Efficient Liveness Assessment for Traffic States in Open, Irreversible, Dynamically Routed, Zone-Controlled Guidepath-based Transport Systems

Spyros Reveliotis and Tomáš Masopust

Abstract—**Open, irreversible, dynamically routed, zone-controlled guidepath-based transport systems model the operation of many automated unit-load material handling systems that are used in various production and distribution facilities. An important requirement for these systems is to preserve the system liveness – i.e., the ability of each system agent to reach any location of the underlying guidepath network – by blocking those traffic states that will result in deadlock and/or livelock. The remaining set of traffic states are characterized as "live". The worst-case computational complexity of the decision problem of assessing the state liveness in the considered class of transport systems is an open issue. As a first contribution of this work, we identify an extensive subclass of these traffic states, defined through the topology of an abstracting graphical representation of the "traffic state" concept, for which the corresponding problem of liveness assessment admits a polynomial solution, and we present the relevant algorithm for this assessment. But the development of the aforementioned results has also led to a new methodological framework for representing and analyzing the qualitative dynamics of the considered transport systems with respect to the reachability and the liveness problems that are the focus of this work. This framework can enable an effective and efficient (but maybe not polynomial-complexity) resolution of the state liveness even for those traffic states that do not belong in the primary state class that is considered in this work; we highlight this additional possibility in the closing part of the paper.**

Keywords: Guidepath-based traffic systems; traffic liveness and its enforcement; deadlock avoidance; discrete event systems

I. Introduction

This work deals with reachability problems in a class of transport systems that are modeled as a fleet of "agents" circulating over the nodes and/or the edges of a connected graph that is known as the underlying "guidepath network". At any time point, the nodes and/or the edges of the guidepath network are allocated exclusively to their occupying agents, and an agent can advance to a neighboring location only when this location is currently free. Furthermore, this advancement must be coordinated by a traffic controller, and it must adhere to an allocation protocol that is defined by the physical attributes of the guidepath network itself, the maneuverability of the circulating agents, and further safety considerations.

S. Reveliotis is with the School of Industrial & Systems Engineering, Georgia Institute of Technology, email: spyros@isye.gatech.edu. Tomáš Masopust is with the Institute of Mathematics, Czech Academy of Sciences and with the Department of Computer Science, Palacky University, Olomouc, Czechia, email: masopust@math.cas.cz. S. Reveliotis was partially supported by NSF grant ECCS-1707695 and T. Masopust by RVO 67985840 and the Czech Science Foundation grant GC19-06175J.

Guidepath-based transport systems of the type that were outlined in the previous paragraph have been used extensively for modeling the operations taking place in some classes of automated material handling systems (MHS) [1], [2], [3], [4], [5], [6], [7], [8] and in other robotics applications [9], [10], [11], [12]. They have also been used for the modeling and the analysis of the physical operations that implement the elementary computations that are supported in quantum computing [13], [14], and for the programming of the fancy animations that are supported by the current video game industry [15].

A primary concern in all the aforementioned applications is the coordination of the agent traffic over the underlying guidepath network in a way that maximizes some measure of the productivity of the underlying system, while ensuring safe and collision-free operations for the traveling agents. But some additional important concerns that can arise from the constricted nature of the underlying guidepath network and the limitations that it enforces upon the generated traffic, are (i) the assessment of the feasibility of the posed requirements for the various agent trips, and (ii) the control of the system traffic in a way that preserves this feasibility.

In fact, some of the earliest and most interesting investigations on the traffic that takes place in guidepath-based transport systems, concern some feasibility problems that are motivated by a popular game that is known as the "15-puzzle" and is defined as follows: Given 15 pebbles labelled uniquely with labels from 1 to 15 and located on a 4×4 grid, rearrange these pebbles in row-major order through a sequence of "pebble-sliding moves" that place into the single free vertex of the grid one of its neighboring pebbles. The work of [16] addressed a generalized version of the "15-puzzle" where $n-1$ uniquely labelled pebbles, placed on the vertices of an n-vertex biconnected graph G, must be re-arranged through a sequence of "sliding" moves to a given "target" configuration. Treating the different placements of the marked pebbles on the vertices of the underlying graph as permutations, and using permutation group theory [17], the work of [16] established that as long as graph G is not bipartite, the addressed generalization of the "15-puzzle" will always be feasible. On the other hand, if graph G is biconnected and bipartite, then, the "reachability" relationship that is defined on the pebble permutations, partitions the entire permutation set into two equivalence classes. The work of [18] extended the results of [16] to problem instances involving less than $n-1$ pebbles (and, therefore, more than one empty vertices in the supporting graph G), and provided a polynomial-time algorithm for assessing the feasibility of any given problem instance, and for constructing a "pebble-move" sequence for feasible problem instances. On the other hand, in [19], the results of [18] were customized and streamlined for the particular case where the involved graph is a tree.

In all of the works that were mentioned in the previous paragraph, the difficulty of the addressed reachability problems was defined by the need to relocate simultaneously all the system agents, under the restrictions on the agent motion that are defined by the presence of the remaining agents in the guidepath network. On the other hand, each agent can move freely from its current vertex in the guidepath network, to any neighboring vertex that is available. In particular, an agent can "undo" its last move into its current vertex, by executing the "reverse" move, assuming that its previous location is still free, and this "reversibility" of the agent motion is at the core of analyzing the dynamics of the resulting traffic through permutation group theory.

But there are many guidepath-based transport systems where the aforementioned reversibility of the agent motion will not hold. More specifically, in many guidepath-based transport systems that abstract the operations of unit-load automated MHS, like the automated guided vehicle (AGV) systems and the overhead monorail systems that are used in various production and distribution facilities [2], [20], the traveling vehicles maintain a sense of direction for their motion with respect to (w.r.t.) their longitudinal axis, that cannot be reverted, due to physical constraints or other safety considerations. Such guidepath-based transport systems will be characterized as "irreversible" in the following. Irreversible guidepath-based transport systems are susceptible to "deadlock" and "livelock"; a deadlock formation taking place in the context of an AGV system is depicted in Fig. 1.

Deadlocks and livelocks will prevent, or, more generally, restrict the future motion of the agents involved, and will impair the ability of these agents to complete their "mission" trips. Hence, an important task of any traffic controller that is deployed for these environments, is the preservation of the traffic *"liveness"*; i.e., the traffic controller must proactively prevent the development of deadlock and livelock by further restricting the admissibility of the possible agent moves in the underlying guidepath network, and preserve, in this way, the ability of every agent to reach every location of the underlying guidepath network.

The problems of (i) formally defining the notion of "liveness", and (ii) developing "liveness-enforcing supervisors (LES)" for the aforementioned transport systems, have been investigated more systematically within a group of the Discrete Event Systems (DES) community that deals with broader problems of complex resource allocation [3], [4], [5], [6], [7], [8]. These studies, and the corresponding results, have been substantially qualified, and facilitated, by some additional operational attributes of the underlying MHS, and of the guidepath-based transport systems that abstract these operations.

One of these attributes concerns the presence of a location in the underlying guidepath-network that will hold all those

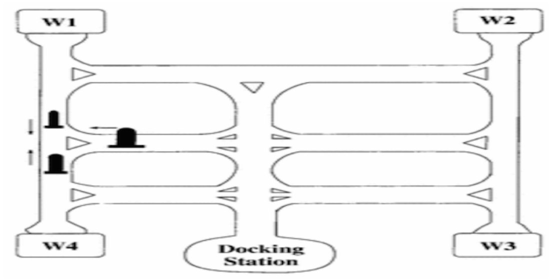


Fig. 1: An AGV deadlock involving three vehicles located at a junction of the underlying guidepath network: In the considered AGV systems, vehicles move through the edges – or "zones" – of the underlying guidepath network, being granted exclusive occupancy of these edges by a coordinating controller, one edge at a time. Furthermore, the system vehicles cannot reverse the direction of their motion in any given edge, and therefore, upon reaching a certain junction of the underlying guidepath network, the only way that they can advance is by transitioning to another free edge that is incident to the considered junction. But in the depicted case, all the edges that are incident to the considered junction are currently occupied by one vehicle heading towards this junction. Hence, all three vehicles will be permanently stalled at this junction.

agents that are not on an active trip; such a location is characterized as the "home" location of the network, and guidepath-based transport systems that possess such a "home" location are characterized as "open".

A second important attribute concerns the specification of the trips that are executed by the different agents. If the routes for these trips are completely defined upon the trip initiation, then, the corresponding routing scheme is characterized as "static". If, on the other hand, the agent trips are specified as a sequence of destinations that must be visited by the corresponding agents, and the agents are free to select their routes between two consecutive destinations in real-time, then, the resulting routing scheme is characterized as "dynamic".

Finally, in the corresponding DES literature, the locations that are allocated exclusively to the traveling agents, are represented by the edges of the guidepath network, and, in line with the relevant MHS terminology, they are referred to as the "zones" of this network. Also, the restriction of allowing no more than one agent at any given zone is known as "zone control".

This work deals with the notion of "liveness" and "liveness enforcing supervision (LES)" in the context of open, irreversible, dynamically routed, zone-controlled guidepath-based transport systems. As already mentioned, this sort of problems for the considered transport systems have been tackled in the past by a group of researchers in the DES community, who have tried to adapt to these problems some broader results concerning efficient LES synthesis for complex, sequential resource allocation systems; the works of [3], [4], [5], [7] are some characteristic examples of these endeavors. On the other hand, a more complete and systematic characterization of the notion of "liveness" for open, irreversible, dynamically routed, zone-controlled guidepath-based transport systems, was provided recently in [8]. That work abstracted the traffic dynamics of the considered transport systems through a pertinent Finite State Automaton (FSA) Φ, and it showed that, in the considered class of transport systems, liveness can be enforced, in a maximally permissive manner, by restricting the system operation to those states of the aforementioned FSA Φ that are co-reachable[1] to the state s_h where all agents are located in the "home" zone h. State s_h is known as the "home" state of FSA Φ, and states s of Φ that are co-reachable to state s_h are characterized as "live" in [8].

In [8] it was also shown that, for statically routed, open, zone-controlled guidepath-based transport systems, assessing the liveness of any given traffic state s of the aforementioned FSA Φ is an NP-complete problem in the strong sense. On the other hand, for open, irreversible, and dynamically routed, zone-controlled guidepath-based transport systems, the work of [8] was able to establish that liveness assessment for totally congested states – i.e., states s where every edge of the guidepath network is occupied by an agent – can be performed with linear worst-case computational complexity w.r.t. the size of the guidepath network. But the worst-case computational complexity of assessing the liveness of any arbitrary state s that comes from the class of open, irreversible, dynamically routed, zone-controlled guidepath-based transport systems, remains an open problem.

This work seeks to contribute to the aforementioned literature on guidepath-based transport systems, along the three following lines:

A) At a first, more immediate and more practical level, the work extends the results of [8] by identifying an additional class of traffic states in open, irreversible, and dynamically routed, zone-controlled guidepath-based transport systems that admits liveness assessment of polynomial computational complexity w.r.t. the size of the underlying guidepath network, and it is defined through an abstracting graphical representation of these states. For this class of states, it also provides the necessary algorithm for performing their liveness assessment.

B) The second contribution of the considered work is of a more methodological nature. More specifically, the technical results that have led to the developments that are claimed in item #1 above, also introduce a novel formal framework for representing and analyzing the qualitative – or "untimed" – dynamics of the considered guidepath-based traffic systems which is instrumental for these developments. In fact, the representational basis of this methodological framework was originally introduced in [6] for investigating issues related to the liveness of *closed*, irreversible, dynamically routed, zone-controlled guidepath-based transport systems (i.e., zone-controlled guidepath-based transport systems that do not possess a "home" location, and each agent circulates perpetually over the primary zones of the underlying guidepath network). But in this work we augment substantially the original developments of [6] by (i) enhancing the representational content of the framework itself, and (ii) complementing this representational capability with a number of operations on the maintained representation of the system state that facilitate analysis and inference w.r.t. the reachability and liveness properties of the underlying traffic systems.

C) Besides enabling all the technical developments that were claimed in item #1, the representational and analytical capabilities of the methodological framework that is delineated in item #2, can also support effective and efficient (but not necessarily polynomial-complexity) liveness assessment of traffic states that transcend the particular classes of traffic states and guidepath-based transport systems that are the primary focus of this work. The imposed space limitations for this document do not allow a complete coverage of these additional possibilities. But in the last part of the paper, we highlight these possibilities and we provide pointers to some recent work that has undertaken a more systematic treatment of these possibilities.

From an organizational standpoint, the rest of the paper is structured as follows: The next section provides a more formal characterization of the considered transport systems and their generated traffic, and it introduces the aforementioned FSA Φ that enables the "(state) liveness" characterizations of [8], which are at the core of this work. Section III introduces the alternative representation of the dynamics of FSA Φ that is borrowed from [6], and establishes certain properties for these dynamics that are necessary for the main results of the paper. These main results are presented in Section IV; namely, Section IV presents the class of traffic states that is the focus of this work, and the corresponding polynomial algorithm for the liveness assessment of these states. The technical developments of this section are further highlighted by some elucidating examples. Section V provides the discussion regarding the possible extension of the presented results so that they can effectively assess the liveness of all the traffic states that can arise in the considered class of open guidepath-based traffic systems, and even the liveness of the traffic states that arise in the closed counterparts of these systems. Finally, Section VI concludes the paper and suggests some directions for future work.

II. The class of guidepath-based transport systems considered in this work and the corresponding notion of state liveness

In this section, (i) first we provide a detailed description of the structure and the operation of the guidepath-based transport systems that are considered in the rest of this work, and subsequently (ii) we overview some results from [8] that concern the notion of "liveness" in this class of transport systems. We organize the corresponding material into two separate subsections.

A. A formal modeling of the considered transport systems

An instance of the particular sub-class of the guidepath-based transport systems considered in this work can be formally defined by a pair $(\mathcal{A}, G)$, where the elements of this pair denote, respectively, (a) the set of the system vehicles (or "agents") circulating in it, and (b) the guidepath graph $G = (V, E \cup \{h\})$ that is traversed by these agents.

[1]We remind the reader that, in a given FSA Φ, a state s is co-reachable to a state s' if and only if state s' is reachable from state s; i.e., there exists a transition sequence σ leading from state s to state s'.

Graph G is assumed to be undirected, connected, and with a minimum vertex degree of 2.[2] The edges $e \in E$ of G model the "zones" of the underlying guidepath network. These edges can be traversed by a traveling agent $a \in \mathcal{A}$ in either direction, and they can hold no more than one agent at a time. On the other hand, edge h models the "home" zone of the guidepath network. This edge is connected to the rest of the guidepath network through a single vertex (i.e., edge h is a self-loop of G), and it can hold an arbitrary number of agents that either have not initiated or have completed their assigned missions. Furthermore, in the following, we shall denote by v_h the vertex of graph G that is the single terminal vertex for the self-loop edge h. Finally, in the considered application context, it is also natural to assume that two vertices v_1, v_2 of graph G may be connected by more than one zones, and therefore, in stricter terms, graph G is actually a multi-graph; but this feature does not impact substantially our subsequent developments, and we shall keep referring to G as a graph in the sequel.

A "mission" trip for an agent $a \in \mathcal{A}$ is defined by a sequence of edges $\Sigma_a = \langle e_i \in E \setminus \{h\} \rangle$ that must be visited by agent a in the specified order.[3] Furthermore, edge h can be perceived as an implicit last edge in sequence Σ_a, a fact that signifies the requirement of retiring those agents a that have completed their mission trips to the "home" location.

While traversing an edge $e \in E$ with $e = \{v_i, v_j\}$, an agent a will have a certain direction of motion that will be indicated by the corresponding ordered pair (v_i, v_j) or (v_j, v_i). Furthermore, we stipulate that agents cannot switch the direction of their motion in the edges that are currently allocated to them; hence, an agent a entering edge $e = \{v_i, v_j\}$ from vertex v_i must leave this edge through vertex v_j, and vice versa.

An additional stipulation for the dynamics of the underlying traffic is that an agent a will move from its current edge e to a neighboring edge $e' \neq h$ only after it has been granted permission by the traffic controller, and such a permission can be granted by this controller only if the requested edge e' is free of any other agents. Besides preventing agent cohabitation in the different zones of the guidepath network, this last stipulation further implies that a set of agents cannot simultaneously swap their current locations.

B. The notions of "Liveness" and "State Liveness" in the considered transport systems

As remarked in the introductory section, the impossibility of edge-swapping among the traveling agents, when combined with the arbitrary topology of the underlying guidepath network, can be a source of *deadlock* and *livelock* in the considered class of transport systems [3], [6]. Such formations will prevent, or, more generally, restrict the future motion of the agents involved, and will impair the ability of these agents to complete their "mission" trips. Hence, an important task of the traffic controller is the preservation of the traffic *"liveness"*; i.e., the traffic controller must proactively prevent the development of deadlock and livelock by further restricting the admissibility of the "zone" allocations that are requested by the traveling agents.

In [8], it was shown that, in the considered traffic systems, "liveness" can be enforced in a maximally permissive manner by abstracting the operation of the underlying system through an FSA Φ, and restricting this operation in a subset of states of FSA Φ that are characterized as "live". In the rest of this subsection, we review these results of [8]. However, due to space considerations, the corresponding exposition is kept at the minimum set of concepts and results that are necessary for a systematic development of the subsequent results of this paper; the reader is referred to [8] for a more complete exposition of this material.

The FSA $\Phi = \langle S, Q, f, s_0, S_M \rangle$ abstracting the traffic dynamics of the considered transport systems: The aforementioned FSA Φ of [8] that enables the liveness characterizations that are necessary for this work, is defined as follows:

The state s of FSA Φ is defined by the two following elements: (i) The placement of the system agents $a \in \mathcal{A}$ on the edges of the guidepath network G. (ii) For agents $a \in \mathcal{A}$ that are not located at the "home" edge h, state s also encodes the direction of their motion in their allocated edges.

Clearly, the set of states, S, that results from all the possible placements of the agents $a \in \mathcal{A}$ on the edges of the guidepath graph G, is finite, and therefore, the considered automaton Φ is finite. Also, in the following, we shall use the function $\epsilon(\cdot; s) : \mathcal{A} \rightarrow (E \cup \{h\})$ to express the edge occupied by agent a at state s.

The event set Q that advances the state s of FSA Φ, is also finite, and it contains all those events q that advance a single agent $a \in \mathcal{A}$ from its current edge $\epsilon(a; s)$ to a free neighboring edge e'. These advancements must also be compatible with the direction of motion of the corresponding agent a on its current edge $\epsilon(a; s)$.

The state transition function $f : S \times Q \rightarrow S$ of the automaton Φ provides a formal representation of the transitional dynamics that are implied by the above definition of state s and the event set Q. Furthermore, following [21], we assume f to be a partial function that is defined only for those (s, q) pairs for which the corresponding state transition is feasible within the scope of the aforestated operational assumptions. We also extend f in the set $S \times Q^*$ in the natural manner, and we use the notation $R(s)$ to denote the states s' of Φ that are reachable from a given state s, through the dynamics that are defined by the extended function f; i.e., $\forall s' \in S,\ s' \in R(s) \iff \exists \sigma \in Q^* : s' = f(s, \sigma)$.

Finally, the initial state, s_0, and the set of marked states, S_M, for the considered FSA Φ are defined by setting $s_0 = s_h$ and $S_M = \{s_h\}$; as discussed in the introductory section, state s_h itself is defined as the "home" state of FSA Φ where all of the system agents are idling in the "home" zone h.

State liveness: In the operational context of FSA Φ, the notion of a "live state" can be defined as follows [8]:

Definition 1: A state $s \in S$ of FSA Φ is *live iff* the corresponding subspace $R(s)$ contains a strongly connected

[2]The imposed requirement of a minimal vertex degree of 2 is necessitated by the presumed irreversibility of the agent motion, since an agent a that reaches a vertex v of degree 1 will deadlock at that vertex.

[3]In order to obtain a more concrete feeling of these "mission" trips, the reader can think of an AGV that, setting out from the "home" edge h, must perform a sequence of transports, where each transport involves the pick up of some material from the zone that is represented by edge e_i in sequence Σ_a and the deposition of this material to the zone represented by edge e_{i+1}.

component $\Psi(s)$ that satisfies the following condition:

$$\forall(a,e) \in \mathcal{A} \times (E \cup \{h\}),\ \exists s' \in \Psi(s) : \epsilon(a; s') = e$$

□

Clearly, by driving, and eventually confining, the considered traffic system in the strongly connected component $\Psi(s)$ of Definition 1, we can establish the ability of bringing any agent $a \in \mathcal{A}$ to any edge $e \in E \cup \{h\}$ *ad infinitum*, which constitutes the essence of liveness for the considered traffic systems. On the other hand, initiating the operation of the considered transport system at a traffic state s where the corresponding subspace $R(s)$ does not contain a strongly connected component $\Psi(s)$ with the property that is specified in Definition 1, there will exist agent-zone pairs $(a, e) \in \mathcal{A} \times (E \cup \{h\})$ for which agent a will not be able to visit zone e in a repetitive manner; hence, the considered transport system will not be live. In [8], we also have the following result:

Theorem 1: In the class of open, zone-controlled guidepath-based traffic systems that are considered in this work, a state $s \in R(s_h)$ is live *iff* it is co-reachable to the "home" state s_h. □

As discussed in the introductory section, for irreversible, dynamically routed, open, zone-controlled guidepath-based transport systems, the computational complexity of assessing state liveness is an open issue. Nevertheless, in the next two sections of this paper, we shall use the result of Theorem 1 for developing an algorithm that can assess efficiently state liveness for a particular sub-class of traffic states of these transport systems. Furthermore, in Section V we also discuss a possible extension of the results of Sections III and IV in order to develop a streamlined (but maybe non-polynomial-complexity) algorithm for assessing the liveness of any traffic state of the considered class of transport systems.

III. An alternative representation of the dynamics of FSA Φ

In this section we provide an alternative representation of the qualitative dynamics modeled by the FSA Φ that was defined in the previous section. This new modeling paradigm is at the core of the algorithmic developments of Section IV regarding the assessment of the state-liveness condition of Theorem 1. The subsequent material was initially developed in [6] for the investigation of the notion of liveness in closed, irreversible dynamically routed guidepath-based transport systems, and it is based on the following graphical representation of the traffic state s that is employed by FSA Φ.

Representing the traffic state s as a labelled, partially directed digraph: The definition of state s for the FSA Φ that models the traffic dynamics of the considered guidepath-based transport systems, implies that state s can be naturally represented by means of a *labelled, partially directed graph*[4] *(PDG)* $\hat{G}(s)$ that is induced by state s and the undirected guidepath graph G. The vertex set of $\hat{G}(s)$ is the same with the vertex set of the original graph G. The undirected edges of $\hat{G}(s)$ are the unoccupied edges $e \in E$ of the guidepath graph G in the considered state s, plus the "home" edge h, which, as stated in Section I, constitutes a self-loop of G. The directed edges of PDG $\hat{G}(s)$ correspond to the edges $e \in E$ of G that are allocated to some agent in state s; the direction of this last set of edges in PDG $\hat{G}(s)$ is defined by the direction of motion of the corresponding agents that occupy these edges. Finally, each directed edge $e \in E$ is also labelled by the agent $a \in \mathcal{A}$ that occupies this edge in state s; in the following, we shall denote these labels by $l(e) \in \mathcal{A}$, and we can also use the notation $l(e) = null$ for the unoccupied, and therefore, undirected edges of $\hat{G}(s)$.[5]

Some useful structural concepts and properties of PDG $\hat{G}(s)$: Next we review some important additional concepts and properties regarding the PDG $\hat{G}(s)$, that were originally introduced in [6] and are essential for the efficient assessment of state liveness that is pursued in this work.

Given a PDG $\hat{G}(s)$, we define a *path* π in this graph as a sequence $\pi = \langle v_0, e_1, v_1, e_2, \ldots, e_n, v_n \rangle$, $n \geq 0$, where, for $i = 0, \ldots, n$, the elements v_i, belong to the vertex set V of $\hat{G}(s)$, and each element e_i appearing in this sequence is an edge connecting the vertices v_{i-1} and v_i. Furthermore, if an edge e_i is a directed edge in $\hat{G}(s)$, then its direction must be from vertex v_{i-1} to vertex v_i; hence, the sense of direction that is induced for path π by the ordering of its vertices v_i, $i = 0, \ldots, n$, is consistent with the direction of motion that is implied by the directed edges of the PDG $\hat{G}(s)$. A path π is *simple iff* all of its vertices are distinct. A *cycle* c of PDG $\hat{G}(s)$ has a structure similar to that of a simple path, but it contains at least one edge and the starting and the ending vertices, v_0 and v_n, are coinciding.[6] A *joint* between two cycles c and c' is a simple path π that belongs to both cycles. A *pass* between two cycles c and c' is a simple path π such that its first vertex lies on c, its last vertex lies on c', and all of the edges of π are undirected and do not belong on either c or c', or on any other (directed) cycle of PDG $\hat{G}(s)$. Finally, an edge e of the original guidepath graph G is (on) a *bridge* of this graph *iff* it does not belong on any of its cycles; hence, the removal of a bridge-edge disconnects the entire graph into two subgraphs.[7]

Example: We highlight the above definitions by means of the top part (part (a)) of Figure 2. This part depicts a state s of a guidepath-based transport system with a guidepath graph $G = (V, E)$ corresponding to the undirected graph that is induced by the depicted PDG $\hat{G}(s)$.[8] The agents $a \in \mathcal{A}$ that are not located on the "home" edge h in the considered state s are represented by the directed edges of the PDG $\hat{G}(s)$; this representation defines, both, the particular edge that is

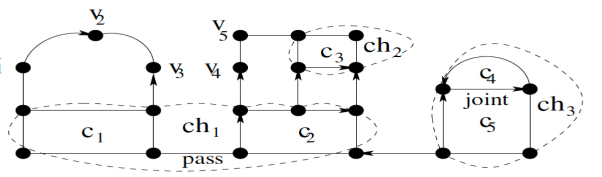
[4]As suggested by its name, a *partially directed graph (PDG)* contains both types of edges, undirected and directed. Undirected and directed graphs can be considered as special cases of a PDG where one of the two types of edges is missing, and all the structural concepts that are defined in the following extend to these special cases, as well.

[5]Since, in the co-reachability problem that defines "state liveness", all agents are destined to the "home" edge h, the labeling scheme that is defined by the function $l(e)$, $e \in E$ will not play any substantial role in the following.

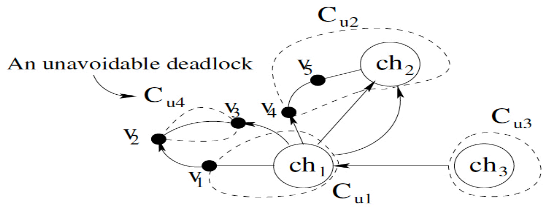
[6]Hence, according to this definition of the "cycle" concept, an edge that constitutes a "self-loop" (like the "home" edge h) is a cycle, but a single vertex is not.

[7]A "bridge"-edge of an undirected graph is also called a "cut-edge" in some part of the corresponding literature [22].

[8]The reader should notice that the "home" edge h has not been depicted in this figure, since its inclusion would complicate the accompanying discussion without adding anything substantial to it.



(a) A PDG $\hat{G}(s)$ and the "chain" structure that is recognized in it.



(b) The condensation $C(\hat{G}(s))$ of the above PDG $\hat{G}(s)$ and its u−connected components

Fig. 2: The content of this figure is adapted from [6], and it exemplifies the definitions and the technical results that are provided in Section III.

occupied by the agent in state s and the direction of its motion in this edge.

The considered PDG has five cycles annotated by $c_1, \ldots, c_5$; the reader can check that these are indeed the only closed paths of PDG $\hat{G}(s)$ that preserve the sense of direction for their directed edges. The (directed) edge labeled as "joint" in the figure constitutes a path belonging to both cycles c_4 and c_5. The edge marked as "pass" in the figure is a path that consists of undirected edges only, and links cycles c_1 and c_2 while possessing no common edges with any of these two cycles. On the other hand, the edge of PDG $\hat{G}(s)$ that links cycles c_2 and c_5 does not constitute a pass for these two cycles because it is directed (and, therefore, occupied by an agent). Finally, this last edge, and also the edge that constitutes the pass between cycles c_1 and c_2, are the only two bridges for the undirected graph G that is the guidepath network for the considered transport system. □

The concepts that are introduced in the next definition play a very central role in the subsequent developments.

Definition 2: A *chain ch* of PDG $\hat{G}(s)$ is the subgraph that is induced by the sequence $ch \equiv \langle c_1, \pi_2, c_2, \pi_3, \ldots, \pi_n, c_n \rangle$, $n \geq 1$, where: (i) $c_i,\ i = 1, \ldots, n$, are cycles; (ii) $\pi_i,\ i = 2, \ldots, n$, are simple paths; and (iii) each path π_i is a joint or a pass between cycles c_{i-1} and c_i.

Furthermore, two edges $e, e' \in E$ will be characterized as *chain-connected* (or, more simply, as *chained*) *iff* there exists a chain ch that contains both e and e'.

Finally, graph $\hat{G}(s)$ will be characterized as *chained iff* every two edges $e, e' \in E$ are chained. □

Example: According to Definition 2, each of the cycles $c_1, \ldots, c_5$ of the PDG $\hat{G}(s)$ that is depicted in part (a) of Figure 2, constitutes also a chain for this PDG. But this PDG also possesses the additional chains annotated by ch_1 and ch_3 in this figure. Chain ch_1 consists of cycles c_1 and c_2 linked by the corresponding pass that was discussed in the previous example, and chain ch_3 consists of the cycles c_4 and c_5 which are linked by the annotated joint. On the other hand, the depicted chain ch_2 comprises cycle c_3 only, which is the only remaining cycle that is not contained in the other two chains. □

More generally, it is easy to see that chain connectivity is symmetric and transitive, and therefore, we can consider the *maximal* chains of a given PDG $\hat{G}(s)$. The subgraphs of PDG $\hat{G}(s)$ that are induced by these maximal chains are characterized as the *chained components* of $\hat{G}(s)$. Furthermore, the PDG $\mathcal{C}(\hat{G}(s))$ that is obtained by replacing each of the chained components of $\hat{G}(s)$ by a simple vertex, is called the *condensation* of $\hat{G}(s)$. Vertices of $\mathcal{C}(\hat{G}(s))$ that correspond to chained components of $\hat{G}(s)$ will be characterized as the *macro-vertices* of the new PDG $\mathcal{C}(\hat{G}(s))$, while the remaining vertices of $\mathcal{C}(\hat{G}(s))$ will be characterized as *simple*.

Example: It should be clear from the discussion in the last example that ch_1, ch_2 and ch_3 constitute the chained components of the PDG $\hat{G}(s)$ of Figure 2. The condensation $\mathcal{C}(\hat{G}(s))$ that results from the reduction of each of these chained components to a single macro-vertex (with the same label), is the PDG that is depicted in part (b) of Figure 2. □

The next proposition highlights two important structural properties of the condensed PDG $\mathcal{C}(\hat{G}(s))$.

Proposition 1: By its construction, condensation $\mathcal{C}(\hat{G}(s))$ is an *acyclic* PDG. Furthermore, each path π in $\mathcal{C}(\hat{G}(s))$ that connects two different macro-vertices n_1 and n_2, contains a directed edge. □

Formal proofs for the results of the above proposition are provided in [6]. Here, we point out that the acyclic structure of $\mathcal{C}(\hat{G}(s))$ parallels the acyclic structure of the digraph that is obtained by "collapsing" the communicating classes of any connected digraph to single "macro-nodes". For the second part of Proposition 1, the reader should notice that if the considered path contained no directed edge, then it would constitute a pass between its terminal macro-vertices, and therefore, the chained components corresponding to the macro-vertices of the considered condensation $\mathcal{C}(\hat{G}(s))$ would not represent correctly the maximal chains of the original PDG $\hat{G}(s)$.

For the needs of the subsequent analysis, it is also pertinent to distinguish the subgraphs of $\mathcal{C}(\hat{G}(s))$ that (i) contain no directed edges, and (ii) are connected to the complement part of $\mathcal{C}(\hat{G}(s))$ by directed edges only.

Definition 3: An *undirected component* (or, more simply, *u-component*) in condensation $\mathcal{C}(\hat{G}(s))$ is a maximal connected subgraph $\mathcal{C}_u$ of $\mathcal{C}(\hat{G}(s))$ that contains no directed edges. The edges of $\mathcal{C}(\hat{G}(s))$ that point to $\mathcal{C}_u$ are the *inputs* of $\mathcal{C}_u$, and those that point away from $\mathcal{C}_u$ are the *outputs* of $\mathcal{C}_u$. $\mathcal{C}_u$ is a *source* if it has no inputs, and a *sink* if it has no outputs. Finally, $\mathcal{C}_u$ is a *complex* u-component if it contains a macro-vertex of the condensed PDG $\mathcal{C}(\hat{G}(s))$, and a *simple* u-component otherwise. □

Example: Part (b) of Figure 2 highlights also the u-components of the depicted condensation $\mathcal{C}(\hat{G}(s))$; these u-components are labelled $C_{u1}, \ldots, C_{u4}$ in the figure. The reader should notice that the further compression of each of these u-components into a single node in PDG $\mathcal{C}(\hat{G}(s))$, results in a reduced multi-graph that is (completely) directed. In particular, the directed edges of this last graph will be (C_{u1}, C_{u2}) with a multiplicity of 3, (C_{u1}, C_{u4}) with a multiplicity of 2, and (C_{u3}, C_{u1}) with a multiplicity of 1. $\square$

The next proposition results straightforwardly from all the above definitions. The results that are claimed in it can be verified in the graph that is depicted in part (b) of Figure 2, while formal proofs for these results can be found in [6].

Proposition 2: A u-component, $\mathcal{C}_u$, in condensation $\mathcal{C}(\hat{G}(s))$ is an undirected tree and it contains at most one macro-vertex of this condensation. Furthermore, the set of the u-components of $\mathcal{C}(\hat{G}(s))$ is partially ordered by the directed edges of this graph. $\square$

Chain capacity and its role in the analysis of the qualitative dynamics of the considered transport systems: In the following, we shall associate with each chain ch of the PDGs $\hat{G}(s)$, $s \in S$, an attribute that constitutes a notion of "capacity" for chain ch; a formal definition of this concept is as follows:

Definition 4: For any traffic state $s \in S$, the *capacity* of a chained component ch of the PDG $\hat{G}(s)$ will be denoted by $\zeta(ch)$ and will be set equal to the number of free edges of ch that are located on its cycles (or, equivalently, they are not "bridge" edges in the undirected graph that is induced by the chained component ch).

Furthermore, we shall use the notation ch_h to denote the chain of the PDG $\hat{G}(s)$ that contains the "home" edge h of the underlying guidepath graph G, and we shall set $\zeta(ch_h) = \infty$. $\square$

The significance of the notion of the "chain capacity" is revealed by the following propositions.

Proposition 3: If the condensation $\mathcal{C}(\hat{G}(s))$, of a traffic state $s \in S$, contains a *simple sink* u-component $\mathcal{C}_u$, then state s is not live. $\square$

Proof: According to Proposition 2, the considered u-component $\mathcal{C}_u$ is an undirected tree. Hence, it can be easily checked that the vehicles in the input edges of $\mathcal{C}_u$ are headed to some unavoidable deadlock. $\square$

Proposition 4: Consider a *complex sink* u-component $\mathcal{C}_u$ of the condensation $\mathcal{C}(\hat{G}(s))$ with macro-vertex ch, and an agent a located on one of the input edges of $\mathcal{C}_u$. The tree structure of $\mathcal{C}_u$ implies that, in the underlying state s, there is a unique path p of free edges through which agent a can access the macro-vertex ch. Then, the following two statements are true:

1) If $\zeta(ch) = 0$, then, any effort to advance agent a on an edge of the subgraph $G(ch)$ of the guidepath graph G corresponding to chain ch, through the aforementioned path p, will lead to an unavoidable deadlock.
2) If $\zeta(ch) \geq 1$, then, it is possible to advance agent a on an edge of the subgraph $G(ch)$, through path p, in a way that, at the resulting state s', the part of the PDG $\hat{G}(s')$ corresponding to the subgraph $G(ch)$ will be chained.

Proof: First, we establish the validity of the first statement in Proposition 4. For this, we start by noticing that while agent a is advancing on path p towards the macro-vertex ch, no agent a' that is located in (the chain corresponding to) the macro-vertex ch in the original state s can move on one of the free paths linking node ch with some input edge e of $\mathcal{C}_u$, since such a move will generate an unavoidable deadlock between agent a' and the agent a'' that is located on edge e.

To complete the proof of this part, we need to consider the following two cases:

Case 1: The acyclic subgraph $G(ch)$ that is induced by chain ch has no "bridge" edges. Then, since $\zeta(ch) = 0$ in the considered state s, all edges of chain ch are occupied by agents, and, as argued above, these agents cannot leave chain ch without causing deadlock. Hence, agent a cannot enter node ch.

Case 2: The acyclic subgraph $G(ch)$ that is induced by chain ch contains some path π consisting of "bridge" edges. First, the reader should notice that, according to Definition 2, path π must be a pass of chain ch, and, thus, every edge of π will be a free edge at the considered state s.

So, in this case, it is possible to accommodate agent a in some cycle c of ch, by relocating an agent a' in ch on path π. But the placement of agent a' on path π destroys the chained structure of ch, and induces a new complex sink u-component $\mathcal{C}'_u$ with agent a' being on an input edge of $\mathcal{C}'_u$. Furthermore, since the macro-vertex ch' of $\mathcal{C}'_u$ is obtained from ch, it does not have any free edges on its cycles. Hence, following a similar line of argumentation as in the case of the macro-vertex ch, we can conclude that the only way that ch' can accommodate one of the agents directed to it is by splitting itself through a path π' that consists of "bridge" edges of the corresponding undirected graph that is induced by ch'. Since, however, the number of such paths in ch is finite, it is clear that this chain splitting will unavoidably lead to a complex sink u-component with a chain that contains no such paths and it is fully allocated. But as we saw in the earlier parts of this proof, this last structure implies an unavoidable deadlock.

The second statement of Proposition 4 can be established through a line of argumentation that is very similar to that used in [6] for the establishment of Theorem 2 in that work; the reader is referred to [6] for the corresponding details. $\square$

Propositions 3 and 4 connect the PDG-based representation of the dynamics of the considered transport systems to the notions of "deadlock" and "(non-)live traffic state" that were defined in the previous sections. More specifically, the combination of these two propositions implies that a state $s \in S$ with its condensation $\mathcal{C}(\hat{G}(s))$ containing some sink u-components can be live only if (i) each of these u-components is complex, and (ii) its macro-vertex ch contains adequate capacity to absorb in it all the agents that are located on a path of $\mathcal{C}(\hat{G}(s))$ that links macro-vertex ch to some other upstream chain ch' of $\mathcal{C}(\hat{G}(s))$. Such an absorption will lead to a new state s' where chains ch and ch' are merged to a new chain ch''. Repeating the above remarks on state s', and recognizing the finiteness of the chains in PDG $\hat{G}(s)$, we can see that if the considered state s is live, then there will exist an event sequence σ leading from state s to a state $\tilde{s}$ where the corresponding PDG $\hat{G}(\tilde{s})$ is chained. The next theorem establishes that this co-reachability of the considered state s to a chained state $\tilde{s}$ provides, in fact, an alternative complete characterization of the liveness of s; this characterization will be at the core of the algorithmic assessment of state liveness that is developed in the next section.

Theorem 2: In the considered transport systems, a state $s \in S$ is live *iff* it is co-reachable to a state $\tilde{s}$ for which the corresponding PDG $\hat{G}(\tilde{s})$ is chained.

Proof: To establish the necessity of the new condition of Theorem 2 for state liveness, we notice that, according to

Theorem 1, every live state $s \in S$ is co-reachable to the "home" state s_h. According to the definition of the PDG $\hat{G}(s)$ that was provided at the beginning of this section, the PDG $\hat{G}(s_h)$ has no directed edges. This fact, when combined with Definition 2 and the additional fact that the minimal vertex degree of the guidepath graph G is 2, imply that the PDG $\hat{G}(s_h)$ is chained.

Next, suppose that state s is co-reachable to a state $\tilde{s}$ for which the corresponding PDG $\hat{G}(\tilde{s})$ is chained. If $\tilde{s} = s_h$, then, s is live. Otherwise, consider an agent $a_1 \in \mathcal{A}$ with $\epsilon(a_1; \tilde{s}) \neq s_h$. Through an argumentation similar to that pursued in the proof of Theorem 2 in [6], we can establish that there exists a state $s_1 \in R(\tilde{s})$ such that (i) $\epsilon(a_1; s_1) = h$, and (ii) the PDG $\hat{G}(s_1)$ is chained. Furthermore, the infinite buffering capacity of the "home" edge h implies that we can obtain the sought state s_1 without having to relocate the agents that are on edge h at state $\tilde{s}$; i.e., $\forall a \in \mathcal{A},\ \epsilon(a; \tilde{s}) = h \Longrightarrow \epsilon(a; s_1) = h$.

But then, a repetitive invocation of the above result, together with the finiteness of the agent set $\mathcal{A}$, imply that the considered state s is co-reachable to state s_h, and therefore, live. $\square$

We also notice, for completeness, that the unique chain of any chained state $s \in S$ will contain the "home" edge h; i.e., this chain will be the corresponding chain ch_h of s and it will have infinite capacity (c.f. Definition 4).

The digraph $\mathcal{U}(\hat{G}(s))$: In the next section, we shall work with a further abstraction of the condensation $\mathcal{C}(\hat{G}(s))$ that is obtained by replacing each u-component of $\mathcal{C}(\hat{G}(s))$ by a single vertex; this graph will be denoted by $\mathcal{U}(\hat{G}(s))$, and it should be clear from its definition that it is a directed acyclic (multi-)graph (DAG) (c.f. also the discussion in the last part of the example on Figure 2 that was presented in the earlier parts of this section).

Furthermore, each vertex v of DAG $\mathcal{U}(\hat{G}(s))$ will be associated with a *capacity* $\chi(v)$ that is defined as follows: For the vertices v of $\mathcal{U}(\hat{G}(s))$ that correspond to simple vertices of the original guidepath graph G, as well as for those vertices v of $\mathcal{U}(\hat{G}(s))$ that correspond to simple u-components of $\mathcal{C}(\hat{G}(s))$, the corresponding capacity $\chi(v)$ is set equal to zero. On the other hand, a vertex v of $\mathcal{U}(\hat{G}(s))$ representing a complex u-component $\mathcal{C}_u$ of $\mathcal{C}(\hat{G}(s))$, will have its capacity $\chi(v)$ set equal to the capacity $\zeta(ch)$ of the chained component that constitutes the unique macro-vertex of $\mathcal{C}_u$.

A special node of DAG $\mathcal{U}(\hat{G}(s))$ is the node containing the "home" edge h and the corresponding chain ch_h. This node will be denoted by n_h, and, according to the previous definitions, we shall also have $\chi(n_h) = \infty$.

Finally, an even more compact representation of the DAG $\mathcal{U}(\hat{G}(s))$ that is particularly convenient for the algorithmic developments that are pursued in the next section, can be obtained as follows:

- This representation recognizes as the *"(major) nodes"* of DAG $\mathcal{U}(\hat{G}(s))$ those vertices that (i) either correspond to a complex u-component, or (ii) have a degree larger than 2 (and, therefore, are "branching" vertices in DAG $\mathcal{U}(\hat{G}(s))$).
- Furthermore, it replaces each simple path π that connects a major nodal pair (n_1, n_2) and contains only non-major vertices of $\mathcal{U}(\hat{G}(s))$ as interior vertices, by a single directed edge (n_1, n_2) weighted by the number of edges in path π; in the following, we shall denote the weight of such an edge (n_1, n_2) by $w(n_1, n_2)$, and unless stated otherwise, we shall assume that the considered DAGs $\mathcal{U}(\hat{G}(s))$ are encoded according to this more compact representation.

Algorithm 1 An efficient algorithm for determining the condensation $\mathcal{C}(\hat{G}(s))$ of any given PDG $\hat{G}(s)$ – borrowed from [6].

Input: The PDG $\hat{G}(s)$
Output: PDGs $\hat{\mathcal{D}}_i,\ i = 1, \ldots, k$, corresponding to the maximal chains of the input PDG

```
Convert the input PDG $\hat{G}(s)$ to a digraph $\mathcal{D}$, by replacing each undirected edge $e$ of $\hat{G}(s)$ with two directed edges $e'$ and $e''$ of opposite directions;
Extract the strongly connected components, $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_k$, of digraph $\mathcal{D}$;
for $i := 1$ to $k$ do
    Convert the digraph $\mathcal{D}_i$ to the PDG $\tilde{\mathcal{D}}_i$, by replacing each edge pair $\{e', e''\}$ introduced in Step 1 with a single undirected edge $e$;
    Remove iteratively all one-degree vertices and their incident edges from PDG $\tilde{\mathcal{D}}_i$, until no such vertex is left, obtaining the corresponding PDG $\hat{\mathcal{D}}_i$;
end for
return the PDGs $\hat{\mathcal{D}}_i,\ i = 1, \ldots, k$;
```

Complexity considerations: Concluding the developments of this section, we also notice that, for any given state $s \in S$, the corresponding condensation $\mathcal{C}(\hat{G}(s))$ can be obtained with a linear computational cost w.r.t. the size of the PDG $\hat{G}(s)$ that represents state s. An algorithm for this computation has been provided in [6], and it is re-stated in this document as Algorithm 1 for the reader's convenience; the complete derivation and justification of the algorithm logic can be found in [6].

On the other hand, the capacity $\zeta(ch_i)$ for each maximal chain $ch_i,\ i = 1, \ldots, k$, that appears in $\mathcal{C}(\hat{G}(s))$, can be obtained by further executing the following two steps:

1) First we find the bridge edges of the undirected graph D_i that is induced by the corresponding PDG $\hat{\mathcal{D}}_i$ in the output of Algorithm 1.
2) Once these edges have been identified, then, we can set the chain capacity $\zeta(ch_i)$ equal to the number of the remaining free edges in the PDG $\hat{\mathcal{D}}_i$.

It is easy to see that the complexity of the above computation is determined by the task of identifying the bridges of the undirected graph D_i. This last task can be performed efficiently through the algorithm of [23]; the worst case complexity of this algorithm is $O(|V_i| + |E_i|)$, where V_i denotes the set of vertices of the graph D_i and E_i denotes the set of its edges.

Finally, once the condensation $\mathcal{C}(\hat{G}(s))$ and the capacity function $\zeta(\cdot)$ have been computed, the DAG $\mathcal{U}(\hat{G}(s))$ and the corresponding capacity function $\chi(\cdot)$ can be obtained from those two elements in a straightforward manner.

IV. The main results

In this section we focus on a class of traffic states of the considered transport systems that satisfy the following condition:

Condition 1: The undirected graph that is induced by DAG $\mathcal{U}(\hat{G}(s))$ is a tree.[9]

For this class of states, we provide an algorithm for assessing their liveness that has polynomial worst-case computational complexity w.r.t. the size of the underlying guidepath graph G. This algorithm is based on the state-liveness characterization of Theorem 2, and it constitutes an iterative "greedy" scheme that, at each iteration, tries to identify with polynomial effort a set of agent advancements that will result in the merging of two or more chains in the graphical representation of the system state that is maintained by the algorithm. For any initial state s, these iterations will either result in a chained state s' or they will reach a point where it will be possible to recognize the non-liveness of state s.

Furthermore, in order to facilitate a more systematic exposition of the presented results, we organize the subsequent developments in two subsections, with the first subsection functioning as a "preamble" to the main algorithmic developments that are presented in the second subsection. More specifically, the first subsection defines a "pre-processing" phase that should be executed at each of the main iterations of the presented algorithm. The computation that is pursued during this pre-processing phase (a) will recognize and resolve some "easy cases", and for the remaining cases, (b) it will reduce the original decision problem to an equivalent one where the induced state s' satisfies certain structural properties for the corresponding DAG $\mathcal{U}(\hat{G}(s'))$. Furthermore, a second part of this first subsection also defines some elementary operations on DAG $\mathcal{U}(\hat{G}(s'))$ that will constitute some of the main "building blocks" of the presented algorithm. The second subsection provides the complete algorithm for assessing the liveness of a given state s that satisfies Condition 1. This section also establishes the polynomial complexity of the derived algorithm w.r.t. the size of the guidepath graph G of the underlying transport system.

A. Preamble

A first processing stage in the iterations of the presented algorithm: We start this subsection by presenting some simplifying steps for the considered algorithm. These steps should be executed at the beginning of each major iteration of the algorithm. Furthermore, as we shall see, the execution of these steps guarantees a certain structure for the PDGs $\hat{G}(s)$, and the corresponding DAGs $\mathcal{U}(\hat{G}(s))$ that are eventually processed at each major iteration of the algorithm. This structure is important because it facilitates the detailed definition of the algorithm and the analysis of its correctness.

[9]We remind the reader that an undirected graph is a tree *iff* it is connected and acyclic [22]. Also, we notice that a necessary and sufficient condition for the presence of states satisfying Condition 1 in a given transport system of the type considered in this work, is the presence of (paths of) "bridge" edges in the underlying guidepath networtk G. In such a case, Condition 1 will be satisfied by a given state s when each maximal 2-edge-connected component of G – i.e., the maximal components of G that are obtained by removing all of its "bridge" edges – belongs in a single chained component of the corresponding PDG $\mathcal{C}(\hat{G}(s))$.

The proposed simplifying steps that were mentioned in the previous paragraph recognize the fact that, at any given state s, there might be some agents $a \in \mathcal{A}$ that can be advanced to the "home" edge h without disturbing the remaining agents. More specifically, we are looking for agents $a \in \mathcal{A}$ that can reach the "home" edge h using only the free edges of the PDG $\hat{G}(s)$ and their currently allocated edge $\epsilon(a; s)$. The detection of these agents can be performed by a simple reachability analysis on the subgraph that is defined by the free edges of $\hat{G}(s)$, and the complexity of the involved computation is polynomial w.r.t. the size of $\hat{G}(s)$. Clearly, the advancement of the aforementioned agents to edge h increases the available free edges of the underlying guidepath network, and therefore, it enhances the prospects of the remaining agents to reach the "home" edge h. Hence, if the original state s is live, then the state s' that results from the proposed advancements should also be live. And if s' is not live, then, s cannot be live either.

In fact, the simplifying step that was outlined in the previous paragraph should be performed in an iterative manner, terminating only when either all agents $a \in \mathcal{A}$ have been brought to edge h, or a state $\tilde{s}$ is reached where no agent a with $\epsilon(a; \tilde{s}) \neq h$ can be brought to h through a path consisting only of free edges in $\tilde{s}$ and the edge $\epsilon(a; \tilde{s})$. In the first case, we have established the liveness of the considered state s.[10] In the second case, we need to proceed with the further execution of the considered algorithm.

Finally, for the detailed specification of the subsequent steps of the presented algorithm, it is important to notice that the state $\tilde{s}$ obtained through the aforementioned iterations, will have a DAG $\mathcal{U}(\hat{G}(\tilde{s}))$ where the node n_h will be a "source" node. Unless indicated otherwise, all the DAGs $\mathcal{U}(\hat{G}(\tilde{s}))$ that we shall consider in the following, will be assumed to possess this particular property regarding node n_h.

Some elementary operations defined on DAG $\mathcal{U}(\hat{G}(\tilde{s}))$: As outlined in the discussion that precedes Theorem 2 in Section III, the computation of an event sequence that leads from a live state s to the "home" state s_h, can be perceived as a sequence of simple-path clearances in the underlying condensation graphs that result to the merging of two (or maybe more) of their nodal chains. This sort of clearances will constitute the main "building blocks" of the presented algorithm. We formalize the corresponding operations through the following definition.

Definition 5: Consider a DAG $\mathcal{U}(\hat{G}(s))$ that possesses the structural properties that were defined in the previous paragraph, and an edge (n_1, n_2) connecting two major nodes of this DAG. Furthermore, let $w(n_1, n_2)$ denote the corresponding weight of edge (n_1, n_2) and $\chi(n_1),\ \chi(n_2)$ denote the nodal capacities. Then, we have the following definitions:

1) The merging of the (chains corresponding to) nodes n_1 and n_2 is *feasible* iff $\chi(n_2) \geq w(n_1, n_2)$. The execution of a feasible merger will substitute the edge (n_1, n_2) in the underlying DAG $\mathcal{U}(\hat{G}(s))$ with a single node n of capacity $\chi(n) = \chi(n_1) + \chi(n_2) - w(n_1, n_2)$.

[10]In fact, the detection of the liveness of state s through the iterative computation that is outlined in these paragraphs is essentially the logic that has been proposed in [3] for adapting Dijkstra's Banker's algorithm to the considered transport systems.

2) The merging of nodes n_1 and n_2 is a *"producer"* iff $\chi(n) \geq \chi(n_2)$, where n denotes the node that will result from this merging. Otherwise, it is a *"consumer"*.

The first part of Definition 5 is motivated by Definition 4, the result of Proposition 4, and the discussion that accompanied that proposition. For a complete understanding of the second part, we further notice that for a feasible merging of a nodal pair (n_1, n_2), it will also hold that $\chi(n) \geq \chi(n_1)$, since $\chi(n) = \chi(n_1) + \chi(n_2) - w(n_1, n_2)$ and $\chi(n_2) \geq w(n_1, n_2)$; hence, when $\chi(n) \geq \chi(n_2)$, as stipulated in this part of Definition 5, the new node n that results from the considered merger will have no less capacity than any of the two nodes that it replaces in the underlying condensation. But then, this merging operation can only enhance the feasibility of the potential mergers that are defined by the remaining edges of the underlying condensation graph, and therefore, we have the following result:

Proposition 5: Consider the DAG $\mathcal{U}(\hat{G}(s))$ of some traffic state s that satisfies Condition 1, and let the edge (n_1, n_2) of this DAG define a feasible merger that is also a producer. Then, the state s' that corresponds to the execution of this merger will be live *iff* the original state s is live.

Proof: Clearly, if s' is live, it is co-accessible to the home state s_h and, therefore, the original state s is live, as well. The fact that the liveness of s implies the liveness of s' results from the remarks that precede the statement of this proposition. □

The practical implication of Proposition 5 is that any feasible *"producer"* merger in the condensations that are maintained by the considered algorithm, can be executed immediately without compromising the correctness of the overall computation. It is also possible to extend the notion of a "producer" merger and the result of Proposition 5 so that they address the clearance of an entire path $\langle n_1, n_2, \ldots, n_k \rangle$ in DAG $\mathcal{U}(\hat{G}(s))$ and the collapse of this path into a new single node n. The next proposition defines this operation and establishes its correctness in the context of the considered algorithm.

Proposition 6: Consider a DAG $\mathcal{U}(\hat{G}(s))$ of some traffic state s that satisfies Condition 1, and a path $\pi = \langle n_1, n_2, \ldots, n_k \rangle$[11] in it. Furthermore, let $\chi(n_i)$ denote the capacity of node n_i, for $i = 1, \ldots, k$, and $w(n_i, n_{i+1})$ denote the weight associated with edge (n_i, n_{i+1}). Finally, suppose that path π satisfies the following three sets of conditions:

1) $\forall i = 2, \ldots, k-1,\ \sum_{j=2}^{i} \chi(n_j) - \sum_{j=2}^{i} w(n_{j-1}, n_j) < 0$
2) $\sum_{j=2}^{k} \chi(n_j) - \sum_{j=2}^{k} w(n_{j-1}, n_j) \geq 0$
3) $\forall i = 2, \ldots, k,\ \sum_{j=1}^{i} \chi(n_j) - \sum_{j=1}^{i-1} w(n_j, n_{j+1}) \geq \chi(n_i)$

Then, path π defines a *feasible (generalized) "producer" merger* and it can be substituted in DAG $\mathcal{U}(\hat{G}(s))$ by a single node n with capacity $\chi(n) = \sum_{j=1}^{k} \chi(n_j) - \sum_{j=1}^{k-1} w(n_j, n_{j+1})$. Furthermore, the traffic state s' that will result from this substitution will have the same "liveness" status as the original state s.

Proof: First, we notice that the conditions in items #1 and #2 in Proposition 6 characterize the feasibility of the considered merger. More specifically, these conditions imply that the nodal capacities in each of the subpaths $\pi_i \equiv \langle n_1, \ldots, n_i \rangle$, $i = 2, \ldots, k-1$, are not adequate for clearing the edges of these subpaths from their occupying agents, but this clearance is possible once node k is added.

On the other hand, the conditions of item #3 establish that the merger defined by each subpath π_i, $i = 2, \ldots, k$, is a potential "producer" w.r.t. the corresponding terminal node n_i (irrespective of the feasibility of these mergers). Next, we show that these last conditions, when considered together with the conditions of items #1 and #2, further imply that the merger resulting from the collapse of path $\pi \equiv \pi_k$ is a (feasible) "producer" for all nodes n_j of path π; i.e., $\forall j = 1, \ldots, k,\ \chi(n) \geq \chi(n_j)$.

To establish the above result for node n_1, we notice that $\chi(n) = \sum_{j=1}^{k} \chi(n_j) - \sum_{j=1}^{k-1} w(n_j, n_{j+1}) = \chi(n_1) + \sum_{j=2}^{k} \chi(n_j) - \sum_{j=1}^{k-1} w(n_j, n_{j+1}) \geq \chi(n_1)$, where the last inequality is due to the condition of item #2 in Proposition 6. For nodes n_i, $i = 2, \ldots, k-1$, the combination of the corresponding inequalities in items #1 and #3 of Proposition 6 implies that $\chi(n_1) \geq \chi(n_i)$, and we already showed that $\chi(n) \geq \chi(n_1)$. Finally, the inequality $\chi(n) \geq \chi(n_k)$ is obtained directly from item #3 in Proposition 6 by setting $i = k$.

With the feasibility and the "producer" nature of the considered merger well established, we can also argue the last part of Proposition 6 – i.e., that the state s' that will result from this merger, will have the same "liveness" status with the original state s – as in the proof of the corresponding result in Proposition 5. □

In the next subsection, the results of Propositions 5 and 6 will be embedded in a complete algorithm for resolving state liveness for those traffic states s that satisfy Condition 1.

B. A polynomial-complexity algorithm for assessing the liveness of states that satisfy Condition 1

A first positioning and motivation of the presented results: Without any loss of generality, in the following discussion of this subsection, we shall consider that the tree mentioned in Condition 1 is "rooted" at node n_h.[12] Furthermore, under the working assumptions regarding state s that were stated in the earlier parts of this section, every edge incident to node n_h in DAG $\mathcal{U}(\hat{G}(s))$ will have a direction leading away from this node. Consider a path $\pi = \langle n_h, n_1, \ldots, n_k \rangle$ originating from node n_h in DAG $\mathcal{U}(\hat{G}(s))$. Then, since the starting node of this path has infinite capacity, the merger corresponding to the collapse of the path into a single node n will always be a "producer", according to the relevant definitions of the previous subsection. Hence, such a merger can always be performed by the considered algorithm, as long as it is feasible, and it will lead to a new state s' with the same liveness status as the original state s and a smaller number of chains in the corresponding PDG $\hat{G}(s')$.

Next, we focus on the case where the potential merger that is defined by each path π emanating from node n_h, is not

[11] Since we are dealing with tree structures, in the rest of this section we have opted to represent a path π by listing only its nodes and omitting the interconnecting edges.

[12] We remind the reader that node n_h of DAG $\mathcal{U}(\hat{G}(s))$ is the node that contains the "home" edge h of the guidepath network, and therefore, its corresponding capacity is infinite.

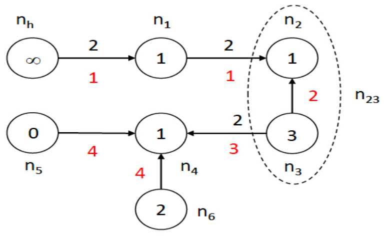


Fig. 3: A DAG $\mathcal{U}(\hat{G}(s))$ that is used for the motivation and the demonstration of the algorithmic developments of Section IV-B.

feasible. The situation is demonstrated in Figure 3. In the DAG that is depicted in this figure, the numbers within each node define the corresponding capacities, and the black numbers on the DAG edges are the labels $w(n_1, n_2)$ that were introduced in the previous subsection; for edges where these numbers are not reported, they are assumed to have the "default" value of '1'.

The reader can easily verify that, in the depicted case, it is not possible to clear the directed path emanating from node n_h from the agents located on its edges by simply utilizing the free capacity of the path nodes. But, on the other hand, it might be possible to generate further capacity for the nodes of such a path by performing a series of mergers on the subtrees that hang from these nodes. For instance, in the DAG of Figure 3, the edge (n_3, n_2) is a feasible "producer" merger, and the node n_{23} that results from this merger has capacity $\chi(n_{23}) = \chi(n_2) + \chi(n_3) - w(n_3, n_2) = 1 + 3 - 1 = 3$. Furthermore, the combined capacity of nodes n_1 and n_{23} is equal to 4, and therefore, the merger defined by the path $\langle n_h, n_1, n_{23}\rangle$ is feasible. Next, we provide a complete algorithm for detecting and effectively exploiting all these additional possibilities.

A "layering" structure on the DAG $\mathcal{U}(\hat{G}(s))$ that is employed by the presented algorithm: The presented algorithm will seek to identify and execute constructive mergers that will provide the maximal possible capacity that is attainable at each node n of the considered DAG $\mathcal{U}(\hat{G}(s))$. The corresponding computation will advance from the leaves of the "tree" structure that is associated with DAG $\mathcal{U}(\hat{G}(s))$, towards the root (which is node n_h). To further systematize (and also motivate) this nodal processing, we need to introduce an additional "layering" structure on the considered DAG $\mathcal{U}(\hat{G}(s))$. This structure will assign a "layer" number to each node and edge of the DAG $\mathcal{U}(\hat{G}(s))$ according to a recursive scheme that is defined as follows:

Definition 6: Consider the DAG $\mathcal{U}(\hat{G}(s))$ of a traffic state s that satisfies Condition 1. The "layering" structure imposed on this DAG by the algorithm that is presented in this subsection, is defined by the following recursion:

1) Nodes belonging into layer 1 are node n_h and every other node n that is reachable from node n_h through a directed path of DAG $\mathcal{U}(\hat{G}(s))$. Also, the edges on all those paths are labeled as "layer 1" edges.
2) Assuming that layers $1, \ldots, i$ are well defined, layer $i+1$ is defined as follows:
    a) If $i+1$ is an odd number, layer $i+1$ contains all nodes n of DAG $\mathcal{U}(\hat{G}(s))$ that are reachable from some node n' of layer i through paths consisting of edges that do not belong in any of the layers $1, \ldots, i$; the edges of all these paths are also labeled as "layer-$(i+1)$" edges.
    b) If $i+1$ is an even number, layer $i+1$ contains all nodes n of DAG $\mathcal{U}(\hat{G}(s))$ that are co-reachable to some node n' of layer i through paths consisting of edges that do not belong in any of the layers $1, \ldots, i$; the edges of all these paths are also labeled as "layer-$(i+1)$" edges.

Example: For the DAG of Figure 3, the layers defined by the recursion of Definition 6 are indicated by the edge labels that are annotated in red. On the other hand, in order to avoid an "over-loading" of this figure, we have omitted the corresponding labeling of the DAG nodes. In the considered case, the only nodes that are accessible from node n_h through a directed path, are nodes n_1 and n_2, and these two nodes together with node n_h define the first layer of the considered DAG. In order to define layer 2, we notice that the only node that is co-accessible to any of the nodes n_h, n_1 and n_2 through edges not belonging to layer 1, is node n_3; hence layer 2 contains node n_3 and edge (n_3, n_2). Also, since the only node reachable from node n_3 through edges not belonging in layers 1 and 2 is node n_4, layer 3 contains node n_4 and the corresponding edge (n_3, n_4). Finally, the co-accessible nodes to node n_4 through edges not belonging in layers 1, 2 and 3, are nodes n_5 and n_6; these nodes and the corresponding edges define layer 4. Since every node and edge of the considered DAG have been assigned to a layer at this point, the definition of the corresponding layering structure is complete. □

From the presented layering structure to the proposed algorithm: The definition of the above layering structure for the considered DAGs is motivated by an intention to capture the orientation of the agent motion on the various edges of these DAGs w.r.t. their "root" node n_h. More specifically, edges with an odd "layer" number are occupied by agents that are heading away from node n_h in the underlying "tree" structure. On the other hand, edges with an even "layer" number are occupied by agents that are moving in the direction of node n_h. This understanding subsequently suggests the following logic for the processing of any given "leaf" node in the considered DAGs:

1) Processing of a "leaf" node n that belongs in an odd layer: In this case, node n is connected to its parent node n' by an edge (n', n). The only way that the agents on edge (n', n) can advance, is by being absorbed in node n in the spirit of Proposition 4. Hence, if the merger that is defined by the edge (n', n) is feasible, it is executed by the considered algorithm, and subsequently the algorithm proceeds to its next major iteration, that will repeat the entire logic that is described in this subsection to the state s' that results from the aforementioned merger. If, on the other hand, the merger that is defined by edge (n', n) is not feasible, the algorithm infers the non-liveness of the considered state s.

2) Processing of a "leaf" node n that belongs in an even layer: In this case, node n is connected to its parent

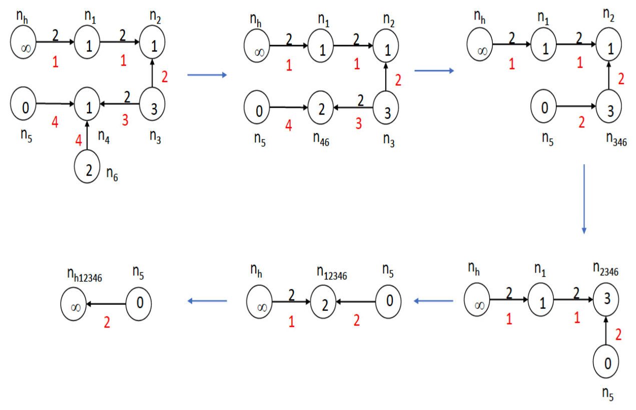


Fig. 4: The execution of the algorithm that is presented in Subsection IV-B on a traffic state s with its DAG $\mathcal{U}(\hat{G}(s))$ being the DAG of Figure 3.

node n' by an edge (n, n'). If this edge defines a feasible merger that is also a "producer", then the algorithm will execute the corresponding merger, obtaining a state s' with a smaller number of nodes in the corresponding DAG $\mathcal{U}(\hat{G}(s'))$ and with improved accessibility to the nodal capacities (c.f. Proposition 5). Furthermore, the algorithm will start a new major iteration focusing on the processing of the obtained state s'. More generally, for a given "leaf" node n that belongs in an even layer, the algorithm will try to identify and execute a generalized feasible "producer" merger that is defined by an entire directed path π emanating from this node.[13] On the other hand, if it is not possible to find such a feasible merger, the node will be marked as "processed", and the algorithm will proceed to process another unprocessed node of DAG $\mathcal{U}(\hat{G}(s'))$ that is either a "leaf" node or an interior node with all of its children already processed.

Example: Next, we demonstrate the execution of the above logic on the example DAG of Figure 3. The two "leaf" nodes in this case are nodes n_5 and n_6; any of them can be used as a starting point for the execution of our algorithm. Hence, starting with node n_5, we see that it belongs in an even layer, and its parent node is node n_4. The merger defined by the edge (n_5, n_4) is feasible, but the resulting node will have a capacity of $1 + 0 - 1 = 0$, and therefore, this merger is not a "producer". Therefore, node n_5 is declared "processed", and we proceed with the processing of node n_6. This node also belongs in an even layer, and it has node n_4 as its parent node. In this case, the merger defined by the edge (n_6, n_4) is feasible and "producer"; the node n_{46} resulting from this merger has capacity $\chi(n_{46}) = 1 + 2 - 1 = 2$. Also, we shall denote the state that results from this merger as s_1, and the corresponding DAG, $\mathcal{U}(\hat{G}(s_1))$, is depicted in Figure 4.

In DAG $\mathcal{U}(\hat{G}(s_1))$, the only "leaf" node is node n_5. This node still belongs in an even layer, its parent node is n_{46}, and the merger that is defined by edge (n_5, n_{46}) is feasible but not a "producer". Hence, the algorithm proceeds to process node n_{46}, which is the only interior node that has all of its children already processed. This node belongs in an odd layer. Its parent node is node n_3, and the edge (n_3, n_{46}) defines a feasible merger. The algorithm executes this merger, and the resulting node is node n_{346} with capacity $\chi(n_{346}) = 2{+}3{-}2 = 3$. Also, the state that results from this merger will be denoted as state s_2, and the corresponding DAG $\mathcal{U}(\hat{G}(s_2))$ is also depicted in Figure 4.

In DAG $\mathcal{U}(\hat{G}(s_2))$, the only "leaf" node is node n_5, with node n_{346} as its parent node. Both of these two nodes are in layer 2. Once again, edge (n_5, n_{346}) defines a feasible but "consumer" merger. Hence, node n_5 is declared "processed", and the algorithm proceeds to process its parent node n_{346}. The parent node of n_{346} is node n_2, and the edge (n_{346}, n_2) defines a feasible "producer" merger. The execution of this merger returns the node n_{2346} with capacity $\chi(n_{2346}) = 1 + 3 - 1 = 3$. Furthermore, the resulting state will be denoted by s_3, and the corresponding DAG $\mathcal{U}(\hat{G}(s_3))$ is depicted in Figure 4.

In DAG $\mathcal{U}(\hat{G}(s_3))$, the only "leaf" node is again node n_5. This node is in layer 2, and the edge that connects this node to its parent node n_{2346} defines a feasible but "consumer" merger. Hence, the algorithm proceeds to process node n_{2346} which is a layer-1 node. The parent node in this case is node n_1, and the edge (n_1, n_{2346}) defines a feasible merger. The node resulting from this merger is node n_{12346} with capacity $\chi(n_{12346}) = 3 + 1 - 2 = 2$. Also, the resulting state will be denoted as s_4, and the DAG $\mathcal{U}(\hat{G}(s_4))$ is depicted in Figure 4.

DAG $\mathcal{U}(\hat{G}(s_4))$ also has node n_5 as the only "leaf" node. This node remains a layer-2 node, and its processing gives the same results as in the previous iterations. Hence, the algorithm proceeds to process node n_{12346}, which is in layer 1, and defines a feasible merger with its parent node n_h. The node n_{h12346} resulting from this merger has infinite capacity, the corresponding state is denoted as s_5, and the DAG $\mathcal{U}(\hat{G}(s_5))$ is depicted in Figure 4.

In DAG $\mathcal{U}(\hat{G}(s_5))$, the only node other than the "home" node n_{h12346} is node n_5, and the two nodes are connected through edge (n_5, n_{h12346}). But then, the liveness of state s_5 will be identified through the execution in this iteration of the "pre-processing" stage that was described in the first part of Subsection IV-A. Hence, the entire algorithm concludes declaring the liveness of the original state s that corresponds to the DAG of Figure 3.

Correctness and complexity analysis of the algorithm that is presented in this subsection: A complete statement of the algorithmic logic that was developed in the earlier parts of this subsection, is presented in the pseudo-code of Algorithm 2. Furthermore, the next two theorems establish the correctness of this algorithm and its polynomial complexity w.r.t. the size of the underlying transport system.

Theorem 3: When executed on a traffic state s that satisfies Condition 1, Algorithm 2 will terminate in a finite number of steps, assessing correctly the liveness of state s.

Proof: First, we notice that when applied on a traffic state s with a PDG $\hat{G}(s)$ satisfying Condition 1, Algorithm 2 will generate, through its various preprocessing and merging operations, a sequence of states $s_k, k = 0, 1, 2, \ldots,$ with

[13]The previous definitions and assumptions further imply that such a directed path π will be on the undirected path that leads from node n to the "root" node n_h in the underlying tree.

Algorithm 2 Assessing the liveness of traffic states s satisfying Condition 1 with polynomial computational complexity w.r.t. the size of the underlying guidepath network.

Input: The PDG $\hat{G}(s)$ of a traffic state s satisfying Condition 1.
Output: A binary variable $LIVE$ indicating whether state s is live or not.

```
$\mathcal{G} := \mathcal{U}(\hat{G}(s))$ of the considered state $s$;
$LIVE :=$ TRUE;
repeat
  $\mathcal{G} := PREPROCESS(\mathcal{G})$;
  IF $\mathcal{G}$ consists of a single node RETURN $LIVE$;
  Compute the layers of $\mathcal{G}$;
  Mark all nodes of $\mathcal{G}$ as "$UNPROCESSED$";
  $BREAK :=$ FALSE;
  while $\exists$ unprocessed nodes AND $\neg BREAK$ do
    Pick a node $n$ of $\mathcal{G}$ which is either a "leaf" node or has all its children processed;
    $n' := PARENT(n)$;
    if node $n$ is in an odd layer AND the merger defined by edge $(n', n)$ is feasible then
      Execute the merger defined by edge $(n', n)$ on DAG $\mathcal{G}$;
      $BREAK :=$ TRUE;
    else if node $n$ is in an odd layer AND the merger defined by edge $(n', n)$ is infeasible then
      RETURN $\neg LIVE$;
    else if node $n$ is in an even layer AND $\exists$ a path $\pi$ in DAG $\mathcal{G}$ emanating from $n$ that defines a generalized feasible "producer" merger then
      Execute the merger defined by path $\pi$ on DAG $\mathcal{G}$;
      $BREAK :=$ TRUE;
    else
      mark node $n$ as "$PROCESSED$";
    end if
  end while
until FALSE
```

$s_0 = s$ and the PDGs $\hat{G}(s_k)$ satisfying Condition 1. The last result can be established through an inductive argument on the index k that further discerns the different types of operations that lead from state s_k to state s_{k+1}; the corresponding details are quite straightforward, and they are left to the reader.

Next, we notice that each major iteration of Algorithm 2 that is defined by the "Repeat"-loop, will either result in the termination of the algorithm through one of the two "RETURN" commands that appear in it, or in the merging of two or more nodes of the DAG $\mathcal{U}(\hat{G}(s))$ that is computed at the beginning of the algorithm. Since each major iteration involves a finite computation, and the DAG $\mathcal{U}(\hat{G}(s))$ has a finite number of nodes, eventually the condition in the second line of the "Repeat"-loop will be satisfied, and the algorithm will terminate, indeed, in a finite number of steps.

The correctness of the liveness assessment for the input state s that is returned by the algorithm, can be argued as follows: First, we notice that the validity of the various merging operations that are performed during the execution of the algorithm, is guaranteed by Propositions 4–6. Furthermore, in the context of the presumed structure for DAG $\mathcal{U}(\hat{G}(s))$ that is implied by Condition 1, the generalized merger that is defined by the three conditions of Proposition 6, is the only type of merger for the nodes in the even layers of DAG $\mathcal{U}(\hat{G}(s))$ that is necessary for utilizing effectively any available nodal capacity in these nodes. Indeed, as discussed in the proof of Proposition 6, the conditions of the inequalities (1) and (2) in that proposition are necessary for characterizing the feasibility of the smallest possible such merger that starts at node n_1 and develops along the considered path. On the other hand, the relaxation of the inequalities of the third type in that proposition, let's say for some node i with $2 \leq i \leq k-1$, would imply that the merger defined by the path $\langle n_1, \ldots, n_i \rangle$ is a "consumer" that will result in a "capacity deficit" for node n_i. Hence, in this case, the execution of the merger that is defined by the sub-path $\langle n_i, \ldots, n_k \rangle$ instead of the merger that corresponds to the entire path $\langle n_1, \ldots, n_k \rangle$, will result in a node of higher capacity, and will enhance the ability of the algorithm to advance its computation towards the root node n_h on the corresponding path.

Also, by working all the aforementioned mergers from the leaves of the underlying tree towards the root, while utilizing the layering structure of Definition 6, the algorithm *does* account for the maximal capacity that can become available at every node before attempting the corresponding mergers. This fact, together with Propositions 3 and 4, ensure the correctness of the "NON-LIVE" outcome that might be returned by the algorithm.

To complete the "correctness" part of the proof, we also need to show that when the algorithm returns a "LIVE" outcome, the assessed state s is indeed live. This fact can be established through the following remarks:

One first possibility to reach a "LIVE" outcome is by reducing the entire DAG $\mathcal{U}(\hat{G}(s))$ during the initial execution of the preprocessing step of the algorithm. But then, the discussion provided in the "preamble" part of this section implies that the considered state s is live.

On the other hand, if the algorithm advances past this first preprocessing stage, then, at the state s' that will result from this preprocessing stage, the "home" node n_h is connected to the rest of the corresponding DAG $\mathcal{U}(\hat{G}(s'))$ through a set of edges that emanate from node n_h and belong to the first layer of this DAG. Furthermore, in order to obtain a "LIVE" outcome, every merger attempted on an odd-layer node n of the DAG $\mathcal{U}(\hat{G}(s'))$ must be successful (otherwise, the algorithm would return a "NON-LIVE" outcome). Therefore, the algorithm computation across all paths π of the underlying tree that corresponds to DAG $\mathcal{U}(\hat{G}(s'))$ will consist of sequences of (generalized) "producer" mergers performed on the even-numbered layers of DAG $\mathcal{U}(\hat{G}(s))$, interleaved with mergers corresponding to odd-layer nodes. This merging process will advance across each path π of the supporting tree until it hits the root node n_h. At this point, either the entire path π has been merged into node n_h, or, if there is any remaining part of the considered path π that has not been merged yet with node n_h, all the edges on this path will be pointing towards node n_h; in the second case, the remaining part of path π will be merged with node n_h during the execution of the preprocessing step in the next iteration. Figure 4 provides a concrete demonstration of the described computation. Furthermore, according to all the previous developments, this computation implies the existence of a vehicle-advancing event sequence $\sigma \in Q^*$ that will bring all vehicles $a \in \mathcal{A}$ to the "home" edge h; hence, the considered state s' is live, which further implies the liveness of the original state s. □

Theorem 4: Algorithm 2 has worst-case computational complexity $O(n^3)$, where n is the number of nodes of the guidepath network G of the underlying transport system.

Proof: We start by noticing that according to (i) the computational logic that is listed in Algorithm 1, and (ii) the accompanying discussion to that algorithm, that are provided in the closing part of Section III, the computational cost of constructing the DAG $\mathcal{U}(\hat{G}(s))$ from the input PDG $\hat{G}(s)$ is $O(|V|+|E|)$, and since $|E|$ is $O(|V|^2)$, this cost is eventually $O(n^2)$ according to the definition of n that is provided in the statement of Theorem 4.

Also, it is clear that the number of nodes of the DAG $\mathcal{U}(\hat{G}(s))$ can be no larger than n, the number of nodes of the guidepath network G of the considered transport system.

Furthermore, since the number of nodes of the DAG $\mathcal{U}(\hat{G}(s))$ is reduced at least by one at each iteration of the repeat-loop of Algorithm 2, the remark of the previous paragraph further implies that the number of executions of the while-loop of Algorithm 2 is upper-bounded by $n+(n-1)+\ldots+2 = O(n^2)$.

Finally, the worst-case computational cost of a single iteration of the while-loop of Algorithm 2 is defined by the cost of the search for a feasible generalized "producer" merger at some leaf node n of the current DAG $\mathcal{U}(\hat{G}(s))$. This search involves the testing of the three inequalities of Proposition 6 in an iterative manner, by advancing through the path that connects node n with the root node n_h, one node at a time; this advancement will proceed until such a feasible generalized "producer" merger is detected, or one of the required inequalities is violated. Hence, the overall search for such a merger is $O(n)$.

The result of Theorem 4 results from the straightforward combination of all the above remarks. □

V. Discussion

In this section, we shift attention to states s of the FSA Φ of Section II-B that will not satisfy Condition 1. It should be clear to the reader that all the results of Section IV-A, and also the liveness-preservation by any nodal merger of a "producer" nature w.r.t. the merged nodes, will hold for this set of states, as well. But the traffic dynamics that are defined by this new set of states, involve some additional possibilities that require a careful reconsideration of the concept of the "producer" merger and the evaluation of the capacity of the resulting node, and prevent the straightforward application of Algorithm 2 to this state set. Next, we briefly discuss those additional possibilities that we have identified as the major sources of further complexity for the considered liveness assessment problem when addressed in this broader class of traffic states, and we also outline an extension of Algorithm 2 that can effectively cope with this increased complexity.

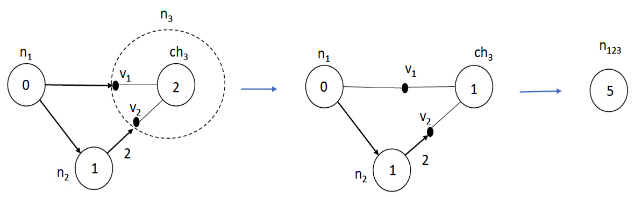


Fig. 5: A "cycle-generating" merger.

"Cycle-generating" mergers: One of the primary complications that are encountered when we move to states s of the FSA Φ that do not satisfy Condition 1, is depicted in Figure 5. The graph on the left side of Figure 5 depicts a DAG $\mathcal{U}(\hat{G}(s))$ where the node n_3 is expanded to reveal the internal structure of the complex u-component that corresponds to this node. The reader can check that all three directed edges in this DAG define feasible mergers of their incident nodes, but none of these mergers is a "producer". Nevertheless, executing the merger that is defined by edge (n_1, n_3) results in a state s' that possesses the graphical representation that is depicted in the middle part of Figure 5. The important feature of this new graph, for the state-liveness-assessment problem that is pursued in this work, is the presence of the cycle that is defined by the nodal sequence $\langle n_1, n_2, v_2, ch_3, v_1, n_1\rangle$; this is a newly formed cycle, that resulted from the clearance of the edge (n_1, n_3) (and the corresponding path $\langle n_1, (n_1, v_1), v_1, (v_1, ch_3), ch_3\rangle$). The creation of this cycle further implies that the entire graph depicted in the middle part of Figure 5, can be collapsed into the single node that is depicted in the right side of this figure.

Furthermore, the capacity of the node n_{123} that results from the aforementioned mergers, is equal to 5, that is larger than the capacity of any of the three nodes n_1, n_2 and n_3 that are involved in these mergers. A straightforward calculation of this capacity can be obtained from the graph depicted in the middle part of Figure 5, as the two units of capacity in each of the nodes n_2 and ch_3, plus the three free edges on the newly formed cycle. Alternatively, the capacity of node n_{123} can be obtained from the left graph of Figure 5 as the sum of the capacities of nodes n_1, n_2 and n_3 that are merged in the new node, plus the number of the free edges in the underlying PDG $\mathcal{C}(\hat{G}(s))$ of the considered DAG $\mathcal{U}(\hat{G}(s))$ that will belong on the newly formed cycle(s); this last calculation explains more clearly the dominance of the capacity of the resulting node n_{123} w.r.t. the capacities of the original nodes n_1, n_2 and n_3.

Generalizing the above example, we can see that, in states s of the FSA Φ not satisfying Condition 1, we can have an additional type of "producer" merger that results from the establishment of new cycles in the PDG $\hat{G}(s)$ that represents the state considered s. Therefore, an algorithm for assessing the liveness of an arbitrary state s of FSA Φ must possess additional logic for detecting and exploiting this additional possibility. This logic can be defined by means of the corresponding DAG $\mathcal{U}(\hat{G}(s))$, but space limitations do not allow its inclusion in this document; some recent developments on this issue can be traced in [24].

Dealing with a new form of "critical choice": An additional complication that arises when considering states s of the FSA

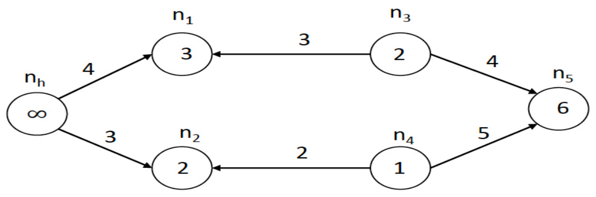


Fig. 6: A traffic state s involving "critical choice" in the advancement of the traveling agents.

Φ that do not satisfy Condition 1, is demonstrated through the example DAG $\mathcal{U}(\hat{G}(s))$ that is depicted in Figure 6. It should be clear from all the previous discussion on the traffic dynamics that are encoded by DAG $\mathcal{U}(\hat{G}(s))$, that for the particular case depicted in Figure 6, any possible advancement to a chained state must involve one of the two mergers that are defined by edges (n_3, n_5) and (n_4, n_5) according to the logic of Definition 5. We invite the reader to verify that the execution of the merger (n_4, n_5) at the depicted state s will lead to a non-live state s', while the state s'' that results from the execution of the merger (n_3, n_5) at state s, is live. Yet, currently, we do not avail of any algorithmic logic that will resolve correctly the corresponding choice (i.e., it will select the merger (n_3, n_5) at state s) and it will execute in polynomial time w.r.t. the size of the underlying guidepath network G.

Towards an efficient algorithm for assessing state liveness for traffic states of the considered transport systems that will not satisfy Condition 1: The example of Figure 6 that was discussed in the previous paragraph, and our current inability to resolve efficiently the critical choice that arises in the context of this example, imply that the worst-case computational complexity of assessing state-liveness for any *arbitrary* state coming from an open, irreversible, dynamically routed, zone-controlled guidepath-based transport system, remains an open issue.

Nevertheless, the compactness of the state representation that is provided by the DAG $\mathcal{U}(\hat{G}(s))$ that has been developed in this work, together with the various merging operations with a "producer" nature that have been identified for this DAG, provide the "seed" for an efficient search-based algorithm for assessing the state liveness of traffic states of open, irreversible, dynamically routed, zone-controlled guidepath-based transport systems that might not satisfy Condition 1. As in the case of Algorithm 2, this new algorithm will execute in an iterative manner, with each (major) iteration consisting of (i) a "pre-processing" stage that will seek to identify and execute "easy" transfers of traveling agents to the "home" node n_h, and (ii) a second stage that will seek to further advance the progress of the algorithm when such an advancement is not further possible through the reductions that are performed in the "pre-processing" stage. This second stage will be supported by the imposition of a layering structure on DAG $\mathcal{U}(\hat{G}(s))$ similar to that employed by Algorithm 2, but the possible outcomes of this stage in the context of this new algorithm will be (a) the identification and execution of a feasible "producer" merger, (b) the resolution of the non-liveness of the considered state s, or, eventually, (c) the branching upon a terminal node of an odd-numbered layer, as in the case of Figure 6. The "backtracking" that is involved in this "branching" logic implies that the resulting algorithm might be of super-polynomial worst-case computational complexity; but the *empirical* computational complexity of the algorithm is expected to be very benign. A more thorough development of the ideas that are outlined in this paragraph, can be found in [24].

Efficient assessment of state liveness in the "closed" counterparts of the considered transport systems: In [6] it has been shown that the result of Theorem 2 applies also to the class of the *closed*, irreversible, dynamically routed zone-controlled guidepath-based transport systems; i.e., a traffic state s from this class of transport systems is live *iff* it is co-reachable to a chained state s'. Furthermore, the basic logic of Proposition 4 for constructing a chained state s' from a given traffic state s through the chain-merging operations that were developed in the earlier parts of this work, applies also to this new class of transport systems, but with one caveat: Since these transport systems do not possess a "home" edge, the corresponding DAG $\mathcal{U}(\hat{G}(s))$ will possess no nodes of infinite capacity. Nevertheless, it is still possible to execute the algorithms that were presented in the previous parts of this document, in an iterative manner, that treats each of the "source" nodes of the DAG $\mathcal{U}(\hat{G}(s))$ as the "home" node n_h. The specification of such a "home" node is important for defining the layering structure to be used by the considered algorithms. But once the "home" node n_h and the corresponding layering structure have been determined in a given iteration, the rest of the algorithm steps during this iteration will be similar to the case of "open" transport systems. Furthermore, the algorithm will exit with a positive outcome as soon as one of these iterations leads to a chained state s'; otherwise, the algorithm will exhaust all possible choices for node n_h among the "source" nodes of the DAG $\mathcal{U}(\hat{G}(s))$, and it will return a negative result.

VI. Conclusions

This work has revisited the problem of determining the worst-case computational complexity of the decision problem of assessing state-liveness in open, irreversible, dynamically routed, zone-controlled guidepath-based transport systems. The study of this problem was originated in [8], and the results of this work have identified a new class of states that admit liveness assessment of polynomial complexity w.r.t. the size of the underlying guidepath graph G. But an additional important contribution of the presented results is the provision of a novel representational framework for the qualitative traffic dynamics of the considered transport systems w.r.t. the issue of liveness-enforcing supervision, that is based on the original developments of [6], and can support an effective and more efficient analysis of these dynamics than the more standard representational frameworks that are provided by the current DES theory. This framework was instrumental for the derivation of the main algorithmic results of this paper, while its enhanced analytical and computational potential has been further highlighted in the last part of the paper that discussed how the considered framework can also support effective and efficient liveness-assessment for additional classes of traffic states that transcend the particular class of traffic states that is the primary focus of this work.

A first part of our future work in this area will seek to further detail and assess the potential of the new methodological framework for the considered transport systems that was discussed in the previous paragraph. The developments that are presented in [24] is a first step in this direction. Furthermore, another task of considerable theoretical interest that remains open and needs further attention, is the complete resolution of the worst-case computational complexity of the decision problem of assessing state-liveness in the considered transport systems. Finally, a third part of our future work will seek to integrate the results that were derived in this paper, and their potential extensions through [24] and the various other research lines that were discussed in the earlier part of this paragraph, to a broader control framework that will seek to address not only the liveness of the underlying transport system, but also its operational efficiency in terms of time-based performance criteria like the maximization of some notion of "throughput" and the minimization of the delays and the congestion to be experienced by the traveling agents; some preliminary results on this last problem can be found in [25].

Acknowledgement

The authors want to acknowledge all the constructive comments that they received during the review process and have helped them enhance the thoroughness and the presentational quality of the paper.

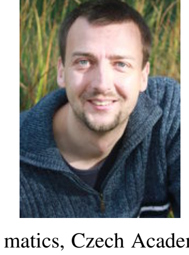
References

[1] W. L. Maxwell and J. A. Muckstadt, "Design of automatic guided vehicle systems," *IIE Trans.*, vol. 14, pp. 114–124, 1982.
[2] S. S. Heragu, *Facilities Design (3rd ed.)*. CRC Press, 2008.
[3] S. A. Reveliotis, "Conflict resolution in AGV systems," *IIE Trans.*, vol. 32(7), pp. 647–659, 2000.
[4] N. Wu and M. Zhou, "Resource-oriented Petri nets in deadlock avoidance of AGV systems," in *Proceedings of the ICRA'01*. IEEE, 2001, pp. 64–69.
[5] M. P. Fanti, "Event-based controller to avoid deadlock and collisions in zone-controlled AGVS," *Inlt. Jrnl Prod. Res.*, vol. 40, pp. 1453–1478, 2002.
[6] E. Roszkowska and S. Reveliotis, "On the liveness of guidepath-based, zoned-controlled, dynamically routed, closed traffic systems," *IEEE Trans. on Automatic Control*, vol. 53, pp. 1689–1695, 2008.
[7] S. Reveliotis and E. Roszkowska, "On the complexity of maximally permissive deadlock avoidance in multi-vehicle traffic systems," *IEEE Trans. on Automatic Control*, vol. 55, pp. 1646–1651, 2010.
[8] S. Reveliotis and T. Masopust, "Some new results on the state liveness of open guidepath-based traffic systems," in *27th Mediterranean Conference on Control and Automation (MED 2019)*. IEEE, 2019.
[9] T. Standley, "Finding optimal solutions to cooperative pathfinding problems," in *Proc. AAAI 2010*, 2010.
[10] Q. Sajid, R. Luna, and K. E. Bekris, "Multi-agent path finding with simultaneous execution of single-agent primitives," in *5th Symposium on Combinatorial Search*, 2012.
[11] J. Yu and S. M. LaValle, "Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics," *IEEE Trans. on Robotics*, vol. 32, pp. 1163–1177, 2016.
[12] H. Ma, C. Tovey, G. Sharon, S. Kumar, and S. Koenig, "Multi-agent path finding with payload transfers and the package-exchange robot-routing problem," in *AAAI 2016*, 2016, pp. 3166–3173.
[13] G. Daugherty, "Multi-agent routing in shared guidepath networks," Ph.D. dissertation, Georgia Tech, Atlanta, GA, 2017.
[14] G. Daugherty, S. Reveliotis, and G. Mohler, "Optimized multi-agent routing for a class of guidepath-based transport systems," *IEEE Trans. on Automation Science and Engineering*, vol. 16, pp. 363–381, 2019.
[15] P. Surynek, "Towards optimal cooperative path planning in hard setups through satisfiability solving," in *Proc. 12th Pacific Rim Int. Conf. Artif. Intell.*, 2012, pp. 564–576.
[16] R. M. Wilson, "Graph puzzles, homotopy, and the alternating group," *Journal of Combinatorial Theory, B*, vol. 16, pp. 86–96, 1974.
[17] T. Barnard and H. Neill, *Discovering Group Theory: A Transition to Advanced Mathematics*. Boca Raton, FL: CRC Press, 2017.
[18] D. Kornhauser, G. Miller, and P. Spirakis, "Coordinating pebble motion on graphs, the diameter of permutation groups, and applications," in *Proc. IEEE Symp. Found. Comput. Sci.* IEEE, 1984, pp. 241–250.
[19] V. Auletta, A. Monti, M. Parente, and P. Persiano, "A linear-time algorithm for the feasibility of pebble motion on trees," *Algorithmica*, vol. 23, pp. 223–245, 1999.
[20] D. Pillai, "The future of semiconductor manufacturing: Factory integration breakthrough opportunities," *IEEE Robotics & Automation Magazine*, vol. 13-4, pp. 16–24, 2006.
[21] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems (2nd ed.)*. NY,NY: Springer, 2008.
[22] D. B. West, *The Art of Combinatorics - Volume A: Introduction to Graph Theory*. Urbana, IL: Preliminary Version - Dept. of Mathematics, UIUC, 1993.
[23] R. E. Tarjan, "A note on finding the bridges of a graph," *Information Processing Letters*, pp. 161–162, 1974.
[24] S. Reveliotis and T. Masopust, "Efficient assessment of state liveness in open, irreversible, dynamically routed, zone-controlled guidepath-based transport systems: The general case," School of Industrial & Systems Eng., Georgia Tech, Tech. Rep. (submitted for publication), 2019.
[25] S. Reveliotis, "An MPC scheme for traffic coordination in open and irreversible, zone-controlled, guidepath-based transport systems," in *15th IEEE Inlt. Conf. on Automation Science and Engineering (CASE 2019)*. IEEE, 2019.

Spyros Reveliotis received the Diploma degree in electrical engineering from the National Technical University of Athens, Greece, the M.Sc. degree in computer systems engineering from Northeastern University in Boston, and the Ph.D. degree in industrial engineering from the University of Illinois at Urbana-Champaign.

He is a Professor with the School of Industrial and Systems Engineering, Georgia Institute of Technology, in Atlanta, GA. His main research interests are in discrete-event systems theory and its applications.

Dr. Reveliotis is an IEEE Fellow and a member of INFORMS. He has served on the editorial boards of many journals and conferences on his areas of interest. Currently, he serves as a Senior Editor for the IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, an Associate Editor for the Journal of Discrete Event Dynamic Systems, and the Editor-in-Chief of the Editorial Board at the IEEE Conference on Automation Science and Engineering (CASE). He has also served as the Program Chair at the 2009 IEEE CASE Conference, and the General Co-Chair of the 2014 edition of the same conference. Finally, he has been a recipient of a number of awards, including the 2014 Best Paper Award of the IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING.



Tomáš Masopust received an M.Sc. degree in computer science from Masaryk University, Brno, Czechia in 2004, and a Ph.D. degee in computer science from Brno University of Technology, Brno, Czechia in 2007.

Dr. Masopust has worked at CWI Amsterdam, the Netherlands, in the Systems and Control Group, at University of Bayreuth, Germany, in the Theoretical Computer Science Group, and at TU Dresden, Germany, in the Knowledge-Based Systems Group. Since 2017 he is a researcher at Institute of Mathematics, Czech Academy of Sciences, Brno, Czechia. Furthermore, since 2018 he is also an Assistant Professor at Palacky University, Olomouc, Czechia.

Dr. Masopust's research interest includes verification and control of discrete-event systems and theoretical computer science. He is an editor of Kybernetika journal.