The definitive version was published in *Science Robotics* 3(20):eaat0430 on 18 July 2018. DOI: 10.1126/scirobotics.aat0430

Autonomous task sequencing in a robot swarm

Lorenzo Garattoni, Mauro Birattari*
IRIDIA, Université libre de Bruxelles, Belgium
*E-mail: mbiro@ulb.ac.be

Abstract: Robot swarms mimic natural systems in which collective abilities emerge from the interaction of individuals. So far, the swarm robotics literature has focused on the emergence of mechanical abilities (e.g., push a heavy object) and simple cognitive abilities (e.g., select a path between two alternatives). In this article, we present a robot swarm in which a complex cognitive ability emerges. This swarm is able to collectively sequence tasks whose order of execution is a priori unknown. As sequencing tasks is an albeit simple form of planning, the robot swarm we present provides a new perspective on one of the most pivotal debates in the history of artificial intelligence: the debate on planning in robotics. Indeed, in the proposed swarm, the two robotics paradigms—deliberative (sense-model-plan-act) and reactive (sense-act)—which are traditionally considered antithetical, coexist in a particular way: the ability to plan emerges at the collective level from the interaction of reactive individuals.

Introduction

Swarm robotics (*1–4*) takes inspiration from collective behaviors of social animals to develop multi-robot systems that, as their natural counterparts, are flexible, robust, and autonomous (*5*). A robot swarm comprises a large number of robots with limited capabilities. The interaction of the robots with each other and with the environment engenders emergent properties: collectively, the swarm displays abilities that a single robot does not possess. So far, research has focused on the emergence of geometrical/spatial properties and mechanical abilities: e.g., aggregating (*6*), covering space (*7*, *8*), forming shapes (*9*, *10*), moving coordinately (*11*), overcoming obstacles (*12*), transporting objects (*13*), clustering objects (*14*), or assembling structures (*15*). Research has been also devoted to the emergence of simple cognitive abilities: e.g., selecting an aggregation area (*16–18*), a behavior (*19*, *20*), a foraging source (*21*, *22*), or a path (*23–26*) between (typically two) alternatives. Here, we study the emergence of a more complex cognitive ability: sequencing tasks. We present TS-Swarm, a robot swarm that sequences tasks autonomously. Several studies have been already devoted to swarms that, inspired by mechanisms of division of

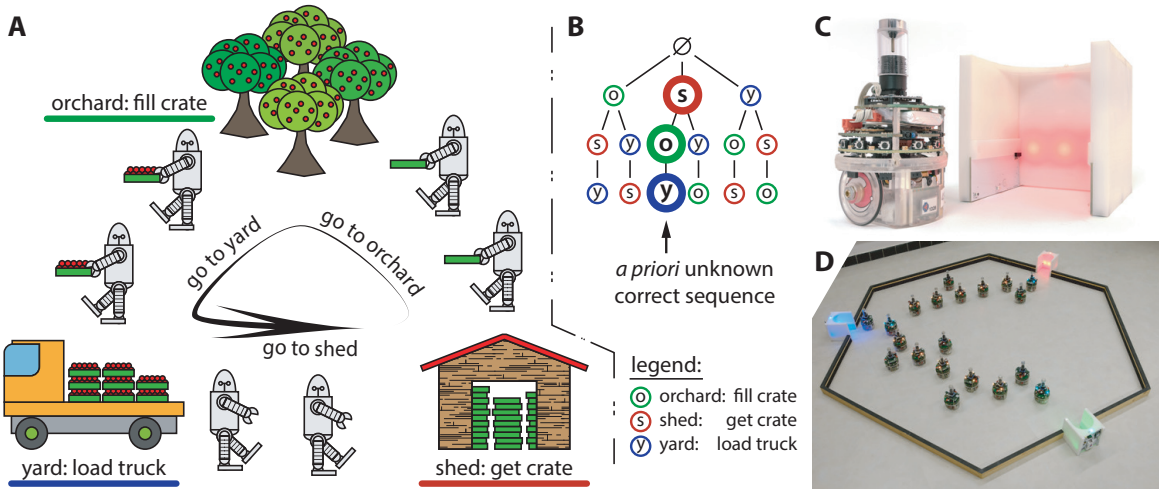


Fig. 1. From an example of task sequencing to TS-Swarm. (**A**) Task sequencing: an example. Three tasks must be performed in a specific order by an individual robot: get a crate at the shed; fill the crate at the orchard; load the crate onto the truck at the yard. The robots initially ignore the correct order of execution. They learn it collectively from successes and failures: for example, a robot faces a failure if it reaches the orchard without a crate to fill, or the truck with an empty one. The correct sequence must be repeated multiple times to fully load the truck. (**B**) Formal representation of the solution space. (**C**) An e-puck and a TAM. (**D**) TS-Swarm: the swarm in its arena, with three TAMs.

labor observed in insect societies (*27–29*), perform multiple tasks transitioning from one to another (*8*, *30–32*). Nonetheless, in all previous studies, the correct order of execution and/or the transition conditions were known at design time. The designers could thus devise and hard-code in the robots the rules that trigger the transition from task to task. Contrary to previously demonstrated swarms, TS-Swarm sequences tasks autonomously and at run time: it can therefore operate even if the correct order of execution is unknown at design time. In TS-Swarm, the two robotics paradigms—deliberative (*33*) (sense-model-plan-act) and reactive (*34*) (sense-act)—which are traditionally considered antithetical (*35*), coexist in a novel way: the ability to sequence tasks and therefore to plan a course of action emerges at the collective level from the interaction of reactive individuals.

We address the case in which m tasks must be performed in a specific order (and without repetitions) by an individual robot of the swarm. Each task must be performed in a certain area and the correct order is *a priori* unknown. The sequence of tasks must be repeated multiple times by the same or by other robots. For an illustrative example, see Fig. 1A-B.

The characterizing feature of TS-Swarm is that some of the robots position themselves to form a chain that fulfills two functions: (i) assist the navigation between the relevant areas; and (ii) identify/encode the order in which tasks must be performed. The chain enables robots with limited capabilities to accomplish a complex mission. Individually, the robots of TS-Swarm would be unable to navigate reliably from area to area or perform the tasks in the correct order. Indeed, they have a limited range of perception, are unaware of the position of the areas, and are unable to localize them-

selves in the environment. Moreover, the robots are not programmed to individually sequence tasks by reasoning symbolically on their order of execution.

Chaining has previously been adopted in swarm robotics to search the environment and assist navigation (*31*, *36–43*): by forming a chain, robots act as waypoints to route other robots. In TS-Swarm, we generalize this scenario. The chain is both a routing mechanism and a means to identify/encode a task sequence. Indeed, the robots in the chain act also as "logical waypoints" in the task space: by following them, other robots perform the tasks in the order encoded.

Results

We implemented TS-Swarm on e-puck robots and we use TAMs (task abstraction modules) to abstract tasks (Fig. 1C; see Materials and Methods). A TAM is a booth which an e-puck can enter. For an e-puck, entering a TAM amounts to performing the task it abstracts. A TAM is equipped with RGB LEDs and can display different colors. In the experiments, each task is identified by a unique color.

We developed four variants of TS-Swarm: Mark I_3, Mark I_4, Mark II_3, and Mark II_4. Mark I_3 assumes that (i) the tasks to be performed are $m = 3$; and (ii) a robot receives negative feedback as soon as it performs a task in an incorrect order, and positive feedback otherwise. A robot receives feedback in the sense that, after performing a task, it becomes immediately aware of whether the task was performed in the correct order or not—see example in Fig. 1A. In practice, as we consider abstract tasks emulated by TAMs, a failure or a success in performing a task amounts to a message that a robot receives from a TAM via an infrared signal—see Materials and Methods.

After studying Mark I_3, we modified the aforementioned assumptions to make the sequencing problem harder. Mark I_4 assumes that the tasks are $m = 4$ and indicates how a larger number of tasks can be handled. Mark II_3 assumes that a robot must perform a complete sequence before receiving any feedback on whether the sequence is correct or not. Mark II_4 assumes that the tasks are $m = 4$ and a complete sequence must be performed before receiving any feedback. Due to the lack of an immediate feedback, the problem faced by Mark II_m is combinatorial. Its computational complexity is $O(m!)$. In all four variants, the swarm operates in a bounded, convex arena surrounded by walls. The TAMs are located at the boundaries of the arena (Fig. 1D; see Materials and Methods). We opted for such a simple scenario to simplify the construction of the chain so that we could focus our attention on the collective and distributed solution of the task-sequencing problem. Thanks to the adoption of this scenario, we were able to implement the chain-based search in a way that presents only minor differences from what already described in the literature (*31*, *39*, *43*). In the following, we outline Mark I_3. We then sketch Mark I_4, Mark II_3, and Mark II_4 by highlighting their differences with respect to Mark I_3. Details are provided in the Supplementary Materials, together with an extensive discussion of the empirical analysis.

Mark I_3 and Mark I_4

In Mark I_3, all robots execute the same control software but autonomously assume different roles depending on the contingencies they encounter. A robot can be a *runner*, *guardian*, *tail*, or *link*. Initially, all robots are runners and move randomly in the arena. Upon encountering a task—more precisely the TAM that abstracts it—a runner performs it and then positions itself in its proximity, becoming its guardian. From then on, no other runner will perform the task, unless directed to do so by its guardian. Eventually, three robots are the guardians of the three tasks. Two of them received negative feedback, as their task is not the first of the sequence. The other guardian received positive feedback: its task is the first one. This guardian initiates the construction of the chain. Runners that encounter the chain being built can contribute to its extension by positioning themselves at its end, one after the other. We refer to the last robot in the chain as the tail and to the others as the links. Tail and links align and keep a target distance between themselves so that the chain is stretched and straight. If the chain reaches a wall, it turns, sweeping the environment. By extending and turning repeatedly, the chain eventually encounters another guardian. When this happens, the tail transfers its role to the guardian and becomes a link. The guardian initiates the construction of a new branch of the chain to ultimately include all the guardians. Robots that have not become chain members remain runners and navigate the environment following the chain. When a runner reaches a guardian, it performs the guarded task if so directed. Guardians learn to direct runners via trial and error. As mentioned, a guardian received positive or negative feedback, depending on whether its task is the first to be performed or not, respectively. The guardian that received positive feedback will direct to its task the runners that have not yet performed any task. The other two guardians learn the correct policy with the help of the runners. The first time they are reached by a runner that has performed exactly one task, they ask it to perform their task and wait to be informed of the outcome. If the feedback is positive, from then on they will direct to their task the runners that have performed one task. If the feedback is negative, they will direct to their task the runners that have performed two tasks. We empirically studied Mark I_3 with hardware and simulated experiments (Fig. 2A-E,F). Moreover, in simulated experiments, we assessed its scalability and robustness (Table 1 and Fig. 3). The results (Fig. 4A,B and Fig. 5A) show that Mark I_3 sequences three tasks reliably and that it operates correctly over a large range of conditions, without requiring any modification.

In Mark I_4, four tasks are sequenced thanks to a minor difference relative to Mark I_3: a single counter that counts to four rather than three. We studied Mark I_4 in simulation (Fig. 2G, Table 1, and Fig. 3). The results show that the first assumptions of Mark I_3 can be overcome (Fig. 4C, Fig. 5B): more than three tasks can be sequenced.

Mark II_3 and Mark II_4

In Mark II_3, runners must perform an entire sequence before receiving any feedback. Due to the lack of immediate feedback, which in Mark I_3 breaks the initial symmetry, all guardians initiate the construction of a branch of the chain immediately after

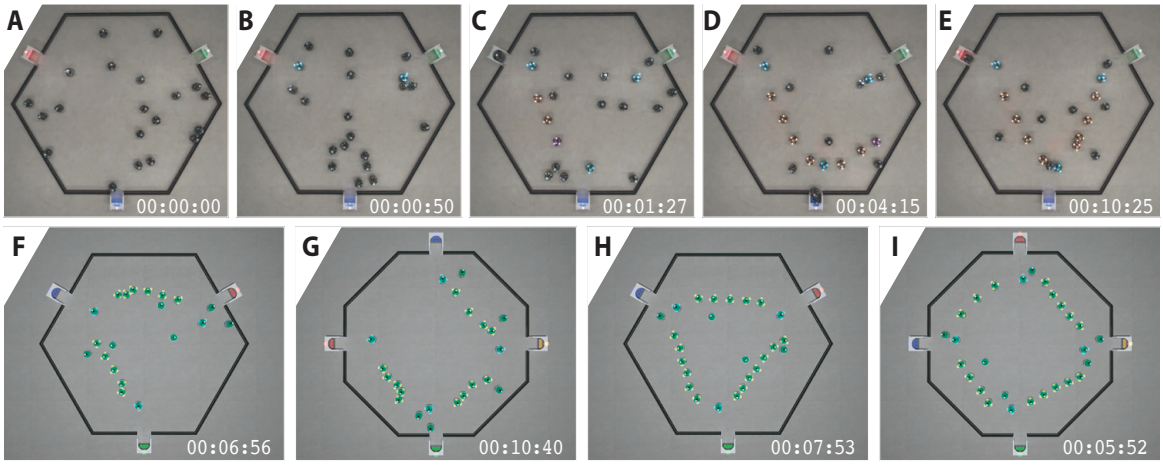


Fig. 2. Overhead snapshots. (**A-E**) Mark I_3, robot experiments (movie S1). (**F**) Mark I_3, simulation (movie S2, side-by-side with a run on the robots). (**G**) Mark I_4, simulation (movie S4). (**H**) Mark II_3, simulation (movie S5). (**I**) Mark II_4, simulation (movie S6).

Table 1. Parameters of the scalability and robustness studies. We report the parameters that characterize each experimental setting in which each variant of TS-Swarm is studied. The scalability study is performed using the default number of robots in each setting. Between one setting and the following one, we double the surface of the arena in which the robots operate (see Materials and Methods). The robustness study is performed varying the number of robots between -20% and +100% with respect to the default number of each setting.

	setting	arena's side (m)	arena's area (m^2)	number of robots								time cap (s)
				-20%	-10%	default	+20%	+40%	+60%	+80%	+100%	
Mark I_3	0	0.90	2.10	16	18	20	24	28	32	36	40	2400
	1	1.27	4.21	23	25	28	34	39	45	50	56	3400
	2	1.80	8.42	32	36	40	48	56	64	72	80	4800
	3	2.55	16.84	45	51	57	68	80	91	103	114	6800
	4	3.60	33.67	64	72	80	96	112	128	144	160	9600
Mark I_4	0	0.66	2.10	18	20	22	26	31	35	40	44	100000
	1	0.93	4.21	25	28	31	37	43	50	56	62	100000
	2	1.32	8.42	35	40	44	53	62	70	79	88	100000
	3	1.87	16.84	50	56	62	74	87	99	112	124	100000
	4	2.64	33.67	70	79	88	106	123	141	158	176	100000
Mark II_3	0	0.90	2.10	20	22	25	30	35	40	45	50	100000
	1	1.27	4.21	28	31	35	42	49	56	63	70	100000
	2	1.80	8.42	40	45	50	60	70	80	90	100	100000
	3	2.55	16.84	56	63	70	84	98	112	126	140	100000
	4	3.60	33.67	80	90	100	120	140	160	180	200	100000
Mark II_4	0	0.66	2.10	22	24	27	32	38	43	49	54	100000
	1	0.93	4.21	30	34	38	46	53	61	68	76	100000
	2	1.32	8.42	43	49	54	65	76	86	97	108	100000
	3	1.87	16.84	61	68	76	91	106	122	137	152	100000
	4	2.64	33.67	86	97	108	130	151	173	194	216	100000

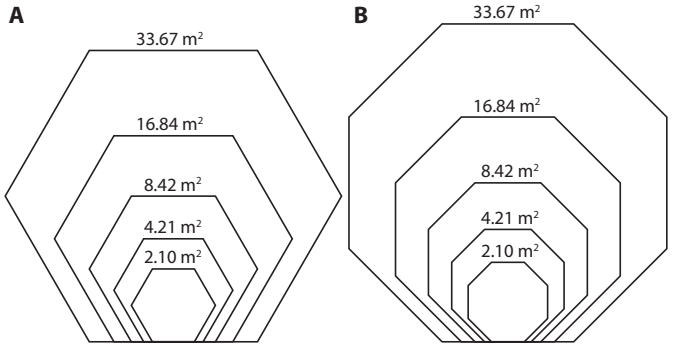


Fig. 3. Scalability and robustness analysis, the arenas. Shape and size of the arenas considered for the scalability and robustness study of (**A**) Mark I_3 and Mark II_3 and (**B**) Mark I_4 and Mark II_4.

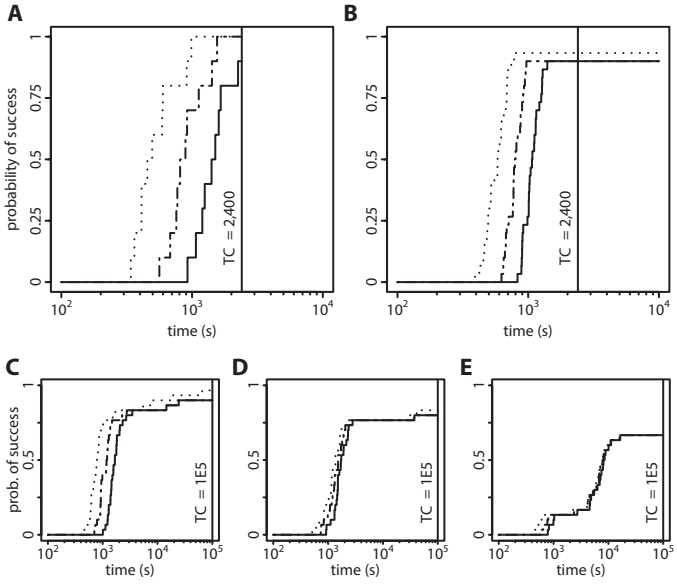


Fig. 4. Empirical assessment. Empirical run-time distribution for the execution of one (dotted), five (dot-dash), and ten (solid) sequences. (**A**) Mark I_3, robot experiments. (**B**) Mark I_3, simulation. (**C**) Mark I_4, simulation. (**D**) Mark II_3, simulation. (**E**) Mark II_4, simulation.

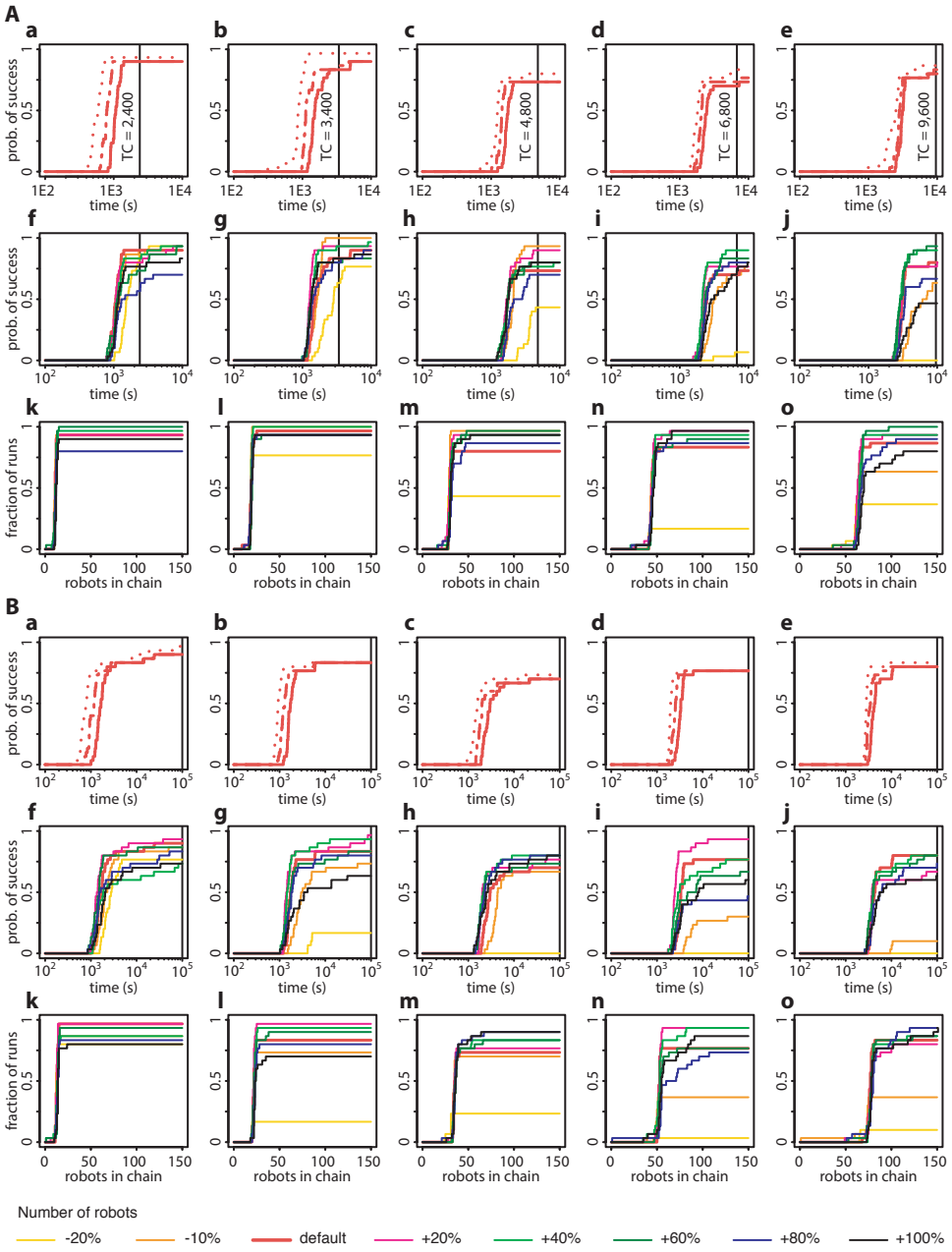


Fig. 5. Scalability and robustness of Mark I_3 and Mark I_4. (**A**) Mark I_3. (**B**) Mark I_4. (**a-e**) Scalability study using the default number of robots in five arenas of different size (see Table 1 and Materials and Methods). Empirical run-time distribution for the execution of one (dotted), five (dot-dash), and ten (solid) sequences. (**f-j**) Robustness to the variation of the number of robots between -20% and +100% of the default number (see Table 1 and Materials and Methods). Empirical run-time distribution for the execution of ten sequences. (**k-o**) Empirical distribution of the number of robots in the chain as a function of the total number of robots. Arena's area: (**a, f, k**) 2.10 m^2; (**b, g, l**) 4.21 m^2; (**c, h, m**) 8.42 m^2; (**d, i, n**) 16.84 m^2; (**e, j, o**) 33.67 m^2.

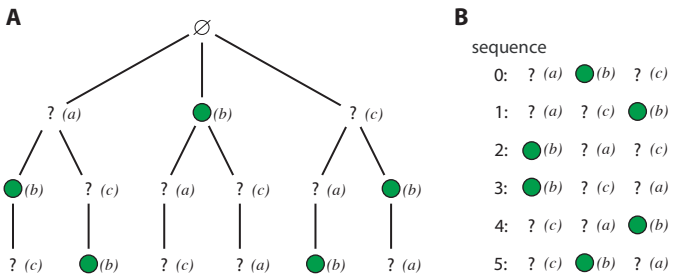


Fig. 6. Exploration of the sequence space in Mark II$_3$, as seen by the guardian of the green task. In this example, the green task is the second of the initial sequence. Its guardian ignores the colors of the first and last tasks; it only knows that its label is b and therefore its task is the second. More precisely, this guardian is in the state in which it directs to its task the runners that have performed exactly one task. This guardian (as the others) directs the runners throughout the search process without knowing what is the sequence that is tested at each step. At the first step its task is the second—it directs to its task runners that have performed exactly one task. At the following step, its task is the third—it directs to its task runners that have performed exactly two tasks. Then, its task is the first—it directs to its task runners that have not yet performed any task; and so on. (**A**) Permutation tree generated by the guardian of the green task on the basis of its partial knowledge of the initial sequence. (**B**) Sequences explored through depth-first search of the permutation tree.

assuming their role. Upon completion, the chain is a closed loop that, besides routing runners as Mark I$_3$'s chain, has the additional function of relaying information. By exchanging messages via the chain, the guardians (i) establish an initial sequence, out of which they generate a permutation tree spanning all possible sequences; and (ii) direct the runners to collectively explore such tree via depth-first search. The guardians establish an initial sequence by ordering themselves via a leader-election algorithm (*44*). Each guardian communicates its unique ID that is relayed by the chain. The guardian with the largest ID takes the label c and sends a message that is relayed clockwise along the closed-loop chain. The message contains the label b. The first guardian that receives the message takes the label b and propagates label a which is eventually taken by the last guardian. Each guardian generates the tree of the permutations of $\langle a, b, c\rangle$. The tree is then collectively explored by the swarm via depth-first search. As a first step, the guardians address the runners to the tasks guarded by a, b, and c, in this order; as a second step, to the tasks guarded by a, c, and b; as a third step to the tasks guarded by b, a, and c, and so on. A failure reported by a runner after completing a sequence triggers the transition to the following one. On the other hand, a success indicates that the correct sequence has been identified. The exploration of the permutation tree is distributed. Throughout the process, all robots act reactively (sense-act) and each guardian has only partial knowledge about the sequence being tested (see Fig. 6 and Supplementary Materials).

In Mark II$_4$, four tasks are sequenced under the assumption that no immediate feedback is received after task execution; the only difference relative to Mark II$_3$ is a counter that counts to four rather than three.

We studied Mark II$_3$ and Mark II$_4$ in simulation (Fig. 2H-I, Table 1, and Fig. 3). The results show that the two assumptions of Mark I$_3$ can be overcome (Fig. 4D-E

and Fig. 7): the task sequencing problem can be solved even if no immediate feedback is received by the robots and the tasks are more than three.

Discussion

As sequencing tasks is an albeit simple form of planning, TS-Swarm provides a new perspective on one of the most pivotal debates in the history of artificial intelligence: the debate on planning in robotics. This debate opposes two competing, antithetical paradigms: the deliberative and reactive (*35*). According to the former, an intelligent robot should necessarily plan a course of action by reasoning on a model (*33*). According to the latter, a robot is more effective in dealing with reality by simply reacting to contingencies, without relying on reasoning and representation (*34*). Although hybrid systems have been proposed, they conceptually juxtapose the two paradigms: deliberative and reactive instances—operating sequentially or in parallel—interact but remain logically distinct (*45*, *46*). By contrast, TS-Swarm associates the two paradigms in a novel way: the ability to plan a sequence of tasks emerges at the collective level from the interaction of robots that, at the individual level, behave reactively without relying on reasoning and representation.

Relations with multi-robot/agent learning

The learning process performed by TS-Swarm bears some resemblance to others described in the multi-robot and multi-agent literature (*47–49*). TS-Swarm learns the correct sequence based on binary rewards: failures and successes experienced after performing tasks. No example of correct behavior is provided to the robots. The learning process performed by TS-Swarm can be therefore classified as reinforcement learning (*50*, *51*). More precisely, as no value function (*52*, *53*) is explicitly learned, the learning process of TS-Swarm could be seen as a form of direct policy search (*54–57*). In Mark I_m, feedback is received immediately after the execution of each single task. On the other hand, in Mark II_m feedback is delayed and is received only after the execution of a complete sequence. As a result, the sequencing problem presents a combinatorial nature: the resulting learning process is much more challenging. The robots learn collectively the correct sequence and the path to reach the areas where the task must be performed. A single learning process takes place, as opposed to collective systems in which each agent/robot individually learns a behavior. In this sense, we can qualify the learning process of TS-Swarm as team learning (*58*, *59*). More precisely, as the behavior that is collectively learned is the same for all robots—the behavior that each robot (runner) must execute to perform the same correct sequence—the learning process can be qualified as a form of homogeneous team learning (*47–49*). Nonetheless, TS-Swarm differs from typical team learning systems (*60–62*) in the fact that the single learning entity is indeed the swarm as a whole, which has an immaterial and distributed nature: the robots operate in an independent manner and no central entity exists that performs the learning process having a global view of the state of the system. Learning takes place at the collective level of the swarm: it is the swarm as a whole that searches the space of possible solutions. Moreover, once the correct solution is identified, the policy to produce is

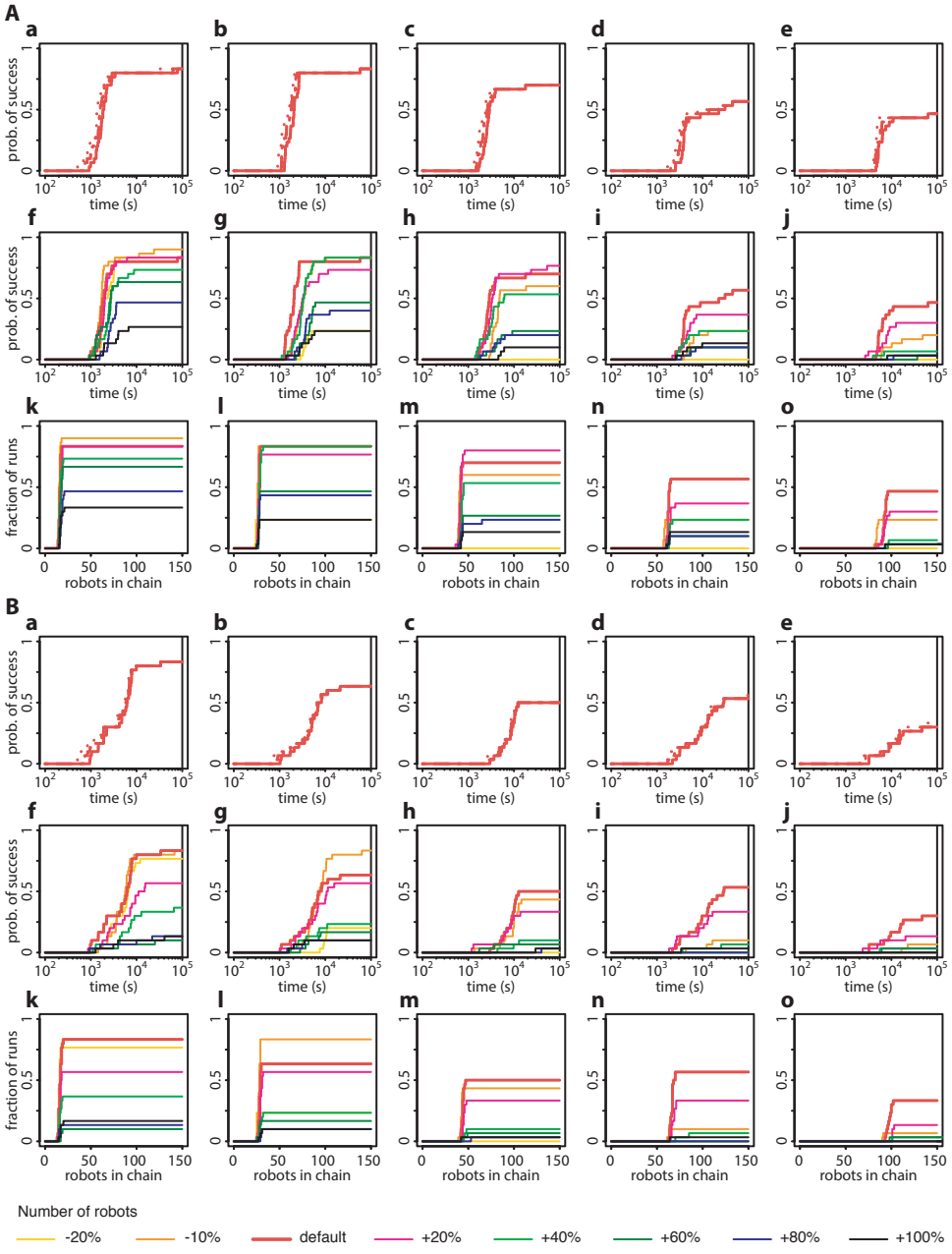


Fig. 7. Scalability and robustness of Mark II$_3$ and Mark II$_4$. (**A**) Mark II$_3$. (**B**) Mark II$_4$. See caption of Fig. 5.

eventually encoded by the chain in a collective and distributed way: each guardian stores the part of policy that concerns the execution of its guarded task. Each runner implements the policy encoded by the chain on the basis of its own state, which is defined by the number of tasks performed and by which guardian is in its proximity, if any.

Limitations and possible improvements

Transmission of robot IDs limits scalability. The scalability of TS-Swarm is limited by the fact that robots include their identifier in the range-and-bearing messages they broadcast (see Materials and Methods and Supplementary Materials). Also, in Mark II_m, guardians use their identifier in the leader-election process.

Possible improvement: we could adopt locally-unique identifiers, which have been successfully demonstrated with a swarm of one thousand robots (*9*).

The number of tasks must be known at design time. All variants of TS-Swarm, assume that the number of tasks to be sequenced is known at design time.

Possible improvement: we could let the swarm determine the number of tasks autonomously at run time. This would be straightforward in Mark II_m: when the closed-loop chain is established and the guardians order themselves using a leader-election algorithm, the information on the number of tasks discovered by the swarm in the environment is readily available to all the guardians.

Chains might overstep guardians. In some cases, the branch of chain being built fails to locate the guardian it is supposed to reach. This may be due to several factors, including (i) the lack of a sufficient number of robots to extend the branch up to the guardian; (ii) a temporary fault in the omnivision module of the tail. These factors cause the branch to overstep the guardian and eventually reach another branch of chain or the guardian from which the branch itself originates. As a result, the branch being built merges with another branch or collapses on itself. In both cases, the system fails.

Possible improvement: the tail could implement a mechanism to detect whether it is approaching another branch of chain, or the originating guardian of its own branch. Should this happen, the tail could invert the sense in which the branch sweeps around its originating guardian. By alternating clockwise and anticlockwise sweeps, the branch being built would explore the environment more effectively and increase the chance to spot the guardian it is supposed to reach.

Robots' movement is unsophisticated. Chain members and runners move in a simple and unsophisticated way. For simplicity, we implemented the links so that they stop moving upon being notified that the branch they form is established. If a runner bumps into a link pushing and it away from the correct position, the continuity of the chain is broken and the functionality of the whole system is compromised. This is more likely to occur when the swarm is comprised by a large number of robots, the arena is large, and no immediate feedback is received by the robots (Mark II_m). In these cases, the branches are long and need to be functional for a long time in order to support the exploration of a large solution space.

Possible improvement: More refined movement mechanisms could be implemented in order to make the motion of chain members and runners more precise and reliable. Links could keep adjusting alignment and spacing even after their branch is established. In particular, they could benefit from a mechanism to regain their original position, should they detect that the functionality of their branch is compromised.

Chaining rests on restrictive assumptions. The chaining behavior works under the assumptions that (i) the arena is convex, (ii) the tasks are located along its perimeter, and (iii) there are no obstacles.

Possible improvement: we could relax these assumptions if the path between guardians were obtained by first covering the space with a lattice-like formation, and then selecting the shortest path on this lattice. Robots that are not on the shortest path could leave; those that are on the shortest path would remain to act as waypoints. An approach to select the shortest path on a lattice has been demonstrated with a swarm of e-pucks (*63*). This approach is based on artificial pheromone.

The search strategy in Mark II$_m$ is sub-optimal. The transition from a candidate sequence to the following one in the permutation tree (Fig. 6) causes all the runners to abort the execution of the sequence being tested and start the execution of the following one from scratch. However, if the sequence being performed and the following one share an identical initial sub-sequence, this solution is not optimal.

Possible improvement: we could implement a sort of backtracking for a more efficient exploration of the permutation tree. Indeed, the runners that have performed only tasks contained in the identical initial sub-sequence could continue the execution of the remaining tasks of the following sequence, without starting from scratch.

Materials and Methods

The e-puck

The e-puck is a mobile two-wheeled differential-drive robot designed for education and research (*64*). It is cylindrical in shape, with a diameter of 70 mm and a height of 50 mm. Its basic version is equipped with a PIC microcontroller and several sensors and actuators. The sensors are: 8 infra-red transceivers, which can be used to sense the presence of obstacles or measure the intensity of ambient light; a color camera at the front of the robot; a microphone; and a 3-axis accelerometer. The actuators are: two stepper motors, which control the motion of the robot by differential steering (one motor for the left wheel and one for the right wheel); a ring of 8 red LEDs; and a speaker.

The e-puck can be enhanced by the addition of various extension boards. For the research presented here, we extended the basic version of the e-puck with a range-and-bearing board (*65*); an omnivision module; and a Gumstix Overo board (Fig. 1C, left). The range-and-bearing board enables local communication between e-pucks via infra-red signals. It comprises 12 emitters and 12 receivers placed all around the body of the e-puck. The range-and-bearing board allows e-pucks to send and receive four-byte messages at a rate of about 30 messages per second. Upon reception of a

message, the board computes the distance (range) and angle (bearing) of the peer e-puck that sent the message. The omnivision module comprises an omni-directional camera and 3 RGB LEDs and enhances the perception and local communication capabilities of the e-puck. Through the camera, an e-puck can see its neighboring peers and the TAMs. Moreover, it can perceive the color coded status that the neighboring peers might display using their RGB LEDs. The Gumstix Overo board increases the computational capabilities of the e-puck and provides the flexibility and potential of a computer running Linux. It allows running C++ code, which is not possible on the PIC micro-controller of the basic version of the e-puck.

The basic version of the e-puck is powered by a rechargeable lithium-ion battery with 5 W h capacity. The omnivision module houses a second battery with the same capacity to cope with the higher energy requirements of the extended e-puck. In a typical experiment, the full battery charge of an extended e-puck lasts about 40 minutes. Indeed, we have observed that after about 45 minutes of continuous operation, the charge of the batteries is low. This negatively affects the behavior of the robots and in particular their ability to successfully transmit and receive messages through the range-and-bearing board.

The TAM

The TAM (*66*), task abstraction module, is a device conceived for facilitating laboratory experiments with e-puck robots. A TAM represents an abstract task to be performed by an e-puck. The goal of the TAM is to abstract from task-specific details that are irrelevant to the objectives of an experiment. The TAM is particularly useful in experiments that focus on group dynamics rather than on the specific tasks performed by the individuals.

The TAM is a booth which an e-puck can enter. For an e-puck, being into a TAM for a given time span amounts to performing the task abstracted by the TAM itself. The TAM has a cubical shape with sides of 120 mm (Fig. 1C, right). The TAM is controlled by a microcontroller (ATmega-1284p, 16 MHz), and is equipped with two light barriers, three RGB LEDs, and an IR transceiver for short-range communication. Each TAM is powered by a rechargeable lithium-ion battery with 5 W h capacity, the same battery used by the e-puck. In a typical experiment, a full battery charge lasts for over 10 hours. The TAM is equipped also with an XBee mesh networking module that allows the synchronization of multiple TAMs. A group of TAMs can be therefore programmed to represent complex relationships between tasks. For example, a task could become activated only upon completion of another one or a group of tasks could be performed successfully only in a specific order. The experimenter implements the logic that defines the relationship between tasks on a central computer. The computer dispatches commands to the TAMs to realize the relationships programmed by the experimenter. The TAMs and the central computer communicate wirelessly via the XBee mesh networking module.

An e-puck perceives the colored LEDs of the TAM using its omni-directional camera. Different tasks are signaled by using different LED colors. An e-puck can decide to perform the task represented by a TAM by moving into it. The TAM detects the presence of the e-puck by means of its light barriers and reacts according to a logic defined by the experimenter. For example, upon the detection of an e-puck,

the TAM could change the color of its LEDs or start communicating with the e-puck itself. The TAM and the e-puck communicate with each other through their infrared transceivers. Communication between e-pucks and TAMs enables experiments in which e-pucks receive individual feedback for the tasks they perform.

ARGoS

ARGoS (*67*) is a modular multi-robot simulator and development environment conceived for being flexible and efficient. ARGoS provides a straightforward way to port control software developed in simulation to the robots, without requiring any modification. To achieve this result, each sensor and actuator presents an interface with two back-end implementations: one for simulation and one for the robot. The control software of the robot directly interacts with this interface without having knowledge of which back-end implementation is being used. At link time, ARGoS takes care that the appropriate back-end implementation is executed, depending on whether the execution is to take place in simulation or on the robot. ARGoS provides a number of physics engines. Some of them are kinematic engines that favor performance over realism, others are dynamics engines, in two or three dimensions, that require more computation but that produce more realistic simulations. As realism plays an important role in our simulations and the system we propose comprises only robots that move on the ground, we use a dynamics engine in two dimensions in all the simulated experiments.

We used ARGoS to develop control software for, and to simulate e-pucks and TAMs. We extended the basic model of the e-puck that was originally provided in ARGoS by implementing models of the range-and-bearing board, the omnivision module, and the Linux board (*68*). We also created the model of the TAM, which was not originally provided by ARGoS (*68*).

Experimental design

The goal of the experiments we present here is to demonstrate TS-Swarm and provide evidence that it is able to successfully sequence tasks in an autonomous and distributed way. First, we demonstrate Mark I_3 both in reality with a swarm of 20 e-puck robots and in simulation. Besides showing the effectiveness of Mark I_3, this first experiment also provides an assessment of the simulator. After having shown that the simulator satisfactorily predicts the behavior of TS-Swarm on the e-puck robots, we adopt the simulator to perform a number of studies that we would be unable to perform with real robots. These studies either involve a large number of robots (more than those that we have available) or last longer than the battery life of the robots. In particular, we perform a study in which we assess the scalability of Mark I_3 by running experiments in which the number of robots ranges from 20 to 80 and the surface of the arena in which they operate ranges from $2.10\,\mathrm{m}^2$ to $33.67\,\mathrm{m}^2$. We also perform a study in which we assess the robustness of Mark I_3 to the number of robots comprised in the swarm (Table 1 and Fig. 3). Finally, we perform three studies to demonstrate Mark I_4, Mark II_3, and Mark II_4. In these studies, each run of the system lasts 100,000 s (i.e., about 28 h), which is much longer than the battery life of the e-puck robot. Also for these three variants, we study their scalability and robustness (Table 1 and Fig. 3).

The focus of the research presented here is on how a swarm can sequence tasks in an autonomous and distributed way, rather than on the specific tasks that it should sequence. For this reason, in these experiments we consider abstract tasks represented by TAMs. Robots operate in a bounded arena delimited by walls, which are 42 mm high. The arena is a regular hexagon when the tasks to be sequenced are three and a regular octagon when the tasks are four. The TAMs abstracting the tasks are distributed along the perimeter of the arena and are positiond in the middle of alternate sides. Each task is associated with a color. When the tasks are three, the colors are red (R), green (G), and blue (B). When they are four, the fourth color is orange (O). In each experimental run, the initial position of the robots, the correct sequence, and the relative position of the tasks are decided randomly. In all the supplementary movies, for clarity, the correct sequence is always RGB when the tasks are three and RGBO when they are four.

In all experiments, the final goal of TS-Swarm is to perform the correct sequence of tasks ten times within a given time cap. As a performance measure, we consider the time required to complete one, five, and ten executions of the correct sequence. The first execution indicates that TS-Swarm has been able to solve the task-sequencing problem. The tenth execution determines the final success of the system and therefore the end of the experiment. The fifth execution represents the mid point of the previous two measures and provides visual information on whether the execution time grows linearly with the number of correct sequences performed or not.

Statistics

We report the performance of TS-Swarm via its empirical run-time distribution. Given: (a) one of the four variants of TS-Swarm—i.e., Mark I_3, Mark I_4, Mark II_3, or Mark II_4; (b) a specific experimental setting—e.g., a setting characterized by the number of robots, the surface of the arena, and the time cap; and (c) a target objective—i.e., the execution of one, five, or ten correct sequences, we perform k independent runs and we observe, for each run, the time required to attain the target objective. The empirical run-time distribution is the empirical distribution of these observations.

Formally, let TC be the time cap of each run, $j \in \{1, \ldots, k\}$ be the index of a run, r_j be the run-time of run j, and $k' \leq k$ be the number of successful runs, that is, those runs $j : r_j < \mathrm{TC}$. The empirical run-time distribution is defined as $RTD(t) = \hat{P}_s(\tau \leq t) = \#\{j \mid r_j \leq t\}/k$. Here, $\hat{P}_s(\tau \leq t)$ is an estimate of the probability that the system attains its target objective in an amount of time τ that is less than or equal to t. In other words, the empirical distribution $RTD(t) = \hat{P}_s(\tau \leq t)$ is an estimate of the probability of success of the system over time (up to TC). For a given target objective and in a given experimental setting, the success ratio of the system within the time cap TC is $S_{\mathrm{TC}} = k'/k$.

Experiments with Mark I_3

To complete a sequence, a robot must perform three tasks in a specific order, which is *a priori* unknown. Upon the execution of each task, the robot immediately receives feedback—a success, if it has performed the task in the right order; a failure, otherwise.

In case of failure, the robot must restart the execution of the sequence from the beginning.

Robot experiments: We run Mark I_3 10 times with 20 e-pucks. The experiments are performed in a controlled environment with a flat surface and uniform light conditions. The arena where the robots operate is a regular hexagon with sides of 0.9 m. A camera operating at about 3 frames per second is mounted on the ceiling with its axis lying on the vertical line passing through the center of the arena. We present the results of 10 consecutive runs. The performance of Mark I_3 in each of these 10 runs concurs to the statistics presented: no observed result is discarded for any reason whatsoever. Once a run starts, it is accounted for in the statistics. The statistics include therefore also the failures. In Table S3 we report the lab notebook, which includes the record of all the information that we collected during each of the 10 runs. A run is terminated either at the tenth execution of the correct sequence or at a time cap of 40 minutes (2,400 s). Results are reported in Fig. 4A.

Assessment of the simulator: Alongside the experiments with the robots, we perform similar experiments in simulation using ARGoS, with the idea of producing an assessment of the simulation environment. The control software used in the two sets of experiments is the same: after performing the robot experiments, we port the control software back to the simulated environment without any modification. Because performing experiments in the simulated environment is much less time consuming than performing them in reality, we gather results on 30 simulated runs. Moreover, because battery life is not a concern in simulation, we extended the duration of runs beyond the time cap of 40 minutes. Results are reported in Fig. 4B.

Scalability study: We perform simulated experiments in five experimental settings. In each setting, we double the surface of the arena with respect to the previous one. We also increase the number of robots by a factor of $\sqrt{2}$. The rationale is that, by increasing the surface of the arena by a factor 2, the distance between the TAMs increases by a factor $\sqrt{2}$. We therefore expect the number of robots that become chain members to grow roughly by the same factor. By increasing the swarm size by a factor $\sqrt{2}$, we expect that the robots will be sufficiently many to connect all the TAMs. The control software adopted in the scalability study is exactly the same in all the settings. The parameters that characterize the five settings are given in Table 1. We ran Mark I_3 30 times in each of the five settings (Fig. 3). Results are reported in Fig. 5A(a-e).

Robustness study: We use the same five experimental settings considered in the scalability study. For each of the five settings, we vary the number of robots with respect to the one adopted in the scalability study. We consider both a smaller (-10% and -20%) and a larger number of robots ($+20\%$, $+40\%$, $+60\%$, $+80\%$, and $+100\%$). For each experimental setting and each number of robots tested, we report the run time distribution for the successful execution of ten sequences and the empirical distribution of the number of robots in the chain. We ran Mark I_3 30 times

for each number of robots considered in each of the five settings (Table 1 and Fig. 3). Results are reported in Fig. 5A(f-o).

Experiments with Mark I_4

To complete a sequence, a robot must perform four tasks in a specific order, which is *a priori* unknown. The arena is a regular octagon with sides of 0.66 m. To connect the four TAMs, Mark I_4 needs to establish three branches of chain: one more than the two that Mark I_3 needs to establish. For this reason, we consider here a swarm of 22 robots, rather than 20 as we did for Mark I_3. We also increase the time cap to 100,000 s (i.e., about 28 h). We run Mark I_4 30 times in simulation. Finally, we study the scalability and the robustness of Mark I_4 (Table 1 and Fig. 3). Results are reported in Fig. 4C and Fig. 5B.

Experiments with Mark II_3

We consider a scenario in which a robot needs to complete an entire sequence of tasks before being notified of a possible error. The tasks to be sequenced are three. The correct sequence is *a priori* unknown. The arena is the same of the experiments with Mark I_3: a regular hexagon with sides of 0.9 m. As Mark II_3 needs to build a closed-loop chain, the number of robots it requires is larger than the one required by Mark I_3. We consider here a swarm of 25 robots rather than the 20 of the experiments performed with Mark I_3. As Mark II_3 must explore a relatively large space of solutions, we increase the time cap to 100,000 s. We run Mark II_3 30 times in simulation. Finally, we study the scalability and the robustness of Mark II_3 (Table 1 and Fig. 3). Results are reported in Fig. 4D and Fig. 7A.

Experiments with Mark II_4

We consider a scenario similar to the one considered for Mark II_3, with the only difference that the tasks to be sequenced are four. The arena is the same of the experiments with Mark I_4: a regular octagon with sides of 0.66 m. We consider here a swarm of 27 robots: more than those considered in the experiments with Mark I_4 because Mark II_4 needs to build a closed-loop chain. Also in this case, the time cap is at 100,000 s. We run Mark II_4 30 times in simulation. Finally, we study the scalability and the robustness of Mark II_4 (Table 1 and Fig. 3). Results are reported in Fig. 4E and Fig. 7B.

References

1. G. Beni, From swarm intelligence to swarm robotics, in *Swarm Robotics* (Springer, Berlin, Germany, 2005), LNCS vol. 3342, pp. 1–9.
2. E. Şahin, Swarm robotics: from sources of inspiration to domains of application, in *Swarm Robotics* (Springer, Berlin, Germany, 2005), LNCS vol. 3342, pp. 10–20.
3. M. Dorigo, M. Birattari, M. Brambilla, Swarm robotics. *Scholarpedia* **9**, 1463 (2014).
4. G.-Z. Yang, J. Bellingham, P. E. Dupont, P. Fischer, L. Floridi, R. Full, N. Jacobstein, V. Kumar, M. McNutt, R. Merrifield, B. J. Nelson, B. Scassellati, M. Taddeo, R. Taylor, M. Veloso, Z. L. Wang, R. Wood, The grand challenges of Science Robotics. *Sci. Robot.* **3** (2018).
5. S. Camazine, J.-L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, E. Bonabeau, *Self-organization in Biological Systems* (Princeton Univ. Press, Princeton, NJ, 2001).
6. M. Gauci, J. Chen, W. Li, T. Dodd, R. Groß, Self-organized aggregation without computation. *Int. J. Robot. Res.* **33**, 1145–1161 (2014).
7. M. Schwager, J. McLurkin, D. Rus, Distributed coverage control with sensory feedback for networked robots, in *Robotics: Science and Systems, Proceedings* (MIT Press, Cambridge, MA, 2006), p. 007.
8. M. Duarte, V. Costa, J. Gomes, T. Rodrigues, F. Silva, S. M. Oliveira, A. L. Christensen, Evolution of collective behaviors for a real swarm of aquatic surface robots. *PLOS ONE* **11**, 1–25 (2016).
9. M. Rubenstein, A. Cornejo, R. Nagpal, Programmable self-assembly in a thousand-robot swarm. *Science* **345**, 795–799 (2014).
10. N. Mathews, A. Christensen, R. O'Grady, F. Mondada, M. Dorigo, Mergeable nervous systems for robots. *Nat. Commun.* **8**, 439 (2017).
11. C. Virágh, G. Vásárhelyi, N. Tarcai, T. Szörényi, G. Somorjai, T. Nepusz, T. Vicsek, Flocking algorithm for autonomous flying robots. *Bioinspiration Biomim.* **9**, 025012 (2014).
12. R. O'Grady, R. Groß, A. L. Christensen, M. Dorigo, Self-assembly strategies in a group of autonomous mobile robots. *Auton. Robot.* **28**, 439–455 (2010).
13. M. Rubenstein, A. Cabrera, J. Werfel, G. Habibi, J. McLurkin, R. Nagpal, Collective transport of complex objects by simple robots: theory and experiments, in *AAMAS 2013, Proceedings* (IFAAMAS, Richland, SC, 2013), pp. 47–54.
14. M. Gauci, J. Chen, W. Li, T. Dodd, R Groß, Clustering objects with robots that do not compute, in *AAMAS 2014, Proceedings* (IFAAMAS, Richland, SC, 2014), pp. 421–428.
15. J. Werfel, K. Petersen, R. Nagpal, Designing collective behavior in a termite-inspired robot construction team. *Science* **343**, 754–758 (2014).

16. J. Halloy, G. Sempo, G. Caprari, C. Rivault, M. Asadpour, F. Tâche, I. Saïd, V. Durier, S. Canonge, J. M. Amé, C. Detrain, N. Correll, A. Martinoli, F. Mondada, R. Siegwart, J. L. Deneubourg, Social integration of robots into groups of cockroaches to control self-organized choices. *Science* **318**, 1155–1158 (2007).

17. S. Garnier, J. Gautrais, M. Asadpour, C. Jost, G. Theraulaz, Self-organized aggregation triggers collective decision making in a group of cockroach-like robots. *Adapt. Behav.* **17**, 109–133 (2009).

18. A. Ozdemir, M. Gauci, S. Bonnet, R. Groß, Finding consensus without computation. *IEEE Robotic. Autom. Lett.* **3**, 1346–1353 (2018).

19. G. Pini, A. Brutschy, M. Frison, A. Roli, M. Dorigo, M. Birattari, Task partitioning in swarms of robots: an adaptive method for strategy selection. *Swarm Intell.* **5**, 283–304 (2011).

20. E. Castello, T. Yamamoto, W. Liu, A. F. Winfield, Y. Nakamura, H. Ishiguro, Adaptive foraging for simulated and real robotic swarms: the dynamical response threshold approach. *Swarm Intell.* **10**, 1–31 (2016).

21. Á. Gutiérrez, A. Campo, F. Monasterio-Huelin, L. Magdalena, M. Dorigo, Collective decision-making based on social odometry. *Neural Comput. Appl.* **19**, 807–823 (2010).

22. G. Valentini, E. Ferrante, H. Hamann, M. Dorigo, Collective decision with 100 kilobots: speed versus accuracy in binary discrimination problems. *Auton. Agents Multi Agent Syst.* **30**, 553–580 (2016).

23. T. Schmickl, K. Crailsheim, Trophallaxis within a robotic swarm: bio-inspired communication among robots in a swarm. *Auton. Robot.* **25**, 171–188 (2008).

24. M. A. Montes de Oca, E. Ferrante, A. Scheidler, C. Pinciroli, M. Birattari, M. Dorigo, Majority-rule opinion dynamics with differential latency: a mechanism for self-organized collective decision-making. *Swarm Intell.* **5**, 305–327 (2011).

25. A. Reina, G. Valentini, C. Fernández-Oto, M. Dorigo, V. Trianni, A design pattern for decentralised decision making. *PLoS ONE* **10**, e0140950 (2015).

26. A. Scheidler, A. Brutschy, E. Ferrante, M. Dorigo, The k-unanimity rule for self-organized decision making in swarms of robots. *IEEE Trans. Syst. Man Cybern.* **46**, 1175–1188 (2016).

27. E. O. Wilson, Caste and division of labor in leaf-cutter ants (Hymenoptera: Formicidae: Atta). *Behav. Ecol. Sociobiol.* **7**, 143–156 (1980).

28. T. D. Seeley, *The Wisdom of the Hive* (Harvard Univ. Press, Cambridge, MA, 1996).

29. E. Bonabeau, G. Theraulaz, J.-L. Deneubourg, Fixed response thresholds and the regulation of division of labor in insect societies. *Bull. Math. Biol.* **60**, 753–807 (1998).

30. M. J. Krieger, J. B. Billeter, L. Keller, Ant-like task allocation and recruitment in cooperative robots. *Nature* **406**, 992–995 (2000).

31. S. Nouyan, R. Groß, M. Bonani, F. Mondada, M. Dorigo, Teamwork in self-organized robot colonies. *IEEE T. Evolut. Comput.* **13**, 695–711 (2009).

32. T. Schmickl, R. Thenius, C. Moslinger, J. Timmis, A. Tyrrell, M. Read, J. Hilder, J. Halloy, A. Campo, C. Stefanini, L. Manfredi, S. Orofino, S. Kernbach, T. Dipper, D. Sutantyo, CoCoRo – The self-aware underwater swarm, in *SASOW 2011, Proceedings* (IEEE Press, Piscataway, NJ, 2011), pp. 120–126.

33. N. J. Nilsson, "Shakey the robot", tech. rep. 323 (SRI AI Center, Menlo Park, CA, 1984).

34. R. A. Brooks, Intelligence without representation. *Artif. Intell.* **47**, 139–159 (1991).

35. R. R. Murphy, *Introduction to AI Robotics* (MIT Press, Cambridge, MA, 2000).

36. S. Goss, J.-L. Deneubourg, Harvesting by a group of robots, in *Towards a Practice of Autonomous Systems* (MIT Press, Cambridge, MA, 1992), pp. 195–204.

37. A. Drogoul, J. Ferber, From Tom Thumb to the dockers: some experiments with foraging robots, in *From Animals to Animats 2* (MIT Press, Cambridge, MA, 1992), pp. 451–459.

38. B. Werger, M. Matarić, Robotic food chains: externalization of state and program for minimal-agent foraging, in *From Animals to Animats 4* (MIT Press, Cambridge, MA, 1996), pp. 625–634.

39. S. Nouyan, M. Dorigo, Chain based path formation in swarms of robots, in *Ant colony optimization and swarm intelligence, Proceedings* (Springer, Berlin, Germany, 2006), pp. 120–131.

40. S. Nouyan, A. Campo, M. Dorigo, Path formation in a robot swarm: self-organized strategies to find your way home. *Swarm Intell.* **2**, 1–23 (2008).

41. V. Sperati, V. Trianni, S. Nolfi, Self-organised path formation in a swarm of robots. *Swarm Intell.* **5**, 97–119 (2011).

42. F. Ducatelle, G. A. D. Caro, C. Pinciroli, F. Mondada, L. Gambardella, Communication assisted navigation in robotic swarms: self-organization and cooperation, in *IROS 2011, Proceedings* (IEEE Press, Piscataway, NJ, 2011), pp. 4981–4988.

43. M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, D. Burnier, A. Campo, A. L. Christensen, A. Decugnière, G. A. Di Caro, F. Ducatelle, E. Ferrante, A. Förster, J. Guzzi, V. Longchamp, S. Magnenat, J. Martinez Gonzales, N. Mathews, M. A. Montes de Oca, R. O'Grady, C. Pinciroli, G. Pini, P. Rétornaz, J. Roberts, V. Sperati, T. Stirling, A. Stranieri, T. Stützle, V. Trianni, E. Tuci, A. E. Turgut, F. Vaussard, Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. *IEEE Robot. Autom. Mag.* **20**, 60–71 (2013).

44. E. Chang, R. Roberts, An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Commun. ACM* **22**, 281–283 (1979).

45. R. C. Arkin, Integrating behavioral, perceptual, and world knowledge in reactive navigation. *Robot. Auton. Syst.* **6**, 105–122 (1990).

46. A. Saffiotti, K. Konolige, E. H. Ruspini, A multivalued logic approach to integrating planning and control. *Artif. Intell.* **76**, 481–526 (1995).

47. T. Haynes, S. Sen, Evolving behavioral strategies in predators and prey, in *Adaption and Learning in Multi-Agent Systems: IJCAI 1995, Proceedings* (Springer, Berlin, Germany, 1996), pp. 113–126.

48. R. P. Sałustowicz, M. A. Wiering, J. Schmidhuber, Learning team strategies: soccer case studies. *Mach. Learn.* **33**, 263–282 (1998).

49. M. Quinn, L. Smith, G. Mayley, P. Husbands, Evolving teamwork and role-allocation with real robots, in *ICAL 2003, Proceedings* (MIT Press, Cambridge, MA, 2003), pp. 302–311.

50. R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction* (MIT Press, Cambridge, MA, 1998).

51. J. Kober, J. A. Bagnell, J. Peters, Reinforcement learning in robotics: a survey. *Int. J. Robot. Res.* **32**, 1238–1274 (2013).

52. R. S. Sutton, Learning to predict by the method of temporal differences. *Machine Learning* **3**, 9–44 (1988).

53. C. J.C. H. Watkins, P. Dayan, Q-learning. *Machine Learning* **8**, 279–292 (1988).

54. L. Baird, A. Moore, Gradient descent for general reinforcement learning, in *NIPS 11, Proceedings* (MIT Press, Cambridge, MA, 1999), pp. 968–974.

55. J. Baxter, P. L. Bartlett, Reinforcement learning in POMDPs via direct gradient ascent, in *ICML'00, Proceedings* (Morgan Kaufmann Publishers Inc., San Francisco, CA, 2000), pp. 41–48.

56. C. W. Anderson, "Approximating a policy can be easier than approximating a value function", tech. rep. CS-00-101 (Colorado State University, Fort Collins, CO, 2000).

57. M. T. Rosenstein, A. G. Barto, Robot weightlifting by direct policy search, in *IJCAI'01, Proceedings* (Morgan Kaufmann Publishers Inc., San Francisco, CA, 2001), pp. 839–844.

58. L. Panait, S. Luke, Cooperative multi-agent learning: the state of the art. *Auton. Agents Multi Agent Syst.* **11**, 387–434 (2005).

59. L. Buşoniu, R. Babuška, B. De Schutter, A comprehensive survey of multiagent reinforcement learning. *IEEE Trans. Syst. Man Cybern. C* **38**, 156–172 (2008).

60. P. Stone, M. Veloso, Multiagent systems: a survey from a machine learning perspective. *Auton. Robot.* **8**, 345–383 (2000).

61. L. E. Parker, Decision making as optimization in multi-robot teams, in *ICDCIT 2012, Proceedings* (Springer, Berlin, Germany, 2012), pp. 35–49.

62. J. Girard, M. R. Emami, Concurrent markov decision processes for robot team learning. *Eng. Appl. Artif. Intell.* **39**, 223–234 (2015).

63. A. Campo, Á. Gutiérrez, S. Nouyan, C. Pinciroli, V. Longchamp, S. Garnier, M. Dorigo, Artificial pheromone for path selection by a foraging swarm of robots. *Biol. Cybern.* **103**, 339–352 (2010).

64. F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, A. Martinoli, The e-puck, a robot designed for education in engineering, in *Robotica 2009, Proceedings* (IPCB, Castelo Branco, Portugal, 2009), pp. 59–65.

65. A. Gutiérrez, A. Campo, M. Dorigo, J. Donate, F. Monasterio-Huelin, L. Magdalena, Open e-puck range & bearing miniaturized board for local communication in swarm robotics, in *ICRA 2009, Proceedings* (IEEE Press, Piscataway, NJ, 2009), pp. 3111–3116.

66. A. Brutschy, L. Garattoni, M. Brambilla, G. Francesca, G. Pini, M. Dorigo, M. Birattari, The TAM: abstracting complex tasks in swarm robotics research. *Swarm Intell.* **9**, 1–22 (2015).

67. C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. A. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, M. Dorigo, ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intell.* **6**, 271–295 (2012).

68. L. Garattoni, G. Francesca, A. Brutschy, C. Pinciroli, M. Birattari, "Software infrastructure for e-puck (and TAM)", tech. rep. TR/IRIDIA/2015-004 (IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2015).

Acknowledgments: We thank A. Roli, M. Brambilla, and G. Lucy for reading a preliminary version of the article. M.B. dedicates his work to the memory of his father. **Funding:** The project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 681872). M.B. acknowledges support from the Belgian Fonds de la Recherche Scientifique, of which he is a Senior Research Associate. **Author contributions:** The authors devised the system together. L.G. realized it and performed the experiments. The authors wrote the manuscript together. M.B. conceived and directed the project. **Competing interests:** The authors declare that they have no competing interests. **Data and materials availability:** All data generated and discussed are included in the article and in the Supplementary Materials.

Supplementary Materials

In Section S1, we provide a detailed description of TS-Swarm and of the implementations of its four variants. In Section S2 we provide an extensive discussion of the results of the empirical analysis. In Section S3, we sketch the highlights of movie S1 to provide insight on a typical run of Mark I_3.

Section S1. Detailed description of TS-Swarm

The characterizing feature of TS-Swarm is that part of the robots of the swarm form a chain that (i) assists the navigation of robots between task and task; and (ii) encodes the order in which tasks must be performed. Chaining has been previously explored in swarm robotics as a means to search the environment and assist navigation (*31*, *36–43*). Nonetheless, to the best of our knowledge, the concept of robot chain has never been associated with path planning nor with planning in general. In our work, we acknowledge chaining as a path planning method and we generalize it to planning task sequences. Generally speaking, in the context of swarm robotics, a chain is a group of robots that align in space thus creating a precedence relation: one robot is positioned after the other. So far, in the swarm robotics literature, chains of robots have been conceived as sequences of robots that landmark the physical space and act as waypoints for other robots that need to navigate from one end of the chain to the other. In TS-Swarm, we generalize this picture to include robots that, in a sense, "align" in the abstract space of the tasks, one after the other, creating a precedence relation between the tasks themselves: one task must be performed after the other. These robots, in a sense, "landmark" the abstract space of the tasks and act as logical waypoints for other robots that need to perform the tasks in the order encoded by the chain. The chain that the robots create in TS-Swarm accomplishes the double role of guiding robots in the physical space and in the abstract space of the tasks. It could be considered as a chain that develops in the physical space augmented with the abstract space of the tasks. In the physical subspace, the chain encodes the information needed for navigation from area to area; in the abstract space of the tasks, it encodes the order in which tasks themselves must be performed.

In the rest of the section, we give a detailed description of Mark I_3 and we then introduce Mark I_4, Mark II_3, and Mark II_4 by highlighting their distinctive traits. Section S1.1 is devoted to Mark I_3. Section S1.2 is devoted to Mark I_4, Mark II_3, and Mark II_4.

S1.1 Mark I_3

Although all the robots of the swarm execute the same control software, they assume different roles at runtime. Roles are not pre-assigned, but are rather taken autonomously by the robots on the basis of their interactions with peers and environment. The robots can assume four different roles: *guardian*, *link*, *tail*, and *runner*. We will collectively refer to guardians, links, and tail as *chain members*.

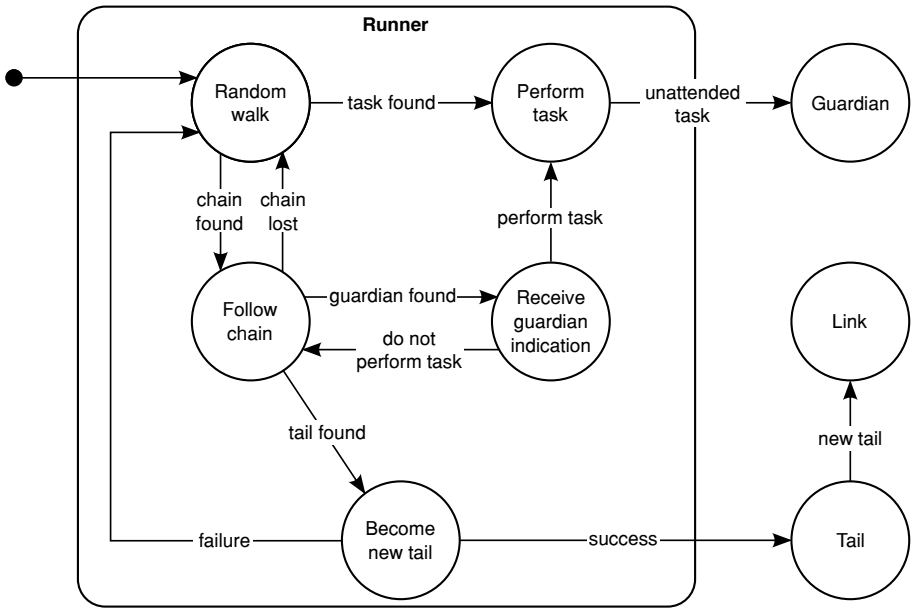


Fig. S1. State machine of TS-Swarm. High-level description of the robots' behavior.

The control software of the robots is realized as a probabilistic finite state machine in which the control cycle has period of 100 ms. A high-level representation of the probabilistic finite state machine is given in Fig. S1.

At every control cycle, all chain members broadcast a four-byte message via their range-and-bearing board. On the other hand, runners broadcast a four-byte range-and-bearing message only in specific situations, as it will be detailed in the following. The general scheme of the message encoding is given in Fig. S2. Details will be provided on a per-role basis in the following of this section.

In the rest of this section, we detail the implementation of the four robot roles of Mark I_3 for the extended e-puck platform (see Materials and Methods).

S1.1.1 Guardian

The guardians are robots that position themselves right in front of a task to signal its presence and to indicate whether a runner should perform it or not. They announce their role by displaying the color cyan through their RGB LEDs (Fig. S3). The guardians also act as end points of the branches of the chain. In particular, they are in charge of starting the construction of a branch when: (i) the task they guard is the first to be performed, or (ii) they are reached by the chain being built (provided that the whole chain is not completed, yet).

Before becoming a guardian, a robot is a runner that explores the environment performing a random walk. A runner becomes a guardian when it encounters an unattended task, performs it, and eventually stand in front of it. After performing the task of which it becomes a guardian, a robot receives positive feedback if the task is the first one to be performed and a negative one otherwise. More precisely, to act as a guardian, a robot positions itself so that the TAM of the guarded task is right behind its back. Indeed, after performing the task, and before assuming the role of

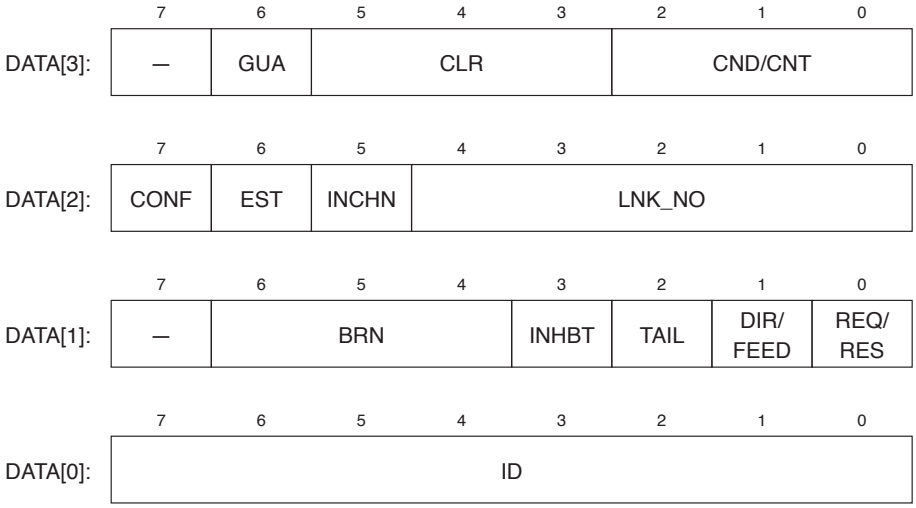


Fig. S2. Encoding of the range-and-bearing message in Mark $\mathbf{I}_3$. Guardians identify their messages by setting DATA[3][6]. Each guardian locally broadcasts its CND and CONF (see Section S1.1.1) by setting DATA[3][2:0] and DATA[2][7], respectively. Additionally, a guardian indicates the color of the guarded task (DATA[3][5:3]), whether it is in chain or isolated (DATA[2][5]), and whether the nearby runners should be inhibited (DATA[1][3])—except the one that initiated the inhibition and whose ID is indicated in DATA[0][7:0]. When it is reached by a branch of chain, a guardian also sets DATA[2][6] to indicate that the branch has been established and increments by 1 the value of BRN (DATA[1][6:4]) received from that branch. BRN enumerates the branches of chain that have been established; it is used to identify each branch and to determine when all the tasks have been connected by the chain. The links along a branch relay the value of BRN indicated by the guardian that initiates the branch. Links indicate also their position within the branch (DATA[2][4:0]) and whether the branch is established or still being built (DATA[2][6]). The tail identifies its messages by setting DATA[1][2]. To indicate the sector in which a new tail can join the chain, the tail sends, only in that sector, a message in which DATA[1][1] is set. To respond to a request for becoming a new tail, the tail sends a response by setting DATA[1][0] and inserting the ID of the selected requesting runner in DATA[0][7:0] (see Section S1.1.3 and Fig. S8). Runners' messages are characterized by the value 0 in the fields GUA, TAIL, and LNK_NO. Runners can send a request to join the chain as the new tail by setting DATA[1][0]. They notify a guardian of the success/failure of a task execution by indicating their value of CNT in DATA[3][2:0] and the feedback received in DATA[1][1] (see Section S1.1.1 and Fig. S5). Finally, a runner can inhibit other neighboring runners while communicating with a guardian or performing a task by sending a message in which DATA[1][3] is set (see Section S1.1.1 and Section S1.1.4).

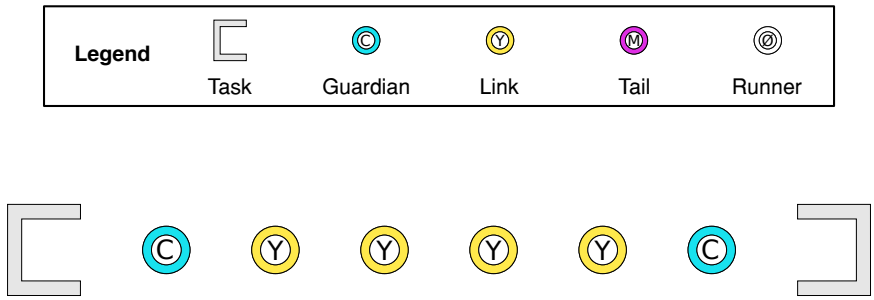


Fig. S3. Guardians. The guardians are the chain members that are at the two ends of each branch. They are identified by the color cyan. They place themselves right in front of a task and instruct other robots on whether they should perform it or not.

guardian, the robot exits the TAM moving forward and in a straight line for about 0.25 m. As a result, when the robot stops and takes the role of the guardian, the guarded task is behind its back.

The guardian can be in four different states, depending on the feedback received after performing the guarded task and on the state of chain construction.

- If the feedback is positive and the chain does not exist yet, the guardian is isolated in front of the first task. In this case, the guardian acts also as the tail of the chain to recruit chain members and form the first branch of the chain.
- If the feedback is negative and the chain is not perceived in the neighborhood, the guardian is isolated in front of either the second or third task. In this case, the guardian waits to be reached by the chain. In the meantime, it indicates that runners that have not performed any task should not perform the one it guards.
- If the feedback is negative and the branch of chain being built has arrived in the vicinity, the guardian joins the chain and completes the construction of the branch. If the construction of the chain is not complete, that is, if some tasks have not been reached yet, the guardian takes also the role of tail and initiates the construction of a further branch.
- Independently of the feedback originally received, after joining the chain and after being relieved from the duties of tail, the guardian limits itself to instruct the runners on the execution of the task guarded.

The transition between these four states is graphically described in Fig. S4.

When acting as a tail, the guardian initiates the construction of a branch of chain on its right-hand side—the guarded task being behind its back. The goal is to construct a branch that departs from the right-hand side of the task and then sweeps the space anticlockwise when its tail reaches a wall. In all other respects, the guardian acts as any other tail—see Section S1.1.3.

The primary goal of a guardian is to guide runners in the execution of tasks. We will refer to the communication protocol between runners and guardians as *guardian*

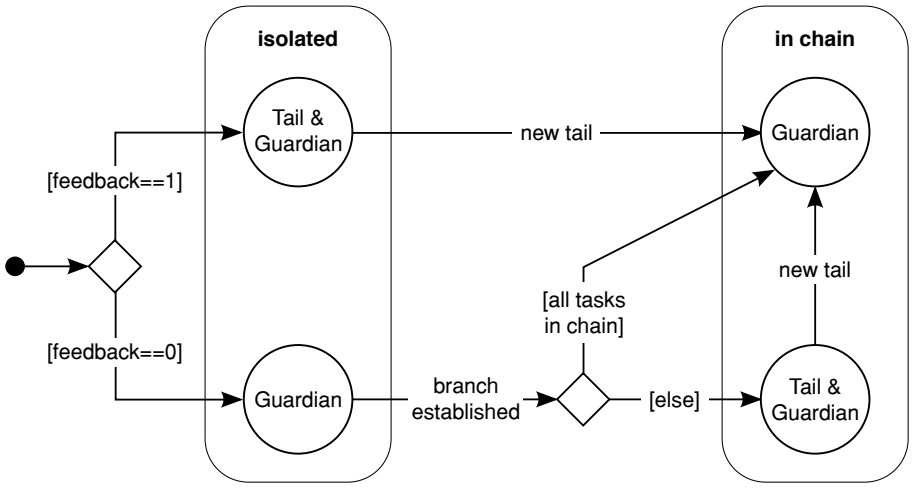


Fig. S4. State machine of a guardian. The four states in which a guardian can be, with respect to the construction of the chain.

Table S1. Guardian protocol (G protocol), description of messages.

message	full name	description
G_ADV	advertisement message	Message locally broadcast by a guardian for nearby runners; it provides the conditions under which the task should be performed and the confidence of the guardian; it contains CND and CONF
G_NTF	notification message	Message sent by a runner to a guardian after executing its task; it notifies the guardian of the feedback received upon execution; it contains CNT and FEED (which is 1 if the execution was successful, 0 otherwise)

protocol (G protocol). The G protocol works as follows—see Fig. S5 for an example of message exchange and Table S1 for a description of the messages. Every guardian locally broadcasts a range-and-bearing message (G_ADV) that contains information about the guarded task. The message contains two pieces of information: (i) the number CND of tasks that a runner must have performed to qualify for performing the guarded one; and (ii) the Boolean CONF, which states whether the guardian is confident about the value indicated in CND. CND and CONF are respectively encoded as DATA[3][2:0] and DATA[2][7]—see Fig. S2.

As mentioned above, a runner that encounters an unattended task performs it and becomes its guardian. As a guardian, if the feedback received after performing the task was positive, it will broadcast CND = 0 and CONF = 1. This means that a runner that has not performed any task will be positively instructed to perform the guarded one. On the other hand, if the feedback was negative, the task is not the first to be performed but the information available is insufficient to tell whether the task is the second or the third of the sequence. In this case, the guardian will broadcast

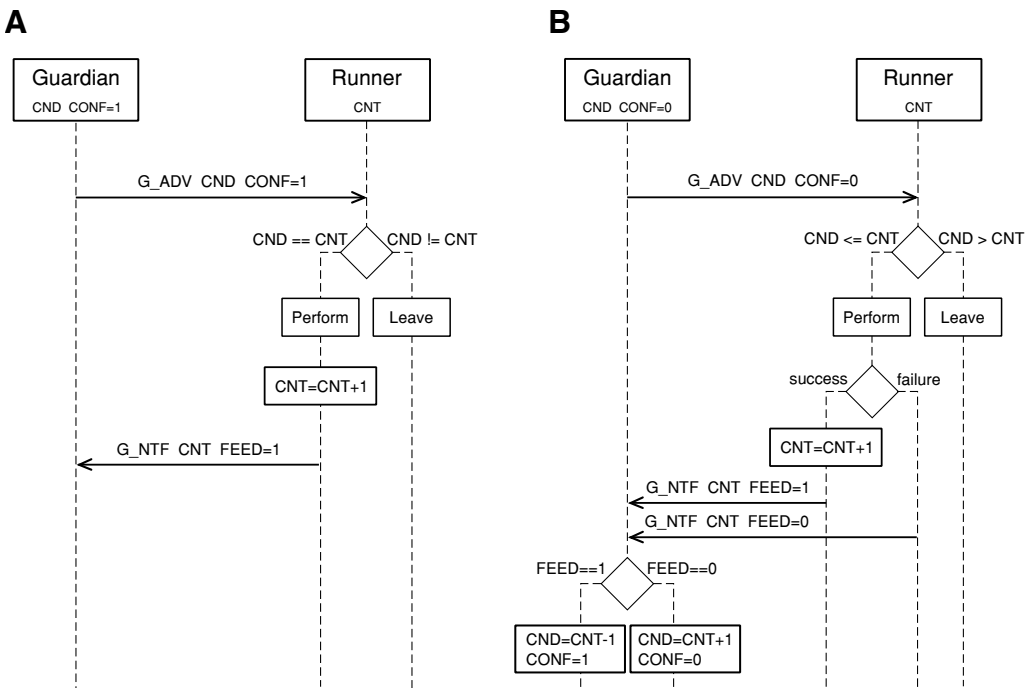


Fig. S5. Guardian protocol (G protocol), sequence diagram. A runner that receives a G_ADV message performs the guarded task depending on whether the guardian is confident (CONF) about the value of CND. (**A**) If the guardian is confident about the value of CND, the runner performs the task provided that CNT = CND. (**B**) If the guardian is not confident about the value of CND, the runner performs the task provided that CND <= CNT. In the latter case, the guardian uses the feedback notified by the runner through a G_NTF message to update CND and CONF.

CND = 1 and CONF = 0. This means that a runner needs to have performed at least one task to qualify for performing the guarded task. Though, this condition is not sufficient for guaranteeing that the execution will be successful (CONF = 0).

The guardian is notified by every runner that performs the guarded task through a range-and-bearing message (G_NTF). Specifically, after performing the task, a runner communicates whether the execution was successful (DATA[1][1], see Fig. S2) and the number CNT (DATA[3][2:0]) of tasks that it has so far successfully performed in the right order—including the guarded one, in case its execution was successful. If the guardian was not confident about the condition CND, that is, if CONF = 0, it updates its G_ADV as follows. If the runner has successfully performed the task in the correct order, the guardian sets CONF = 1 and CND = CNT − 1; otherwise, the guardian confirms CONF = 0 and sets CND = CNT + 1. The rationale is that, if the execution was unsuccessful, the runner failed to perform its task number CNT + 1 in the correct sequence. This means that the task guarded by the guardian is not the number CNT + 1 in the sequence. At the same time, as the runner has already performed CNT tasks, the one guarded by the guardian is not among the first CNT of the sequence, otherwise the guardian would have been previously notified of a success and would have already set CONF. All in all, this means that the guarded task must be at least in position CNT + 2 of the correct sequence. As a consequence, the guardian will indicate that a runner must have performed at least CNT + 1 tasks before it qualifies for performing the one at hand.

A further duty of a guardian is to relay inhibition messages produced by the runners to avoid that they crowd around a task—see Section S1.1.4 for the details.

S1.1.2 Link

The chaining behavior we implemented for TS-Swarm is loosely inspired by previous works (*31*, *40*). Instead of the polychromatic cyclic pattern previously used (*31*, *40*) to indicate the sense in which the chain should be followed, the links in TS-Swarm adopt a monochromatic pattern. They all indicate their role and mark their position in space by displaying the color yellow through their RGB LEDs. The goal of the chain is to assist the navigation of runners and ultimately lead them to the guardians of the tasks. To facilitate the navigation of runners, each link positions itself and aligns properly with respect to its two neighboring chain members: the one that precedes and the one that follows in the chain. At every control step, each link adjusts its position through displacements that consist of two components: distance adjustment and alignment (Fig. S6).

The direction of the *distance adjustment* component is defined by the line connecting the link itself and the neighboring chain member that is the farthest away (in absolute value) from a target distance of 0.15 m. The distance adjustment component is oriented towards this neighboring chain member if the latter is farther that the target distance (Fig. S6A). Otherwise, it is oriented in the opposite sense (Fig. S6B).

The direction of the *alignment* component is given by the mean of the angles α and β under which the preceding and following chain members are seen. Formally,

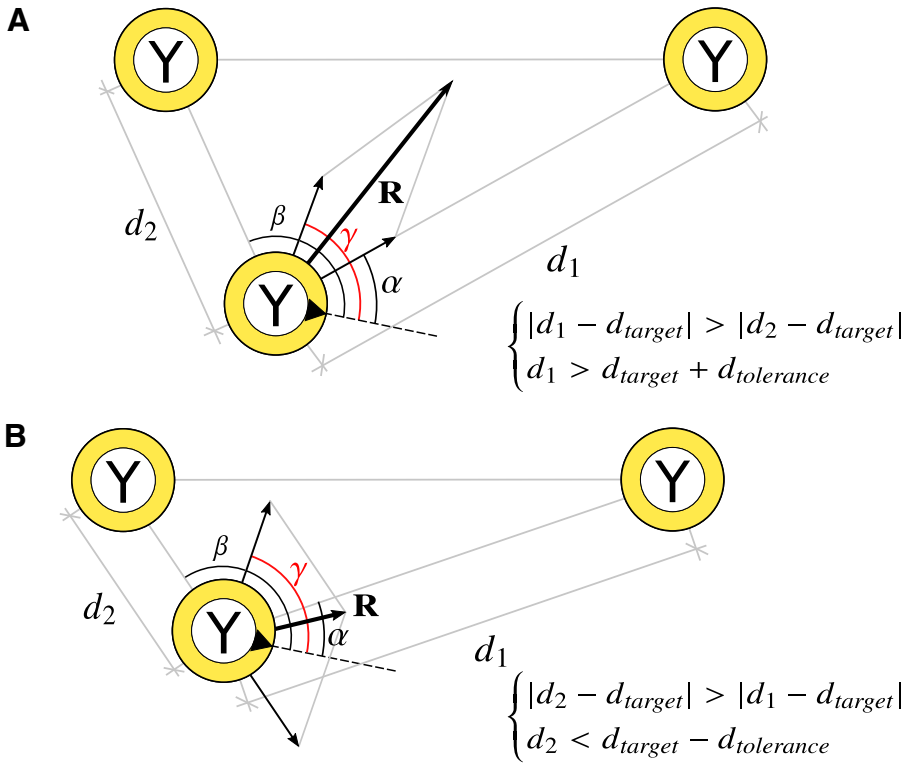


Fig. S6. Motion of a link. Every movement of a link consists of two components: distance adjustment and alignment. The two components are computed with respect to the two closest neighboring chain members that the link perceives. The alignment component lies in the direction given by the mean (γ) of the angles under which the link perceives the following chain member (α), and the preceding one (β). Angles are measured with respect to the heading of the link, which is indicated by the black triangle and the dashed line. The distance adjustment component lies in the direction of the neighboring chain member whose distance is the farthest from the target spacing distance between chain members. (**A**) If the neighboring chain member is too far, the distance adjustment component is oriented towards it. (**B**) If the neighboring chain member is too close, the distance adjustment component is oriented in the opposite sense.

the angle γ between the heading of a link and its alignment component is:

$$\gamma = \begin{cases} \arctan\left(\frac{\sin(\alpha)+\sin(\beta)}{\cos(\alpha)+\cos(\beta)}\right), & \text{if } \cos(\alpha)+\cos(\beta) > 0; \\ \arctan\left(\frac{\sin(\alpha)+\sin(\beta)}{\cos(\alpha)+\cos(\beta)}\right) + \pi, & \text{if } \cos(\alpha)+\cos(\beta) < 0 \text{ and } \sin(\alpha)+\sin(\beta) \geq 0; \\ \arctan\left(\frac{\sin(\alpha)+\sin(\beta)}{\cos(\alpha)+\cos(\beta)}\right) - \pi, & \text{if } \cos(\alpha)+\cos(\beta) < 0 \text{ and } \sin(\alpha)+\sin(\beta) < 0; \\ +\frac{\pi}{2}, & \text{if } \cos(\alpha)+\cos(\beta) = 0 \text{ and } \sin(\alpha)+\sin(\beta) > 0; \\ -\frac{\pi}{2}, & \text{if } \cos(\alpha)+\cos(\beta) = 0 \text{ and } \sin(\alpha)+\sin(\beta) < 0; \\ \text{undefined}, & \text{if } \cos(\alpha)+\cos(\beta) = 0 \text{ and } \sin(\alpha)+\sin(\beta) = 0. \end{cases}$$

To avoid oscillations, a link does not adjust its distance or alignment if its position with respect to its neighboring chain members is sufficiently close to the ideal one. The tolerance on the distance is ± 0.05 m and the one on the alignment angle is $\pm 10°$.

A link stops in its current position, halting the adjustment of its alignment and spacing, upon being notified that the branch it composes is established. The notification (encoded in DATA[2][6]—see Fig. S2) is sent by the guardian that has established the branch upon becoming the new tail of the chain; it is then relayed by the links.

S1.1.3 Tail

The tail marks the position that the chain has reached and the point from which the construction should proceed. To announce its role and location, the tail uses a unique color for its RGB LEDs: magenta (Fig. S7). In addition, the tail locally broadcasts, through its range-and-bearing board, a message that announces its role by setting DATA[1][2]—see Fig. S2. The runners that navigate along the chain are thus informed when they approach the end of the chain and can attempt to join it to take the role of the new tail. Joining the chain is a critical process that must be robust to failure, promote the formation of a well-organized chain, and prevent inconsistent states. In particular, the process must guarantee that the chain has always one and only one tail. This is achieved via a communication protocol that regulates the interaction between the incumbent tail and the prospective one. The communication is performed via the range-and-bearing board. We call the protocol *tail protocol* (T protocol). The T protocol works as follows—see Fig. S8 for an example of message exchange and Table S2 for a description of the messages.

The tail first makes sure to be at the right distance from the preceding chain member. Then, the tail starts broadcasting a directional message T_DIR (DATA[1][1], see Fig. S2). Through this message, the tail indicates the circular sector in which the new tail should position itself. The ideal direction along which the new tail should position itself is computed by extending the segment connecting the tail with the preceding chain member: as a result, the new tail will be already correctly aligned. If a runner receives the directional message, it is because it is in the intended circular sector. In this case, the runner stops in place and sends the tail a request message T_REQ (DATA[1][0]). The request message carries the unique ID of the runner (DATA[0][7:0]). The tail might receive multiple requests from different runners. Requests are handled following a first-come-first-served policy. The tail responds with a T_RES (DATA[1][0]) that contains the ID of the runner selected as the new tail (DATA[0][7:0]). Upon reception of a response containing its own ID, the selected

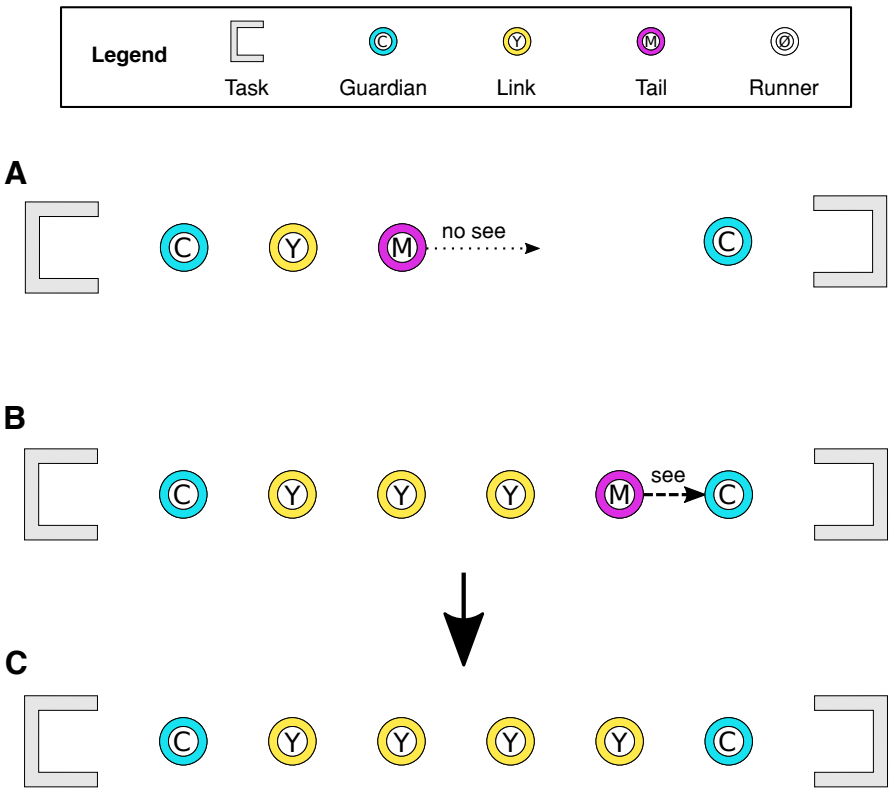


Fig. S7. Tail. (**A**) The tail marks the position that the chain has reached at a given moment in time during its construction. It is identified by the color magenta. (**B**) When the tail sees a guardian in its vicinity, it halts the construction of the current branch of the chain. (**C**) The tail becomes a link and transfers its role of tail to the guardian.

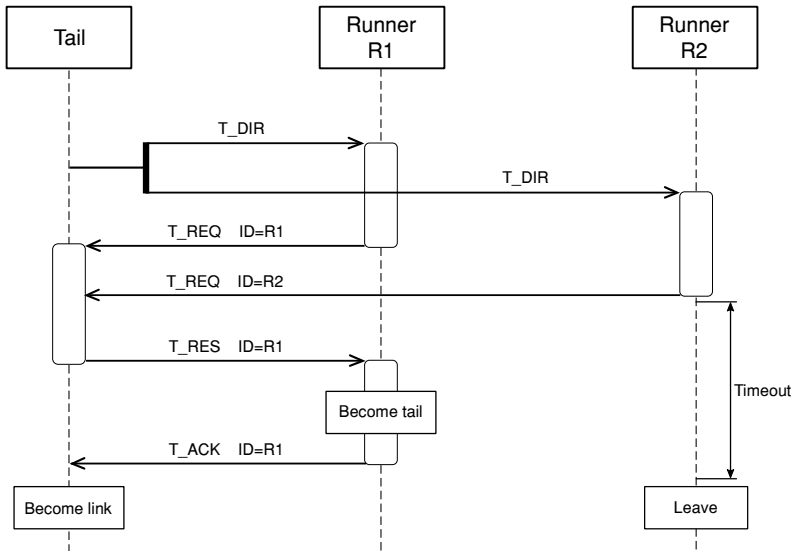


Fig. S8. Tail protocol (T protocol), sequence diagram. The tail broadcasts a T_DIR message in the circular sector in which a new tail should join the chain. Both runners R1 and R2 receive this message and request to join as the new tail through a T_REQ. The T_REQ contains the ID of the runner that is sending the request. The tail selects the first request received, in the example, the one of runner R1. Then, the tail answers by sending a T_RES that contains the ID of the request selected, in this case, R1. After receiving the T_RES containing its own ID, R1 acknowledges the tail with a T_ACK message and becomes the new tail. When the tail receives the T_ACK of R1, it becomes a link. In the meantime, the request of R2 expires without a response and runner R2 leaves.

Table S2. Tail protocol (T protocol), description of messages.

message	full name	description
T_DIR	directional message	Message broadcast by a tail only in the circular sector in which a new tail should place itself; it indicates to a runner that it is in the right position to join the chain as the new tail
T_REQ	request message	Message sent by a runner to the tail; it represents the request to become the new tail; it contains the unique ID of the runner
T_RES	response message	Message sent by the tail to a runner that previously sent a request; it represents for the runner the confirmation of being the selected one to become the new tail; it contains the unique ID of the runner selected as the new tail
T_ACK	acknowledgment message	Message sent by the selected runner to the tail; it represents the acknowledgment of the response message and the acquisition of the role of tail; it contains the ID of the runner, which is now the tail

runner becomes the new tail and closes the communication via an acknowledgment message T_ACK (DATA[1][2] is set and its own ID is inserted in DATA[0][7:0]). The tail transfers its role to the runner after either receiving a T_ACK or detecting another robot that displays the magenta tail color.

Besides managing the communication with runners that attempt to join the chain, the tail is also in charge of searching for a guardian. When a guardian is found, the tail includes it in the branch of chain being built to complete its construction. To search for a guardian, the tail drives the branch of chain being built so that it sweeps the environment in an anticlockwise motion. The tail slowly moves perpendicularly to the direction along which the branch is being built, on the left-hand side of the latter (Fig. S9C). As links react by adjusting spacing and alignment, the movement of the tail creates a domino effect: the whole branch turns pivoting around the guardian from which the branch originates, which stays still in its position in front of its task. As we have seen in Section S1.1.1, a branch departs from its originating guardian on its right-hand side—with the TAM on the back of the guardian (Fig. S9B). During its construction, a branch reaches the wall of the arena, which prevents its further extension (Fig. S9C). Thanks to the sweeping motion generated by the movement of the tail, the branch being built disentangles from the wall and allows further runners to join and extend it (Fig. S9D).

To decide when it should start to move and trigger the sweeping motion of the branch, the tail cannot rely on its ability to detect the wall of the arena in its vicinity. This is due to the limited range of the proximity sensors and to the noise that affects their readings. As an alternative, the tail measures the amount of time during which no robot attempts to join the branch. Past a predefined threshold, the tail assumes

that the branch being built is stuck against a wall and triggers the sweeping motion. Once triggered, the sweeping motion is interleaved with pauses to allow links to align and, possibly, new members to join.

When the tail spots a guardian (Fig. S9E), the sweeping motion stops and the tail aligns with the guardian and the preceding chain member. The guardian and the tail then establish the branch of the chain: the tail becomes a link and transfers its role of tail to the guardian (Fig. S9F). The guardian then broadcasts a notification (DATA[2][6]—see Fig. S2) for the links that compose the branch. Upon receiving this notification, the links halt the adjustment of their alignment and spacing, stopping in place. A new branch of the chain can now depart from the guardian. This mechanism eventually allows the swarm to form a chain that connects all the tasks, under the assumptions that the arena is convex, the tasks are located along its perimeter, and there are no obstacles in the environment (Fig. S9).

S1.1.4 Runner

Runners are all the robots that are not part of the chain. When the system is deployed, all the robots of the swarm start as runners. At runtime, they might assume the role of chain members—i.e., guardians, links, or tail—depending on the interactions they have with peers and environment. Runners do not adopt any color to announce their role. Indeed, their goal is not to landmark the space nor to provide information about task execution, but rather to follow the indications provided by the chain (Fig. S10).

A runner might become a guardian if, while performing a random walk, it encounters an unattended task. This typically happens at the beginning of the lifetime of the system. The number of runners that become guardians equals the number of tasks to be performed.

The other runners continue to explore the environment by performing a random walk. If they encounter a chain, they follow it. Runners perceive the position of chain members via their omnivision module. Runners navigate along the chain by keeping it on their left. This generates an anticlockwise movement of the runners around the chain. The traffic of runners flows neatly along a branch of the chain in both senses without interfering: the runners that travel in one sense remain on one side of the chain, those that travel in the other sense remain on the other (Fig. S11).

At every control step, a runner that follows a chain considers up to two neighboring chain members to define its direction of motion. The direction is given by the sum of a radial and a tangential component. The radial component keeps the robot at a target distance $\bar{d}$ from the chain while the tangential component allows the robot to proceed along it. The two components refer to a circle centered in a reference point. If the runner perceives only one chain member, the reference point is the position of the chain member perceived (Fig. S12A,B). If the runner perceives two chain members, the reference point is the intersection between the line passing through the two chain members perceived and its perpendicular passing through the runner itself (Fig. S12C,D).

Runners are led by the chain towards a guardian. When a runner reaches a guardian, it is instructed on whether to perform the guarded task—see Section S1.1.1 and Fig. S5. In particular, upon receiving a G_ADV message, a runner evaluates the

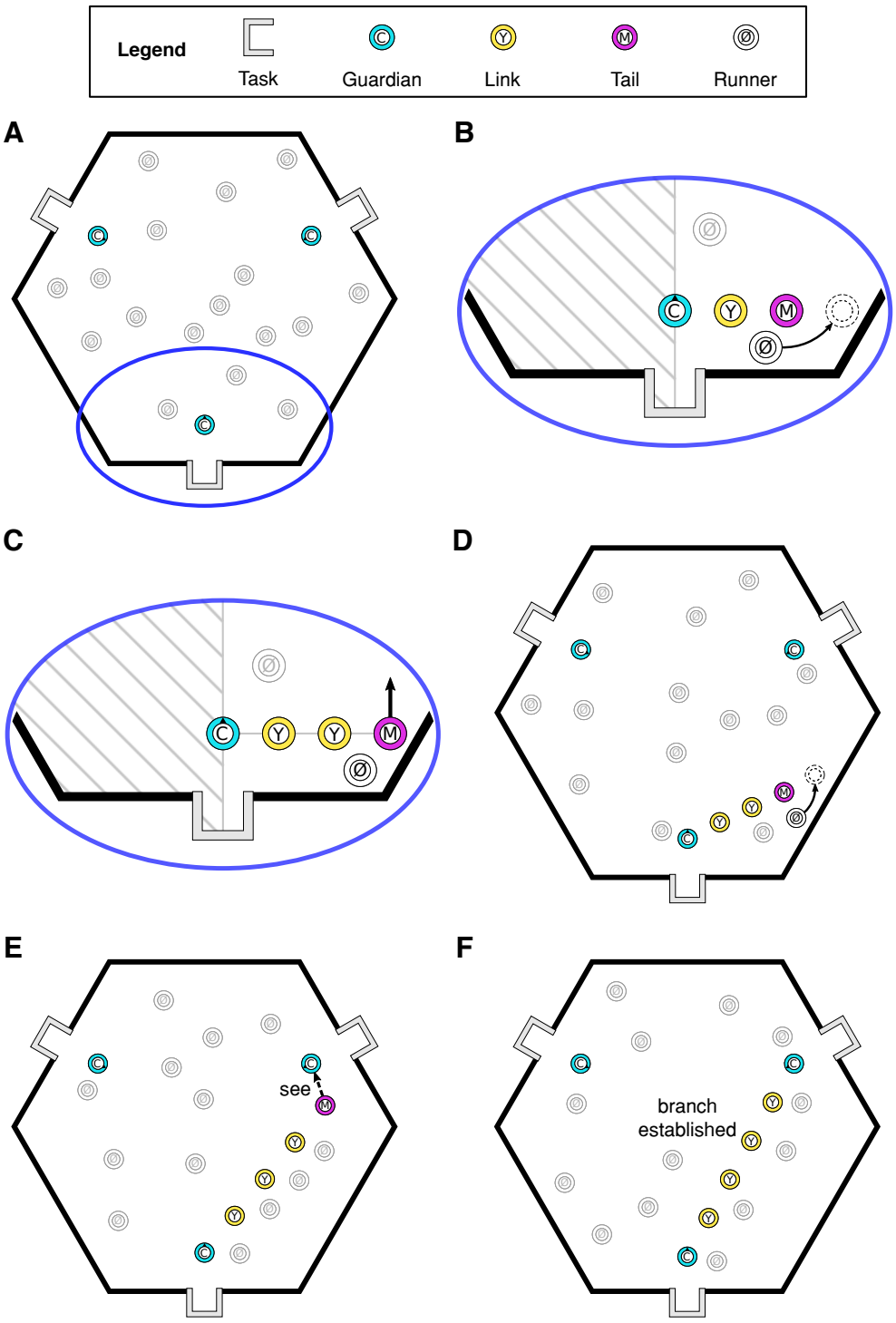


Fig. S9. Construction and motion of a branch of chain. (**A**) Three robots become guardians of the three tasks. (**B**) The one that receives positive feedback after performing its guarded task initiates the construction of a branch of chain on its right-hand side. (**C**) The branch extends until it reaches the wall of the arena, which prevents other runners from joining it. The tail then moves perpendicularly to the direction of the branch, on the left-hand side of the latter. (**D**) The movement of the tail triggers a sweeping motion of the whole branch, which disentangles the branch from the wall and enables its further extension. (**E**) The process is repeated until the tail spots the guardian of a task. (**F**) The tail transfers its role to the guardian and completes the construction of the branch.

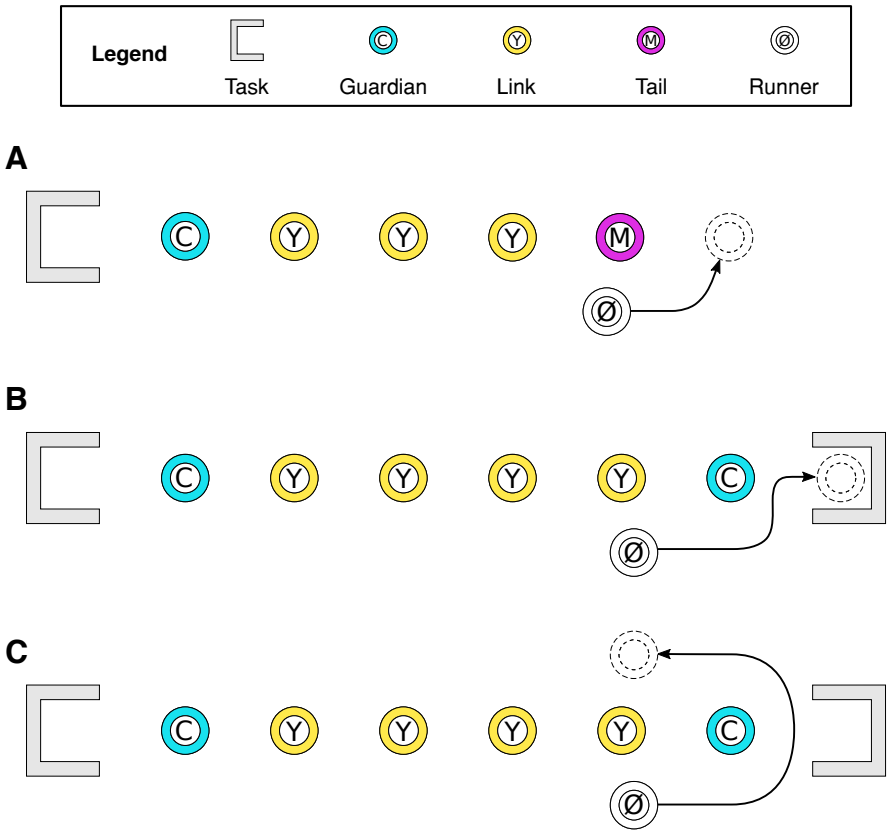


Fig. S10. Runners. Runners are not identified by a color. (**A**) If a runner navigates along a branch of chain that has not been completed yet, it eventually encounters the tail and attempts to join the chain as the new tail. (**B**) If a runner navigates along a branch of chain that is complete, it eventually encounters a guardian. The guardian might indicate that the runner must (**B**) perform the task, or (**C**) skip it.

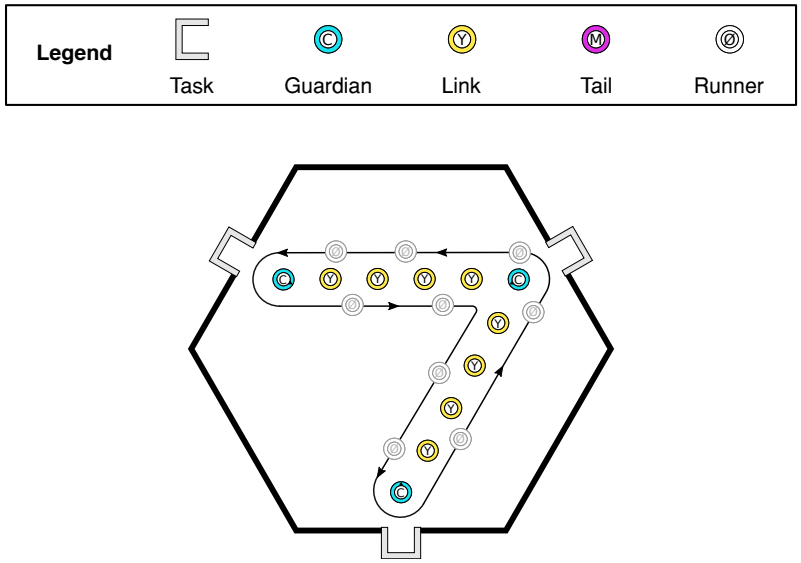


Fig. S11. Trajectory followed by the runners around the chain. Runners navigate along the chain by keeping it on their left.

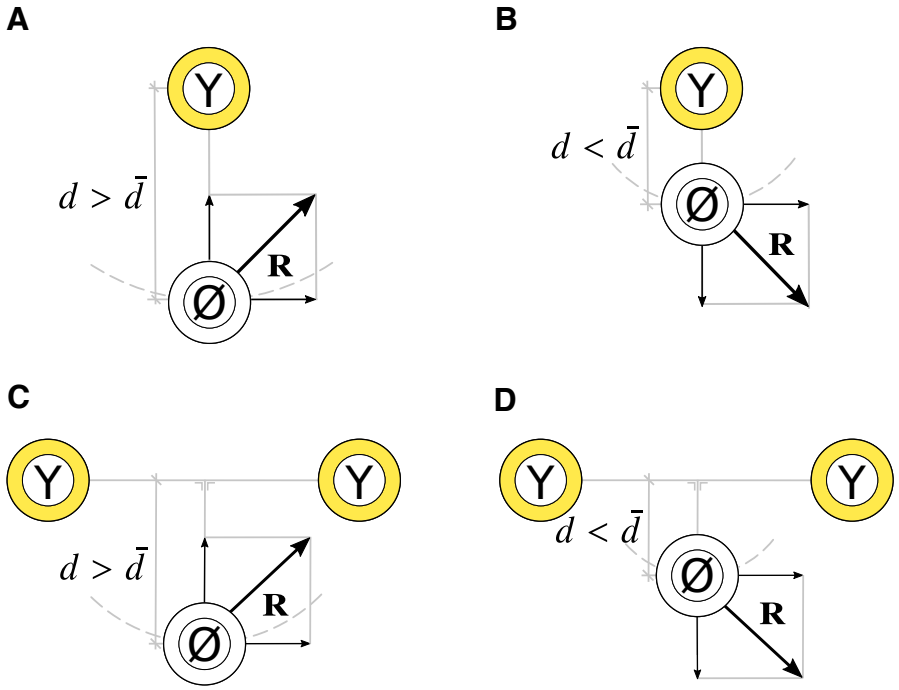


Fig. S12. Motion of a runner along a branch of the chain. The direction of motion is given by the sum of a radial and a tangential component that refer to a circle (dashed gray line) centered in a reference point. (**A**, **B**) When only one chain member is perceived, the reference point is the position of that chain member. (**C**, **D**) When two chain members are perceived, the reference point is the intersection between the line passing through those chain members and its perpendicular passing through the runner itself. (**A**, **C**) If the runner is too far from the reference point ($d > \bar{d}$), the radial component is oriented towards it. (**B**, **D**) If it is too close ($d < \bar{d}$), the radial component is oriented in the opposite sense.

information provided in light of the number CNT of tasks it has already performed in the correct order. The runner performs the task either if (i) CONF = 1 and CND = CNT; or if (ii) CONF = 0 and CND <= CNT. Otherwise, the runner skips the task signaled by the guardian. After performing the task—or skipping it—the runner revolves anticlockwise around the guardian until it finds a branch of the chain to follow. If the task was performed, while revolving around it, the runner notifies the guardian of the feedback received upon task execution—see also Section S1.1.1. The runner notifies the guardian via a G_NTF in which it encodes its (possibly incremented) value of CNT as DATA[3][2:0] and the feedback received as DATA[1][1]—see Fig. S2. Eventually, the chain leads each runner through all the tasks. By using the indication provided by the guardians, runners are able to perform the tasks along the chain in the correct order. When a runner completes the execution of a full sequence, it starts a new execution, in a cyclic manner.

To avoid overcrowding around the guardians and to enable an ordered execution of the tasks, runners adopt a mechanism of inhibition. When a runner starts approaching a task, it broadcasts an inhibition message through its range-and-bearing board. The inhibition message is echoed by the guardian of the task, if present. The nearby runners that perceive the inhibition message transition to an inhibition state or, with a certain probability, leave the area. Inhibited runners clear the way in front of the task to avoid hindering other robots' movement. After a certain period of time elapsed without receiving any inhibition message, an inhibited runner transitions back to normal operation.

S1.2 Mark I_4, Mark II_3, and Mark II_4

In this section, we modify the assumptions under which Mark I_3 is developed, thus making the sequencing problem harder. In Section S1.2.1, we modify the assumption that the tasks to be sequenced are three. We do this by introducing TS-Swarm Mark I_4, a system that sequences four tasks. In Section S1.2.2, we modify the assumption that runners receive feedback after performing each single task. Under this assumption, a runner is notified that the sequence being performed is wrong as soon as it performs the first task that departs from the correct sequence. We modify this assumption by considering the case in which a runner has to perform an entire sequence before receiving any feedback. We introduce TS-Swarm Mark II_3, a system that sequences three tasks without needing immediate feedback after the execution of a task. In Section S1.2.3, we modify both assumptions at the same time. We do so by introducing TS-Swarm Mark II_4, a system that sequences four tasks without needing immediate feedback after the execution of a task.

S1.2.1 TS-Swarm Mark I_4

TS-Swarm Mark I_4, hereafter Mark I_4, sequences four tasks instead of the three sequenced by Mark I_3. The only difference between Mark I_3 and Mark I_4 is that in the latter the task counter of the robots counts up to four. In all other respects, the two systems are identical. In particular, the implementations, the values of all the parameters, and the encoding of the range-and-bearing messages (Fig. S2) are identical.

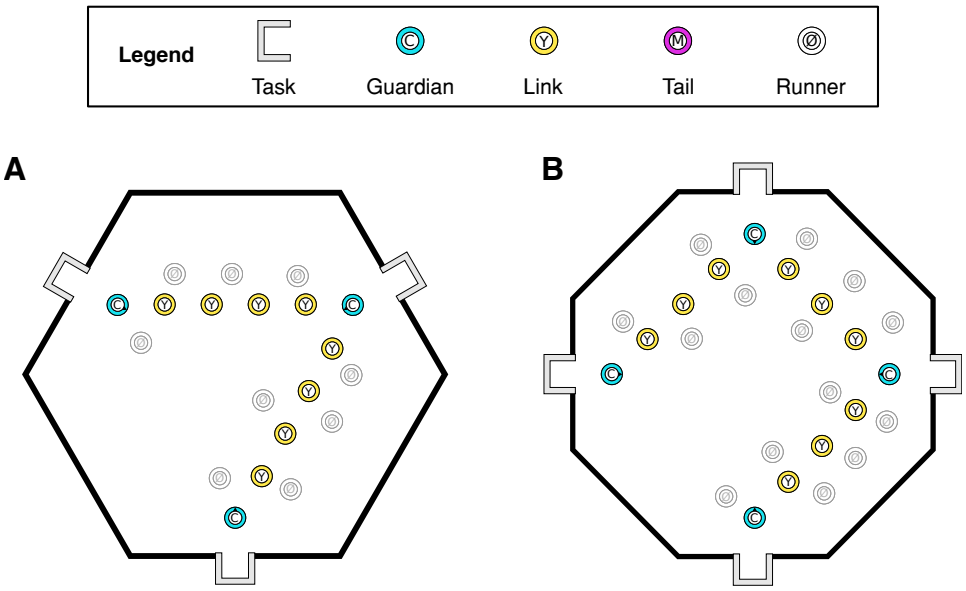


Fig. S13. The chain in Mark I$_3$ and Mark I$_4$. (**A**) The chain built by Mark I$_3$ is composed of two branches. (**B**) The one built by Mark I$_4$ is composed of three branches.

In Mark I$_4$, four robots take the role of guardian, one per task, and the chain comprises three branches (Fig. S13B).

S1.2.2 TS-Swarm Mark II$_3$

TS-Swarm Mark II$_3$, hereafter Mark II$_3$, assumes that a runner has to perform an entire sequence before knowing whether the sequence itself is correct or not. Mark II$_3$ differs from Mark I$_3$ in the following main points:

- All guardians initiate the construction of a branch of the chain in parallel. When all the branches are complete, the chain is a closed loop that connects all the guardians. The guardians use the closed-loop chain to exchange information and to establish an initial sequence of the tasks.
- Once the initial sequence is established, the guardians direct the runners so that the initial sequence and all its permutations are tested, one after the other. The swarm collectively explores the tree of the permutations of the initial sequence. Each robot (guardians, links, and runners) contributes to the collective exploration by acting reactively, relying only on partial knowledge of the sequences being tested.
- After completing a sequence, a runner receives feedback on whether the sequence is correct or not. It notifies the feedback to the guardian of the last task, which then shares it with all other guardians via the closed-loop chain. If the sequence is correct, from then on all runners are instructed so as to perform it. Otherwise, the following sequence in the depth-first search is tested.

All guardians initiate the construction of a branch because, after executing a task for the first time, they do not have any way to break the symmetry among themselves.

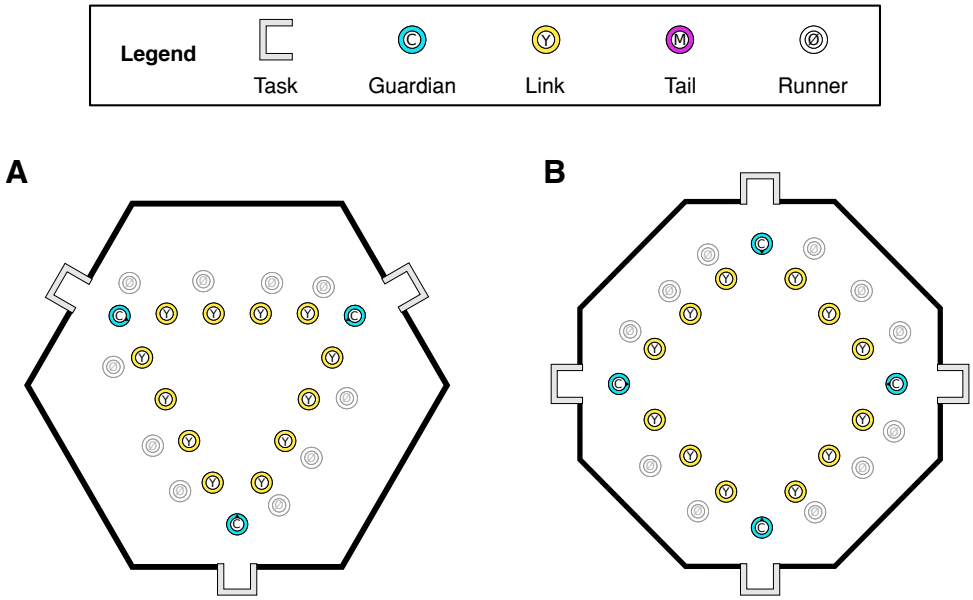
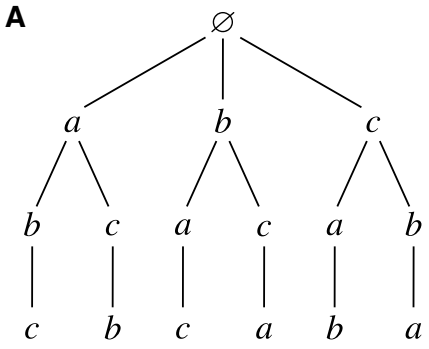


Fig. S14. The chain in Mark II$_3$ and Mark II$_4$. The guardians do not have any way to break the symmetry among themselves after performing their guarded task. Therefore, they all initiate the construction of a branch of the chain. When all branches are complete, the chain forms a closed loop that the guardians use for communicating. (**A**) The chain built by Mark II$_3$; and (**B**) the one built by Mark II$_4$.

Indeed, in Mark I$_3$ (and Mark I$_4$), after executing their respective tasks, one of the guardians receives positive feedback while the others a negative one. The one that receives the positive feedback is the one associated with the first task to be performed in the sequence. This is the guardian that initiates the construction of the chain. In Mark II$_3$, no guardian receives any feedback so the above mechanism cannot be used to break the symmetry at this moment of the system's lifetime. As an alternative, all guardians initiate the construction of a branch. When all branches are complete and the chain is closed (Fig. S14A), the guardians communicate via range-and-bearing messages that are relayed by the chain members. The guardians use the communication medium that they have established to break the symmetry and order themselves. The order of the guardians (and therefore of the tasks they guard) defines the initial sequence to be tested and eventually the whole permutation tree that will be then searched. The guardians order themselves using a leader-election algorithm (*44*) through which they assign the label c to the one of them with the largest ID. Guardian c sends a message that is relayed clockwise along the closed-loop chain. The first guardian that receives the message is assigned the label b and the following the label a.

Mark II$_3$ explores the space of the possible sequences by traversing the tree of the permutations of $\langle a, b, c\rangle$ (Fig. S15). The tree is explored via depth-first search. In practice, the guardians coordinate themselves to direct the runners so that they test the possible sequences one after the other. As a first step, the guardians address the runners to the tasks guarded by a, b, and c, in this order; as a second step, to the tasks guarded by a, c, and b; as a third step to the tasks guarded by b, a, and c, and so on. The exploration of the permutation tree is distributed. At any moment in



B

UPI			
0:	*a*	*b*	*c*
1:	*a*	*c*	*b*
2:	*b*	*a*	*c*
3:	*b*	*c*	*a*
4:	*c*	*a*	*b*
5:	*c*	*b*	*a*

Fig. S15. Exploration of the space of possible sequences in Mark II$_3$. (**A**) Permutation tree. (**B**) List of sequences explored via a depth-first search of the permutation tree. UPI is the *unique permutation identifier*: the sequence with UPI = 0 is the initial sequence defined via the election algorithm. UPI = 1 is its first permutation encountered by searching the permutation tree via depth-first search; UPI = 2 is the second one, and so on.

time, each guardian has only partial knowledge about the sequence being tested. To clarify this, consider the example in Fig. 6, in which the guardian standing in front of the green task is assigned the label *b* via the election mechanism described above.

The transition between a step and the following one is triggered by a failure reported by a runner. When a runner that has performed all the three tasks in the order prescribed at that step receives negative feedback, it reports the failure to the guardian of the last task performed. This guardian initiates the transition to the following step by sending a range-and-bearing message that is relayed along the closed-loop chain. The guardians that receive the message transition to the following step. The exploration of the permutation tree is thus reactive: guardians transition from one sequence to the following one in response to the notification of a failure from a runner or from the closed-loop chain.

As they approach a guardian, the runners are notified of the transition and start from scratch the execution of the new sequence, as directed by the guardians.

Eventually, the search of the permutation tree leads the swarm to hit the correct sequence. When the first runner receives positive feedback after performing the three tasks in the correct order, it reports the success to the guardian of the last task performed. This guardian communicates the event to the other guardians via the closed-loop chain. The search process terminates and the guardians continue directing the runners following the policy that produced the correct sequence.

To summarize: in Mark II$_3$, the search process relies on the cooperation of guardians, links, and runners. Each robot acts reactively and no robot has complete knowledge of the sequence being performed: (i) runners navigate the environment following the chain and perform tasks if directed to do so by the guardians; (ii) links relay information; (iii) guardians transition from state to state on the basis of the feedback reported by the runners. In this context, the state of a guardian is "direct to the guarded task runners that have already performed X tasks". The sequence of states through which each guardian transitions is uniquely determined by its position in the initial sequence, as determined by the election process. At no moment in time, a guardian has knowledge of the whole sequence being performed by the runners. Also

in the absence of immediate feedback, TS-Swarm collectively converges to the correct sequence without relying on representations of the environment or symbolic reasoning. The ability to sequence the given tasks emerges from the interaction of individual robots that act reactively.

Implementation details. The range-and-bearing message that the guardians exchange to transition from a step of the search to the following one contains a 2 bit field M4PI, *modulo 4 permutation identifier*. M4PI is an integer ranging from 0 to 3. Its value is M4PI = UPI mod 4, where UPI is the *unique permutation identifier*, of the new permutation to be tested (Fig. S15B). The general scheme of the range-and-bearing message encoding in Mark II_3 is given in Fig. S16.

S1.2.3 TS-Swarm Mark II_4

TS-Swarm Mark II_4, hereafter Mark II_4, sequences four tasks under the assumption that a runner has to perform an entire sequence before knowing whether the sequence itself is correct or not. Mark II_4 is based on Mark II_3, the only difference being that the task counter of the robots counts up to four. In all other respects, the two systems are identical: the implementations, the values of all the parameters, and the encoding of the range-and-bearing messages (Fig. S16) are the same.

In Mark II_4, four robots take the role of guardian, one per task, and the chain is a closed loop that comprises four branches (Fig. S14B). The tree of permutations and the list of sequences explored by Mark II_4 is given in Fig. S17.

Section S2. Discussion of the results

We further discuss the experimental results reported in the article—Fig. 4, Fig. 5, and Fig. 7.

S2.1 Experiments with Mark I_3

Fig. 4A,B compares the results obtained by Mark I_3 in the robot experiments and in the simulated ones, under the same experimental setting. The two plots show the empirical run-time distributions for the execution of one, five, and ten correct sequences—see Materials and Methods for an introduction to the notion of empirical run-time distribution. The plots indicate that Mark I_3 is typically able to perform the tasks in the correct order. It does so both on the robots and in simulation: the performance is similar in the two cases.

Concerning the experiments on the robots, Mark I_3 was able to complete ten sequences before the time cap on 9 runs out of 10. In the remaining run, it was nonetheless able to determine the correct order of the tasks and to perform 9 sequences before the time cap. In the laboratory notebook (Table S3), we report the record of all the information we collected during the 10 runs with the robots. Concerning the simulated experiments, Mark I_3 achieved a success ratio of 90%: in 27 runs out of 30, Mark I_3 was able to complete ten sequences before the time cap. All in all, the results show that Mark I_3 is able to sequence and perform the three given tasks both on the

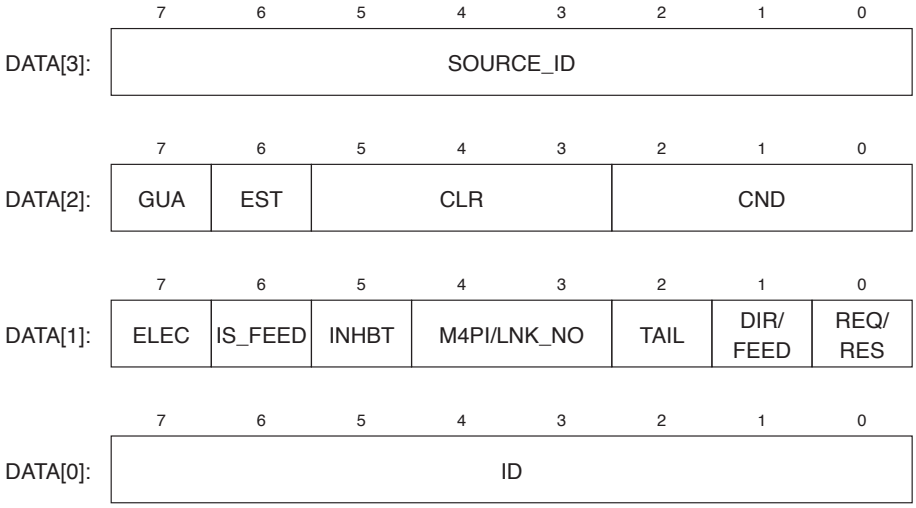


Fig. S16. Encoding of the range-and-bearing message in Mark II$_3$. Guardians identify their messages by setting DATA[2][7]. When the closed-loop chain is complete, the guardians use a leader-election algorithm to break the symmetry among themselves. In the messages exchanged during the phase of leader election, the guardians set DATA[1][7]. Once a first order among themselves has been established, the guardians start to test the possible sequences of the tasks. By setting DATA[1][4:3], each guardian broadcasts locally the counter M4PI (modulo 4 permutation identifier). Its value is M4PI = UPI mod 4, where UPI is the unique permutation identifier associated with the permutation of the initial sequence that is currently being tested—see Fig. S15. Each guardian broadcast locally also the number CND of tasks that a runner must have performed to qualify for performing the guarded one. This is done by setting DATA[2][2:0]. Runners notify the guardians of the success/failure of the sequence being tested by setting DATA[1][6] and inserting the feedback received in DATA[1][1]. If the feedback is negative, the guardian that receives the notification switches to the next sequence and sends a message to the other guardians, which in turn complete the transition to the next sequence. In all the messages that they send along the closed-loop chain, guardians insert their ID in both DATA[0][7:0] (ID) and DATA[3][7:0] (SOURCE_ID). This enables a guardian that receives a message from the chain to identify the source guardian that sent it by reading the field SOURCE_ID. The field ID is instead overwritten by every link that relays the message along the chain. The fields that are not described here are equivalent to the ones used in Mark I$_3$ (and Mark I$_4$)—see Fig. S2.

Table S3. Laboratory notebook. Results of 10 consecutive runs of Mark I_3 on 20 e-puck robots operating in an hexagonal area of $2.10\,m^2$. Record of all the information that we collected during each of the 10 runs.

run	date	begin	end	e-puck failures	time to completion of				notes
					chain	1 seq	5 seq	10 seq	
1	Oct 11, 2016	1:26 pm	1:45 pm	Runner fails at $370\,s$ without causing problems	$290\,s$	$410\,s$	$758\,s$	$1{,}198\,s$	Chain functional and built quickly; smooth execution of 10 sequences
2	Oct 11, 2016	3:57 pm	4:23 pm	Runner fails near the chain at $1{,}120\,s$ without causing problems	$565\,s$	$599\,s$	$916\,s$	$1{,}506\,s$	Chain functional but too spaced near the last guardian; this slows the execution
3	Oct 12, 2016	11:43 am	12:06 pm	No failures	$490\,s$	$590\,s$	$890\,s$	$1{,}250\,s$	Chain highly functional thanks to the two branches rather open; smooth execution of 10 sequences
4	Oct 12, 2016	2:40 pm	3:18 pm	No failures	$595\,s$	$909\,s$	$1{,}420\,s$	$2{,}248\,s$	Communication problems in the construction of the chain, but the construction completes; slow execution due to a guardian too far from its TAM (blue)
5	Oct 13, 2016	11:09 am	11:27 am	No failures	$302\,s$	$340\,s$	$680\,s$	$1{,}070\,s$	Chain straight and functional; smooth execution of 10 sequences
6	Oct 13, 2016	2:48 pm	3:24 pm	No failures	$350\,s$	$495\,s$	$772\,s$	$1{,}410\,s$	Chain overcrowded but functional, first branch not perfectly aligned; smooth execution of 10 sequences
7	Oct 14, 2016	11:13 am	11:33 am	Runner fails at $600\,s$ without causing problems	$320\,s$	$362\,s$	$564\,s$	$923\,s$	Chain well aligned and functional; smooth execution of 10 sequences
8	Oct 14, 2016	2:40 pm	3:06 pm	Runner fails at $600\,s$ near a TAM hindering other runners. Communication problems in the construction of the first branch. Runners have problems to enter the blue TAM	$425\,s$	$455\,s$	$1{,}125\,s$	$1{,}600\,s$	Communication problems in the construction of the first branch; a runner fails and hinders other runners slowing the execution
9	Oct 18, 2016	10:28 am	11:08 am	Runner fails at $1{,}850\,s$ without causing problems	$605\,s$	$988\,s$	$1{,}553\,s$	time cap	Chain overcrowded and poorly aligned near the guardians; this slows the execution; at time cap, 9 sequences performed
10	Oct 18, 2016	1:26 pm	1:54 pm	Two runners fail (at $530\,s$ and $1{,}460\,s$) without causing problems	$319\,s$	$411\,s$	$810\,s$	$1{,}656\,s$	Chain functional despite the first branch is sparse; smooth execution of 10 sequences

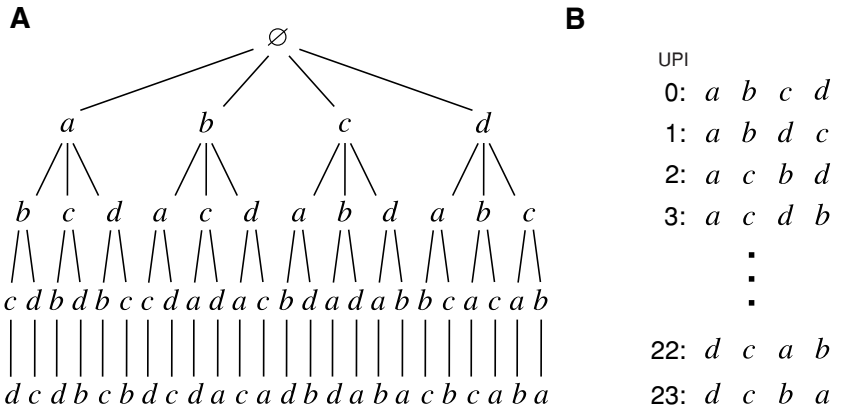


Fig. S17. Exploration of the space of possible sequences in Mark II$_4$. (**A**) Permutation tree. (**B**) List of sequences determined by exploring the permutation tree via depth-first search. UPI is defined in the caption of Fig. S15.

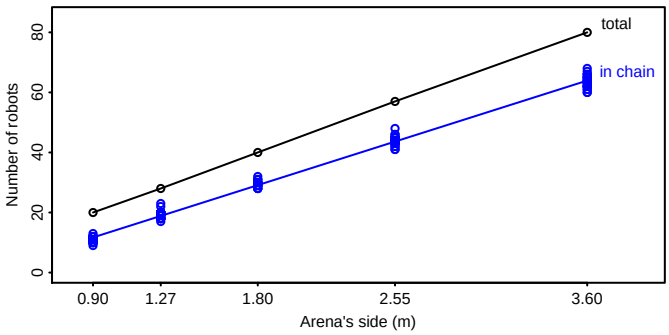


Fig. S18. Number of chain members in Mark I$_3$. Chain members in the scalability study presented in Fig. 5. The plot confirms that the number of chain members grows linearly with the side of the arena, as we assumed in Materials and Methods.

robots and in simulation. Moreover, they indicate that the simulation environment we adopted allows us to predict the performance of the robots.

In Fig. 5A(a to e), we present the results of the scalability study by reporting the empirical run-time distributions for each setting. They comprise curves for the successful execution of one, five, and ten correct sequences. The results show that Mark I$_3$ scales well with the size of the arena and the number of robots. They also confirm our hypothesis that by increasing the number of robots by the same factor of the arena's side, the robots are sufficiently many to connect all the TAMs—see Materials and Methods. This is made explicit by the plot presented in Fig. S18.

In Fig. 5A(f to o), we present the results of the robustness study in which we analyze the robustness of Mark I$_3$ to the variation of the number of robots w.r.t. the default. The run-time distributions in Fig. 5A(f to j) confirm that Mark I$_3$ is in general robust to the variation of the number of robots. Increasing sightly the number of robots is even beneficial to the performance in all the settings, while decreasing it quickly degrade the performance, as there are no longer enough robots to reliably connect the three TAMs. The distributions of the number of chain members as a function of the total number of robots—reposrted in Fig. 5A(k to o)—show that, given the size of the arena, Mark I$_3$ places roughly the same number of robots to connect the three TAMs, independently of the total number of robots that the swarm comprises.

S2.2 Experiments with Mark I_4

In Fig. 4C, we report the results of Mark I_4. They indicate that TS-Swarm can successfully sequence and perform more that three tasks without being subject to substantial modifications. As it should have been expected, it takes slightly longer to sequence four task than three, but the success ratio is the same obtained by Mark I_3: 27/30.

The scalability study reported in Fig. 5B(a to e) indicates that Mark I_4 scales well with the size of the arena and the number of robots. Fig. 5B(f to j) confirm that Mark I_4 is also robust to the variation of the number of robots. Slight increases of the number of robots are mostly beneficial to the performance in all the settings, whereas decreases degrade the performance due to a lack of a sufficient number of robots to reliably connect the four TAMs. Differently from Mark I_3, drastically increasing the number of robots w.r.t. the default also produces a lower performance. This is probably due to the fact that when the density is too high robots might interfere with each other or hinder each other's movements. The distributions of the number of robots in chain as a function of the total number of robots—reported in Fig. 5B(k to o)—confirm the observations made for Mark I_3: given the size of the arena, Mark I_4 uses roughly the same number of robots to connect the four TAMs, independently of the total number of robots that the swarm comprises. In all the settings, for at least 50% of the runs the number of chain members is independent of the total number of robots—with the exception of the -20% and -10% cases in which the number of robots is apparently insufficient for establishing a chain reliably.

S2.3 Experiments with Mark II_3

In Fig. 4D, we report the results obtained by Mark II_3. The performance of Mark II_3 is only slightly lower than the one of Mark I_3 but remains above the 75% line. In 24 out of the 30 runs, Mark II_3 successfully performs ten correct sequences within the time cap. This shows that TS-Swarm can cope with a sequencing problem in which the feedback is received only at the end of a sequence of tasks. The slightly lower success ratio with respect to Mark I_3 is possibly due to the challenge of building three branches of chain in parallel. Indeed, when the branches are built in parallel they risk to interfere or even merge with each other. The scalability study—reported in Fig. 7A(a to e)—indicate that Mark II_3 scales less well than Mark I_m, although the success ratio remains close to 50% even in the largest setting. A factor that might impact negatively the performance of Mark II_3 is that links no longer adjust their position after their branch is established—see Supplementary Materials for a detailed description of the robot control software and the main article for possible improvements. As a result, if a runner bumps into a link and pushes it away from the ideal position, the chain might be interrupted and the system might become unable to complete its mission. This is more likely to happen when the swarm comprises many robots and the arena is large. The possible improvement discussed in the main article (under the heading "Robots' movement is unsophisticated") could contribute to increase the scalability of Mark II_3. The robustness study—reported in Fig. 7A(f to j)—indicates that, compared to Mark I_3, Mark II_3 is also less robust to variations of the number of robots, again possibly because these variations affect the chain construction pro-

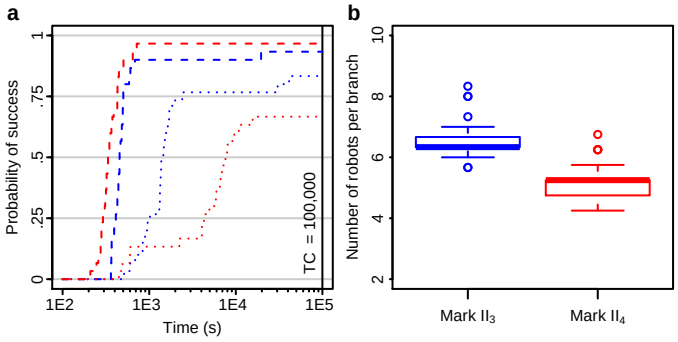


Fig. S19. Comparison between Mark II$_3$ (blue) and Mark II$_4$ (red). (**A**) Empirical run-time distributions for the construction of the chain (dashed line) and the successful execution of the first sequence (dotted line). (**B**) Number of robots comprised in each branch of the chain.

cess. Also in this case, we expect that the improvements to the chain construction and robots' movement that we propose in the main article could affect positively the performance of Mark II$_3$ in large arenas, when the number of robots departs from an ideal value. The distributions of the number of chain members as a function of the total number of robots—reported in Fig. 7A(k to o)—indicate that, in the successful runs, the number of robots that are used to connect the three TAMs is not affected by the total number of robots.

S2.4 Experiments with Mark II$_4$

In Fig. 4E, we report the results of Mark II$_4$. As it should have been expected, the success ratio of Mark II$_4$ is lower than the one of Mark II$_3$. Nonetheless, in 20 out of the 30 runs, Mark II$_4$ successfully sequences the four given tasks and performs ten correct sequences within the time cap. A comparison of the run-time distributions also shows that the time required by Mark II$_4$ to perform ten sequences of four tasks is longer than the one required by Mark II$_3$ to perform ten sequences of three tasks. Indeed, in 50% of the runs Mark II$_3$ performs ten correct sequences in less than 2,000 s. On the other hand, Mark II$_4$ reaches the 50% mark after 8,000 s. Fig. S19A shows that Mark II$_3$ and Mark II$_4$ require roughly the same time to construct the closed-loop chain. Actually, Mark II$_4$ is slightly faster. This is explained by the fact that in Mark II$_4$, the number of robots comprised in each branch of the chain is lower than in Mark II$_3$ (Fig. S19B). On the other hand, the time needed by Mark II$_4$ to perform the first sequence—that is, to solve the sequencing problem—is larger than the one needed by Mark II$_3$. Two factors contribute to this difference: (i) it takes longer to a runner to test a sequence in Mark II$_4$ than in Mark II$_3$ (4 tasks to perform vs. 3); and (ii) the space of the possible sequences explored by Mark II$_4$ is larger than the one explored by Mark II$_3$ ($4! = 24$ sequences searched by Mark II$_4$ vs. $3! = 6$ searched by Mark II$_3$). The longer duration of a run in Mark II$_4$ with respect to Mark II$_3$—in particular, the longer time needed to explore the space of the possible sequences—is likely the reason why Mark II$_4$ achieves a lower success ratio than Mark II$_3$. Indeed, the longer the time needed to explore the space of the sequences, the longer the system (particularly the chain) needs to remain functional, and eventually, the higher the chance that something goes wrong. For example, the runners might push the links and/or the guardians out of position thus breaking the continuity of the chain—see the main article for a description of possible improvements.

The results shown in Fig. 7B support the observations drawn so far. The challenges of constructing m branches of chain in parallel prevent Mark II$_m$ from achieving a

success ratio similar to the one of Mark I_m. It also lowers its scalability and robustness properties—Fig. 7B(a to e) and Fig. 7B(f to j). However, when Mark II_m completes the chain and solves the task-sequencing problem (execution of the first sequence), it reliably executes ten correct sequences. When this happens, the number of chain members is independent of the total number of robots in the swarm— Fig. 7B(k to o).

Section S3. Highlights of movie S1

We sketch the highlights of movie S1 to provide insight on a typical run of Mark I_3. Each block of text is preceded by a timestamp or an interval (typeset in boldface), which identifies the moment or the segment of the movie described by the block itself. In the movie, timestamps are displayed in the lower right corner of the frame.

00:00:00 – 00:00:19 The 20 robots are distributed randomly in the hexagonal arena. The three tasks to be performed by the robots—i.e., red, green, and blue—are randomly assigned to the three TAMs that are placed along the perimeter of the arena. In this example, the red task is assigned to the TAM on the left of the image, the green task is assigned to the right TAM, and the blue task is assigned to the TAM at the bottom of the image. The correct order of execution is red, green, blue, and it is unknown to the robots at the beginning of the run. All robots start by assuming the role of runner and move randomly in the environment.

00:00:20 – 00:01:22 Robots that see a task attempt to perform it. Three robots, one per TAM, engage in task execution. The robot that performs the red task receives positive feedback as this task is the first to be performed in the correct sequence. The robots that perform the green and the blue tasks receive negative feedback. After receiving the feedback, the three robots exit the respective TAMs by moving along a straight line for about 0.25 m; then stop and become guardians. They signal their role by displaying the color cyan with their LEDs. Because of the positive feedback received, the guardian of the red task locally broadcasts a range-and-bearing message that contains CND = 0 and CONF = 1. The other two guardians, which received negative feedback, both broadcast CND = 1 and CONF = 0.

00:00:52 – 00:01:49 The guardian of the red task signals nearby runners that they should initiate the construction of a branch of chain. The guardian indicates that the chain should be built on its right-hand side by sending a directional message only from the range-and-bearing emitters placed on the right side of its body. The chain extends on the right side of the first guardian, one robot after the other. The current tail of the chain displays the color magenta and the chain links the color yellow.

00:01:50 – 00:03:49 The branch of chain extends until its tail spots a guardian and establishes a connection. Here, the guardian spotted by the tail of the first branch is the one of the blue task. When the guardian of the blue task is reached by the first branch, it initiates the construction of a new branch that, following the same process, eventually reaches and connects the guardian of the green task.

While the chain is being built, the runners navigate along it and perform the tasks they encounter on their way, if instructed to do so by the respective guardian.

00:02:02 – 00:02:55 A runner that has not performed any task before arrives to the red task and is instructed by the guardian to perform it. This happens because the guardian knows to be guarding the first task in the sequence. After executing the task and receiving positive feedback, the runner navigates along the first branch of chain by keeping the chain members on its left.

00:02:56 – 00:05:47 The runner arrives to the guardian of the blue task. As no other robot has performed the blue task before our runner (excluding the guardian itself), the guardian is still broadcasting CONF = 0 and CND = 1. As the task counter of the runner, after executing correctly the first task, has value CNT = 1, the condition CONF = 0 and CND <= CNT is verified. The runner thus is instructed to perform the blue task as the second task. Here, the runner has to struggle to enter the TAM as other runners block its way.

00:03:50 – 00:04:40 In the meantime, the second branch of the chain is being built. When it reaches the wall of the arena, the entire branch turns, sweeping anticlockwise around the originating guardian. The sweeping motion is triggered by the tail: its goal is to disentangle the branch of chain from the wall and enable its further extension.

00:05:48 – 00:06:21 The runner that we were following eventually enters the blue TAM. However, its execution results in a failure, as (we know that) the blue task is the third task to be performed. The runner receives therefore negative feedback. The notification of this negative feedback causes the guardian to update its information to CONF = 0 and CND = 2. From this update on, all further runners having performed only the first task will be instructed by the guardian to skip the blue task and continue their navigation along the chain. (The runner that we were following aborts the execution of the sequence. It will continue along the chain until it reaches again the guardian of the red task—the first of the sequence. From there, instructed by the guardians, it will start from scratch the execution of a sequence.)

00:06:22 – 00:07:21 A runner reaches the green task after executing the red one and skipping the blue. The runner's task counter is CNT = 1 and the guardian broadcasts CONF = 0 and CND = 1. Therefore, the condition CONF = 0 and CND <= CNT is verified. The runner is therefore instructed to perform the task. The execution results in a success and the runner updates its counter to CNT = 2. Upon notification, the guardian updates its information to CONF = 1 and CNT = 1. (It should be noted that the runner enters the TAM at 00:06:38 but cannot establish a communication with it. It exits to enter again at 00:06:58.)

00:07:22 – 00:07:33 The runner has now reached the end of the chain and revolves around the third (and last) guardian in the chain that acts as a turning point.

00:08:10 – 00:08:36 Proceeding along the chain, the runner reaches the blue task—which it had previously skipped. As its task counter is now CNT = 2

and the information of the blue task's guardian is CONF = 0 and CND = 2, the condition CONF = 0 and CND <= CNT is verified. The runner thus performs the blue task. The execution is successful and the notification of the positive feedback allows the guardian to update its information to CONF = 1 and CND = 2. A first sequence has been successfully performed and the task-sequencing process is complete: all the guardians have converged to the correct policy for instructing the runners. From now on, runners will repeatedly travel along the chain and perform the tasks in the correct order.

00:18:25 The run stops after the tenth execution of the correct sequence.