ECAI 2004
R. Lopez de Mantaras and L. Saitta (Eds.)
IOS Press, 2004

Generation of desired emergent behavior in swarm of micro-robots

Sergey Kornienko and **Olga Kornienko** and **Paul Levi** [1]

Abstract. Emergent behavior of swarm-like-systems results from interactions among system's components and cannot be directly pre-programmed. This kind of behavior is very efficient, flexible and is closely related with collective (swarm-) intelligence. In the presented work we consider a derivation of local rules, creating such interactions, that lead to desired emergent behavior. As an example of collective emergence we choose forming spatial groups and assembling microobjects by a swarm of microrobots.

1 INTRODUCTION

Manipulation of organic and anorganic matter on micro- and nanoscales becomes a new paradigm of modern science. This paradigm appears from two sides. Firstly, technologies from material science, biology and other disciplines dealing traditionally on these scales. Secondly, a development of fully functional molecular-scale devices and robots. Microrobots of the projects MINIMAN, MiCRoN and I-Swarm [8] represent the second trend (see fig. 1). A problem of these robots is that they have very limited computational and communicating resources on board, but have a wide spectrum of tasks (cleaning, microassembling, transportation, collective perception) to be solved. There is no central instance being in charge of coordination.

Figure 1. Microrobots "MINIMAN" (with permission of the Institute for Process Control and Robotics, University of Karlsruhe, Germany).

An approach to control microrobots consists in creating desired collective behavior (like insect swarm-behavior). If the number of robots is large enough (several hundreds of "I-Swarm" robots), they collectively accomplish the common goal. Swarm-like behavior is an emergent property of the system, that cannot be directly preprogrammed. It is created by specific interactions among microrobots. These interactions, in turn, are determined by local rules, governing behavior of every robot.

For "insect-standard" problems, like foraging, route optimization, collective defence and so forth we can find and adopt the rules from the insect-world [4]. However for technical activities, like assembling of microconstructions, we have to derive artificial rules, leading to desired emergent behavior.

Another problem, that we encounter here, consists in regularity and irregularity of desired collective behavioral pattern. We can derive some compact optimization principles or fitness criteria for evolutional generation (like genetic programming) of regular behavioral patterns. However for generating irregular patterns, the irregularities have to be completely described (the Kolmogorov complexity of generating grammar is much higher, than the generated pattern itself). Unfortunately, most of the technically useful behavioral patterns are irregular.

In this paper we consider a derivation of local rules for both regular and irregular behavioral patterns. Examples of tasks to be collectively solved are formations of spatial groups and assembling of microobjects by prototypes of "I-Swarm" robots.

2 TOP-DOWN RULE GENERATION

Generally, local rules can be generated in the bottom-up (local rules $\rightarrow$ emergent behavior) and top-down (emergent behavior $\rightarrow$ local rules) way. The bottom-up approach, or rule-based programming [9], originates from the domain of parallel and distributed computing. Generation of these rules is mostly considered in a context of refining sequential program into concurrent one [3] with correspondingly defined cooperation and coordination [2]. The general problem of bottom-up approach is that we cannot say in advance, which emergent behavior will be generated by the chosen rules (by analogy with the well-known "three-body problem" from nonlinear dynamics [1]). The origin of this problem lies in the enormous complexity of a nonlinearly interacting system. Therefore we suggest to derive local rules from emergent behavior in the top-down way.

The idea behind the top-down approach originates from the distributed AI field. Assume, we have an algorithm, that can decompose the common task Ω into n-subtasks Ω_i. We also have a set of agents $\{Ag\}$ with corresponding elementary activities so as to solve collectively each of Ω_i. The decomposition algorithm splits up each of Ω_i further, up to elementary agent's activities. Thus, we have $\{\Omega_{i=1...n}^{j=1...m}\}$ sequences of activities, where an agent Ag_k needs m steps to solve Ω_i. Since this algorithm decomposes systematically, we can assume that all agents can solve Ω by executing $\{\Omega_i^j\}$. Remark, that a cooperation between agents arises naturally as the top-down decomposition of common task.

From the agents viewpoint, each agent Ag_k has a sequence of activities $S_k = \{\Omega_1, \Omega_2, ..., \Omega_m\}$. Now, calculating Kolmogorov complexity of sequence S_k (finding the smallest grammar [5]), we can derive local rules R_k that can generate S_k. The set of these rules $\{R\}$ defines a cooperation between agents that allows the agent's

[1] University of Stuttgart, Universitätsstr. 38, D-70569 Stuttgart, Germany, email: korniesi@informatik.uni-stuttgart.de

group cooperatively to solve the common task Ω. Thus, the task decomposition algorithm is the kern of the top-down rule generator. Such a decomposition approach (algorithm of symbolic task decomposition - ASTD) is described in [6], here we only sketch the main ideas applied to deriving local rules.

The rule generating machine Rg is a triple (Pl, Ag, Ob) consisting of a planning system Pl, agents $\{Ag\}$, each with activities $\{A_i\}$ and agent-features $\{F_A\}$ and, finally, objects $\{Ob\}$, each with object-features $\{F_O\}$. We say that an activity A and a feature $F_{A,O}$ are of the same type if A can modify $F_{A,O}$ ($F_{A,O} = F_A$ or F_O). For example, the activity "move" modifies the feature "position". More formally, an activity A, parameterized by a technology (modality) D, by technological descriptions of feature $F_{A,O}$, bounded by constraints C_l composes a construction that we denote as the working step WS of an agent: $WS = \{A(D) \to F_{A,O}, \{C_l\}\}$, where $WS \in \mathbb{WS}$. C_l defines local constraints of agents activities, applied only to WS. Now we connect activities with features of the same type. Moreover we require that all feature are connected with corresponding activities (closeness condition). We denote a network of coupled **F**eatures-**A**ctivities as the FA-network, $N_{FA} = (\{A \times F_{A,O}\})$. FA-network plays an important role in the ASTD approach.

Pl is a transition system $Pl = \{\mathbb{T}r(\mathbb{S}) \to \mathbb{S}\}$ and

$$Tr = \{Tr^j : S_i^j \to S_{i+1}^j\}, \;\; S = \{Ob \times WS \to T, \{C_g\}\}, \quad (1)$$

where $S \in (\mathbb{S}^{P_1} \cup \mathbb{S}^{P_2} \cup \mathbb{S}^{P_3} \cup ... \equiv \mathbb{S})$ and $\mathbb{S}$ is a state space of all plans, $\mathbb{S}^{P_i}$ are corresponding subspaces, and $Tr \in \mathbb{T}r$, where $\mathbb{T}r$ is a space of all transitions. State of this plan is a mapping between a working step WS and an object Ob into a time window T, bounded by constraints C_g. A technology (modality) D used in WS is defined in the state S. Here we point to difference between the global constraints C_g originating from the plan and the local ones C_l, originating from activities of an agent.

Remember, that WS consists of different quantities, being of elementary nature. Specifying a complete set of these atomic quantities, we suppose that each state of a plan can be composed from them by a generator Γ. Now we rewrite (1) with the generator Γ

$$\begin{aligned} Tr &= \{Tr^j : S_i^j \to (Ob \times (\Gamma \to F_{A,O}, \{C_l\}) \to T, \{C_g\})\}, \\ S &= \{Ob \times (\Gamma \to F_{A,O}, \{C_l\}) \to T, \{C_g\}\}. \quad (2) \end{aligned}$$

As seen from this expression, the S_i^j is known, but $S_{i+1}^j = \Gamma(S_i)$ is yet unknown. Therefore Tr, besides transition, has in this context a role of a decomposition algorithm. It tells Γ which new state is required on the next step. We divide the space $\mathbb{T}r$ into two subspaces of primary $\mathbb{T}r_p$ and of secondary $\mathbb{T}r_s$ activities. The first one defines a regular planning and the second one is directed to repair a regular planning. $\mathbb{T}r_p$ is predefined and may not be changed, whereas $\mathbb{T}r_s$ is in charge of reactions to disturbances, includes functional decomposition and should be so flexible as possible. Rewriting finally (1), we get

$$Pl = \begin{cases} prim. : Tr_p(Ob \times WS \to T, \{C_g\}), \\ sec. : Tr_s(Ob \times (\Gamma \to F_{A,O}, \{C_l\}) \to T, \{C_g\}). \end{cases} \quad (3)$$

Let us introduce a new transition Tr_{dam}, representing a disturbance $Tr_{dam} = \{Tr_{dam}^j : S_i^j \to S'^j_{i+1}\}$, where the state S'^j_{i+1} is a new perturbed state, deviating from the desired state S_{i+1}^j. If the transition Tr_{dam} perturbs only one feature of an object, we speak about single Tr_{dam}^s, if Tr_{dam} perturbs simultaneously several features of one or more objects, we speak about multiple Tr_{dam}^m. In this work we generally focus only on Tr_{dam}^s. The aim of a planning system is to create a repairing plan Pl_{sec} that returns the system into the state S_{i+1}^j

$$Pl_{sec} = \{Tr_s^j(S'^j_{i+1}) \to S_{i+1}^j\} \quad (4)$$

Considering this expression, we claim there are one-step plans and many-step plans satisfying (4).

Statement 1 (from [6]) *Let N_{FA} be a FA-network and Tr_{dam}^s is a single perturbing transition. If Pl_{sec} is defined as one-step plan, there is always Tr_s in sense of (4), if and only if the corresponding C_l are satisfied.*

Statement 2 (from [6]) *Let N_{FA} be a FA-network and Tr_{dam}^s is a single perturbing transition. If Pl_{sec} is defined as many-step plan, there is always a sequence of Tr_s in sense of (4) if and only if*
- (1) all local constraints C_l are satisfied;
- (2) all global constraints C_g are satisfied;
- (3) Tr_{dam}^s and Tr_s never intersects.

Now the question is of how to derive Tr_s. From (1) we have

$$S = \{Ob \times WS \to T, \{C_g\}\}, S' = \{Ob' \times WS' \to T', \{C_g\}\}. \quad (5)$$

Let us define a difference between S and S' as $\triangle S'$. We assume that any modifications of time T can be absorbed by rescheduling. Therefore functional decompositions from $\triangle S'$ concern only working steps (denoted as $\triangle WS'$). The goal of Tr_s is to minimize $\triangle S'$, i.e. we can write

$$Tr_s(Ob \times WS' \to T, \{C_g\}) = \triangle S' \quad (6)$$

or with the generator Γ: $Tr_s(Ob \times (\Gamma \to F_{A,O}, \{C_l\}) \to$
$\to T, \{C_g\}) = \triangle S'$, where

$$(\Gamma \to F_{A,O}, \{C_l\}) = \triangle WS'. \quad (7)$$

Expressions (6) and (7) give us a practical way to derive Tr_s. Thus, a *task decomposition represents a systematic way to find a difference between real state and desired state in a form of working steps, generated by* Γ. *For many-step plans this rule should be applied on each step of executing, moreover this sequence of generated working states converges in sense of (4).* So far as the problem, before decomposition, should be first formulated in a symbolic form (as the FA-network and the planning system), we call this approach as the algorithm of symbolic tasks decomposition.

Now return to the generation of rules. The perturbed state S'^j_{i+1} represents the initial state of the agents' system. The state S_{i+1}^j represent a desired state, where a common task Ω is solved. The decomposition algorithm produces a sequence of $\{Tr_s\}$ for each agent, so that to achieve S_{i+1}^j from S'^j_{i+1}. **We assume that the sequences $\{Tr_s\}$ have an internal structure, that can be reproduced by a set of generating (local) rules.** This problem is known as estimation of Kolmogorov complexity or approximation of the smallest grammar. There are several known approaches - Bisection algorithm, the scheme LZ77 (see e.g. [5]). **We point out that the local rules generate only a specific behavioral pattern, but do not preprogram each step of an agent.** We illustrate this idea by two following examples.

3 EMERGENT SPATIAL BEHAVIOR

In the first example we derive local rules for the classical problem of spatial formations. We are going to show a construction of the generator Γ and difference between a formation of spatially regular and irregular configuration. Moreover, we compare the efficiency of the "bottom-up" and "top-down" local rules.

Construction of FA-network. We have n agents that have activities of type "move" and feature "position", $Ag_i = (A = \{M\}, F_A = (x, y))$. All positions are calculated in local agent's coordinates, where own position of an agent represents the origin of coordinates. "Move" consists if 9 activities: 8 one-step movements in each compass direction of the 8-neighborhood, as shown in fig. 2(c), and one activity "do nothing", $M = (1, 2, 3, ..., 9)$. In the FA-network we connect a position of one agent with activity "move" of other agent, $N_{FA} = (A(move) \times F_A(x, y))$. It means, *if an agent has to change a position of another agent, it has to "move" itself.* The very simple action system is chosen by the reason of presentation's clearness. Disadvantage consists in a set of initial deadlocks (e.g. all agents are placed on the diagonal line), where agents cannot make any progress.

Construction of the planning system. States of a planer are global spatial positions (x, y) of corresponding corners of spatial shapes. Transitions are distance-relations between these corners. The state S^j_{i+1} is a final constellation of all corner-points, representing this shape, see figs. 2(a)-(c). **Perturbation.** Perturbed state of system S'^j_{i+1} is a set of agents' random initial positions.

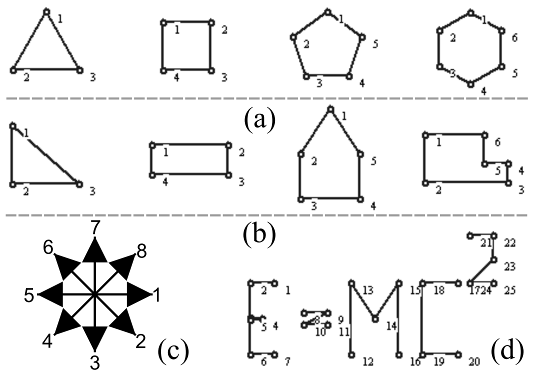


Figure 2. **(a)** The first row: regular spatial formations; **(b)** The second row: irregular spatial formations; **(c)** Sensor's compass directions of target, the same directions are used for actors; **(d)** Complex irregular spatial pattern.

Construction of generator Γ. The generator Γ composes each agent's working step WS so that to minimize $\triangle S'$, i.e. deviation between S'^j_{i+1} and S^j_{i+1} in terms of available activities. It reads a target and a distance from the planning system (the corresponding spatial pattern) and tries to minimize distance between itself and the chosen target

```
D=distance(itself and target[from plan]);

for (int i=1; i<=9; i++) {
 do (virtual Activity i);
 D[i]=distance(itself and target[from plan]);}

find minimal(delta=D-D[j]); do (Activity j);
```

Executing. In the simulation we analyzed sequence of agents' activities in dependence of sensor data ("evolutional" rules). Since the actor's system consists only of 8 movements of the same type, for analysis we do not need complex approaches. For a typical situation, the rules, generating these sequences, have the following form (Ds - distance between agent and target from sensors, i - direction of target, Dp - distance between agent and target from the plan (patterns in figs. 2):

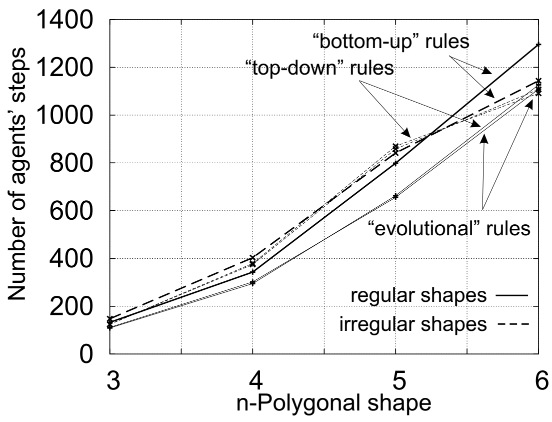


Figure 3. Comparison between the number of steps, needed to reproduce the shapes from figs. 2(a), 2(b). Agents start from random initial conditions, 100×100 square, shown is the average result of 100 000 simulation cycles.

```
if (Ds<Dp) {for even i -> do (Activity i-4);
            for odd i  -> do (Activity i-3);}
if (Ds>Dp) {for even i -> do (Activity i);
            for odd i -> do (Activity i-1);}
```

This result is very surprising. For even sensor directions (2,4,6,8), see fig. 2(c), agent moves directly towards or directly away from a target. However for odd sensor's directions (1,3,5,7) it does not use direct movement, it moves sideways! To prove this unexpected result we compare these "top-down" rules with the following ones

```
if (Ds<Dp)  for i -> do (Activity i-4);
if (Ds>Dp)  for i -> do (Activity i);
```

obtained by a "common sense logic" in the "bottom-up" way. Here we use only direct movement towards a target or away from a target. Comparison between the number of steps, needed to reproduce the given pattern, with different sets of local rules is shown in fig. 3. As shown in this figure, the "bottom-up" rules require the most numbers of steps to reproduce the corresponding shape. The "evolutional" and "top-down" rules differs in $\approx 1\%$, that points to a good quality of approximating rules. By increasing the number of simulation cycles, they are expected to be coincided. The "top-down" and the "bottom-up" sets differ in 5-20%. Another interesting result is that the **"more regular" and "less regular" shapes with the same area** requires

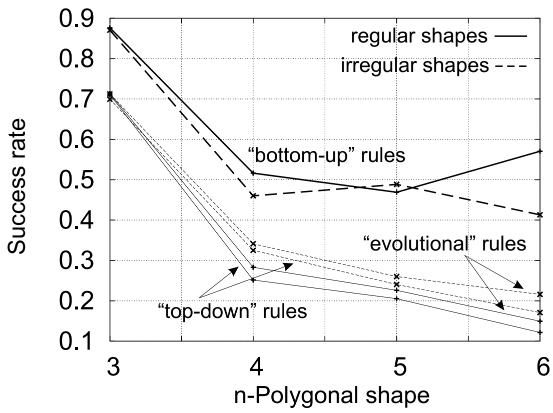


Figure 4. The reproduction rate for random initial conditions.

different number of steps. "Less regular" shapes from fig. 2(b) require more steps.

As already mentioned, the "rudimentary" actor system has some initial deadlocks. As turned out, the "bottom-up" rules are more stable to different initial conditions, as shown in fig. 4. To compare the rules in "ideal" conditions, we place agent in natural order (but also randomly) on the circle. Moreover, we require that a difference between the pattern and the reproduced shape is less then the threshold $Tresh$. In this way the reproduction rate is of ≈99% for all rules for all shapes. In these conditions we compare again the number of steps required to reproduce the shapes for three sets of rules, as shown in fig. 5. We see, that although the difference between "bottom-up" rules and others gets smaller, but anyway they remain less efficient, than "evolutionary" and "top-down" rules.

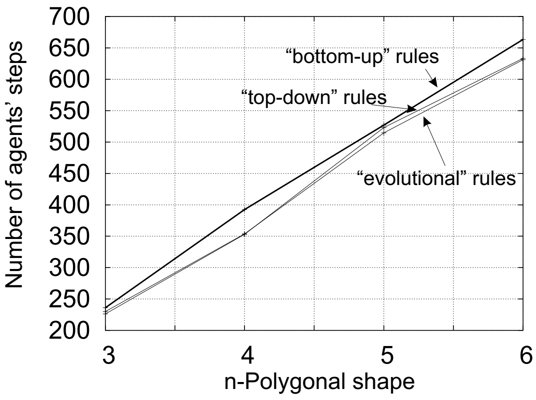


Figure 5. Comparison between the number of steps, needed to reproduce the shapes from fig. 2. Agents start in natural order from random initial conditions in a circle of radius 100, $Tresh = 50$, shown is the average result of 100 000 simulation cycles.

Can the shapes from fig. 2(a)-(b) be generated without reading distances from the pattern ? For equilateral triangle in fig. 2(a) it is possible (for a large number of agents it appears in hexagonal structures). However can the shape from fig. 2(d) be generated in "evolutional" way ? We know only one type of evolution that can reproduce it - namely, the evolution of human civilization ! If we need more "compact" generator, we need to describe all irregularities in the shape. *Remark, that local rules do not predetermine the behavior of agent, they create a specific group's behavioral pattern, that can reproduce any of the shapes from fig. 2.*

4 EMERGENT FUNCTIONAL BEHAVIOR: ASSEMBLING

In the previous example we have shown, that the rules for emergent **spatial behavior** can be derived in the top-down way, and they are more efficient, than the bottom-up rules. However, in the microrobotic scenario, as well as in the industrial manufacturing scenario, we need also different types of **functional behavior**. We consider such a functional behavior on the example of assembling of microobjects.

In this scenario there are two different kinds of agents with different abilities and two kinds of objects with different geometry, see fig. 6. The common task, to assemble them into a construction, can be solved only by a cooperation between agents. An appearance of cooperation we consider as an emergent property of this system.

Construction of FA-network. The first type of agents, Ag^1 can rotate an object, $Ag^1 = (A = \{move, rotate\}, F_A = (x, y))$,

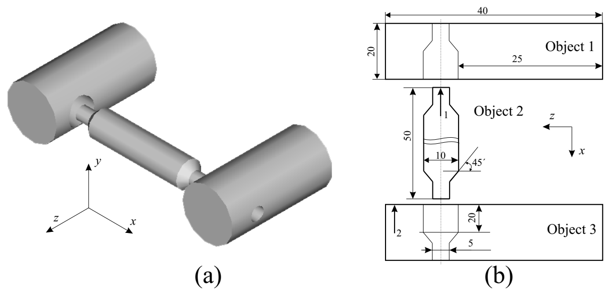


Figure 6. The workpieces to be assembled; **(a)** 3D Representation; **(b)** The x-z section of objects.

where as the second one can transport an object $Ag^2 = (A = \{move, transport\}, F_A = (x, y))$. Both agents have the feature "position" (x, y) in the agent's local coordinate system and have a movement system like the agents from the previous section. Objects have the features "position" (x, y), "rotation angle" α and "geometry" (h, l), $Ob^1 = (F_O = \{(x, y), \alpha, (h_1, l_1)\})$, $Ob^2 = (F_O = \{(x, y), \alpha, (h_2, l_2)\})$. The FA-network N_{FA} connects "rotation" with "angle", "transport" with "position" of objects, and "move" with "position" of agents, $N_{FA} = (A(rotate) \times F_O(\alpha); A(transport) \times F_O(x, y); A(move) \times F_A(x, y))$. Each agent observes neighbors in some radius R_{vis} and within this radius can recognize a distance to target and a rotational angle of target. In order to simplify the FA-network, we do not consider collisions between agents and an agent takes an object by placing itself in the geometric origin of an object (x_0, y_0). Activity of each agent can be represented in the form of Petri-nodes. In order to start an activity, a lot of local restrictions C_l has to be fulfilled.

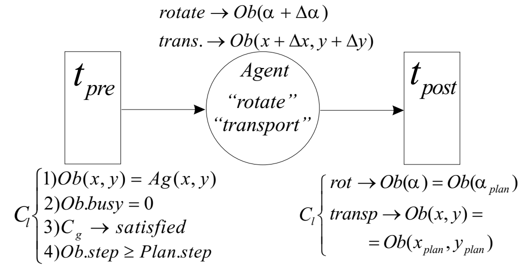
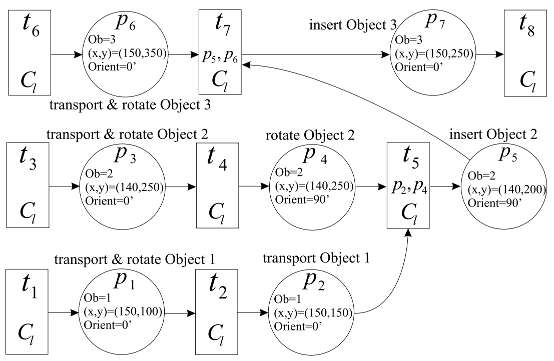

$rotate \rightarrow Ob(\alpha + \Delta\alpha)$
$trans. \rightarrow Ob(x + \Delta x, y + \Delta y)$
t_{pre}
Agent
"rotate"
"transport"
t_{post}
$C_l \begin{cases} 1) Ob(x, y) = Ag(x, y) \\ 2) Ob.busy = 0 \\ 3) C_g \rightarrow satisfied \\ 4) Ob.step \geq Plan.step \end{cases}$
$C_l \begin{cases} rot \rightarrow Ob(\alpha) = Ob(\alpha_{plan}) \\ transp \rightarrow Ob(x, y) = \\ \quad = Ob(x_{plan}, y_{plan}) \end{cases}$

Figure 7. Activity "transport" and "rotate" of agent with local constraints C_l. Activity "move" (see Sec. 3) is called automatically if a position of agent do not coincide with a position of target.

Construction of the planning system. A plan of an assembling has the form of Petri Net, shown in fig. 8. The plan consists of 7 steps, shown as phases p_1-p_7, where the positions and rotation angles are shown. There is the defined order of assembling: the phases p_1, p_3 and p_6 can be started in parallel. However, other phases have to be proceeded sequentially. The phases p_5 and p_7 can be started if p_2, p_4 and p_5, p_6 are finished. These restrictions are the global restrictions C_g. Agents read from plan only relative distances between objects (position of assembling place is marked by a mark). If an agent starts some activity with an object, it marks this object by putting a number of current phase on the mark (e.g. in the electromagnetic way).

Perturbation S'_{n+1}. Agents and objects are placed randomly, but without intersections between objects.

Construction of generator Γ. The generator Γ composes activities so that to minimize $\Delta S'$. However we see, that activities of agents are bounded by constraints C_l and C_g so that they can not be



t_6 C_l
p_6 Ob=3 (x,y)=(150,350) Orient=0'
t_7 p_5, p_6 C_l
insert Object 3
p_7 Ob=3 (x,y)=(150,250) Orient=0'
t_8 C_l
transport & rotate Object 3
transport & rotate Object 2
t_3 C_l
p_3 Ob=2 (x,y)=(140,250) Orient=0'
t_4 C_l
rotate Object 2
p_4 Ob=2 (x,y)=(140,250) Orient=90'
t_5 p_2, p_4 C_l
insert Object 2
p_5 Ob=2 (x,y)=(140,200) Orient=90'
transport & rotate Object 1
t_1 C_l
p_1 Ob=1 (x,y)=(150,100) Orient=0'
t_2 C_l
transport Object 1
p_2 Ob=1 (x,y)=(150,150) Orient=0'

Figure 8. The assembling plan. P_i are phases, where t_i are transitions with conditions shown in the squares (e.g. conditions for t_7 are the satisfaction of local constraints C_l from fig. 7, and the finished phases p_5, p_6).

generated in the way, shown in Sec. 3 (generally we solve this problem as the constraint satisfaction problem (CPS) [7]). In this case the generator Γ minimizes $\Delta S'$ simply by choosing different order of allowed phases p_i (here we do not consider other group's strategies):

```
i=take random phase; do (planning activities);
choice i so that to minimize common time;
```

Executing. Each agent looks for objects within R_{vis} and reads the objects' marks. It takes C_g and the modality D from the plan. The agent's local rules consist of C_l from fig. 7 and the rule "do close phase(Activity)", obtained from the generator Γ:

```
Ob=look for (visible objects); read mark (Ob);
if (constraints(Ob)) do close phase (Activity);
```

This additional rule optimizes cooperation between agents and means, if an agent has a choice, it chooses an activity most closely to the first phase. The generated agent-agent cooperation is shown in fig. 9. Now we try to "improve" this cooperation by putting the

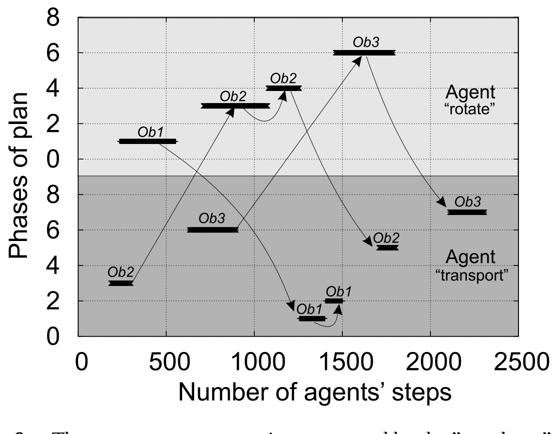


Figure 9. The agent-agent cooperation, generated by the "top-down" rules.

additional "bottom-up" cooperation rule:

```
I'm Ag_i; if ($Ag_j$=take the same Ob as I){
Ob belongs to Ag with smaller distance to it;}
```

In fig. 10 we show the comparison between the "bottom-up" and "top-down" rules. For small n, the "top-down" rules are more efficient. However, if n grows, new group's strategies appear and we have correspondingly to modify the generator Γ.

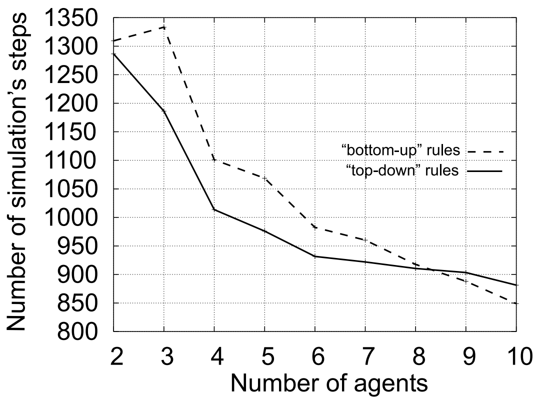


Figure 10. Comparison between the "bottom-up" and "top-down" rules. Agents start from random initial conditions,100×100 square,$R_{vis} = 400$, shown is the average result of 100000 simulation cycles.

5 CONCLUSION

Top-down derivation of rules. In this paper we have shown the top-down derivation of rules that generate the desired emergent behavior. As shown in spatial and functional cases, these rules lead to more efficient behavior, than the corresponding bottom-up derived rules. The obtained local rules are very compact and can be implemented as chip-built-functions in each microrobot. Treatment of this point as well as an evolutional generation of the desired patterns (figs. 2 and 8) remains for an extended version of this paper.

Scalability. The system based on top-down derived rules is scaled-free (tested up to 1000 agents). However if the number of agents grows, the system can change collective strategy, as shown in fig. 10. Because of limited size, this point also remains for an extended paper.

Acknowledgment. The presented work is made in the framework of SFB 467 "Transformable Business Structures for Multiple-Variant Series Production" (supported by the German Research Foundation) as well as EU-Project "Intelligent Small World Autonomous Robots for Micro-manipulation" (I-Swarm).

REFERENCES

[1] V.I. Arnold(Ed.), *Dynamical systems III*, Springer Verlag, Berlin, Heidelberg, New York, 1988.

[2] R. J. R. Back and F. Kurki-Suonio, 'Distributed cooperation with action systems', *ACM Trans. Program. Lang. Syst.*, **10**(4), 513–554, (1988).

[3] R. J. R. Back and K. Sere, 'Stepwise refinement of action systems', *Structured Programming*, **12**, 17–30, (1991).

[4] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm intelligence: from natural to artificial systems*, Oxford University Press, New York, 1999.

[5] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Rasala, A. Sahai, and A. Shelat, 'Approximating the smallest grammar: Kolmogorov complexity in natural models', in *Proc. of the 34th ACM symposium on Theory of computing*, pp. 792–801. ACM Press, (2002).

[6] S. Kornienko, O. Kornienko, and P. Levi, 'Multi-agent repairer of damaged process plans in manufacturing environment', in *Proc. of the 8th Conf. on Intelligent Autonomous Systems (IAS-8), Amsterdam, NL*, pp. 485–494, (2004).

[7] S. Kornienko, O. Kornienko, and J. Priese, 'Application of multi-agent planning to the assignment problem', *Computers in Industry*, **54**(3), 273–290, (2004).

[8] MINIMAN, MiCRoN, and I-Swarm, '(miniman) esprit-project-33915, (micron) ist-2001-33567, (i-swarm) ist fet-open project 507006'. EU Projects.

[9] G.-C. Roma, R. F. Gamble, and W. E. Ball, 'Formal derivation of rule-based programs', *IEEE Trans. Softw. Eng.*, **19**(3), 277–296, (1993).