The Knowledge Engineering Review, Vol. 35, e26, 1 of 40.
doi:10.1017/S0269888920000235

Adaptable and stable decentralized task allocation for hierarchical domains

VERA A. KAZAKOVA and GITA R. SUKTHANKAR

Intelligent Agents Laboratory, University of Central Florida, Orlando, FL, USA
e-mails: kazakova.cs@ucf.edu, gitars@eecs.ucf.edu

Abstract

Many real-world domains can benefit from adaptable decentralized task allocation through emergent specialization, especially in large teams of non-communicating agents. We begin with an existing bio-inspired response threshold reinforcement approach for decentralized task allocation and extend it to handle hierarchical task domains. We test the extension on self-deployment of a large team of non-communicating agents to patrolling a hierarchically defined set of areas. Results show near-ideal performance across all areas, while minimizing wasteful task switching through the development of specializations and subsequent respecializations when area demands change. A genetic algorithm is then used to evolve even more adaptable and stable task allocation behavior, by incorporating weight and power coefficients into agents' response threshold reinforcement action probability calculations.

1 Introduction

We investigate decentralized task allocation for dynamic domains with non-communicating workers who are neither limited nor informed by the actions of others or by task availability. Our goal is decentralized and communication-free task allocation that can adapt to dynamic environments of multiple tasks, by maximizing per-task performance while also minimizing how often agents switch between tasks. An existing dynamic task allocation approach is adapted for a hierarchically defined patrolling domain, consisting of multiple areas patrolled by 1000 agents. We assess whether the expected number of agents is present in each area over time, the stability of agents' self-assignments when per-area patrolling demands remain stable, and the adaptability of these self-assignments when the patrolling demands change.

We focus on decentralized, communication-free, emergent cooperation among agents whose task choices are neither limited nor informed by the choices of the other agents. Many real-world domains involve ubiquitously available tasks which can be taken up by any number of agents at any time, though the actual task demands may vary. These are referred to as *ongoing* tasks. Consider patrolling: 10 agents may be required to patrol an area at all times. Less patrolling would decrease security, while excess patrolling is wasteful and can even cause interference. Additionally, patrolling cannot be accumulated: patrolling more now does not reduce patrolling needs later. Patrolling is also never completed, requiring ongoing action from the agents. Other examples of *Ongoing* tasks include system monitoring, equipment diagnostics and maintenance, and perishable-resource gathering. Existing patrolling studies often ignore a need for flexible and scalable task allocation, neglect considerations of communication and load balancing, and focus on deterministic and centralized approaches (Portugal & Rocha, 2011) which usually rely on some form of task availability limitations (e.g., only one agent can win a task auction). *Ongoing* tasks without availability limits are seldom discussed in existing task allocation literature. Note that, while we focus our discussion on the self-allocation of patrolling agents to multiple areas with patrolling needs,

the tasks in this work can be equivalently seen as any set of tasks with ongoing demands. Our goal is task allocation that is: (1) effective, in that it optimizes per-task performance (i.e., not too much or too little patrolling); (2) stable, in that any unnecessary switching between tasks is minimized; and (3) adaptable, in that agents can self-reallocated as system needs change, while preserving that effecting and stable task allocation.

Response threshold reinforcement (Theraulaz *et al.*, 1998) is a probabilistic bio-inspired decentralized approach that uses adapting agent thresholds to achieve decentralized task allocation, which has been previously applied to *ongoing* tasks (Theraulaz *et al.*, 1998; Wu & Kazakova 2017; Kazakova *et al.* 2018). We note that other approaches exist where agents respond based on adapting thresholds, such as when agents act deterministically when a stimulus exceeds an agent's threshold (Liu *et al.*, 2007). To avoid ambiguity, we uniquely refer to the model defined in Theraulaz *et al.* (1998) as StimHab, referencing its use of **stimuli** and **habit** thresholds to calculate agents' action probabilities.

Biologically inspired techniques for emergent task allocation show comparatively high scalability, adaptability, and resource utilization, as well as low complexity and communication requirements (Zhang *et al.*, 2014). Approaches that do require communication can improve scalability (Murciano *et al.* 1997) and robustness in environments where communication is unreliable or not feasible (Kanakia *et al.*, 2016). Decentralized multi-agent approaches offer increased robustness and scalability, as they are not dependent on the availability of a central element (Almeida *et al.*, 2004; van Lon & Holvoet 2017). Entomology shows that insect societies achieve complex decentralized behavior among simple individuals such as ants and bees, leading to successful cooperation that facilitates colony life. Observing and modeling insect behaviors evolved in nature can then serve as inspiration for artificial multi-agent systems. StimHab, originally proposed as a model of observed insect behavior, employs task ***stimuli*** and action-reinforced task ***habit*** *thresholds* to calculate agents' action probabilities, which increase for tasks with higher stimuli and lower habit thresholds (Theraulaz *et al.*, 1998). StimHab agents do not communicate, nor are they aware of each other's capabilities, preferences, circumstances, or even existence, making the approach highly scalable. Agents sense current task stimuli, which change over time based on the system's performance on each task. Agents also maintain individual adaptable habit thresholds, indicating their preferences for each task. An agent's probability to act on a task increases with given a higher stimulus and a lower habit threshold. An agent decreases its threshold for the chosen task and increases its thresholds for the other tasks, becoming more likely to repeatedly select the same action, leading to emergent specialization over time. Here, *specialization* refers to an agent's preference and the resulting increased propensity to act on some task over the alternatives. Emergent cooperation among decentralized agents is characterized by adaptable behavior, beneficial to dynamic applications, such as patrolling (Almeida *et al.*, 2004; Portugal & Rocha 2011). Specialized agents lead to decreased interference and task switching, as well as to increased performance (Ono & Fukumoto 1996; Murciano *et al.* 1997; Li *et al.*, 2002; Nitschke 2008; Hsieh *et al.*, 2009; Campbell & Wu 2011; Agmon *et al.*, 2011; Román *et al.*, 2014).

Within StimHab, the basis of agents' habit thresholds and current system needs is domain-specific. In addition to action-reinforced habit, agents' thresholds can reflect experience, current circumstances, or physical suitability. Task demands are unknown to the agents, and system needs are inferred from performance levels on each task (see Section 4 for details). Performance can be relayed to the agents by surveillance cameras, via a central informer transmitting global state of performance, or even through agents individually observing storage levels, lengths of request or production queues, counts of encounters of each task type, etc. As the cost of monitoring performance is less affected (if at all) by increasing numbers of agents, emergent StimHab coordination is highly scalable.

In this work, we further investigate decentralized task allocation by extending a model of insect behavior (Theraulaz *et al.*, 1998), StimHab, to handle hierarchical sets of *ongoing* tasks by a team of 1000 agents, without relying on any additional methods of coordination. The natural behavior modeled by StimHab evolved to over millennia. Drawing further inspiration from this evolution and from the model itself, in this work, we also employ a genetic algorithm (GA) to evolve some newly defined model parameters, to better suit the results of natural evolution to our artificial system needs. The resulting task allocation is tested on a high-level patrolling domain composed of nested areas, each with dynamic

patrolling demands. We assess (1) the performance for each area over time, measured by how closely agents match the established area deployment requirements; (2) how well are the agents able to specialize, as measured by the amount of area switches (i.e., task switches) over time, and (3) how well agents reallocate when area demands change. Our tests show that StimHab allows agents to self-allocate proportionately, while promoting specialization and maintaining adaptability. GA results further show that task allocation quality and stability can be improved by changes to the relation between stimuli and agent thresholds.

2 Related work

There is little research on *ongoing* tasks allocation without communication. In this section, we review some existing approaches to decentralized task allocation and discuss their applicability to domains with *ongoing* tasks.

Task allocation often relies on task supply limitations and information about the choices of others. Discrete tasks are commonly presented one at a time (e.g., bidding on truck-painting jobs; Cicirello & Smith (2004)). When multiple tasks are available in unlimited quantities, agents must decide which task is needed more and for which is the agent better suited. Not having to recruit or confer with others can allow for faster responses to system changes, as well as for improved scalability, but having no communication nor limits on task availability does preclude the use of approaches employing auctions (McIntire *et al.*, 2016; Nunes *et al.*, 2016; Zheng & Koenig 2011), token-passing (Farinelli *et al.*, 2006; Ma *et al.*, 2017), or interagent recruitment (dos Santos & Bazzan, 2009, 2011; Ducatelle *et al.*, 2009; Wawerla & Vaughan 2010).

Ongoing task allocation without supply limitations has been addressed by probabilistic state transitions and by StimHab. For proportionate agent deployment to multiple locations (e.g., multi-area surveillance), locations/tasks can be assigned probabilities for agents to transition from one task to another (Berman *et al.*, 2007; Halász *et al.*, 2007; Hsieh *et al.*, 2008). Relying on global transition probabilities does not promote specialization, while the need for explicitly defined transition values complicates adaptability in dynamic environments. StimHab can address generalized task allocation with no communication for a non-hierarchical set of *ongoing* tasks, promoting specialization to reduce wasteful task switching (Wu & Kazakova 2017). StimHab may struggle during respecialization in dynamic domains, but performance can be improved by strategically resetting agent specializations (Kazakova *et al.*, 2018). While many other applications of StimHab exist, they commonly rely on limited task supply, having agents compete for each 'job' (e.g., RoboCup Rescue, dos Santos & Bazzan (2012); factory job assignments, Campos *et al.* (2000); Nouyan (2002); Cicirello & Smith (2004); Nouyan *et al.* (2005); Ghizzioli *et al.* (2005); threshold dependent competition, Merkle & Middendorf (2004); and token-passing for Unmanned Aerial Vehicle surveillance, Schwarzrock *et al.* (2018)). To our knowledge, *ongoing* hierarchical tasks are not addressed in the literature.

3 Domain: hierarchical deployment

We apply StimHab to a high-level patrolling domain, with hierarchically defined areas to be patrolled by a large group of non-communicating agents. Below we discuss why hierarchical tasks are interesting and how patrolling fits multi-agent task allocation for hierarchical *ongoing* tasks.

A hierarchical domain can be seen as a layered version of multiple sets of linear tasks, to which StimHab has been successfully applied (Kazakova *et al.*, 2018). This layering groups tasks under parent tasks, where the amount of work needed for the parent equals the sum of the amounts of work needed by the siblings, that is, if insufficient agents allocate themselves to a parent task, then there will be insufficient allocations for its children tasks. Thus, choosing among parent tasks obscures the individual needs of the children tasks, which could hinder the agents' ability to task allocate effectively. However, selecting among all the tasks in a flattened domain does not scale well to domains with many tasks. When selection is layered, large sets of tasks can be eliminated by deciding against a single parent task, resulting

in better scaling. Additionally, a hierarchical task breakdown into subdomains allows agents to self-allocate at higher levels, while lower levels can employ alternative task allocation methods, potentially better suited for these subdomains. For example, while explicit scheduling may be prohibitive in very large systems as a whole, it may be the optimal choice for smaller sets of tasks that require more precise task allocation or more explicit cooperation. Hierarchically defined StimHab would allow a subset of agents to individually self-allocate to the subdomain of interest and then switch to an alternative multi-agent control method. Even if StimHab is used at all levels, other parameters can be varied per level, such as how often agents reconsider their current task choice, given that different subdomains may be more or less dynamic, have higher or lower task-switching costs, etc. A final benefit of hierarchical domain definition is that it can allow for a more intuitive domain breakdown, potentially simplifying system design.

Multi-area patrolling fits multi-agent allocation for *ongoing* tasks: multiple locations in constant need of patrolling, a natural hierarchical breakdown of the overall space to be patrolled, patrolling demands per area that vary over time, and performance that benefits from each agent specializing on any one area. Decentralized approaches to patrolling usually involve some form of communication among the agents. Patrolling approaches that rely solely on marking patrolling frequency, such as using a pheromone diffusion model (Chu *et al.*, 2007), do not promote specialization, which has been shown to be beneficial for patrolling (Agmon *et al.*, 2011). Hierarchical patrolling represents an intuitive breakdown of a large area to be patrolled (e.g., a school campus subdivided into quadrants, each quadrant with multiple buildings, each building with multiple floors). Each subarea represents a subtask of the larger area (or task) that encompasses it. Specializing on patrolling a single area diminishes the time agents spend traveling between areas, thus increasing patrolling efficiency.

To minimize the dependence of our results on domain-specific area adjacency and traveling costs, we model patrollable areas as an abstract set of *ongoing* tasks. Patrolling performance then depends on whether agents self-deploy proportionately to demands, using stimuli for indirect coordination. The patrolling performance of each area per time unit is the number of agents that patrolled the area during that time, seen as a percentage of the desired number of patrolling units for that area, where the desired number can depend on expected targets within the area, scheduled events, etc. The actual performance calculations are highly domain-specific, but for physical areas it can be set up as counters of patrolling units versus targets entering and exiting an area. Agents need no direct awareness of where the other agents are currently patrolling, only whether an area is currently sufficiently patrolled.

4 Probabilistic action using StimHab

We propose an extension for an existing approach for decentralized task allocation (Theraulaz *et al.*, 1998), which we term StimHab (for brevity and to distinguish it from other threshold-response methods). First, we review how stimuli and habit thresholds are defined, updated, and used within StimHab to calculate agents' action probabilities, as well as how the resulting actions lead to specialization. We then extend StimHab formulation to handle hierarchical task domains.

4.1 Global task stimuli

Agents perceive system needs through globally observable stimuli (e.g., dimensions of a fire to be put out Kanakia *et al.* (2016) or task performance Kazakova *et al.* (2018)) and use them for decentralized coordination.

StimHab stimuli do not directly indicate task demands, instead being a sort of gas pedal to incite agents to act more or less on a given task. Knowing the total number or ratio of agents needed by a task per some time unit does not help an agent decide whether to take on that task. Instead, agents can decide whether their services are needed based on how well the task is being handled, that is, task performance. Using inverse performance values directly as stimuli, however, results in an environment that is too unstable for specialization. No activity on a task corresponds to maximal stimulus (1.0), while the correct amount of activity corresponds to minimal stimulus (0.0). Achieving the correct work distribution leads to no

stimulus, a sharp drop in activity, and a subsequent drop in performance, leading to an increasing stimulus, an increase in activity back to previous levels, leading to zero stimulus again, etc. This oscillating stimulus precludes agents from stabilizing their task assignments.

Task t's stimulus s_t is updated as a change in the previous stimulus, with the assumption that the absence of work on a task with non-zero demand must lead to a stimulus increase. This can mean a drop in performance (e.g., fewer than desired agents patrolling a given area) or a drop in the level of a stored resource (e.g., materials being expended faster than collected).

$$\begin{aligned} s'_t &= s_t + (\Delta s_t \text{ given no work}) * (1 - (\text{step performance})) \\ &= s_t + \left(\frac{1}{\text{steps in cycle}}\right) * \left(1 - \frac{\text{step work done}}{\text{step work needed}}\right) \end{aligned}$$

up to a $\min(s_t) = 0.0$ and $\max(s_t) = 1.0$. If agents perform the correct amount of work, s_t remains unchanged; excess work leads to s_t decrease, while insufficient work leads to increase. We set Δs_t to increase the task's stimulus from minimum 0.0 to maximum 1.0 within a single *cycle*, defined as some number of consecutive decision *steps*. After any one step, if agents do no work on task t, s_t will increase by 1.0/(steps in cycle), that is, given a cycle of 100 steps, $s'_t = s_t + 0.01$.

4.2 Individual task habit thresholds

Each agent maintains its own habit threshold for every task. These are often referred to as 'response thresholds', which can be misleading when agents do not respond based solely on these thresholds. Thus, we regard these thresholds as preferences toward each task.

When agent a acts on a task t, its threshold for that task $\theta_{a,t}$ is reduced, while thresholds for the other tasks are increased. Lower thresholds increase action probability (see Section 4.3), causing agents to become more likely to act on the same task in the future, leading to specialization over time. As agent a repeatedly acts on task t, $\theta_{a,t}$ tends to 0.0, while all other $\theta_{a,t' \neq t}$ tend to 1.0 according to the following threshold reinforcement rules:

$$\begin{aligned} \theta_{a,t} &= \theta_{a,t} - \xi && \text{(where } \xi \text{ is the affinity rate)} \\ \theta_{a,t' \neq t} &= \theta'_{a,t} + \phi && \text{(where } \phi \text{ is the aversion rate)} \end{aligned}$$

with θ restricted to the range [0.0, 1.0]. Affinity and aversion rates dictate how fast agents specialize based on their actions.

4.3 Action selection

StimHab achieves task allocation through probabilistic actions based on global task stimuli and agents' individual habits (Theraulaz *et al.*, 1998). An increase in a task's stimulus indicates that more work is needed on that task, while a lower agent's task threshold indicates that agent has developed an affinity for this task, inciting it to act at lower stimuli than agents with higher thresholds for that task.

Every time step, each agent a calculates the probability $P_{s,\theta}$ to act on every task t, by combining the task's stimulus s_t with its own affinity for that task $\theta_{a,t}$:

$$\begin{aligned} P_{s,\theta} &= s_t^2/(s_t^2 + \theta_{a,t}^2) && \text{where } s \in [0.0, 1.0], \theta \in [0.0, 1.0] \\ P_{s,\theta} &= 0.5 && \text{where undefined } (s_t = \theta_{a,t} = 0.0) \end{aligned}$$

The redefinition at $s_t = \theta_{a,t} = 0$ avoids division by zero, while leading to a 50%/50% chance to select the task or not, which falls precisely between the adjacent values $P_{s,\theta} = 1.0$ for $(s_t > 0.0, \theta_{a,t} = 0.0)$ and $P_{s,\theta} = 0.0$ for $(s_t = 0.0, \theta_{a,t} > 0.0)$, while also matching the values along the rest of the diagonal where $s_t = \theta_{a,t}$, as every $[s_t = \theta_{a,t}] > 0.0$ results in $P_{s,\theta} = (s_t^2/(s_t^2 + s_t^2) = 0.5$.

Having all *ongoing* tasks always available, agents must choose whether to act on any one of them. In prior work, we extend StimHab to allow agents to choose among multiple tasks (Wu & Kazakova, 2017),

as the original definition presents agents with one task at a time (Theraulaz *et al.*, 1998). Agents consider tasks in descending order of $P_{s,\theta}$ (different across agents as their $\theta_{a,t}$ differ), which promotes proportionate task allocation and specialization. When considering each task, if (random value) $\in [0.0, 1.0] < P_{s,\theta}$, the agent acts on that task; else the task with the next highest $P_{s,\theta}$ is considered. If no task triggers action, the agent idles until the next time step.

4.4 Hierarchical task assignment

We extend StimHab to hierarchical domains composed of tasks and subtasks. To this end, task selection and the reinforcement of task habit thresholds both require newly recursive definitions.

Agents will choose from among a subset of tasks at each level, recursively diving deeper in the hierarchy until selecting a task with no subtasks, or defaulting to idle. Under StimHab with a linearly defined set of tasks, once an action is selected, the agent can act. For a hierarchical set of tasks, a chosen task must then be checked for subtasks: if the task has subtasks, the agent will repeat the task selection process, this time choosing among the subtasks of the originally selected task; if the task has no subtasks, the agent will act on the chosen task. At any level of the hierarchy, if the probabilities $P_{s,\theta}$ for the tasks under consideration do not trigger any one task to be selected, the agent will idle. Thus, action selection always ends at a 'leaf' task (either an actual task or at idling, which is a type of 'leaf' task implicitly available at every level).

Selecting a task must trigger habit threshold updates for that task and its sibling tasks, that is, we are choosing to specialize on that task and against the alternatives. Under StimHab with a linearly defined set of tasks, acting on a task causes its threshold to reduce, while the thresholds for all other tasks increase. For a hierarchical set of tasks, only the tasks at the same level as the chosen task (i.e., sibling tasks) will have their thresholds increased. Additionally, since choosing a task means its parent task (if any) was also chosen along the path down the hierarchy, updates must then propagate upward. This means the parent task's threshold will also be reduced, while the thresholds of the parent task's siblings will be increased; the update will then be moved further up to the grandparent task and so on until the top level is reached.

5 Testing domain setup

Our patrolling scenario has 12 hierarchically arranged areas, that is, 12 tasks (see Figure 1). Agents are not directly aware of task demands, instead sensing globally monitored task performance values. Ideal behavior corresponds to continuously having the correct number of agents patrolling each area. To establish an optimal baseline, we ensure that demands at each level in the hierarchy always add up to 100% of the available agents (i.e., demands for T1–T5 always add up to 100% of agents; if demand for T2 = x%, demands for its subtasks T2.1, T2.2, and T2.3 add up to x%). Thus, any misallocation will necessarily cause some tasks to fall below 100% performance. The team consists of 1000 non-communicating, homogeneous agents with uniformly random initial habit thresholds $\in [0.0, 1.0]$. Each simulation lasts for 500 cycles = 50 000 steps. Patrolling demands change 10 times, once every 50 cycles (5000 steps or task-choosing decisions by each agent). Patrolling demand values are provided in Table 1.

Adaptability of decentralized solutions often depends on population diversity, which can be lost after initial adaptation, complicating readaptation (Mavrovouniotis *et al.*, 2017; Price & Tiño 2004; Kazakova *et al.*, 2018). Partial or full resetting of conditioning can be used to improve performance: forgetting older observations in adversarial decision-making (Villacorta *et al.*, 2013), forgetting older training environments in case-based reasoning robotic navigation (Kira & Arkin 2004), and resetting habits in a task allocation using StimHab (Kazakova *et al.*, 2018). Thus, we test two versions of each considered approach: one *without resetting* of thresholds $\theta_{a,t}$ back to uniformly random values when demands change, requiring agents to respecialize from the previously developed specializations, and one *with resetting*.

Table 1 Patrolling demands per area (T1 through T5 and subareas). Demands change every 50 cycles, 10 times per simulation (e.g., on cycle 100, the domain changes from needing 4% of all agents patrolling T1 to needing 16%. T2 changes from needing 16% to 40%: 4% on T2.1, 20% on T2.2, 15% on T2.3)

CYCLES	T1	T2	T2.1	T2.2	T2.3	T3	T4	T4.1	T4.2	T4.2.1	T4.2.2	T5
1–50	4%	20%	4%	8%	8%	32%	20%	4%	16%	8%	8%	24%
51–100	4%	16%	8%	4%	4%	24%	32%	8%	24%	12%	12%	24%
101–150	16%	40%	4%	20%	16%	4%	12%	4%	8%	4%	4%	28%
151–200	24%	12%	4%	4%	4%	28%	16%	4%	12%	4%	8%	20%
201–250	4%	32%	12%	8%	12%	16%	36%	12%	24%	8%	16%	12%
251–300	36%	16%	4%	4%	8%	4%	20%	8%	12%	4%	8%	24%
301–350	16%	16%	4%	4%	8%	8%	12%	4%	8%	4%	4%	48%
351–400	56%	12%	4%	4%	4%	4%	20%	12%	8%	4%	4%	8%
401–450	12%	16%	8%	4%	4%	4%	52%	36%	16%	12%	4%	16%
451–500	32%	24%	8%	12%	4%	20%	12%	4%	8%	4%	4%	12%

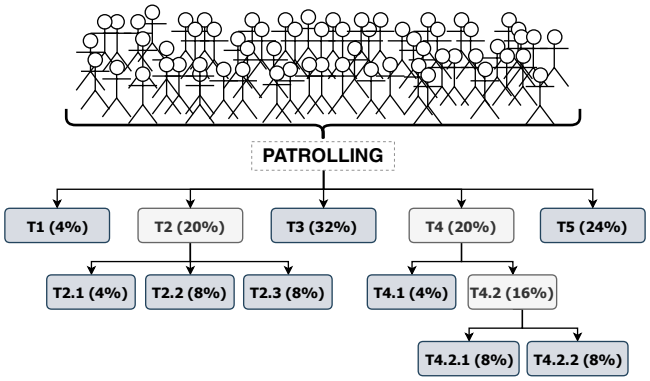

Figure 1 Hierarchical patrolling domain: nested areas T1–T5 and their subareas have patrolling demands (shown as percentages of total work possible per step) that change over time. Without communicating, agents choose areas based on the area stimuli and agents' own evolving habit thresholds for each area. If a chosen area has subareas (shown in light gray), selection repeats among subareas

6 Behavioral metrics

To assess respecialization capabilities of each approach, we employ the following terms:

Demand period: A period during which task demands remain stable; when demands change, a new demand period begins and will end when demands change again; observing average behavior across demand periods showcases the agents' average ability to adapt to changes in task demands.

Task performance: ongoing tasks are never finished, and thus, performance represents the proportion of work done as compared to work needed per unit of time, such as per time step (a period of one action from each agent) or per cycle (a period of multiple action steps).

|Deviation| from 100% task performance: The percentage of work misallocation with respect to a task, either above or below the ideal task requirement; as task performance can exceed or fall below 100%, averaging deviations prevents positive and negative misallocations from canceling each other (e.g., average task performance of two steps of 95% and 105% performance is 100%, but absolute average deviation is 5%, more meaningfully showcasing the task allocation behavior).

Task switch: A change in agent activity, switching from acting on one task to acting on another (including switching to or from idling).

Task-switching rate: For any time period, the percentage of all actions that involved a task switch (e.g., given 100 steps per cycle and 100 agents, 10 000 actions take place in a cycle; a 20% task-switching rate indicates 2000 task switches for that cycle).

Ideal task allocation: That resulting in continuous task performances of 100% or, equivalently, average task performance deviations of 0%, with average task switching approaching 0% over time.

Adaptation period: A period between demand changes, during which agents must respecialize in order to fulfill new system needs. Looking at average adaptation period behavior, we can assess the expected task allocation behavior in dynamic environments.

Performance is assessed using: (1) task performance percentages over time (ideal at 100%, i.e., no over- or underworking), (2) average task deviations calculated for each cycle of a 50-cycle demand period (ideal at 0%), and (3) average task-switching rates for each cycle of a 50-cycle demand period (ideal at 0%, although some switches are expected during adaptation).

Observing task performance percentages over all 10 demand periods of a simulation showcases the average adaptation behavior of each approach in a dynamic environment. Ideally satisfied tasks are those that quickly reach and maintain 100% *task performance* after task demands change, while task performance below and above 100% indicates insufficient and excess agents on that task, respectively. Observing average task performance deviations and average task switch counts showcases how each approach is able to respond to changes in task demands. Average deviations show the agents' ability to fulfill newly updated task demands, with deviations approaching 0% over time indicating near ideal task fulfillment resulting from an appropriate self-allocation. Average task switch counts show the stability of the task assignments, with values nearing 0% over time indicating agents specializing and continuously working on a single task.

7 Experiments, part I: StimHab vs. Average(s, θ)

To verify the benefits of StimHab for decentralized allocation with specialization, we compare agents behavior under StimHab to that resulting from a direct averaging of the two signals for our objectives, that is, stimuli and habit thresholds. While agents could be made to follow stimulus or thresholds alone, neither would achieve the dual aim of maximizing performance while minimizing task switching. Below we present the specifics of the tested approaches, the hierarchical set of tasks used, and the actual demand values used for the different tasks throughout the simulations.

To assess the ability of a decentralized team to self-allocate to a set of *ongoing* hierarchically defined tasks, we compare the capabilities of StimHab and Average(s, θ) approaches to deploy agents to a set of areas with dynamic patrolling demands. Both approaches combine stimuli and habits to maximize performance and minimize task switching. Average(s, θ) uses the same definitions of stimuli and thresholds, while its probability to act is the average of the two values, accounting for the inverse relationship of thresholds and action probability:

$$\textbf{Average(s, } \boldsymbol{\theta}\textbf{)} : P_{s,\theta} = \frac{s_t + (1 - \theta_{a,t})}{2} \quad \text{vs.} \quad \textbf{StimHab: } P_{s,\theta} = \frac{s_t^2}{s_t^2 + \theta_{a,t}^2}$$

where a is an agent, t is a task, s_t is the task's stimulus, and $\theta_{a,t}$ is the agent's habit threshold for that task. All s_t and $\theta_{a,t}$ are initialized to uniformly random $\in [0.0, 1.0]$.

To assess performance with and without resetting of agents' thresholds, we test four approaches: StimHab with resets, StimHab without resets, Average(s, θ) with resets, and Average(s, θ) without resets. We compare their ability to provide the expected number of agents to each area as demands change, as well as to allow for specialization by reducing task switching over time. First, we look at the behavior over the entire simulation and then over an average demand period, to establish the average adaptation that can be expected when demands change.

Figures 2, 3, 4, and 5 depict per-area (or per-task) performances for each cycle of a 500-cycle representative simulation run. The x-axis shows cycles and the y-axis shows $performance_t = (workachieved_t / workneeded_t)$. Each color line represents the patrolling performance for a single area.

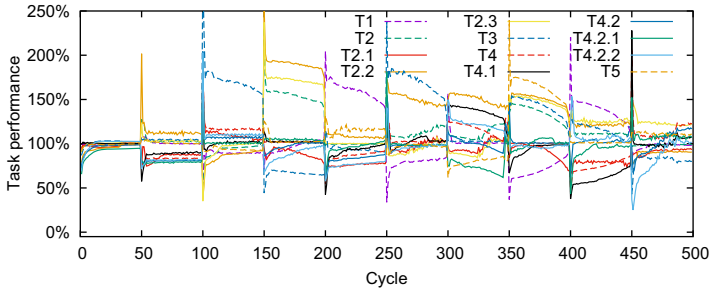

Figure 2 StimHab **without** resets—full simulation

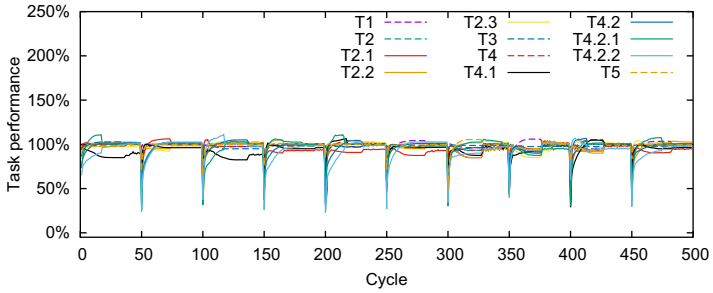

Figure 3 StimHab **with** resets—full simulation

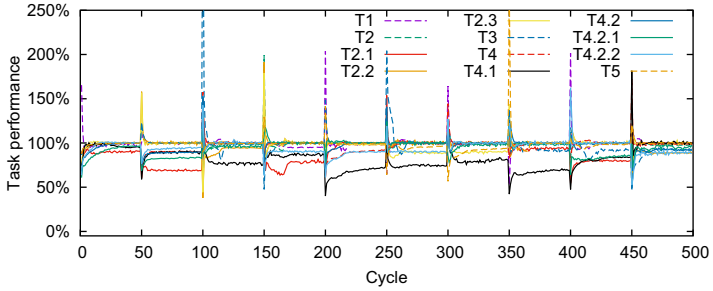

Figure 4 Average(s, θ) **without** resets—full simulation

Ideal task allocation corresponds to 100% performance in every area, as we do not want too many nor too few patrolling agents. Patrolling demands change every 50 cycles, causing spiking from 100% performance. Under StimHab with resetting (Figure 3) and Average(s, θ) with resetting (Figure 5), performances approach 100% performances faster than their versions without resetting, corroborating prior research that indicates that threshold reinforcement is more effective starting from random thresholds (i.e., specialization) than starting from previously reinforced thresholds (i.e., respecialization) (Kazakova *et al.*, 2018). Figures 6, 7, 8, and 9 provide an averaged view of the same runs, showcasing average adaptation behavior between changes in patrolling demands. The x-axis represents each nth cycle between demand changes, and the y-axis displays the average percentage *deviation* from the work needed on each task. Deviations are used in place of raw performance to ensure that values above and below 100% do not cancel out: $deviation_t = |100 - performance_t|$. Ideal patrolling performance corresponds to 0% deviation for each area. StimHab with resetting (Figure 7) and Average(s, θ) with resetting (Figure 9) both reach near 0% deviation for all areas over time, while versions without resetting remain at as high as 25% (Figure 6) and 20% (Figure 8) deviation.

Figure 10 plots the task switches for each of the four approaches, averaged over the 10 changes in demands. The x-axis depicts the average nth cycle of a 50-cycle period (i.e., between changes in patrolling demands), and the y-axis shows the average number of times agents switch tasks during that cycle. Given 1000 agents and 100 steps per cycle, a maximum number of switches per cycle is 100 000. We see

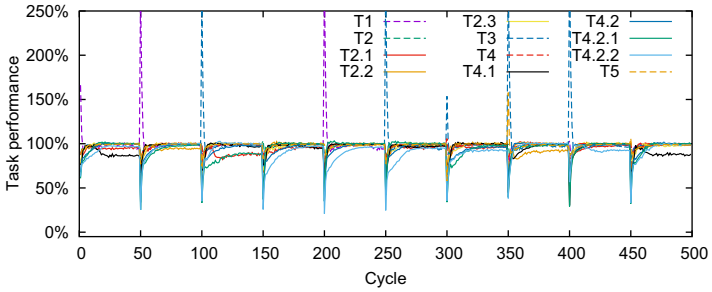

Figure 5 Average(s, θ) **with** resets—full simulation

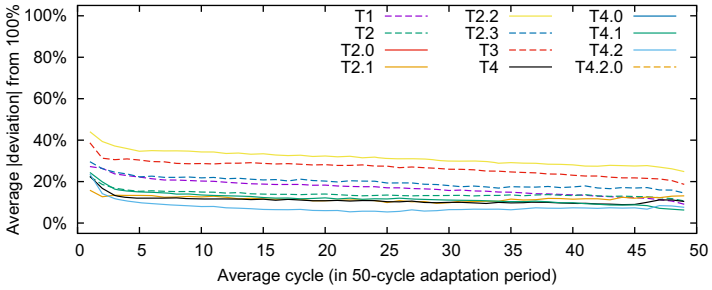

Figure 6 StimHab **without** resets—average adaptation

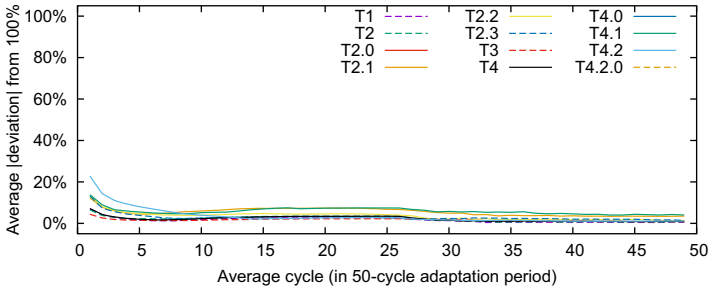

Figure 7 StimHab **with** resets—average adaptation

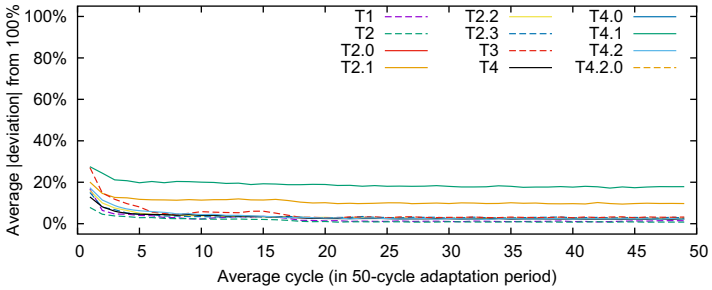

Figure 8 Average(s, θ) **without** resets—average adaptation

here that although StimHab and Average(s, θ) (both with resetting) had similar ability to fulfill average patrolling demands, StimHab with resetting is able to reduce task switches to approximately 2000 per cycle, amounting to two switches per agent per 100 task-choosing decisions (one decision per step, thus 100 per cycle). StimHab without resetting comes in second, only reducing task switching to approximately 18 000 times per cycle, while Average(s, θ) with and without resetting result in twice as much task switching. Thus, we see that StimHab greatly reduces task switching, though never quite eliminates it.

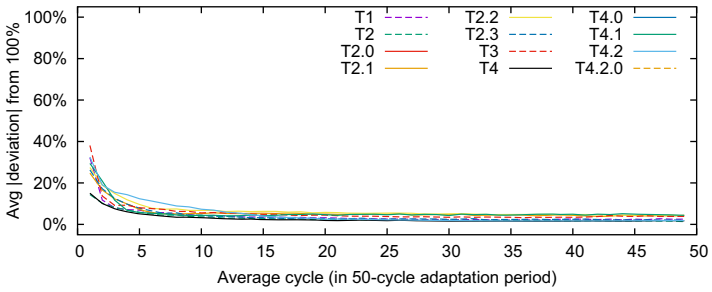

Figure 9 Average(s, θ) **with** resets—average adaptation Numbered color lines represent individual area (or task) performance. LEFT: entire simulation, with demands changing on every 50th cycle; values near 100% are ideal; RIGHT: adaptation behavior, averaged over 10 demand periods; values near 0% are ideal.

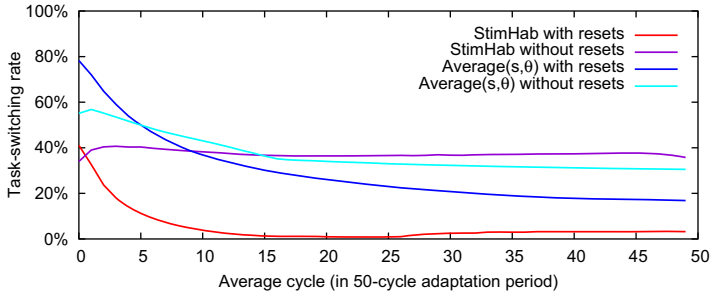

Figure 10 Average task-switching adaptation (approaches **with** threshold resetting result in fewer task switches over time, i.e. better stabilization)

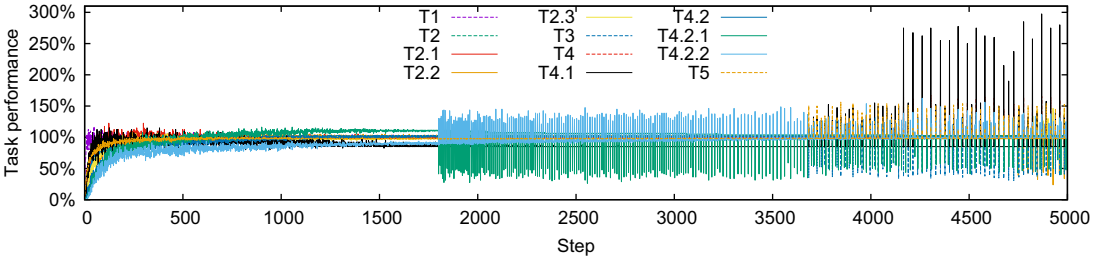

Figure 11 StimHab **with** resets—first 5000 steps of a simulation

To further investigate the cause of these task switches, we graph task allocation performance at each step of a representative sample run for the approaches with lowest overall task switching: StimHab with resetting in Figure 11 and Average(s, θ) with resetting in Figure 12. To ensure visibility, only the first 5000 steps (of the total 50 000) are shown, but the discussed behavior repeats itself every 5000 steps. We see that although task allocations averaged for each cycle were nearing the ideal 100%, at the step level there is considerable oscillation in both approaches. It is nevertheless clear that while Average(s, θ) performance oscillates continuously throughout the entire 5000-step period (notice the dense spiking in Figure 12), StimHab performance (Figure 11) initially approaches the desired task allocation and then begins to oscillate. The spikes here are sparser than under Average(s, θ) but have higher amplitude, indicating intermittent large shifts in agent assignment.

Looking closer at the simulation data, we see that StimHab's performance begins to oscillate once tasks with too many agents (performance line above 100%) reach $s_t = 0$. By this point, agents have strongly developed habits ($\theta_{a,t} = 1.0$ one of the tasks), causing $P_{s,\theta}$ to drop from 1.0 to 0.5, and leading to a decrease of activity on that task and an increase on the others. A sharp decrease in activity brings s_t down, restarting the cycle. This oscillation hinders fine-tuning hierarchical specializations and is precisely why resetting is beneficial for threshold adaptation: fully specialized agents have trouble choosing what

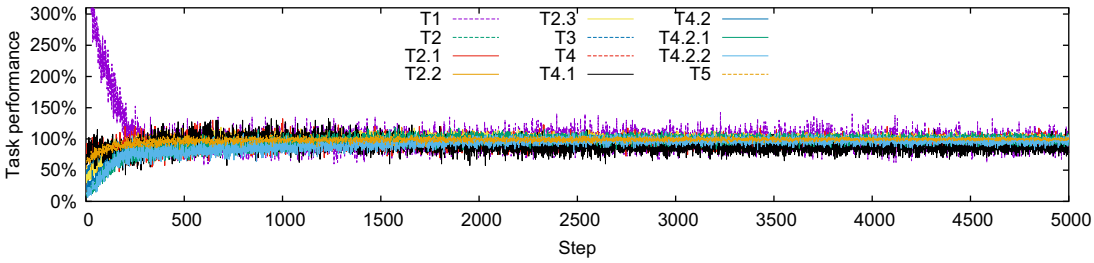

Figure 12 Average(s, θ) **with** resets—first 5000 steps of a simulation

is needed over what they now prefer (Kazakova *et al.*, 2018). We hypothesize that respecialization can be improved by altering probability calculation in favor of tasks where $s_t = 1.0$.

8 Favoring readaptation: altering behavior when $(s_t = \theta_{a,t} = 1.0)$

Comparing StimHab to a direct averaging of the stimulus and habits shows that agents can approach the expected task allocation at the cycle level, but oscillations are observed at the step level after an initial adaptation period. In this section, we consider how altering the probability formula used by StimHab can improve agents' respecialization, thus making a decentralized system more adaptable within dynamic domains.

Consider the probability values for StimHab shown in Figure 13. The depicted map shows all possible probabilities (up to two decimal discretization) resulting from stimulus and threshold value pairings. Threshold values are shown below the probability map, representing the x-axis, while stimulus values are shown vertically to the left of the map, representing the y-axis, with the pairing (0.0, 0.0) at the bottom-left corner.

The behavioral oscillations observed in our earlier experiments (Figure 11) can be explained by comparing the probabilities in the bottom-left versus top-right corners of the map: both ends of the $P_{s,\theta} = 0.5$ diagonal have equal action probability, while representing vastly different stimulus-threshold pairings. The bottom-left corner $s_t = \theta_{a,t} = 0.0$ corresponds to tasks that are minimally needed, but for which an agent has developed maximal specialization (or preference). The top-right corner $s_t = \theta_{a,t} = 1.0$ corresponds to tasks that are maximally needed, but for which an agent has developed minimal specialization (or aversion). A single agent may find itself in both of these situations simultaneously (albeit for different tasks). A need for system adaptability would dictate the agent should respecialize, but since both corners have identical probability, alternating choices often take place, resulting in the oscillatory behavior.

To aid respecialization, we propose favoring $(s_t = \theta_{a,t} = 1.0)$ by altering the $P_{s,\theta}$ formula. We hypothesize that an alteration may not only reduce oscillations at the step level but may also help respecialization when existing specializations are not reset to random values beforehand. StimHab probability formula can be adjusted to favor $(s_t = \theta_{a,t} = 1.0)$, that is, to favor the task that agents have specialized against, but which has now reached maximum stimulus due to continuous insufficient activity. Assigning different exponents to stimuli and thresholds will curve the diagonal $P_{s,\theta} = 0.5$ line, but will not alter the point $P_{s=1,\theta=1} = 0.5$ (see the example in Figure 14(a)). In order to favor tasks that fall into the $s = \theta = 1.0$ corner of the probability map, we can assign different weights to the stimulus and threshold components in the probability formula (see the example in Figure 14(b)).

9 Evolving the s $-$ θ relationship

We hypothesize that improvement can be achieved by fine-tuning the relationship between the system's stimuli and the habit thresholds developing over time. To this end, we will employ a simple GA to evolve the exponents and weights for the $P_{s,\theta}$ formula. GAs are a type of optimization algorithms inspired by the mechanisms of natural selection, such as survival of the fittest, genetic mutations, and inheritance

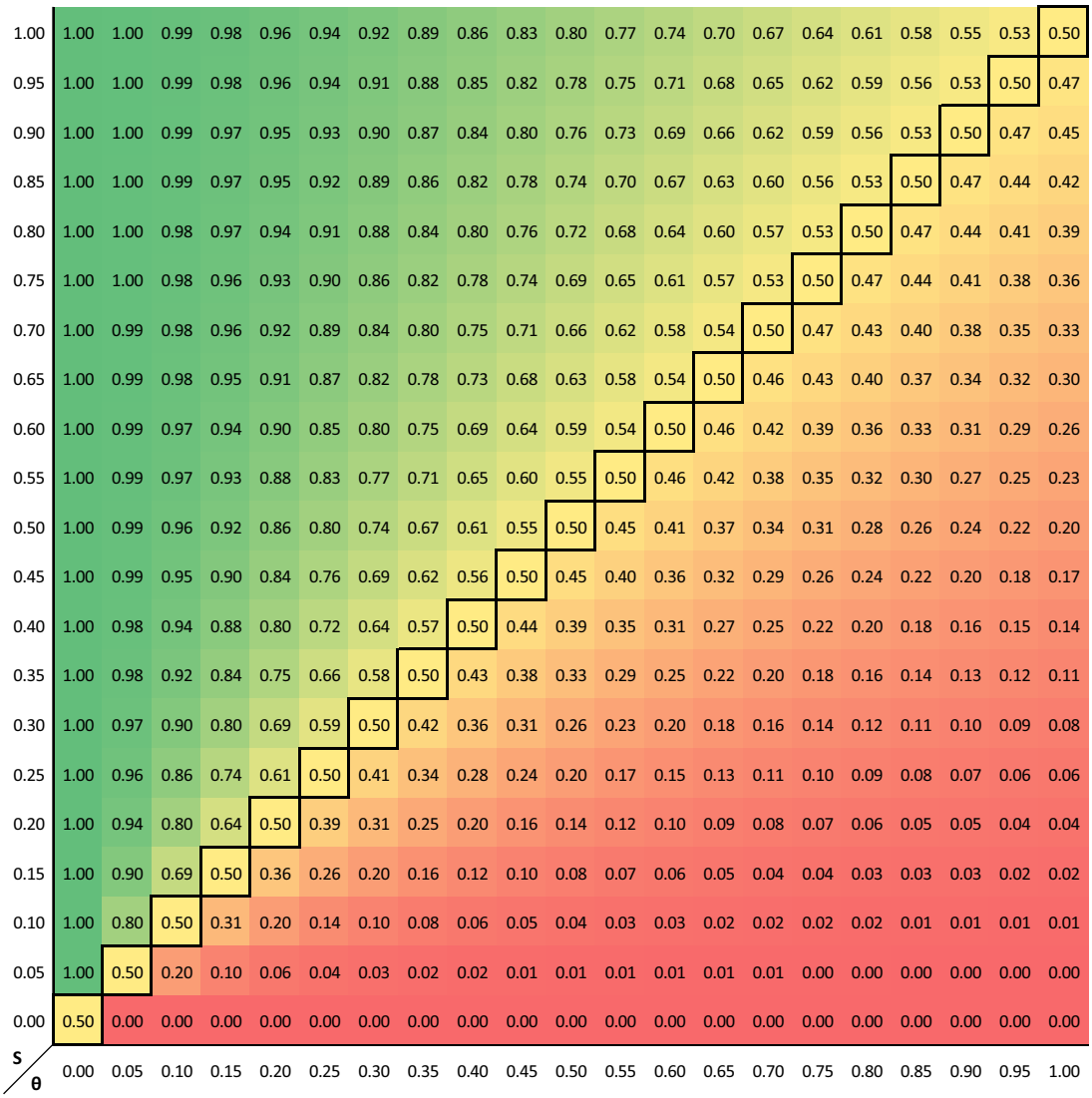

Figure 13 StimHab probability map: $P_{s,\theta} = \frac{s^2}{s^2+\theta^2}$

through gene recombination. GAs have been successfully applied toward a variety of complex optimization problems, such as evolving atom positions within metallic nano-cluster formations (Kazakova *et al.*, 2013), flying drone path planning (Ragusa *et al.*, 2017), and even the evolution of neural network topologies (Stanley & Miikkulainen, 2002). GAs are a subset of evolutionary computation approaches. For an overview of GAs and other evolutionary computation approaches, please see De Jong (2006). In this section, we describe the specifics of our GA design: solution encoding, fitness functions, genetic operators, and evolution parameters. We then compare the performance of the evolved solutions to that of the standard StimHab algorithm.

9.1 Chromosome formulation

For ease of discussion, consider the line where $P_{s,\theta} = 0.5$. Values above the line correspond to higher likelihood of acting, while values below correspond to lower likelihood of acting. Thus, the curvature and placement of this like are crucial to the overall behavior. Our goal is to search the space of $P_{s,\theta}$ mappings by (1) changing the curvature of the $P_{s,\theta} = 0.5$ line by varying the power to which s_t and $\theta_{a,t}$ values are elevated and (2) changing the overall height of the curve to allow for altering the tug-of-war oscillatory behavior observed between the tasks at $(s_t = \theta_{a,t} = 0.0)$ and those at $(s_t = \theta_{a,t} = 1.0)$, that is,

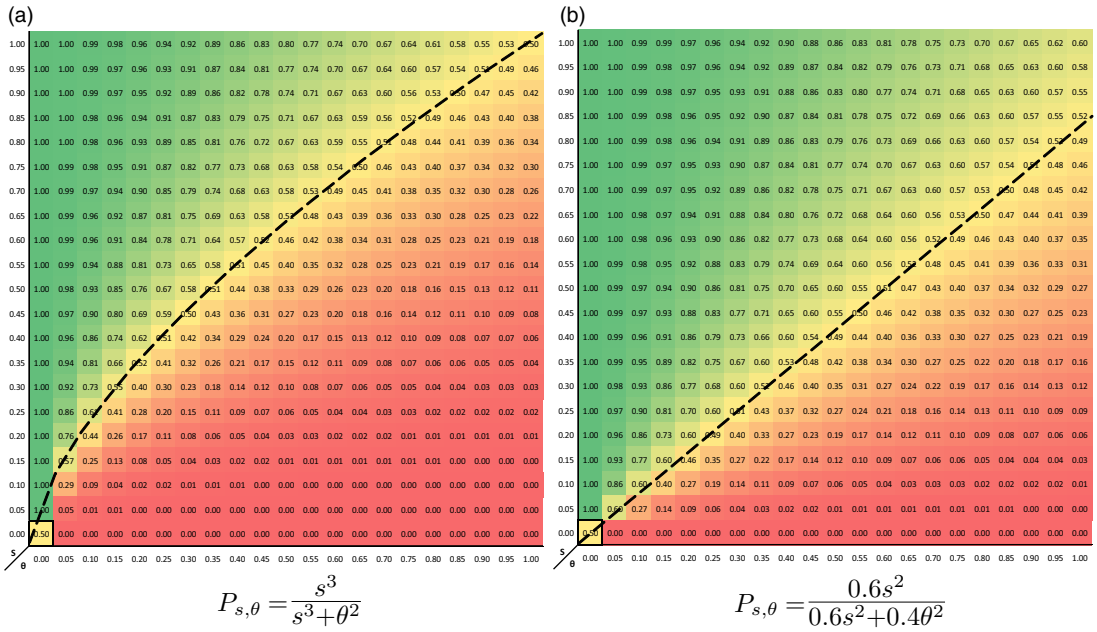

Figure 14 Sample probability maps for reference: how StimHab probabilities change if we alter (a) exponents or (b) weights

tasks that agents have specialized toward but that are not needed versus tasks that agents have specialized against but that are now needed, which under StimHab both result in $P_{s=0,\theta=0} = P_{s=1,\theta=1} = 0.5$

To explore the curvature and height variations of the $P_{s,\theta} = 0.5$ curve, we consider the following variables: n exponent of s_t, m exponent of θ, and w weight of s_t, to calculate new $P_{s,\theta}$ as:

$$P_{s,\theta} = \frac{\mathrm{w} * s^{\mathrm{n}}}{\mathrm{w} * s^{\mathrm{n}} + (1.0 - \mathrm{w}) * \theta^{\mathrm{m}}}$$

where, exponent of s_t n $\in [0.0, 15.0]$, exponent of $\theta_{a,t}$ m $\in [0.0, 15.0]$. With exponents greater than 15.0, the probability map becomes almost entirely filled with values of 0.0 and 1.0, excepting the diagonal, which remains at 0.5. Such a steep landscape may hinder gradual adaptation, so we use 15.0 as the highest exponent our GA will consider during evolution.

Note that we evolve the weight of stimuli s_t to be between 0.0 and 1.0 and calculate the weight of $\theta_{a,t}$ as the remaining 1.0 − (weight of s_t). While the original StimHab places equal weight on stimulus and habit contributions, we allow our GA to decide what the weights should be.

The space of solutions covered by these variables is, however, still larger than will be useful. Consider again the probability maps presented in Figure 13. In order to give priority to the tasks at $P_{s=1,\theta=1}$, the $P_{s,\theta} = 0.5$ diagonal must curve downward. Concavity will occur when the exponent on $\theta_{a,t}$ is smaller than the exponent on s_t. Thus, regardless of the evolved exponent of s_t, the exponent of $\theta_{a,t}$ should be a fraction of it. Consequently, the following three values are evolved instead: weight of s_t w, exponent of s_t n, and a scaling coefficient θ k that will be used to calculate the exponent m from the evolved exponent n. The new formula is then

$$P_{s,\theta} = \frac{\mathrm{w} * s^{\mathrm{n}}}{\mathrm{w} * s^{\mathrm{n}} + (1.0 - \mathrm{w}) * \theta^{\mathrm{k}*}}$$

where weight of s_t w $\in [0.0, 1.0]$, exponent of s_t n $\in [0.0, 15.0]$, and m is calculated as k ∗ n, with the new scaling factor of exponent of s_t k $\in [0.0, 1.0]$, which reduces the space of solutions to be searched by the GA. Additionally, as the relative scale of the exponents is what dictates the curvature of the $P_{s,\theta} = 0.5$ line, we expect this scaling factor k to be more indicative of higher performing solutions

than the actual evolved n and calculated m exponents themselves. Thus, our evolved chromosomes each consist of a three-real-valued gene vector: $\langle n, k, w \rangle$.

9.2 Fitness function

To allow better solutions to survive from one generation to the next and to share 'genetic' information (i.e., the values of the evolving variables), GAs require a *fitness function*, that is, a formula for calculating the relative or absolute quality of each solution. In this section, we discuss our evolutionary goals, quantify the dimensions by which we measure the quality of each candidate solution in our population, and then provide a set of five fitness functions that could be of interest for the decentralized allocation of ongoing tasks.

The decentralized task allocation domain we are addressing involves the simultaneous optimization of two objectives: to have all ongoing tasks to be continuously performed exactly as much as they are needed, while reducing the amount of superfluous task switching. We are not interested in the extremes, that is, in neglecting task fulfillment in favor of minimizing task switching nor in optimizing task fulfillment while task switching freely. Instead, we seek a balance between the two performance metrics. Given our two main task allocation quality indicators, task performance and task switching, to quantify the quality of our evolved solutions, we establish a formula for penalizing each approach for (1) deviating from the ideal performance as expected from task demand values and (2) for any amount of task switching (although some switches are inevitable during readaptation after demands change).

- *penalty-DEV: average performance deviation* (allocation quality)

$$\frac{\sum_{\text{first step}}^{\text{last step}} |\text{deviation from 100\% perfromance this step}|, \text{averaged across tasks}}{\text{number of steps in simulation}}$$

- *penalty-TS: average task-switching rate* (allocation stability)

$$\frac{\sum_{\text{first step}}^{\text{last step}} \text{task switches this step/agent count representing maximum switches possible}}{\text{number of steps in simulation}}$$

In order to facilitate the comparison of our evolving solutions, we combine these two penalties into a single value that will represent the fitness of each solution. One intuitive way to combine the two penalty components is to view them as x and y components of a solution vector, the magnitude of which becomes the solution fitness to be optimized (see Figure 15):

- *overall penalty: composite solution fitness*

$$\text{fitness} = \sqrt{(\text{penalty-DEV})^2 + (\text{penalty-TS})^2}$$

  composite of penalty-DEV and penalty-TS, such that minimizing either of both or the penalties reduces the overall resulting fitness. Solutions with smaller fitnesses will thus represent lower under- or overworking and/or reduced task switching than solutions with higher fitnesses.

The GA is, therefore, tasked with minimizing both penalty components by minimizing the fitness of the evolved solutions. The average task switching rate is within 0–100%, as are most average performance deviations (although higher is possible, e.g., 30 agents on a task that needs 10 corresponds to a 200% deviation), keeping the two components generally on the same scale.

The theoretical optimal performance in our environment with multiple dynamic task needs is to have agents act on the task where the work is needed and to remain there for as long as it is needed. Without a centralized controller and no information about the choices of other agents, the only source of information is the task stimuli changes over time. Thus, we assume that a period of adaptation after task demands change is inevitable. Nevertheless, there are trade-offs when it comes to the adaptation speed, task performance fluctuations, and the quality of the new resulting task allocation.

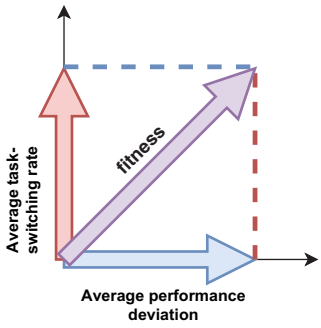

Figure 15 Fitness function components

With these trade-offs in mind, we consider several possible fitness functions, each designed to favor a specific potentially desirable behavior. Considering the general applicability of decentralized task allocation, we want to assess whether a variety of potentially desirable dynamic task allocation behaviors can be evolved. The following five adaptation behaviors are defined and encoded as fitness functions, to be used by our GA in five separate evolutionary runs:

(i) *Arithmetic average fitness.* Consider only the quantity of performance deviations and task switches, without any regard for when the values occurred. Thus, penalty components are calculated exactly as defined above, as a direct arithmetic average of the values.

(ii) *Timing-weighted average fitness.* Deficiencies in behavior that happen further in time from a demand switch carry higher penalty. Consider that as demands change, agents need time to adjust, which inevitably leads to a period of higher performance deviations and higher task switching. Over time agents should respecialize, adapting to the new system needs, ideally reaching 0% performance deviations and 0% task switching. To evolve solutions that adapt quicker to system changes, we employ weighted averaging, where each step's average task deviation is given a $\text{weight}_{step} = (numberofstepssincedemandschanged)^2$. Thus, in a 15 000-step simulation with demands changing every 5000 steps, the weights for steps 1 through 15 000 are [1, 2, 3, ...5000, 1, 2, 3, ...5000, 1, 2, 3, ...5000]. These weights are applied both to the average task performance deviations and to the task switch rates.

(iii) *Improvement-discounted fitness.* Any performance deficiencies are not penalized as long as there is improvement as compared to the previous step. While the previous fitness (ii) incites faster adaptation, speed may hinder quality. In this fitness formulation, we consider that some systems may be tolerant of longer adaptation periods that ultimately lead to a more stable task allocation, such as systems where demands changes less frequently. To evolve solutions that approximate the '(possibly) slowly but surely' approach the ideal behavior, we compare each step's penalty to that of the previous step and set it to zero if the new penalty is lower. This is done for both the average task performance deviations as well the average task switch rates, independently.

(iv) *Acceptable-range-discounted fitness.* Performance deficiencies within some small range are not penalized. Here, we consider systems where small deviations from ideal performance are acceptable on a continuous basis. In some domains, precise task allocation may not be as vital as faster adaptation or prevention of big changes in performance. To evolve solutions that allow for a predefined range of tolerable deviations from the ideal behavior, we compare each step's penalty to a predefined acceptable value (e.g., 0.05, representing a 5% non-penalty range from 0% task performance deviations and from 0% task switching). Note, however, that such acceptability is a per-task quality, and therefore, it is the raw per-task deviations that are compared to the acceptable value and set to zero before the average task deviation for the step is calculated (this is in contrast to the previous fitness (iii), as stabilization is a system-wide quality).

(v) *Smaller-value-discounted fitness.* Small performance deficiencies are penalized less than larger deficiencies. In general, smaller deviations from the ideal behavior are more tolerable than larger fluctuations. In some cases, especially during initial adaptation to changes in demand, performance

on a task can jump to above 2.0 (i.e., above 200% of the needed work). Such jumps may be particularly detrimental in certain domains. To evolve solutions that increase penalty for values further from the ideal behavior, we employ a predefined exponent to scale the penalties. Specifically, we cube each step's average performance deviation and average task switching rate before calculating the final fitness, making the small-valued penalties smaller and the large penalties larger.

These fitness functions provide relative quality of the solutions within each population with respect to the goal assumed by each fitness function and thus should not be directly compared across fitness functions. Instead, to compare the best evolved solutions to each other and the StimHab baseline, we assess their average adaptation behavior by comparing (1) the graphs of average deviations from ideal task performance per step and (2) the graphs of average task switching per step. To provide a better sense of the actual behaviors, the unaveraged per-task per-step deviations and task switching are also provided.

9.3 Genetic operators and parameters

GAs employ a variety of evolutionary techniques inspired by natural evolution. In this section, we describe our general evolution settings, as well as the specific elitism, mutation, selection, and crossover operators employed by the GA.

For our search, we setup a population of 30 randomly generated candidate solutions, to be mutated and recombined the course of 30 generations. To facilitate comparison with experiments in 7 and 8, we use the same hierarchical task set to evolve behavior (tasks and their demands per time period are provided in Table 1). A separate evolution is performed twice for each of the five fitness functions, once with and once without resetting of agent specializations when demands change. Each test is conducted using a team of 100 agents, as evolving with 1000 agents in our sequential simulation setup would take considerably longer. We also later test the evolved approaches on a different task set to assess generalization capabilities of the evolved solutions.

To ensure that best found solutions are not lost over the course of evolution, we also employ *elitism*: the top performing 10% of chromosomes of each generation are automatically copied into the next generation. This is on the higher end of elitism rates, but we expect there to be some variety of equivalently good solutions and thus must allow for a larger set of good solutions to persist through the generations.

We employ two types of real-valued *mutation* to balance large exploratory changes with smaller fine-tuning changes. Each chromosome in the current generation has a 25% chance to be mutated. Any time a chromosome is mutated, it will undergo either an exploratory or a local mutation, with equal probability. (I) The *exploratory mutation* considers each gene and mutates it with $1/(\text{number of genes}) = 1/3$ probability, by adding an amount determined by a random Gaussian variable with zero mean and a standard deviation of 15% of the gene's range. Note that all genes could be simultaneously mutated here, but none is also possible. (II) The *local mutation* selects a single gene and adds to it a value determined by a random Gaussian variable with zero mean and a standard deviation of 5% of the gene's range. Exactly one gene is locally mutated at a time, ensuring that each change can be evaluated independently by the algorithm.

After elites and mutated chromosomes are copied into the next generation, the remainder of the new generation's chromosomes are generated by genetic recombination. We employ *rank proportional selection* to choose parents for crossover to balance exploration with exploitation through the lowering of selection pressure (Baker, 1985). Solutions are sorted by their fitness values and given a rank, with the lowest fitness receiving the highest rank, as we are minimizing. Then, a *roulette-wheel* selection is performed on the chromosome ranks instead of on the raw fitness values. Each chromosome_i thus has a chance to be selected for crossover equal to:

$$(\text{Ranked selection probability})_{chromosome_i} = \frac{\text{rank of } chromosome_i}{\text{sum of all ranks}} = \frac{\text{rank}_i}{\sum_{r=1}^{popSize}(r)}$$

where *popSize* is the number of candidate solutions (i.e., chromosomes) in each generation.

Once two distinct parents are selected for *crossover*, their chromosomes are recombined via a weighted averaging of each of their real-valued genes. During each crossover, a random value between 0.0 and 1.0 is selected to indicate what proportion of each gene will come from the first parent, with the remaining portion being contributed by the second parent as follows:

given $weight_1 = \text{random} \in (0.0, 1.0)$, $weight_2 = 1.0 - weight_1$, the resulting progeny is:

$$child: \left\langle \left(weight_1 * \mathrm{n}_1 + weight_2 * \mathrm{n}_2\right), \left(weight_1 * \mathrm{k}_1 + weight_2 * \mathrm{k}_2\right), \left(weight_1 * \mathrm{w}_1 + weight_2 * \mathrm{w}_2\right) \right\rangle$$

We do not weight the genes based on relative fitness, to avoid adding selection pressure beyond that of the initial parent selection, so as to protect our GA from early convergence. Additionally, through the use of the random weight component, even if the same parents are selected for crossover again, a new child may be generated, thus promoting exploration. Pairs of parents are selected repeatedly until the new population is full.

10 Experiments, part II: GA evolution results

In this section, we review the best GA solutions found by each fitness function and compare them to standard StimHab as a baseline. We consider evolution with and without specialization resets separately, as they are amenable to different task set changes, thus applying to different domains.

10.1 Evolving without resetting prior specializations

We test respecialization (specialization from previously adapted thresholds) by evolving and testing solutions without resetting agents' $\theta_{a,t}$ when demands change. For the existing per-agent per-task thresholds to be at all applicable (even if outdated) across changes in system demands, the tasks and their hierarchical structure must remain the same across demand periods. Thus, the approaches presented in this section are evolved on a task set with a static hierarchy and dynamic demands, employing a team of 100 agents, over 10 demand periods. To assess generalization and scalability, the approaches are later retested on a new static hierarchy and a team of 1000 agents.

In Table 2, we list the best solutions evolved with each fitness function; baseline StimHab is listed for comparison. All evolved solutions have a much higher s_t exponent than StimHab, a $\theta_{a,t}$ exponent that is slightly above half of s_t exponent, and over 90% of the weight is given to s_t. To visually compare these relationships, we provide their probability maps in Figure 16, which show that all evolved versions have an almost binary breakdown of action probabilities, as well as concave downward $P_{a,t} = 0.5$ lines (shown in yellow). Comparing upper-right quarters of the maps, agents under the evolved solutions are guaranteed to act at higher stimuli, regardless of thresholds. Additionally, we see a clear preference for moving the $P_{a,t} = 0.5$ line down and away the $s_t = \theta_{a,t} = 1.0$ corner, allowing it to be favored over the $s_t = \theta_{a,t} = 0.0$, which we hypothesized would benefit respecialization (Section 8). Comparing lower-left quarters, we see that under evolved solutions agents will not act at lower stimuli, unless they are maximally specialized ($\theta_{a,t} = 0.0$) toward that task. Overall, in the evolved solutions, all finer probability changes are restricted to a much narrower area, making agents actions less probabilistic and closer to a step function.

First, we compare how these solutions handle the original *training set:* 100 agents and a static task hierarchy with dynamic demands, used for fitness assessments during evolution. The tasks and demands, provided in Table 3, match our earlier per-cycle tests, although per-step behavior is considered here, as it provides a more detailed view. Results are presented in Figure 17: average task performance deviations shown in the LEFT column and average task switching rates shown in the RIGHT column. While values will spike as demands change and agents begin readapting, we want to see a decrease over time, indicating that the correct agent allocation is achieved (average deviation near 0%) and remains stable over time (average task switching rate near 0%). The x-axis represents average steps during adaptation (each nth step since demands changed); the y-axis shows performance deviations in graphs on the LEFT and task switch rates on the RIGHT. We see no significant difference among the evolved solutions, but each

Table 2 StimHab baseline versus best solutions evolved **without** specialization resets (The extended $P_{s,\theta}$ formula is reiterated on the right-hand side, for reference.)

	FITNESS FUNCTION	n	k	w
WITH RESETTING	(baseline) StimHab:	2.000	1.000	0.500
	(i) Arithmetic average:	0.424	0.130	0.995
	(ii) Timing-weighted:	13.885	0.399	0.385
	(iii) Improvement:	13.872	0.394	0.988
	(iv) Acceptable range:	14.032	0.693	0.579
	(v) Smaller value:	11.915	0.632	0.606

$$P_{s_t,\theta_{a,t}} = \frac{w * s_t^{\,n}}{w * s_t^{\,n} + (1.0 - w) * \theta_{a,t}^{\,k*n}}$$

Table 3 TRAINING SET for evolution **without** resets: task demands for a static task hierarchy; demands change every 5000 steps, 10 times per simulation; values match those used in our earlier test

STEPS	T1	T2	T2.1	T2.2	T2.3	T3	T4	T4.1	T4.2	T4.2.1	T4.2.2	T5
1–5000	4%	20%	4%	8%	8%	32%	20%	4%	16%	8%	8%	24%
5001–10 000	4%	16%	8%	4%	4%	24%	32%	8%	24%	12%	12%	24%
10 001–15 000	16%	40%	4%	20%	16%	4%	12%	4%	8%	4%	4%	28%
15 001–20 000	24%	12%	4%	4%	4%	28%	16%	4%	12%	4%	8%	20%
20 001–25 000	4%	32%	12%	8%	12%	16%	36%	12%	24%	8%	16%	12%
25 001–30 000	36%	16%	4%	4%	8%	4%	20%	8%	12%	4%	8%	24%
30 001–35 000	16%	16%	4%	4%	8%	8%	12%	4%	8%	4%	4%	48%
35 001–40 000	56%	12%	4%	4%	4%	4%	20%	12%	8%	4%	4%	8%
40 001–45 000	12%	16%	8%	4%	4%	4%	52%	36%	16%	12%	4%	16%
45 001–50 000	32%	24%	8%	12%	4%	20%	12%	4%	8%	4%	4%	12%

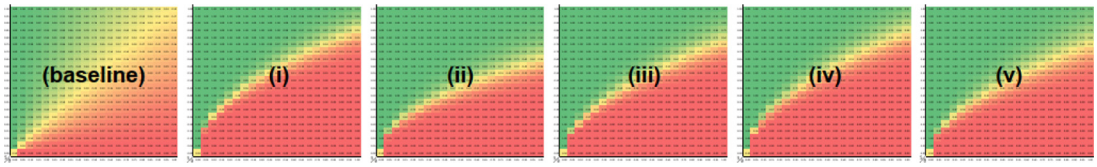

Figure 16 Probability maps: baseline versus best evolved **without** resets (same format as Figure 13)

greatly outperforms the StimHab baseline in both average deviations and average task switching rates. Some spiking long after demands changed are present in all solutions, meaning that agents are not able to find the exact correct assignments and remain there, but they appear to get quite close. The interested reader can refer to Appendix A-I for graphs of the actual per-task deviations (Figure A1) and the overall task switching rate (Figure A2) for each of the 50 000 simulation steps of each approach.

Next, we assess whether the evolved solutions can scale to larger teams and generalize toward new task hierarchies. For our *testing set*, we use 1000 agents and a new randomly generated static hierarchy with dynamic demands, provided in Table 4. Results are provided in Figure 18. Again, the evolved solutions perform similarly across all fitness functions. Additionally, when comparing training and testing set results (Figures 17 vs. 18), we see that StimHab performs better on the testing set, but the averages of

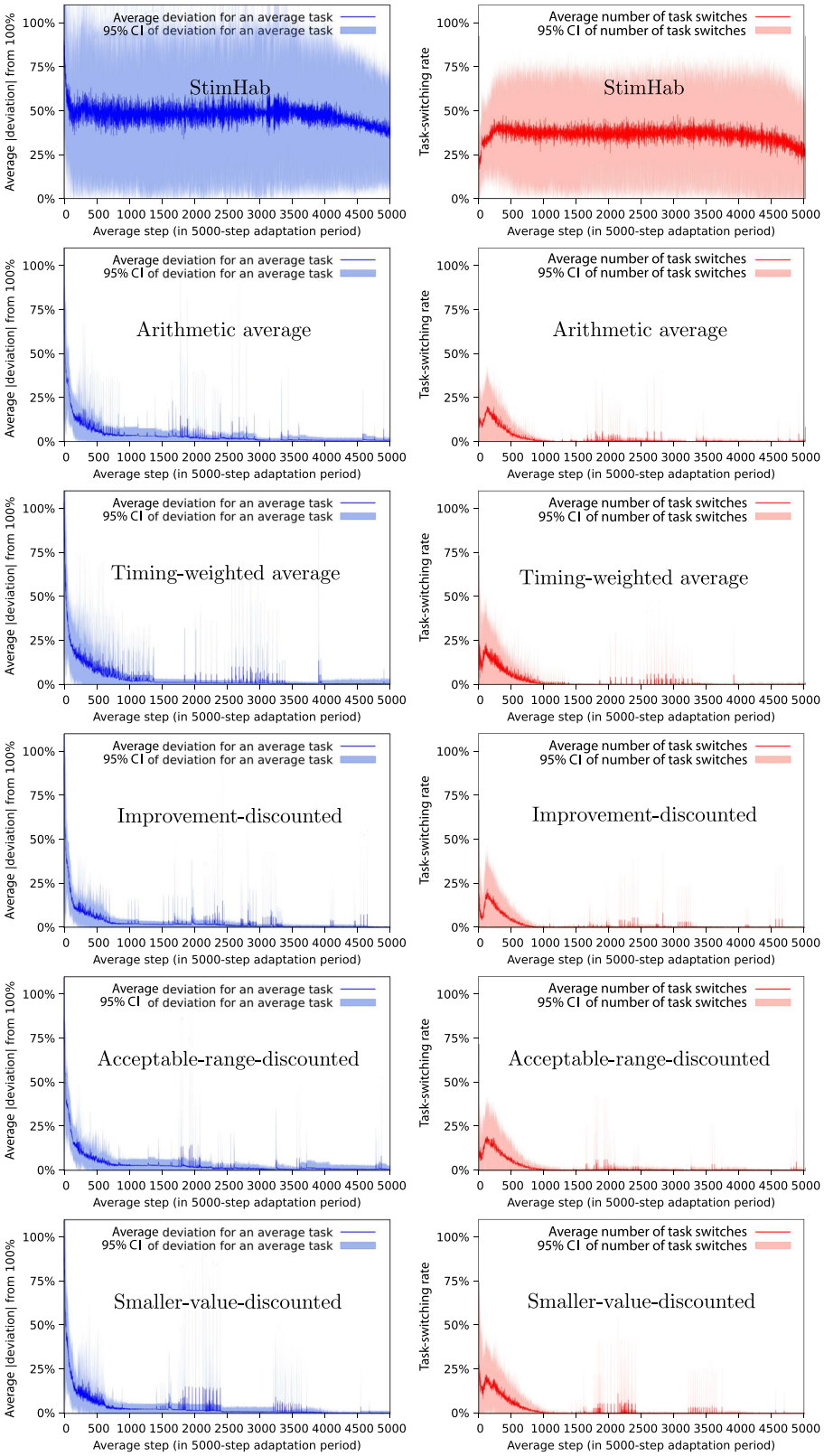

Figure 17 TRAINING SET for evolution **without** resetting: average adaptation to demand changes % deviations from work needed (LEFT) and % of agents that switched tasks (RIGHT)

Table 4 TESTING SET for solutions evolved **without** resets: per-task dynamic demands for a static task hierarchy. Demands change every 5000 steps, 10 times per simulation. This static task hierarchy differs from the one used during evolution, to test for potential overfitting

STEPS	T1	T1.1	T1.1.1	T1.1.2	T1.1.2.1	T1.1.2.2	T1.1.2.2.1	T1.1.2.2.2	T1.1.2.2.3	T1.2	T2	T2.1	T2.2
1–5000	93.5%	71.6%	21.0%	50.6%	13.9%	36.7%	20.0%	13.0%	3.7%	21.9%	6.5%	5.0%	1.5%
5001–10 000	81.0%	71.9%	11.2%	60.7%	6.4%	54.3%	42.0%	5.3%	7.0%	9.1%	19.0%	4.5%	14.5%
10 001–15 000	89.7%	83.5%	24.0%	59.5%	4.6%	54.9%	42.2%	10.0%	2.7%	6.2%	10.3%	7.0%	3.3%
15 001–20 000	90.2%	86.6%	14.3%	72.3%	40.8%	31.5%	18.1%	7.0%	6.4%	3.6%	9.8%	4.7%	5.1%
20 001–25 000	88.1%	82.9%	35.1%	47.8%	24.5%	23.3%	7.3%	4.0%	12.0%	5.2%	11.9%	6.5%	5.4%
25 001–30 000	90.4%	50.4%	28.8%	21.6%	6.5%	15.1%	4.5%	6.1%	4.5%	40.0%	9.6%	3.4%	6.2%
30 001–35 000	88.6%	63.6%	15.2%	48.4%	29.4%	19.0%	6.2%	4.4%	8.4%	25.0%	11.4%	4.2%	7.2%
35 001–40 000	66.0%	41.8%	7.1%	34.7%	14.2%	20.5%	4.0%	13.0%	3.5%	24.2%	34.0%	2.0%	32.0%
40 001–45 000	87.5%	62.5%	2.5%	60.0%	30.0%	30.0%	8.5%	9.5%	12.0%	25.0%	12.5%	8.8%	3.7%
45 001–50 000	62.5%	54.0%	3.8%	50.2%	15.0%	35.2%	30.0%	1.0%	4.2%	8.5%	37.5%	7.5%	30.0%

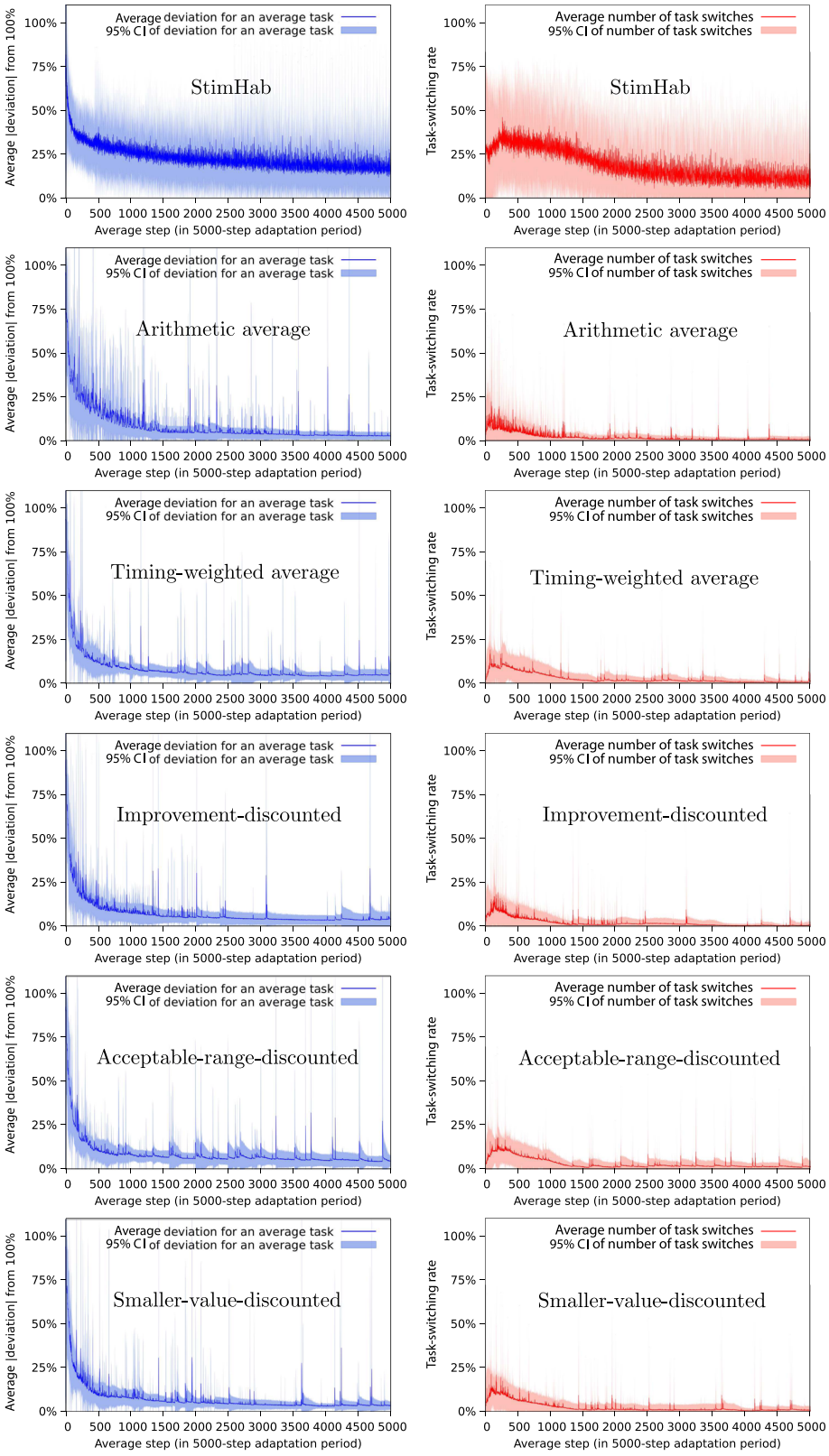

Figure 18 TESTING SET for evolution **without** resetting: average adaptation to demand changes % deviations from work needed (LEFT) and % of agents that switched tasks (RIGHT)

Table 5 Baseline StimHab versus best solutions evolved **with** specialization resets (The extended $P_{s,\theta}$ formula is reiterated on the right-hand side, for reference.)

	Fitness function	n	k	w
WITH RESETTING	(baseline) StimHab:	2.000	1.000	0.500
	(i) Arithmetic average:	0.424	0.130	0.995
	(ii) Timing-weighted:	13.885	0.399	0.385
	(iii) Improvement:	13.872	0.394	0.988
	(iv) Acceptable range:	14.032	0.693	0.579
	(v) Smaller value:	11.915	0.632	0.606

$$P_{s_t,\theta_{a,t}} = \frac{w * s_t^{\,n}}{w * s_t^{\,n} + (1.0 - w) * \theta_{a,t}^{\,k*n}}$$

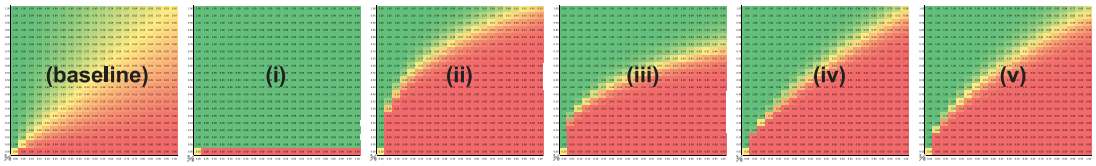

Figure 19 Probability maps: baseline versus best evolved **with** resets (same format as Figure 13)

the evolved solutions are still better. Deviation spikes in the evolved solutions may seem to imply lower stability than the baseline but recall that we are looking at the behavior of an average task. Observing the actual per-task fluctuations in Figure A3 (Appendix A-I) can provide a clearer view: StimHab exhibits continuous spiking across all tasks, while the evolved solutions' spikes are concentrated to a few steps. Per-step task switching is provided in Figure A4 (Appendix A-I). Recall that given no explicit penalty for task switching in these experiments, performance is not directly affected. If there was an activity delay with each task switch, higher task switching under StimHab would further increase its performance deviations, widening the task allocation quality gap between the baseline and the evolved versions.

Overall, results show that more adaptable $s-\theta$ relationships for $P_{s,\theta}$ calculations are possible and can be successfully evolved with a variety of fitness functions. As evolved solutions performed slightly better on the training set, it may be possible to evolve highly tailored solutions. Nevertheless, the testing set simulations also reach low-performance deviations and task switching, while vastly outperforming the baseline, indicating an ability to generalize. Additionally, recall from earlier tests (Figures 2 and 3) that resetting was crucial for respecialization. In the evolved solutions, no resetting takes place, demonstrating that the newly defined and evolved parameters obviate the need for resetting specializations when the task hierarchy remains static.

10.2 Evolving with resetting prior specializations

We now test the GA's ability to evolve solutions that allow agents to task allocate effectively when presented with entirely new task sets each time. We conduct a series of evolution experiments where agents' $\theta_{a,t}$ thresholds are reset every time demands change. As all specializations get reset, there is no requirement for the task set to remain consistent across demand switches, thus giving us the freedom to evolve solutions using randomly generated task hierarchies for each demand period. Thus, the approaches presented in this section are evolved on sets of randomly generated task hierarchies and demands, using a team of 100 agents, over 10 demand periods. To assess generalization and scalability, these approaches are later tested on a new set of randomly generated hierarchies and a team of 1000 agents.

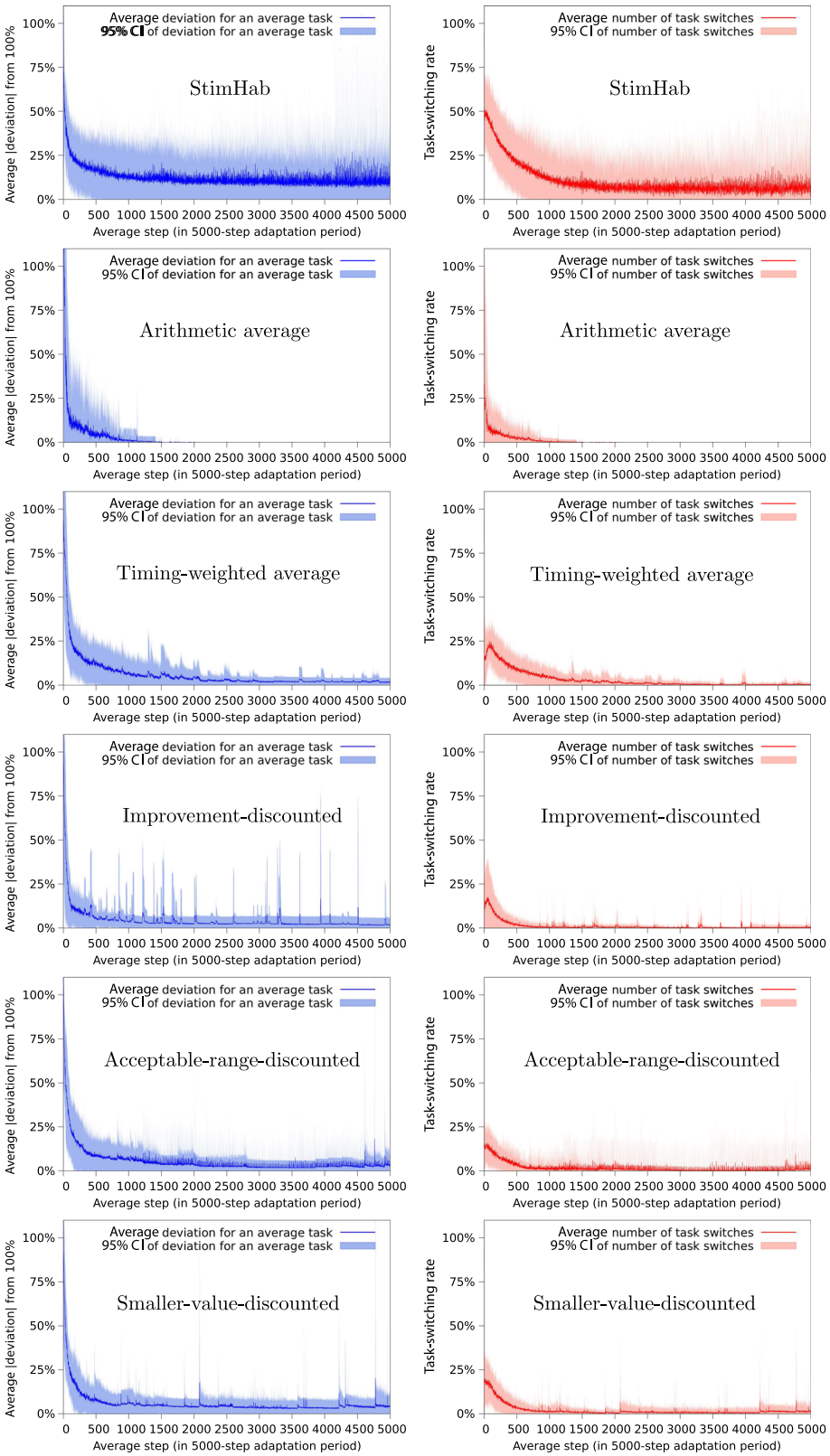

Figure 20 TRAINING SET for evolution **with** resetting: average adaptation to task changes % deviations from work needed (LEFT) and % of agents that switched tasks (RIGHT)

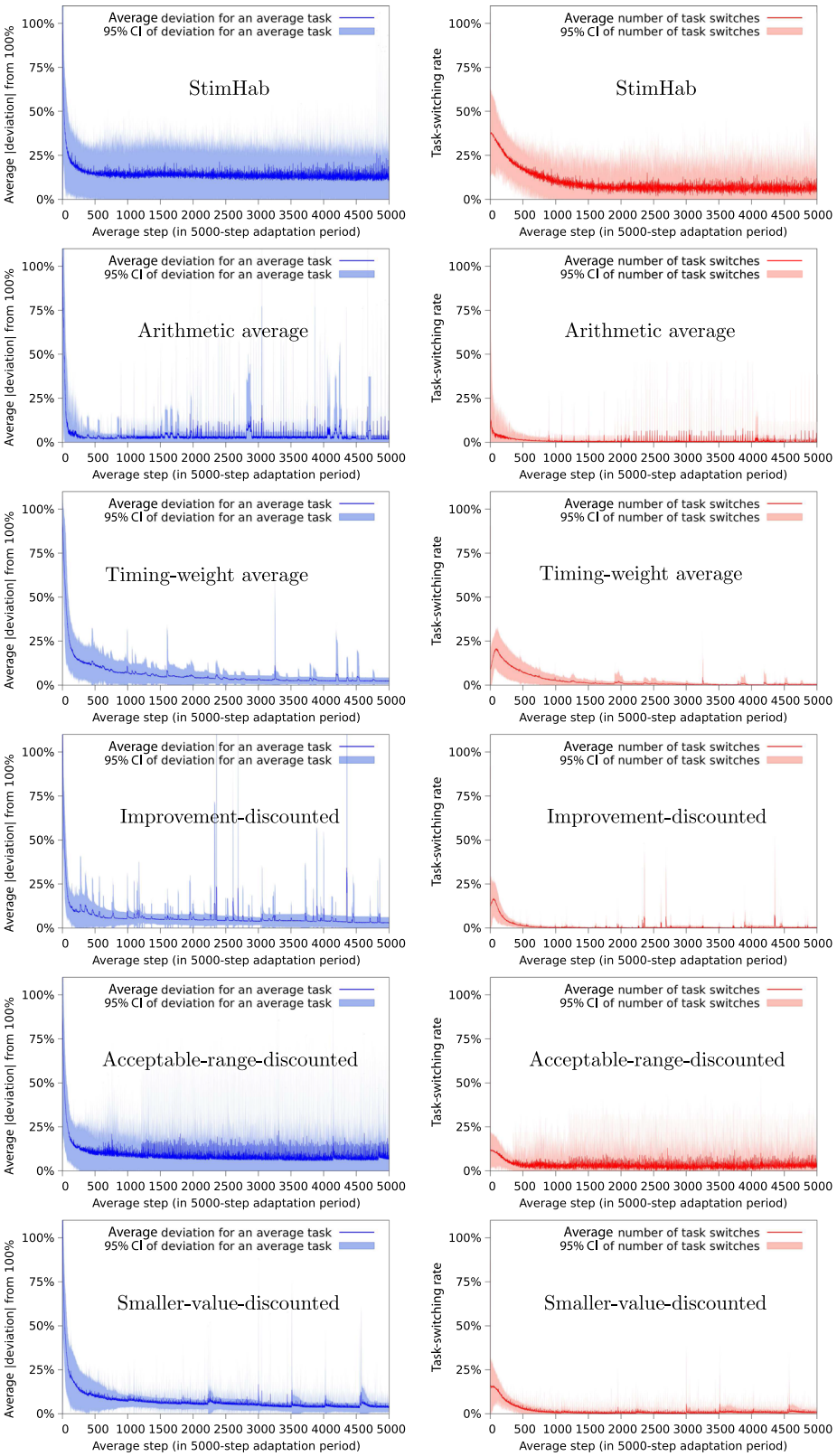

Figure 21 TESTING SET for evolution **with** resetting: average adaptation to task changes % deviations from work needed (LEFT) and % of agents that switched tasks (RIGHT)

In Table 5, we list the best solution vectors evolved with each fitness function (StimHab baseline is also listed, for reference). Unlike with our earlier solution vectors, there is no clear pattern across the evolved values. Particularly interesting is the vector evolved using the arithmetic averaging fitness function, which employs a considerably smaller s_t exponent (0.424 vs. 9.443–14.125 in the other vectors) and a $\theta_{a,t}$ exponent that is only 13% of the s_t exponent (as compared to 39.9–69.3% in the other vectors). To visually compare the evolved relationships, we provide the corresponding probability maps in Figure 19. First, we see more variation in the $P_{a,t} = 0.5$ curve (shown in yellow) than in the no-reset solutions (Figure 16), as this time not all solutions move the line away from the top-right corner $s_t = \theta_{a,t} = 1.0$. Recall that our initial motivation for the GA was to evolve parameters that favor the $s_t = \theta_{a,t} = 1.0$ corner over the bottom-left corner $s_t = \theta_{a,t} = 0.0$, since under standard StimHab both pairings lead to $P_{a,t} = 0.5$, hindering agent respecialization (Section 8). As these solutions are tested with resetting, no respecialization takes place, with agents specializing from random $\theta_{a,t}$ each time demands change. Comparing lower-left quarters, we see that under evolved solutions agents will not act at lower stimuli, unless they are maximally specialized ($\theta_{a,t} = 0.0$) toward that task. Once again, in all evolved solutions, the finer probability changes are restricted to a narrow line, making agents actions less probabilistic and closer to a step function. The most extreme case is the arithmetic average solution (i), which appears to result in $P_{a,t} = 1.0$ for all $s_t > 0$. Looking closer, however, there are small differences in probabilities, ranging from 0.9824 in the bottom-right to 1.0 at the top-left (a detailed view is available in Appendix C, Figure C1). The result is that (1) the tiny differences sort agents' tasks by giving most preference to full specialization ($\theta_{a,t} = 0.0$), then ordered by decreasing need (high to low s_t), and finally with a small focus on other specialization levels (low to high $\theta_{a,t} > 0.0$) and (2) the first task in this ordering will be acted on with a near 100% chance.

We first compare how the evolved solutions handle the initial *training set:* 100 agents and dynamic task sets identical to those used during evolution fitness testing, provided in Table 6 (note that we cannot follow the earlier column format, as the tasks for each demand period now differ). Per-step results are presented in Figure 20, with average task performance deviations shown in the LEFT column and average task switching rates shown in the RIGHT column. Recall that values near 0% are best in both cases and successful adaptation corresponds to averages decreasing quickly over the depicted 5000-step average adaptation period. As expected from our experiments in Section 7, baseline StimHab performance deviations and task switching are lower when specializations are reset between changes in demand (compare StimHab in Figures 17 vs. 20). Nevertheless, all of the evolved solutions still significantly outperform the baseline. Arithmetic average showcases ideal behavior, with fast and stable adaptation, as indicated by performance deviations and task-switching rates dropping to 0% early in the adaptation period. It is evident that its unique values (Table 5) and resulting probability mapping (Figures 19(i) and C1) are highly beneficial for this training set, which could suggest overfitting, but also the viability of highly domain-tailored solutions. All other evolved solutions perform similarly. While the spiking in the improvement-discounted solution may seem potentially worse than even the more consistent baseline behavior, this is once again a result of graphing performances of an average task and how concentrated spiking deviations are across the steps. Based on the unaveraged behavior provided in Appendix B-I, Figures B1 and B2, we see that there is no clear difference among the remaining evolved solutions, while the baseline shows significantly more per-task performance and task- switching fluctuations throughout.

Next, we assess whether the evolved solutions can scale to larger teams and generalize to new changing task hierarchies. For our *testing set* presented in Table 7, we use 1000 agents and a new randomly generated task hierarchy for each demand period. Results are shown in Figure 21. When comparing the earlier training and these testing set results (Figures 20 vs. 21), we see that performance on the testing set is worse, with more frequent spiking, less obvious improvement over time, and higher averages, although still better than StimHab in all cases. StimHab appears to oscillate indefinitely at higher averages after a small initial adaptation. The arithmetic average solution performs significantly worse than for the training set, supporting our earlier overfitting hypothesis. It is likely that when a more general solutions is required, evolution will need longer fitness tests, with a larger variety of task sets. Despite overfitting, improvement over baseline StimHab is easily found. Looking at the actual task performances

Table 6 TESTING SET for evolution **with** resets: per-task demands for a dynamic task hierarchy. The entire task set changes every 5000 steps, 10 times per simulation

Steps	Training Dynamic Task Sets and Demands
1–5000	T1 92.0%, T2 8.0%
50 01–10 000	T1 72.0%, T1.1 48.0%, T1.1.1 20.0%, T1.1.2 24.0%, T1.1.2.1 4.0%, T1.1.2.2 4.0%, T1.1.2.3 12.0%, T1.1.2.4 4.0%, T1.1.3 4.0%, T1.2 24.0%, T1.2.1 20.0%, T1.2.2 4.0%, T2 28.0%, T2.1 4.0%, T2.2 4.0%, T2.3 20.0%
10 001–15 000	T1 24.0%, T1.1 16.0%, T1.1.1 4.0%, T1.1.2 8.0%, T1.1.3 4.0%, T1.2 8.0%, T2 76.0%, T2.1 32.0%, T2.1.1 4.0%, T2.1.2 16.0%, T2.1.2.1 8.0%, T2.1.2.2 4.0%, T2.1.2.3 4.0%, T2.1.3 12.0%, T2.1.3.1 4.0%, T2.1.3.2 8.0%, T2.2 4.0%, T2.3 32.0%, T2.3.1 24.0%, T2.3.2 8.0%, T2.4 8.0%
15 001–20 000	T1 28.0%, T1.1 4.0%, T1.2 12.0%, T1.3 12.0%, T2 16.0%, T2.1 4.0%, T2.2 4.0%, T2.3 8.0%, T3 24.0%, T3.1 16.0%, T3.2 8.0%, T4 32.0%
20 001–25 000	T1 72.0%, T2 28.0%
25 001–30 000	T1 52.0%, T1.1 32.0%, T1.1.1 8.0%, T1.1.2 12.0%, T1.1.2.1 4.0%, T1.1.2.2 8.0%, T1.1.3 12.0%, T1.2 12.0%, T1.3 4.0%, T1.4 4.0%, T2 48.0%, T2.1 32.0%, T2.1.1 16.0%, T2.1.2 16.0%, T2.1.2.1 8.0%, T2.1.2.2 4.0%, T2.1.2.3 4.0%, T2.2 8.0%, T2.3 8.0%
30 001–35 000	T1 48.0%, T1.1 36.0%, T1.2 8.0%, T1.3 4.0%, T2 52.0%, T2.1 4.0%, T2.2 8.0%, T2.3 40.0%, T2.3.1 4.0%, T2.3.2 20.0%, T2.3.3 16.0%, T2.3.3.1 4.0%, T2.3.3.2 8.0%, T2.3.3.3 4.0%
35 001–40 000	T1 60.0%, T2 40.0%, T2.1 4.0%, T2.2 8.0%, T2.3 28.0%, T2.3.1 12.0%, T2.3.1.1 4.0%, T2.3.1.2 8.0%, T2.3.2 8.0%, T2.3.3 8.0%
40 001–45 000	T1 28.0%, T2 72.0%, T2.1 36.0%, T2.2 36.0%, T2.2.1 20.0%, T2.2.1.1 12.0%, T2.2.1.1.1 8.0%, T2.2.1.1.2 4.0%, T2.2.1.2 4.0%, T2.2.1.3 4.0%, T2.2.2 16.0%
45 001–50 000	T1 88.0%, T1.1 20.0%, T1.1.1 4.0%, T1.1.2 4.0%, T1.1.3 8.0%, T1.1.4 4.0%, T1.2 68.0%, T1.2.1 32.0%, T1.2.2 36.0%, T2 12.0%, T2.1 8.0%, T2.2 4.0%

in Figure B3 and task switches in Figure B4, we see that while arithmetic average adapts faster, that adaptation is less stable over time than under timing-weighted average, resulting in more spiking throughout. This distinction matters because in systems where demands change infrequently, slow and steady is beneficial in the long term, while in highly dynamic environments, fast adaptation to acceptable performance can be crucial to reap any specialization benefits. We also see that the acceptable-range-discounted solution has significantly more spiking here than the other evolved solutions, perhaps most evident in its task switching, but all still outperform the baseline.

Overall, results indicate that we can evolve sets of coefficients that easily improve upon the standard StimHab baseline in the general case, but we can also evolve highly specialized (i.e., overfitted) coefficients that will optimize agents' adaptability and task allocation stability for a specific task set with dynamic demands. Note, however, that the $P_{a,t}$ probability formula provides no way for each solution to learn any specific shifts in demand, so overfitting can only adapt the general $s_t - \theta_{a,t}$ relationship. Approaches without resetting appear to benefit the most from evolution, but this may also be due to the fact that they have the most to gain, given the extremely poor performance of standard StimHab during readaptation (compare StimHab against each of the evolved solutions in Figures 17 and 18, for training and testing data, respectively).

Table 7 TRAINING SET for evolution **with** resets: per task demands for a dynamic task hierarchy. The entire task set changes every 5000 steps, 10 times per simulation. These tasks and demands differ from those used during evolution in order to test for potential overfitting

Steps	Testing Dynamic Task Sets and Demands
1–5000	T1 72.0%, T2 20.0%, T2.1 3.3%, T2.2 16.7%, T2.2.1 4.4%, T2.2.2 12.3%, T2.2.2.1 8.0%, T2.2.2.2 4.3%, T3 8.0%
5001–10 000	T1 12.4%, T2 47.6%, T3 15.0%, T4 5.0%, T5 6.1%, T6 5.0%, T7 4.9%, T8 4.0%
10 001–15 000	T1 82.0%, T1.1 63.0%, T1.2 9.5%, T1.3 9.5%, T2 18.0%
15 001–20 000	T1 76.0%, T1.1 12.0%, T1.2 64.0%, T1.2.1 30.0%, T1.2.2 34.0%, T1.2.2.1 4.0%, T1.2.2.2 10.0%, T1.2.2.2.1 6.7%, T1.2.2.2.2 3.3%, T1.2.2.3 16.0%, T1.2.2.4 4.0%, T2 24.0%
20 001–25 000	T1 45.0%, T1.1 28.0%, T1.1.1 10.0%, T1.1.2 13.0%, T1.1.3 5.0%, T1.2 17.0%, T2 55.0%
25 001–30 000	T1 6.0%, T2 94.0%
30 001–35 000	T1 88.0%, T1.1 12.0%, T1.2 40.5%, T1.3 3.4%, T1.4 32.1%, T1.4.1 8.0%, T1.4.2 20.1%, T1.4.2.1 4.0%, T1.4.2.2 5.2%, T1.4.2.3 6.9%, T1.4.2.4 4.0%, T1.4.3 4.0%, T2 12.0%, T2.1 4.0%, T2.2 8.0%
35 001–40 000	T1 24.0%, T1.1 7.4%, T1.2 8.5%, T1.3 4.1%, T1.4 4.0%, T2 40.0%, T2.1 10.3%, T2.2 24.0%, T2.2.1 16.0%, T2.2.1.1 4.0%, T2.2.1.2 12.0%, T2.2.1.2.1 4.2%, T2.2.1.2.2 7.8%, T2.2.2 3.0%, T2.2.3 5.0%, T2.3 5.7%, T3 36.0%
40 001–45 000	T1 16.5%, T1.1 8.5%, T1.2 4.2%, T1.3 3.8%, T2 19.5%, T3 28.0%, T4 36.0%, T4.1 32.0%, T4.1.1 4.0%, T4.1.2 28.0%, T4.2 4.0%
45 001–50 000	T1 44.0%, T2 56.0%, T2.1 40.0%, T2.1.1 8.0%, T2.1.2 20.0%, T2.1.2.1 4.0%, T2.1.2.2 8.9%, T2.1.2.3 7.1%, T2.1.3 3.6%, T2.1.4 8.4%, T2.2 3.7%, T2.3 12.3%

11 Conclusions and future work

We apply a bio-inspired model of decentralized task allocation (Theraulaz *et al.*, 1998) (which we refer to as StimHab) to a dynamic hierarchically defined set of ongoing tasks. Patrolling is used as a sample domain, where agents self-allocate proportionately to the dynamic patrolling needs of a set of nested areas using per-area stimuli for coordination. After initial adaptation, task switching in the system reduces to about 2% of the actions at every step, indicating highly specialized behavior, which has been proven beneficial in many multi-agent domains. A team of 1000 agents self-allocate effectively without communicating, demonstrating StimHab's scalability. Drawing further inspiration from biology, we then employ a GA to show that the adaptability and stability of decentralized task allocation based on task stimuli and agents' specialization thresholds can be improved further by altering coefficients in StimHab's action probability formula.

Although StimHab produces successful average task allocation over cycles of multiple steps, per-step behavior exhibits some undesirable oscillations. We employ a simple GA to test whether alternate stimulus-threshold relationships can improve the emerging task allocation. Our results show that even after relatively short evolutions on small test sets of only 10 changes in system needs, all tested fitness functions are able to find stimulus-threshold relationships with better per-step behavior than standard StimHab, that is, those with more accurate and more stable agent self-assignment and reassignment to

tasks. A variety of coefficients lead to similar overall behavior that approximates a step function, in that all probabilistic variation is restricted to a small portion of the possible stimulus-threshold pairings.

The benefits of improved emergent task allocation and specialization extend to many decentralized dynamic domains, hierarchical or not, that have multiple continuously needed tasks always available to the agents (e.g., perishable-resource gathering, maintenance, on-demand production, etc.), many agents, and no interagent communication. As StimHab agents are not responding to changes in demand, but rather to changes in performance, the approach is suitable for a variety of dynamic environments, such as those with agent failure/replacement, or with environmental variations that can change how work translates into performance. This means that triggering a reset of specializations when demands change may not be sufficient if respecialization is needed for other reasons. The evolved solutions show that altering the relationships between stimuli and threshold in the probability formula can be sufficient to respecialize effectively without threshold resetting. StimHab task allocating to hierarchical task sets can reduce micromanagement in dynamic multi-agent systems: agents can self-allocate to large higher level subdomains, where the newly formed subgroups can subdivide further using StimHab, or switch to a more controlled but less scalable approach, such as scheduling, auctions, or graph algorithms (Almeida *et al.*, 2004; Portugal & Rocha 2011), depending on subdomain specifics. Additionally, although we model no explicit cost to task switching, StimHab still results in highly specialized agent actions, beneficial in many domains.

Future work includes expanding testing into systems with asynchronous agent actions resulting in asymmetrical information, subsets of agents with preset capabilities (static $\theta_{a,t}$), explicit task switching costs, and mapping the hierarchical domain to a graph topology to evaluate the behavior on actual physical layouts. The simplistic GA and fitness functions employed here can be expanded further, such as by combining timing and value weighting on the penalties into a single fitness function. Evolving on larger sets of testing data can further guard against overfitting, potentially leading to discovering even more adaptable stimulus-threshold relationships. Different stimulus-threshold relationships may benefit deeper hierarchies as opposed to flatter setups, as specialization will propagate slower toward deeper subtasks. For domains where task-switching costs are known, a more targeted evolution can be implemented (e.g., if switching tasks cost two steps of inactivity, performance will be directly affected and thus performance deviations alone can be used as the solution fitness). In this work, we focus on potential improvements to StimHab itself, as it has been widely used for decentralized task allocation, but comparison to other systems is needed and work is currently underway to directly test StimHab's task allocation against that achieved by other probabilistic Learning Automata approaches.

Appendix A-I Solutions WITHOUT resetting, tested on TRAINING data

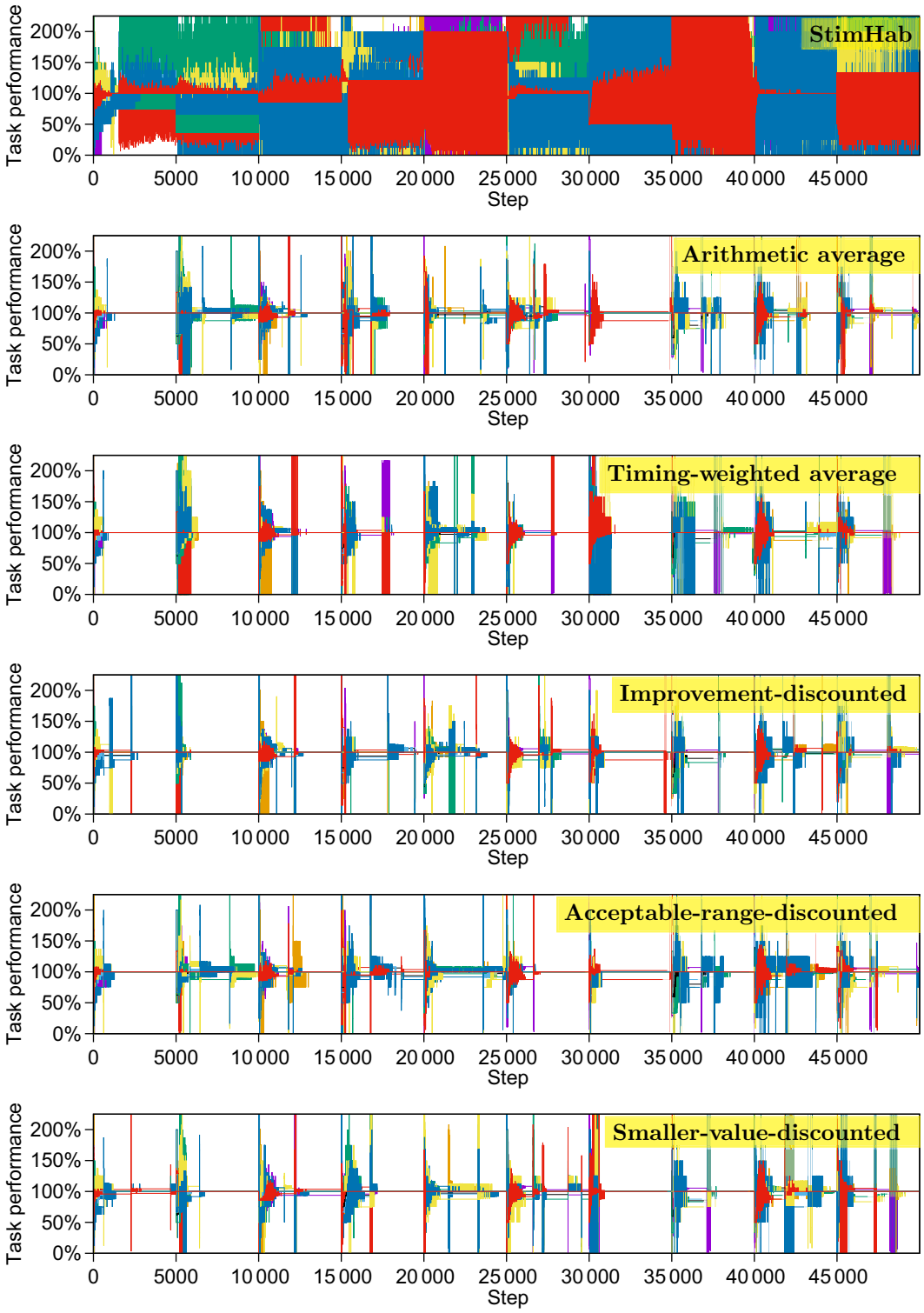

Figure A1 NO RESET + TRAINING DATA: task performances (color lines; 100% is ideal) throughout all 10 task switches (every 5000 steps) for the best solutions evolved **without** resetting. Solutions are **tested on a static set of tasks with dynamic demands identical to those used during evolution** (task set and demands are provided in Table 3)

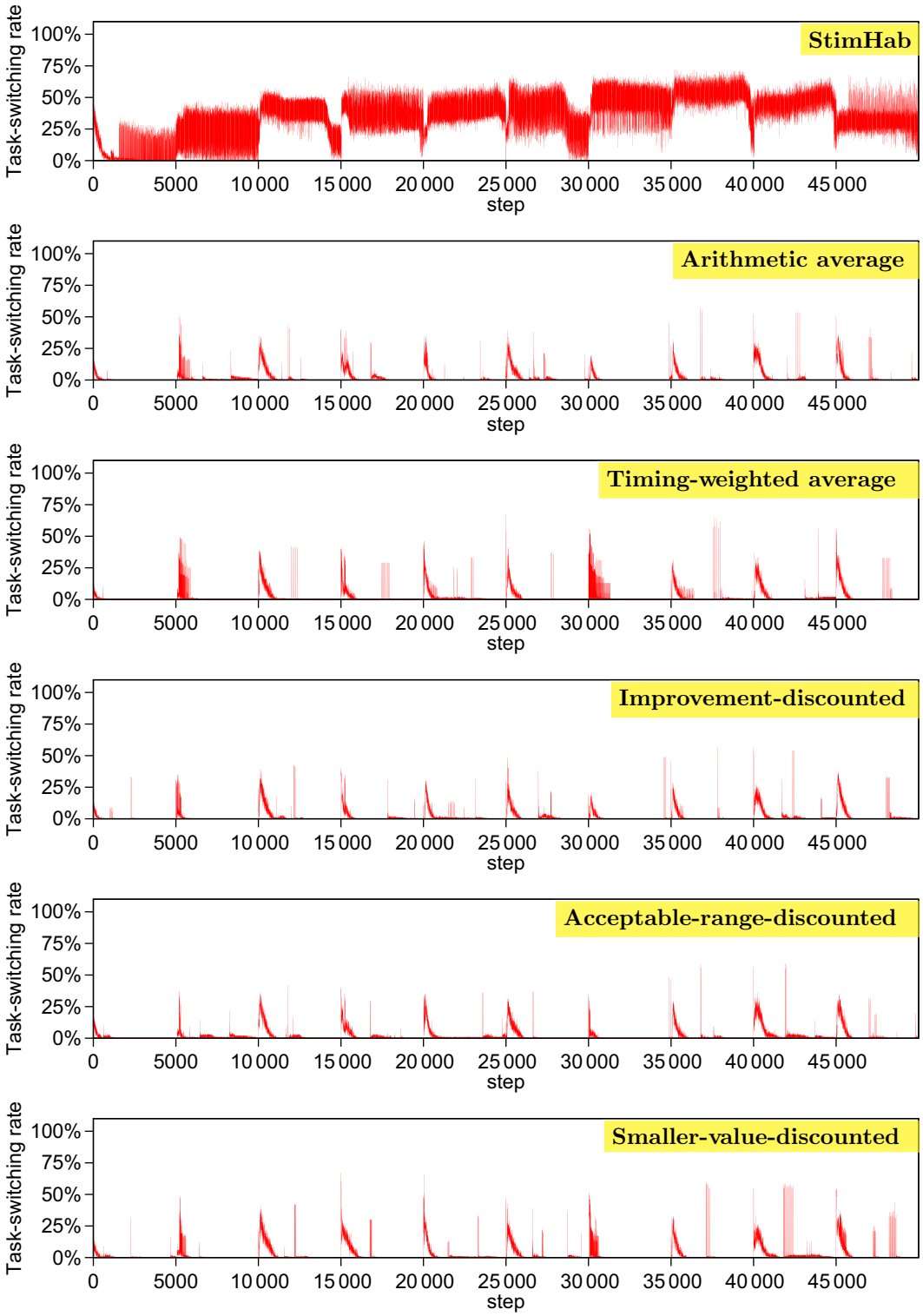

Figure A2 NO RESET + TRAINING DATA: task-switching rates (red line; 0% is ideal) throughout all 10 task switches (every 5000 steps) for the best solutions evolved **without** resetting. Solutions are **tested on a static set of tasks with dynamic demands identical to those used during evolution.** (task set and demands are provided in Table 3)

Appendix A-II Solutions WITHOUT resetting, tested on TESTING data

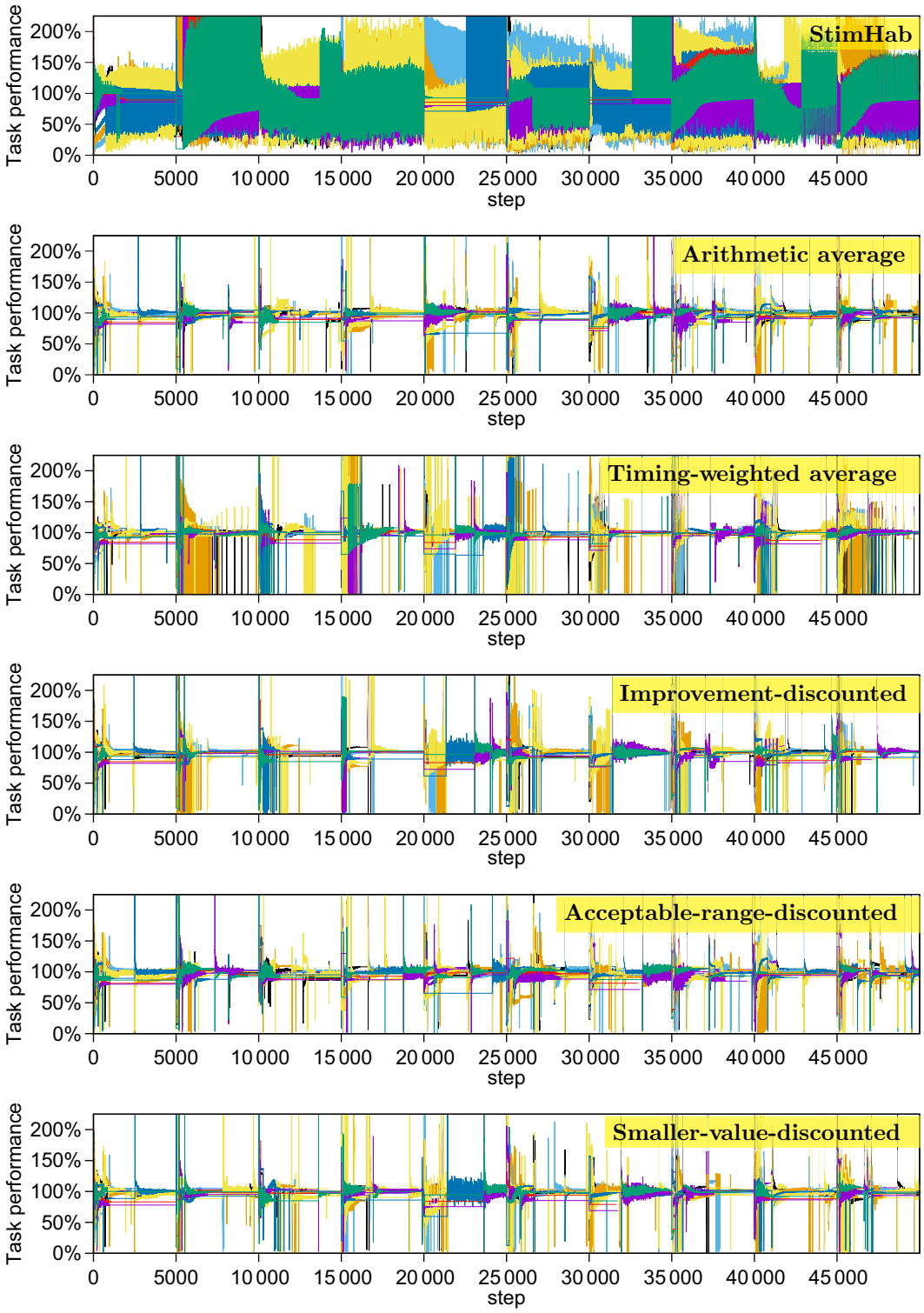

Figure A3 NO RESET + TESTING DATA: task performances (color lines; 100% is ideal) throughout all 10 task switches (every 5000 steps) for the best solutions evolved **without** resetting. Solutions are **tested on a static set of tasks with dynamic demands different from those used during evolution.** (task set and demands are provided in Table 4)

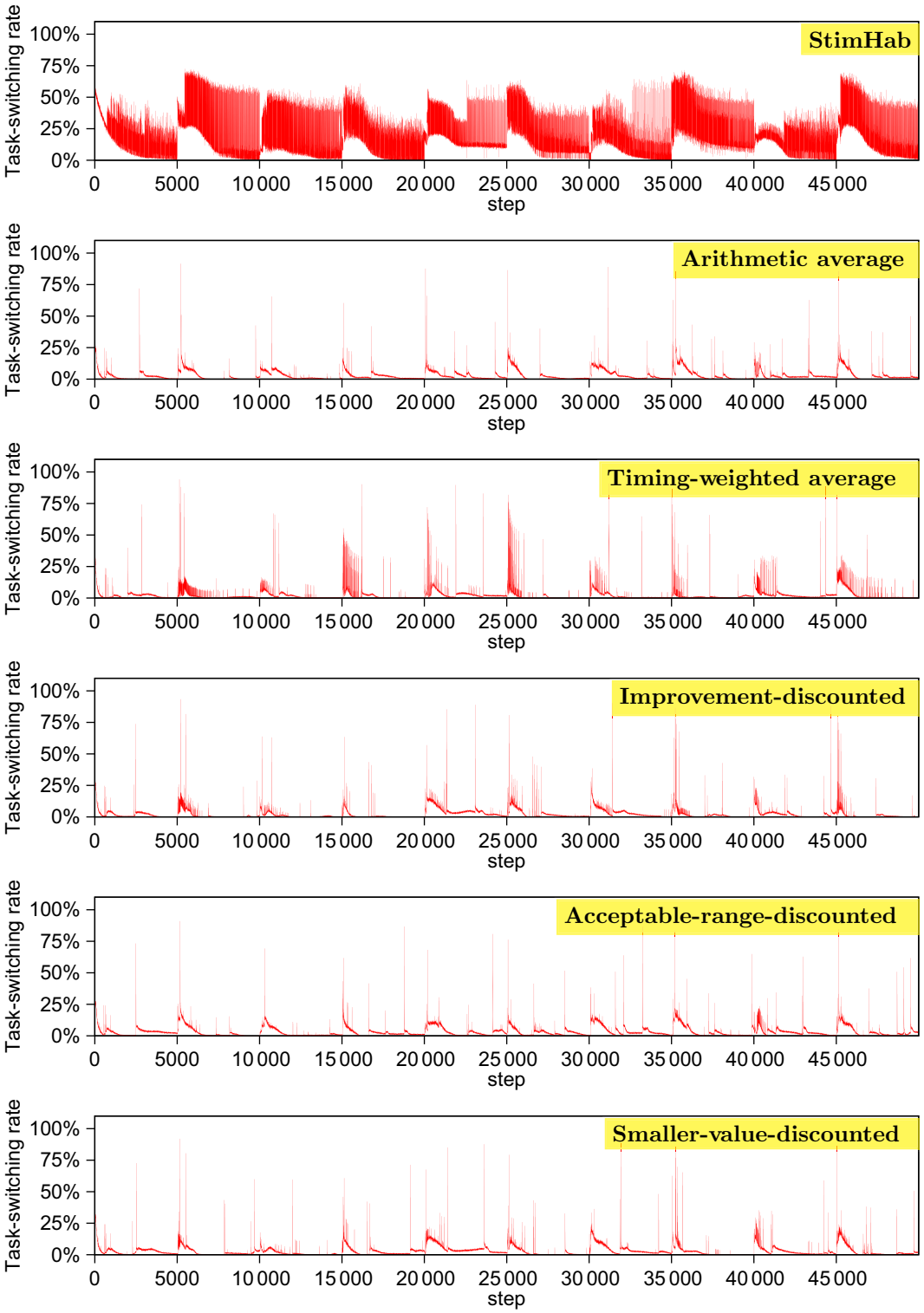

Figure A4 NO RESET + TESTING DATA: task-switching rates (red line; 0% is ideal) throughout all 10 task switches (every 5000 steps) for the best solutions evolved **without** resetting. Solutions are **tested on a static set of tasks with dynamic demands different from those used during evolution.** (task set and demands are provided in Table 4)

Appendix B-I Solutions WITH resetting, tested on TRAINING data

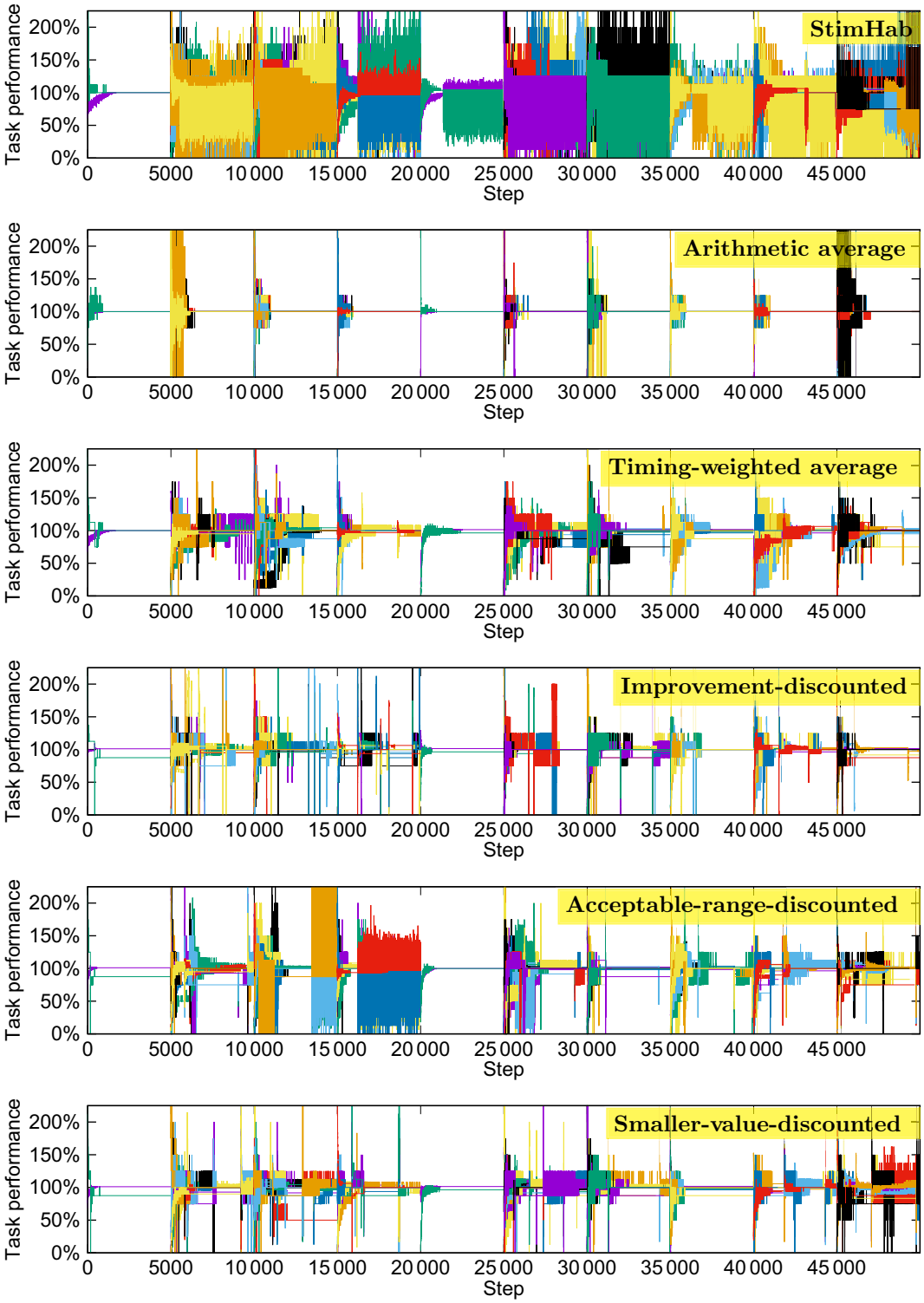

Figure B1 RESET + TRAINING DATA: task performances (color lines; 100% is ideal) throughout all 10 task switches (every 5000 steps) for solutions evolved **with** resetting. Solutions are **tested on dynamically changing task sets, identical to those used during evolution.** (task set and demands are provided in Table 6)

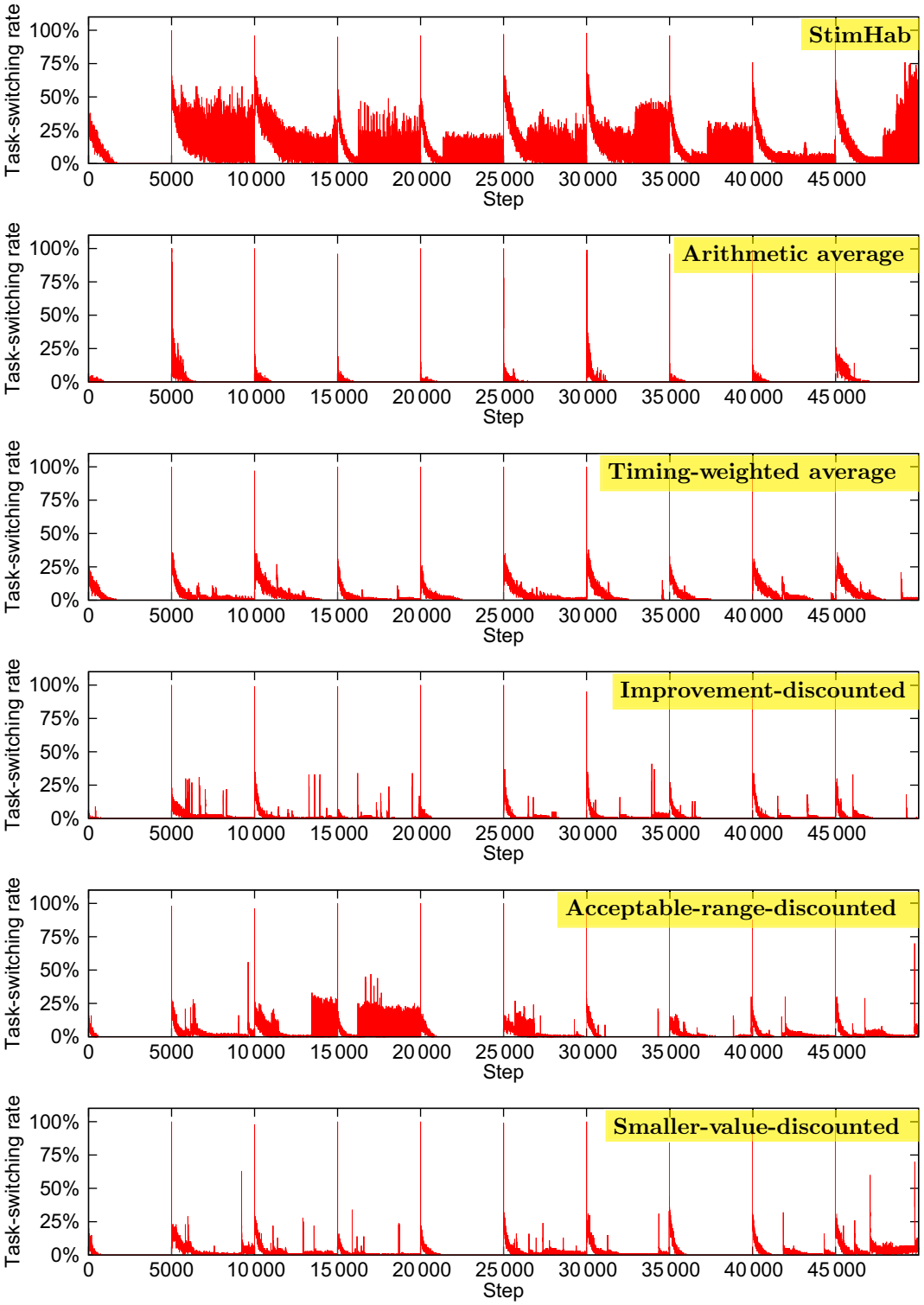

Figure B2 RESET + TRAINING DATA: task-switching rates (red line; 0% is ideal) throughout all 10 task switches (every 5000 steps) for solutions evolved **with** resetting. Solutions are **tested on dynamically changing task sets, identical to those used during evolution.** (task set and demands are provided in Table 6)

Appendix B-II Solutions WITH resetting, tested on TESTING data

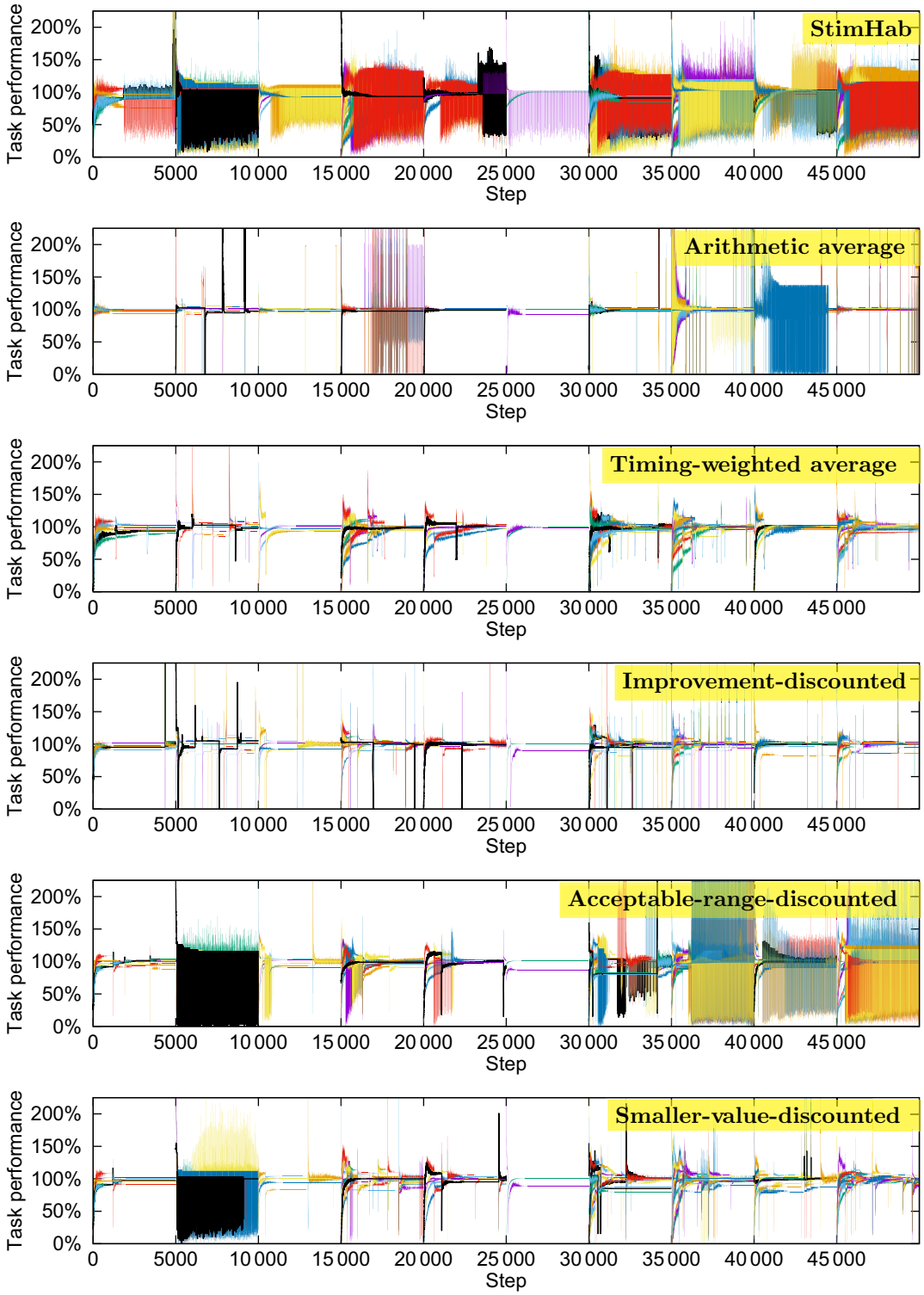

Figure B3 RESET + TESTING DATA: task performances (color lines; 100% is ideal) throughout all 10 task switches (every 5000 steps) for solutions evolved **with** resetting. Solutions are **tested on dynamically changing task sets, different from those used during evolution.** (task set and demands are provided in Table 7)

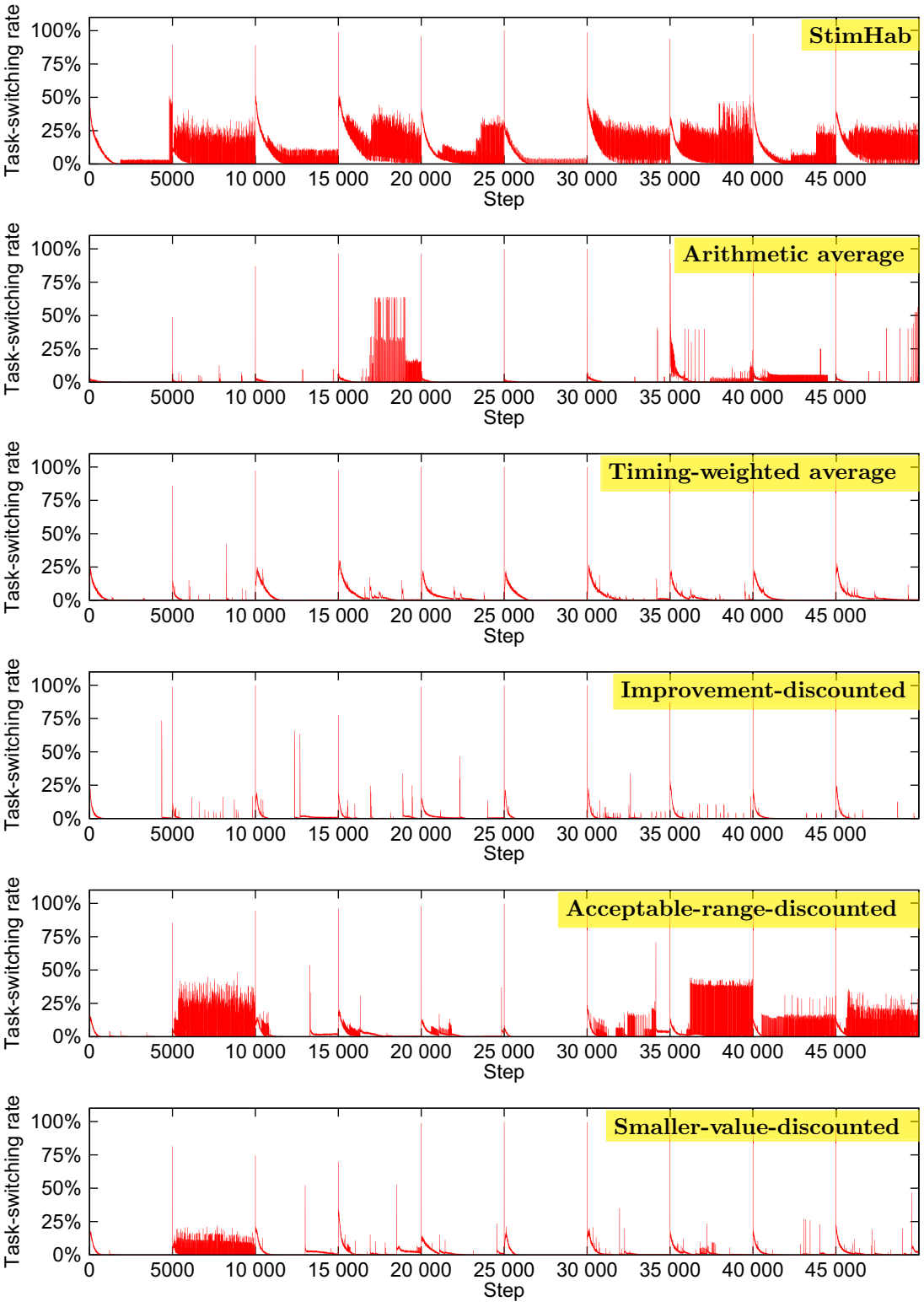

Figure B4 RESET + TESTING DATA: task-switching rates (red line; 0% is ideal) throughout all 10 task switches (every 5000 steps) for solutions evolved **with** resetting. Solutions are **tested on dynamically changing task sets, different from those used during evolution.** (task set and demands are provided in Table 7)

Appendix C

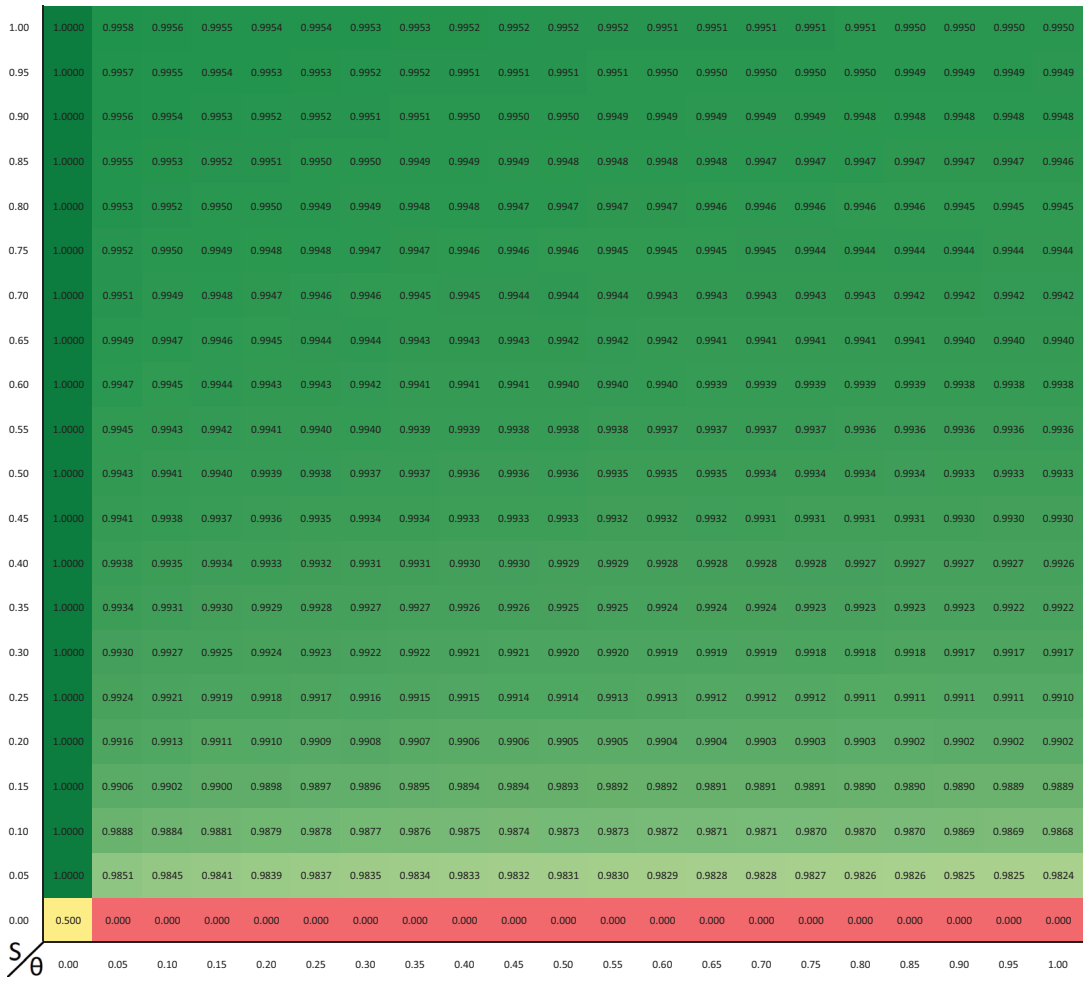

Figure C1 Probability $P_{a,t}$ mapping across all pairs of s_t and $\theta_{a,t}$ (discretized into 0.05 intervals) for the solution evolved **with** resetting, using the arithmetic averaging fitness function

References

Agmon, N., Urieli, D. & Stone, P. 2011. Multiagent patrol generalized to complex environmental conditions. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence (AAAI 2011)*.

Almeida, A., Ramalho, G., Santana, H., Tedesco, P., Menezes, T., Corruble, V. & Chevaleyre, Y. 2004. Recent advances on multi-agent patrolling. In *Advances in Artificial Intelligence – SBIA 2004*, Bazzan, A. L. C. & Labidi, S. (eds). Springer Berlin Heidelberg, 474–483. ISBN: 978-3-540-28645-5.

Baker, J. E. 1985. Adaptive selection methods for genetic algorithms. In *Proceedings of an International Conference on Genetic Algorithms and Their applications*, Hillsdale, New Jersey, 101–111.

Berman, S., Halasz, A., Kumar, V. & Pratt, S. 2007. Bio-inspired group behaviors for the deployment of a swarm of robots to multiple destinations. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2318–2323.

Campbell, A. & Wu, A. S. 2011. Multi-agent role allocation: Issues, approaches, and multiple perspectives. *Autonomous Agents and Multi-Agent Systems* **22**(2), 317–355.

Campos, M., Bonabeau, E., Theraulaz, G. & Deneubourg, J.-L. 2000. Dynamic scheduling and division of labor in social insects. *Adaptive Behavior* **8**(2), 83–95.

Chu, H. N., Glad, A., Simonin, O., Sempe, F., Drogoul, A. & Charpillet, F. 2007. Swarm approaches for the patrolling problem, information propagation vs. pheromone evaporation. In *19th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2007*, 1, 442–449. 2007.

Cicirello, V. A. & Smith, S. F. 2004. Wasp-like agents for distributed factory coordination. *Autonomous Agents and Multi-Agent Systems* **8**(3), 237–266. ISSN: 1573-7454.

De Jong, K. A. 2006. *Evolutionary Computation: A Unified Approach*, MIT Press, Cambridge, MA, USA.

dos Santos, F. & Bazzan, A. L. 2009. An ant based algorithm for task allocation in largescale and dynamic multiagent scenarios. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, 73–80. ACM.

dos Santos, F. & Bazzan, A. L. 2011. Towards efficient multiagent task allocation in the robocup rescue: a biologically-inspired approach. *Autonomous Agents and Multi-Agent Systems* **22**(3), 465–486.

dos Santos, D. S. & Bazzan, A. L. 2012. Distributed clustering for group formation and task allocation in multiagent systems: a swarm intelligence approach. *Applied Soft Computing* **12**(8), 2123–2131. ISSN: 1568-4946.

Ducatelle, F., Förster, A., Di Caro, G. A. & Gambardella, L. M. 2009. New task allocation methods for robotic swarms. In *9th IEEE/RAS Conference on Autonomous Robot Systems and Competitions*.

Farinelli, A., Iocchi, L., Nardi, D. & Ziparo, V. A. 2006. Assignment of dynamically perceived tasks by token passing in multirobot systems. *Proceedings of the IEEE* **94**(7), 1271–1288.

Ghizzioli, R., Nouyan, S., Birattari, M. & Dorigo, M. 2005. An Ant-Based Algorithm for the Heterogeneous Dynamic Task Allocation Problem, Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle (IRIDIA), Technical Report TR/IRIDIA/2005-005.

Halász, A., Hsieh, M. A., Berman, S. & Kumar, V. 2007. Dynamic redistribution of a swarm of robots among multiple sites. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2007*, 2320–2325. IEEE.

Hsieh, M. A., Halász, Á., Berman, S. & Kumar, V. 2008. Biologically inspired redistribution of a swarm of robots among multiple sites. *Swarm Intelligence* **2**(2–4), 121–141.

Hsieh, M. A., Halász, Á., Cubuk, E. D., Schoenholz, S. & Martinoli, A. 2009. Specialization as an optimal strategy under varying external conditions. In *IEEE International Conference on Robotics and Automation, ICRA 2009*, 1941–1946.

Kanakia, A., Touri, B. & Correll, N. 2016. Modeling multi-robot task allocation with limited information as global game. *Swarm Intelligence* **10**(2), 147–160.

Kazakova, V. A., Wu, A. S. & Rahman, T. S. 2013. Cluster energy optimizing genetic algorithm. In Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, 1317–1324. ACM.

Kazakova, V. A. & Wu, A. S. 2018. Specialization vs. re-specialization: Effects of hebbian learning in a dynamic environment. In *Florida Artificial Intelligence Research Society Conference FLAIRS-31*.

Kira, Z. & Arkin, R. C. 2004. Forgetting bad behavior: memory for case-based navigation. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS)*, 4, 3145–3152.

Li, L., Martinoli, A. & Abu-Mostafa, Y. S. 2002. Emergent specialization in swarm systems. In *International Conference on Intelligent Data Engineering and Automated Learning*, 261–266. Springer.

Liu, W., Winfield, A. F., Sa, J., Chen, J. & Dou, L. 2007. Towards energy optimization: emergent task allocation in a swarm of foraging robots. *Adaptive Behavior* **15**(3), 289–305.

Ma, H., Li, J., Kumar, T. & Koenig, S. 2017. Lifelong multi-agent path finding for online pickup and delivery tasks. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, International Foundation for Autonomous Agents and Multiagent Systems, 837–845.

Mavrovouniotis, M., Li, C. & Yang, S. 2017. A survey of swarm intelligence for dynamic optimization: algorithms and applications. *Swarm and Evolutionary Computation* **33**, 1–17.

McIntire, M., Nunes, E. & Gini, M. 2016. Iterated multi-robot auctions for precedenceconstrained task scheduling. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, International Foundation for Autonomous Agents and Multiagent Systems, 1078–1086.

Merkle, D. & Middendorf, M. 2004. Dynamic polyethism and competition for tasks in threshold reinforcement models of social insects. *Adaptive Behavior* **12**(3–4), 251–262.

Murciano, A., del R. MillÁn, J. & Zamora, J. Specialization in multi-agent systems through learning. *Biological Cybernetics* **76**(5), 375–382. ISSN: 1432-0770.

Nitschke, G., Schut, M. & Eiben, A. 2008. Emergent specialization in biologically inspired collective behavior systems. In *Intelligent Complex Adaptive Systems*, 215–253. IGI Global.

Nouyan, S. 2002. Agent-based approach to dynamic task allocation. In *International Workshop on Ant Algorithms*, 28–39. Springer.

Nouyan, S., Ghizzioli, R., Birattari, M. & Dorigo, M. 2005. An insect-based algorithm for the dynamic task allocation problem. KI **19**(4), 25–31.

Nunes, E., McIntire, M. & Gini, M. 2016. Decentralized allocation of tasks with temporal and precedence constraints to a team of robots. In *IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, 197–202. IEEE.

Ono, N. & Fukumoto, K. 1996. Multi-agent reinforcement learning: a modular approach. In *Second International Conference on Multiagent Systems*, 252–258.

Portugal, D. & Rocha, R. 2011. A survey on multi-robot patrolling algorithms. In *Doctoral Conference on Computing, Electrical and Industrial Systems*, 139–146. Springer.

Price, R. & Tiño, P. 2004. Evaluation of adaptive nature inspired task allocation against alternate decentralised multiagent strategies. In *International Conference on Parallel Problem Solving from Nature*, 982–990. Springer.

Ragusa, V. R., Mathias, H. D., Kazakova, V. A. & Wu, A. S. 2017. Enhanced genetic path planning for autonomous flight. In *Proceedings of the Genetic and Evolutionary Computation Conference*, ACM, 2017, pp. 1208–1215.

Román, J. A., Rodríguez, S. & Corchado, J. M. 2014. Improving intelligent systems: specialization. In *International Conference on Practical Applications of Agents and Multi-Agent Systems*, 378–385. Springer.

Schwarzrock, J., Zacarias, I., Bazzan, A. L., de Araujo Fernandes, R. Q., Moreira, L. H. & de Freitas, E. P. 2018. Solving task allocation problem in multi unmanned aerial vehicles systems using swarm intelligence. *Engineering Applications of Artificial Intelligence* **72**, 10–20.

Stanley, K. O. & Miikkulainen, R. 2002. Evolving neural networks through augmenting topologies. *Evolutionary Computation* **10**(2), 99–127.

Theraulaz, G., Bonabeau, E. & Deneubourg, J.-L. 1998. Response threshold reinforcement and division of labour in insect societies. *Proceedings of the Royal Society of London B* **265**, 327–332.

van Lon, R. R. & Holvoet, T. 2017. When do agents outperform centralized algorithms? *Autonomous Agents and Multi-Agent Systems* **31**(6), 1578–1609.

Villacorta, P. J., Pelta, D. A. & Lamata, M. T. 2013. Forgetting as a way to avoid deception in a repeated imitation game. *Autonomous Agents and Multi-Agent Systems* **27**(3), 329–354.

Wawerla, J. & Vaughan, R. T. 2010. A fast and frugal method for team-task allocation in a multi-robot transportation system. In *ICRA*, 1432–1437.

Wu, A. S. & Kazakova, V. A. 2017. Effects of task consideration order on decentralized task allocation using time-variant response thresholds. In *Florida Artificial Intelligence Research Society Conference FLAIRS-30*, 466–471.

Zhang, Z., Long, K., Wang, J. & Dressler, F. 2014. On swarm intelligence inspired self-organized networking: its bionic mechanisms, designing principles and optimization approaches. *IEEE Communications Surveys & Tutorials* **16**(1), 513–537.

Zheng, X. & Koenig, S. 2011. Generalized reaction functions for solving complex-task allocation problems. IJCAI Proceedings-International Joint Conference on Artificial Intelligence, **22**, 478.