Autonomous Units for Solving the Traveling Salesperson Problem Based on Ant Colony Optimization[1]

Sabine Kuske, Melanie Luderer, Hauke Tönnies

University of Bremen, Department of Computer Science
P.O.Box 330440, D-28334 Bremen, Germany
e-mail: {kuske,melu,hatoe}@informatik.uni-bremen.de

Abstract Communities of autonomous units are rule-based and graph-transformational systems with a well-defined formal semantics. The autonomous units of a community act and interact in a common environment while striving for their goals. Ant colony systems consist of a set of autonomously behaving ants and are often employed as a metaheuristics for NP-hard logistic problems. In this paper, we demonstrate how communities of autonomous units can be used as a formal graph-transformational framework for modeling ant colony systems. As a first example we model an ant colony system for the Traveling Salesperson Problem as a community of autonomous units.

1 Introduction

In logistics, one is often faced with optimization problems that are NP-hard. One successful way towards efficient solutions of complex optimization problems is swarm intelligence that makes use of the emergent behavior of a set of autonomously and independently acting individuals. Ant colony optimization (ACO) algorithms are famous representatives of this type. ACO algorithms are inspired by the way how ants find short routes between food and their formicary. They have been shown to be well-suited not only for the solving of shortest path problems, but for a series of more complex problems, typically occurring in logistics (cf. (Dorigo and Stützle 2004)). In particular, the Traveling Salesperson Problem (TSP) plays an important role in ACO algorithms. It not only occurs in many logistic applications, it also serves to illustrate the basic features of ACO algorithms.

[1] The first and the third author would like to acknowledge that their research is partially supported by the Collaborative Research Centre 637 (Autonomous Cooperating Logistic Processes: A Paradigm Shift and Its Limitations) funded by the German Research Foundation (DFG).

This paper proposes to use communities of autonomous units (see e.g., (Hölscher et al. 2007) and (Hölscher et al. 2009)) as a formal graph-transformational and rule-based framework for modeling ACO algorithms. A community of autonomous units consists of a set of autonomous units, a global control condition, and an overall goal. Autonomous units are equipped with a set of rules, a set of auxiliary units, a control condition, and a goal. Independently from each other, they try to reach their individual goals by applying their rules and auxiliary units according to their control conditions. Applications of rules and auxiliary units transform the common environment, and autonomous units can react to these transformations. Hence, the autonomous units interact and act in a dynamically changing common environment. For solving the TSP, the ants are realized as autonomous units, the complete graph forms the common environment and the overall goal consists in finding the shortest Hamiltonian cycle in the graph.

The use of communities of autonomous units for ant colony optimization algorithms has the following advantages:

- It is not uncommon that realistic logistic problems, when solved by an ant colony optimization algorithm, require a great number of ants performing some non-trivial actions. Especially for complex problem solution strategies, it quickly becomes difficult to see whether the algorithm actually solves the problem. Community of autonomous units have a well-defined operational semantics which opens possibilities to prove interesting properties, like termination, by induction on the length of the transformation sequences and thus increasing the probability that the algorithm is actually performing the task that it is supposed to perform.
- The graph- and rule-based representation allows to visualize and specify the ant colony optimization algorithms more naturally and can help to gain a better understanding of the algorithm. Furthermore, there exist tools that are suitable to provide an implementation platform for communities of autonomous units; in particular, in (Hölscher 2008) it is shown how an algorithm based on autonomous units can be implemented with the tool GrGEN (Geiß and Kroll 2008) for finding shortest paths in graphs.
- Communities of autonomous units are suitable to integrate autonomous control into the model of the logistic processes and algorithms (cf. (Hölscher et al. 2007), (Hölscher et al. 2008) and (Hölscher 2008)). The advantage of autonomous control in logistics is the focus of the Collaborative Research Centre 637 *Autonomous Cooperating Logistic Processes: A Paradigm Shift and Its Limitations*.

The paper is organized as follows. In Section 2, ant colony systems for the heuristic solving of optimization problems are briefly introduced. In Section 3, autonomous units and communities of autonomous units are presented. In Section 4, it is shown how ant colony systems solving the TSP can be modeled by communities of autonomous units in a formal and visual way. The conclusion is given in Section 5.

2 Ant Colony Systems

Ant colony optimization (ACO) is an algorithmic framework for the heuristic solving of combinatorial optimization problems. The idea is based on the observation how ants find short routes between food and their formicary. A moving ant leaves a chemical substance, called pheromone, on the ground which can be sensed by other ants. The higher the concentration of pheromone along a way, the more probable it is that ants will choose this way. Since on short ways ants leave their pheromone in a shorter time than on longer ways the concentration of pheromone will be higher the shorter a way is. The more ants follow a specific route, the more attractive it becomes for other ants, thus resulting in a positive feedback loop. Furthermore, pheromone evaporates with time, so a too fast convergence to a short route is avoided.

ACO imitates this behavior for solving combinatorial optimization problems which often have a complete graph as input. A problem solution is encoded as an ordered sequence of edges in the graph. The artificial ants autonomously construct solutions by exploring the graphs. Since the ants start their work at a randomly chosen node in the graph, the crucial point consists in the decision which node is to visit next.

A common method is to assign a probability *prob*$_{ij}$ to every possible decision, where i denotes the node currently visited by the ant and j is a node that can be reached from i by traversing an edge. The value of *prob*$_{ij}$ is calculated as follows:

$$prob_{ij} = \frac{p_{ij}^{\alpha} \cdot x_{ij}^{\beta}}{\sum_{k \in J^i} p_{ik}^{\alpha} \cdot x_{ik}^{\beta}} \quad \forall i \in \{1,\ldots,n\}, \quad j \in J^i .$$

The value p_{ij} simulates the pheromone intensity of the edge between the nodes i and j. Consequently, every edge of the graph has its own p-value. The value x_{ij} is a heuristic value describing an estimated probability that the solution includes the way between the nodes i and j. Like the pheromone intensity, every edge has its own x-value. The value x_{ij} is often called the *desirability* of the edge between i and j. The exponents α and β are problem-dependent parameters to control the influence of p and x. The set J^i consists of all nodes that are reachable from the node i via a single edge. In other words, *prob*$_{ij}$ is a value in the closed interval [0…1] giving a procentual estimation of how worthy it is to visit the node j being at node i.

With the help of this estimation, every artificial ant searches its way through the graph. If an ant succeeds in constructing a solution, e.g. having found a way from the start node to the goal node, it updates the pheromone value of all the edges by some value that reflects the quality of the solution. In the case of the TSP for example this value could be the reciprocal distance of the complete route. Afterwards the artificial ants will use different *prob*$_{ij}$ values according to the routes already found. Since the *prob*$_{ij}$ values of routes will be higher the shorter the route is, artificial ants will converge slowly to the shortest route found. This basic idea has been extended and modified in some ways to improve the performance. Details can be found for example in (Dorigo and Stützle 2004).

3 Communities of Autonomous Units

In this section, we briefly introduce communities of autonomous units. For more detailed introductions see (Kreowski and Kuske 2008) and (Hölscher et al. 2009).

Communities are composed of autonomous units that act and interact in a common environment which is typically a graph. The ingredients of communities are given by an underlying graph transformation approach (cf. (Rozenberg 1997) for an overview of graph transformation approaches).

Definition (Graph transformation approach) A graph transformation approach is a system $(\mathcal{G},\mathcal{R},\mathcal{X},\mathcal{C})$ where $\mathcal{G}$ is a class $\mathcal{G}$ of *graphs*, $\mathcal{R}$ is a class of *rules* such that every rule specifies a binary relation on $\mathcal{G}$, $\mathcal{X}$ is a class of *graph class expressions* each of which specifies a set of graphs in $\mathcal{G}$, and $\mathcal{C}$ is a class of control conditions each of which specifies a set of sequences of graphs.

Every autonomous unit consists of a set of pairs of rules, a set of auxiliary transformation units, a control condition, a specification of initial private states, and a goal. When a rule pair (r_1,r_2) is applied, the first component r_1 transforms the current common environment and the second component r_2 modifies the current private state of the unit. Every auxiliary transformation unit encapsulates a set of rule pairs, a set of further auxiliary transformation units, and a control condition.

Definition (Units) An *auxiliary transformation unit* is a system $tu = (P,U,C)$ where $P \subseteq \mathcal{R}\times\mathcal{R}$, U is a set of auxiliary units, and $C \in \mathcal{C}$. An *autonomous unit* is a system $aut = (I,U,P,C,G)$ where $I \in \mathcal{X}$ is the *initial private state specification*, U is a set of *auxiliary transformation units*, $P \subseteq \mathcal{R}\times\mathcal{R}$, $C \in \mathcal{C}$, and $G \in \mathcal{X}$ is the *goal*.

In the following, we use auxiliary transformation units with a hierarchical import structure. This means that every auxiliary transformation unit of import depth zero does not contain any auxiliary transformation unit, and every auxiliary transformation unit of import depth $n+1$ can use only auxiliary transformation units of import depth at most n.

Every community is composed of a set of autonomous units, a specification of a set of initial environments, a global control condition, and a goal.

Definition (Community) A *community* is a system $(Init,Aut,Cond,Goal)$ where $Init,Goal \in \mathcal{X}$ are graph class expressions called the *initial environment specification* and the *overall goal*, respectively, Aut is a set of autonomous units, and $Cond \in \mathcal{C}$ is a *global control condition*.

The semantics of communities for modeling ACO algorithms is a parallel one, i.e. in every computation step, several autonomous units may perform their actions simultaneously. More precisely, the parallel semantics of a community of autonomous units *COM* = (*Init*,*Aut*,*Cond*,*Goal*) consists of a set of state sequences that represent the transformation processes. Every state in such a sequence is composed of a common environment and a private state for every autonomous unit. The initial state of every transformation process *s* in the semantics of *COM* must be composed of a common environment specified by *Init* and an initial private state for each $aut \in AUT$. Moreover, *s* must be allowed by the *Cond* as well as by the control conditions of all units in *Aut*.

4 Modeling Ant Colony Sytems by Communities

In this section, we demonstrate how ant colony systems can be modeled with communities of autonomous units by translating an ACO algorithm for the Traveling Salesperson Problem (TSP) into a community.

The ACO algorithm for the TSP gets as input a complete graph without multiple edges in which every edge is labeled with a natural number denoting the distance between its two attached nodes, and a real number denoting an initial pheromone quantity. An optimal solution of the TSP is a Hamiltonian cycle in *G* whose distance is minimal. Basically, the TSP is solved by ACO systems according to the following procedure. First, every ant chooses nondeterministically a start node. Second, every ant traverses the graph by going along a Hamiltonian cycle. In each step it passes through exactly one edge which is chosen according to the probability rule given in Section 2. Moreover, it stores the cycle and its distance in its memory. Third, pheromone evaporation takes place and every ant traverses the cycle again while augmenting the pheromone quantity of every passed edge by $1/s$ where *s* is the distance of the cycle traversed by the ant. Then the next iteration starts.

For modeling the described algorithm as a community, we employ as underlying graph class undirected edge-labeled graphs. A *graph transformation rule r* is defined as $N \supseteq L \supseteq K \subseteq R$, where *N, L, K,* and *R* are graphs such that *L* is a subgraph of *N*, and K is a subgraph of *L* and *R*. The rule *r* is applied to a graph *G* according to the following steps (cf. also (Habel et al. 1996) and (Corradini et al. 1997)). (1) Choose a homomorphic image of *L* in *G*. (2) If *L* is a proper subset of *N*, make sure that the image of *L* cannot be extended to an image of *N*. (3) Delete the image of *L* up to the common part *K* of *L* and *R*, provided that the result is a graph again, i.e., no dangling edges should be produced. (4) Add a copy of *R* to the resulting graph such that *K* is identified with its image.

A rule $N \supseteq L \supseteq K \subseteq R$ is depicted as $N \to R$ where the parts of *N* not belonging to *L* are crossed out. The common part *K* consists of the nodes and edges that have the same forms, labels, and relative positions in *N* and *R*. A node with a loop is often depicted as a node with the loop label inside. We assume the existence of a special label *unlabelled* that is omitted in graph drawings. A pair of rules

($N_1 \supseteq L_1 \supseteq K_1 \subseteq R_1, N_2 \supseteq L_2 \supseteq K_2 \subseteq R_2$) is depicted as $N_1 \mid N_2 \to R_1 \mid R_2$, where again the items of $N_i - L_i$ (i = 1,2) are crossed out.

As graph class expressions we use *all* denoting all graphs and *complete*(A) where A is some set of labels. The expression *complete*(A) characterizes all complete graphs without multiple edges in which every edge is labeled with an element of A. Moreover, every graph G is a graph class expression which specifies itself. The control conditions of our approach consist of regular expressions over rules and auxiliary units equipped with the operator *!* standing for *as long as possible.* Moreover, as global control conditions, we use regular expressions over sets of autonomous units, where a set of units means that they should run in parallel. For reasons of space limitations we do not introduce a formal semantics of control conditions but explain their meaning when they are used.

The community of autonomous units that models the presented ACO algorithm for the TSP is equal to

$$(complete(\mathbb{N} \times \mathbb{R}), \{Ant_1,...,Ant_k, Update(\rho)\}, (\{Ant_1,...,Ant_k\}\ ;\ Update(\rho))^*, all)$$

where $k \in \mathbb{N}$, $0 < \rho \leq 1$, and the condition ($\{Ant_1,..., Ant_k\}$; $Update(\rho)$)* means that in each iteration the autonomous units $Ant_1,..., Ant_k$ run in parallel and then the autonomous unit $Update(\rho)$ becomes active, where ρ is a pheromone decay parameter.

For $j=1,...,k$, the autonomous unit Ant_j is equal to

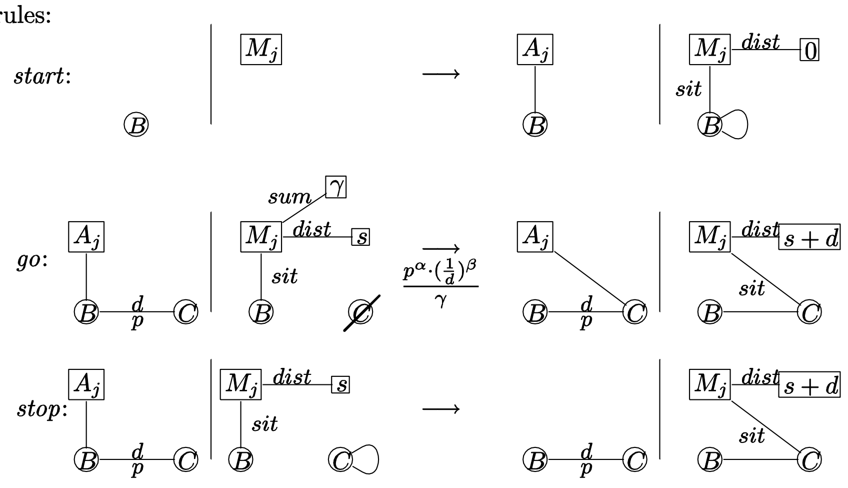
$$(M_j^{\bullet}, \{traverse_j, putpher_j\}, \emptyset, traverse_j\ ;\ putpher_j, all)$$

where $M_j^{\bullet}$ denotes the graph consisting of a node with an M_j-labeled loop (representing the initial memory of Ant_j), and the control condition requires that the auxiliary units $traverse_j$ and $putpher_j$, be executed sequentially in this order.

The unit $traverse_j$ in Figure 1 searches for a Hamiltonian cycle, guided by the present pheromone trails. It uses the auxiliary unit $prob_j$ and contains the three rule pairs *start*, *go*, and *stop*. With the first component of every rule pair, the common environment is transformed whereas the second component updates the memory of Ant_j.

The rule pair *start* puts the ant on a randomly chosen node in the common environment. In its memory, the ant inserts a node for the distance of the cycle found so far (which initially is equal to zero) and it stores its current location. The unlabeled loop at the B-node serves to remember the start node of the cycle. The rule pair *go* models a movement of the ant to a neighbor node. The crossed node guarantees that the ant only moves to a neighbor node C if the node is not stored in the memory, yet. Moreover, in the memory, the cycle distance is updated, the node C (together with the passed edge) is added to the visited path, and the current location of the ant is changed to node C by redirecting the *sit*-edge. The rule pair *go* is applied according to the probability presented in Section 2 and depicted under the arrow of the rule. The value of γ in the formula is inserted in the memory by the auxiliary unit $prob_j$, which for reasons of space limitations is not depicted. The rule pair *stop* closes the cycle. According to the control condition of $traverse_j$, the pair *start* is applied first. Second the unit $prob_j$ and *go* are applied as long as possible in this order, and finally, the pair *stop* is applied.

$traverse_j$
uses: $prob_j$
rules:



conds: $start$; $(prob_j$; $go)!$; $stop$

Fig. 1. The auxiliary unit *traverse*$_j$

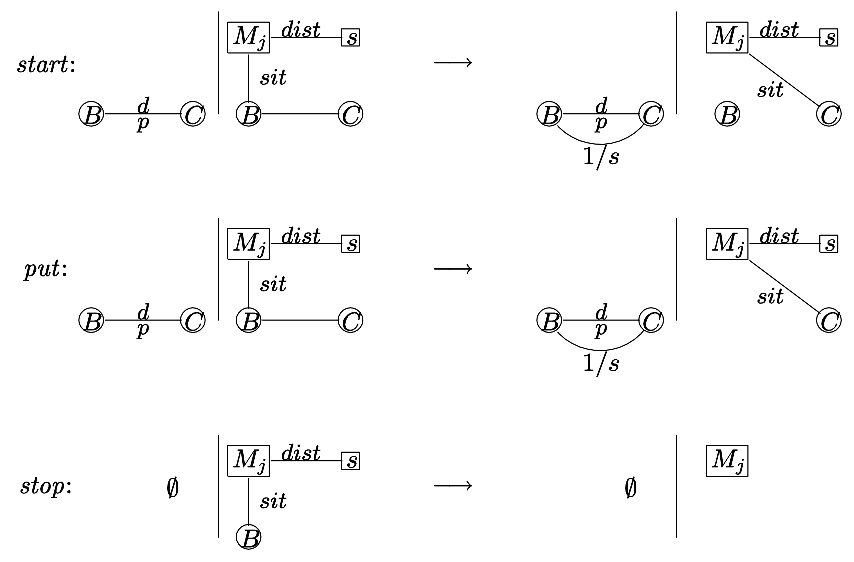
It can be shown that the unit *traverse*$_j$ finds a Hamiltonian cycle in the underlying graph.

The auxiliary unit *putpher*$_j$ in Figure 2 leaves a pheromone trail on the passed cycle with the intensity $1/s$, where *s* is the distance of the cycle. According to the cycle stored in the memory, *putpher*$_j$ places additional "pheromone-edges", labeled with $1/s$, in parallel to the edges where trail update should take place. During this procedure the cycle is deleted from the local memory. It should be noted that the fact that each ant inserts separate edges for its individual pheromone trail allows the ants to put pheromone in parallel.

It can be shown that all transformation sequences of *putpher*$_j$ are finite and that the parallel edges are inserted along the Hamiltonian cycle in the memory of the ant.

The autonomous unit *Update*(ρ) in Figure 3 is responsible for the evaporation and the summation of the pheromone updates made by the ant units. According to its control condition the rule *evap* is executed as long as possible to lessen the amount of pheromone at every edge. To remember where the evaporation has taken place, a parallel edge with label *done* is inserted.

putpher$_j$
rules:

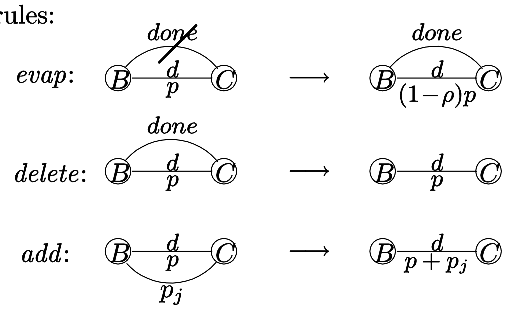


conds: *start* ; *put*! ; *stop*

Fig. 2. The auxiliary unit *putpher$_j$*

When the evaporation for all edges is completed, *delete*! removes all the *done*-edges. Now for each edge e with a label in $\mathbb{N}\times\mathbb{R}$, *add*! adds up the values of the "pheromone-edges" placed by the ant units and updates the pheromone amount of e with the respective result.

Update(ρ)
rules:



conds: *evap*! ; *delete*! ; *add*!

Fig. 3. The autonomous unit *Update*(ρ)

Since *Update*(ρ) needs no local memory, the second components of the rule pairs of *Update*(ρ) are not depicted. Formally these second components consist of empty graphs, only.

It can be shown that all transformation processes of *Update*(ρ) terminate and model the update of pheromone trails in the required way.

5 Conclusion

In this paper, we have proposed communities of autonomous units as a modeling framework for ACO algorithms. In particular, we have modeled an ant colony optimization algorithm for the Traveling Salesperson Problem as a community of autonomous units.

It has turned out that rule-based graph transformation allows to specify ant algorithms in a very natural way because (artificial) ants modify graphs and move along edges. This yields the following advantages:

- The specification of ants as autonomous units provides the ants with a well-defined operational semantics so that correctness results can be proved.
- The graph transformation rules of autonomous units allow for a visual specification of ants behavior instead of string-based pseudo code as it is often used in the literature.
- The existing graph transformation systems (cf. e.g. (Ermel et al. 1999) and (Geiß and Kroll 2008)) facilitate the visual simulation of ant colonies in a straightforward way (see also (Hölscher 2008)).
- Implementing ACO algorithms with graph transformational systems is useful for verification purposes, i.e., to check whether the algorithms behave properly for specific cases.

In the future, this and further case studies should be implemented with one of the existing graph transformation systems so that (1) the emerging behavior of ant colonies can be visually simulated by representing transformations of the common environment and transformations of the private states on different visualization levels, and (2) ACO algorithms can be verified. For the implementation purpose we plan to use GrGen (Geiß and Kroll 2008) because it is one of the fastest and most flexible graph transformation systems.

Further case studies should take into account local search, elitist and rank-based ant systems as well as dynamic aspects (cf. (Eyckelhof and Snoek 2002), (Dorigo and Stützle 2004), (Montemanni et al. 2005), (Reimann et al. 2004) and (Rizzoli et al. 2007) to mention only a few examples).

Acknowledgments We are grateful to Hans-Jörg Kreowski and the anonymous referees for their valuable comments.

References

Corradini, A., Ehrig, H., Heckel, R., Löwe, M., Montanari, U., and Rossi, F. (1997). Algebraic approaches to graph transformation part I: Basic concepts and double pushout approach. In G. Rozenberg, *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations* (pp. 163-245). Singapore: World Scientific.

Dorigo, M., and Stützle, T. (2004). *Ant Colony Optimization.* MIT-Press.

Ermel, C., Rudolf, M., and Taentzer, G. (1999). The AGG-approach: Language and environment. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 2: Applications, Languages and Tools* (pp. 551-603). Singapore: World Scientific.

Eyckelhof, C.J., and Snoek, M. (2002). Ant systems for a dynamic TSP - Ants caught in a traffic jam. In M. Dorigo, G. Di Caro , and M. Sampels (Eds.), *Ant Algorithms - Third International Workshop, ANTS 2002*, *Volume 2462 of Lecture Notes in Computer Science*, pp. 88-98.

Geiß, R., and Kroll, M. (2008). GrGen.NET: A fast, expressive, and general purpose graph rewrite tool. In A. Schürr, M. Nagl, and A. Zündorf (Eds.), *Proc. 3rd Intl. Workshop on Applications of Graph Transformation with Industrial Relevance (AGTIVE '07)*, *Volume 5088 of Lecture Notes in Computer Science*, pp. 568-569.

Habel, A., Heckel, R., and Taentzer, G. (1996). Graph grammars with negative application conditions. *Fundamenta Informaticae, 26* (3,4), pp. 287--313.

Hölscher, K. (2008). Autonomous Units as a Rule-based Concept for the Modeling of Autonomous and Cooperating Processes. Logos Verlag. PhD thesis.

Hölscher, K., Klempien-Hinrichs, R., Knirsch, P., Kreowski, H.-J., and Kuske, S. (2007). Autonomous units: Basic concepts and semantic foundation. In M. Hülsmann, and K. Windt, *Understanding Autonomous Cooperation and Control in Logistics. The Impact on Management, Information and Communication and Material Flow.* Berlin Heidelberg New York, USA: Springer.

Hölscher, K., Knirsch, P., and Luderer, M. (2008). Autonomous units for communication-based dynamic scheduling. In H.-D. Haasis, H.-J. Kreowski, and B. Scholz-Reiter, *Dynamics in Logistics, Proceedings of the First International Conference on Dynamics in Logistics (LDIC 2007)* (pp. 331-339). Springer.

Hölscher, K., Kreowski, H.-J., and Kuske, S. (2009). Autonomous Units to Model Interacting Sequential and Parallel Processes. *Fundamenta Informaticae , 92* (3), pp. 233-257.

Kreowski, H.-J., and Kuske, S. (1999). Graph transformation units with interleaving semantics. *Formal Aspects of Computing , 11* (6), pp. 690-723.

Kreowski, H.-J., & Kuske, S. (2008). Communities of autonomous units for pickup and delivery vehicle routing. In A. Schürr, M. Nagl, and A. Zündorf (Eds.), *Proc. 3rd Intl. Workshop on Applications of Graph Transformation with Industrial Relevance (AGTIVE '07)*, *Volume 5088 of Lecture Notes in Computer Science*, pp. 281-296.

Montemanni, R., Gambardella, L., Rizzoli, A., and Donati, A. (2005). Ant colony system for a dynamic vehicle routing problem. *Journal of Combinatorial Optimization, 10* (4), pp. 327-343.

Reimann, M., Doerner, K., and Hartl, R. F. (2004). D-Ants: Savings based ants divide and conquer the vehicle routing problem. *Computers and Operations Research , 31* (4), pp. 563-591.

Rizzoli, A., Montemanni, R., Lucibello, E., and Gambardella, L. (2007). Ant colony optimization for real-world vehicle routing problems. *Swarm Intelligence, 1* (2), pp. 135-151.

Rozenberg, G. (1997). Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations. Singapore: World Scientific.