Population Coding: A New Design Paradigm for Embodied Distributed Systems

Heiko Hamann[1], Gabriele Valentini[2], and Marco Dorigo[2]

[1] Heinz Nixdorf Institute, Department of Computer Science, University of Paderborn, Germany
[2] IRIDIA, Université Libre de Bruxelles, Brussels, Belgium
heiko.hamann@uni-paderborn.de, gvalenti@ulb.ac.be, mdorigo@ulb.ac.be

Abstract. Designing embodied distributed systems, such as multi-robot systems, is challenging especially if the individual components have limited capabilities due to hardware restrictions. In self-organizing systems each component has only limited information and a global, organized system behavior (macro-level) has to emerge from local interactions only (micro-level). A general, structured design approach to self-organizing distributed systems is still lacking. We develop a general approach based on behaviorally heterogeneous systems. Inspired by the concept of population coding from neuroscience, we show in two case studies how designing an embodied distributed system is reduced to picking the right components from a predefined set of controller types. In this way, the design challenge is reduced to an optimization problem that can be solved by a variety of optimization techniques. Our approach is applicable to scenarios that allow for representing the component behavior as (probabilistic) finite state machine. We anticipate the paradigm of population coding to be applicable to a wide range of distributed systems.

1 Introduction

The complexity of engineered systems is getting more and more difficult to govern. We require novel methodologies to enable us to reliably engineer systems also in the future. Combining distributed systems with self-organization has high potential to solve this curse of complexity and is a promising pathway. However, distributed computing systems are more difficult to program than a single-CPU computer due to parallelism, asynchronism, and uncertain interactions between components. This problem is even more pronounced for self-organizing systems whose high standards of scalability and robustness are usually achieved through methods restricted to local interactions and local information. In embodied distributed systems we additionally face hardware limitations concerning computational power, communication range, and energy autonomy. Examples of embodied self-organizing systems are self-organizing networks [8], multi-robot systems [11], and robot swarms [6]. The main challenge is due to the strict locality of individual components which requires the program code to be written from the local component perspective (micro-level) while tasks are defined at the system level (macro-level) [1, 2]. An exhaustive design strategy needs to establish a

so-called micro-macro link [14, 3] that connects the micro-level information processing with its effect on the macro-level. An additional challenge of distributed systems that have mobile components (e.g., robot swarms) is their dynamic interaction network (time-variant neighborhoods) which complicates the derivation of a micro-macro link.

In some applications of embodied distributed systems, as nanorobotics [15], the hardware requirements are extremely strict [17] and limit the core functions of such robots. While robots are usually built as reprogrammable devices, it might prove difficult to build nanorobots with that property. In certain scenarios the only feasible approach might be to hardwire the control logics. This potential lack of flexibility in programming individual robots motivates us to implement flexibility at the macro-level. We do so by composing heterogeneous swarms of robots with different hardwired controllers to obtain a desired macro-level behavior. In other words, our swarm is behaviorally heterogeneous, that is, individual components or subpopulations are allowed to differ in their controller. Examples of heterogeneous robot swarms include the Swarmanoid project [7] and an approach inspired by honeybees [16].

Related to our approach are the methods proposed by Berman et al. [4] and Prorok et al. [19]. These methods have the advantage of high scalability because they operate on continuous, macroscopic models. Their disadvantage is that they either rely on non-communicating agents or they require a centralized authority that gathers information. Hence, there is either no cooperation or a single point of failure which is not coherent with swarm intelligence.

2 Population coding and hardwired controllers

Our main idea is inspired by the concept of population coding which is a method from neuroscience to relate a stimulus to the activity of a neuron population [18, 10]. For example, the direction of movement can be encoded within a population of neurons with each of them representing a preferred direction. Population coding, according to Georgopoulos et al. [10, p. 1416], can be understood in the following way: "When individual cells were represented as vectors that make weighted contributions along the axis of their preferred direction [...] the resulting vector sum of all cell vectors (population vector) was in a direction congruent with the direction of movement". Population coding can be mathematically interpreted as a function approximation with basis functions [18]. Each neuron's preference for a certain direction is represented by a Gaussian function as basis function and each neuron's activity gives the weight of the respective basis function to approximate a 'stimulus function'. We define an approach to compose heterogeneous distributed systems by combining subpopulations of components of different predetermined, possibly hardwired behavioral types (see Fig. 1). In analogy to population coding, we select quantities of different controller types of the micro-level (weighted contributions of neurons) that sum up to a system behavior of the macro-level (population vector). The choice of controller type compositions is implemented as an optimization process that minimizes deviations from the user-specified system behavior.

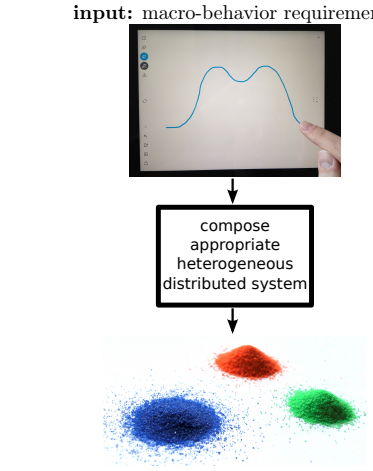


Fig. 1. Schematic representation of our approach. A user specifies the desired system behavior, for example, in the form of a potential field. Our algorithm derives an appropriate heterogeneous composition of controller types (microscopic behavior), and outputs the respective type numbers to compose the macroscopic behavior.

We allow to specify the desired system behavior in two different ways. The preferable approach is to define the macroscopic behavior, that is, working at the same level at which the actual task is defined. However, this approach requires to first define the control states and possible transitions and then, in a second step, to analytically derive a micro-macro link [14] to guide the optimization process. The alternative approach is to skip the second step. Then control states and a performance measure of the macroscopic behavior are required but not the micro-macro link. For example, one specifies the states of a finite state machine (FSM) where the conditions of state transitions are undetermined but parameterized. In this way we define a simple, user-friendly methodology to design the behavior of a distributed system. In the following, we give an example for both options: the definition of the desired system via a macroscopic behavior together with a micro-macro link or via a microscopic behavior using a FSM.

3 Scenario A: Task allocation

As first example we investigate a task allocation problem in the form of a special collective decision-making scenario [20]. Instead of having a single optimal assignment of agents to tasks, we assume that our task allocation problem allows two or more different assignments of equal utility. This is therefore a collective decision-making problem because the group of agents is free to choose one of the different assignments but has to collectively agree on which option to choose. We allow the agent group to switch between the desired task assignments at stochastic time intervals to allow for a probabilistic control approach. We focus on a binary task allocation problem [5] that requires a group of robots to obtain a desired allocation to tasks T_1 and T_2. All robots in the swarm are capable of performing both tasks and the overall goal is to reach an appropriate distribution of workforce that maximizes the swarm performance. For example, the user

could specify a swarm fraction $s = N_1/(N_1+N_2)$ of how many robots N_1 should be assigned to task T_1 which in turn defines the fraction $1-s = N_2/(N_1+N_2)$ to be assigned to task T_2. The user can also specify several swarm fractions s_1, s_2, etc., which requires a multistable system. We take a bistable distribution as an example (e.g., see Fig. 3b). Both peaks in the distribution define a particular allocation of robots (the variance around each peak influences the switching time between allocations).

Similarly to [13, 20] we define the controller types (i.e., the microscopic behavior) as a chemical reaction network. We assume that a robot perceives the current task allocations of its neighbors and decides to switch between tasks or to recruit a neighbor based on this information. For simplicity, we define reaction rules depending on a fixed neighborhood size of seven (i.e., the considered robot and its six neighbors). We define a rule for each neighborhood configuration (except for neighborhoods where all robots are assigned to the same task):

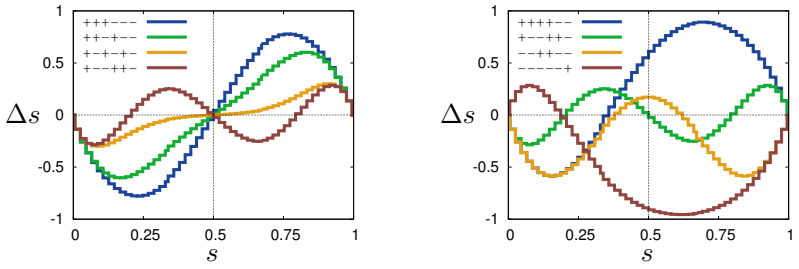
$$(7-x)T_1 + xT_2 \xrightarrow{r} (7-x+\delta_i)T_1 + (x-\delta_i)T_2, \tag{1}$$

where $x \in \{1, 2, \ldots, 6\}$ is the number of neighbors assigned to task T_2, $\delta_i \in \{+1, -1\}$ define the behavior of the focal robot, $i \in \{1, \ldots, 6\}$ is the rule index, and r is the reaction rate coefficient. The parameter δ_i defines the effect of a rule, that is, whether the current number of robots assigned to task T_1 is increased or decreasded by 1 unit. Depending on its current task, the considered robot either switches its own task assignment or recruits a neighbor to do so. The combinatorics of all assignments δ_i, $i \in \{1, 2, \ldots, 6\}$ gives $2^6 = 64$ different controller types that we enumerate and their index gives the δ_i as binary encoding. For example, controller type $R^{56} = (+++ \text{ - - -})$ implements a majority rule because an observed majority of T_1 (respectively T_2) has the effect of increasing the majority by one robot. Type $R^7 = (\text{- - - } +++)$ implements a minority rule because an observed minority of T_2 (respectively T_1) has the effect of increasing the minority by one robot. Finally, we define two more reactions to model a spontaneous switching behavior: $T_1 \xrightarrow{e} T_2$ and $T_2 \xrightarrow{e} T_1$ with a reaction rate coefficient e. Spontaneous switching is required to avoid absorption in the macroscopic states where all robots of the swarm are assigned to one of the two tasks.

We defined 64 different robot controller types that we can use to compose heterogeneous swarms. The idea of producing 64 different hardwired controllers might seem to come with considerable overhead, however, in the following we show that only a few types are used in a controller composition and minimizing the number of controller types can be an additional optimization objective (sparsity).

3.1 Micro-macro model

In the second step of our approach we need to establish a micro-macro link to model mathematically the contribution of each controller type to the macroscopic behavior of the swarm. Based on this micro-macro link we can derive a proper composition C of controller types that forms a heterogeneous swarm satisfying



(a) symmetric (b) asymmetric

Fig. 2. Examples of individual contributions to the expected change Δs of the swarm behaviors according to eq. 3 with $e = 0$. Symbols '+' and '-' corresponds to values of $\delta_i \in \{+1, -1\}$.

the input of the user. A composition is defined as $C = (n_1, \ldots, n_{64})$ based on the above defined 64 controller types, where each n_i gives the number of required robots of the corresponding type and $\sum_i n_i = N$ for swarm size N. Similarly to [13], we define a micro-macro link based on the expected contribution of each controller type to the change of the swarm state (macro-level). The swarm state is the current fraction $s \in [0, 1]$ of robots that are assigned to task T_1 (respectively, $1 - s$ for task T_2). The swarm state s varies because robots interact with each other. The dynamics of the swarm state is modeled by the expected change Δs of s which, in turn, is the sum over the contributions of each controller type.
In the following we assume for simplicity that the robots are spatially well-mixed with respect to their current task allocation. That allows us to calculate the probability of a certain configuration of a robot's neighborhood using the hypergeometric distribution

$$P(k, m) = \frac{\binom{N-m}{G-k}\binom{m}{k}}{\binom{N}{G}}, \tag{2}$$

whereas k is the number of robots allocated to task T_1 in the neighborhood and m is the number of robots allocated to task T_1 in the whole swarm. With probability $P(k, m)$ a controller of type j contributes to the swarm behavior with a change $\Delta R_k^j \in \{+1, -1\}$. As a consequence, its contribution to the expected change is $\Delta R_k^j P(k, m)$. A robot also contributes by spontaneously switching its task allocation. With probability $P = s$ a robot allocated to task T_1 switches to task T_2 contributing a change of -1 robots allocated to T_1 and vice versa. The contribution to the expected change is $-1s + 1(1 - s) = (1 - 2s)$.
In Fig. 2 we give examples of the expected change Δs for a few controller types. The main effect of Δs on the system can be interpreted visually. For $s > 0.5$ in Fig. 2, $\Delta s > 0$ represents positive feedback that drives the system towards the extreme of $s = 1$, while $\Delta s < 0$ represents negative feedback that drives the system towards the balanced state of $s = 0.5$. In Fig. 2a we show the symmetric

expected change Δs of 'symmetric' controller types. In Fig. 2b we give expected changes of 'asymmetric' controller types.
The micro-macro link is established by calculating the expected swarm change Δs which is a sum over all controller types in a chosen composition C and for each of them all possible neighborhood configurations are considered. We obtain

$$\Delta s(s, C) = \sum_{j=1}^{64} \frac{n_j}{N} \left[\frac{r}{r+e} \sum_{k=1}^{G-1} \Delta R_k^j P(k, m = sN) + \frac{e}{r+e}(1-2s) \right], \quad (3)$$

where n_j is the number of robots of controller type j in the swarm and hence n_j/N weights the contribution of each controller type.

3.2 Evolutionary approach

For a given user input, we formulate the derivation of the appropriate heterogeneous swarm as an optimization problem using the micro-macro link. A user provides as input a potential field $p(x)$ that characterizes the desired allocation of robots. For swarm size N the potential field p is defined by $N+1$ data points $p(i) = y$, $i \in \{0, 1, \ldots, N\}$. Each maximum defines a desired allocation. For example, a maximum $p(j) = c_{\max}$ defines that the user requires the allocation of j robots to T_1 and $N-j$ robots to T_2. First, we translate the desired potential field to its corresponding expected change $\widehat{\Delta s}$ via the discrete derivative

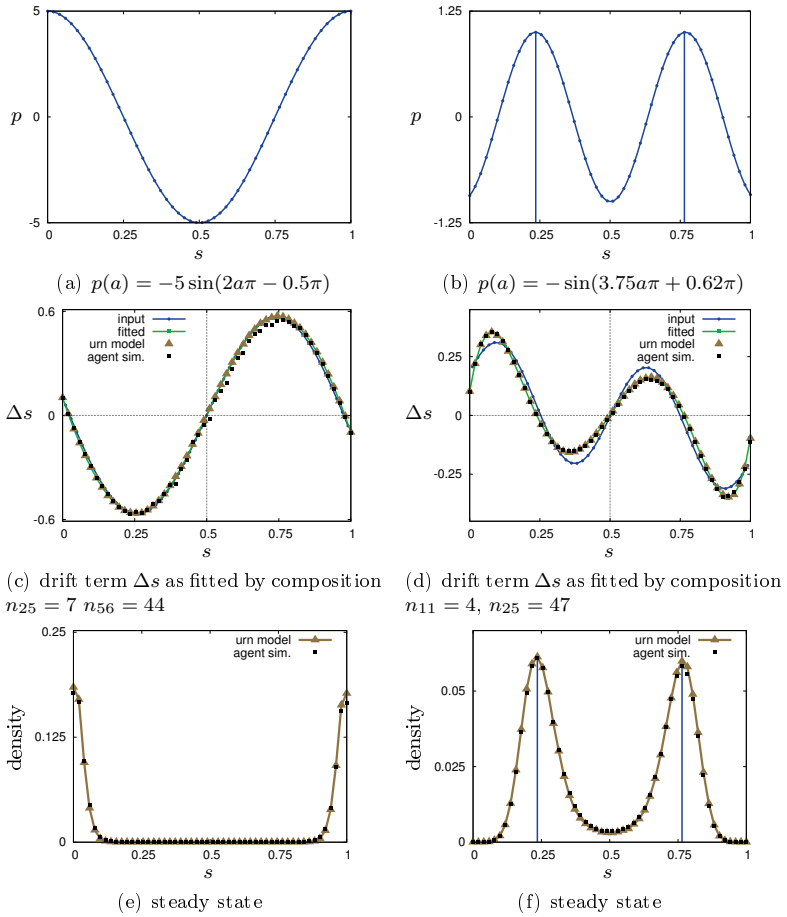
$$\widehat{\Delta s}(s) = \frac{d}{ds} p(sN). \quad (4)$$

Second, we formulate an optimization problem. We consider the error between the user input $\widehat{\Delta s}(s)$ and the micro-macro link Δs as defined in eq. 3. We want to find the optimal controller type composition C_{opt} (and optimal rates r and e) that minimizes the squared error

$$C_{\text{opt}} = \arg\min_{C} \sum_{x \in \{0,1,\ldots,N\}} (\widehat{\Delta s}(xN) - \Delta s(xN, C))^2. \quad (5)$$

Third, we solve the optimization problem with an appropriate technique. There are many options, in [21] we present an efficient, model-driven approach using lasso regression. In this paper, we use a genetic algorithm[3].
We evolve controller type compositions with a population size of 50 compositions for 50 generations. Mutation is implemented as bit flip with probability 0.02 and mutated compositions are corrected to guarantee $\sum_i n_i = N$. The fitness function is defined by the sum in the right side of eq. 5. We have tested this approach for the two different user inputs shown in Fig. 3. Figs. 3a and b show the user input as potential fields (2 maxima, hence 2 optimal task assignments, which requires collective decision-making at runtime). The desired macroscopic

[3] We use an implementation for the R project "NMOF: Numerical Methods and Optimization in Finance" ('NMOF') by Enrico Schumann, see http://cran.r-project.org/web/packages/NMOF/.



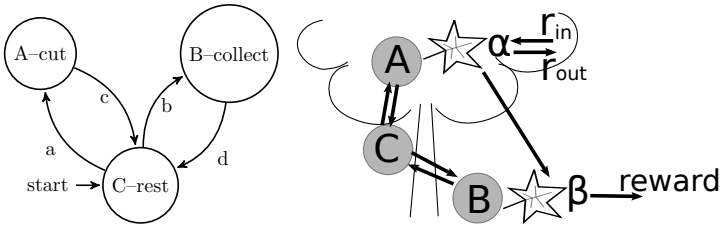
(a) $p(a) = -5\sin(2a\pi - 0.5\pi)$ (b) $p(a) = -\sin(3.75a\pi + 0.62\pi)$

(c) drift term Δs as fitted by composition $n_{25} = 7$ $n_{56} = 44$ (d) drift term Δs as fitted by composition $n_{11} = 4$, $n_{25} = 47$

(e) steady state (f) steady state

Fig. 3. Scenario A, (a, b) examples of potential fields $p(x)$ defined using sine waves as user input, (c, d) expected change $\widehat{\Delta s}$, resulting expected change using the evolved controller type compositions, and the validation (urn model and agent-based simulations), (e, f) steady states for the evolved controller type compositions.

behavior can be understood as a hill climber that is greedily walking uphill towards the peaks but is also subject to fluctuations. In Figs. 3c and d we give the expected change $\widehat{\Delta s}$ as defined by the user input (eq. 4) and the resulting expected change of the evolved (fitted) controller type composition. In the first example, we obtain an almost perfect fit (Fig. 3c) by a composition of $n_{25} = 7$ and $n_{56} = 44$ (all other $n_j = 0$). Controller type 56 corresponds to decision rule (+++ - - -) that implements a majority decision. Controller type 25 corresponds to rule (- ++ - -+) which shifts the maxima towards the boundaries ($s = 0$, $s = 1$) and lowers the amplitude of the expected change Δs to match the user input. In the second example we obtain a controller type composition consisting



(a) FSM (b) schematic

Fig. 4. Scenario B, finite state machine (FSM) and schematic representation of FSM showing a tree, leaves on tree α, leaves at ground β.

of types $n_{11} = 4$ (- - + - ++) and $n_{25} = 47$ (- ++ - - +), see Fig. 3d. Despite some errors, the zeros of the user input (i.e., desired task allocations) are well approximated.We validate our approach using a simple urn model following [12] and determine the expected change and the steady state numerically (see Fig. 3c-f). We also use a simple, agent-based simulator. Agents are massless points randomly walking in a square area. Each agent is characterized by its controller type and task allocation. Results are averaged over 10^4 simulations of 10^4 time units each (see Fig. 3). We obtain very good fits.

4 Scenario B: Sensor-based transitions

As a second example, we test the alternative method without a micro-macro link where the user specifies a template of a micro-behavior as a FSM. We take a scenario loosely inspired by leafcutter ants as described by Ferrante et al. [9]. While Ferrante et al. focus on the emergence of task specialization and evolve individual behaviors starting from predefined low-level primitives, we get a predefined task specialization (e.g., cutting, collecting) as user input in the form of the control states of a FSM (see Fig. 4a). This can be considered as a template of a class of allowed microscopic behaviors and is in contrast to scenario A where the macroscopic behavior was predefined but not the microscopic behavior. We then enumerate a finite set of FSMs with determined conditions of state transitions and evolve the desired macroscopic behavior (here, maximization of the number of collected leaves) by composing a heterogeneous swarm of these predefined individual behaviors.

Leaves are added to the system at a rate r_{in} and decay with a rate of r_{out}. An agent can be in one of three states (see Fig. 4a): cutting leaves (A), collecting leaves (B), or resting (C). The area is divided into two spaces: leaf cutting agents A are on the tree and collecting agents B are at the ground. Once an agent has cut a leaf α on the tree, the leaf falls down to the floor and becomes β ($\alpha \rightarrow \beta$, see Fig. 4b). We only allow transitions between states A and C and between B and C but not directly between A and B to simplify the scenario. Leaf cutting agents A can only perceive 20% of other leaf cutting agents and

20% of the leaves on the tree. Collecting agents B can only perceive 20% of other collecting agents and 20% of leaves on the floor. The desired global behavior is an efficient system that maximizes the number of collected leaves. We say that agents in state resting (C) save energy and give a reward for each agent spending a time step in C.

Next, we define the controller types used to compose a heterogeneous system. Each agent locally perceives the number of neighboring agents that are in the same state and also the number of nearby leaves. For each transition as labeled in Fig. 4a, we define a rule using the notation of reaction equations:

$$a : x_1\alpha + C \to x_1\alpha + A, \tag{6}$$

$$b : x_2\beta + C \to x_2\beta + B, \tag{7}$$

$$c : x_3\alpha + y_1 A \to x_3\alpha + (y_1 - 1)A + C, \tag{8}$$

$$d : x_4\beta + y_2 B \to x_4\beta + (y_2 - 1)B + C, \tag{9}$$

and for two transitions that model the leaves we have $\alpha + A \to \beta + A$ and $\beta + B \to B$ (associated with a reward for collecting a leaf). We limit the six parameters $x_i, y_j \in \{0, 1, 2, 3\}$ for simplicity giving us a total of $6^4 = 1296$ rules to choose from. We can represent a controller type by a 6-tuple: $(x_1, x_2, x_3, x_4, y_1, y_2)$. A particular controller type can have any choice of x_i and y_j for each individual rule (eqs. 6-9). An example controller is

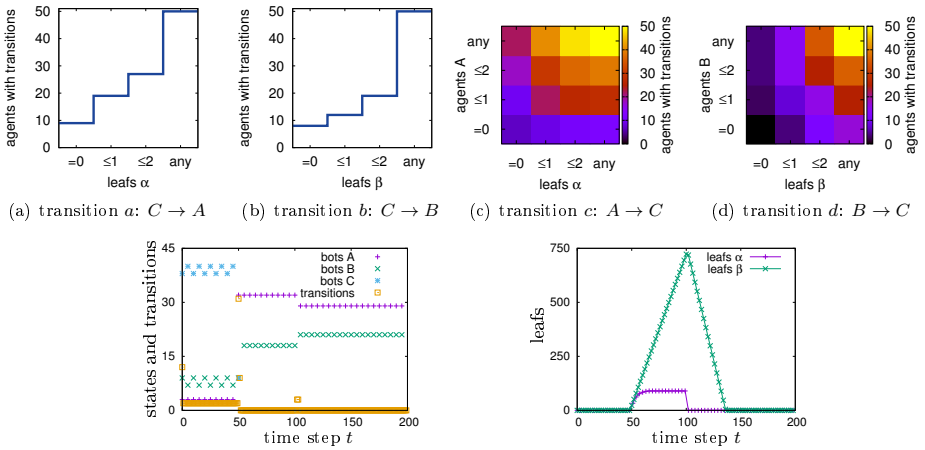
$$a : \alpha + C \to \alpha + A, \tag{10}$$

$$b : 2\beta + C \to 2\beta + B, \tag{11}$$

$$c : 3\alpha + 2A \to 3\alpha + A + C, \tag{12}$$

$$d : 2\beta + B \to 2\beta + C. \tag{13}$$

We test a setup with $N = 50$ agents. The experiment is done in four phases by changing the leaf adding rate r_{in} every 50 time steps for a total of 200 time steps. We evaluate each controller type composition in six tests (the fitness is the average over these six tests) with different sequences of r_{in}: (0, 50, 0, 20), (50, 0, 20, 0), (20, 20, 20, 20), (10, 10, 10, 10), (1, 5, 10, 15), (15, 10, 5, 1). Such a task allocation problem could be solved with adaptive response thresholds in each agent, but we restrict the agents to be non-adaptive. Hence, the adaptivity has to emerge at the macro-level. We use a genetic algorithm to find good controller type compositions. We reward (each with equal weight) for each collected leaf, for agents staying in state C (rest) per time step, and for sparsity (i.e., use of few different controller types). In Fig. 5 we give results for the best evolved controller type composition. It receives a fitness of 0.64 averaged over all six tested leaf inflow sequences. The best homogeneous swarm (i.e., using the same controller type for all agents) received a fitness of 0.56. The best heterogeneous composition assigns 17 different controller types to the 50 agents, most agents (eight) are assigned the controller type with values $(x_1 = 3, x_2 = 3, x_3 = 0, y_1 = 2, x_4 = 2, y_2 = 0)$ (cf. eqs. 6-9). Figs. 5a and b give the cumulative number of agents with transitions for 0, 1 or less, 2 or less, and any number of seen leaves



(a) transition a: $C \to A$ (b) transition b: $C \to B$ (c) transition c: $A \to C$ (d) transition d: $B \to C$

(e) num. agents in each state and num. transitions (f) leafs

Fig. 5. Scenario B, (a-d) resulting number of state transitions (transition labels as in Fig. 4a) for evolved controller type composition based on 17 different controller types, (e) number of agents per state, their transitions, and (f) the number of leaves in the system over time for the evaluation with leaf inflow r_{in} sequence (0, 50, 0, 20).

as sum over all controller types used by the evolved composition for reaction rules 6 and 7 that depend on leaves only. We observe that fewer leaves suffice for transition a while transition b requires more leaves, hence a switch from C to state A is done more easily. Figs. 5c and d give the cumulative number of agents with transitions as weighted sum over all controller types used by the evolved composition for reaction rules 8 and 9 that depend on two variables each (agents and leaves). We notice that few leaves suffice for transition c while transition d requires more. Hence, a switch back from A to C is done more easily which corresponds to our finding for transition a. Figs. 5e and f give the number of agents per state, their transitions, and the number of leaves in the system over time for the fitness evaluation with leaf adding rate sequence (0, 50, 0, 20). For $t < 50$ we observe only up to 12 agents that are not in state C which is efficient although suboptimal. For $50 < t < 100$ there are 32 agents cutting leaves (A), 18 agents collecting leaves (B), and none resting. Having no resting agents in this period is important because otherwise more α leaves would be lost due to $r_{\text{out}} > 0$ (see Fig. 5f). For $t < 100$ the number of collecting agents B is increased at the cost of cutting agents A which is reasonable to process the pile of β leaves. Note that an optimal solution for this particular leaf inflow sequence may limit the success for other sequences, hence, the controller type composition needs to be a compromise.

5 Discussion and Conclusion

We have described our new design paradigm for embodied distributed systems which is inspired by population coding. Our approach relies on predetermined controller types that can be combined in appropriate amounts to compose a heterogeneous swarm. In scenario A, we have shown how we can evolve a heterogeneous swarm with the desired behavior for a user-specified macroscopic behavior by leveraging on a micro-macro link. In scenario B, we have shown how a heterogeneous swarm can be evolved for a user-specified microscopic behavior template in the form of a FSM using agent-based simulations. Scalability and robustness might impose a challenge to our approach.

Scaling the number of states and state transitions in the FSM is potentially problematic due to the combinatorial explosion of the search space. This issue can be addressed by limiting a priori the number of possible state transitions as done in scenario B where the FSM is not a complete graph. An alternative approach is to leverage on optimization methods specifically conceived to achieve sparsity and that are computationally efficient also in high-dimensional search spaces (e.g., lasso for regularized regression as done in [21]).

Robustness might be problematic due to the heterogeneity of the system. In a homogeneous system any loss of an agent is compensated by all other agents. In a heterogeneous system, however, agents are different and cannot be replaced by every other agent. In our approach we tackle this problem by pushing towards sparse solutions, that is, controller type compositions that make use of only a few different controllers. With this approach, the probability that a single controller type is represented by few agents is lowered. Still, sparsity is not sufficient because only few robots may be assigned to a certain controller type. A solution could be to maximize this quantity in addition to minimizing the number of controller types. Furthermore, a radical approach could be to provide each agent with all controller types (i.e., back to homogeneous swarms) and to allow the agents to switch between controller types probabilistically. Then each agent would execute each controller from the composition with an appropriate probability that depends on how many agents are assigned to that controller type in the composition. However, we would loose the property that we can prefabricate the different controller types. Hence, it seems that there is an unavoidable tradeoff between robustness and design flexibility at the macro-level.

In future work, we plan to experiment with many different case studies both in simulation and in hardware to show the generality of our approach and also to show how it scales to more complex tasks.

References

1. Abelson, H., Allen, D., Coore, D., Hanson, C., Homsy, G., Knight, T., Nagpal, R., Rauch, E., Sussman, G., Weiss, R.: Amorphous computing. Communications of the ACM 43(5), 74–82 (May 2000)
2. Beal, J., Dulman, S., Usbeck, K., Viroli, M., Correll, N.: Organizing the aggregate: Languages for spatial computing. In: Formal and Practical Aspects of Domain-Specific Languages: Recent Developments (IGI Global, 2012), pp. 436–501 (2012)

3. Beckers, R., Holland, O.E., Deneubourg, J.L.: From local actions to global tasks: Stigmergy and collective robotics. Artificial Life 4, 181–189 (1994)
4. Berman, S., Halasz, A., Hsieh, M., Kumar, V.: Optimized stochastic policies for task allocation in swarms of robots. IEEE Transactions on Robotics 25(4), 927–937 (2009)
5. Brutschy, A., Pini, G., Pinciroli, C., Birattari, M., Dorigo, M.: Self-organized task allocation to sequentially interdependent tasks in swarm robotics. Autonomous Agents and Multi-Agent Systems 28(1), 101–125 (2014)
6. Dorigo, M., Birattari, M., Brambilla, M.: Swarm robotics. Scholarpedia 9(1), 1463 (2014)
7. Dorigo, M., et al.: Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. IEEE Robotics & Automation Magazine p. in press (2013)
8. Dressler, F.: Self-organization in sensor and actor networks. John Wiley & Sons (2008)
9. Ferrante, E., Turgut, A.E., Duez-Guzmn, E., Dorigo, M., Wenseleers, T.: Evolution of self-organized task specialization in robot swarms. PLoS Comput Biol 11(8), 1–21 (2015)
10. Georgopoulos, A.P., Schwartz, A.B., Kettner, R.E.: Neuronal population coding of movement direction. Science 233(4771), 1416–1419 (1986)
11. Gerkey, B.P., Matarić, M.J.: A formal analysis and taxonomy of task allocation in multi-robot systems. Intl. J. of Robotics Research 23(9), 939–954 (2004)
12. Hamann, H.: Towards swarm calculus: Urn models of collective decisions and universal properties of swarm performance. Swarm Intelligence 7(2-3), 145–172 (2013)
13. Hamann, H., Valentini, G., Khaluf, Y., Dorigo, M.: Derivation of a micro-macro link for collective decision-making systems: Uncover network features based on drift measurements. In: Bartz-Beielstein, T. (ed.) 13th International Conference on Parallel Problem Solving from Nature (PPSN 2014), LNCS, vol. 8672, pp. 181–190. Springer (2014)
14. Hamann, H., Wörn, H.: A framework of space-time continuous models for algorithm design in swarm robotics. Swarm Intelligence 2(2-4), 209–239 (Oct 2008)
15. Hogg, T.: Coordinating microscopic robots in viscous fluids. Autonomous Agents and Multi-Agent Systems 14(3), 271–305 (Jun 2006)
16. Kengyel, D., Hamann, H., Zahadat, P., Radspieler, G., Wotawa, F., Schmickl, T.: Potential of heterogeneity in collective behaviors: A case study on heterogeneous swarms. In: Chen, Q., Torroni, P., Villata, S., Hsu, J., Omicini, A. (eds.) PRIMA 2015: Principles and Practice of Multi-Agent Systems, LNCS, vol. 9387, pp. 201–217. Springer (2015)
17. Lenaghan, S., Wang, Y., Xi, N., Fukuda, T., Tarn, T., Hamel, W., Zhang, M.: Grand challenges in bioengineered nanorobotics for cancer therapy. IEEE Transactions on Biomedical Engineering 60(3), 667–673 (2013)
18. Pouget, A., Dayan, P., Zemel, R.: Information processing with population codes. Nature Reviews Neuroscience 1, 125–132 (2000)
19. Prorok, A., Hsieh, M.A., Kumar, V.: Fast redistribution of a swarm of heterogeneous robots. In: International Conference on Bio-inspired Information and Communications Technologies (BICT) (2015)
20. Valentini, G., Ferrante, E., Hamann, H., Dorigo, M.: Collective decision with 100 Kilobots: Speed versus accuracy in binary discrimination problems. Autonomous Agents and Multi-Agent Systems 30(3), 553–580 (2015)
21. Valentini, G., Hamann, H., Dorigo, M.: Global-to-local design for self-organized task allocation in swarms. Tech. Rep. TR/IRIDIA/2016-002, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium (March 2016)