitations on the capacity of the road segments that vehicles drive on (dispatch and conflict-free routing problem (DCFRP) [6]). The problem we are tackling in this work, henceforth called the Electric Conflict-Free Vehicle Routing Problem (CF-EVRP), involves all these features and also additional constraints related to the customers’ service.

For relatively small size problem instances, mixed integer linear programming solvers (MILP, [7]) can provide good solutions to VRPs in a short time. However, for larger problems, MILP solvers are often not fast enough and specific-purpose algorithms involving local search [8], column generation [9] or stochastic methods [10], [11] are used instead. Recent work focusing on fleets of electric vehicles [12], as well as conflict-free routing [13] show applications of such approaches to real-world problems.

In our previous work [14], we presented a monolithic formulation (MonoMod) to model the CF-EVRP; based on our previous findings from [15], we decided to formulate the model in SMT standard Language (Satisfiability Modulo Theory, [16], [17]) and solve it using the state-of-the-art SMT solver Z3 [18]; as expected, our approach was not able to solve large problem instances in reasonable time. In fact, though Z3 has proven very efficient in solving combinatorial optimization problems [19], our formulation quickly leads to a state-space explosion as the number of vehicles and jobs, and the time horizon (a fixed point of time in the future when certain processes will be evaluated or assumed to end) for the jobs’ execution increases. This is mainly due to the necessity of discretizing time in order to keep track of the vehicle’s locations and avoid collisions. The compositional algorithm (ComSat) we present in this work breaks down the CF-EVRP into sub-problems so that time discretization can be avoided and a feasible solution can be reached quickly. The algorithm exploits the strengths of the SMT solvers by reducing the problem to smaller Job Shop Problems [20] (JSPs). It first selects a set of paths to uniquely connect any two customers, since in a real plant there can exists multiple paths; it then solves a VRP to design routes to serve all customers within their time windows; if such a solution exists, it matches some of the available vehicles with the generated routes; finally, if this phase is also successful, it checks the current solution against the capacity constraints on the road segments (in terms of number of vehicles that can travel through them at the same time). Whenever one sub-problem turns out to be infeasible, the algorithm backtracks and finds a different solution for the previous phase that will hopefully lead to a feasible solution for the current phase. It terminates when the last phase is feasible or all the combinations have been checked (in which case it declares the problem infeasible).

The contributions of this paper are: (i) Designing an algorithm for the CF-EVRP that can quickly provide feasible solutions; (ii) Compare the performance of ComSat against MonoMod, presented in [14] over a set of generate instances of the CF-EVRP.

We gratefully acknowledge financial support from Chalmers AI Research Centre (ChAIr), AB Volvo (Project ViMcOr), the support from Per-Lage Götavall at Volvo Group Truck Operation, and the Wallenberg AI, Autonomous Systems and Software program (WASP) funded by the Knut and Alice Wallenberg Foundation. ¹Department Electrical Engineering, Chalmers University of Technology, Göteborg, Sweden {rsabino, fabian, knut}@chalmers.se

2021 IEEE 17th International Conference on Automation Science and Engineering (CASE) August 23-27, 2021. Lyon, France
In the following, Section II provides a formal description of the problem, and Section III introduces the algorithm together with a mathematical model and detailed descriptions of its constraints. Finally, conclusions are drawn in Section IV. The extended version of the paper is available at https://arxiv.org/abs/2106.07387

II. PROBLEM FORMULATION

In the CF-EVRP the plant layout is represented by a strongly connected, weighted, directed graph, where road segments are represented by the edges and the intersections between them are represented by the nodes. Customers are located at nodes and they are defined by a name (typically a number) and time window, i.e., a lower and an upper bound that represents the earliest and latest arrival allowed to serve the customer. Edges have two attributes: the road segment’s length, and its capacity in terms of number of vehicles that can simultaneously travel through it.

The following definitions are provided:

- **Task:** either a pickup or a delivery operation. A task is always associated with a node (see below) where the task is executed. Each task has an associated duration, and the vehicle can execute the task. If a task is not executed by any vehicle, it is considered to be zero, as these times can be considered negligible compared to the travelling time.
- **Job:** one or more pickup tasks and one delivery task. Pickup may have precedence constraints among them, all pickup for that job are completed.
- **Vehicle:** a transporter, e.g., an AGV, that is able to move between locations in the plant and perform pickup and delivery operations.
- **Node:** a location in the plant. A node can only accommodate one vehicle at the time unless otherwise specified.
- **Depot:** a node at which one or more vehicles start and must return to after completing the assigned jobs. The depot can be defined to accommodate an arbitrary number of vehicles at the same time.
- **Edge:** a road segment that connects two nodes.

The requirements of the problem are summarized as follows:

- All jobs have to be completed; for a job to be completed, a vehicle has to be assigned to it and visit the locations of the job’s tasks within their respective time windows.
- When a vehicle is assigned to more than one job, it has to execute the job’s tasks sequentially; all tasks of a job must be completed before it can execute any task of another job.
- Vehicles are powered by batteries with limited capacity but with the ability to recharge at a charging station. It is assumed that charging and discharging of the batteries is linear with respect to distance.
- There is only one depot, which is also the charging station; vehicles have to return to the depot they were dispatched from.
- Different road segments in the plant have different capacities in terms of number of vehicles they can accommodate.
- Pickup and delivery duration (the time when the pickup and delivery tasks respectively are executed) are considered to be zero, as these times can be considered negligible compared to the travelling time.
- Not all vehicles are eligible to execute all jobs.
- It is assumed that one unit of distance is covered in one time-step, hence distance and duration are interchangeable when talking about vehicles’ movements.
- For the algorithm to work, two additional jobs, both having only one task located at the depot, are added: start and end; they are needed in the routing problem to make sure that routes begin and end at the depot.

III. THE COMPOSITIONAL ALGORITHM

In this section we are going to introduce the sub-problems that form ComSat and are iteratively solved to find a feasible solution to the CF-EVRP. In the implementation, each sub-problem is formulated using the SMT standard language and solved using Z3. Therefore, for each sub-problem, a set of variables is declared and set as a set of constraints over those variables; the solver takes variables and constraints and returns an assignment for each variable such that no constraint is broken.

The following logical operators are used to express cardinality constraints [21] in the sub-problems:

- $\text{EO}(a)$: exactly one variable of the set $a$ is true;
- $\text{EN}(a, n)$: exactly $n$ variables of the set $a$ are true;
- $\text{If}(c, a_1, a_2)$: if $c$ is true returns $a_1$, else returns $a_2$.

The algorithm begins with a pre-processing phase where a number of paths to connect any pair of customers is computed. In theory we would like to compute all possible paths for each pair of customers, but in practice this is not efficient because there can be too many. Also, many of these paths are not going to be used, since we are interested in the shortest ones to connect customers.

The first sub-problem aims at selecting one path for each pair of customers among all the paths computed in the pre-processing phase. This is an optimization problem whose goal is to find one path for each pair of customers such that the cumulative length of all paths is minimal. If we had computed too many paths in the pre-processing phase, this phase would be rather slow, since there can easily be millions of possible paths to connect any two customers, even for a rather small graph, and each of them is a variable for this optimization problem.
Once we have only one path between any pair of customers, we can treat the problem as a VRPTW, with some additional constraints on the order in which we serve customers. Therefore we can set up an optimization problem to find the smallest number of routes to serve the customers within their time windows. If we cannot find a feasible set of routes, we need to go back one step and change the paths we are currently using to connect the customers.

In the next phase, we check whether we can match the routes we computed with the available vehicles. A route may involve more than one job, and since not all vehicles are eligible to execute one route, we have to find a vehicle that can execute all jobs involved in a route. Also, routes have a latest starting time that depends on the time windows of their jobs, so we have to make sure that by that time, the vehicle selected has enough charge to finish the route. If we cannot find a feasible match, we re-run the routing problem and find another set of routes that combines the jobs differently.

Finally, if a match between vehicles and routes can be found, the last sub-problem ensures that there is no conflict among the vehicles while they execute the routes, i.e., there is never a larger number of vehicles on a location of the graph (node or edge) than the number of vehicles allowed (based on the location’s capacity).

A. The Path Finder

In this phase the goal is to find a sequence of edges connecting any tasks’ locations such the overall sum of distances of each path is minimized. Let \( Q \) be the set of all pairs of task’s locations. For any pair of points \( q_i \in Q \), all possible non-cyclic paths are computed and then stored in a list \( f_i \). The paths are then stored in the list \( Paths: p_i \rightarrow f_i \).

In order to find the sequence of paths, it is possible to set up an optimization problem that uses the following decision variables:

- \( path_{qr} \): Booleans that are True if the \( r\)-th path of the \( q\)-th pair of points is selected.

The model is as follows:

\[
EO_{r \in f_q} (path_{qr}) \quad \forall q \in Q \tag{1}
\]

\[
\sum \text{If}(path_{qr}, |r| > 0) \quad \forall q \in Q, r \in f_q \tag{2}
\]

Constraint (1) ensures that only one path per pair can be selected; (2) is the cost function that minimizes the overall length of the paths (in terms of nodes to visit), where \( |r| \) is the length of a path \( r \).

Since the solution found may not be feasible for the following steps of the algorithm, it is necessary to store it so that it can be ruled out in the next iteration. Let \( S_{path} = \bigcup_{q \in Q} \{path_{qr}\} \) be the optimal solution to the problem; also, let \( UsedPaths \) be a list containing all the previous solutions. In order to find another feasible solution, the following constraint must be added to the model:

\[
\bigvee_{path_{qr} \in S_{path}} \neg path_{qr} \quad \forall S_{path} \in UsedPaths \tag{3}
\]

Based on the model described above, it is possible to define the function \( pathfinder \) that takes the list \( Paths \) of all non-cyclic paths between any two points and the list \( UP \) that contains the previously used paths as inputs, and returns \( CP \), the shortest feasible combination of non-cyclic paths that has not been selected yet (\( CP \) is empty if the problem is unfeasible).

B. The Routing Problem

The goal is now to find feasible routes using the non-cyclic paths currently provided by the \( pathfinder \) function to calculate the distance \( d_{j_1,k_1,j_2,k_2} \) between tasks’ locations. Also, let \( M_j \) be the set of mutually exclusive jobs for job \( j \) (i.e. the same vehicles cannot execute job \( j \) and any of the jobs in \( M_j \) due to requirements on the vehicle type); let \( Perm_j \) be the set of possible orderings of tasks belonging to job \( j \), where each possible ordering \( ord_{m} \in Perm_j \) contains all tasks of job \( j \) sorted differently. The set of variables used to build the model for the routing problem are:

- \( dir_{j_1,k_1,j_2,k_2} \): Boolean variable that is true if a vehicle travels from the location of task \( k_1 \) of job \( j_1 \) to the location of task \( k_2 \) of job \( j_2 \)
- \( cs_{jk} \): integer variable that tracks the arrival time of a vehicle at the location of task \( k \) of job \( j \)
- \( rc_{jk} \): integer variable that tracks the remaining charge of a vehicle when at the location of task \( k \) of job \( j \)

The model is as follows:

\[
(dir_{j_1,k_1,j_2,k_2}) \quad \forall j_1,j_2 \in J, k_1 \in K_{j_1}, k_2 \in K_{j_2} \tag{4}
\]

\[
(dir_{j_1,k_1,k_2}) \quad \forall j_1 \in J, k_1 \in K_{j_1}\tag{5}
\]

\[
(dir_{j_1,k_2}) \quad \forall j_1 \in J, k_2 \in K_{j_1}\tag{6}
\]

\[
(dir_{j_1,k_1,j_2,k_2}) \quad \forall j_1,j_2 \in J, k_1 \in K_{j_1}, k_2 \in K_{j_2} \tag{7}
\]

\[
(cs_{jk} \geq 0 \land rc_{jk} \geq 0 \land rc_{jk} \leq OR) \quad \forall j \in J, k \in K_j \tag{8}
\]

\[
(-dir_{j_1,k_1,k_2}) \quad \forall j \in J, k \in K_j \tag{9}
\]

\[
(rc_{jk} \geq l_j \land cs_{jk} \leq u_j) \quad \forall j \in J, k \in K_j \tag{10}
\]

\[
(dir_{j_1,k_1,j_2,k_2}) \quad \forall j_1 \in J, k_1 \in K_{j_1}, k_2 \in K_{j_2} \tag{11}
\]

\[
(EO_{j_1 \in J, j_1 \notin j_2, k_1 \in K_{j_1}} \quad \forall j \in J, k \in K_j \tag{12}
\]

\[
(EN_{j_1 \in J, j_1 \notin j_2, k_1 \in K_{j_1}} \quad \forall j \in J, k \in K_j \tag{13}
\]

\[
(EN_{j_2 \in J, j_2 \notin j_1, k_2 \in K_{j_2}} \quad \forall j \in J, k \in K_j \tag{14}
\]

(4) restricts the variables to be positive integers and the remaining charge to be lower than the maximum operating range; (5) forbids to travel from and to the same location; (6) and (7) state that a vehicle can never travel to the start, nor travel from the end: \( start \) and \( end \) are physically located at the same node, but they play different roles in the routing problem, hence two different jobs; (8) regulates the difference in the arrival time based on the distance for a direct travel between two points; (9) enforces the time windows on the routes; (10) defines the decrease of charge based on the distance travelled; (11) states that each customer must be visited exactly once; (12) guarantees the flow conservation between start and end; (13) states that if a number of tasks belongs to one job, they have to take place in sequence; (14) guarantees that deliveries take place after pickups.

Finally, the cost function for the model to minimize (15)
is the total number of vehicles:

\[ \sum \text{if}(d_{\text{dir}j,k_{\text{start}j,k} \cdot 1,0}) \quad \forall j \in J, k \in K_j \quad (15) \]

If the solution of the routing problem turns out to be inconsistent with the vehicles’ assignment or the conflict-free constraints in the next two phases, a new solution must be computed in order to find alternative routes for the same combination of non-cyclic paths. Therefore it is necessary to keep track of the combinations of routes that have already been generated so that we can rule them out when solving the routing problem again. Let \( \text{Routes} = \bigcup_{j \in J} \{ \text{es}_{jk} \} \) be the optimal solution to the routing problem found at iteration \( n \) and \( PR \) the set containing the optimal solutions found until the \((n-1)\)-th iteration. In order to find the optimal set of routes, different from the ones found before, the following constraint must be added:

\[ \bigvee_{d_{\text{dir}j,k_{1}j_{2}k_{2}} \in \text{Routes}} \neg d_{\text{dir}j_{1}k_{1}j_{2}k_{2}} \forall \text{Routes} \in PR \quad (16) \]

Based on the model described above, it is possible to define the function \( \text{router} \) that takes the current combination of non-cyclic paths \( CP \) and the set \( PR \), and returns a set of routes that have not been selected yet \( CR \) (if the problem is unsat, \( CR \) is empty).

C. The Assignment Problem

The assignment problem allocates vehicles to the routes \( CR \) generated in the routing problem, based on the time window and eligibility requirements. In the previous phase routes were generated based only on the time windows and on the vehicles’ operating range; now the actual availability of each type of vehicle is given. Moreover, the \( \text{router} \) may generate routes that involve mutually exclusive jobs and, while it would be possible to avoid this by adding additional constraints, it would be inconvenient to do in the routing problem, since there is no information about the vehicles assigned to the routes. On the other hand, once a set of routes is given, it can be easily checked in the assignment problem whether a vehicle is actually eligible for a route.

Therefore, for each route \( r \), we can define a list of jobs \( J' \subseteq J \) that are executed by the vehicle assigned to \( r \), and the list of eligible vehicles for \( r \) \( \text{El}_{r} = \bigcap_{j \in J'} \forall j \). Also, based on the time windows of the jobs forming the routes, it is possible to work out the latest start of a route \( \text{late}_r \); for instance, if the time window’s upper bound for the delivery task of a job is at time \( t \) and the distance between the task’s location and the depot is \( d \), then the latest start is \( t - d \) (remember that we assume time and distance travelled to be interchangeable). Since a route can include more than one job, the stricter deadline will define the latest start for the route.

The assignment problem is therefore treated as a JSP where routes are jobs (whose duration depends on their length \( \text{length}_r \)) and vehicles are resources, with some additional requirements on the jobs starting time. The set of variables used to build the model are:

\( \text{allo}_i \): Boolean variable that is True if vehicle \( i \) is assigned to route \( r \), False otherwise;
\( \text{start}_r \): A Non-negative integer variable that is the start time of route \( r \);
\( \text{end}_r \): Non-negative integer variables that keep the end time of route \( r \).

The model formulation for the assignment problem is as follows:

\[ \text{end}_r = \text{start}_r + \text{length}_r \quad \forall r \in R \quad (17) \]
\[ \text{start}_r \leq \text{late}_r \quad \forall r \in R \quad (18) \]
\[ \text{EO}_{i \in V}(\text{allo}_i) \quad \forall r \in R \quad (19) \]
\[ \bigvee_{i \in \text{El}_r} \text{allo}_i \quad \forall r \in R \quad (20) \]

(17) connects the start and end variables based on the route’s length; (18) sets the latest start time of a route based on the stricter time window among the ones of its jobs; (19) states that exactly one vehicle must be assigned to a route; (20) states that one (or more) among the eligible vehicles must be assigned to a route; (21) states that any two routes assigned to the same vehicle cannot overlap in time; either one ends before the other starts or the other way around.

Based on the (17)-(21) it is possible to define the function \( \text{assign} \) that takes the routes \( CR \) from the routing problem as input and returns \( \text{AS} \), which states which vehicle will drive on which route (and, of course, execute its jobs) and when it starts (if the assignment problem is infeasible \( \text{AS} \) will be empty).

D. The Scheduling Problem

Finally, in this phase a feasible schedule is found for the vehicles, meaning that the routes they are assigned to are checked to verify that they are conflict-free. In order to do this, we generate a list of nodes that each route visits \( AN_r \), \( \forall r \in \text{Routes} \), and one list of edges \( AE \), \( \forall r \in \text{Routes} \), since so far the focus was only on the tasks’ locations. Note that for the same route \( r \), \( AE \) will always be one element shorter than \( AN_r \) since there is an edge in between two nodes: this way the first edge in \( AE \) is always the edge following the first node in \( AN_r \), the second edge in \( AE \) is always the edge following the second node in \( AN_r \), and so on. Also, for each node in \( AN_r \) it is necessary to specify whether there exist a time window, since some of the nodes are only intersection of road segments in the real plant, while others are actual pickup or delivery points. Let \( t_{\text{len}} \) and \( t_{\text{ur}} \) be the earliest and latest arrival time at node \( n \) on route \( r \) respectively. Finally, let \( \text{len}(e) \) be the length to travel of edge \( e \) and let \( \epsilon(1) \) and \( \epsilon(2) \) be the source and sink node of the edge respectively (since it is a directed graph, the edge connecting \( n \) and \( n' \) is different from the one connecting \( n' \) and \( n \)).

This phase is also treated as a JSP, where jobs are routes, tasks within the jobs are the different nodes and edges to visit along the route while nodes and edges themselves are the resources. Also, each route has a starting time \( \text{start} \), defined by the assignment model. The decision variables in the scheduling problem are:

\( \text{node}_{rn} \): Non-negative integer variables to keep track of when route \( r \) is using node \( n \);
\( \text{edgere}_{rn} \): Non-negative integer variables to keep track of when route \( r \) is using edge \( e \);
The model for the scheduling problem is defined as follows:

\[ \text{node}_{rO} \geq \text{start}_{r} \quad \forall r \in \text{Routes} \quad (22) \]

\[ \text{edge}_{ri} \geq \text{node}_{ri} \quad \forall r \in \text{Routes}, i = 1, \ldots, |AE_{r}| \quad (23) \]

\[ \text{node}_{ri+1} \geq \text{edge}_{ri} + \text{len}(e) \quad \forall r \in \text{Routes}, i = 1, \ldots, |AE_{r}| \quad (24) \]

\[ (\text{node}_{ri} \geq l_{ri} \land \text{node}_{ri} \leq u_{ri}) \quad \forall r \in \text{Routes}, i \in \text{AN}_{r} \quad (25) \]

\[ (\text{node}_{ri} \geq \text{edge}_{ri+1} + 1 \lor \text{edge}_{ri+1} \geq \text{node}_{ri} + 1) \quad \forall r \in \text{Routes}, i = 1, 2, i \in AE_{r1} \cap AE_{r2} \quad (26) \]

\[ (\text{edge}_{ri+1} \geq \text{edge}_{ri2} + \text{len}(e_{2}) \lor \text{edge}_{ri2} \geq \text{edge}_{ri+1} + \text{len}(e_{1})) \quad \forall r, i = 1, 2, i \in AE_{r1} \cap AE_{r2}, \quad r \neq r_{1}, e_{1}(1) = e_{2}(2), e_{1}(2) = e_{2}(1) \quad (27) \]

(22) constrains the beginning time of a route; (23) and (24) define the precedence among nodes and edges to visit in a route; (25) enforces time windows on the nodes that correspond to pickup or delivery points; (26) prevents vehicles for using the same node at the same time (the +1 in the constraints forbids swapping of positions between a node and the previous or following edge) (27) and (28) constrains the transit of vehicles over the same edge. If two vehicles are crossing the same edge from the same node, one has to start at least one time step later than the other and if two vehicles are traversing the same edge from opposite nodes, one has to be done transiting, before the next one can start.

Based on the (22)-(28) it is possible to define the function scheduler that takes the AS from the assignment problem as input and returns SC, a list that expresses where each vehicle is at each time step (and, as for the previous phases, is empty if the problem is unfeasible).

E. The Algorithm

In this section, the compositional algorithm to solve the CF-EVRP is presented. The input for the algorithm is a problem instance, consisting of a weighted, directed graph representing the plant layout and a list of jobs. The function path_enumerator takes the graph and the jobs as input and, using Dijkstra’s algorithm [22] returns the previously mentioned list Paths.

The output of the algorithm is twofold: the variable solution, which is initialized as unknown and will possibly become either sat or unsat, and the schedule, which contains the information about the location of each vehicle at each time step if the problem is sat or is empty otherwise.

First, the algorithm finds a feasible combination of non-cyclic paths to connect any two tasks’s locations: this is done through the function pathfinder; if no combination of paths can be found (either because there are none or because all feasible solutions have been used already), the algorithm terminates and the solution is unsat. If a path list can be found, then such solution is added to the UP and it is used as an input to generate feasible routes, if such exist; if no solution exists to the routing problem, there are two possible outcomes depending on the list PR:

- PR is empty: the algorithm terminates and returns unsat;
- PR is non-empty: a new combination of non-cyclic paths is computed.

The condition on the emptiness of PR can save time based on one assumption: every time a new combination of non-cyclic paths is computed, it is the shortest still available. If no routing is possible, i.e. time windows could not be met with the current paths, there is no other combination of paths that will satisfy the routing problem, since they will be longer than the current one. On the other hand, if PR is not empty, this means that routing is possible with the current combination of non-cyclic paths and it would be premature to declare the instance unsat. Instead, the list PR is emptied, since it only makes sense to store the old routes as long as the combination of non-cyclic paths is the same. If a solution to the routing problem does exist, the current routes CR are added to PR and then checked against the assignment problem and the scheduling problem. If one of these problems turns out to be unfeasible, then the function router will look for another solution; otherwise, when both assign and scheduler return feasible solutions a feasible, sub-optimal schedule for the overall problem has been found. The flow chart in Fig. 1 shows graphically how the different sub-problems interact with each other.

IV. Conclusion

In this paper we have presented a compositional algorithm to solve the Conflict-Free Electric Vehicle Routing Problem (CF-EVRP). We have evaluated the performance of the algorithm in handling problem instances of the CF-EVRP and we have compared it with a monolithic model we presented.
in our previous work. The implementation of the model presented in Section II, as well as the problem instances and the experimental results are available at https://github.com/sabinoroselli/VRP.git. The algorithm proved to be significantly faster than the monolithic model to solve problem instances that are inherently satisfiable, while its performance was rather slow for unsatisfiable instances. For this reason we set up a termination criterion based on the number of iterations so that, if the algorithm cannot prove the problem either satisfiable or unsatisfiable within a certain number of iterations, it declares it unknown. We are currently working on figuring out the right conditions for the algorithm to spot unsatisfiability quicker. Also, as of today, we generate a limited number of paths to connect each pair of customers for the sub-problem Path Finder to select one path for each pair of customers and proceed to the next sub-problems. However, in order to correctly declare a problem instance unsatisfiable, the algorithm should check all possible combinations of paths. Since the number of possible paths between two nodes can increase exponentially with the graph size, finding all paths between all pairs of nodes would be untractable. Even if finding all paths could be done instantly, having too many paths to choose from would make the Path Finder the bottle-neck of the algorithm. Therefore we are working on an algorithm to provide the next shortest combination of paths given the current state, without enumerating all of them.

REFERENCES

[1] K Azadeh, M. deKoster, and D Roy, “Robotized warehouse systems: Developments and research opportunities,” ERIM Report Series Research in Management, no. ERS-2017-009-LIS, 2017.
[2] J. Theunissen, H. Xu, R. Y. Zhong, and X. Xu, “Smart AGV system for manufacturing shopfloor in the context of industry 4.0,” in 2018 25th International Conference on Mechatronics and Machine Vision in Practice (M2VIP), IEEE, 2018, pp. 1–6.
[3] K. Braekers, K. Ramaekers, and I. Van Nieuwenhuyse, “The vehicle routing problem: State of the art classification and review,” Computers & Industrial Engineering, vol. 99, pp. 300–313, 2016.
[4] M. Desrochers, J. Desrosiers, and M. Solomon, “A new optimization algorithm for the vehicle routing problem with time windows,” Operations research, vol. 40, no. 2, pp. 342–354, 1992.
[5] M. Schneider, A. Stenger, and D. Goekle, “The electric vehicle-routing problem with time windows and recharging stations,” Transportation Science, vol. 48, no. 4, pp. 500–520, 2014.
[6] N. N. Krishnamurthy, R. Batta, and M. H. Karwan, “Developing conflict-free routes for automated guided vehicles,” Operations Research, vol. 41, no. 6, pp. 1077–1090, 1993.
[7] N. Brahimi and T. Aouam, “Multi-item production routing problem with backordering: A MILP approach,” International Journal of Production Research, vol. 54, no. 4, pp. 1076–1093, 2016.
[8] O. Bräysy and M. Gendreau, “Vehicle routing problem with time windows, part I: Route construction and local search algorithms,” Transportation science, vol. 39, no. 1, pp. 104–118, 2005.
[9] S. Riazi, T. Diding, P. Falkman, K. Bengtsson, and B. Lenartson, “Scheduling and routing of AGVs for large-scale flexible manufacturing systems,” in 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE), IEEE, 2019, pp. 891–896.
[10] B. M. Baker and M. Ayechew, “A genetic algorithm for the vehicle routing problem,” Computers & Operations Research, vol. 30, no. 5, pp. 787–800, 2003.
[11] Y.-J. Gong, J. Zhang, O. Liu, R.-Z. Huang, H. S.-H. Chung, and Y.-H. Shi, “Optimizing the vehicle routing problem with time windows: A discrete particle swarm optimization approach,” IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 42, no. 2, pp. 254–267, 2011.
[12] F. Rossi, R. Iglésias, M. Alizadeh, and M. Pavone, “On the interaction between autonomous mobility-on-demand systems and the power network: Models and coordination algorithms,” IEEE Trans. on Control of Network Systems, vol. 7, no. 1, pp. 384–397, 2019.
[13] E. Thanos, T. Wauters, and G. Vanden Berghe, “Dispatch and conflict-free routing of capacitated vehicles with storage stack allocation,” Journal of the Operational Research Society, pp. 1–14, 2019.
[14] S. Roselli, M. Fabian, and K. Åkesson, “Solving the Electric-Conflict-Free-Vehicle Routing Problem Using SMT Solvers,” MED 2021, The 29th Mediterranean Conference on Control and Automation, 2021.
[15] S. F. Roselli, K. Bengtsson, and K. Åkesson, “SMT solvers for job-shop scheduling problems: Models comparison and performance evaluation,” in 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE), IEEE, 2018, pp. 547–552.
[16] C. W. Barrett, R. Sebastiani, S. A. Seshia, C. Tinelli, et al., “Satisfiability modulo theories,” Handbook of satisfiability, vol. 185, pp. 825–885, 2009.
[17] L. De Moura and N. Bjørner, “Satisfiability modulo theories: Introduction and applications,” Commun. ACM, vol. 54, no. 9, pp. 69–77, Sep. 2011.
[18] L. De Moura and N. Bjørner, “Z3: An efficient SMT solver,” in International conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2008, pp. 337–340.
[19] N. Bjørner, A.-D. Phan, and L. Fleckenstein, “ν-Z: an optimizing SMT solver,” in International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2015, pp. 194–199.
[20] J. Carlier and ´E. Pinson, “An algorithm for solving the job-shop problem,” Management science, vol. 35, no. 2, pp. 164–176, 1989.
[21] C. Sinz, “Towards an optimal CNF encoding of boolean cardinality constraints,” in International conference on principles and practice of constraint programming, Springer, 2005, pp. 827–831.
[22] E. W. Dijkstra et al., “A note on two problems in connexion with graphs,” Numerische mathematik, vol. 1, no. 1, pp. 269–271, 1959.