Submitted to *Transportation Science*
manuscript TS-2016-0199.R3

Path and Speed Optimization for Conflict-Free Pickup and Delivery under Time Windows

Tommaso Adamo
Department of Engineering for Innovation, Università del Salento, Via Monteroni 73100 Lecce, Italy, tommaso.adamo@unisalento.it

Tolga Bektaş
Centre for Operational Research, Management Science and Information Systems (CORMSIS)
Southampton Business School, University of Southampton, Southampton, Highfield SO17 1BJ, United Kingdom, T.Bektas@soton.ac.uk

Gianpaolo Ghiani
Department of Engineering for Innovation, Università del Salento, Via Monteroni 73100 Lecce, Italy, gianpaolo.ghiani@unisalento.it

Emanuela Guerriero
Department of Engineering for Innovation, Università del Salento, Via Monteroni 73100 Lecce, Italy, emanuela.guerriero@unisalento.it

Emanuele Manni
Department of Engineering for Innovation, Università del Salento, Via Monteroni 73100 Lecce, Italy, emanuele.manni@unisalento.it

This article introduces a variant of the Conflict-Free Pickup and Delivery Problem with Time Windows in which speeds can be regulated. The problem arises in several areas of transportation and logistics including routing and scheduling of automated guided vehicles in port terminals and coordination of unmanned aerial vehicles in controlled airspace. A particular aspect of this problem is that at most one vehicle can traverse an arc of the transportation network at any time. The problem studied in this paper is to determine the vehicle paths and speeds on each arc of the path in such a way that no conflicts arise, the time windows are met and the total energy consumption is minimized. A branch-and-bound algorithm is described in which a lower bound is obtained by solving a separable nonlinear problem in quadratic time. If the solution of the relaxation is not conflict-free, a set of space-based and time-based branching constraints are generated to resolve the detected conflicts. Computational experiments show that, when compared with a state-of-the-art approach, the proposed method is able to generate a larger number of feasible solutions (42% on average) and reduce the computation time by an order of magnitude. Moreover, the approach results in an average energy saving of around 70%.

Key words: pickup and delivery; speed optimization; conflict-free routing

1. Introduction

Conflict-free routing problems mainly arise in material handling systems where automated guided vehicles (AGVs) are used to move loads between pickup and delivery points. In such systems, vehicles travel on a network made up of arcs called *tracks* or *track-segments*, which in turn form paths between the points. When each track or a track-segment has a unit capacity, it allows the traversal of only a single vehicle at a time. In this case, if two or more vehicles try to enter a track at the same time, or they are scheduled to arrive at a given point at the same time, a conflict arises. In this case, at least one of the vehicles has to be delayed or re-routed. One practical setting in which the problem arises is in the traffic control of AGVs in warehouses with very narrow aisles (VNAs). VNA is a type of layout used for improving the utilization of the storage space, where it suffices to leave as little as a couple inches of clearance on each side of the AGV for it to travel safely in a narrow aisle. Similar problems also arise in routing and scheduling of automated guided vehicles in port terminals and coordination of unmanned aerial vehicles in controlled airspace.

If the arc capacity constraints are relaxed and the speeds are fixed, the problem becomes that of routing with Pickup and Delivery (Dumas et al. 1991, Berbeglia et al. 2007). A variety of conflict-free routing and dispatching problems have been studied in the literature, which generally involve assigning a set of requests and their sequence to each vehicle (*dispatching*), as well determining the paths that they will traverse (*routing*) such that they are conflict-free. These problems are typically formulated and solved in two-stages. In the first stage, a master problem determines a solution to the dispatching problem, in which the arc capacity constraints are relaxed, where the objective is often the minimization of the longest route or the minimization of the production delays. The solution found at this level, however, does not specify the sequence of the network arcs to be traversed by the vehicle while going from a pickup point to the next one. A second-stage problem is subsequently solved to check whether a set of conflict-free routes, satisfying the constraints of the master problem, exists. This subproblem has so far been solved with the assumption that the vehicle speed is constant.

We now summarize the literature related to the Conflict-Free Pickup and Delivery Problem with Time Windows. Earlier work on the problem includes that of Krishnamurthy et al. (1993), who solve the conflict-free routing problem given that a solution to the dispatching problem has already been determined. The authors describe a set partitioning formulation for the problem, the variables of which are the routes for each AGV, and solve it using a column generation based heuristic. Desaulniers et al. (2003) extend the problem to take into account the assignment of requests to vehicles, and propose an exact algorithm in which column generation is used to solve the relaxation at each node of a branch-and-cut search tree. Later studies on the problem have adopted the two-stage solution algorithm mentioned above. In particular, Corréa et al. (2007) propose a

decomposition framework where the master problem is formulated as a constraint programming model and the second-stage problem as a mixed integer programming model. If infeasible, a pool of logic cuts is identified and added to the master problem to prune solutions that include conflicts. This particular decomposition suits the problem well in that: (i) the constraint programming model can easily handle nonlinear constraints, e.g., disjunctions on timing decisions, and provides very good bounds for the problem, and (ii) the second-stage problem has a minimum cost flow structure, which can be solved efficiently using the network simplex method within a branch-and-bound algorithm. The problem studied by Nishi et al. (2011) involves an additional layer of production scheduling decisions, and is formulated as a mixed integer programming model. The master problem, which models the production scheduling and request assignment to the vehicles, is solved by Lagrangian relaxation and yields a lower bound for the original problem. A subproblem checks the existence of a conflict-free routing of the vehicles, which can either be solved using constraint programming or, as implemented by the authors, by a distributed optimization algorithm. The algorithm stops if the solution is conflict-free. Otherwise, a feasible solution to the problem is constructed to obtain an upper bound, and cuts are generated to prevent any previously identified conflicts from appearing again before solving the master problem. One reason behind the choice of the suggested decomposition is the substantial reduction of the number of integer variables in the subproblem when they are fixed in the master problem. Other studies on the problem include that by Nishi and Tanaka (2012) who propose a Petri network decomposition algorithm for the problem, where the choice of this method is primarily motivated by the need to solve large-scale instances, for which reason the authors exploit the decomposability of Petri nets. Finally, Saidi-Mehrabad et al. (2015) describe a similar two-stage partitioning of the problem into a master problem and a second-stage problem, where both problems are solved by an ant colony algorithm.

As the brief review above indicates, the Conflict-Free Pickup and Delivery Problem with Time Windows has been examined in some detail under the assumption that vehicle speed is fixed and cannot be regulated. However, the literature on energy-efficient and conflict-free routes is much less developed. This is quite surprising since the optimization of the usage of batteries is of primary concern in AGV-based systems (Onori et al. 2016). Energy consumption and emissions have been addressed in van Duin and Geerlings (2011) at a strategic level. Xin et al. (2013) investigate how to improve the performance at the operational level, when combining objectives related to throughput and energy consumption. The authors describe a speed optimization problem arising in an automated container terminal, formulated as a quadratic programming model, but does not consider the potential collisions of vehicles.

In this paper, we study a conflict-free pickup and delivery problem in which a set of pickup and delivery requests have to be transported between specified pairs of nodes within specified time

windows by using a homogeneous fleet of vehicles. Time windows are particularly relevant when the pickup and delivery requests arise as part of a larger distribution problem, where parts to pick up might not necessarily be readily available at the beginning of the planning horizon and where the delivery times may be constrained by the subsequent shipment of the parts.

We assume that requests are known in advance, travel times are deterministic and decisions pertaining to vehicle assignment and sequencing (i.e., vehicle dispatching) have already been made. The problem we address here is to determine the paths to be traversed by each vehicle between two consecutive pickup and delivery points, and the speed at which a vehicle travels on each track, such that there are no conflicts in the resulting solution, and that the overall energy usage of the vehicles is minimized. One way to avoid conflicts is to operate the vehicles at constant (e.g., maximum) speed and to insert delays by allowing the vehicles to wait at the end of tracks. We pursue an alternative approach and provide the vehicles with the ability to change their speeds, which, to the best of our knowledge, has not yet been considered in the relevant literature. The main advantage of our approach is that it can reduce the overall energy consumption, for which speed is known to be one of the main determinants. Regulating the vehicle speeds also avoids unnecessary waiting at nodes that may block other vehicles and eliminates potential infeasibilities. It may seem that combining travel at maximum speed with delays allows for the construction of solutions equivalent to those with variable speeds. If waiting was allowed along the arcs, then the two approaches would lead to the same set of feasible solutions. We adopt a design assumption here similar to that of Nandula and Dutta (2000) and Farahvash and Boucher (2004), in which the network has crossover or intersection points, which are the only locations where an AGV is allowed to wait to access a link in case of a conflict. Restricting waiting at nodes is sometimes necessitated for technological reasons, in particular to be able to accurately track an AGV. The position of an AGV can be determined precisely, only when the vehicle is in aligned with one of a discrete number of so-called artificial landmarks. Otherwise, the position is estimated. It common practice to select the waiting locations as a subset of the landmarks (Ronzoni et al. 2011). Whilst the landmark placement in itself is a difficult problem, it has been suggested that placing landmarks at intersection points in a grid provides very good, if not optimal, solutions, mainly due to triangulation effects (see Sinriech and Shoval 2000, for further details).

Our model assumes that the choice of the speed on an arc is completely independent of the speed in the preceding and the succeeding arc on the path. We also assume that the time spent accelerating or decelerating is negligible and that a turn does not slow down the vehicle significantly. This is realistic in port terminal management and in routing Unmanned Aerial Vehicles where the track length is large enough to justify both assumptions: tens of meters or even a few hundred meters in the former case (Branch 2012) and several miles in the latter (Mueller et al. 2013). As

far as energy consumption is concerned, modern AGVs, whilst decelerating, use a regenerative breaking mechanism that can be used to recover the kinetic energy consumed in the acceleration phase (Young et al. 2013, Fiori et al. 2016).

We contribute to the literature by (1) introducing and formally defining the problem mentioned above, (2) describing a lower bounding procedure and an exact algorithm based on branch-and-bound, and (3) quantifying the benefits of using speed optimization for reducing energy requirements through computational experiments.

The rest of the paper is organized as follows. Section 2 presents a formal definition of the problem studied here along with an illustrative example. The branch-and-bound algorithm is described in detail in Section 3. Section 4 reports the results of computational experimentation. Conclusions are presented in Section 5.

2. Problem Statement

Let $G = (V, E)$ be a directed graph, where V is a vertex set and E is the set of arcs. In this representation, an arc corresponds to a track-segment of a path in the graph, and the vertices represent endpoints of track-segments. Each arc $(i, j) \in E$ represents the movement of a vehicle $i \in V$ to an adjacent node $j \in V$, and its length is denoted by c_{ij}. Each path is bi-directional, meaning that, for a pair of adjacent vertices $i \in V$ and $j \in V$, there exists an arc $(i, j) \in E$ and another one $(j, i) \in E$ in the opposite direction. For convenience, we will assume that all track-segments have the same length, that is $c_{ij} = 1$ for all $(i, j) \in E$, although the ensuing exposition and the algorithm presented apply to networks with non-uniform length track-segments.

Transportation tasks are associated to a subset of vertices $V_1 \subseteq V$. A set $R = \{r = (a_r, b_r) : a_r \in V_1, b_r \in V_1\}$ of unit loads have to be transported between pairs of workstations by a homogeneous fleet of K vehicles of unit capacity. As in Sarker and Gurav (2005), we assume that there exists a *buffer* at each node in V_1, which is an area off a lane where a vehicle may enter to perform a pickup or delivery, thereby allowing another vehicle to traverse the same lane at the same time. Vehicles may all be based at the same depot $i_0 \in V$, as is assumed in the following, or at different depots, as would be in the case of Material Handling Systems. A time window is associated with each transportation request $r \in R$, relating to pickup or delivery. A vehicle can travel at a speed $v \in (0, \overline{v}]$, and the energy required per distance unit is described by a non-decreasing convex function $f(v)$. As stated in Section 1, the model assumes that the choice of the speed on an arc is completely independent of the speed in the preceding and succeeding arc.

Given a planning horizon $[t_0, T]$, we assume that the dispatching problem has been solved and is therefore an input to our problem. The task dispatching prescribes a sequence S_k consisting of H_k pickup/delivery locations to be visited, defined for each vehicle $k = 1, \ldots, K$. The node in position

$h \in \{1, \ldots, H_k\}$ on a sequence S_k of vehicle $k \in \{1, \ldots, K\}$ is shown by $S_k[h]$, where $S_k[h] \in V_1$, for which $\alpha_{kh} = [\underline{\alpha}_{kh}, \overline{\alpha}_{kh}]$ is the time window, with $\underline{\alpha}_{kh}$ denoting the earliest and $\overline{\alpha}_{kh}$ the latest possible times for service to commence at that node. The set of all time windows α_{kh} assigned to a vehicle $k = 1, \ldots, K$ is denoted by $\mathcal{C}_k$.

As far as vehicle routing is concerned, each track has a capacity equal to one. There are two types of possible conflicts: the first, namely *Node-conflict*, arises when two or more vehicles traveling on different arcs are moving towards the same node, such that they all arrive at the destined node at the same time. The second, namely *Arc-conflict*, arises when one vehicle attempts to enter a track-segment that is already occupied by another vehicle, regardless of direction. In the following, we define ϵ as the smallest significant time duration, which is used for discretization of time, and using which the conflict definitions will be defined.

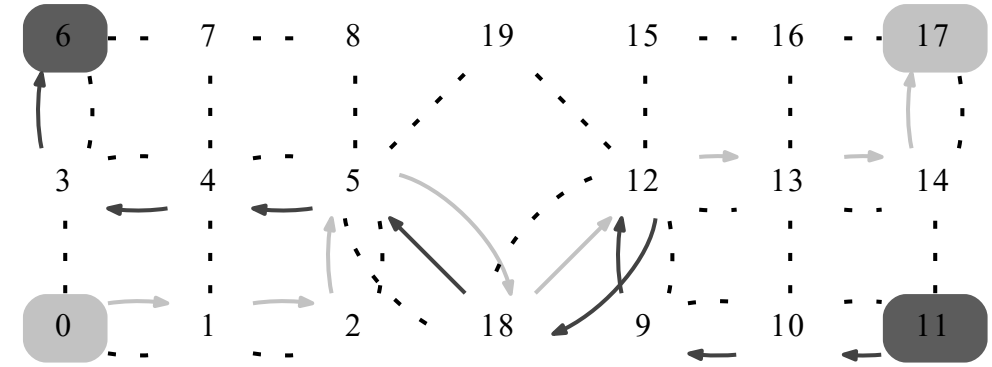
The Conflict-Free Pickup and Delivery Problem with Time Windows is to determine, for each vehicle $k = 1, \ldots, K$, a conflict-free path starting at time t_0 from a given location i_0, and the speed at which the vehicles will travel on each track-segment of the path, so that the requests assigned to the vehicles according to the sequence S_k are served within the time windows as specified in the set $\mathcal{C}_k$. The objective of the problem is to minimize a convex function describing the total energy required to serve the requests.

Figures 1(a) and 1(b) show an example of conflict and two ways in which it can be resolved. The example is based on a graph with 20 nodes, which are connected by 28 track-segments shown by the dashed lines in both figures. Figure 1(a) shows two shortest paths, one from node 0 to node 17 (grey arcs) and another from node 11 to node 6 (black arcs).

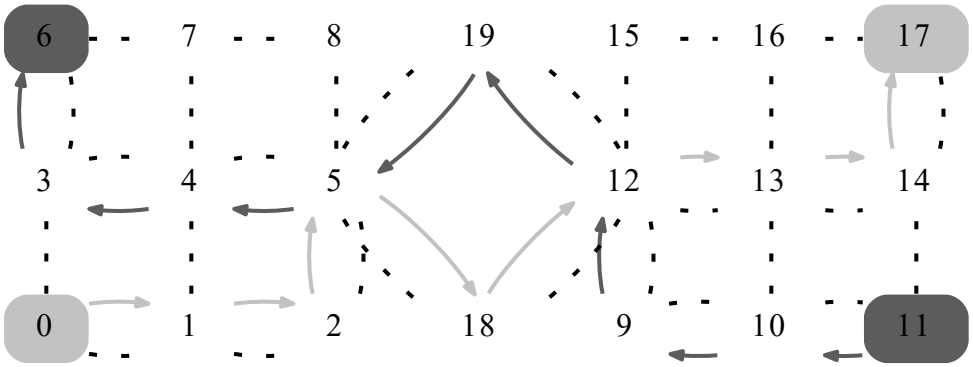
Assume that there exist two vehicles A and B, initially located at node 0 and node 11, respectively. Vehicle A starts traveling on the shortest path from node 0 to node 17 at time $t = 0$. Similarly, vehicle B starts travel from node 11 to node 6 at the same time on the shortest path. If the two vehicles travel at a constant speed of one track-segment per second, a node conflict will occur at node 18 at time $t = 4$. This conflict can be avoided by either modifying the speed or the path of one of the vehicles. In the former case, changing the speed of vehicle A to half a track-segment per second will avoid a potential collision. In the latter option, vehicle B can be routed on a different shortest path, as shown in Figure 1(b) using the black arcs, in order to avoid the conflict, without the need to change the speed from one track-segment per second.

3. A Branch-and-Bound Algorithm

In this section, we describe a Branch-and-Bound algorithm in order to solve the Conflict-Free Pickup and Delivery Problem with Time Windows. In particular, we propose a lower bounding procedure, a feasibility-check procedure and branching rules.



(a) Time-based Deviation



(b) Spatial Deviation

Figure 1 **Conflict Example: gray arcs refers to path of vehicle A and black arcs refers to path of vehicle B**

As will be further explained in the following sections, each branching operation adds a pool of time, sequencing and logical constraints to a node $\mathcal{P}$ of the branch-and-bound tree, which are modeled by the following operations, respectively:

- updating the time constraints in $\mathcal{C}_k$,
- adding new nodes in S_k and the corresponding time windows in $\mathcal{C}_k$,
- forbidding some arcs to be part of a path between two consecutive stops.

Hence, we characterize each node $\mathcal{P}$ as a sequence S'_k of H'_k nodes, with $H'_k \geq H_k$, and a set $\mathcal{C}'_k$ of time windows, for each vehicle $k \in \{1, \ldots, K\}$. Moreover, to manage arc conflicts arising in the search of an optimal solution, we define, for each vehicle $k \in \{1, \ldots, K\}$, a set $\Omega'_{kh} \subseteq E$ of arcs where $h \in \{1, \ldots, H'_k - 1\}$. Each arc of the set Ω'_{kh} is associated to a branching logical constraint defined as follows: if $(i, j) \in \Omega'_{kh}$ then the arc (i, j) cannot be included in a path from $S'_k[h]$ to $S'_k[h+1]$ traversed by vehicle k. In the following, we refer to the overall set of branching logical constraints associated to node $\mathcal{P}$ as Ω'_k, i.e. $\Omega'_k = \Omega'_{k1} \bigcup \cdots \bigcup \Omega'_{kH'_k-1}$. At the root node S'_k, $\mathcal{C}'_k$ and Ω'_k are initialized to S_k, $\mathcal{C}_k$ and $\emptyset$, respectively.

3.1. Lower bounding procedure

In order to compute a lower bound for a given node $\mathcal{P}$ of the branch-and-bound tree, we define a relaxation by removing the arc capacity constraint. The resulting relaxation $\mathcal{P}_R$ is a nonlinear separable problem, where the optimal solution can be determined by solving a Path and Speed Optimization Problem (SOP_k) for each vehicle $k \in \{1, \ldots, K\}$. We encode a feasible solution of SOP_k as follows. The sequence of n_{kh} arcs of G corresponding to a path from node $S'_k[h]$ to node $S'_k[h+1]$ that satisfy the logical constraints associated to Ω'_{kh} is denoted by $\mathbf{p}_{kh}$, where each arc $\ell = 1, \ldots, n_{kh}$ on the sequence is shown by $\mathbf{p}_{kh}[\ell]$. Let $\mathbf{v}_{kh}$ be a vector of n_{kh} travel speeds, stating that vehicle k traverses arc $\mathbf{p}_{kh}[\ell]$ at speed $\mathbf{v}_{kh}[\ell]$, for $\ell = 1, \ldots, n_{kh}$. We denote a feasible solution of SOP_k by $(\mathbf{p}_k, \mathbf{v}_k)$, where $\mathbf{p}_k = [\mathbf{p}_{k1}, \ldots, \mathbf{p}_{kh}, \ldots, \mathbf{p}_{kH'_k-1}]$ and $\mathbf{v}_k = [\mathbf{v}_{k1}, \ldots, \mathbf{v}_{kh}, \ldots, \mathbf{v}_{kH_k-1}]$, meaning that if a vehicle $k \in \{1, \ldots, K\}$ starts traversing the sequence $\mathbf{p}_k$ of paths at time t_0 using speeds $\mathbf{v}_k$, then the arrival times satisfy the time window constraints of $\mathcal{C}'_k$.

Each SOP_k can be formally defined as:

$$\min_{\mathbf{p}_k} \min_{\mathbf{v}_k} F(\mathbf{p}_k, \mathbf{v}_k), \tag{1}$$

where,

$$F(\mathbf{p}_k, \mathbf{v}_k) = \sum_{h=1}^{H'_k-1} \sum_{\ell=1}^{n_{kh}} f(\mathbf{v}_{kh}[\ell]). \tag{2}$$

It is worth noting that if we fix the routing component of (1) to a given routing solution $\mathbf{p}^\star_k$, the corresponding optimal speed values $\mathbf{v}^\star_k$ can be determined by solving the following optimization problem:

$$F(\mathbf{p}^\star_k, \mathbf{v}^\star_k) = \min_{\mathbf{v}_k} (F(\mathbf{p}_k, \mathbf{v}_k) | \mathbf{p}_k = \mathbf{p}^\star_k). \tag{3}$$

If $f(v)$ is a non-decreasing convex function, the optimization problem (3) can be solved in quadratic time by the exact algorithm presented by Hvattum et al. (2013). The optimization problem studied by Hvattum et al. (2013), namely a Ship Routing Problem (SRP), is defined with respect to a sequence of ports that a ship must visit with given time windows, where the objective is to determine the speed on each leg of the route so as to minimize the total fuel consumption.

We cast the optimization problem (3) as a SRP, where the sequence of nodes S'_k corresponds to a sequence of port calls with time windows defined according to $\mathcal{C}'_k$. The main difference between the SRP studied in Hvattum et al. (2013) and the optimization problem (3) concerns the physical network. In the SRP, there are no transit nodes in G, but instead only a single arc connecting two consecutive ports on a route. In Hvattum et al. (2013) the authors prove that, if the energy cost per distance unit $f(v)$ is a convex non-decreasing function, then there always exists an optimal solution where each leg on the route is traversed at a constant speed. Therefore, the algorithm

proposed in Hvattum et al. (2013) determines for each path $\mathbf{p}^{\star}_{kh}$ an optimal travel speed value $\omega^{\star}_{kn}$, such that $\mathbf{v}^{\star}_{k}[\ell] = \omega^{\star}_{kn}$, for $\ell = 1, \ldots, n_{kh}$. This implies that:

$$F(\mathbf{p}^{\star}_{k}, \mathbf{v}^{\star}_{k}) = \sum_{h=1}^{H'_k - 1} d^{\star}_{kh} f(\omega^{\star}_{kh}),$$

where $d^{\star}_{kh}$ is the length of path $\mathbf{p}^{\star}_{kh}$. Let G'_{kh} be a graph obtained from G by deleting each arc $(i,j) \in \Omega'_{kh}$. Theorem 1 provides optimality conditions for problem (1).

THEOREM 1. *If each path $\mathbf{p}^{\star}_{kh}$ of $\mathbf{p}^{\star}_{k}$ corresponds to a shortest path on G'_{kh}, then $(\mathbf{p}^{\star}_{k}, \mathbf{v}^{\star}_{k})$ is optimal for (1), that is:*

$$F(\mathbf{p}^{\star}_{k}, \mathbf{v}^{\star}_{k}) = \min_{\mathbf{p}_k} \min_{\mathbf{v}_k} F(\mathbf{p}_k, \mathbf{v}_k). \tag{4}$$

Proof of Theorem 1 The proof is by contradiction. Assume that there exists a feasible solution $(\mathbf{p}^{(1)}_{k}, \mathbf{v}^{(1)}_{k}) \neq (\mathbf{p}^{\star}_{k}, \mathbf{v}^{\star}_{k})$ of (1) for which the following holds:

$$F(\mathbf{p}^{(1)}_{k}, \mathbf{v}^{(1)}_{k}) < F(\mathbf{p}^{\star}_{k}, \mathbf{v}^{\star}_{k}),$$

where we assume, without loss of generality, that $\mathbf{v}^{(1)}_{k}$ is determined as the optimal solution of the following subproblem:

$$F(\mathbf{p}^{(1)}_{k}, \mathbf{v}^{(1)}_{k}) = \min_{\mathbf{v}_k} (F(\mathbf{p}_k, \mathbf{v}_k) | \mathbf{p}_k = \mathbf{p}^{(1)}_{k}).$$

Using the result by Hvattum et al. (2013), we have

$$F(\mathbf{p}^{(1)}_{k}, \mathbf{v}^{(1)}_{k}) = \sum_{h=1}^{H'_k - 1} d^{(1)}_{kh} f(\omega^{(1)}_{kh}),$$

where $\omega^{(1)}_{kh}$ and $d^{(1)}_{kh}$ are, respectively, the optimal travel speed and the length associated to path $\mathbf{p}^{(1)}_{kh}$ of $\mathbf{p}^{(1)}_{k}$. It is worth noting, by the hypothesis that each path $\mathbf{p}^{\star}_{kh}$ of $\mathbf{p}^{\star}_{k}$ corresponds to a shortest path on G'_{kh}, that $d^{(1)}_{kh} \geq d^{\star}_{kh}$, given that $d^{\star}_{kh}$ is the length of the corresponding shortest path. We now construct a solution $(\mathbf{p}^{\star}_{k}, \overline{\mathbf{v}}_{k})$ feasible for (3), where the traversal time of each path $\mathbf{p}^{\star}_{kh}$ is equal to the traversal time of path $\mathbf{p}^{(1)}_{kh}$ in $(\mathbf{p}^{(1)}_{k}, \mathbf{v}^{(1)}_{k})$, namely:

$$\overline{\omega}_{kh} = d^{\star}_{kh} \times \frac{\omega^{(1)}_{kh}}{d^{(1)}_{kh}},$$

where $\overline{\mathbf{v}}_{kh}[\ell] = \overline{\omega}_{kh}$ for $\ell = 1, \ldots, H'_k$. The total energy consumption $F(\mathbf{p}^{\star}_{k}, \overline{\mathbf{v}}_{k})$ for the new solution $(\mathbf{p}^{\star}_{k}, \overline{\mathbf{v}}_{k})$ can be computed as follows:

$$F(\mathbf{p}^{\star}_{k}, \overline{\mathbf{v}}_{k}) = \sum_{h=1}^{H' - 1} d^{\star}_{kh} f(\overline{\omega}_{kh}).$$

Since $\overline{\omega}_{kh} \leq \omega^{(1)}_{kh}$ and $f(v)$ is non-decreasing, the following holds:

$$F(\mathbf{p}^{\star}_{k}, \overline{\mathbf{v}}_{k}) \leq F(\mathbf{p}^{(1)}_{k}, \mathbf{v}^{(1)}_{k}) < F(\mathbf{p}^{\star}_{k}, \mathbf{v}^{\star}_{k}). \tag{5}$$

Inequalities (5) contradict the hypothesis that $F(\mathbf{p}_k^\star, \mathbf{v}_k^\star)$ is optimal for optimization problem (3). $\square$

According to Theorem 1 we determine the optimal solution $(\mathbf{p}_k^\star, \mathbf{v}_k^\star)$ of (1) in two steps.

Step 1. Determine each path $\mathbf{p}_{kh}^\star$ of $\mathbf{p}_k^\star$ by solving a shortest path problem between origin $S_k'[h]$ and destination $S_k'[h+1]$ on graph G_{kh}' where $E' = E \setminus (i,j)$ with $(i,j) \in \Omega_{kh}'$.

Step 2. Set $\mathbf{v}_{kh}^\star[\ell] = \omega_{kh}^\star$, with $\ell = 1, \ldots, n_{kh}^\star$. In particular $(\omega_{k1}^\star, \ldots, \omega_{kH_k'-1}^\star)$ is determined by means of the algorithm presented in Hvattum et al. (2013) as the optimal solution of a SRP defined by:

- the sequence S_k' of locations visited within time windows $\mathcal{C}_k'$, and
- the distance $d_{kh}^\star$ between two consecutive locations.

Let $(\mathbf{p}^\star, \mathbf{v}^\star)$ be the optimal solution of the relaxed problem $\mathcal{P}_R$, i.e., $\mathbf{p}^\star = (\mathbf{p}_1^\star, \ldots, \mathbf{p}_K^\star)$ and $\mathbf{v}^\star = (\mathbf{v}_1^\star, \ldots, \mathbf{v}_K^\star)$. If the total energy consumption $\sum_{k=1}^{K} F(\mathbf{p}_k^\star, \mathbf{v}_k^\star)$ is not greater than the total energy of the current incumbent solution, then the branch-and-bound algorithm checks if $(\mathbf{p}^\star, \mathbf{v}^\star)$ is conflict free. For this purpose we propose a mechanism that will be described next.

3.2. Feasibility check

To check whether an optimal solution $(\mathbf{p}^\star, \mathbf{v}^\star)$ to the relaxed problem $\mathcal{P}_R$ corresponds to a conflict-free routing on G, and to identify the type of conflict if not, we use a FeasibilityCheck procedure that is shown in Algorithm 1. The procedure is called on each pair of distinct vehicles k and k'. If FeasibilityCheck returns **None** for any pair of vehicles k and k', then there are no conflicts and the solution $(\mathbf{p}^\star, \mathbf{v}^\star)$ is feasible. Otherwise, the procedure identifies the *type* of the conflict by using a subroutine called the FindConflict procedure shown in Algorithm 2 and operates as explained below.

Let us consider two distinct vehicles; vehicle k traveling on arc (i,j) during the time interval $[t_i, t_j]$, and vehicle k' on arc (r,s) during the time interval $[t_r, t_s]$. No conflicts occur between k and k' during the time interval $[t_i, t_j] \cap [t_r, t_s]$, if the following conditions hold:

$$\text{if } (r,s) = (i,j) \vee (r,s) = (j,i) \Rightarrow [t_i, t_j] \cap [t_r, t_s] = \emptyset, \tag{6}$$

$$\text{if } j = s \Rightarrow t_j \neq t_s. \tag{7}$$

The FindConflict procedure checks conflict by verifying whether conditions (6) and (7) are satisfied. If no conflicts occur FindConflict returns the value **None**. Otherwise, FindConflict returns the type of the conflict identified. In particular, if conditions (6) or (7) do not hold, then FindConflict returns an *Arc-conflict* or a *Node-conflict*, respectively.

The type of conflict, once detected, is returned by the FeasibilityCheck procedure in its space-time in the form $((i,j),(r,s),[t_i,t_j],[t_r,t_s])$ and the corresponding indices h, h', ℓ, ℓ', and is subsequently given as input to the branching procedure that is described in the following section.

Algorithm 1 Feasibility check procedure

function FEASIBILITYCHECK(k,k',$\mathbf{p}$,$\mathbf{v}^\star$)
  $t_i \leftarrow 0$
  $t_r \leftarrow 0$
  for all $h \in [1,\dots,H'_k-1]$ **and** $h' \in [1,\dots,H'_{k'}-1]$ **do**
    for all $\ell \in [1,\dots,n_{kh}]$ **and** $\ell' \in [1,\dots,n_{k'h'}]$ **do**
      $(i,j) \leftarrow \mathbf{p}_{kh}[\ell]$
      $(r,s) \leftarrow \mathbf{p}_{k'h'}[\ell']$
      $t_j \leftarrow t_i + 1/\mathbf{v}^\star_{kh}[\ell]$
      $t_s \leftarrow t_r + 1/\mathbf{v}^\star_{kh'}[\ell']$
      type $\leftarrow$ FINDCONFLICT($(i,j),(r,s),[t_i,t_j],[t_r,t_s]$))
      if type$\neq$ **None then**
        return (type,$(i,j),(r,s),[t_i,t_j],[t_r,t_s],h,h',\ell,\ell'$)
      else
        $t_i \leftarrow t_i + 1/\mathbf{v}^\star_{kh}[\ell]$
        $t_r \leftarrow t_r + 1/\mathbf{v}^\star_{kh'}[\ell']$
  return None
end function

Algorithm 2 Conflict detection procedure

function FINDCONFLICT((i,j),(r,s),$[t_i,t_j]$, $[t_r,t_s]$)
  if $(i,j)=(r,s)$ **or** $(j,i)=(r,s)$ **then**
    if $[t_i,t_j]\cap[t_r,t_s]\neq\emptyset$ **then**
      return *Arc-conflict*
  if $j=s$ **and** $t_j=t_s$ **then**
    return *Node-conflict*
  return None
end function

3.3. Branching strategy

When FEASIBILITYCHECK procedure determines a conflict, the branching procedure defines a partition of the feasible region of $\mathcal{P}$ into N_B subsets, so that the infeasible solution $(\mathbf{p}^\star,\mathbf{v}^\star)$ is cut off and no feasible solution is lost. The basic idea is as follows. When a conflict arises on an arc (or in a node), there are two viable alternatives:

- at least one vehicle is re-routed (*space-based branching*);
- at least one of the vehicles is time-constrained (*time-based branching*).

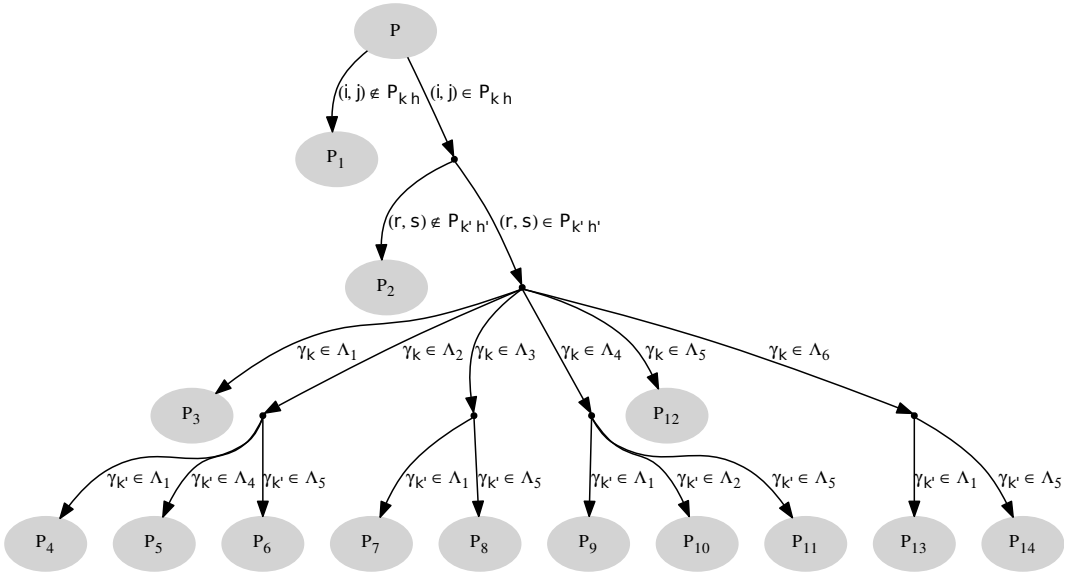


Figure 2 Time-based and space-based branching for an Arc-Conflict

The latter option can be implemented by adding dummy time windows to the conflicting vehicles in such a way that their traversal of the conflicting arc (or node) is delayed or anticipated. Formally, for each conflict *type*, we devise a branching procedure that produces N_B child nodes, each of which is shown as $\mathcal{P}_\ell$, where $\ell = 1, \ldots, N_B$. As depicted in Figures 2 and 3, each branching procedure is modeled as a decision tree where branch-and-bound nodes correspond to the root node (i.e., the parent node $\mathcal{P}$) and the leaf nodes (i.e., the child nodes $\mathcal{P}_\ell$, with $\ell = 1, \ldots, N_B$). In particular the number N_B of child nodes is equal to six and 14 for *Node-conflict* and *Arc-conflict*, respectively. Each child node $\mathcal{P}_\ell$ is obtained by adding to the parent node $\mathcal{P}$ all constraints associated to the path of the decision tree starting from the root node to the leaf node associated with $\mathcal{P}_\ell$, for $\ell = 1, \ldots, N_B$. For example, in Figure 2, the child node $\mathcal{P}_1$ is obtained by adding to the parent node $\mathcal{P}$ the constraint $(i,j) \notin \mathbf{p}_{kh}$.

3.3.1. Space-based branching. For both arc and node conflicts, the algorithm partitions the feasible solutions of $\mathcal{P}$ into two sets:

- feasible solutions satisfying constraint $(i,j) \notin \mathbf{p}_{kh}$, which states that the vehicle k **does not traverse** arc (i,j) when traveling from node $S'_k[h]$ to node $S'_k[h+1]$;
- feasible solutions satisfying constraint $(i,j) \in \mathbf{p}_{kh}$, which states that the vehicle k **traverses** arc (i,j) when traveling from node $S'_k[h]$ to node $S'_k[h+1]$.

A further branching partitions the feasible solutions satisfying $(i,j) \in \mathbf{p}_{kh}$ in two sets:

- feasible solutions satisfying constraint $(r,s) \notin \mathbf{p}_{k'h'}$, which states that the vehicle k' **does not traverse** arc (r,s) when traveling from node $S'_{k'}[h']$ to node $S'_{k'}[h'+1]$;
- feasible solutions satisfying constraint $(r,s) \in \mathbf{p}_{k'h'}$, which states that the vehicle k' **traverses** arc (r,s) when traveling from node $S'_{k'}[h']$ to node $S'_k[h'+1]$.

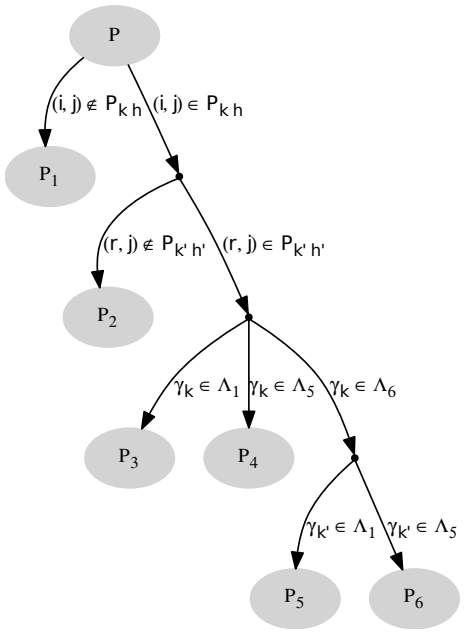


Figure 3 Time-based and space-based branching for a Node-conflict

The remaining time-based branching refers to cutting the solution $(\mathbf{p}^\star, \mathbf{v}^\star)$ off from the set of solutions satisfying the constraints $(i,j) \in \mathbf{p}_{kh}$ and $(r,s) \in \mathbf{p}_{k'h'}$. The space-based branching constraints $(i,j) \notin \mathbf{p}_{kh}$ and $(i,j) \in \mathbf{p}_{kh}$ are added to SOP_k of the child nodes by modifying the input data $\mathcal{C}'_k, S'_k$ and Ω'_k of the parent node $\mathcal{P}$ as follows:

- constraint $(i,j) \notin \mathbf{p}_{kh}$ is modeled by adding the arc (i,j) to the set Ω'_{kh};
- constraint $(i,j) \in \mathbf{p}_{kh}$ is modeled by inserting the arc $(i,\, j)$ between nodes $S'_k[h]$ and $S'_k[h+1]$.

For this purpose, we first set $S'_k[h+t]$ equal to $S'_k[h+t+2]$, where $1 \leq t \leq H'_k - h$. Then, we set $S'_k[h+1] = i$ and $S'_k[h+2] = j$. Finally, we increment H'_k by 2. The set $\mathcal{C}'_k$ is updated by adding the time windows $\alpha_{k,h+1} = [t_0, T]$ and $\alpha_{k,h+2} = [t_0, T]$ on the new nodes of the sequence.
In a similar way, the space-based branching constraints $(r,s) \notin \mathbf{p}_{k'h'}$ and $(r,s) \in \mathbf{p}_{k'h'}$ are added to $\text{SOP}_{k'}$ of the child nodes by modifying $\mathcal{C}'_{k'}, S'_{k'}$ and $\Omega'_{k'}$ of the parent node $\mathcal{P}$. It is worth noting that, removing arc (r,s) from $\mathbf{p}_{k'h'}$ corresponds to traversing a shortest path whose length is greater than (or the same as) the previous one.

3.3.2. Time-based branching. In the following we describe how the time-based branching procedure has been tailored for each conflict type. In particular, we refer to arcs (i,j) and (r,s) of Algorithm 1 as $(S'_k[h+1], S'_k[h+2])$ and $(S'_{k'}[h'+1], S'_{k'}[h'+2])$, respectively.

Arc-conflict. As reported in Algorithm 1, an *Arc-conflict* occurs if vehicles k and k' are traversing the same track-segment during the time interval $[t_1, t_2] = [t_i, t_j] \cap [t_r, t_s]$. We first partition the planning horizon as follows:

$$[t_0, T] = [t_0, t_1[\cup [t_1, t'_2] \cup]t'_2, T], \tag{8}$$

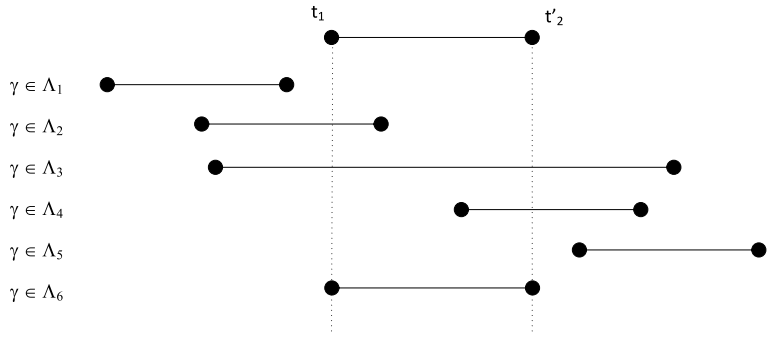


Figure 4 **The subset Λ_q with $q = 1, \ldots, 6$**

Table 1 **Time windows $\mathcal{C}'_k$ imposed to subproblem SOP_k for each branching constraint $\gamma_k \in \Lambda_q$ ($q = 1, \ldots, 6$)**

Branching constraint	Time window $\mathcal{C}'_k$
$\gamma_k \in \Lambda_1$	$\alpha_{kh+1} = [t_0, t_1[, \alpha_{kh+2} = [t_0, t_1[$
$\gamma_k \in \Lambda_2$	$\alpha_{kh+1} = [t_0, t_1[, \alpha_{kh+2} = [t_1, t'_2]$
$\gamma_k \in \Lambda_3$	$\alpha_{kh+1} = [t_0, t_1[, \alpha_{kh+2} =]t'_2, T]$
$\gamma_k \in \Lambda_4$	$\alpha_{kh+1} = [t_1, t'_2], \alpha_{kh+2} =]t'_2, T]$
$\gamma_k \in \Lambda_5$	$\alpha_{kh+1} =]t'_2, T], \alpha_{kh+2} =]t'_2, T]$
$\gamma_k \in \Lambda_6$	$\alpha_{kh+1} = [t_1, t'_2], \alpha_{kh+2} = [t_1, t'_2]$

where $t'_2 = \min(t_2, t_1 + 1/\overline{v})$ and $1/\overline{v}$ is the minimum time to traverse a track-segment of unit length.

Let Λ be the set of all time intervals $\gamma \subseteq [t_0, T]$ whose durations are greater than or equal to the minimum travel time $1/\overline{v}$. According to (8), we use Λ to form six subsets of intervals $\Lambda_1, \ldots, \Lambda_6$, as shown in Figure 4. More formally, Λ_1, Λ_2 and Λ_3 consist of time intervals starting in $[t_0, t_1[$ and ending in $[t_0, t_1[$, $[t_1, t'_2]$ and $]t'_2, T]$, respectively. Similarly, Λ_4 and Λ_5 consist of time intervals starting in $[t_1, t'_2]$ and in $]t'_2, T]$, respectively, but both ending in $]t'_2, T]$. Finally, Λ_6 corresponds to $[t_1, t'_2]$. The time windows $[t_0, t_1[$ are modeled as $[t_0, t_1 - \epsilon]$. We recall that ϵ induces an implicit time discretization and observe that, with respect to a given ϵ, the following branch-and-bound algorithm is an exhaustive search. In particular, running the algorithm to completion determines feasibility.

Let us denote by z_{kh} the arrival time at node $S'_k[h]$. For notational convenience, we define the branching constraints in terms of constraints on time intervals $\gamma_k \in \Lambda$ and $\gamma_{k'} \in \Lambda$, where $\gamma_k = [z_{kh+1}, z_{kh+2}]$ and $\gamma'_k = [z_{k'h'+1}, z_{k'h'+2}]$. For example, $\gamma_k \in \Lambda_2$ is equivalent to stating that $z_{kh+1} \in [t_0, t_1[$ and $z_{kh+2} \in [t_1, t'_2]$. In particular, the time-based partition for the *Arc-conflict* is defined by all pairs of time constraints $(\gamma_k \in \Lambda_q, \gamma_{k'} \in \Lambda_{q'})$, with $q, q' = 1, \ldots, 6$. As reported in Table 1, given a vehicle k, constraint $\gamma_k \in \Lambda_q$ is added to the child nodes by updating time windows α_{kh+1} and α_{kh+2}, with $q = 1, \ldots, 6$.

The time-based branching shown in Figure 3 cuts off a set of infeasible solutions satisfying the following condition:

$$\gamma_k \cap \gamma_{k'} \cap [t_1, t'_2] \neq \emptyset. \tag{9}$$

In particular, condition (9) holds for all solutions satisfying (10)–(13), where:

$$(\gamma_k \in \Lambda_2, \gamma_{k'} \in \Lambda_{q'}), \qquad q' = 2, 3, 6, \tag{10}$$

$$(\gamma_k \in \Lambda_3, \gamma_{k'} \in \Lambda_{q'}), \qquad q' = 2, 3, 4, 6, \tag{11}$$

$$(\gamma_k \in \Lambda_4, \gamma_{k'} \in \Lambda_{q'}), \qquad q' = 3, 4, 6, \tag{12}$$

$$(\gamma_k \in \Lambda_6, \gamma_{k'} \in \Lambda_{q'}), \qquad q' = 2, 3, 4, 6. \tag{13}$$

It is worth noting that the infeasible solution $(\mathbf{p}^\star, \mathbf{v}^\star)$ satisfies constraints $(\gamma_k \in \Lambda_3, \gamma_{k'} \in \Lambda_3)$. On the other hand, feasible solutions do not satisfy condition (9), i.e. they do not comprise vehicles colliding in $[t_1, t_2']$.

Node-conflict. As mentioned earlier, a *Node-conflict* occurs at node j, when vehicles k and k' are traveling on arcs (i, j) and arc (r, j), respectively, visit node j at the same time t. We observe that a *Node-conflict* is a special case of an *Arc-conflict*, occurring on a dummy arc (j, j) with $c_{jj} = 0$. Taking into account that $t_1 = t_2 = t$, the time partition (8) becomes:

$$[t_0, T] = [t_0, t[\cup[t, t]\cup]t, T]. \tag{14}$$

Since subsets Λ_2, Λ_3 and Λ_4 are empty sets for the dummy arc (j, j), the time-based branching for a *Node-conflict* can be modeled as shown in Figure 3. In particular, a first time-based partition is applied to feasible solutions as follows.

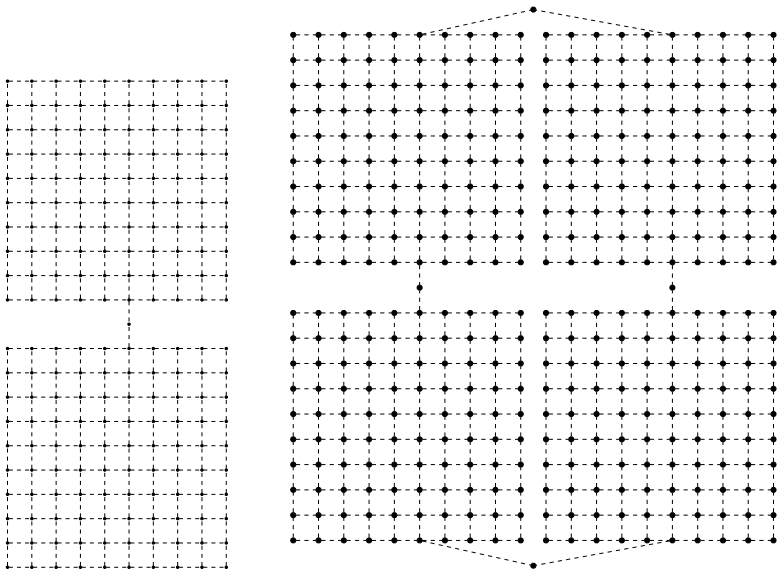
- Feasible solutions satisfying constraint $\gamma_k \in \Lambda_1$, which states that the vehicle k arrives at node j **before** t.
- Feasible solutions satisfying constraint $\gamma_k \in \Lambda_5$, which states that the vehicle k arrives at node j **after** t.
- Feasible solutions satisfying constraint $\gamma_k \in \Lambda_6$, which states that the vehicle k arrives at node j **at time** t.

Then, a further time-based branching cuts off the infeasible solution $(\mathbf{p}^\star, \mathbf{v}^\star)$ from child nodes $\{\mathcal{P}_5, \mathcal{P}_6\}$ by the following time constraints:

- the vehicle k' arrives at node j **before** t, i.e., $\gamma_{k'} \in \Lambda_1$;
- the vehicle k' arrives at node j **after** t, i.e., $\gamma_{k'} \in \Lambda_5$.

4. Computational Results

This section presents the results of the computational experiments conducted to (a) evaluate the performance of the branch-and-bound algorithm, and (b) quantify the value of speed optimization in terms of the potential savings that can be achieved in energy consumption. All experiments reported in this section are run on a standalone Linux machine with a processor clocked at 2.67 GHz and equipped with 24 GB of RAM. The algorithm is coded in Java. We first describe the generation of input data, and then present the results.



(a) 2-bay graph (b) 4-bay graph

Figure 5 Grid graphs used to generate the instances

4.1. Input data

Instances have been generated to resemble the uniform layout typically found in a warehouse or in a port terminal. Such layouts are naturally partitioned into interconnected groups of nodes called bays. To this end, we first create a 10×10 grid network $G_B = (V_B, E_B)$ corresponding to a bay. Then, we generate two graphs $G = (V, E)$ by linking together two or four copies of G_B using single arcs between bays as shown in Figure 5, where the number $|V|$ of vertices is either 200 or 400.

For each graph G generated, we use $K = 3, 5, 10$ vehicles. For each vehicle $k \in \{1, \ldots, K\}$, we generate a random sequence S_k of $H_k = 5, 10$ requests such that $S_k[h-1]$ and $S_k[h]$ belong to different bays with a probability equal to p for $h = 2, \ldots, H_k$. In order to generate graphs with different level of sparsity, for each instance, we identify arcs $(i, j) \in V_B$ that do not belong to any shortest path $\mathbf{p}^\star_{kh}$, and delete them from the graph with probability $\pi \in \{0, 0.5\}$.

As for the time windows, we generate two types of instances, A and B, defined as follows. First, we randomly generate a speed value $v' \in]0, \overline{v}]$. For instances of type A, all nodes in S_k for a given vehicle k share a common time window $[0, T_k]$, where:

$$T_k = \sum_{h=1}^{H_k} n^\star_{kh} / v'.$$

For instances of type B, for each node $S_k[h]$, we set the time window to $[\beta \cdot \overline{\alpha}_{kh}, \overline{\alpha}_{kh}]$, where the latest arrival time $\overline{\alpha}_{kh}$ is computed as $(1+\mu) \cdot \sum_{\ell=1}^{h-1} n^\star_{k\ell}/v'$, with μ randomly generated in $[-\theta, +\theta]$. For each configuration of parameters $|V|$, K, H_k and π, we have generated and solved 30 instances. For type B instances, we test three values of θ as 0.1, 0.3 and 0.5 and two values of β as 0 and

0.5. Each combination (θ, β) corresponds to a different level of tightness of the time windows. In the following we report results for instances with wide ($\beta = 0$, $\theta = 0.1$) and tighter ($\beta = 0.5$, $\theta = 0.3$ and $\beta = 0.5$, $\theta = 0.5$) time windows, where θ controls the variability of the latest arrival times. The combination of these parameters yields a total of 1920 instances. In our experimentation, ϵ was set equal to 1 millisecond. Given that the problem we solve arises as part of a more complex routing problem, the computation time should be kept reasonably short. Therefore, a maximum of 300 seconds was imposed on the solution time of each instance.

We model energy consumption as the total tractive power requirements as in Young et al. (2013), which is particularly relevant for vehicles that run on batteries. In particular, if we assume zero acceleration and zero road angle, the total energy (in Joule = kg $\cdot$ m^2/s^2) required by a vehicle traversing a segment of d (meters) at a constant speed v (m/s) is calculated as follows,

$$f(v) = (0.5 C_d A \rho v^2 + M g C_r) d / 1000,$$

where M is the total vehicle weight (kg), g is the gravitational constant, C_d and C_r are the coefficient of the aerodynamic drag and rolling resistance, A (m^2) is the frontal surface area of the vehicle and ρ is air density (kg/m^3). Table 2 shows the parameter settings used in the energy consumption model.

Table 2 Parameter settings

Parameter	Value
C_d	0.70
C_r	0.01
g	9.81 m/s^2
A	2.86 m^2
$\overline{v}$	1.00 m/s
M	320 kg
ρ	1 kg/m^3

A series of preliminary experiments were carried out to assess the impact of the search strategy. We start by observing that due to the time-based branching the depth of the branch-and-bound tree is not known a priori. In particular, we observe that the ratio between the lower bound at the root node and the optimal solution is very close to 1. This implies that the Best-First Strategy (BFS) is equivalent to the Depth-First Strategy (DFS), which might be inefficient in determining a first feasible solution if the search depth is not known a priori. We overcome this drawback by using Iterative Deepening Depth-First Search (IDDFS), in which a depth-limited version of DFS is run repeatedly with increasing depth limits until a dead end is found. Our experiments suggested that the IDDFS is superior to the BFS as it provides a reduction of about 50% in the number of instances unsolved, i.e., instances for which no feasible solution can be found within the time limit. For this reason, all tests are carried out with the IDDFS.

4.2. Results on the performance of the algorithm

The experiments reported in this section are conducted to assess the effectiveness of the branch-and-bound algorithm in solving the instances to optimality. Results are reported in Table 3 for instances of type A, and in Tables 4–6 for instances of type B. All four tables have the same format, where the first four columns are self-explanatory. The remaining column headings are as follows.

- $TIME$: time spent to find an optimal solution (in seconds);
- $NODES$: number of nodes in the search tree;
- $SUCC$: number of instances solved (see below for a detailed explanation);
- ρ: the depth of the branch-and-bound node corresponding to the best solution available at the end of the search;
- LB/OPT: ratio between the initial lower bound value on the best objective function available at the end of the search;
- GAP_f: the final optimality gap ($\times 10^{-3}$);
- GAP_I: optimality gap of the first feasible solution found ($\times 10^{-3}$);
- $TIME_I$: time spent to determine the first feasible solution (in seconds).

We report the results for each instance configuration in three distinct rows, where the first row presents results for the set of instances solved to optimality within the time limit, the number of which is reported in column $SUCC$. The second row pertains to the set of instances for which at least one feasible solution was found, but optimality was not proven within the time limit. The third row reports results for the set of instances certified as infeasible within the time limit. For the sake of conciseness, the second and the third row have been omitted whenever the corresponding set of instances is empty, i.e., when the corresponding value of $SUCC$ column is 0. Similarly, we do not report any results on instances for which the algorithm was neither able to find a feasible solution nor prove infeasibility within the time limit. For columns from LB/OPT to $TIME_I$ we only report averages for instances with at least one feasible solution determined within the time limit. All other columns report results which are averaged across the corresponding number of instances reported in column $SUCC$. Optimality gaps were computed as $10^3 \times (upper\_bound - lower\_bound)/lower\_bound$. The final optimality gap GAP_f is zero if the instance was solved to optimality. Otherwise, GAP_f is evaluated with respect to the best lower and upper bounds found within the time limit of 300 seconds. Similarly, GAP_I is calculated with respect to the first feasible solution identified and the best lower bound.

Computational results show, for the 480 instances of type A tested, that the algorithm is able to solve 445 to optimality, and find at least one feasible solution for the remaining 35 instances. As for the 480 type B instances with wide time windows (i.e. $\beta = 0$ and $\theta = 0.1$), the algorithm has successfully solved 369 to optimality, and produced at least one feasible solution for 73 of

Table 3 Computational results for instances of type A

$\|V\|$	π	K	$\|R\|$	$TIME$	$NODES$	$SUCC$	ρ	LB/OPT	GAP_f	GAP_I	$TIME_I$
200	0	3	5	0.11	9	30	1.00	1.000000	0.000	0.570	0.10
			10	0.38	61	28	1.93	1.000000	0.000	1.125	0.31
				300.05	371635	2	3.00	0.999817	0.183	0.183	0.34
		5	5	3.13	241	28	4.14	1.000000	0.000	1.311	2.82
				300.05	301362	2	8.50	0.999928	0.072	0.089	4.56
			10	8.70	1164	29	6.48	1.000000	0.000	0.420	6.11
				300.05	360680	1	5.00	0.999543	0.431	0.596	0.58
	0.5	3	5	0.08	4	30	0.60	1.000000	0.000	2.300	0.08
			10	0.15	12	28	0.82	1.000000	0.000	0.058	0.14
				300.05	235059	2	1.50	0.999817	0.183	0.183	0.24
		5	5	0.32	26	28	1.89	0.999998	0.000	0.721	0.30
				300.05	313055	2	5.00	0.999927	0.073	0.089	1.64
			10	0.59	29	27	2.19	1.000000	0.000	0.962	0.54
				300.05	220491	3	4.33	0.999678	0.322	12.907	0.50
400	0	5	5	0.31	32	28	1.68	1.000000	0.000	0.049	0.23
				300.05	208448	2	1.50	0.995611	4.428	13.155	0.21
			10	1.83	121	28	3.43	1.000000	0.000	5.783	1.62
				300.05	172035	2	7.00	0.956482	47.648	51.610	8.03
		10	5	13.24	513	25	6.64	1.000000	0.000	0.368	11.72
				300.08	149029	5	6.80	0.996529	3.493	3.901	11.17
			10	15.33	379	25	6.64	0.999999	0.000	0.476	12.20
				300.06	147221	5	8.00	0.987245	13.623	15.804	6.60
	0.5	5	5	0.18	14	30	0.80	1.000000	0.000	1.196	0.15
			10	0.37	33	29	1.10	1.000000	0.000	0.602	0.34
				300.05	172325	1	4.00	0.999879	0.121	0.128	1.02
		10	5	1.66	36	26	2.69	1.000000	0.000	0.660	1.64
				300.05	218898	4	3.50	0.996655	3.368	3.878	0.80
			10	2.10	31	26	2.69	1.000000	0.000	0.355	1.95
				300.05	106808	4	5.25	0.998096	1.913	1.972	1.72
Average				24.53	15116	480	2.90	0.999582	0.445	1.605	2.47

these instances, leaving only 38 instances for which no feasible solution was found. The average solution time across all the instances is 7.65 seconds, with an average optimality GAP_I equal to 3.746×10^{-3}.

The results presented in Tables 5 and 6 give an indication on the difficulty of the problem with increasing values of tightness for time windows. For $\beta = 0.5$ and $\theta = 0.3$ ($\theta = 0.5$) the algorithm has successfully solved 354 (148) to optimality, and produced at least one feasible solution for 55 (27) of these instances, leaving only 69 (7) instances for which no feasible solution was found and 2 (298) instances declared infeasible.

For instances solved to optimality, the time $TIME_I$ needed to determine a first feasible solution is close to the value of the total execution $TIME$. Indeed, since the ratio LB/OPT is often either equal to 1 or is very close to 1, the algorithm stops as soon as the incumbent solution is updated with an optimal solution. We also note that the values shown under column ρ could be interpreted as the number of conflicts identified and resolved by the algorithm for an infeasible solution at the root node. The values indicate that the algorithm is able to identify a feasible solution within

Table 4 Computational results for instances of type B, wide time windows with $\beta = 0$, $\theta = 0.1$

$\|V\|$	π	K	$\|R\|$	$TIME$	$NODES$	$SUCC$	ρ	LB/OPT	GAP_f	GAP_I	$TIME_I$
200	0	3	5	0.13	24	30	1.37	0.999983	0.002	0.527	0.12
			10	0.29	57	29	2.03	0.999975	0.000	0.153	0.26
				300.05	340822	1	3.00	0.999599	0.401	0.401	0.92
		5	5	4.19	703	28	4.14	0.999957	0.000	0.000	4.19
				300.05	216926	1	2.00	0.999999	0.001	0.001	0.55
			10	18.06	4657	23	6.30	0.999986	0.000	0.406	12.14
				300.05	535473	4	5.50	0.999753	0.038	5.150	1.49
	0.5	3	5	0.10	20	28	0.96	0.999981	0.002	0.151	0.09
				300.05	242743	2	1.50	0.988319	11.844	11.844	0.13
			10	0.18	73	26	1.08	0.999986	0.000	2.555	0.17
				300.41	217222	4	2.00	0.989332	10.734	38.457	0.21
		5	5	1.20	329	21	2.62	0.999173	0.000	0.440	1.18
				300.05	285300	8	2.75	0.993443	6.872	15.675	0.26
			10	9.71	1630	17	3.06	0.999784	0.000	1.779	9.63
				300.05	283810	11	4.18	0.998432	1.142	10.091	2.12
400	0	5	5	8.02	4780	29	2.17	0.999999	0.001	0.742	5.51
				300.05	209306	1	2.00	0.990554	9.536	9.536	0.25
			10	2.63	302	25	4.48	0.999998	0.001	3.418	1.45
				300.05	197604	4	1.50	0.950191	55.567	101.470	0.27
		10	5	30.79	1422	17	8.94	0.999996	0.000	1.372	25.68
				300.20	170144	4	8.25	0.996231	3.696	3.684	69.61
			10	31.59	1756	16	7.25	0.998881	0.000	1.254	30.07
				300.07	112612	3	10.00	0.974535	27.170	27.170	5.25
	0.5	5	5	0.15	7	23	0.65	0.999997	0.001	0.424	0.14
				300.05	248195	7	3.71	0.999116	0.620	19.540	0.75
			10	0.78	163	23	2.09	0.999892	0.002	1.960	0.66
				300.06	206824	6	2.50	0.994730	5.311	5.710	1.24
		10	5	45.22	4894	14	5.36	0.997492	0.000	4.489	39.88
				300.06	154561	11	5.00	0.998411	1.319	2.462	21.47
			10	25.28	2560	20	5.10	0.999408	0.000	3.057	24.11
				300.30	115348	6	5.83	0.997949	1.906	6.986	4.98
Average				56.87	39220	442	3.42	0.998649	1.188	3.746	7.65

the time limit when the number of conflicts is generally less than 10. The results also suggest that on instances with tightly constrained time windows the algorithm is more effective when $\pi = 0$, as sparser instances appear to be more difficult to solve. A similar behavior can be observed with respect to the number of vehicles. Indeed, as the fleet size K increases the number of instances solved by the algorithm decreases, because of the higher network congestion. For instance, when $\beta = 0.5$, $\theta = 0.3$, $|V| = 400$, $\pi = 0.5$ and $R = 10$ all instances are solved with $K = 5$ whilst only six instances are solved with $K = 10$. This trend becomes even more evident with tighter time windows ($\beta = 0.5$, $\theta = 0.5$). In this case, the number of instances solved are three with $K = 5$, while no instances are solved with $K = 10$.

4.3. The value of speed regulation

To quantify the value of speed regulation in searching for a conflict-free solution, we also solved our instances using our algorithm with all speeds fixed to the maximum value $\overline{v}$ and the time-based

Table 5 Computational results for instance of type B with tight time windows $\beta = 0.5, \theta = 0.3$

$\|V\|$	π	K	$\|R\|$	$TIME$	$NODES$	$SUCC$	ρ	LB/OPT	GAP_f	GAP_I	$TIME_I$
200	0	3	5	0.07	11	28	0.75	1.000000	0.000	1.734	0.07
				300.05	800345	2	2.00	0.997700	2.237	2.462	0.14
			10	1.53	740	28	2.11	0.999907	0.000	2.490	1.26
				300.05	198399	1	1.00	0.999279	0.722	0.722	0.17
		5	5	1.42	1081	29	3.59	0.999954	0.000	2.500	0.85
			10	14.00	2188	23	6.87	1.000000	0.000	0.069	10.32
				300.05	103139	1	10.00	0.999996	0.004	0.004	5.85
	0.5	3	5	0.06	40	25	0.60	0.999680	0.000	9.562	0.05
				300.52	595882	5	3.40	0.998157	0.261	12.686	0.25
			10	7.95	3602	24	1.92	0.999260	0.000	10.179	2.64
				300.05	272931	6	3.17	0.995233	2.540	14.518	1.09
		5	5	1.20	1236	24	2.67	0.995049	0.000	11.729	0.78
				300.05	550542	5	2.40	0.998534	1.450	10.041	0.25
			10	13.97	2979	16	4.63	0.997346	0.000	5.050	11.49
				300.05	209570	5	6.20	0.997735	0.138	0.332	64.57
400	0	5	5	0.89	199	27	2.07	0.999836	0.000	13.076	0.85
				300.06	255834	3	3.00	0.996791	2.556	22.986	0.71
			10	2.75	401	25	3.88	0.999967	0.000	3.819	2.32
				300.05	236848	5	6.00	0.991224	9.006	18.568	3.92
		10	5	9.87	1880	23	5.74	0.998515	0.000	1.240	9.48
				300.05	102799	4	7.50	0.984593	14.717	27.541	11.37
				0.01	1	1	0.00	–	–	–	–
			10	63.30	1911	12	8.25	1.000000	0.000	0.000	63.30
	0.5	5	5	0.35	308	27	1.11	0.999515	0.018	4.343	0.27
				300.05	359574	3	4.00	0.995520	3.656	13.779	0.78
			10	6.55	1751	21	3.05	0.999072	0.000	7.015	5.23
				300.05	269085	9	4.56	0.999344	0.403	8.842	5.56
		10	5	21.43	6445	17	4.06	0.997639	0.000	3.478	13.50
				300.05	331370	5	5.80	0.997404	1.054	5.825	2.31
				0.01	1	1	0.00	–	–	–	–
			10	43.61	6910	5	5.20	0.999025	0.000	0.666	9.24
				300.10	26529	1	8.00	0.997064	0.002	0.971	189.18
Average				46.86	45000	409	3.34	0.998728	0.395	6.047	6.54

branching disabled (Algorithm H), as well as with the approach proposed by Corréa et al. (2007), where conflicts are avoided by operating the vehicles at maximum constant speed and allowing the vehicles to wait at the end of edges (Algorithm C). As stated in Section 1, the master problem proposed by Corréa et al. (2007) is a Constraint Programming model to determine the requests assigned to each AGV (assignment decisions), the order in which they are visited (sequencing decisions) and the arrival times at each destination node (scheduling decisions). The second stage Mixed Integer Programming model checks if there exists a conflict-free solution satisfying the constraints of the master problem (conflict-free routing decisions). The algorithm stops as soon as a conflict-free solution is determined. Since our algorithm makes scheduling and conflict-free routing decisions, we solved each instance with the approach proposed by Corréa et al. (2007), where assignment and sequencing variables are fixed according to what is prescribed by S_k, with $k = 1, \ldots, K$. The results are reported in Table 7 for type A instances, and in Tables 8, 9 and 10 for B instances for wide and tight time windows, respectively. The column headings of these tables are

Table 6 Computational results for instance of type B with tight time windows $\beta = 0.5, \theta = 0.5$

$\|V\|$	π	K	$\|R\|$	$TIME$	$NODES$	$SUCC$	ρ	LB/OPT	GAP_f	GAP_I	$TIME_I$
200	0	3	5	0.51	677	23	0.96	0.999808	0.000	0.072	0.51
				0.01	1	7	0.00	–	–	–	–
			10	0.28	151	8	2.13	0.999987	0.001	0.165	0.26
				300.05	318805	1	4.00	0.999966	0.034	4.540	0.29
				0.01	1	21	0.00	–	–	–	–
		5	5	0.90	415	15	3.20	0.999995	0.000	1.093	0.87
				300.05	828228	1	2.00	0.999109	0.892	0.892	0.25
				0.01	1	14	0.00	–	–	–	–
			10	17.34	2215	1	12.00	1.000000	0.000	0.000	17.34
				300.05	28861	1	9.00	0.999551	0.449	0.595	10.36
				0.01	1	26	0.00	–	–	–	–
	0.5	3	5	0.08	58	19	0.63	0.999873	0.000	1.139	0.07
				300.05	661959	4	3.25	0.995888	0.906	1.320	0.39
				0.01	1	7	0.00	–	–	–	–
			10	0.30	237	8	1.75	0.999883	0.025	2.696	0.22
				300.05	542396	1	11.00	0.999251	0.531	1.201	2.53
				0.01	1	21	0.00	–	–	–	–
		5	5	1.04	1153	13	2.92	0.996334	0.000	5.508	0.71
				300.05	606412	3	3.00	0.998975	0.948	32.934	0.33
				0.01	1	14	0.00	–	–	–	–
			10	11.12	4292	1	6.00	0.998972	0.000	7.449	1.82
				300.05	89671	1	15.00	0.998871	0.429	0.584	23.47
				0.01	1	26	0.00	–	–	–	–
400	0	5	5	0.52	156	19	1.74	0.997378	0.000	1.199	0.49
				300.05	311447	2	1.00	0.996523	3.501	24.041	0.14
				0.01	1	9	0.00	–	–	–	–
			10	0.96	295	3	2.67	0.996197	0.000	4.237	0.45
				0.01	1	27	0.00	–	–	–	–
		10	5	41.32	7560	13	5.46	0.995279	0.001	0.303	40.33
				300.05	36785	1	5.00	0.984943	15.287	15.287	3.11
				0.01	1	15	0.00	–	–	–	–
			10	0.01	1	30	0.00	–	–	–	–
	0.5	5	5	0.23	206	16	1.00	0.998893	0.000	3.600	0.21
				300.05	370201	5	4.60	0.997406	1.799	22.469	1.98
				0.01	1	9	0.00	–	–	–	–
			10	1.55	351	3	2.00	0.999990	0.000	0.081	0.33
				0.01	1	27	0.00	–	–	–	–
		10	5	13.71	4371	6	4.00	0.994386	0.000	1.725	13.70
				300.05	252039	7	5.29	0.994837	3.682	4.154	25.79
				0.01	1	15	0.00	–	–	–	–
			10	0.01	1	30	0.00	–	–	–	–
Average				50.37	61315	175	2.61	0.998146	0.378	3.234	5.22

self-explanatory with the following exceptions. The column IT reports the number of iterations of the algorithm proposed by Corréa et al. (2007). The column N is the number of instances either solved to optimality or for which a feasible solution has been found, using fixed speeds. The columns $SUCC_S$ and $SUCC_H$ present the same statistic for our algorithm and Algorithm H, respectively. Similarly, columns ES_H and ES_C present the savings in energy (in percent) yielded by the solutions found by our algorithm over those found by Algorithms H and C, respectively. Finally, column W reports the overall average waiting time in the Corréa et al. (2007) solutions.

Table 7 The value of speed regulation on instances of type A

$\|V\|$	π	K	$\|R\|$	$TIME$	IT	N	$W(\%)$	$ES_C(\%)$	$SUCC_S$	$SUCC_H$	$TIME_H$	$ES_H(\%)$
200	0	3	5	17.33	1.00	27	16.20	73.63	30	30	0.09	63.85
			10	41.79	1.00	24	20.78	72.04	30	30	0.35	63.93
		5	5	30.04	1.00	24	12.62	74.79	30	28	0.77	63.82
			10	75.55	1.00	22	12.11	74.98	30	24	24.84	63.92
	0.5	3	5	14.48	1.00	21	15.19	74.05	30	27	0.06	64.11
			10	32.18	1.00	17	19.34	72.90	30	29	0.18	63.96
		5	5	25.63	1.00	16	14.07	74.35	30	25	0.23	64.28
			10	55.67	1.00	6	20.20	72.30	30	27	20.08	64.21
400	0	5	5	64.96	1.00	24	16.15	73.65	30	30	0.40	63.88
			10	182.47	1.00	19	15.17	73.96	30	30	12.57	64.01
		10	5	145.79	1.00	14	10.10	75.60	30	24	24.17	63.79
			10	292.60	1.00	5	8.49	76.30	30	9	51.41	63.91
	0.5	5	5	56.10	1.00	20	15.25	74.09	30	28	0.14	63.89
			10	135.11	1.00	9	17.85	73.26	30	26	1.10	64.08
		10	5	110.00	1.00	1	8.73	76.15	30	15	3.14	64.22
			10	252.00	1.00	1	8.30	76.32	30	14	61.12	64.20
Average				66.72	1.00	250	15.37	73.94	480	396	8.99	63.99

Table 8 The value of speed regulation on instances of type B with wide time windows, i.e. $\beta = 0, \theta = 0.1$

$\|V\|$	π	K	$\|R\|$	$TIME$	IT	N	$W(\%)$	$ES_C(\%)$	$SUCC_S$	$SUCC_H$	$TIME_H$	$ES_H(\%)$
200	0	3	5	17.19	1.00	27	16.20	71.12	30	30	0.10	60.30
			10	41.50	1.00	24	20.78	67.91	30	30	0.36	58.57
		5	5	29.79	1.00	24	12.62	72.57	29	28	0.88	60.62
			10	75.50	1.00	22	12.11	71.32	27	24	24.67	58.65
	0.5	3	5	14.29	1.00	21	15.19	71.71	30	27	0.06	60.42
			10	32.00	1.00	17	19.34	68.95	30	29	0.18	58.58
		5	5	25.69	1.00	16	14.07	72.03	29	23	0.22	60.53
			10	55.50	1.00	6	20.20	68.39	28	26	9.15	58.58
400	0	5	5	65.75	1.00	24	16.15	71.39	30	30	0.39	60.60
			10	184.11	1.00	19	15.17	70.14	29	29	13.47	58.70
		10	5	146.43	1.00	14	10.10	73.43	21	21	24.49	60.01
			10	295.20	1.00	5	8.49	72.91	19	9	50.62	58.51
	0.5	5	5	57.60	1.00	20	15.25	71.95	30	28	0.15	60.51
			10	135.67	1.00	9	17.85	69.60	29	26	0.79	58.63
		10	5	111.00	1.00	1	8.73	73.67	25	15	3.50	60.23
			10	256.00	1.00	1	8.30	73.25	26	13	43.48	58.55
Average				67.06	1.00	250	15.37	70.97	442	388	7.61	59.57

The results shown in Tables 7–10 have several implications. First, they indicate that the instances become harder to solve for fixed speeds. In particular, the number of type A instances solved, either to optimality or for which a feasible solution has been identified, is equal to 250 and 396 for Algorithms C and H, respectively. This is in stark contrast to the 480 feasible instances identified when speed regulation is allowed. More remarkably, our approach takes 2.47 seconds to determine a feasible solution, whilst Algorithms C and H take 66.72 and 8.99 seconds on average, respectively. The results are similar with type B instances with wide time windows, i.e., $\beta = 0$ and $\theta = 0.1$. Such gain in terms of success rate decreases with tighter time windows. In particular, the number of instances with tight time windows solved by Algorithms C and H is equal to 265 and 365 for $\theta = 0.3$, and 97 and 144 for $\theta = 0.5$. In this case, speed regulation allows to solve 409 and 175 instances

Table 9 The value of speed regulation on instances of type B with tight time windows ($\beta = 0.5, \theta = 0.3$)

$\|V\|$	π	K	$\|R\|$	$TIME$	IT	N	$W(\%)$	$ES_C(\%)$	$SUCC_S$	$SUCC_H$	$TIME_H$	$ES_H(\%)$
200	0	3	5	17.22	1.00	27	15.87	65.92	30	29	0.08	52.61
			10	50.04	1.35	23	17.64	61.30	29	28	0.64	47.06
		5	5	31.41	1.11	27	13.78	66.85	29	29	0.72	53.04
			10	72.79	1.00	19	15.33	61.75	24	22	16.87	47.49
	0.5	3	5	15.17	1.13	24	15.23	66.04	30	27	0.06	52.72
			10	68.12	3.12	17	20.84	59.64	30	24	0.17	47.28
		5	5	56.16	3.53	19	16.74	65.43	29	24	0.26	53.08
			10	48.80	1.00	5	19.87	59.90	21	21	25.68	46.66
400	0	5	5	68.26	1.00	27	14.50	66.58	30	30	0.22	53.00
			10	167.13	1.00	16	16.67	61.32	30	30	18.00	46.64
		10	5	137.94	1.00	17	10.58	67.86	27	27	38.21	52.84
				0.30	1.00	1	–	–	1	1	0.01	–
			10	292.22	1.00	9	8.79	65.76	12	7	97.64	48.02
	0.5	5	5	71.77	1.45	22	16.37	60.40	30	24	0.14	53.27
			10	130.44	1.00	9	14.08	56.59	30	22	0.78	46.45
		10	5	142.00	1.50	4	13.01	65.78	22	15	1.31	52.66
				0.28	1.00	1	–	–	1	1	0.01	–
			10			0			6	6	45.54	48.63
Average				73.56	1.42	265	15.44	64.15	409	365	10.35	50.36

Table 10 The value of speed regulation on instances of type B with tight time windows ($\beta = 0.5, \theta = 0.5$)

$\|V\|$	π	K	$\|R\|$	$TIME$	IT	N	$W(\%)$	$ES_C(\%)$	$SUCC_S$	$SUCC_H$	$TIME_H$	$ES_H(\%)$
200	0	3	5	20.67	1.33	15	17.06	59.85	23	22	0.08	43.68
				0.21	1.00	14	–	–	7	8	0.01	–
			10	40.86	1.00	7	16.69	55.68	9	7	0.24	39.61
				0.24	1.00	23	–	–	21	23	0.01	–
		5	5	30.30	1.00	10	10.07	64.93	16	15	0.75	47.35
				0.21	1.00	18	–	–	14	15	0.01	–
			10	81.00	1.00	1	11.86	57.73	2	1	5.97	38.04
				0.21	1.00	28	–	–	26	26	0.01	–
	0.5	3	5	21.29	1.93	14	16.48	59.92	23	21	0.06	43.10
				0.22	1.00	14	–	–	7	9	0.01	–
			10	32.00	1.00	4	16.67	56.22	9	6	0.18	40.37
				0.21	1.00	23	–	–	21	24	0.01	–
		5	5	50.63	3.00	8	12.77	64.31	16	12	0.25	47.56
				0.21	1.00	18	–	–	14	18	0.01	–
			10			0			2	1	6.17	37.97
				0.23	1.00	28	–	–	26	26	0.01	–
400	0	5	5	72.15	1.08	13	15.18	66.28	21	21	0.46	44.60
				0.25	1.00	13	–	–	9	9	0.01	–
			10	155.00	1.00	1	7.62	62.93	3	3	0.80	38.84
				0.29	1.00	28	–	–	27	27	0.01	–
		10	5	140.43	1.00	7	12.16	67.75	14	14	15.78	46.40
				0.25	1.00	19	–	–	15	15	0.01	–
			10	0.25	1.00	30	–	–	30	30	0.01	–
	0.5	5	5	62.70	1.20	10	18.22	60.71	21	12	0.12	46.54
				0.26	1.00	13	–	–	9	18	0.01	–
			10	127.00	1.00	2	24.63	49.88	3	2	0.24	37.22
				0.25	1.00	28	–	–	27	28	0.01	–
		10	5	167.00	1.80	5	9.04	65.48	13	7	0.84	46.53
				0.25	1.00	19	–	–	15	23	0.01	–
			10	0.25	1.00	30	–	–	30	30	0.01	–
Average				57.76	1.42	97	14.96	61.86	175	144	1.99	44.46

for $\theta = 0.3$ and $\theta = 0.5$, respectively. Savings in computation time are more remarkable. When speed regulation is not permitted, a first feasible solution for instances with tight time windows is determined in 73.56 and 10.35 (57.76 and 1.99) seconds on average for $\theta = 0.3$ ($\theta = 0.5$). This is in contrast to the average computation time of 6.54 (5.22) seconds needed to solve the problem using speed regulation. Speed regulation also allows to achieve remarkable savings in energy consumption across the four sets of instances. Indeed, as shown in Tables 7–10, when compared to Algorithm C, our procedure achieves an average energy saving equal to 73.94%, 70.97%, 64.15% and 61.86%, respectively. As far as Algorithm H is concerned, average energy savings amount to 63.99%, 59.57%, 50.36% and 44.46%, respectively. These results suggest that, whilst a great majority of the energy savings reported are achieved by speed regulation, additional improvements are also possible due to the methodology proposed here, ranging from approximately 10% to just over 17%.

5. Conclusions

In this paper we described a path and speed optimization algorithm for the conflict-free pickup and delivery problem under time windows, which arises in automated material handling systems, port terminals and controlled airspace. We proposed a branch-and-bound algorithm in which a lower bound is obtained by relaxing the arc capacity constraints. The relaxed problem is separable, for which an optimal solution can be found in quadratic time. If the optimal solution of the relaxed problem violates the arc capacity constraints, then a conflict is detected through a feasibility-check procedure. Finally, a branching procedure determines a set of space-based and time-based branching constraints cutting off infeasible solutions associated to the detected conflicts. Computational experiments showed that the algorithm can successfully solve instances of up to 400 vertices in grid graphs within a few seconds of computational time. When compared with a state-of-the-art approach (Corréa et al. 2007), our procedure generated an average of 42% additional feasible solutions and reduced the computation time by an order of magnitude. Furthermore, our algorithm achieved a remarkable energy saving (around 70%).

There are two future research directions that we propose. The first is to extend this study to account for tracks of sufficiently small length, as would be the case in small warehouses, where arc speeds cannot be assumed to be independent and for which acceleration and deceleration profiles have to be considered. The second is to consider disturbances to a solution as it is executed (e.g., increased travel times). In this case the model proposed here can be used in a rolling-horizon scheme to ensure that the feasibility of the solution is retained.

Acknowledgments

This work was partly supported by Ministero dell'Istruzione, dell'Università e della Ricerca (MIUR) of Italy. This support is gratefully acknowledged (PRIN Project 2015JJLC3E_005 "Transportation and Logistics Optimization in the Era of Big and Open Data"). The authors also thank the Associate Editor and two anonymous referees for their valuable comments.

References

Gerardo Berbeglia, J-F Cordeau, Irina Gribkovskaia, and Gilbert Laporte. Static pickup and delivery problems: a classification scheme and survey. *TOP*, 15(1):1–31, 2007.

Alan Branch. *Elements of port operation and management.* Springer Science & Business Media, London, 2012.

Ayoub Insa Corréa, André Langevin, and Louis-Martin Rousseau. Scheduling and routing of automated guided vehicles: A hybrid approach. *Computers & Operations Research*, 34(6):1688–1707, 2007.

Guy Desaulniers, André Langevin, Diane Riopel, and Bryan Villeneuve. Dispatching and conflict-free routing of automated guided vehicles: An exact approach. *International Journal of Flexible Manufacturing Systems*, 15(4):309–331, 2003.

Yvan Dumas, Jacques Desrosiers, and François Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54(1):7–22, 1991.

Pooya Farahvash and Thomas O. Boucher. A multi-agent architecture for control of agv systems. *Robotics and Computer-Integrated Manufacturing*, 20:473–483, 2004.

Chiara Fiori, Kyoungho Ahn, and Hesham A Rakha. Power-based electric vehicle energy consumption model: Model development and validation. *Applied Energy*, 168:257–268, 2016.

Lars Magnus Hvattum, Inge Norstad, Kjetil Fagerholt, and Gilbert Laporte. Analysis of an exact algorithm for the vessel speed optimization problem. *Networks*, 62(2):132–135, 2013.

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

Eric Mueller, Confesor Santiago, Andrew Cone, and Todd Lauderdale. Effects of UAS performance characteristics, altitude, and mitigation concepts on aircraft encounters and delays. *Air Traffic Control Quarterly*, 21(1):93–123, 2013. URL `http://arc.aiaa.org/doi/abs/10.2514/atcq.21.1.93`.

Murali Nandula and S.P. Dutta. Performance evaluation of an auction-based manufacturing system using coloured Petri nets. *International Journal of Production Research*, 38(10):2155–2171, 2000.

Tatsushi Nishi and Yuki Tanaka. Petri net decomposition approach for dispatching and conflict-free routing of bidirectional automated guided vehicle systems. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 42(5):1230–1243, 2012.

Tatsushi Nishi, Yuichiro Hiranaka, and Ignacio E. Grossmann. A bilevel decomposition algorithm for simultaneous production scheduling and conflict-free routing for automated guided vehicles. *Computers & Operations Research*, 38(5):876–888, 2011.

Simona Onori, Lorenzo Serrao, and Giorgio Rizzoni. *Hybrid Electric Vehicles Energy Management Strategies*. SpringerBriefs in Electrical and Computer Engineering, Control, Automation and Robotics. Springer, Berlin, 2016.

Davide Ronzoni, Roberto Olmi, Cristian Secchi, and Cesare Fantuzzi. Agv global localization using indistinguishable artificial landmarks. In *Proceedings of the 2011 IEEE International Conference on Robotics and Automation*, pages 287–292, Shanghai, 2011. IEEE.

Mohammad Saidi-Mehrabad, Saeed Dehnavi-Arani, Farshid Evazabadian, and Vahid Mahmoodian. 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, 2015.

Bhaba R. Sarker and Sanjay S. Gurav. Route planning for automated guided vehicles in a manufacturing facility. *International Journal of Production Research*, 43(21):4659–4683, 2005.

David Sinriech and Shraga Shoval. Landmark configuration for absolute positioning of autonomous vehicles. *IIE Transactions*, 32:613–624, 2000.

Ron van Duin and Harry Geerlings. Estimating CO_2 footprints of container terminal port-operations. *International Journal of Sustainable Development and Planning*, 6(4):459–473, 2011.

Jianbin Xin, Rudy R. Negenborn, and Gabriel Lodewijks. Hierarchical control of equipment in automated container terminals. In *Computational Logistics: 4th International Conference, ICCL 2013, Copenhagen, Denmark, September 25–27, 2013. Proceedings*, pages 1–17, Berlin, Heidelberg, 2013. Springer.

Kwo Young, Caisheng Wang, Le Yi Wang, and Kai Strunz. Electric vehicle battery technologies. In Rodrigo Garcia-Valle and João A. Peas Lopes, editors, *Electric Vehicle Integration into Modern Power Networks*, Power Electronics and Power Systems, pages 15–56. Springer New York, 2013.