Logist. Res. (2016) 9:20
DOI 10.1007/s12159-016-0147-6

ORIGINAL PAPER

Modeling of decentralized processes in dynamic logistic networks by means of graph-transformational swarms

Larbi Abdenebaoui[1] · **Hans-Jörg Kreowski**[1]

Received: 16 December 2015 / Accepted: 10 August 2016 / Published online: 31 August 2016

Abstract In this paper, we propose to employ the framework of graph-transformational swarms for the modeling of dynamic logistic networks with decentralized processing and control. The members of a graph-transformational swarm act and interact in a common environment graph with massive parallelism of rule-based activities according to local control conditions and a global cooperation condition. This corresponds directly to the logistic hubs and their processes in a logistic network where the processes run simultaneously and autonomously with a proper way of coordination. This covers also dynamic changes on the network structures as the members of a swarm can change the environment anyhow. The approach is illustrated by the modeling of automated guided vehicles.

Keywords Dynamic logistic networks · Decentralized control · Automated guided vehicles · Graph-transformational swarms · Swarm computation · Graph transformation

This article is part of a focus collection on “Dynamics in Logistics: Digital Technologies and Related Management Methods.”

✉ Larbi Abdenebaoui
larbi@informatik.uni-bremen.de

Hans-Jörg Kreowski
kreo@informatik.uni-bremen.de

[1] University of Bremen, P.O. Box 330440, 28334 Bremen, Germany

1 Introduction

As logistic networks get larger and larger and more and more complex, they become more difficult to handle and to control. The traditional central control does not work flexibly and efficiently enough in any case so that one must look for alternative approaches. This applies particularly if the logistic network may change dynamically. One of the most significant current paradigms that faces this complexity is the so-called autonomous control approach (cf. [11]). This approach proposes that each logistic object such as a container or an automated guided vehicle receives its own computing processor and makes its decision autonomously. Therefore, the components can react locally and quickly to changes in the environment. However, a major challenge within this kind of decentralized approach is how the individuals act and cooperate with each other to reach a desired global goal. In this paper, we introduce and discuss graph-transformational swarms as a formal modeling approach to dynamic logistic networks with decentralized control. As an illustrative example, we consider and discuss the routing problem of automated guided vehicles.

The concept of graph-transformational swarms combines the ideas of swarm computing and the methods of graph transformation. The basic framework is introduced in [1], where a simple ant colony, cellular automata, and discrete particle systems are modeled to demonstrate the usefulness and flexibility of the approach. A graph-transformational swarm consists of members that act and interact simultaneously in an environment, which is represented by a graph. The members are all of the same kind or of different kinds. Kinds and members are modeled as graph-transformational units (see, e.g.,[15]); each unit consists of a set of graph-transformational rules specifying the capability of members and a control condition which regulates the application of rules.

This paper is organized as follows. In Sect. 2, graph-transformational swarms are recalled starting with the basic concepts of graph transformation. Section 3 discusses how graph-transformational swarms can be used to model dynamic logistic networks. To illustrate this, essential aspects of the routing problem of automated guided vehicles are modeled as graph-transformational swarms in Sect. 4. Section 5 concludes the paper.

2 Graph-transformational swarms

This section recalls the concept of graph-transformational swarms as introduced in [1] starting with the basic components of the chosen graph-transformational approach as far as needed in this paper (for more details, see, e.g., [8, 14, 16, 20]).

2.1 Basic concepts of graph transformation

2.1.1 Graphs and rules

A *(directed edge-labeled) graph* $G = (V, E, s, t, l)$ consists of a set V of *nodes*, a set E of *edges* such that every edge is directed and labeled, i.e., $s : E \to V$ and $t : E \to V$ are mappings assigning a *source* $s(e)$ and a *target* $t(e)$ to each $e \in E$ and $l : E \to \Sigma$ is a *labeling* function for some labeling alphabet Σ. If the target is equal to the source, then the edge is called a *loop*. In the following, we call a node an i-node if it has an i-loop for $i \in \Sigma$. The components V, E, s, t, and l of G are also denoted by V_G, E_G, s_G, t_G, and l_G, respectively.

Given two graphs $G = (V, E, s, t, l)$ and $H = (V', E', s', t', l')$, a *graph morphism* $g : G \to H$ is given by two mappings $g_V : V \to V'$ and $g_E : E \to E'$ such that $s'(g_E(e)) = g_V(s(e))$, $t'(g_E(e)) = g_V(t(e))$ and $l'(g_E(e)) = l(e)$ for all $e \in E$. The image $g(G) = (g_V(V), g_E(E), s'', t'', l'')$ where s'',t'' and l'' are restrictions of s', t' and l' to the subsets $g_E(E)$ and $g_V(V)$ is called a *match* of G in H. If g_E and g_V are inclusions, G is called a *subgraph* of H, denoted by $G \subseteq H$. In particular, a match of G in H is a subgraph. It should be noted that the graph morphisms yielding matches are not assumed to be injective.

A *rule* $r = (L, K, R)$ consists of three graphs, the *left-hand side* L, the *gluing graph* K, and the *right-hand side* R such that $L \supseteq K \subseteq R$. In this paper, we consider rules that manipulate only edges (i.e., the nodes are neither deleted nor added). And when depicted, the gluing graph is omitted using the same relative positions of nodes in L and R. The edges in L and R that have the same sources, targets, labels, and form are also not changed by the rules. K can be identified in this way as the identical parts of L and R.

In order to permit more flexibility in modeling, we consider in this paper rules with negative application conditions. This extends the notion of rules permitting to express what should not be present in a graph in order to apply a given rule. A *rule with negative application condition* $r = (N, L, K, R)$ consists of four components such that (L, K, R) is a rule and N is the associated *negative application condition* composed from a finite set of graphs $N = \{C_1, \ldots, C_k\}$ called each a *negative context* such that $L \subset C_i$ for $i \in [k]$. Every negative context specifies a *negative part* $C_i - L$ which consists of the items of C_i that do not belong to L.

We depict a rule $r = (N, L, K, R)$ with $N = \{C_1, \ldots, C_k\}$ as $N \longrightarrow R$ where N is represented as a graph with subgraph L and extra information such that the negative contexts are identified. In this representation, the dashed items of N belong to the negative parts, the remainder is L. In one case in Fig. 8, the negative part contains two edges which are enclosed by a dotted line. In all other cases, the negative part contains a single edge. In this way, all negative contexts are easily recognized

Figure 1 shows the rule *move* which is used in a modified form in Sect. 4 to model the movement of an automated guided vehicle with the name a having the target T which is represented by an edge labeled by a, T. The (T, d)-edge indicates the direction in which the target T can be reached in the distance d. In this rule, there are two negative contexts. The first one is specified by the negative part consisting of the dashed edge labeled by a'', T'' and its source node. It means that the vehicle a can move into the new position only if there is no other (concurrent) vehicle that can occupy the same next position. The second negative context is specified by the negative part that consists of the dashed edge labeled by a', T' requiring that the vehicle a can move into the new position only if there is no other vehicle present there. The gluing graph consists of the three nodes and the edge labeled by T, d.

The application of a rule $r = (L, K, R)$ to a graph G replaces a match of L in G by R such that the match of K is kept. More explicitly, let $g : L \to G$ be the graph morphism yielding the match $g(L)$. Then, the resulting graph H is obtained by removing $g(L) - g(K)$ from G and adding $R - K$. All edges keep theirs sources, targets, and labels with the exception of edges in $R - K$ with sources or

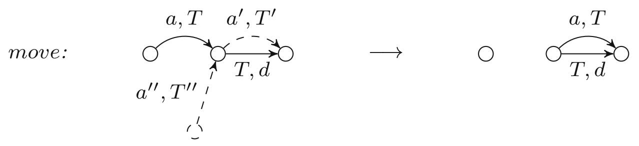


Fig. 1 A graph-transformational rule

targets in K. If $e \in E_R - E_K$ with $s_R(e) \in V_K$ (or $t_R(e) \in V_K$), then $s_H(e) = g_V(s_R(e))$ (or $t_H(e) = g_V(t_R(e))$ resp.).

A rule with negative condition (N, L, K, R) with $N = \{C_1, \ldots, C_k\}$ is applied in the same way as (L, K, R) provided that, for $i \in [k]$, the match of L cannot be extended to C_i. This means that a rule can only be applied if none of its negative contexts is around.

Hence, the application of the rule *move* moves forward an (a, T)-edge (i.e., the target node becomes the source node) provided that in the new position, there is no other (a', T')-edge. And there is no other (a'', T'')-edge having the same target node.

A rule application is denoted by $G \underset{r}{\Longrightarrow} H$ where H is the resulting graph and called a *direct derivation* from G to H.

As an example of a rule application, let us consider the graph G with the explicitly given subgraph in Fig. 2 (The dots in the graph indicate that the graph G may actually contain more items than the depicted subgraph). There is an (a, T)-edge meaning that the vehicle a has the target T. Moreover, there are four edges ahead, but one is occupied by vehicle a' and another one is not accompanied by a (T, d)-edge so that target T is not reachable with minimal distance in this direction. Two options remain to complete a match of the left-hand-side of rule *move* that avoid the negative context. One is chosen non-deterministically to derive the graph H where the vehicle a is moved forward by one edge. Continuing in this way, the vehicle a will reach its destination T eventually following always the directions with minimal distance.

A sequence $G = G_0 \underset{r_1}{\Longrightarrow} G_1 \underset{r_2}{\Longrightarrow} \cdots \underset{r_m}{\Longrightarrow} G_m = H$ is called a *derivation* from G to H of length m.

Given the rules $r_i = (N_i, L_i, K_i, R_i)$ for $i = 1, \ldots, n$, the *parallel rule* $p = (L, K, R)$ is given by the disjoint unions of the components, i.e., $L = \biguplus_{i=1}^{n} L_i, K = \biguplus_{i=1}^{n} K_i, R = \biguplus_{i=1}^{n} R_i$. If $g_i : L_i \to G$ for $i = 1, \ldots, n$ are some graph morphisms from L_i in some graph G, then this induces a graph morphism $g : L \to G$ defined, for all $i = 1, \ldots, n$, by $g_V(v) = g_{i,V}(v)$ for all $v \in V_{L_i}$ and $g_E(e) = g_{i,E}(e)$ for all $e \in E_{L_i}$. Therefore, matches of parallel rules are composed of matches of their component rules. As we use parallel rules only in this way, there is no need to carry the negative application conditions over to the level of parallel rules. Negative application conditions are only checked for component rules.

Two direct derivations $G \underset{r}{\Longrightarrow} H_1$ and $G \underset{r'}{\Longrightarrow} H_2$ of rules r and r' are *(parallel) independent* if the corresponding matches intersect only in gluing items. Due to the parallelization theorem in [13], we can use the following fact: A parallel rule $p = \sum_{i=1}^{n} r_i$ can be applied to G if the rules r_i for $i = 1, \ldots, n$ can be applied to G and the matches are pairwise independent. This allows the use of massive parallelism in the context of graph transformation based on local matches of component rules which are much easier to find than matches of parallel rules.

2.1.2 Control conditions and graph class expressions

A *control condition* C is defined over a finite set P of rules and specifies a set *SEM(C)* of derivations. Typical control conditions are regular expressions over P. The regular expressions have the form r for some rule r or $e; e'$ or $e|e'$ or e^* where e and e' are regular expressions themselves. The satisfaction of a control condition in the form of a regular expression is defined as follows. A direct derivation applying r satisfies r. A derivation satisfies $e; e'$ if an initial section of the derivation satisfies e and the rest e'. It satisfies $e|e'$ if it satisfies e or e'. It satisfies e^* if it satisfies e^n for some $n \in N$ where $e^0 = \lambda$ and $e^{(n+1)} = e^n; e$. A derivation of length 0 satisfies λ. Alternatively to e^*, $e!$ is used. A derivation satisfies $e!$ if it satisfies e^n, but not $e^{(n+1)}$, i.e., e is followed as long as possible. We use also priority conditions of the form $r < r'$ requiring that r can only be applied if r' is not applicable. Moreover, control conditions can be combined by logical operations of which we use the conjunction and with the obvious meaning: $C\&C'$ are satisfied if C and C' are satisfied. The expression $e|e'$ can be restricted by the priority condition $e < e'$ which requires that e is applied only if e' is not applicable. Other examples of control conditions that are used in this paper

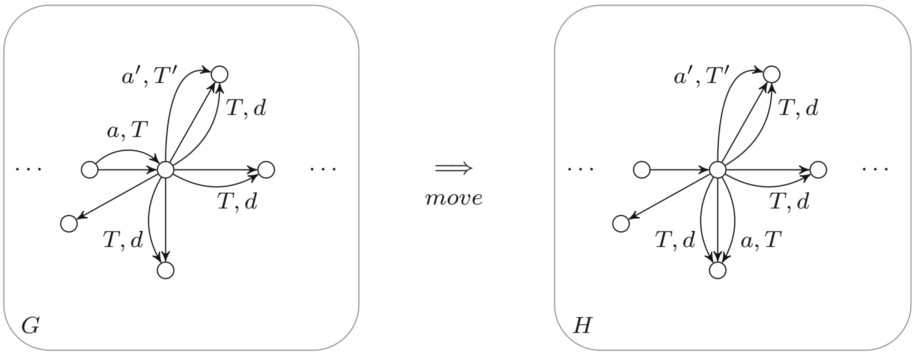


Fig. 2 A rule application

are the expression $\|r\|$ and $[r]$. $\|r\|$ requires that a maximum number of rule r be applied in parallel. $[r]$ requires that the rule r may be applied or not.

A *graph class expression* X specifies a set of graphs denoted by $SEM(X)$. We use the graph class expressions *distance* and $id-$looped(distance). The set $SEM(distance)$ contains all graphs without loops and without parallel edges (i.e., loop-free and simple graphs) where each edge is labeled with a distance (i.e., a value $d \in \mathbb{N}$). The set $SEM(id-\text{looped}(\text{distance}))$ contains all graphs that are obtained from the graphs in *SEM*(*distance*) as follows: The nodes are numbered from 1 to the number of nodes, and every node gets a loop labeled with its number. These graphs are called *id-looped distance graphs*. While the underlying distance graphs provide the significant information, the *id*-loops are added for technical reasons because they allow a direct access to nodes via rule applications.

2.1.3 Graph-transformational units

A *graph-transformational unit* is a pair $gtu = (P, C)$ where P is a set of rules and C is a control condition over P. The *semantics* of gtu consists of all derivations of the rules in P allowed by C. A unit gtu_0 is *related* to a unit gtu if gtu_0 is obtained from gtu by renaming identifiers and relabeling edges. The set of units related to gtu is denoted by $RU(gtu)$.

2.2 Graph-transformational swarms

A graph-transformational swarm consists of members of the same kind or of different kinds. All members act simultaneously in a common environment represented by a graph. The number of members of each kind is given by the size of the kind. While a kind is a graph-transformational unit, the members of this kind are modeled as units related to the kind so that all members of same kind are alike.

A swarm computation starts with an initial environment. It consists of iterated rule applications requiring massive parallelism meaning that each member of the swarm applies one of its rules in every step. The choice of rules depends on their applicability and the control conditions of the members as well as on a cooperation condition. Moreover, a swarm may have a goal given by a graph class expression. A computation is considered to be successful if an environment is reached that meets the goal.

Definition 1 (*swarm*) A *swarm* is a system $S = (I, K, s, m, c, g)$ where I is a graph class expression specifying the set of *initial environments*, K is a finite set of graph-transformational units, called *kinds*, s associates a *size* $s(k) \in \mathbb{N}$ with each kind $k \in K$, m associates a family of *members* $(m(k)_i)_{i\in[s(k)]}$ with each kind $k \in K$ with $m(k)_i \in RU(k)$ for each $i \in [s(k)]$, c is a control condition called *cooperation condition*, and g is a graph class expression specifying the *goal*.[1]

A swarm may be represented schematically displaying the components *initial, kinds, size, members, cooperation*, and *goal* followed by their respective values.

Definition 2 (*swarm computation*) A *swarm computation* is a derivation $G_0 \underset{p_1}{\Longrightarrow} G_1 \underset{p_2}{\Longrightarrow} \cdots \underset{p_q}{\Longrightarrow} G_q$ such that $G_0 \in SEM(I)$, $p_j = \sum_{k\in K} \sum_{i\in[s(k)]} r_{j_{ki}}$ with a rule $r_{j_{ki}}$ of $m(k)_i$ for each $j \in [q]$, $k \in K$ and $i \in [s(k)]$, and c and the control conditions of all members are satisfied. The computation is *successful* if $G_q \in SEM(g)$.

That all members must provide a rule to a computational step is a strong requirement because graph-transformational rules may not be applicable. In particular, if no rule of a swarm member is applicable to some environment, no further computational step would be possible and the inability of a single member stops the whole swarm. To avoid this global effect of a local situation, we assume that each member has the empty rule $(\emptyset, \emptyset, \emptyset, \emptyset)$ in addition to its other rules. The empty rule gets the lowest priority and is only applied if no other rule of the member can be applied or is allowed by the control condition of the member or the cooperation condition of the swarm. In this way, each member can always act and is no longer able to terminate the computation of the swarm. In this context, the empty rule is called *sleeping rule*. It can always be applied, is always parallel independent with each other rule application, but does not produce any effect. Hence, there is no difference between the application of the empty rule and no application within a parallel step.

3 From swarms in nature to logistic networks as graph-transformational swarms

In this section, we argue that graph-transformational swarms as introduced in the previous section are appropriate means to model dynamic logistic networks. Several approaches to swarm computation including graph-transformational swarms mimic swarms in nature as pointed out in Sect. 3.1 in more detail. The interesting aspect is that already swarms in nature solve problems closely related to logistics. Moreover and more interesting in the context of this paper, a closer look the other way round at dynamic logistic networks in Sect. 3.2 reveals that they can be considered as graph-transformational swarms.

[1] $[n] = \{1, \ldots, n\}$.

3.1 Relating swarms in nature with logistics

The proposed framework is inspired by the swarm behavior in nature which describes the group behavior of social animals. Several studies agree on the assumption that the swarm behavior results from relatively simple rules on the individual level (see, e.g., [4–6, 19]). In biology, the underlying mechanism is also known as *self-organization*: The individuals in the group interact locally with other group members and have no knowledge of the global behavior of the entire group. Furthermore, all members play the same role without any hierarchical structure [4].

Using swarm behavior, social animals solve continuously complex problems. For instance, ant colonies as well as bee hives build nests and manage the resources inside it. Furthermore, they forage for food, transporting it in an efficient and flexible way. Schools of fishes and flocks of birds travel over long distances. Obviously, such phenomena have logistic aspects. Therefore, it is somewhat evident that the behavior of swarms in nature inspires to introduce concepts of artificial swarms and swarm computation that are based on the idea of self-organization to solve logistic problems. One encounters some approaches to swarm computation in the literature (see, e.g., [2, 3, 9, 12, 18]) where logistic problems are solved as typical examples like the shortest path problem, the traveling-salespersons problem and others. One may summarize that the passage from swarms to logistics is not very long.

3.2 Dynamic logistic networks as graph-transformational swarms

On the other hand, consider dynamic logistic networks. Their underlying structures consist of nodes and connecting edges. The nodes represent logistic hubs of different types such as production sites, storage facilities, and car pools or—on a more detailed level—packages, containers, cars, and trucks, and the edges represent transport lines or information channels or the like. Without loss of generality, one can assume that there is always some start structure. To manage the material and information flows in a logistic network, various logistic processes are running. If the network is large and widely distributed, then it may not be meaningful to control the processes centrally. Alternatively, the logistic processes in the network may run simultaneously and independently of each other each performing its own actions and following its own autonomous control. But such a decentralized control requires coordination and cooperation whenever material or information must be exchanged carrying out the overall tasks. To coordinate autonomous logistic processes in a network in such a way that the cooperation works properly, becomes even more difficult if the network structure is dynamically changing. One needs appropriate modeling methods like those provided by graph-transformational swarms.

Table 1 Correspondence between dynamic logistic networks and graph-transformational swarms

Dynamic logistic network	Graph-transformational swarm
Underlying structure	Environment graph
Types of logistic entities	Kinds
Logistic entities	Members
Possible actions	Rules
Autonomous control	Control conditions
Start structures	Initial environments
Coordination	Cooperation conditions
Tasks	Goals
Simultaneous and decentralized processing	Massively parallel rule application

The underlying structures of dynamic logistic networks are defined as graphs so that they correspond directly to environment graphs of graph-transformational swarms where the initial environments play the role of the start structures. The various types of logistic entities like hubs, sites, carriers, and containers together with the actions that are performed on them or affect them can be seen as kinds so that the entities themselves are the swarm members. In particular, the possible process actions correspond to the rules, and the autonomous control is reflected by the control conditions. Finally, the coordination of the processes running on the logistic networks is embodied by the cooperation conditions and the overall tasks by the goals.

Summarized in Table 1, there is a very close relationship between the main features of dynamic logistic networks and the syntactic components of graph-transformational swarms. Moreover and most interestingly, the idea of autonomous processes that run simultaneously and decentralized in a logistic network is well reflected on the semantic level of graph-transformational swarms as all the members act always in parallel.

3.3 The potentials of the approach

We propose in this paper to model dynamic logistic networks by means of graph-transformational swarms. In the previous subsection, one can see that the notion of such swarms covers all the main features one expects and finds in dynamic logistic networks. Nevertheless, one may wonder which particular potentials and advantages this approach provides:

1. The concept of graph-transformational swarms offers a formal framework with a precise mathematical semantics based on massive parallelism of rule applications.

2. As the environments are graphs and the processing is modeled by graph-transformational rules specified by four graphs each, the approach provides a fundament for visualization so that it can be considered as a visual modeling approach.
3. In fact, graph-transformational swarms can be executed on graph-transformational engines like GrGen.NET (see [10]) or AGG (see [23]) so that visual simulation is possible for illustrations, tests, and experiments of various kinds. In the next section for example, we use illustrations in Figs. 5 and 9 generated from GrGen.-NET. They visualize the computational steps in a simple environment in order to make it easier for a reader to understand how the developed swarm behaves. Moreover, the implementation in GrGen.NET allows us a visual testing using different graphs.
4. The formal semantics is based on derivations which are sequences of rule applications. Therefore, a proof technique is provided by induction on the lengths of derivations.
5. If one fixes the initial environment and bounds the lengths of derivations, then the behavior of graph-transformational swarms can be translated into formulas of the propositional calculus so that SAT-solvers can be employed for automatic proving of properties, as far as they are expressible in propositional calculus. A typical correctness property one would like to prove in this way is: Will the goal be reached? Another property of interest that can be proved in this way is deadlock freeness.
6. The approach is very flexible and generic because all the modeling concepts can be chosen from a variety of possibilities. This applies to the kind of graphs which may be directed or undirected, labeled or unlabeled, connected, simple, etc. It applies similarly to the kind of rules, of control conditions and of graph class expressions. The actual choice may depend on the application at hand or the taste of the network designers.
7. Graph-transformational swarms do not need extra features to make logistic networks dynamic, i.e., to allow the modeling of dynamic changes in the underlying structure. The members of the swarm perform their tasks by applying rules to the environment graph. This includes the possibility of members to change the environment structurally.

4 Routing of AGVs by a graph-transformational swarm

In this section, we propose a solution to the routing problem of the automated guided vehicles (AGVs) using the notion of graph-transformational swarms. Automated guided vehicles are driverless transportation engines that follow traditionally guide paths like lines on the ground. Their use is expanding rapidly in the last decades. Beside the classical application in small manufacturing systems, nowadays, the tendency is to use AGVs more and more for transport in highly complex systems including external areas like container terminals (for a general overview, see, e.g., [17, 26]). One of the important problems that a designer of an AGV system faces in complex areas is the collision-free routing problem. The classical way to solve this problem is the central time windows planning (see, e.g., [22, 24, 25]). However, the tendency in the last years is to explore more decentralized approaches (e.g., [21, 27]). In the same vein, this section proposes a decentralized solution using the notion of graph-transformational swarms.

4.1 The routing swarm

We model the infrastructure where the AGVs operate as an $id-$looped distance graph. In a graphical representation, the nodes correspond to the ends or intersections of paths including important stations like pickup and delivery locations. The edges represent the paths or segments of paths in the infrastructure depending on their lengths. The distance of an edge can correspond to the distance of the corresponding path or to some cost of traversing it.

We propose a solution based on two stages. The first one consists of the preparation of the layout in a such way that the AGVs follow later only local information. The second one consists of the navigation process of the AGVs depending on an arbitrary task assignment.

The parameter m is the number of AGVs and can be chosen freely. The swarm has four kinds: *preparator*, *resolver*, *assigner*, and *navigator*. Their sizes are n,n, m, and m, respectively, where n is the number of nodes in the underlying graph $G \in SEM(id{-}\text{looped}(\text{distance}))$. The members are obtained by relabeling in such a way that every node in the graph gets assigned two members, one of kind *preparator* and the other of kind *resolver*, and similarly, every AGV gets assigned a member of kind *assigner* and a member of kind *navigator*. How relabeling is achieved is described below in the detailed introduction of the kinds. Syntactically, the cooperation condition is a regular expression as introduced in Sect. 2.1.2, but for kinds instead of rules. Semantically, the used cooperation condition requires that *preparator* is applied realizing the layout preparation followed by an arbitrary repetition of assignments each followed by an arbitrary number of conflict resolving and navigation steps. The application of a kind means that all members of this kind act in parallel according to theirs' own control conditions while all other members "sleep", i.e., they apply their sleeping rule by

routing(m)

initial:	$\mathit{id\text{-}looped}(\mathit{distance})$
kinds :	$\mathit{preparator}, \mathit{resolver}, \mathit{assigner}, \mathit{navigator}$
size :	$n = \#\mathit{nodes}, n, m, m$
members:	$\mathit{preparator}_i$ for $i \in [n]$
	$\mathit{resolver}_j$ for $j \in [n]$
	$\mathit{assigner}_k$ for $k \in [m]$
	$\mathit{navigator}_l$ for $l \in [m]$
coop:	$\mathit{preparator}; (\mathit{assigner}; (\mathit{resolver}; \mathit{navigator})^*)^*$
goal:	*all vehicles arrived*

Fig. 3 Schematic representation of the graph-transformational swarm *routing*

default. The goal is that all AGVs reach their assigned targets. The swarm is schematically presented in Fig. 3.

As mentioned before, we have implemented the swarm *routing* in the graph-transformational tool GrGen.NET. The resulting computation steps of an experiment with an environment composed from a very small graph, and three AGVs are used in this section to accompany the explanation of the behavior of the swarm *routing*. Fig. 5 summarizes the layout preparation process, and Fig. 9 illustrates the remainder of the computation which consists of the assignment and the conflict-free navigation of the AGVs.

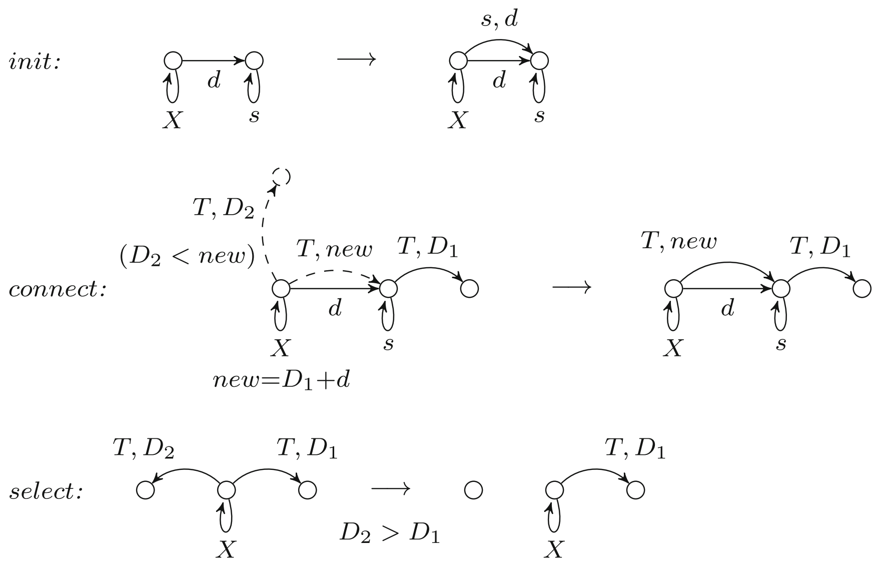
4.2 Layout preparation

The layout preparation equips the underlying graph with additional edges in such a way that every node in the graph can indicate to an AGV having the target T which next node can be visited to reach T with the minimal distance possible. Given an i-node, we code such an *indicator* as an outgoing edge e labeled with a pair T, D. We say that i has an indicator to T with the distance D using the successor s, where s is the target of e (i.e., $s = t(e)$). If D is minimal considering simple paths up to the maximal lengths l, we say that the indicator is l-minimal. If D is minimal considering all possible paths, then the indicator is optimal. A path composed from indicators to a target T is called an indicator path to T. If every node in the graph has only optimal indicators to every reachable node, then the graph is called *fully indicated*.

The members of kind *preparator* realize the layout preparation process. The kind *preparator* specified in Fig. 4 initializes this process with rule *init*. It adds an indicator in an X-node to a direct successor s provided that such indicator does not yet exist. The rule *connect* connects an X-node with an existing indicator path to T. It is applied if a direct successor s of X exists having an indicator to T with a distance D_1 provided that there is no other direct successor of X having the same target T with a distance D_2 such that $D_2 < D_1 + d$. The rule *connect* generates an indicator to T with the new distance D_1+d using the successor s. If an X-node has two indicators to a target T with different distances, the rule *select* deletes the one with the larger distance selecting in this way the best one to be kept. The control condition requires that the rule *init* is applied with maximum parallelism. Afterward, the rule *connect* is applied followed by *select* both with maximum parallelism. Because of the negative application condition of *init*, *init* is applied only once for a given successor node in the whole swarm computation while *connect* and *select* are iterated as long as possible according to the control condition of *preparator*.

Fig. 4 Unit *preparator*

preparator
rules:



control: $\|\mathit{init}\|; (\|\mathit{connect}\|; \|\mathit{select}\|)!$

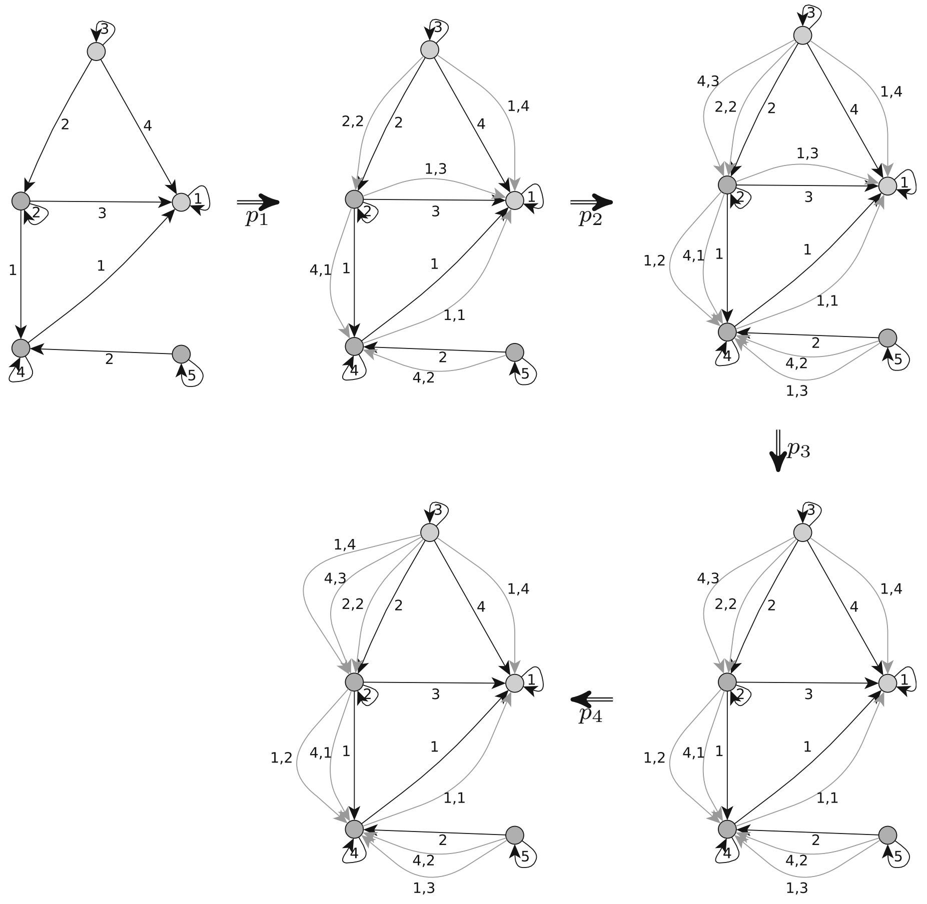


Fig. 5 A sample computation of the swarm *routing* illustrating the layout preparation process

In the swarm, there are n members of kind *preparator*. The member $preparator_i$ for $i \in [n]$ is obtained from *preparator* by relabeling all occurring X with i.

Then i becomes a fixed label in $preparator_i$. The role of the other labels must be explained now: They are placeholders for all possible values so that the rules are rather rule schemata that must be instantiated before they are applied. A control condition like $\|init\|$ means accordingly that the maximum number of the instantiations of the rule *init* must be applied in parallel. This mechanism that keeps the representation of rule sets small is used in all our examples of transformation units.

In the following, we describe how the members work together using the computation in Fig. 5 as illustrating example. In the first step, all members apply the rule *init* in parallel generating in every node indicators to all successor nodes (see the result of the derivation p_1 in the example). In the second step, the parallel application of the rule *connect* in parallel connects all nodes to construct indicator paths of length 2. It connects also those that are already connected to indicator path of length 1 if the new distance is smaller, then the old one (in the example, p_2 adds indicators in the nodes 2 and 3). In the third step, all members apply *select* in parallel deleting all indicators using path of lengths 2 and 1 that are not 2-minimal (p_3 deletes the indicator in 2 to 1 with distance 3 keeping the minimal indicator to 1 with distance 2). Note that a node can have more than one minimal indicator to the same target (in the example, the node 3 gets two indicators to 1 with the same distance 4). By induction, one can prove that in $2L - 1$ steps, all L-minimal indicators are constructed. If the longest path with a minimal distance is constructed, then the *preparator*-members cannot apply any rule anymore (except the sleeping rule). And the constructed indicators are optimal. Because the length of such a path is shorter or equal $n - 1$, the number of steps is bounded by $2n - 3$. Summarizing, the following correctness result holds:

Theorem 1 *Given an id-looped distance graph G, the swarm routing transforms it by the initial preparation phase in a fully indicated graph in a number of steps bounded by $2n - 3$ where n is the number of nodes in G.*

Note that the layout preparation process can be considered as a distributed version of the Dijkstra's shortest path algorithm (cf. [7]).

The behavior of the swarm in the layout preparation stage can also be interpreted as follows. In the first step, a change in the environment is introduced using the rules *init*. The swarm reacts by propagating backwards this information over all nodes combining the rules *connect* and *select* of all members of kind *preparator*. In more sophisticated versions of the underlying swarm, one can consider that additional changes can occur in the environment like suppression of indicator edges. This can simulate for example a traffic congestion. Such a change can be also handled in the same way by propagating the information backward to all concerned members. For illustration purposes, we keep the preparation process simple and show in the next subsection how the automated guided vehicles can use the generated information to navigate conflict-free to their assigned targets.

4.3 Assignment, conflicts resolving, and navigation

The kinds *assigner* and *navigator* model the task assignment and navigation process from the point of view of the AGVs. However, the task assignment has the most simple form serving solely the simulation purposes of the computational steps. The kind *resolver* models the conflict resolving from the point of view of a node where multiple AGVs have it as next destination and would like to visit the same next position which is determined by a direct successor of the underlying node.

In the following, we encode an AGV as an AGV-edge labeled by a, T, p where a is the name of the AGV, $T \in [n]$ corresponds to its assigned target and $p \in \mathbb{N}$ is its current priority. We call therefore a vector of nodes $\langle n_1, n_2 \rangle$, such that an indicator (T, d) from n_1 to n_2 exists, an AGV position. We consider that AGVs with target T can occupy such position with the restriction that at most one AGV can be present in a position in a given time. The priority is needed to resolve conflicts if more than one AGV compete for one position.

The kind *assigner*, as specified in Fig. 6, has just a single rule *assign* that creates a vehicle edge labeled with a, T, p between two arbitrary nodes provided that this position leads to the target T and is free and that the vehicle edge is not yet present in the whole graph. Note that the edges labeled by T, d in the rule *assign* have different forms meaning that they do not belong to the gluing graph of the rule. The control condition [*assign*] requires that the rule may be applied or not so that not every vehicle must be present any time. The members $assigner_j$ for $j \in [m]$ are obtained from the kind *assigner* by relabeling all occurring a with a_j and the a' by a_k for $j \neq k$.

The kind *resolver* has a single rule *reserve* (see Fig. 7). It reserves for an incoming AGV-edge labeled by a, T, p a next possible position $\langle X, s \rangle$ having an indicator T, d provided that the following four negative contexts are all satisfied. (1) There is no other concurrent AGV (represented in the rule by the incoming AGV-edge a_1, T_1, p_1) with a higher priority $(p_1 > p)$, and can visit too the position $\langle X, s \rangle$ (see the edge (T_1, d_1) in the rule). This negative context with two edges is bordered by a dotted line to indicate that the two parts should be satisfied together. (2) The position $\langle X, s \rangle$ is free: There is no other outgoing AGV-edge labeled by a_2, T_2, p_2 parallel to the (T, d)-edge. (3) There is no reservation a_3 for any other vehicle in the next position $\langle X, s \rangle$. (4) The AGV a has not yet a reservation: There is no outgoing edge labeled by a. The rule *reserve* adds an outgoing edge labeled by a parallel to the (T, d)-edge which indicates that the underlying position is reserved for the AGV a. The rule *reserve* may be applicable for two AGVs with the same priority both claiming the same next possible position $\langle X, s \rangle$. But the control condition requires that the rule is applied sequentially as long as possible so that only one of the potential reservations is chosen non-deterministically. The member $resolver_j$ for $j \in [n]$ is derived from *resolver* by relabeling all occurring X with j. This means in particular that reservations are done sequentially at the node with the j-loop, but in parallel for different nodes.

The kind *navigator*, which is specified in Fig. 8, contains three rules *wait*, *move*, and *arrive*. The rule *wait* increments the priority p with 1. The rule *move* is responsible of the forward movement of the AGV until the target is reached. It moves forward the AGV a with the target T following an a-edge (which is added by a *resolver* member). If the target node is reached, the rule *arrive* can be applied. The rule *arrive* deletes the AGV-edge signaling in this way to the task assigner that the AGV a is free for a new assignment. The control condition requires that one of the rules *move*, *arrive*, or *wait* is applied. Therefore, *wait* has the lowest priority. The member $navigator_k$ for $k \in [m]$ is obtained from *navigator* by relabeling all occurring a with a_k.

After the layout preparation, only members of kind *assigner*, *resolver*, and *navigator* are active. The *assigner* members create an arbitrary number less or equal m of AGV-edges in parallel. According to the parallelization theorem together with the fact that the T, d edges in the rule *assign* do not belong to the gluing graph, the positions of the created AGVs are pairwise different ensuring a conflict-free assignment. Afterward, all created AGV-edges act in parallel by moving forward or waiting depending on the decision of the *resolver* members, which

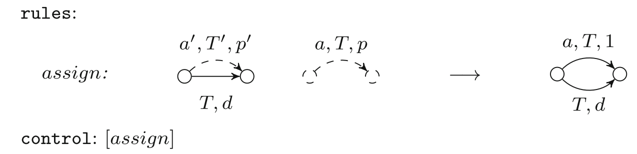


Fig. 6 Unit *assigner*

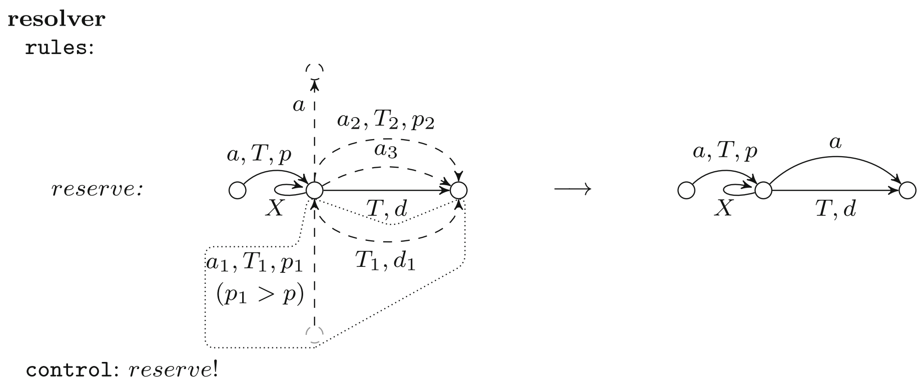


Fig. 7 Unit *resolver*

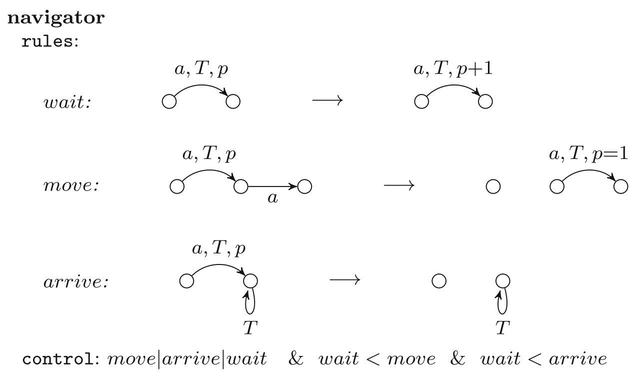


Fig. 8 Unit *navigator*

are present in every node to check for and to resolve conflicts. They reserve the next possible position of the AGVs based on their priorities. The AGVs with a reservation are moved forward setting their priorities to one, the others that arrive to their targets become their corresponding edge deleted, all others have to wait incrementing their priorities by one. If the number of repetition is high enough, the swarm reaches its goal, otherwise the process starts again by assigning new tasks to inactive vehicles. The swarm repeats this process until the goal is reached. Especially, we have the following result.

Theorem 2 *If the swarm routing reaches its goal, each AGV that has been assigned to a target reaches this target collision-free.*

Proof Consider a computation $G_0 \overset{*}{\Longrightarrow} G_n$ of the swarm *routing*. According to the cooperation condition, an initial section $G_0 \overset{*}{\Longrightarrow} G_i$ for some i prepares the initial environment G_0 into a graph with the properties stated in Theorem 1. And the tail $G_i \overset{*}{\Longrightarrow} G_n$ is composed from sections of the form $G_{k_j} \Longrightarrow G_{k_j+1} \overset{*}{\Longrightarrow} G_{k_{j+1}}$ for $i = k_1 < \cdots < k_m = n$, $m \geq 1$ where, for $j = 1, \ldots, m$, the first step is an *assign*-step and the remainder repeats *resolver*-steps followed by *navigator*-steps. For $m = 0$, this is the empty sequence. Then, the theorem can be proved by induction on m. For $m = 0$, no car moves so that no collision can happen. Let now the computation have $m{+}1$ *assign*-steps. Due to the induction hypothesis, the vehicles run collision-free for the first m *assign*-steps. The $(m{+}1)$-st *assign*-step adds some further AGVs, but only if none of these is already present and the edges where the vehicles are assigned are not occupied. All further steps are applications of the rule *reserve* alternated with the applications of the rules *wait*, *move*, and *arrive*. A collision would only happen whenever two AGVs move onto the same edge at the same time. But such collision is impossible because the entered edge is reserved before by exactly one vehicle as discussed in detail above in the explanations of the kinds. $\square$

The swarm *routing* is designed to solve conflict-free situations where two or more concurrent AGVs want to traverse the same node. However, it should be mentioned that the presented swarm does not handle deadlocks caused from circular waits. The characterization, detection, and avoidance of such situations should be treated in future work.

Figure 9 illustrates the computations in the navigation process of three AGVs a_1,a_2 and a_3 starting with the fully connected graph resulting from the preparation process in Fig. 5. In the first step in this stage, two AGVs are arbitrarily chosen to get assignments and the reminder is kept inactive. a_1 and a_3 get assigned, respectively, the targets 1 and 4 and the start positions $\langle 3, 1\rangle$ and $\langle 3, 2\rangle$ as a result of the application of the rule $p_5 = assign_1 + assign_3 + sleep_2$ where the indices in the rules correspond to the indices of the members that apply them. After the assignment, the member $resolver_2$ which resolves conflict in node 2 reserves the next position for the AGV a_3 through generating an a_3-edge while all other members of kind *resolver* apply their sleeping rule, i.e., $p_6 = reserve_2 + \sum_{i \in [5]\setminus\{2\}} sleep_i$. The rule $p_7 = arrive_1 + move_3 + sleep_2$ is applied making a_1 available for other assignments because it has already arrived at its target and moving forward the vehicle a_3 to its reserved position. The AGV a_2 is still sleeping in this step. At this point, the repetition of

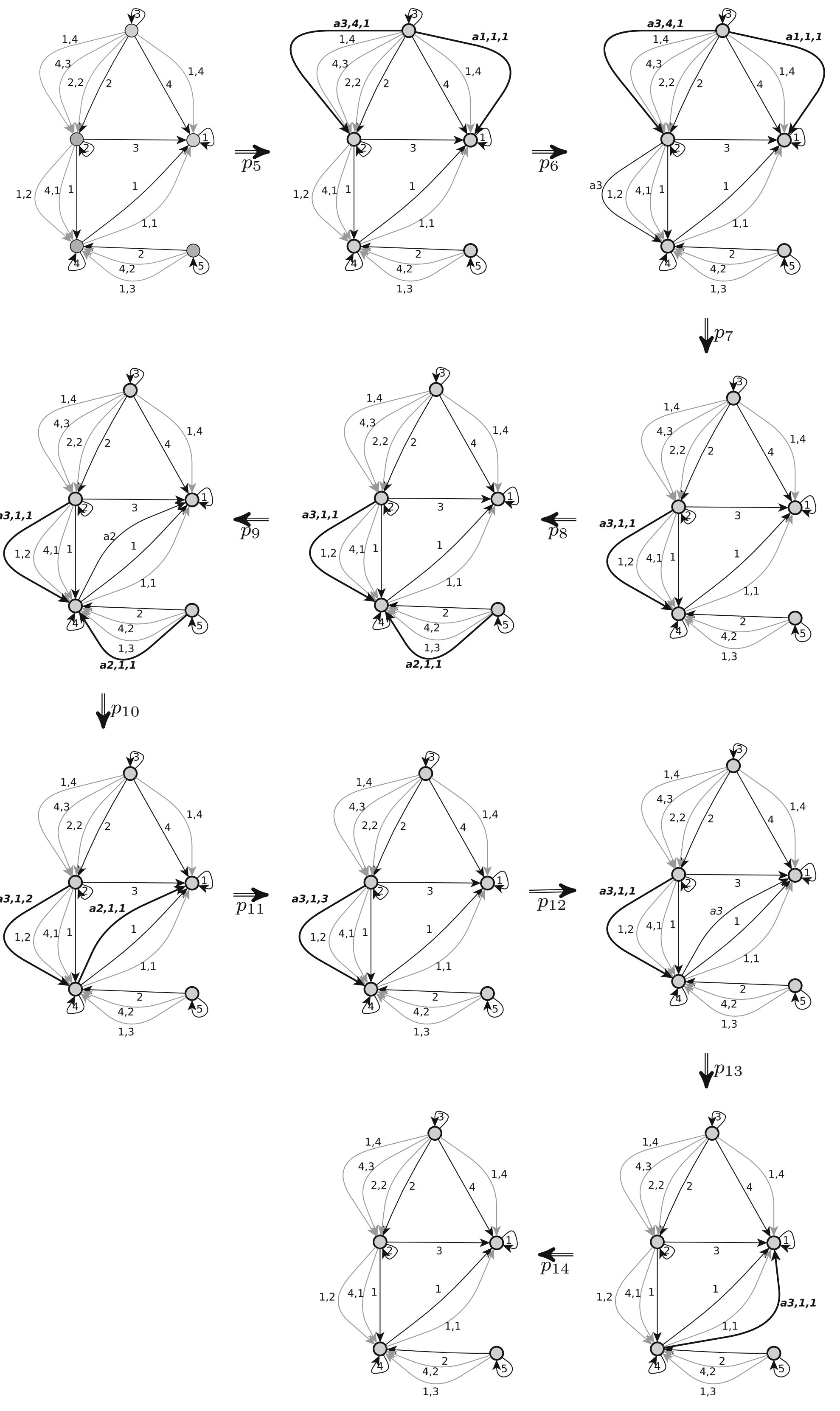


Fig. 9 A sample computation of the swarm *routing* illustrating the navigation process

resolving and navigation is finished allowing that the assignment starts again. $p_8 = assigner_2 + \sum_{i \in [3] \setminus \{2\}} sleep_i$ assigns the target 1 and the start position $\langle 5, 4 \rangle$ to the AGV a_2 The other members of kind *assigner* sleep because the a_1 is not chosen to be assigned and a_3 is already assigned. In the next step, the rule $p_9 = resolver_4 + \sum_{i \in [5] \setminus \{4\}} sleep_i$ chooses a_2 to move forward reserving the only possible next edge for it. This step illustrates the behavior of a member resolver in a situation where concurrent AGVs having the same maximal priority and want to traverse the assigned node. Namely one is chosen arbitrarily. Following this decision, a_2 moves forward and a_3 waits augmenting its priority by 1 as a result of the rule $p_{10} = move_2 + wait_3 + sleep_1$. The navigation rules yield that a_2 applies its *arrive* rule and that a_3 waits again because the next position is occupied, that is, $p_{11} = arrive_2 + wait_3 + sleep_1$. In the next two steps, $p_{12} = resolver_4 + \sum_{i \in [5] \setminus \{4\}} sleep_i$ reserves the next position for a_3 followed by $p_{13} = move_3 + \sum_{i \in [2]} sleep_i$ which moves it to the reserved position. In the last step, and because the last active AGV a_3 arrives at its target $p_{14} = arrive_3 + \sum_{i \in [2]} sleep_i$, the swarm reaches its goal.

5 Conclusion

In this paper, we have proposed to model dynamic logistic networks with decentralized processing and control by means of graph-transformational swarms. The members of such a swarm act and interact in a common environment graph. It is a rule-based approach, the semantics of which is based on massive parallelism according to local control conditions of the members and a global cooperation condition of the swarm as a whole. As we have discussed above, this corresponds to dynamic logistic networks with their logistic hubs and their processes which run simultaneously and autonomously with a proper way of coordination. We have sketched how automated guided vehicles and their routing can be modeled by a graph-transformational swarm as an illustrative example. In this example, we have demonstrated the capability of the approach regarding visualization in the design level as well as the computation level. Furthermore, we have provided two theorems using the advantage of the formal semantics of graph-transformational swarms. In order to shed more light on the significance of the approach, we will study the following topics in future research.

1. More case studies are needed including real applications. This would allow one to test the implementation of a logistic network against a formal specification by means of graph-transformational swarms rather than against informal or semiformal models or just against the intuition of the designers.
2. The use of tools must be made more comfortable. At the moment, one must adapt each graph-transformational swarm separately by hand to simulate and visualize it on a graph-transformational engine or to verify properties on a SAT-solver. By fixing the syntactic features of swarm modeling, one can construct translators into the tools so that the tools run automatically on swarms and simulate and verify logistic networks in this way.
3. It may be meaningful to translate the modeling concepts of graph-transformational swarms into explicit modeling concepts of dynamic logistic networks. In this way, modelers of networks do not need to make themselves familiar with the swarm ideas, and they could follow their intentions directly within the edifice of ideas of logistic networks.

Acknowledgments We are grateful to the anonymous reviewers for their valuable comments on former versions of this paper.

References

1. Abdenebaoui L, Kreowski H-J, Kuske S (2013) Graph-transformational swarms. In: Bensch S, Drewes F, Freund R, Otto F (eds) Fifth workshop on non-classical models for automata and applications-NCMA 2013, Umeå, August 13–August 14, Proceedings. Österreichische Computer Gesellschaft, pp 35–50
2. Blum C, Merkle D (eds) (2008) Swarm intelligence: introduction and applications. Natural computing series. Springer, New York
3. Bonabeau E, Dorigo M, Theraulaz G (1999) Swarm intelligence: from natural to artificial systems. Oxford University Press, Oxford
4. Camazine S, Franks NR, Sneyd J, Bonabeau E, Deneubourg J-L, Theraula G (2001) Self-organization in biological systems. Princeton University Press, Princeton
5. Couzin ID, Krause J (2003) Self-organization and collective behavior in vertebrates, volume 32 of advances in the study of behavior. Academic Press, Cambridge
6. Deneubourg JL, Aron S, Goss S, Pasteels JM (1990) The self-organizing exploratory pattern of the argentine ant. J Insect Behav 3(2):159–168
7. Dijkstra EW (1959) A note on two problems in connection with graphs. Numer Math 1(5):269–271
8. Ehrig H, Ehrig K, Prange U, Taentzer G (2006) Fundamentals of algebraic graph transformation (monographs in theoretical computer science. An EATCS series). Springer, Berlin
9. Engelbrecht A P (2006) Fundamentals of computational swarm intelligence. Wiley, New York

10. Geiß R, Kroll M (2008) GrGen.NET: a fast, expressive, and general purpose graph rewrite tool. In: Schürr A, Nagl M, Zündorf A (eds) Proceedings of 3rd international symposium on applications of graph transformation with industrial relevance (AGTIVE '07), volume 5088 of Lecture notes in computer science, pp 568–569
11. Hülsmann M, Scholz-Reiter B, Windt K (2011) Autonomous cooperation and control in logistics. Springer, Berlin
12. Kennedy J, Eberhart RC (2001) Swarm intelligence. Evolutionary computation series. Morgan Kaufman, San Francisco
13. Kreowski H-J (1977) Manipulationen von Graphmanipulationen. Ph.D. thesis, Technische Universität Berlin
14. Kreowski H-J, Klempien-Hinrichs R, Kuske S (2006) Some essentials of graph transformation. In: Ésik Z, Martín-Vide C, Mitrana V (eds) Recent advances in formal languages and applications, vol 25., Studies in computational intelligenceSpringer, Berlin, pp 229–254
15. Kreowski H-J, Kuske S, Rozenberg G (2008) Graph transformation units—an overview. In: Degano P, Nicola RD, Meseguer J (eds) Concurrency, graphs and models, essays dedicated to Ugo Montanari on the occasion of his 65th birthday, volume 5065 of Lecture notes in computer science (LNCS). Springer, New York, pp 57–75
16. Kreowski H-J, Kuske S, Rozenberg G (2008) Graph transformation units—an overview. In: Degano P, Nicola RD, Meseguer J (eds) Concurrency, graphs and models. Springer, New York
17. Le-Anh T, Koster MD (2006) A review of design and control of automated guided vehicle systems. Eur J Oper Res 171(1):1–23
18. Olariu S, Zomaya AY (2005) Handbook of bioinspired algorithms and applications. Chapman & Hall/CRC, Boca Raton
19. Partridge BL (1982) The structure and function of fish schools. Sci Am 246:114–123
20. Rozenberg G (ed) (1997) Handbook of graph grammars and computing by graph transformation. Foundations, vol 1. World Scientific, Singapore
21. Schwarz C, Sauer J (2012) Towards decentralised agv control with negotiations. In: Kersting K, Toussaint M (eds) Proceedings of the sixth starting AI researchers symphosium, volume 241 of frontiers in artificial intelligence and applications. IOS Press
22. Smolic-Rocak N, Bogdan S, Kovacic Z, Petrovic T (2010) Time windows based dynamic routing in multi-AGV systems. IEEE Trans Autom Sci Eng 7(1):151–155
23. Taentzer G (2000) Agg: a tool environment for algebraic graph transformation. In: in AGTIVE, ser. Lecture notes in computer science. Springer, pp 481–488
24. Taghaboni-dutta F, Tanchoco JMA (1995) Comparison of dynamic routeing techniques for automated guided vehicle system. Int J Prod Res 33(10):2653–2669
25. Ter Mors A, Witteveen C, Zutt J, Kuipers FA (2010) Context-aware route planning. In: Dix J, Witteveen C (eds) Multiagent system technologies, 8th German conference, MATES 2010, Leipzig, Germany, volume 6251 of Lecture notes in computer science. Springer, pp 138–149
26. Vis IF (2006) Survey of research in the design and control of automated guided vehicle systems. Eur J Oper Res 170(3):677–709
27. Weyns D, Holvoet T, Schelfthout K, Wielemans J (2008) Decentralized control of automatic guided vehicles: applying multi-agent systems in practice. In: Companion to the 23rd ACM SIGPLAN conference on object-oriented programming systems languages and applications, OOPSLA Companion '08, New York, ACM, pp 663–674