Electronic Communications of the EASST
Volume 26 (2010)

Manipulation of Graphs, Algebras and Pictures

Essays Dedicated to Hans-Jörg Kreowski on the Occasion of His 60th Birthday

Autonomous Units for Solving the Capacitated Vehicle Routing Problem Based on Ant Colony Optimization

Sabine Kuske, Melanie Luderer

23 pages

 ISSN 1863-2122

Autonomous Units for Solving the Capacitated Vehicle Routing Problem Based on Ant Colony Optimization

Sabine Kuske[1], **Melanie Luderer**[2]*

[1] kuske@informatik.uni-bremen.de, http://www.informatik.uni-bremen.de/~kuske
[2] melu@informatik.uni-bremen.de, http://www.informatik.uni-bremen.de/theorie
Department of Computer Science
University of Bremen, Germany

Abstract: Communities of autonomous units and ant colony systems have fundamental features in common. Both consists of a set of autonomously acting units that transform and move around a common environment that is usually a graph. In contrast to ant colony systems, the actions of autonomous units are specified by graph transformation rules which have a precisely defined operational semantics and can be visualized in a straightforward way. In this paper, we model an ant colony system solving the capacitated vehicle routing problem as a community of autonomous units. The presented case study shows that the main characteristics of ant colony systems such as tour construction and pheromone updates can be captured in a natural way by autonomous units.

Keywords: Graph transformation, autonomous units, ant colony optimization

1 Introduction

In computer science there exists a large variety of relevant problems that are too complex to be solved by a deterministic algorithm in an acceptable time. Hence, heuristics are employed that in many cases can help to find good solutions. In this context, swarm intelligence plays an important role where, roughly speaking, a swarm is a large number of autonomous and self-interested agents that act and interact in parallel. In general, a swarm as a whole can produce good solutions for complex problems whereas a stand-alone agent is not able to do so. One well-studied kind of swarms are ant colonies which consist of a set of autonomously behaving artificial ants that move around a common graph and make their decisions according to the pheromone concentration in their neighborhood. They are inspired by the way how ants find short routes between food and their formicary and 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. [DS04]).

Basically, in an ant colony system, a set of ants constructs solutions for a given problem (mostly NP-hard) by moving along the edges of an underlying graph. According to the quality of the constructed solutions the ants walk back and put some pheromone on the traversed items, i.e.,

* The first author would like to acknowledge that her 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).

the better the solution is the more pheromone is placed by an ant. During solution construction the pheromone concentration as well as some further heuristic value help the ants to decide where to go in each step. Every ant has a memory for storing important information such as the length of the traversed path, etc.

In order to prove correctness properties of ant colony optimization algorithms, a formal modeling framework with a well defined semantics is needed. Moreover, ant colony systems can be visually represented in a straightforward way so that for simulation and verification purposes, it is desirable to have a graphical modeling framework whose operational semantics provides graphical representations of system states. Since ants work on graphs, graph transformation is a suitable approach to specify the actions of the ants. Not only has graph transformation a well defined semantics and a wide theory but there exist also some graph transformation tools that could be used to implement ACO algorithms (cf. [Roz97, EEKR99, EKMR99]). Moreover, a suitable concept for modeling the autonomous behavior of ants is needed. A promising concept to achieve this is that of communities of autonomous units because on the one hand they incorporate rule-based graph transformation and on the other hand autonomous units act and interact autonomously in a common environment (cf. [KK07, KK08, HKK09]).

Essentially, every autonomous unit is composed of a set of graph transformation rules, a control condition, and a goal. Moreover, it can ask auxiliary units for help and it can be equipped with a specification of initial private states where the latter may be used to represent the memory of an ant. Autonomous units transform the common environment and their private states simultaneously while striving for their goals, can communicate with each other via the common environment, and may act in parallel. A community is composed of a set of autonomous units, a specification of initial common environments, a global control condition, and an overall goal. A current state of a community consists of a current common environment plus a private state for every autonomous unit. The semantics of a community consists of all state sequences obtained by composing the semantics of the autonomous units in the community in such a way that the global control condition is not violated and the start state consists of an initial common environment and an initial private state for every unit. A transformation process is successful if it reaches the overall goal. The basic components of communities are provided by a graph transformation approach consisting of a class of graphs, a class of graph class expressions, a class of rules with a rule application operator, and a class of control conditions. In the literature there exists a variety of graph transformation approaches (cf. [Roz97]). They all can be used as underlying approaches for communities.

In [KLT09], it was shown that communities of autonomous units are suitable to model an ant colony solving the Traveling Salesperson Problem. The present paper focuses on a more complicated problem that can be solved in an intuitive way by ant colony optimization algorithms: the Capacitated Vehicle Routing Problem (CVRP) (cf., e.g., [RDH04, DS04]). Concretely, we present a community of autonomous units that models an ant colony that solves the CVRP. The aim of this paper is to consolidate the conjecture that communities of autonomous units are suitable as a formal framework for modeling ant colony systems.

The advantages of modeling ant colony systems as communities of autonomous units are the following. (1) Autonomous units provide ant colony systems with a well-founded operational semantics so that verification techniques for graph transformation can be applied to ant colony systems. (2) The fact that ant actions can be specified as graph transformation rules allows for a

visual modeling of ant algorithms and hence for a visual representation of ant colony behavior. (3) Existing graph transformation tools such as GrGEN [GK08] or AGG [ERT99] can be used to implement ant algorithms.

This paper is organized as follows. In Section 2, ant colony systems for the heuristic solving of optimization problems are briefly introduced and a particular ant colony optimization algorithm for solving the CVRP is recalled. Section 3 presents a graph transformation approach that is used throughout this paper. Section 4 introduces autonomous units and communities of autonomous units. Section 5 shows how fundamental features of ant colony systems can be modeled with autonomous units by translating an ant colony system solving the CVRP into a community. The conclusion is given in Section 6.

2 Ant Colony Optimization

Ant colony optimization (ACO) systems are algorithmic frameworks for the heuristic solving of optimization problems, typically problems belonging to the complexity class NP-hard, since no efficient algorithms for this kind of problems are known that always solve the problem. The idea of ACO originates in the observation of how ants find short ways between food and their formicary. An individual ant can hardly see and has a very narrow perspective of its environment. While searching for food, it leaves a chemical substance on the ground, called pheromone, which can be sensed by other ants and influence their route decision. The higher the concentration of pheromone along a way, the higher the probability that an ant will choose this way as well, thus leaving even more pheromone. The crucial point is that pheromone evaporates with time. An ant following a short route to food will return sooner to the formicary so that the pheromone concentration on shorter routes becomes more intense than on longer routes. The higher pheromone concentration makes more ants choose the short route which in turn raises the pheromone concentration further. Finally, almost all ants end up choosing one short route, although not necessarily the shortest one. Since typical optimization problems can be nicely modeled as graphs, it is the prefered data structure for ACO. In this paper we use edge-labeled undirected graphs with multiple (i.e. parallel) edges.

Graphs. A *graph* is a tuple $G = (V, E, att, m)$, where V is a finite set of *nodes*, E is a finite set of *edges* such that V and E are disjoint, $att : E \to \bigcup_{k\in\{1,2\}} \binom{V}{k}$ is a mapping that assigns to every edge a set of one or two *sources* in V, and m is a mapping that assigns a *label* to every edge in E.[1] A graph with no nodes and no edges is called the *empty graph* which is denoted by $\emptyset$. The components of G are also denoted by V_G, E_G, att_G, and m_G respectively. The set of all graphs is denoted by $\mathscr{G}$.

A solution to an optimization problem consists typically of a tour (e.g. an ordered sequence of nodes) within the given graph. Intuitively, the complexity of most NP-hard optimization problems lies in the exponentially growing number of possible tours when new nodes and edges are added. The lack of an efficient search method for the 'best' way requires an (almost) exhaustive

[1] For $k \in \mathbb{N}$, $\binom{V}{k}$ denotes the set of subsets of V with k elements, i.e., $\binom{V}{k} = \{V' \subseteq V \mid |V| = k\}$ where $|V|$ denotes the number of elements in V'.

search of all the possible tours. To solve an optimization problem with ACO, some additional information is needed. We define optimization problems as follows.

Optimization Problem. An optimization problem is a 6-tuple $(CG, d, \tau, \eta, S, g)$ where $CG \in \mathscr{G}$ is a *construction graph*, d is a function that associates every edge with a cost value (e.g. the distance), τ is a function that associates every edge with a pheromone value, η is a function that associates every edge with a number as an heuristic value for the quality of the edge, $S \subseteq V^*$ is the set of *solutions*, and g assigns a *cost* $g(s)$ to every $s \in S$.

Basically, ACO works as follows. At first, a predefined number of ants are placed randomly at some nodes. These ants decide in parallel which edge they follow in the next step according to a transition rule. Let a be an index to choose one of n ants and U_a the set of all edges that can be chosen from ant a residing at some node. The decision, which edge $e \in U_a$ to take, is probability-based. The probabilities are calculated as follows.

$$p_a(e) = \frac{[\tau(e)]^\alpha \cdot [\eta(e)]^\beta}{\sum_{e \in U_a} [\tau(e)]^\alpha \cdot [\eta(e)]^\beta} \quad \forall e \in U_a \tag{1}$$

In words this formula states that ants prefer edges with low cost and a high concentration of pheromone. The experimental parameters α and β control the influence of the pheromone resp. heuristic value in the decision. In every step this formula is applied, until all the ants have constructed a complete tour.

The next step concerns the pheromone values. Simulating the evaporation, the values of τ are reduced: $\tau(e) \leftarrow (1-\rho) \cdot \tau(e) \quad \forall e \in E_{CG}$ where ρ is a pheromone decay parameter in the interval $(0,1]$. Furthermore the release of pheromone of the ants is simulated:

$$\tau(e) \leftarrow \tau(e) + \sum_{a=1}^{n} \Delta\tau_a(e), \text{ with } \Delta\tau_a(e) = \begin{cases} \frac{1}{length(tour_a)} & , e \in tour_a \\ 0 & otherwise \end{cases}$$

where $tour_a$ is the solution constructed by ant a. In contrast to nature, the release of pheromone takes place after the ants constructed a complete tour, since the amount of pheromone corresponds to the overall quality of the tour (e.g. the length of the tour). Furthermore, in some ACO systems not every ant leaves pheromone, but just the ones having constructed the best tours.

Now the ants are placed again at some randomly chosen nodes and the algorithm starts with the modified values of pheromone. Some variants of this basic ACO yielding better performance have been proposed in the literature. Details can be found in [DS04].

2.1 Application: Capacitated Vehicle Routing Problem

An important application field of ACO concerns all kinds of tour planning with the Traveling Salesperson Problem (TSP) as the most famous one. Another problem often occurring in distribution logistics is the so called Capacitated Vehicle Routing Problem (CVRP), which can be described as follows. A number of customers must be served with some goods that are stored at a central depot. A number of vehicles with finite and equal capacity is available. The aim is to find a set of tours such that the demands of all customers are met and the total cost (the sum of

the distances of the tours) is minimized. Combinatorially, a solution can be formally described as a partition of the cities into m routes $\{R_1,\ldots,R_m\}$. Each route must satisfy the condition $\sum_{j\in R_i} dem_j \leq k$, where dem_j describes the demand of the j-th customer and k is the capacity restriction of the vehicles. Within each partition, an associated permutation function specifies the customer order.

Relaxing the conditions by allowing any partition (respectively setting $k=\infty$), the CVRP is transformed into an instance of the Multiple Traveling Salesperson Problem. Leaving the condition unchanged but with a cost function that counts the number of partitions CVRP becomes the well-known bin packing problem. CVRP contains in this sense two NP-hard problems, which in practice makes it a lot more complicated to solve than TSP for example and it seems a good idea to use ACO. A formulation of CVRP according to the definition of optimization problems is quickly found. Nevertheless, there are different ways to design the function $\eta : E_{CG} \to \mathbb{R}$. One easy possibility consists of the reciprocal cost-value of the edge.

Nevertheless, sometimes other methods are used to calculate the heuristic values; one elegant way is based on the so-called *Savings algorithm*. Starting from the initial (and unfavored) solution, where every route consists of exactly one customer, it is calculated, how the quality of the solution changes (how much one would save), putting two customers i and j in one route. Let d_{i0} denote the distance between customer i and the depot and d_{ij} the distance between customer i and j. Then the saving value obtained by merging the routes R_i and R_j together is calculated as follows:

$$\begin{aligned} s_{ij} &= 2*d_{i0}+2*d_{j0}-(d_{i0}+d_{ij}+d_{j0}) \\ &= d_{i0}+d_{j0}-d_{ij} \end{aligned}$$

Elaborated experiments concerning the performance of ACO and Saving Algorithm for the CVRP can be found in [RDH04].

3 A Graph Transformation Approach

Graph transformation approaches provide the main ingredients for communities of autonomous units. They consist of a class of graphs, a class of rules, a rule application operator, a class of control conditions, and a class of graph class expressions. The graphs are used to represent the common environments and the private states of communities. The rules are needed to transform these graphs. Moreover, control conditions restrict the non-determinism of rule application, and with graph class expressions one can specify specific graph sets such as initial environments or goals to be reached. In the literature, there exists a series of different graph transformation approaches (cf. [Roz97]).

In the following, we present a particular graph transformation approach that is suitable for modeling the CVRP based on ACO. Concretely, the graph class and the rule class together with the rule application operator are a variant of the double-pushout approach [CEH+97].

3.1 Graphs and Rules

The graph class consists of edge-labeled undirected graphs with multiple edges as presented in Section 2. For the modeling of the CVRP in Section 5 we use the following types of edge labels.

The symbol $*$ for denoting unlabeled edges; strings in $\{a,\ldots,z\}^*$ to denote site names; *cap*, *len*, *sit*, *load*, *feas*, *sum*, *depot*, *ant*, *dem*, and *depot* to denote attributes such as the capacity of the trucks, the length of a tour, etc.; labels in $\{x : y \mid x \in \{\tau, dist\}, y \in \mathbb{R}\}$ for pheromone quantities and distances between locations; labels in $\{\eta : y \mid y \in \mathbb{R} \cup \{\infty\}\}$ for the values of the function η; labels in $\mathbb{N}$ and $\mathbb{R}$ to denote demands, capacities, loads, lengths of tours, etc.; and A_j and M_j with $j \in \mathbb{N}$ to denote ants and memories.

It is worth noting that undirected graphs can be transformed into directed graphs as used in the double-pushout approach by replacing each undirected edge by a pair of directed edges pointing in opposite directions. The class of directed graphs obtained in this way is a subclass of edge-labeled directed graphs. Subgraphs and graph morphisms are defined as follows.

Subgraphs and graph morphisms. For $G, G' \in \mathscr{G}$, the graph G is a *subgraph* of G', denoted by $G \subseteq G'$, if $V_G \subseteq V_{G'}$, $E_G \subseteq E_{G'}$, $att(e) = att'(e)$, and $m(e) = m'(e)$ for all $e \in E_G$. A *graph morphism* $g\colon G \to G'$ is a pair (g_V, g_E) of mappings with $g_V : V_G \to V_{G'}$ and $g_E : E_G \to E_{G'}$ such that labels and sources are kept, i.e., for all $e \in E_G$, $g_V(att_G(e)) = att_{G'}(g_E(e))$ and $m_{G'}(g_E(e)) = m_G(e)$.[2] The image of G in G' is the subgraph $g(G)$ of G' such that $V_{g(G)} = g_V(V_G)$ and $E_{g(G)} = g_E(E_G)$. In the following, the subscripts V and E of g_V and g_E are often omitted, i.e., $g(x)$ means $g_V(x)$ for $x \in V$ and $g_E(x)$ for $x \in E$.

Graphs are depicted as usual with round or boxed nodes and lines as edges. A loop is sometimes omitted by putting its label inside the node to which the loop is attached. Since a node can have several loops this is always done for at most one loop per node. A node with a label x inside will also be called an x-node. The label * is omitted in graph drawings.

Graphs can be modified by rules consisting of a negative context, a left-hand side, a gluing graph, and a right-hand side. Roughly speaking, the negative context specifies components that must not occur in the graph to which the rule is applied. The left-hand side, the gluing graph, and the right-hand side are used to determine which components should be deleted, kept and added, respectively. In every computation step of a community, the autonomous units transform the common environment and their private states simultaneously. For this purpose, every unit applies pairs of rules (r_1, r_2), where the first rule r_1 is applied to the common environment and r_2 to the private state.

Rules and rule pairs. A *rule* r is a quadruple (N, L, K, R) of graphs with $N \supseteq L \supseteq K \subseteq R$ where N is the *negative context*, L is the *left-hand side*, K is the *gluing graph*, and R is the *right-hand side*. If all components of r are empty, r is the *empty rule*. The set of all rules is denoted by $\mathscr{R}$. A *rule pair* is a pair of rules $r = (r_1, r_2)$ where r_1 is called the *global rule* and r_2 the *private rule*. The set of all rule pairs is denoted by $\widetilde{\mathscr{R}}$.

A rule pair $r = (r_1, r_2)$ where r_2 is the empty rule can be regarded as a single rule. Hence, in the following, we often do not distinguish between single rules and rule pairs with an empty private rule.

[2] For a mapping $f\colon A \to B$ and $C \subseteq A$ the set $f(C)$ is defined as $\{f(x) \mid x \in C\}$, i.e., $g_V(att_G(e)) = \{g_V(v) \mid v \in att_G(e)\}$.

A rule (N,L,K,R) is depicted as $N \to R$ where the nodes and edges of K have the same forms, labels, and relative positions in N and R. The forbidden nodes (i.e., the nodes of N that do not belong to L) are colored gray. The forbidden edges are dashed. Figure 1 shows a rule where the left-hand side consists of a round node, a rectangle a-node and an edge connecting both. The gluing graph consists of the round node, and the right-hand side is obtained from the gluing graph by connecting the round node with a new b-node. The gray rectangle node as well as its incident edges are forbidden.

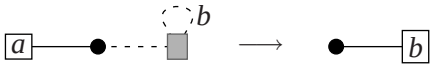


Figure 1: A rule

A rule pair $r = ((N_1,L_1,K_1,R_1),(N_2,L_2,K_2,R_2))$ (with non-empty private rule) is depicted as $L_1|L_2 \to R_1|R_2$ where the negative contexts and the gluing graphs are represented as in single rules.

A rule (N,L,K,R) is applied to a graph as follows. (1) Choose an image $g(L)$ of L in G. (2) Check whether $g(L)$ has no forbidden context given by N up to L. (3) Delete $g(L)$ up to $g(K)$ from G provided that no dangling edges are produced. (4) Glue R and the remaining graph in K. This means that the subgraph K of R is identified with its image in Z. This construction can be defined as follows.

Gluing of graphs. Let $K \subseteq R$ and $h\colon K \to Z$. Then the *gluing of Z and R in K with respect to h* is constructed as follows. Let $\approx_V$ be the equivalence relation generated on $V_Z + V_R$ by the relation $\{(h_V(v),v) \mid v \in V_K\}$ and let $\approx_E$ be the equivalence relation on $E_Z \times E_R$ generated by $\{(h_E(e),e) \mid e \in E_K\}$.[3] Let $(V_Z+V_R)/\approx_V$ and $(E_Z+E_R)/\approx_E$ be the respective quotient sets. Then the gluing of Z and R in K with respect to h yields the graph

$$D = ((V_Z+V_R)/\approx_V, (E_Z+E_R)/\approx_E, att, m)$$

where for all $e \in (E_R+E_Z)/\approx_E$

$$att(e) = \begin{cases} [att_Z(\overline{e})] \text{ if } e = [\overline{e}] \text{ for some } \overline{e} \in E_Z \text{ [4]} \\ [att_R(\overline{e})] \text{ if } e = [\overline{e}] \text{ for some } \overline{e} \in E_R - E_K \end{cases}$$

$$m(e) = \begin{cases} m_Z(\overline{e}) \text{ if } e = [\overline{e}] \text{ for some } \overline{e} \in E_Z \\ m_R(\overline{e}) \text{ if } e = [\overline{e}] \text{ for some } \overline{e} \in E_R - E_K \end{cases}$$

The application of a rule to a graph is formally defined as follows.

[3] + denotes the disjoint union of sets

[4] For a quotient set $A/\approx$, $[]\colon A \to A/\approx$ denotes its natural associated function.

Rule application. Let $r = (N, L, K, R) \in \mathscr{R}$, let $G \in \mathscr{G}$. Then r is applied to G by performing the following steps. (1) Choose an injective graph morphism $g\colon L \to G$ such that the following conditions are satisfied. (a) If $L \subset N$, there exists no $g'\colon N \to G$ with $g'(x) = g(x)$ for all $x \in V_L \cup E_L$. (b) For all $e \in E_G - E_{g(L)}$, $att_G(e) \subseteq V_G - (V_{g(L)} - V_{g(K)})$. (2) Construct the intermediate graph Z by deleting $V_{g(L)} - V_{g(K)}$ and $E_{g(L)} - E_{g(K)}$ from G, (3) construct the gluing of Z and R in K with respect to $g|K\colon K \to Z$ where $g|K(x) = g(x)$ for all $x \in V_K \cup E_K$.

The *semantic relation of* r is denoted by $SEM(r)$ and consists of all pairs (G, G') such that G' can be derived from G via the application of r. For a set $P \subseteq \mathscr{R}$, we define $SEM(P) = \bigcup_{r \in P} SEM(r)$. For $(r_1, r_2) \in \widetilde{\mathscr{R}}$, the semantic relation is equal to $\{((G_1, G_2), (G_1', G_2')) \mid (G_i, G_i') \in SEM(r_i), i = 1, 2\}$, i.e., $SEM(r_1, r_2)$ consists of all pairs $((G_1, G_2), (G_1', G_2'))$ where for $i = 1, 2$ the graph G_i' can be obtained by applying r_i to G_i.

The rule in Figure 1 can be applied to a graph containing a node v connected to an a-node but not connected to a b-node. Its application removes the a-node plus the edge to v and adds a b-node and an edge from this b-node to v. Because of condition (b) of the preceding definition, the a-node is only connected to v but not to other nodes; otherwise its deletion would produce dangling edges.

The described kind of applying graph transformation rules is a variant of the double-pushout approach presented in e.g. [CEH+97], where also non-injective matchings of the left-hand side are allowed and graphs are directed and node- and edge-labeled. Replacing all undirected edges by directed ones as described above, the application of a rule as presented here is performed in the same way as in the double-pushout approach restricted to injective matchings and edge-labeled graphs. A node with a single x-loop could be also modeled as a node with node label x in the case where not only edge labels but also node labels are allowed. However, in the double-pushout approach, relabeling of nodes via a graph transformation rule is often not possible because this may violate condition (b) in the second step of rule application. For this reason we use edge-labeled graphs where this problem does not occur. An approach that includes node relabeling explicitly can be found in [HP02].

In general, the autonomous units of a community apply their rules in parallel. A parallel rule application step involving two rules can be defined as follows.

Parallel rule application. Let $G \in \mathscr{G}$ and for $i = 1, 2$, let $r_i = (N_i, L_i, K_i, R_i)$ be two rules. Let $g_i\colon L_i \to G$ be two injective graph morphisms that satisfy the conditions (a) and (b) of the definition of rule application and the *independence condition* $g_1(L_1) \cap g_2(L_2) \subseteq g_1(K_1) \cap g_2(K_2)$.[5] Then r_1 and r_2 can be applied in parallel to G by (1) deleting $V_{g_i(L)} - V_{g_i(K)}$ and $E_{g_i(L)} - E_{g_i(K)}$ (for $i = 1, 2$), and (2) constructing the gluing of the resulting graph D and $R_1 + R_2$ in $K_1 + K_2$ with respect to $g\colon K_1 + K_2 \to D$, where $g(x) = g_i(x)$ if $x \in V_{K_i} \cup E_{K_i}$, for $i = 1, 2$.[6]

The definition of parallel rule application can be extended in a straightforward way from two rules to arbitrary non-empty multisets of rules. For a multiset m of rules, $SEM(m)$ denotes the set of all $(G, G') \in \mathscr{G} \times \mathscr{G}$ where G' is derived from G via the parallel application of the rules

[5] For $G_1, G_2 \in \mathscr{G}$ the intersection $G_1 \cap G_2$ yields the pair (V, E) where $V = V_{G_1} \cap V_{G_2}$ and $E = E_{G_1} \cap E_{G_2}$. Moreover, we have $(V_1, E_1) \subseteq (V_2, E_2)$ if $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$.

[6] The morphism g may be non-injective.

in m. A multiset m of rules will be called a *parallel rule*, and for a set $P \subseteq \mathscr{R}$, the set of all parallel rules over P is denoted by P_*. For a rule pair $r = (r_1, r_2)$, $SEM(r||m)$ denotes all $((G_1, G_2), (G_1', G_2')) \in (\mathscr{G} \times \mathscr{G}) \times (\mathscr{G} \times \mathscr{G})$ where G_1' is derived from G_1 by applying the multiset obtained from adding r_1 to m, and $(G_2, G_2') \in SEM(r_2)$.

3.2 Control Conditions

It is often desirable to restrict the non-determinism of rule application. This can be achieved with control conditions. Concretely, we use as control conditions regular expressions equipped with *as long as possible*.

Control conditions. Let ID be a set such that $P \subseteq ID$ for some set P of rule pairs. Then the class $\mathscr{C}(ID)$ of *control conditions* over ID is inductively defined as follows: $\{lambda\} \cup ID \cup \{x! \mid x \in P\} \subseteq \mathscr{C}(ID)$. For $c, c_1, c_2 \in \mathscr{C}(ID)$, we have $(c_1 + c_2), (c_1 ; c_2), (c^*) \in \mathscr{C}(ID)$.

For practical applications, the set ID would consist of names referring to rule pairs (or units) but for technical simplicity we do not distinguish between rule pairs (units) and their names.

If ID consists only of rule pairs, a semantics of control conditions can be defined in an intuitive way. Roughly speaking, the condition $lambda$ applies no rule. Every rule pair r is a control condition that prescribes one application of r. The condition $c_1 + c_2$ stands for applying c_1 or c_2, $c_1 ; c_2$ means that c_1 must be applied before c_2, c^* applies c arbitrarily often, and $r!$ requires that the pair r be applied as long as possible. The operator ! applies only to rules because the possibility to iterate other control conditions as long as possible is not needed in the following.

The semantics of control conditions are sequences of graph pairs where every pair consists of a common environment and a private state of the unit the control condition is part of. Each pair in the sequence is obtained from the previous pair by one of the following actions: (1) An application of a rule pair occurring in the control condition; (2) an application of a parallel rule to the common environment where the parallel rule is composed of global rules of other autonomous units in the community; (3) a parallel composition of (1) and (2). This means in particular that the semantics of control conditions is defined w.r.t. a set of active rules that comprises the global rules of all other units in the community.

Semantics of control conditions. Let $\mathscr{AR} \subseteq \mathscr{R}$ be a set of rules called *active rules* and let $P \subseteq \widetilde{\mathscr{R}}$. Then for each control condition in $\mathscr{C}(P)$ its *semantics* is defined as follows.

1. $SEM_{\mathscr{AR}}(lambda)$ consists of all sequences $(G_0, \ldots, G_n)$ of graph pairs such that for $i = 1, \ldots, n$, $(G_{i-1}, G_i) \in SEM(m)$ for some $m \in \mathscr{AR}_*$.[7]

2. $SEM_{\mathscr{AR}}(r)$ consists of all sequences $s = (G_0, \ldots, G_n)$ for which there exist some $j \in \{1, \ldots, n\}$ and $m_1, \ldots, m_n \in \mathscr{AR}_*$ such that for $i = 1, \ldots, j-1$ and $i = j+1, \ldots, n$, the pair (G_{i-1}, G_i) is in $SEM(m_i)$, and $(G_{j-1}, G_j) \in SEM(r||m_j)$.

3. $SEM_{\mathscr{AR}}(c_1 + c_2) = SEM_{\mathscr{AR}}(c_1) \cup SEM_{\mathscr{AR}}(c_2)$.

[7] In this transformation, the second component of every graph pair remains unchanged, because m is a multiset of single rules.

4. $SEM_{\mathscr{AR}}(c_1;c_2) = SEM_{\mathscr{AR}}(c_1) \circ SEM_{\mathscr{AR}}(c_2)$.[8]
5. $SEM_{\mathscr{AR}}(c^*) = SEM_{\mathscr{AR}}(c)^*$.
6. $SEM_{\mathscr{AR}}(r!)$ consists of all sequences $(G_0,\ldots,G_n) \in SEM_{\mathscr{AR}}(r^*)$ such that r is not applicable to G_n.

In Section 4 we show how this definition can be employed for the more general case where *ID* contains units, too.

3.3 Graph Class Expressions

In order to use graph transformation in a meaningful way, it should be possible to specify initial and terminal graphs of graph transformation processes with graph class expressions. In general, a graph class expression can be any expression that specifies a set of graphs. In particular, the graph class expressions used in this paper are the following.

Graph class expressions. The class $\mathscr{X}$ of graph class expressions is recursively defined as follows: $all, empty, red(P) \in \mathscr{X}$ with $P \subseteq \mathscr{R}$ where $SEM(all) = \mathscr{G}$, $SEM(empty) = \emptyset$, and $SEM(red(P))$ consists of all graphs G to which no rule of P can be applied. Moreover, for $I,T \in \mathscr{X}$, $P \subseteq \mathscr{R}$, and $C \in \mathscr{C}(P)$, $(I,P,C,T) \in \mathscr{X}$ where $SEM(I,P,C,T)$ consists of all graphs $G \in SEM(T)$ for which there is a sequence $(G_0,\ldots,G_n)$ such that $G_n = G$, $G_0 \in SEM(I)$, for $i = 1,\ldots,n$ $(G_{i-1},G_i) \in SEM(P)$, and $(G_0,\ldots,G_n) \in SEM_{\emptyset}(C)$.[9]

One example of a graph class expression of the last type is

$$complete = (empty, \{nodes, edges\}, nodes^*; edges^*, red(\{edges\})),$$

where *nodes* and *edges* are the rules in Figure 2.

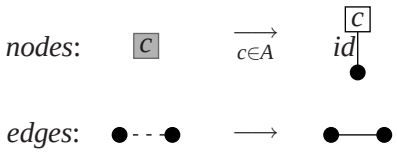


Figure 2: The rules *nodes* and *edges*

The left-hand side and the gluing graph of the rule *nodes* are empty. The negative context consists of a boxed node with an incident *c*-loop, and the right-hand side is composed of a round node, a boxed *c*-node, and an *id*-edge connecting both. The application of *nodes* to a graph G inserts a new round node connected to a new boxed *c*-node via an *id*-edge provided that there

[8] For sets of sequences S,S' of graph pairs, their sequential composition is denoted by $S \circ S'$, and S^* is defined as $\bigcup_{i\in\mathbb{N}} S^i$ with $S^0 = \mathscr{G} \times \mathscr{G}$ and $S^{i+1} = S^i \circ S$.

[9] Control conditions can be used to define sequences of graphs (instead of sequences of graph pairs) because, as stated before, rules can be regarded as rule pairs with empty private component.

is no boxed c-node in G. The rule *edges* connects two existing round nodes by an unlabeled edge. Given some alphabet A, the expression *complete* specifies all complete graphs composed of round nodes in which each round node is associated with a different element from A via an *id*-edge. (The *id*-edge can be considered as an attribute of round nodes which has type A.) It is worth noting that the rule *edges* cannot produce loops because we only use injective morphisms to choose a match of the left-hand side. In addition, we technically distinguish between round and boxed nodes by using particularly labeled loops that indicate the respective node type (*round* or *boxed*).

4 Communities of Autonomous Units

Every community is mainly composed of a set of autonomous units that act and interact in a common environment (see e.g. [HKK09] where a sequential and a parallel semantics of communities is introduced).

4.1 Autonomous Units

Autonomous units transform a common graph and have an additional private graph where they can store private information. Since the rule set of an autonomous unit can be very large, structuring concepts should be provided to keep it manageable. Autonomous units allow to import auxiliary units and provide control conditions as well as graph class expressions. Auxiliary units differ from autonomous units in the sense that they do not contain graph class expressions. The graph class expressions of every autonomous unit are used to specify the initial private states as well as the goal. The latter consists of a private goal concerning the private state and a goal concerning the common environment that the autonomous unit wants to reach.

Autonomous units. A *unit* of *import depth* 0 is a system $unit = (I,U,P,C,g)$ where $I \in \mathscr{X}$ is the *initial private graph class expression*, $U = \emptyset$, $P \subseteq \widetilde{\mathscr{R}}$, $C \in \mathscr{C}(P \cup U)$, and $g \in \mathscr{X} \times \mathscr{X}$ is the *goal*. A *unit* of *import depth* $n+1$ is a system $unit = (I,U,P,C,g)$ where U is a set of units of import depth at most n, and I, P, C, and g are defined as above. A unit (I,U,P,C,g) is an *auxiliary unit* if $I = all$, $g = (all,all)$, and every $u \in U$ is an auxiliary unit. A unit (I,U,P,C,g) is an *autonomous unit* if every $u \in U$ is an auxiliary unit. The set of autonomous units is denoted by AUT. The components of *unit* are also denoted by I_{unit}, U_{unit}, P_{unit}, C_{unit}, and g_{unit}, respectively.

Every autonomous unit can be converted into a flattened unit with import depth zero. The rule set and the control condition of the flattened unit can be constructed as follows.

Flattening. For $unit = (I,U,P,C,g)$ its *flattened rule set* $Rules(unit)$ and its *flattened control condition* $flC(unit)$ is defined as follows. If $U = \emptyset$, $Rules(unit) = P$ and $flC(unit) = C$. If $U \neq \emptyset$, $Rules(unit) = P \cup \bigcup_{u \in U} Rules(u)$ and $flC(unit) = C[a]$ where $a\colon U \to \mathscr{C}(\widetilde{\mathscr{R}})$ is defined as $a(u) = flC(u)$ and $C[a]$ is obtained by replacing every occurrence of u with $a(u)$ (for each $u \in U$).

The parallel semantics of autonomous units consists of all sequences of graph pairs $s = ((G_0,G'_0),\ldots,(G_n,G'_n))$ such that G'_0 is an initial private graph and s is allowed by the flattened

control condition with respect to some underlying set of active rules. Moreover, s is successful if the last graph pair in s satisfies the goal of the unit.

A community consists of a set of autonomous units, a specification of all possible initial environments, a global control condition, and an overall goal. In the following, global control conditions are regular expressions equipped with the parallel operator $||$.

Global control conditions. Let $Aut \subseteq AUT$. Then the set of *global control conditions* $\mathscr{GC}(Aut)$ is recursively defined as follows: $\{aut_0||\cdots||aut_k \mid aut_i \in Aut, i = 0,\ldots,k\} \subseteq \mathscr{GC}(Aut)$. For $c, c_1, c_2 \in \mathscr{GC}(Aut)$, we have $(c_1 + c_2), (c_1; c_2), (c^*) \in \mathscr{GC}(Aut)$.

Global control conditions specify sequences of states where every state consists of a common environment plus a private state for every autonomous unit in a community. The global control condition $aut_0||\cdots||aut_k$ prescribes the parallel running of $aut_0,\ldots,aut_k$. The semantics of the remaining control conditions are defined as expected. In the following we define states and the semantics of global control conditions.

Semantics of global control conditions. For $Aut \subseteq AUT$, a *state* is a pair (G, map) where $G \in \mathscr{G}$ and $map\colon Aut \to \mathscr{G}$ is a mapping. The *semantics* of each global control condition in $\mathscr{GC}(Aut)$ is defined as follows.

1. $SEM_{Aut}(aut_0||\cdots||aut_k)$ consists of all sequences $((G_0, map_0),\ldots,(G_n, map_n))$ such that for $i = 0,\ldots,k$, $((G_0, map_0(aut_i)),\ldots,(G_n, map_n(aut_i))) \in SEM_{\mathscr{AR}(aut_i)}(flC(aut_i))$, where $\mathscr{AR}(aut_i) = \bigcup_{aut \in \{aut_0,\ldots,aut_k\} - \{aut_i\}} Rules(aut)$, and for each $aut \in Aut - \{aut_0,\ldots,aut_k\}$, $map_0(aut) = \cdots = map_n(aut)$.
2. $SEM_{Aut}(c_1 + c_2) = SEM_{Aut}(c_1) \cup SEM_{Aut}(c_2)$,
3. $SEM_{Aut}(c_1; c_2) = SEM_{Aut}(c_1) \circ SEM_{Aut}(c_2)$, and
4. $SEM_{Aut}(c^*) = SEM_{Aut}(c)^*$.

The components of communities are given as follows.

Community. A *community* is a tuple $(Init, Aut, Cond, Goal)$ where $Init, Goal \in \mathscr{X}$, $Aut \subseteq AUT$, and $Cond \in \mathscr{GC}(Aut)$.

The parallel semantics of a community consists of all state sequences that are allowed by the global control condition and start with an initial state consisting of an initial common environment and an initial private state for each autonomous unit. The state sequences are successful if they reach the overall goal.

Parallel community semantics. Let $COM = (Init, Aut, Cond, Goal)$ be a community. Then the *parallel community semantics* of COM, denoted by $PAR(COM)$ consists of all state sequences $s = ((G_0, map_0),\ldots,(G_n, map_n))$ such that $G_0 \in SEM(Init)$, $map_0(aut) \in SEM(I_{aut})$ (for each $aut \in Aut$), and $s \in SEM_{Aut}(Cond)$. Moreover, s is *successful* if $G_n \in SEM(Goal)$.

5 An ACO Community for Solving the CVRP

In this section we present the components of the ACO community COM_{CVRP} for modeling the Capacitated Vehicle Routing Problem (CVRP) introduced in Section 2. The initial environment specification of COM_{CVRP} specifies the construction graph of the problem; the set of autonomous units consists of the autonomous units $Ant_1, \ldots, Ant_k$ ($k \in \mathbb{N}$), and *Evap&Select*; and the global control condition *Cond* is equal to $(Ant_1 || \ldots || Ant_k || Evap\&Select)^*$. In our first approach the overall goal is equal to *all*.

Roughly speaking, the community COM_{CVRP} works as follows. The ant units $Ant_1, \ldots, Ant_k$ model the ants, which in parallel traverse the graph according to the savings heuristics introduced in Section 2 and the current pheromone trails, and search for a solution for the CVRP. When all ants have finished their search, the autonomous unit *Evap&Select* first carries out evaporation of the current pheromone trails. After that it selects w ants with best solutions. Now each selected ant leaves a pheromone trail on its solution path according to the quality of the solution. All the units act in parallel. To ensure the described order we use negative contexts as well as control conditions.

5.1 The Initial Environment

The underlying structure of the construction graph of the ACO system modeling the CVRP is a complete graph with some additional information such as initial pheromone concentration, distances, etc. This construction graph can be defined by the graph class expression depicted in Figure 3. It uses as initial expression the graph class expression *complete* introduced in Subsection 3.3. Its rule *depot* selects the depot and has to be applied exactly once. The rule *cust* adds a number representing the demand to every customer node, i.e., to every node apart from the depot. The rule *init* labels every edge e of the initial graph with a *distance* d and it inserts two edges between each two nodes of the graph, one labeled with the *heuristic value* ∞ the other with an *initial pheromone value* z. The rule *save* computes the heuristic value of every edge based on the savings heuristics. The control condition requires that the depot is selected first. The terminal graph class expression $red(\{init, save, cust\})$ guarantees that the rules *cust*, *init*, and *save* are applied as long as possible.

The rules *cust*, *init*, and *save* of *Construction_graph* are parameterized, i.e., their labels contain variables. Each of these parameterized rules represents an infinite set of rules: one for each possible instantiation of its variables. Concretely, the variable x can be instantiated with a natural number, and d, d_1, and d_2 with non-negative real numbers. (The value z is fixed and represents the initial pheromone value.) Hence, when applying a parameterized rule, a value for each of its variables must be chosen. More information and particular aspects concerning parameterized rules and their application can be found in e.g. [EEPT06, PS04, Kus02].

The meaning of the graph class expression *Construction_graph* is to formally specify the class of initial environments consisting of all terminal graphs that can be generated from a complete graph by the rules such that the control condition is satisfied. In practice, the community COM_{CVRP} would rather start its work on already existing initial construction graphs instead of generating them nondeterministically.

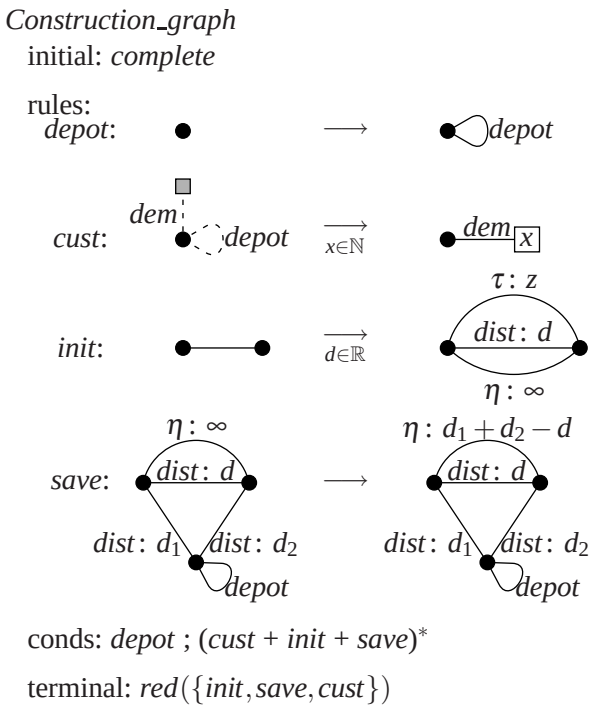


Figure 3: The graph class expression *Construction_graph*

5.2 The Ant Units

In general, every ant builds a solution tour by traversing the common environment according to the current pheromone trails. It first selects its initial position. Afterwards, it constructs a solution tour t. Then it puts some pheromone on t if it is selected to do so. Every ant unit Ant_j uses the auxiliary units $tour_j$, and put_phero_j. The control condition is equal to

$$initial\_position_j \ ; \ tour_j \ ; \ put\_phero_j$$

where $initial\_position_j$ is the rule pair depicted in Figure 4.

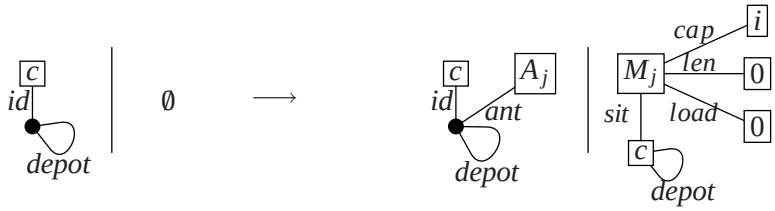


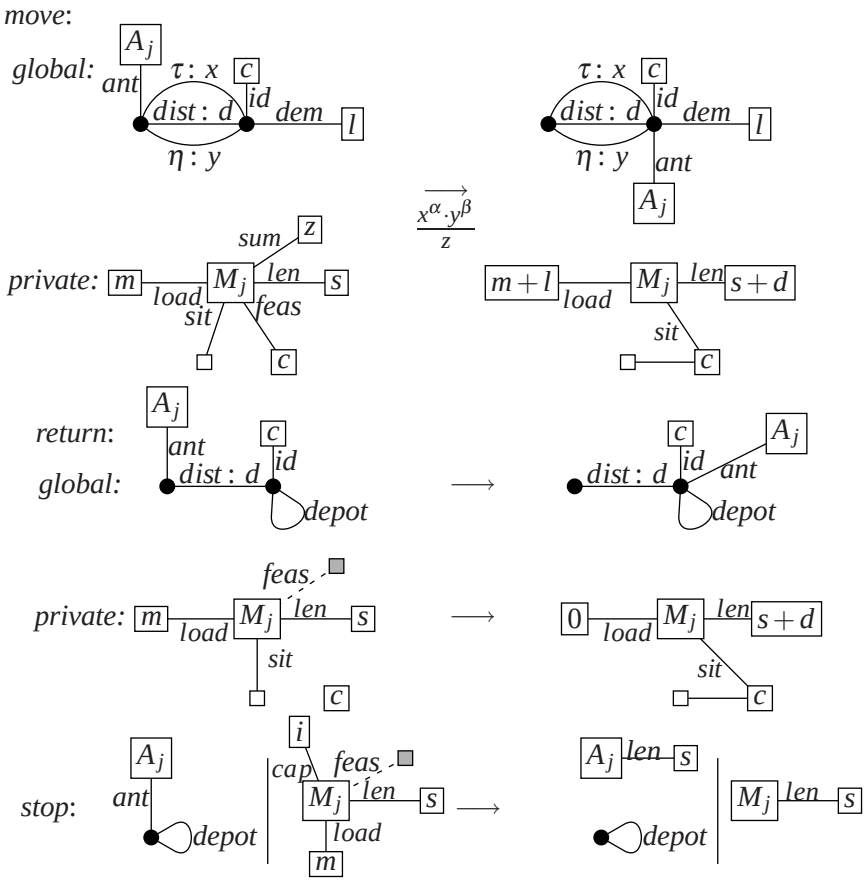
Figure 4: The rule $initial\_position_j$

It puts the ant Ant_j to the depot and generates its memory M_j where it stores the current load

$tour_j$

uses: $feasible\_neighbors_j$, $prob_j$

rules:



conds: $(feasible\_neighbors_j\ ;\ (prob_j\ ;\ move + return))^*\ ;\ feasible\_neighbors_j\ ;\ stop$

Figure 5: The auxiliary unit $tour_j$

of the vehicle represented by Ant_j (*load*), the capacity of the vehicle (*cap*), its current location (*sit*) and the total length of the tours (*len*). This information is represented by edges labeled with the respective labels (*load*, *cap*, *sit* and *len*), which are each attached to a node with the corresponding value inside.

The unit $tour_j$ is given in Figure 5. The global and private parts of the unit's rule pairs are depicted one below the other. With $tour_j$ the ant builds a solution tour depending on probabilities for the next move to a feasible neighbor calculated from the savings heuristics and the current pheromone trails. It contains the auxiliary units $feasible\_neighbors_j$ and $prob_j$, and the rule pairs *move*, *return* and *stop*. The control condition requires to apply the rule pairs *move* or *return* arbitrarily often, and afterwards, the rule pair *stop* is applied once. Before each application of

move the unit $prob_j$ is called. Moreover, the unit $feasible\_neighbors_j$ is executed before each application of $prob_j$ and of *return* as well as before the application of the last rule pair *stop*.

The unit $feasible\_neighbors_j$ is given in Figure 6. It computes the feasible neighbors for an ant unit Ant_j and stores them in the memory of the ant. Feasible neighbors are customer-nodes that are not yet visited and whose demand still fits into the vehicle. Every application of the only rule pair *feas* adds one feasible neighbor to the memory. Moreover, it uses the auxiliary unit *delete_nonfeasible* that removes all neighbors from the memory that are connected via a *feas*-edge to M_j and whose demand exceeds the remaining capacity of the vehicle. (We assume that the demand of each customer fits into one vehicle.) This is necessary because after adding a feasible customer to a tour, the former feasible neighbors may not fit into the vehicle anymore. For reasons of space limitations a drawing of *delete_nonfeasible* is omitted.

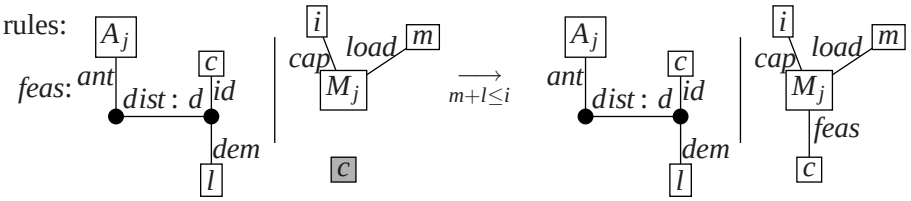


Figure 6: The auxiliary unit $feasible\_neighbors_j$

The unit $prob_j$ is given in Figure 7. It provides the denominator of the probability that a feasible neighbor is chosen for a next move (see Equation 1 of Section 2). The rule *begin* initializes this value with 0. The rule pair *sum* must be applied as long as possible. For not counting a feasible neighbor several times *sum* changes each label *feas* into *ok*. At the end the unit *relabel_all_private*$_j$*(ok,feas)* is applied which undoes this relabeling, i.e., it changes all *ok*-edges into *feas*-edges. It is very simple and hence not depicted.

With the rule pair *move* of the unit $tour_j$ the ant moves to a feasible neighbor with the probability depicted under the arrow of the rule pair *move* in Figure 5. Moreover, in the memory the current load of the vehicle, the path followed so far, and the total length of the tour are updated. With the rule pair *return* the ant returns to the depot if no feasible neighbor is left and resets its current load to 0. Afterwards it starts to construct a new subtour. Finally, when all nodes are visited, the rule pair *stop* is applied to delete the load and the capacity from the memory as well as the edge between the ant and the depot in the common environment because none of them are needed for the pheromone update which is the next and last step of one run of Ant_j. Moreover, the rule pair *stop* communicates the information about the length of the found solution via the common environment by inserting an edge labeled with *len* from the ant-node A_j to a new node labeled with the length of the solution.

The unit put_phero_j is depicted in Figure 8. It works a little different for ants, who should leave a pheromone trail and those who should not. Both kinds of ants apply different rules,

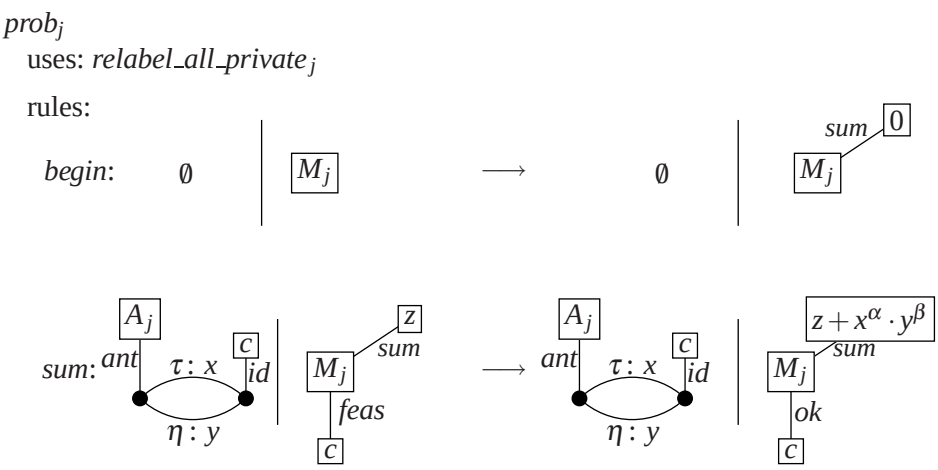


Figure 7: The auxiliary unit $prob_j$

but the structure of rule applications is the same. In both cases the ant traverses the solution path stored in its memory and meanwhile deletes it. (Because the path stored in the memory is shaped like a blossom with the depot in the middle, first the ”petals” (subtours) are deleted and finally the depot.) This behavior is represented by the rule pairs *start_a* (resp. *start_b*) and *put* (resp. *delete_only*) and the subexpression of the control condition ((*start_a* + *start_b*) ; (*put*! + *delete_only*!))*. One application of *start* followed by applications of *put* (resp. *delete_only*) as long as possible traverses one subtour of the found tour beginning and ending at the depot. The rule pairs delete the traversed path from the memory (leaving the depot); *put* additionally leaves a pheromone trail in the common environment with the value $1/s$, where s is the length of the solution tour. Afterwards the remaining subtours are traversed until no further subtour is left in the memory. Then the respective *stop*-rules can be applied, which deletes the ant A_j from the common environment as well as its complete memory. Please note that due to the independence condition for parallel rule application the rules *start_a* and *put* can only be applied in parallel by different ants to different pheromone edges so that several pheromone updates of the same edge are always executed sequentially.

5.3 The Unit *Evap&Select*

Evap&Select is given in Figure 9. It is responsible for the evaporation of pheromone trails, for the selection of the w best solutions provided by the ants, and for marking these w ants with a *put_phero*-loop.

With the rule *check*, which is applied only once at the beginning, the unit checks whether all ants have finished their search. This is the case if all ants have written the length of the found solution into the common environment. With the help of the unit *relabel_all_global* evaporation takes place by multiplying the pheromone value of every pheromone edge in the common en-

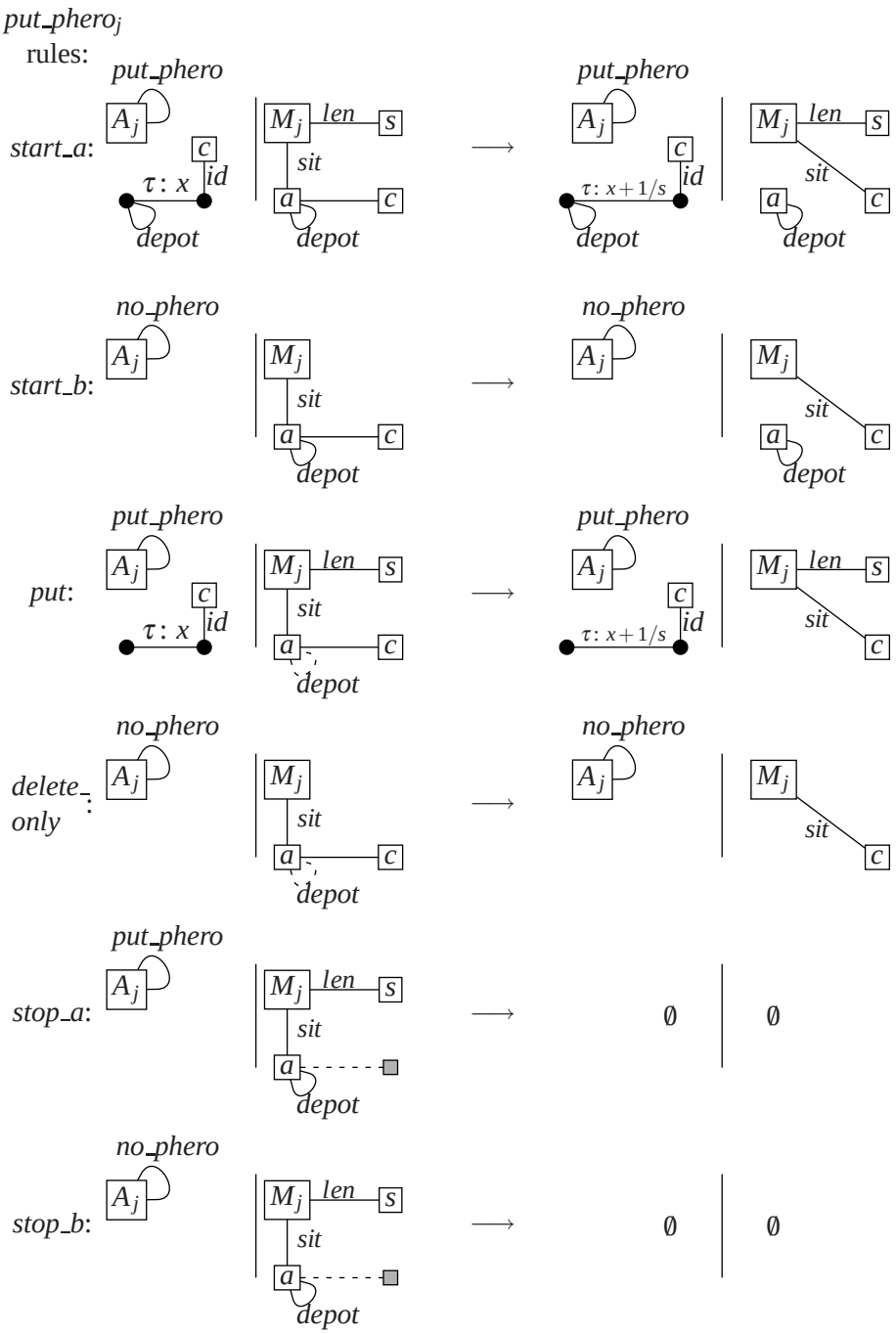
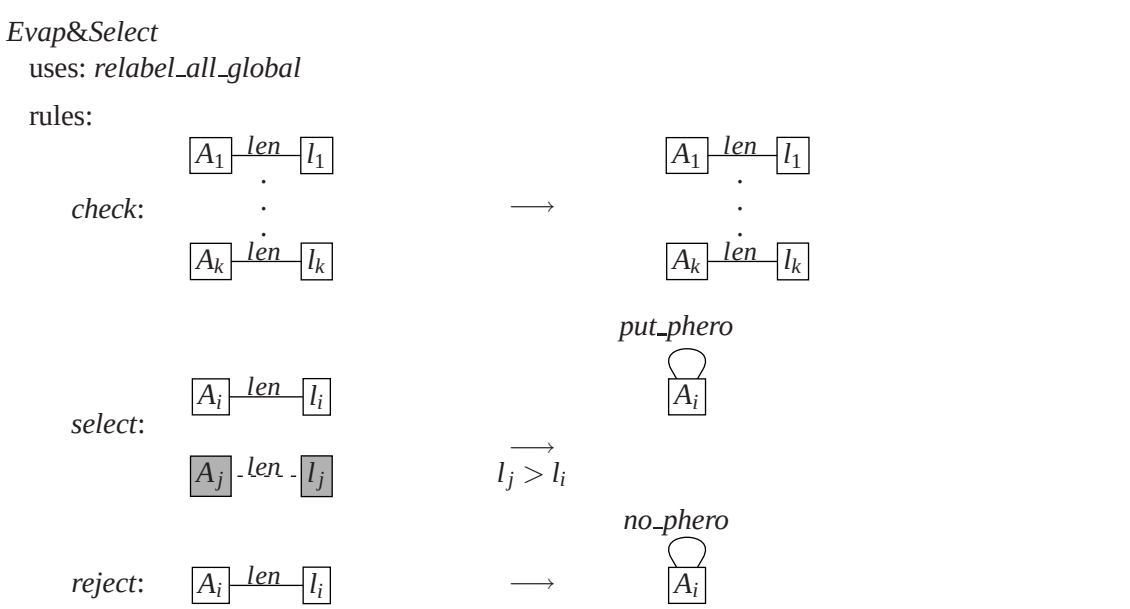


Figure 8: The auxiliary unit put_phero_j



conds: *check* ; *relabel_all_global*(τ: z,τ: $(1-\rho)*z$) ; $select^w$; *reject*!

Figure 9: The autonomous unit *Evap&Select*

vironment with $(1-\rho)$, where $\rho \in (0,1]$ is a pheromone decay parameter. After that, the rule *select* is applied w times (in the control condition this is abbreviated by $select^w$). It finds an ant with the best solution, marks it with a loop *put_phero*, and deletes the information about the length of the ant's solution from the common environment. Each further application of *select* finds a next best solution. When w best solutions are found, the rule *reject* is applied as long as possible to equip the remaining ant nodes with a *no_phero*-loop. This rank-based approach could be extended by the *elitist strategy* (see e.g. [DS04]). In this strategy the best solution so far is memorized and when pheromone update takes place, this tour gets additional pheromone. (In our modeling of the CVRP, we do not consider this strategy because of space limitations.)

Remark. The presented modeling can be used to prove correctness properties a few of which are informally described here.

1. In every execution of $initial\_position_j; tour_j$ a solution is constructed, i.e., a set of cycles of the construction graph is traversed by Ant_j and stored in its memory such that the depot belongs to every cycle, and every customer occurs exactly once in exactly one cycle.

2. The unit *relabel_all_global* models pheromone evaporation.

3. The unit put_phero_j models pheromone update, removes Ant_j from the common environment, and deletes its memory.

4. Each execution of $(Ant_1||\ldots||Ant_k||Evap\&Select)$ models an iteration of the corresponding ACO-Algorithm, i.e., (1) solution construction, (2) pheromone evaporation, and (3) pheromone update.

For reasons of space limitations, proofs of the first three properties are omitted and the proof of the fourth is roughly sketched.

Proof sketch of the fourth property. Let $Aut\_CVRP = \{Ant_1,\ldots,Ant_k,Evap\&Select\}$. Moreover, let $Rules\_CVRP = \bigcup_{aut\in Aut\_CVRP} Rules(aut)$. Before the first iteration the current state is equal to (G,map) with $G \in SEM(Construction\_graph)$ and $map_0(aut) = \emptyset$ for all $aut \in Aut\_CVRP$. Assume that after n iterations the current state is equal to (G',map) where G' is obtained from G via the pheromone evaporation and the pheromone updates of the n previous iterations. Let

$$s \in SEM_{Aut\_CVRP}(Ant_1||\cdots||Ant_k||Evap\&Select)$$

be the transformation sequence of the $(n+1)^{th}$ iteration. Then $s = ((G_0,map_0),\ldots,(G_m,map_m))$ with $G_0 = G'$ and $map_0 = map$ and for each $aut \in Aut\_CVRP$,

$$((G_0,map_0(aut)),\ldots,(G_m,map_m(aut))) \in SEM_{Rules\_CVRP-Rules(aut)}(flC(C_{aut})).$$

This means by the definition of the semantics of control conditions that at first every ant has to execute its rule *initial_position*, then its *tour*-unit, and finally its *put_phero*-unit. The rules in $\bigcup_{j=1}^{k} Rules(Ant_j) - (\bigcup_{j=1}^{k} Rules(put\_phero_j))$ satisfy the independence condition so that they can be applied in parallel. (Please note that according to the definition of the semantics of global control conditions, only rules of different autonomous units can be applied in parallel which also implies that the independence condition of parallel rule application has to be checked only parallel applications of global rules.) The rules in *put_phero* can only be applied if the corresponding ants are equipped with a *put_phero*- or a *no_phero*-loop. This loops can only be generated by the autonomous unit *Evap&Select* which in turn can start working as soon as every unit has finished to execute its *tour*-unit, because its first rule *check* can only be applied after each ant has applied the rule *stop* which is the last rule that is applied in *tour*.

According to the first property of the remark, every ant constructs a tour by executing the rule *initial_position* and then its *tour*-unit. After the application of *check*, the unit *relabel_all_global* is applied which according to the second property models pheromone evaporation. Afterwards, the autonomous unit *Evap&Select* executes $select^w$ followed by as many applications as possible of *reject*. It can be easily shown by induction that for each ant either the rule *select* or the rule *reject* is applied exactly once for every ant. As soon as an ant has got its *put_phero*- or its *no_phero*-loop by the rule *select* or *reject*, it can start to apply the rules of *put_phero*. (Due to the independence condition of parallel rule application the rules of *put_phero* (of different ants) can be applied in parallel if and only if they do not augment the pheromone quantity of the same edge.) By the third property the execution of *put_phero* models pheromone update and deletes the ants from the common environment as well as all private states. Hence, the environment G_m is obtained from G' via pheromone evaporation and pheromone update. Altogether we get that each execution of $Ant_1||\cdots||Ant_k||Evap\&Select$ models solution construction, pheromone evaporation and pheromone update in this order. □

6 Conclusion

In this paper, we have modeled an ACO algorithm for the Capacitated Vehicle Routing Problem as a community of autonomous units. The autonomous behavior of every ant as well as evaporation and selection of the best ants have been modeled as autonomous units running in parallel. The construction graph has been specified by the initial environment specification of the community, and the order in which solution construction, pheromone evaporation, and pheromone update take place has been modeled with the units' control conditions as well as with (negative) context conditions of the units' graph transformation rules. Since all ACO algorithms basically work according to the same underlying algorithm, we believe that they all can be modeled as communities of autonomous units in a natural way.

For solving ACO algorithms in a proper way, we have extended the parallel working autonomous units of [HKK09] by auxiliary units that allow to encapsulate auxiliary tasks in separate units and to manage large rule sets. However, this extension is merely "syntactic sugar" because the parallel semantics is defined for the flattened unit. We have used a separate state for every autonomous unit in order to represent memories of ants. A very similar notion of private states has been introduced in [KK08]. In general, communities with private states can be simulated by communities without private states by adding each private state disjointly to the common environment and labeling it in such a way that it can be accessed by exactly one unit. However, for a proper modeling of ACO algorithms it is meaningful to separate private states (i.e., memories) from the common environment.

Furthermore, we have defined the syntax and semantics of concrete classes of global and unit control conditions consisting of regular expressions extended by a parallelism operator in the global case and extended by and an operator that prescribes to apply a rule as long as possible in the case of control conditions for units.

The modeling of ACO systems as communities of autonomous units has the following advantages. (1) The specification of ants as autonomous units provides the ants with a well-defined operational semantics. (2) The graph transformation rules of autonomous units allow for a visual specification of ant behavior instead of string-based pseudo code as it is often used in the literature. (3) The existing graph transformation systems (cf. e.g. [ERT99, GK08]) are likely to facilitate the visual simulation of ant colonies and hence their verification (see also [Höl08]).

In the future, the following points will be further investigated. (1) The formal semantics of communities of autonomous units constitutes a basis for proving correctness results by induction on the length of the transformation sequences or for examining other characteristics (such as termination) by making use of the wide theory of rule-based graph transformation (see [Roz97]). This should be further explored. (2) This and further case studies should be implemented with one of the existing graph transformation systems so that the emerging behavior of ant colonies can be visually simulated, and ACO algorithms can be verified. For the implementation purpose we plan to use GrGen [GK08] because it is one of the fastest and most flexible graph transformation systems. Further case studies could take into account more advanced elitist strategies as well as dynamic aspects (see e.g. [ES02, DS04, MGRD05, RDH04, RMLG07]). (3) Another interesting task is to investigate how communities of autonomous units can serve as a modeling framework for swarm intelligence in general.

Acknowledgements: We are grateful to Hauke Tönnies for his contribution to a previous version of this paper and to the referees for their valuable comments.

Bibliography

[CEH+97] A. Corradini, H. Ehrig, R. Heckel, M. Löwe, U. Montanari, F. Rossi. Algebraic Approaches to Graph Transformation Part I: Basic Concepts and Double Pushout Approach. Pp. 163–245 in [Roz97].

[DS04] M. Dorigo, T. Stützle. *Ant Colony Optimization*. MIT-Press, 2004.

[EEKR99] H. Ehrig, G. Engels, H.-J. Kreowski, G. Rozenberg (eds.). *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 2: Applications, Languages and Tools*. World Scientific, Singapore, 1999.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs. Springer, 2006.

[EKMR99] H. Ehrig, H.-J. Kreowski, U. Montanari, G. Rozenberg (eds.). *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 3: Concurrency, Parallelism, and Distribution*. World Scientific, Singapore, 1999.

[ERT99] C. Ermel, M. Rudolf, G. Taentzer. The AGG-Approach: Language and Environment. In Ehrig et al. (eds.), *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 2: Applications, Languages and Tools*. Pp. 551–603. World Scientific, Singapore, 1999.

[ES02] C. J. Eyckelhof, M. Snoek. Ant Systems for a Dynamic TSP - Ants Caught in a Traffic Jam. In Dorigo et al. (eds.), *Ant Algorithms - Third International Workshop, ANTS 2002*. Lecture notes in Computer Science 2462, pp. 88–98. 2002.

[GK08] R. Geiß, M. Kroll. GrGen.NET: A Fast, Expressive, and General Purpose Graph Rewrite Tool. In Schürr et al. (eds.), *Proc. 3rd Intl. Workshop on Applications of Graph Transformation with Industrial Relevance (AGTIVE '07)*. Lecture Notes in Computer Science 5088, pp. 568–569. 2008.

[HKK09] K. Hölscher, H.-J. Kreowski, S. Kuske. Autonomous Units to Model Interacting Sequential and Parallel Processes. *Fundamenta Informaticae* 92(3):233–257, 2009.

[Höl08] K. Hölscher. *Autonomous Units as a Rule-based Concept for the Modeling of Autonomous and Cooperating Processes*. Logos Verlag, 2008. PhD thesis.

[HP02] A. Habel, D. Plump. Relabelling in Graph Transformation. In Corradini et al. (eds.), *Proc. First International Conference on Graph Transformation (ICGT '02)*. Lecture Notes in Computer Science 2505, pp. 135–147. 2002.

[KK07] H.-J. Kreowski, S. Kuske. Autonomous Units and Their Semantics - The Parallel Case. In Fiadeiro and Schobbens (eds.), *Recent Trends in Algebraic Development Techniques, 18th International Workshop, WADT 2006*. Lecture Notes in Computer Science 4408, pp. 56–73. 2007.

[KK08] H.-J. Kreowski, S. Kuske. Communities of Autonomous Units for Pickup and Delivery Vehicle Routing. In Schürr et al. (eds.), *Proc. 3rd Intl. Workshop on Applications of Graph Transformation with Industrial Relevance (AGTIVE '07)*. Lecture Notes in Computer Science 5088, pp. 281–296. 2008.

[KLT09] S. Kuske, M. Luderer, H. Tönnies. Autonomous units for solving the traveling salesperson problem based on ant colony optimization. In *Proc. 2nd International Conference on Dynamics in Logistics (LDIC 2009)*. 2009. To appear.

[Kus02] S. Kuske. Parameterized Transformation Units. In Bauderon and Corradini (eds.), *Proc. GETGRATS (General Theory of Graph Transformation Systems) Closing Workshop*. Electronic Notes in Theoretical Computer Science 51. 2002. 12 pages.

[MGRD05] R. Montemanni, L. M. Gambardella, A. E. Rizzoli, A. V. Donati. Ant Colony System for a Dynamic Vehicle Routing Problem. *Journal of Combinatorial Optimization* 10(4):327–343, 2005.

[PS04] D. Plump, S. Steinert. Towards Graph Programs for Graph Algorithms. In Ehrig et al. (eds.), *Proc. 2nd Intl Conference on Graph Transformation (ICGT'04)*. Lecture Notes in Computer Science 3256, pp. 128–143. 2004.

[RDH04] M. Reimann, K. Doerner, R. F. Hartl. D-Ants: Savings Based Ants divide and conquer the vehicle routing problem. *Computers & OR* 31(4):563–591, 2004.

[RMLG07] A. E. Rizzoli, R. Montemanni, E. Lucibello, L. M. Gambardella. Ant colony optimization for real-world vehicle routing problems. *Swarm Intelligence* 1(2):135–151, 2007.

[Roz97] G. Rozenberg (ed.). *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations*. World Scientific, Singapore, 1997.