Efficient Assessment of State Liveness in Open, Irreversible, Dynamically Routed, Zone-Controlled Guidepath-based Transport Systems: The General Case

S. Reveliotis * **T. Masopust** **

* *School of Industrial & Systems Engineering Georgia Institute of Technology, USA (email: spyros@isye.gatech.edu)*
** *Institute of Mathematics Czech Academy of Sciences (email: masopust@math.cas.cz)*

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. This work capitalizes upon some recent developments on the problem of assessing state liveness in the considered transport systems in order to provide a novel algorithm for this problem. The worst-case computational complexity of this algorithm is not polynomially bounded with respect to the size of the underlying transport system, but the empirical complexity of the algorithm is expected to be very benign for the reasons that are explained in the paper.

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

1. INTRODUCTION

Motivation and an operational characterization of the guidepath-based transport systems considered in this work: Open, irreversible, dynamically routed, zone-controlled, guidepath-based transport systems is a modeling abstraction for the traffic dynamics in many industrial material handling systems (MHS); some specific examples of these systems include the automated guided vehicle (AGV) systems that are used in various production and distribution facilities, the overhead monorail systems that are used in modern semiconductor fabs, and the complex crane systems that are used in various major ports and railway yards (Heragu (2008)).

In all these environments, the system "agents" that perform the transporting operations are moving through the links of a complex guidepath network that is defined either by the physical structure of the transport system itself (as in the case of the overhead monorail and the crane systems), or it is enforced externally in an effort to isolate the traffic that takes place in these systems from its surrounding environment (as in the case of the AGV systems).

Furthermore, in an effort to ensure collision-free operation among the traveling agents, it is further stipulated that each link of the underlying guidepath network cannot be occupied by more than one agent at a time. In the corresponding terminology, these links are characterized as the "zones" of the guidepath network. The advancement of an agent from its current zone to a neighboring one must be negotiated with the traffic coordinator, and it can be granted only if the requested zone is currently free. [1]

More generally, the zone-based control scheme that was described in the previous paragraph, turns the agent trips between their various destinations into a "resource allocation" process where the requested resources are the necessary zones of the guidepath network. Also, in the considered class of guidepath-based transport systems, the agent routes among their different destinations are synthesized dynamically, one zone at a time, based on the prevailing conditions and the experienced congestion in the underlying guidepath network.

Furthermore, agents can move only in the forward direction of their longitudinal axis, which implies a notion of "irreversibility" for their motion; in particular, in the con-

[1] This last condition further implies that zone swapping among agents that occupy neighboring zones of the underlying guidepath network, is prohibited.

sidered guidepath-based transport systems, agents cannot "back up" within their current zone.

Finally, the considered guidepath-based transport systems are also "open", in that they possess a "home" location where all idling agents can retire, possibly receiving service, recharging their batteries, etc.

The problem of liveness-enforcing supervision in the considered transport systems, and a brief literature review: The guidepath networks of the considered transport systems are supposed to be connected and with a minimal vertex degree of 2 (since the presumed irreversibility of the agent motion further implies that an agent reaching a vertex of degree 1 would get stuck at that vertex). But, otherwise, the topology of these guidepath networks can be quite arbitrary. This arbitrary structure of the guidepath network, when combined with the dynamic and irreversible routing of the system agents that was described in the previous paragraphs, further imply that the considered transport systems are susceptible to deadlock and/or livelock.

Hence, a significant task of the traffic coordinator is to restrict further the aforementioned zone allocation process in order to establish "traffic liveness". In practical terms, traffic liveness implies the preservation of the ability of each traveling agent to complete successfully its current "mission" trip and engage repetitively into similar "mission" trips in the future. On the other hand, an analytical characterization of this concept can be provided using concepts and representations borrowed from the Discrete Event Systems (DES) theory (Cassandras and Lafortune (2008)). Indeed, recognizing also the connection of the considered traffic dynamics to the notion of "sequential resource allocation" that was mentioned in the previous paragraphs, a group of researchers from the DES community has tried to establish liveness-enforcing supervision (LES) [2] for the considered transport systems by adapting to this problem ideas and techniques that have been developed for LES synthesis in the context of the more general resource allocation systems studied in Reveliotis (2017). Some characteristic examples of this line of past work on the considered supervisory control problem can be found in Reveliotis (2000); Wu and Zhou (2001); Fanti (2002).

More recently, in some work of ours presented in Reveliotis and Masopust (2019, 2020), we have tried to address the further notion of "maximal permissiveness" for the considered supervisory control problem. It turns out that the underlying traffic will be live as long as the system retains its ability to bring all traveling agents to the "home" location (without necessarily having completed their "mission" trips). This result further motivates the definition of a "live" traffic state as any state that is co-reachable to the aforementioned traffic state where all agents are at the "home" location; this last state will be characterized as the "home" traffic state in the following. Furthermore, the maximally permissive LES will admit an agent advancement to a requested neighboring zone if and only if (*iff*) this zone is currently free and the resulting traffic state is live.

[2] In the following, LES will imply either "liveness-enforcing supervision" or "liveness-enforcing supervisor", depending on the context.

But the computational complexity of the decision problem of assessing traffic-state liveness in the considered guidepath-based transport systems is currently an open issue. Motivated by this situation, in Reveliotis and Masopust (2019, 2020) we have been able to identify classes of states from the considered transport systems where liveness assessment can be performed with worst-case complexity that is polynomial with respect to the size of the underlying guidepath network. Furthermore, the corresponding results of Reveliotis and Masopust (2020) develop an entire methodological framework – in terms of novel representations for the dynamics of the underlying traffic, and the supporting computational techniques – that eventually provides the aforementioned liveness-assessment algorithm for the traffic-state class that is the focus of that work.

The intended contribution and the further content of this work: In this work, we extend the developments of Reveliotis and Masopust (2020) in order to provide an efficient liveness-assessment algorithm for the entire set of traffic states that can arise in the considered transport systems. More specifically, this new algorithm cannot be claimed to be of polynomial worst-case complexity with respect to the size of the underlying transport system, but still it is expected to possess a very benign empirical complexity, due to the way that it takes advantage of, and it builds upon, the various representations, the analytical results, and the broader insights that have been developed in Reveliotis and Masopust (2020). This last assessment is supported by the technical developments that are provided in the later parts of the paper, and by some expository examples.

In view of the above positioning of the paper content and its intended contribution, the rest of it is structured as follows: The next section provides a more formal characterization of the structure and the traffic dynamics of the considered transport systems, and of the notion of "traffic state liveness" that is at the center of this work. Section 3 overviews the results of Reveliotis and Masopust (2020) that enable the developments to be presented in this work. These new developments themselves are presented in Section 4. Finally, Section 5 concludes the paper and suggests some directions for future work. [3]

2. A FORMAL CHARACTERIZATION OF THE CONSIDERED TRANSPORT SYSTEMS AND THE CORRESPONDING PROBLEM OF ASSESSING STATE LIVENESS

A formal representation of the considered transport systems is provided by the tuple $(\mathcal{A}, G)$, where $\mathcal{A}$ denotes the set of the system agents, and G is the underlying guidepath network. More specifically, $G = (V, E \cup \{h\})$ is an undirected connected multi-graph with a minimal vertex degree of 2. The edge set E corresponds to the set of zones of the guidepath network, while h is a self-loop edge that represents the "home" location of the underlying

[3] Due to the imposed space limitations, Sections 2 and 3 provide the minimal possible information that is necessary for a meaningful discussion of the main results of Section 4. The reader is referred to Reveliotis and Masopust (2019, 2020) for a more expansive treatment of this material.

transport system. We shall also use the notation v_h to denote the single terminal vertex of the self-loop edge h.

Since in the liveness-assessment problem that is considered in this work all agents are destined to the "home" location h, there is no need to maintain a distinct identity for each agent. Hence, we can define the "traffic state" s by (i) the distribution of the system agents on the various edges of the underlying guidepath network, and (ii) for those agents that are located on an edge $e \neq h$, we must also encode the orientation of their motion in their current edges. A pertinent encoding of the traffic state s is by a partially directed graph (PDG) $\hat{G}(s)$; this graph is obtained from the original graph G by turning each edge $e \in E$ that is occupied by some traveling agent a in s into a directed edge that indicates the direction of motion of agent a on this edge. State s evolves by advancing a single agent a from its current edge $e(a; s)$ to a neighboring edge e' that (a) is free in s and (b) the corresponding advancement is consistent with the direction of motion of agent a.

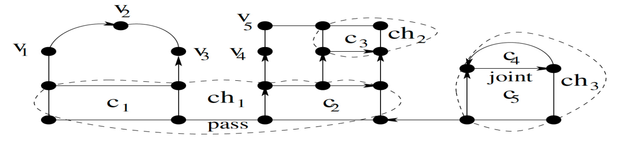
Let S denote the entire set of traffic states under the above representation. Clearly, S is finite, and the dynamics that are defined in the previous paragraph define a finite state automaton (FSA) Φ (Cassandras and Lafortune (2008)). In the following, we shall use the notation s_h to denote the "home" state, i.e., the state where all agents $a \in \mathcal{A}$ are located on the "home" edge h. Furthermore, a traffic state $s \in S$ will be characterized as "live" *iff* it is co-reachable to state s_h in the dynamics of Φ. Let S_l denote the set of all live states of Φ. Then, the problem addressed in this work is as follows: Given a traffic state $s \in S$, does s belong in S_l?

As remarked in the introductory section, in Reveliotis and Masopust (2020) we have developed an algorithm for resolving efficiently this state-liveness assessment problem for a particular traffic-state subclass. The efficiency of this algorithm results from a pertinent processing of the information that is encoded in the aforementioned PDG $\hat{G}(s)$. We review the main results of Reveliotis and Masopust (2020) in the next section, and in Section 4 we shall extend these results to an algorithm that will provide efficient liveness assessment for any state $s \in S$.

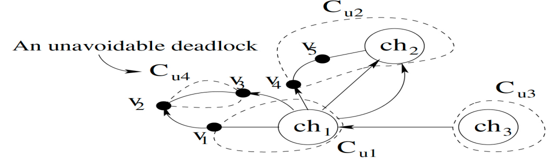
3. THE METHODOLOGICAL DEVELOPMENTS OF REVELIOTIS AND MASOPUST (2020)

Representations: In the first part of this section we present a series of more compressed representations of the traffic state s, that are induced by the PDG $\hat{G}(s)$ and will enable an efficient computation of an agent-advancing event sequence that leads from the considered state s to the target state s_h, whenever such a sequence exists. These representations are introduced through an extensive sequence of definitions, and the reader is also referred to Figure 1 for some concrete examples of the various concepts and structures that are introduced by these definitions.

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,



(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. 1. This figure exemplifies the definitions and the technical results that are provided in Section 3.

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. [4] 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.

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

Definition 1. 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) π_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. □

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

[4] 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.

$\hat{G}(s)$ 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*.

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 v_1 and v_2, contains a directed edge (since otherwise, the chains corresponding to the macro-vertices v_1 and v_2 would not be maximal).

The next abstraction is defined on PDG $\mathcal{C}(\hat{G}(s))$ and distinguishes 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 2. 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. □

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. In the following, we shall work primarily with the directed acyclic (multi-)graph (DAG) $\mathcal{U}(\hat{G}(s))$ that is obtained by the condensation $\mathcal{C}(\hat{G}(s))$ by replacing each u-component of $\mathcal{C}(\hat{G}(s))$ by a single vertex. Also, in order to distinguish this graph structure from the previous ones, we shall refer to the vertices of this DAG as "nodes" to be denoted by n. And we shall associate a notion of "capacity" with each node n of DAG $\mathcal{U}(\hat{G}(s))$ as follows:

Definition 3. For each node n of DAG $\mathcal{U}(\hat{G}(s))$ the corresponding *capacity* $\chi(n)$ is defined as follows: For the nodes n of $\mathcal{U}(\hat{G}(s))$ that correspond to simple vertices of the original guidepath graph G, as well as for those nodes n of $\mathcal{U}(\hat{G}(s))$ that correspond to simple u-components of $\mathcal{C}(\hat{G}(s))$, the corresponding capacity $\chi(n)$ is set equal to zero. On the other hand, a node n of $\mathcal{U}(\hat{G}(s))$ representing a complex u-component $\mathcal{C}_u$ of $\mathcal{C}(\hat{G}(s))$, will have its capacity $\chi(n)$ set equal to the number of free edges on the cycles of the chained component that constitutes the unique macro-vertex of $\mathcal{C}_u$. Furthermore, for the vertex n_h of DAG $\mathcal{U}(\hat{G}(s))$ that contains the "home" edge h, we set $\chi(n_h) = \infty$. □

Finally, an even more compact representation of the DAG $\mathcal{U}(\hat{G}(s))$ that is particularly convenient for the algorithmic developments that were pursued in Reveliotis and Masopust (2020) and also for the developments to be presented in the next section, can be obtained as follows: (I) This representation recognizes as the *"(major) nodes"* of DAG $\mathcal{U}(\hat{G}(s))$ those nodes that (i) either correspond to a complex u-component, or (ii) have their in-degree or out-degree greater than 1. (II) Furthermore, it replaces each simple path π that connects a major nodal pair (n_1, n_2) and contains only non-major nodes of $\mathcal{U}(\hat{G}(s))$ as interior nodes, 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.

The work of Reveliotis and Masopust (2020) also provides a detailed algorithm for the construction of the weighted DAG $\mathcal{U}(\hat{G}(s))$ from the original PDG $\hat{G}(s)$ and for the computation of the corresponding nodal capacities $\chi(n)$. The worst-case computational complexity of this algorithm is $O(|V| + |E|)$, where V and E are, respectively, the vertex set and the zone set of the underlying guidepath network. Therefore, the construction of this more compressed representation of any given traffic state s is very efficient.

Inference: The nodal capacities $\chi(n)$ of the DAG $\mathcal{U}(\hat{G}(s))$ that are defined in Definition 3, denote the *maximal* numbers of additional agents that can be absorbed in the corresponding nodes n from the incoming edges to these nodes, without compromising the chained structure associated with these nodes. In particular, given an edge (n_1, n_2) of DAG $\mathcal{U}(\hat{G}(s))$ with $w(n_1, n_2) \leq \chi(n_2)$, it is possible to advance all the agents on edge (n_1, n_2) into node n_2, obtaining, thus, a new maximal chained component in the resulting state s' that contains the chained components of both nodes n_1 and n_2 (now linked through the new pass that is defined by the freed edge (n_1, n_2)). On the other hand, terminal nodes of the DAG $\mathcal{U}(\hat{G}(s))$ with zero capacity indicate the development of unavoidable deadlocks in any trace that emanates from the corresponding state s. Furthermore, the presumed connectivity properties for the guidepath network G imply that the condensation $\mathcal{C}(\hat{G}(s_h))$ for the "home" state s_h will consist of a single chained component, and the corresponding DAG $\mathcal{U}(\hat{G}(s))$ will consist of a single node n_h of infinite capacity. Finally, the above remarks further imply that, in the semantics of the DAG $\mathcal{U}(\hat{G}(s))$, the construction of an agent-advancing event sequence that will collect all agents $a \in \mathcal{A}$ to the "home" edge h, reduces to the identification of a series of "merging" operations on the DAG $\mathcal{U}(\hat{G}(s))$ that will keep reducing the number of nodes of the DAGs $\mathcal{U}(\hat{G}(s_i))$ that correspond to the derived state sequence $\langle s_1, s_2, \ldots \rangle$, by reducing the maximal chained components in these states.

In order to ensure the computational efficiency of the resulting algorithm, we are particularly interested in, so called, "producer" mergers, i.e., merging operations that when executed on the current DAG $\mathcal{U}(\hat{G}(s))$ will only enhance the merging potential among the remaining nodes of this DAG, and therefore, there is no need to backtrack on these mergers during the search for a complete merging sequence leading to the "home" state s_h. Next, we present two such "producer" mergers that have been identified in Reveliotis and Masopust (2020).

Definition 4. Consider a DAG $\mathcal{U}(\hat{G}(s))$ and a path $\pi = \langle n_1, e_1, n_2, e_2, \ldots, e_{k-1}, n_k \rangle$ in it. Furthermore, let $\chi(n_i)$

denote the capacity of node n_i, for $i = 1, \ldots, k$, and $w(e_j)$ denote the weight associated with edge e_j, $j = 1, \ldots, k-1$. Then, we have the following definitions:

(1) Path π defines a *feasible path-based* merger of its nodes $n_i, i = 1, \ldots, k$, *iff* $\forall i = 1, \ldots, k-1$, $\sum_{j=i+1}^{k} \chi(n_j) - \sum_{j=i}^{k-1} w(e_j) \geq 0$.
(2) Path π constitutes a *minimal* feasible path-based merger emanating from node n_1 if, in addition to (1) above, it also holds that $\forall i = 2, \ldots, k-1$, $\sum_{j=2}^{i} \chi(n_j) - \sum_{j=1}^{i-1} w(e_j) < 0$.
(3) Finally, path π is a minimal feasible path-based *"producer"* merger if, in addition to (1) and (2) above, it also holds that $\forall i = 2, \ldots, k$, $\sum_{j=1}^{k} \chi(n_j) - \sum_{j=1}^{k-1} w(e_j) \geq \chi(n_i)$. $\square$

Definition 5. Consider a DAG $\mathcal{U}(\hat{G}(s))$ and two paths π_1 and π_2 with the same starting and ending nodes n_1 and n_2, and no other common nodes or edges among them. Furthermore, assume that (at least) one of these two paths is a feasible merger (according to part #1 of Definition 4). Then, paths π_1 and π_2 define a *cycle-generating* "producer" merger. $\square$

Definition 4 is self-explanatory. As for Definition 5, notice that the clearance of (let's say) path π_1 from its occupying agents will lead to a new state s' that will possess a new cycle in the corresponding PDG $\hat{G}(s')$ consisting of the undirected edges of the cleared path π_1 and the directed edges of path π_2. This newly formed cycle implies that the freed edges of path π_1 will be part of the nodal capacity of the new node that will result from the effected merging, and therefore, this merger can only improve the merging potential of the remaining nodes of the DAG $\mathcal{U}(\hat{G}(s))$.

In Reveliotis and Masopust (2020), we focused on the particular class of traffic states $s \in S$ for which the undirected graph induced by the DAG $\mathcal{U}(\hat{G}(s))$ is a tree. These states are not amenable to "cycle-generating" mergers. Hence, we developed a greedy algorithm for the resolution of the corresponding liveness-assessment problem, based on a systematic detection and execution of feasible path-based "producer" mergers that are interleaved with the execution of some additional feasible (but not necessarily "producer") mergers which are unavoidable due to the tree structure of DAG $\mathcal{U}(\hat{G}(s))$. In the next section we extend the original developments of Reveliotis and Masopust (2020) to an algorithm that can assess the liveness of any traffic state $s \in S$ from the considered transport systems. At the core of this extension is a new methodology for searching for feasible path-based "producer" mergers and "cycle-generating" mergers in the DAG $\mathcal{U}(\hat{G}(s))$ that is processed at each major iteration of the algorithm.

4. MAIN RESULTS

Preamble: As already stated in the previous section, the algorithm to be presented in the following will take as input a given traffic state $s \in S$, and it will seek to construct an agent-advancing event sequence that will lead from the considered state s to the "home" state s_h. This will be attained by performing a sequence of mergers on the DAGs $\mathcal{U}(\hat{G}(s_i))$, where $\langle s_i, i = 0, 1, 2 \ldots \rangle$ denotes the sequence of traffic states with $s_0 \equiv s$ and s_i, $i = 1, 2, \ldots$, being the traffic states that result from the execution of the aforementioned mergers.

At every state s_i, $i = 0, 1, 2, \ldots$, the algorithm will construct the corresponding DAG $\mathcal{U}(\hat{G}(s_i))$, and as in the case of the corresponding algorithm in Reveliotis and Masopust (2020), it will first check for paths in the DAG $\mathcal{U}(\hat{G}(s_i))$ that either

(1) are directed to the "home" node n_h, and therefore, the corresponding agents can be immediately absorbed into this node; or
(2) emanate from node n_h and define minimal feasible mergers (the infinite capacity of the starting node n_h for these paths further implies that the corresponding feasible mergers will also be "producers" – c.f. part #3 of Definition 4). [5]

The search for the above two types of paths, and the execution of the corresponding mergers, will be carried out in an iterative manner, and the completion of this pre-processing stage might either lead to the total absorption of all agents into node n_h, in which case, the current state s_i, and also the original state s, can be declared to be live, or it will result in a new state s'_i where the node n_h of the corresponding DAG $\mathcal{U}(\hat{G}(s'_i))$ will be a "source" node of this DAG, and there will be no "producer" merger originating from this node. At this point, the algorithm will need to search for a pertinent feasible merger in the new DAG $\mathcal{U}(\hat{G}(s'_i))$, or infer the non-liveness of the considered state s.

The search for a pertinent merger will be facilitated by the imposition of a layering structure on the DAG $\mathcal{U}(\hat{G}(s'_i))$ that is defined through 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 $\mathcal{U}(\hat{G}(s'_i))$. Also, the edges on all those paths are labeled as "layer 1" edges.
(2) Assuming that layers $1, \ldots, j$ are well defined, layer $j+1$ is defined as follows:
    (a) If $j+1$ is an odd number, layer $j+1$ contains all nodes n of DAG $\mathcal{U}(\hat{G}(s'_i))$ that are reachable from some node n' of layer j through paths consisting of edges that do not belong in any of the layers $1, \ldots, j$; the edges of all these paths are also labeled as "layer-$(j+1)$" edges.
    (b) If $j+1$ is an even number, layer $j+1$ contains all nodes n of DAG $\mathcal{U}(\hat{G}(s'_i))$ that are co-reachable to some node n' of layer j through paths consisting of edges that do not belong in any of the layers $1, \ldots, j$; the edges of all these paths are also labeled as "layer-$(j+1)$" edges.

The above layering structure for the considered DAGs was also used in Reveliotis and Masopust (2020), and it is motivated by an intention to capture the orientation of the agent motion on the various edges of these DAGs with respect to their "source" node n_h. More specifically, edges with an odd "layer" number are occupied by agents that

[5] An efficient method for searching for this second set of paths is provided in a later part of this section.

are heading away from node n_h in the underlying DAG $\mathcal{U}(\hat{G}(s_i'))$, while edges with an even "layer" number are occupied by agents that are moving in the direction of node n_h. This understanding is important for the further specification of the processing that will take place within each of these layers.

The proposed algorithm will search for a pertinent merger in the DAG $\mathcal{U}(\hat{G}(s_i'))$ on a layer-by-layer basis, starting with the largest numbered layer. The detailed logic for conducting this search is a novel development of this work and the subject of the next paragraphs.

Searching for a feasible "producer" merger within a single layer: As already discussed, a key concept underlying the execution logic of the considered algorithm is that of a feasible "producer" merger. In particular, Definitions 4 and 5 characterize two such "producer" mergers, respectively known as path-based and cycle-generating. Next, we outline a procedure that will search for such mergers in any given layer of the DAGs $\mathcal{U}(\hat{G}(s_i'))$ to be processed by the considered algorithm.

We start by noticing that the edges of some layer j of DAG $\mathcal{U}(\hat{G}(s_i'))$, together with their terminal nodes, define an acyclic subnet of DAG $\mathcal{U}(\hat{G}(s_i'))$; for further reference, let us denote this subnet by $\mathcal{G}_j = (\mathcal{N}_j, \mathcal{E}_j)$. [6] Since $\mathcal{G}_j$ is acyclic, we can determine a topological ordering $o_j(\cdot)$ of its nodes, i.e., a bijective mapping from $\mathcal{N}_j$ to $\{1, \ldots, |\mathcal{N}_j|\}$ such that any edge $(n_1, n_2) \in \mathcal{E}_j$ will have $o_j(n_1) < o_j(n_2)$; an efficient algorithm for the computation of $o_j(\cdot)$ can be found in Ahuja et al. (1993). Nodes $n \in \mathcal{N}_j$ will be processed one at a time, in increasing value of $o_j(n)$, searching for a minimal feasible path-based merger that will emanate from node n and it will specify a "producer" merger of one of the two categories mentioned above.

For any given node $n \in \mathcal{N}_j$, the aforementioned search is organized as follows: First, we compute the subnet $\mathcal{S}_j(n)$ of $\mathcal{G}_j$ that consists of node n itself and all of its successors in net $\mathcal{G}_j$; this computation can be performed by a basic forward-reaching algorithm. Then, we execute the following recursion on the nodes u of the net $\mathcal{S}_j(n)$:

$$\text{if } \ u = n \ \text{ then } \ \delta(u) = 0 \ ;$$
$$\text{else } \ \delta(u) = \min_{u' \in \texttt{ImPred}(u;\mathcal{S}_j(n))} \{\delta(u') + w(u', u)\} - \chi(u). \quad (1)$$

In the above equation, $\texttt{ImPred}(u; \mathcal{S}_j(n))$ is the set collecting the nodes u' of $\mathcal{S}_j(n)$ with an edge (u', u) in this net, i.e., the "immediate predecessors" of node u in $\mathcal{S}_j(n)$. Then, the reader can verify that, for any node $u \neq n$, $\delta(u)$ expresses the minimal capacity deficit for establishing a feasible path-based merger spanning from node n to node u; in particular, $\delta(u) \leq 0$ implies the presence of such a feasible path-based merger for node u.

Hence, the proposed algorithm will execute the recursion of Equation 1 until it detects a node $u \neq n$ with $\delta(u) \leq 0$. At this point, the algorithm will check whether the detected path-based feasible merger also implies the presence of a "producer" merger. More specifically, the algorithm will perform the following two tests:

- **Testing for a cycle-generating "producer" merger:** If it holds $|\texttt{ImPred}(u; \mathcal{S}_j(n))| > 1$ for the aforementioned node u, then, the feasibility of the detected path-based merger implies the existence of a cyclical merger between some node u' on the path π that corresponds to aforementioned merger, and node u itself. Node u' can be detected by scanning the nodes of path π, starting from node n itself, and searching for a node u' for which there exists a path $\pi'(u')$ from node u' to node u with no common interior nodes with path π.
- **Testing for a path-based "producer" merger:** If $|\texttt{ImPred}(u; \mathcal{S}_j(n))| = 1$, the algorithm will check whether the path π that corresponds to the detected feasible merger is also a "producer". This can be checked through the conditions of item #3 of Definition 4. In the semantics of the considered algorithm that were defined in the previous paragraphs, these conditions can be expressed as follows:

$$\forall u' \in \pi \text{ with } u' \neq n, \ \ \chi(n) - \chi(u') \ \geq \ \delta(u) \quad (2)$$

As soon as one of the above two tests is satisfied, the algorithm will execute the corresponding "producer" merger on the underlying traffic state s_i', and proceed with the execution of a new major iteration on the resulting traffic state s_{i+1} (along the lines explained in the opening part of this section). Otherwise, the algorithm will continue the exploration of the current layer j, by continuing the execution of the recursion of Equation 1, and possibly initiating the processing of the next node n', according to the topological ordering $o_j(\cdot)$, if the execution of the recursion upon the net $\mathcal{S}_j(n)$ has already been completed. In this way, the algorithm will detect and execute any possible "producer" merger in the considered layer j of the DAG $\mathcal{U}(\hat{G}(s_i'))$.

An outline of the entire algorithm: Having described the complete processing of a single layer of the DAGs $\mathcal{U}(\hat{G}(s_i'))$ in an effort to detect the feasible "producer" mergers in it, next we outline how this capability can define a basis for a complete algorithm that will effectively assess the liveness of any input traffic state $s \in S$.

As already mentioned in the preamble of this section, the algorithm will start with the DAG $\mathcal{U}(\hat{G}(s))$ and it will go through a series of major iterations. Each such iteration either (i) will terminate with an executed merger, or (ii) will return a liveness assessment for the evaluated state s.

More specifically, the algorithm will terminate at its i-th iteration declaring the considered state as live *iff* the corresponding DAG $\mathcal{U}(\hat{G}(s_i))$ or $\mathcal{U}(\hat{G}(s_i'))$ has been reduced to a single node.

Furthermore, while processing a given layer j in the current DAG $\mathcal{U}(\hat{G}(s_i'))$, the algorithm will either manage to eliminate this layer from the subsequent states, through the execution of some "producer" mergers, or it will execute all possible such mergers without eliminating this layer. If this processed but non-eliminated layer is even-numbered, then all agents located on the edges of this layer are destined towards the "home" node n_h, and the algorithm will just maintain this layer structure in the subsequent DAGs, recognizing that all these agents eventually will reach the "home" node n_h under a positive

[6] For a complete understanding of this net, we also notice that the undirected graph that is induced by it is not necessarily a connected graph.

outcome of the processing of the remaining layers of the DAG. In the meantime, by going through the processing of this layer, the algorithm has extracted from this layer the maximal possible capacity that can be utilized in the processing of the remaining (smaller-numbered) layers.

On the other hand, if the processed but non-eliminated layer is odd-numbered, then, the algorithm first must check that the terminal nodes of this layer do not imply any unavoidable deadlock. This will happen if there is a terminal node with no adequate capacity to effect a merger between this node and one of its immediate predecessors in, both, the considered layer j and also in layer $j+1$ (if such a layer is present). As soon as such a situation is detected, the algorithm will exit declaring the original state s as non-live. In the remaining cases, the algorithm will proceed to check whether a terminal node n has sufficient capacity $\chi(n)$ to merge (i) simultaneously with all its immediate predecessors in the considered layer, or (ii) with only one particular immediate predecessor. If such a node n is detected, the corresponding merger will be executed immediately, and the algorithm will proceed to its next major iteration. Finally, if all terminal nodes in the considered layer can merge with some (more than one) of their predecessors, but not all, then, the algorithm is faced with "choice", and this choice will be resolved through a "branching" process; in particular, each possible merger of a terminal node n with one of the immediate predecessors n' will define a separate branch for the algorithm. For each selected branch, the algorithm will pursue the entire subtree of the underlying search process, and either it will terminate with a positive decision within this subtree, or it will return to this branching node and pursue an alternative unexplored branch.

Next, we demonstrate the execution logic of the proposed algorithm through a specific example.

Example: In this example we apply the presented algorithm to a traffic state s possessing the DAG $\mathcal{U}(\hat{G}(s))$ that is depicted at the left-top part of Figure 2. In fact, Figure 2 presents the complete execution of the algorithm. For each depicted DAG in this figure, black labels within each node n_i denote the corresponding nodal capacity $\chi(n_i)$, while black labels on the graph edges (n_i, n_j) denote the corresponding weights $w(n_i, n_j)$. Red labels denote the layer number of each edge (and this information can be used in order to deduce the layer number of each node in the graph, as well).

Since node n_h is a "source" node of the depicted DAG $\mathcal{U}(\hat{G}(s))$, the algorithm starts its first iteration by searching for "producer" mergers in layer 4. The blue numbers that are reported next to each of the nodes n_5, n_6 and n_7, that are the terminal nodes for layer-4 edges in DAG $\mathcal{U}(\hat{G}(s))$, are the δ-values that are computed through the recursion of Equation 1 during this first iteration of the algorithm. From these values we can see that there is a feasible path-based merger between node n_7 and node n_5, and this path-based merger is indicated with the green arrows in this DAG. Furthermore, since node n_5 has two incoming arrows in the $\mathcal{S}_4(n_7)$ subnet, the presence of the aforementioned merger further implies the presence of a cycle-generating "producer" merger. Also, the presence of the alternative path from node n_7 to node n_5 implies that the newly generated cycle will involve all nodes n_5, n_6 and n_7. Finally, the depicted DAG $\mathcal{U}(\hat{G}(s))$ also reveals the internal structure of the u-component that corresponds to node n_7, and we can see that there is an undirected edge on the alternate path that links the macro-vertex corresponding to the chained component of node n_7 with node n_5. Assuming that there is no further such "hidden" capacity that will be introduced by the newly generated cycle, we can compute the capacity of the resulting new node as the sum of the original capacities of the merged nodes plus 1 (which accounts for the aforementioned edge); hence, the total capacity of the new node n_{567} is 2+1+2+1=6.

Node n_{567} in the DAG that results from the cycle-generating merger that was discussed above, is the single terminal node in the largest-numbered layer of that DAG. Furthermore, this node possesses adequate free capacity to support a merger with any of its two predecessor nodes, n_3 and n_4, but not with both of them. Hence, the algorithm must "branch" at this point on this choice, and the computation that will result from each of the two options is presented in the next two parts of Figure 2.

As we can see, the execution of the algorithm in the case that node n_{567} is merged with node n_4, will result in a DAG with two layers, no "producer" merger in layer 2, and a terminal node for layer 1 (and for the entire DAG) where all possible mergers are infeasible. The development of this formation manifests the non-liveness of the traffic state that corresponds to this DAG, and, therefore, the inability of the currently pursued path to establish a "liveness" certificate for the original state s. Hence, the algorithm will backtrack at this point, in order to pursue the second of the two merging options that was discussed above. The second path *does* provide a merger sequence that leads to a state s' with a single node for the DAG $\mathcal{U}(\hat{G}(s'))$, or equivalently, a single maximal chained component for the PDG $\hat{G}(s')$. Hence, state s' can be pronounced as live, which further implies the liveness of the original state s.

A "correctness" analysis of the considered algorithm: The next theorem states the correctness and the finiteness of computation of the proposed algorithm.

Theorem 1. When applied on any given traffic state $s \in S$, the liveness-assessment algorithm that was described in the earlier parts of this section will terminate in finite time and will return a correct classification of the considered state s.

Proof (sketch): The imposed space limits for this document do not allow for an expansive proof for this theorem. But the finiteness of the computation of the proposed algorithm can be seen from the following facts: (i) Every major iteration of the algorithm in every direction of the underlying search tree will strictly reduce the number of the maximal chains in the underlying DAGs $\mathcal{U}(\hat{G}(s_i))$. (ii) The possible branches at each branching node of the underlying search tree are finite. (iii) Each basic operation executed by the algorithm is also a finite computation.

On the other hand, the correctness of the presented algorithm results from the following elements: (i) the semantics that are encoded in the deployed DAGs $\mathcal{U}(\hat{G}(s_i))$; (ii) the layered structure that is superimposed on these DAGs,

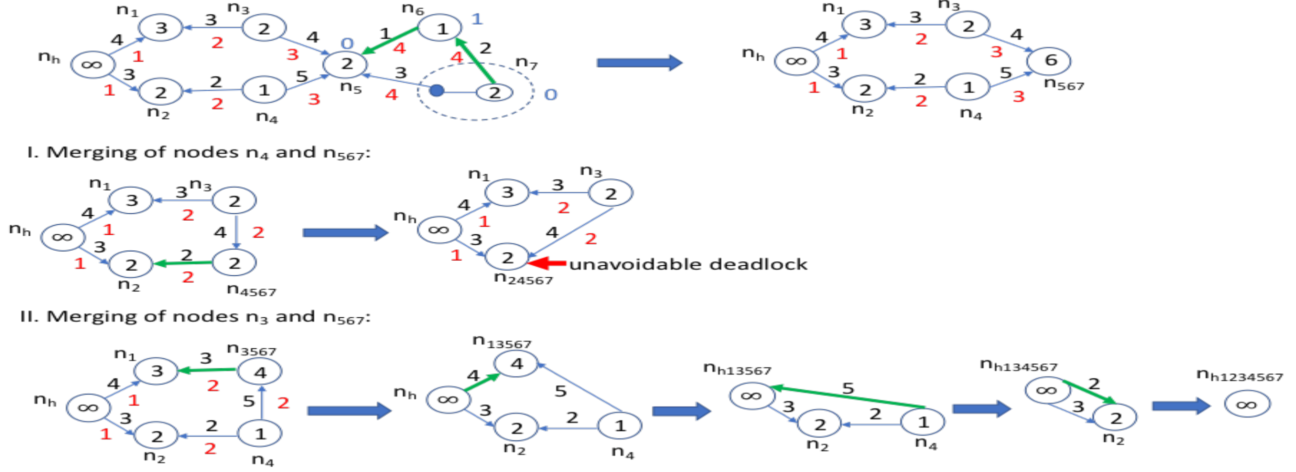


Fig. 2. The execution of the presented algorithm on the example of Section 4. Arrows annotated in green correspond to the feasible “producer” merger at each iteration that leads to the next DAG.

and the way that this structure is employed by the algorithm in order to ensure that the free capacity in each layer is utilized to the fullest possible extent in order to facilitate pertinent mergers in the lower-numbered layers; and (iii) the notion of the “feasible producer merger” that is pursued during the processing of the various layers of the DAGs $\mathcal{U}(\hat{G}(s_i))$, and the way that this processing is organized through the recursion of Equation 1 and the accompanying logic that was described in the corresponding part of this section. □

Complexity considerations: When applied on a traffic state s with a DAG $\mathcal{U}(\hat{G}(s))$ possessing the tree structure of Reveliotis and Masopust (2020), the presented algorithm essentially reduces to the algorithm that is presented in that previous work and it runs with a polynomial complexity with respect to the size of the underlying transport system. But for more general traffic states s, the branching that might take place during the execution of the algorithm, and the backtracking that will ensue from this branching, do not allow the claiming of such a polynomial worst-case complexity. It is expected, however, that the extent of this branching will be rather limited during any particular run of this algorithm. Furthermore, we remind the reader that the algorithm will exit as soon as it gets a single search path with a positive outcome, which alleviates even further the expected empirical complexity of the algorithm. Finally, the DAGs $\mathcal{U}(\hat{G}(s_i))$, which are the primary objects that are processed by the considered algorithm, are of a much smaller size than the size of the guidepath graph G that determines the size of the underlying transport system. Hence, even though not polynomial in the strict sense, the presented algorithm is still an efficient solution to the considered problem.

5. CONCLUSIONS

This work has presented a novel algorithm for assessing the liveness of any traffic state s that will arise in a class of guidepath-based transport systems that model the operation of various contemporary MHS. The algorithm is based on a pertinent representation and processing of the qualitative dynamics of the underlying transport system with respect to traffic liveness, and it is expected to have a very benign empirical computational complexity with respect to the size of the underlying guidepath network. However, the worst-case computational complexity of the algorithm might not be polynomially bounded with respect to the size of this network. In fact, the worst-case computational complexity of assessing the liveness of the considered traffic states remains an open issue, and it is part of our ongoing investigations.

Acknowledgement

The first author was partially supported by NSF grant ECCS-1707695. The second author was supported by RVO 67985840, the Czech Science Found. grant GC19-06175J, and the INTER-EXCELLENCE project LTAUSA19098.

REFERENCES

Ahuja, R.K., Magnanti, T.L., and Orlin, J.B. (1993). *Network Flows: Theory, Algorithms and Applications.* Prentice Hall, Englewood Cliffs, NJ.

Cassandras, C.G. and Lafortune, S. (2008). *Introduction to Discrete Event Systems (2nd ed.).* Springer, NY,NY.

Fanti, M.P. (2002). Event-based controller to avoid deadlock and collisions in zone-controlled AGVS. *Intl. Jrnl Prod. Res.*, 40, 1453–1478.

Heragu, S.S. (2008). *Facilities Design (3rd ed.).* CRC Press.

Reveliotis, S. (2017). Logical Control of Complex Resource Allocation Systems. *NOW Series on Foundations and Trends in Systems and Control*, 4, 1–223.

Reveliotis, S. and Masopust, T. (2019). Some new results on the state liveness of open guidepath-based traffic systems. In *27th Mediterranean Conference on Control and Automation (MED 2019).* IEEE.

Reveliotis, S. and Masopust, T. (2020). Efficient liveness assessment for traffic states in open, irreversible, dynamically routed, zone-controlled guidepath-based transport systems. *IEEE Trans. on Automatic Control*, 65.

Reveliotis, S.A. (2000). Conflict resolution in AGV systems. *IIE Trans.*, 32(7), 647–659.

Wu, N. and Zhou, M. (2001). Resource-oriented Petri nets in deadlock avoidance of AGV systems. In *Proceedings of the ICRA'01*, 64–69. IEEE.