Abstract—Real-world domains can often benefit from adaptable decentralized task allocation through emergent specialization of large groups of agents. Agents working within a shared environment can use environmental cues as a means of indirect coordination to maximize performance. We compare how groups of homogeneous agents fulfill the demands of a set of “ongoing” dynamic tasks over time, when employing Response Threshold Reinforcement, Ant Colony Optimization, or Linear Reward-Penalty Learning Automata. Results show that efficient, stable, and adaptable task allocation for large teams choosing from large sets of always-available tasks is possible, but excess and deficit workforce can present some performance challenges.

Index Terms—multi-agent systems, adaptive systems, emergent phenomena, decentralized control, ongoing tasks

I. INTRODUCTION

Task allocation in real-world domains often requires decentralized, scalable, and adaptable approaches. Decentralized systems demonstrate higher scalability and increased robustness to failure than centralized alternatives [1]. Adaptable task allocation is beneficial in dynamic domains, as task reallocation may be needed to handle agent failures and system changes [2]. We focus on communication-free approaches, as they can increase scalability and robustness of task allocation in large groups of simple and cheap agents [3], as well as in domains where communication is not reliable or feasible [4]. We compare three communication-free approaches for decentralized MultiAgent Systems (MAS) task allocation that are applicable to domains without task availability limitation. Their emergent task allocation stems from agents sensing their environment: i.e. no exchange of information/directives, no awareness of other agents or their choices, and no modeling or memory of the environment over time.

We focus on decentralized task allocation in environments with multiple always available tasks of changing demands. There are many tasks that can be acted on without limitation but with limited benefits per step. For example, while all agents could patrol, we may need exactly 10 agents patrolling every step, until security needs change or until other tasks become more urgent. Patrolling less decreases security; patrolling more wastes resources and causes interference. Patrolling cannot be accumulated: we cannot have 100 agents patrol for an hour, followed by 0 agents for 9 hours. Patrolling is also never completed, requiring some number of agents patrolling at all times. Other always available tasks include resource gathering, cleanup, maintenance, and agent deployment for multi-area surveillance, environmental monitoring, or aerial coverage for units on the ground. We refer to these as “ongoing” tasks.

Existing task allocation techniques commonly operate on discrete task units or roles explicitly assigned to individual agents. Market-based approaches use auctions or dominance contests for assigning task units [5]–[8]. Token-passing approaches allow agents to take turns claiming jobs from a list of options [9]–[11]. Inter-agent recruitment methods form teams and assign roles responsible for each task [11], [12]. All of these approaches require limits on task availability: once an agent gets a task, that task is off the market, reducing task choices for other agents. Applying these approaches to “ongoing” tasks requires an artificial discretization of continuous system needs into individually assignable tasks.

Some decentralized approaches are directly applicable to “ongoing” tasks. One method relies on transition probability matrices: agent probabilities to transition from one task to another [13]. Performance of these systems is sensitive to initial conditions, chosen transition probabilities, and matrix layout. An alternative is cooperation through emergent specialization, often fundamental in complex adaptive systems [14] and capable of outperforming market-based solutions [15]. Specialization-based cooperation scales better than generalist approaches by: (1) allowing for simpler software and cheaper hardware, by limiting agents’ focus to a subset of the requirements [16]; (2) reducing interference from agents attempting to take on the same tasks [17]; (3) helping synchronize timesensitive actions, as switching from one task to another can cause delays or even hinder functionality (e.g. herding [18]); 4) diversifying a homogeneous workforce, leading to increased individual adaptability and improved joint performance in unpredictable environments [3], [19], [20]; and (5) improving scalability through breaking down large problems into tractable subproblems (e.g. agents tasked with patrolling a large area vs. patrolling a set of smaller subareas) [21].

We compare how non-communicating agents self-assign when not limited by task availability nor informed by each others’ choices. Response Threshold Reinforcement (RTR), Ant Colony Optimization (ACO), and Learning Automata (LA) are all amenable to “ongoing” task allocation through their emergent adaptation. RTR employs threshold reinforcement, resulting in agents repeatedly choosing the same tasks [22]. ACO uses pheromone deposit and decay rules, allowing agents to jointly explore the solution space [23]. LA uses proba-
ability updates, allowing agents to learn the correct actions by repeated interactions with their environment [24]. These approaches to not need discrete tasks to be defined and repeatedly auctioned off for dynamic ongoing needs. Agents do not observe/model each other, nor share information; system changes affect the sensed stimuli, affecting agent choices.

RTR, ACO, and LA can allow agents to complete sets of discrete tasks [25], but it is unclear how these approaches differ in their ability to specialize and flexibility to respecialize, i.e. to adapt from pre-existing conditioning, as “ongoing” needs change. Adaptable specialization must work around failures and changes, but readaptability is not guaranteed given an ability to adapt to initial conditions. Adaptive solutions often depend on agent diversity, which can diminish after initial adaptation, complicating re-adaptation [26]–[28]. For instance, RTR struggles to change pre-adapted specializations to suit a changing environment [27], [28].

We compare dynamic adaptation under RTR, ACO, and LA, as decentralized re-adaptable task allocation for “ongoing” tasks is not yet well studied. Results show that efficient, stable, and adaptable task allocation for non-communicating large teams choosing from sets of “ongoing” tasks is possible, even given environment-cue estimation errors and task switching delays, but agent deficit or excess present some challenges.

II. COMPARISON APPROACHES

A. Response Threshold Reinforcement (RTR)

We use the same RTR setup as [25] and [28], as these works handle decentralized task allocation of discrete and “ongoing” tasks, respectively. Below we review how stimuli and specialization thresholds are defined, updated, and used to calculate action probabilities, as well as how the resulting actions reinforce agents’ thresholds, leading to specialization.

Agents perceive overall system needs through globally observable stimuli (e.g. dimensions of a fire to be put out [4] or task performance [28]). A higher task stimulus leads to a higher probability to act on that task. Task k stimulus \( s_t \in [0.0,1.0] \) is updated as:

\[
s_t = s_t + (\Delta s_t \text{ given no work}) \times (1 - \text{step performance})
\]

where \( \Delta s_t \) represents an increasing urgency of neglected tasks, defined here as 0.01/step. Having the correct number of agents on a task to match demand leads to no change in \( s_t \). Excess work decreases \( s_t \), decreasing activity on the task; insufficient work increases stimulus, increasing activity.

Each agent maintains its own response threshold \( \theta \) per task. When agent \( a \) acts on task \( k \), its threshold \( \theta_{a,t} \) reduces as \( \theta_{a,t} = \theta_{a,t} - \xi \), while thresholds for the other tasks \( t' \neq t \) increase as \( \theta_{a,t'} \neq \theta_{a,t'} + \phi \), where \( \xi \in [0.0,1.0] \), \( \phi = 1/100 \), and \( \phi = \xi / \text{task count - 1} \).

Every time step, each agent \( a \) calculates its probability \( P_{a,t} \) to act on every task, using global \( s_t \) and individual \( \theta_{a,t} \):

\[
P_{a,t} = s_t^2 / (s_t^2 + \theta_{a,t}^2)
\]

where \( s_t \in [0.0,1.0] \), \( \theta_{a,t} \in [0.0,1.0] \), \( P_{a,t} = 0.5 \) where undefined \( (s_t = \theta_{a,t} = 0.0) \)

Lower thresholds increase action probability, leading to repeated actions and to specialization over time.

B. Ant Colony Optimization (ACO)

The ACO variant used here is described in [23] and used with a binary reward/punishment for decentralized task allocation in [29]. Below we show how action probabilities are defined and updated based on learning rates and environmental cues.

To facilitate comparison and illustrate that this ACO implementation uses pheromones similarly to how RTR and LA use probabilities, we replace the standard \( \tau \) denoting pheromone levels with \( P \), updated for the chosen task \( t \) only, once per step:

\[
P_{a,t} = P_{a,t} \ast (1 - \rho) + \rho \ast \beta_t
\]

where pheromone decay \( \rho = 0.2 \) (matching [29]) and environmental response \( \beta_t \) is calculated for each task \( t \) as:

\[
\beta_t = \begin{cases} 
0, & \text{if } [\text{agents on task}_t] > [\text{agents needed on task}_t] \\
1, & \text{otherwise (i.e. up to exact agents needed on task}_t) 
\end{cases}
\]

For example, \( \beta = 0 \) can indicate production exceeding consumption. The environmental response \( \beta_t \in [0,1] \) is sensed by all agents acting on task, that step. \( P_{a,t} \in [0.0,1.0] \) are initialized to indifference: \( P_{a,t} = 1/|\text{tasks}| \) where \( |\text{tasks}| \) is the task count. An agent’s pheromone for a task decays every time the agent acts on this task, but is only increased if the environment returns a reward \( \beta_t = 1 \). Note how this differs from other ACO versions, where agents affect joint pheromone levels on shared solution components (e.g. Traveling Salesman [23]). Here, the decay is applied only to the agent’s personal pheromone for the chosen task. Rather than a colony learning together, we have individual learners.

C. Learning Automata: Linear Reward-Penalty (L\(_{R-P}\))

The variant of LA used in the original comparison [30] is Linear Reward-Inaction (L\(_{R-I}\)). L\(_{R-I}\) is known to have absorbing states [31], not allowing agents to respecialize when needs change. As we dealing with dynamic environments, we implement the Linear Reward-Penalty L\(_{R-P}\) [24], instead. While in L\(_{R-I}\) action probabilities update only when reward is received, under L\(_{R-P}\) probabilities update for both reward and penalty, allowing for respecialization.

To facilitate comparison and show that L\(_{R-P}\) updates probabilities similarly to ACO above, we replace the learning rate \( \lambda \) from the original definition, with \( \rho \). Agents’ task probabilities \( P_{a,t} \) are updated after every action (one per step) as follows:

\[
P_{a,t} = P_{a,t} + \rho \ast \beta_t \ast (1 - P_{a,t}) - \rho \ast (1 - \beta_t) \ast P_{a,t}
\]

Proportional \( P_{a,t} \in [0.0,1.0] \) are again initialized to indifference: \( P_{a,t} = 1/|\text{tasks}| \); \( \beta_t \) is defined and calculated identically to the ACO implementation above. Correct action moves the action probability closer to 1.0, while incorrect action moves the probabilities of the other tasks closer to indifference among them, i.e. \( P_{a,t} = 1/(|\text{tasks}|-1) \), as opposed to the initial \( P_{a,t} = 1/|\text{tasks}| \), thus excluding the task that lead to a penalty (see the \( \beta \) and \( 1-\beta \) “switches” in the formulas: green segments activate when \( \beta = 1 \) and red segments when \( \beta = 0 \)).

III. EXPERIMENTAL SETUP & METRICS

We compare RTR, ACO, and L\(_{R-P}\) approaches without and with resetting specializations in four testing environments.
Preliminary Definitions. Let us first define some terms:

“Ongoing” Task: any system need that could be acted on up by all agents or no agents on any given step (e.g. patrolling)

Task Demand: agents needed on a task per step (as “ongoing” tasks are never finished, requiring agent actions each step).

Demand Period: set of steps when demands are unchanged (we alter demands every 5000 steps, 10 times per test)

Task Performance: how well a task’s demand is satisfied each step (given demand, if task, had \{5, 11, 10\} agents over 3 steps, per step performance was \{50%, 110%, 100\%\}).

Without vs. With Resetting Specializations. We compare how the approaches adapt and readapt, as initial conditions can impact emergent behavior. When specializations (θ, or \( P_{a,t} \)) are reset before readaptation, readaptation is transformed into adaptation. Thus, the approaches are tested:

1) Without ever resetting the learned values: any adapted parameters that had previously emerged are not altered as agents begin to readapt to new conditions. These tests show how agents reallocate when demands change over time, without learning from scratch, since for small changes a complete system-wide relearning may be unnecessarily destructive.

2) With resetting learned values when demands change: any adapted parameters are reset to pre-adaptation values before agents begin adapting to new conditions; specifically, \( \theta_{a,t} \) is reset to random values \( \in [0.0, 1.0] \), \( s_{t} \) to 0.0, and ACO \( L_{R-P} \) to indifference (1.0/|tasks|). These tests show how initial adaptation differs on average, as every adaptation period corresponds to learning from scratch. Resetting also demonstrates behavior when the entire task set changes, making any previously adapted values non-applicable.

Without limit on task supply, agents must choose from among all possible tasks each step. In this work, we show results for deterministic task selection: agent, chooses task, with highest \( P_{a,t} \) across all tasks, idling if \( \forall t, P_{a,t} = 0 \). We also tested probabilistic selection (agent, chooses task, in roulette-wheel selection performed on \( P_{a,t} \), rescaled so agent’s \( P_{a,t} \) sum to 1.0 [30]; idling when \( \forall t, P_{a,t} = 0 \), as well as a probabilistic fall-through approach (each agent considers tasks in descending order of \( P_{a,t} \), acting with probability \( P_{a,t} \) and otherwise considering the next task down the list, until an agent runs out of tasks to consider, idling as default). In our tests, deterministic task selection vastly outperformed the other approaches, so we omit their results due to space constraints.

Metrics. Adaptability is compared on 5000-step average adaptation periods, beginning when task demands change and continuing until demands change again, 5000 steps later. Average per-step values calculated over 10 such periods show the expected performance in dynamic environments. We calculate:

1) Average task performance deviation from 100%, per step: the percentage of work misallocation on a task, above or below demand; averaging deviations prevents positive and negative misallocations from canceling each other (e.g. average performance of two steps of 95% and 105% is 100%, but average deviation is 5%, more meaningfully showcasing the task’s misallocation). Average deviations show the agents’ ability to fulfill demands over time, with deviations near 0% indicating ideal task fulfillment resulting from appropriate self-allocation. 2) Average task switching rate, per step: percentage of agents shows the stability of task assignments, with 0% indicating agents specializing to continuously work on a single task.

Testing Environments. Adaptability is tested without and with resets on 10 “ongoing” tasks, whose demands add up to 1000 total actions needed per step and change every 5000 steps (10 times per test). Setups are tested in 4 environments:

1) no errors, instant task switch, exact needed agents: i.e. 1000 for 1000 total demand per step (idealized baseline)
2) sensing errors, task switch delay, exact needed agents
3) sensing errors, task switch delay, insufficient agents: 800
4) sensing errors, task switch delay, excess agents: 1200

IV. IDEALIZED ENVIRONMENT TEST (BASELINE)

For an initial test, we setup an idealized environment: no delay when agents switch from a task to another, no error on environmental cue estimations (\( s_{t} \) and \( \beta \)), and the exact number of agents needed to fulfill all demands: demands sum to 1000 actions needed per step and the team has 1000 agents. Average adaptation without and with resetting is shown in fig. 1. Horizontal axis shows steps of an average adaptation period; vertical axis shows average performance deviation in blue (95%C.I. in translucent blue) and average task switch rates in red (95%C.I. in translucent red). Deviations and task switching near 0% are best. For added detail, in fig. 2 we show the corresponding un-averaged per-task performances, with horizontal axis showing 5000*10 simulation steps and vertical axis showing performance percentages for each task as different color lines (which line matches to what task is inconsequential); values near 100% are best. Given limited space, un-averaged results for other tests are omitted, but provided at sites.google.com/site/verakazakovaucf/publications.

ACO behaves ideally both without and with resets: performance deviations and task switching quickly drop to 0%. In the
un-averaged graph, performances spike when demands change, but quickly return to 100% ideal performance for all tasks. L_{R-P} averages and C.I. are significantly higher, also reflected in the un-averaged performance fluctuations in fig. 2.

Only RTR benefits from resetting: while RTR without resets struggles to readapt, RTR with resets approximates the ideal behavior observed in ACO. ACO and L_{R-P} appear to not be hindered by pre-existing conditioning and, therefore, do not benefit from resetting learned behavior when conditions change. Additionally, no delays are introduced into readaptation through potentially destructive resets, as ACO and L_{R-P} behavior with and without resetting is identical. In the un-averaged graph without resetting (fig. 2), we see that ACO adapts quickly each time, with all task lines reaching 100% performance and remaining there until the next change in demands, 5000 steps later. RTR approximates this ideal behavior, but only during the first 5000 steps, as agents begin adapting from random θ_{a,t} thresholds. Without resets, the remaining demand periods show highly variable task performances (see the high amplitude of line movements), as agents fail to respecialize from pre-adapted θ_{a,t}. In the un-averaged graph with resets (omitted for brevity), all RTR demand periods closely match ACO, as expected from the averages in fig. 1.

V. PERFORMANCE NOISE AND TASK SWITCHING DELAYS

We now repeat this test under more realistic conditions, by incorporating a 2-step delay when an agent switches tasks, as well as a Gaussian error on environmental cues estimated by each agent. The systems are again provided the exact number of agents needed for all “ongoing” tasks (1000 agents, while task demands add up to 1000 actions needed per step).

For RTR, the Gaussian error is added to each s_t on the agents’ side, ensuring that stimuli remain within [0.0, 1.0]:

(s_t sensed by agent_{a,t}) = max(0.0, min(1.0, s_t*(1 + error)),

with error drawn every step from a Gaussian distribution with mean μ = 0.0 and st.dev. σ = 0.01 (i.e. ±3%). Thus, although actual stimuli are globally defined per task, each agent may sense a slightly different s_t, depending on the random error.

Given that ACO and L_{R-P} use binary environmental response, a small error was imitated by altering the task demand used in the response calculation for each agent. Thus, for calculating β sensed by each agent_{a,t} we alter demand for

\[ \text{demand}_{a,t} = \text{demand}_{a,t} \times (1 + \text{error}) \]

with error drawn every step from a Gaussian distribution \( \mu = 0.0, \sigma = 0.01 \). Thus, although true reward/penalty β is defined per task, each agent may estimate a different β, depending on the random error.

Average adaptation without and with resetting is shown in fig. 3. ACO again performs nearly ideally, although some fluctuations appear due to noise. Interestingly, the same noise seems to help L_{R-P}, which now exhibits similarly near-ideal behavior. This can be explained by repeating fluctuations being broken up with errors and delays, allowing for attenuation instead of getting in a task switching live-lock.

Resetting only affects RTR, drastically improving performance, but not enough to approximate ACO and L_{R-P}. Looking at un-averaged performances, RTR fluctuations grow over time. This may be a result of θ_{a,t} changing by a static amount (ζ and φ), complicating attenuation toward the correct values. ACO and L_{R-P} both employ a dynamic adjustment of their probabilities, although the learning rate (ρ) itself is static. A dynamic adjustment allows for faster initial adaptation and more gradual adjustment as agents become specialized.

![Fig. 2: Performance (values near 100% are best): exact needed agents, no errors, no delays; setups without specialization resetting.](image)

![Fig. 3: Average adaptation: exact agents, with errors and delays.](image)
VI. WORKFORCE SHORTAGE TEST

We now reduce the number of agents available in the system to test behavior under workforce shortages. Task demands still add up to 1000 actions needed per step, but the team now only has 800 agents. Thus, on average, tasks can only be 800/1000=80% satisfied each step, resulting in performance deviations of 20%. We again apply small errors in sensed environmental cues and a 2-step task switching delay.

Average adaptation behaviors without and with resetting are presented in fig. 4. Rather than looking for optimal behavior, we assess how deficit is handled. In the absence of relative task importance, we may want agents to distribute proportionally across tasks. Alternatively, minimizing average performance deviation may be sensible, by losing agents on tasks where each agent represents a lower percentage of demand in favor of tasks where their presence/absence is more impactful.

As expected, RTR is improved by resetting, but still lags behind ACO and L$_{R-P}$. Interestingly, ACO actually performs worse with resetting: average deviation remains near 25%, while without resetting it was near 15%. Without resetting, ACO deviation below the 20% expected from proportionate allocation shows that is possible to achieve lower average deficits at the cost of neglecting some tasks more than others. With resetting, ACO deviation above the 20% indicates its adaptation behavior is not geared toward minimizing average deviation across tasks. L$_{R-P}$ remains near 25% without and with resetting. The flat deviation line and non-existent task switching line under ACO and L$_{R-P}$ is a result of their binary reward $\beta$, causing agents to remain where they are still needed (given the overall worker shortage), regardless of whether they are needed more elsewhere.

Looking at the un-averaged performances, we see that all approaches have some flat performance lines at different heights, indicating that some tasks are consistently neglected (lower percentage lines) in favor of others (higher percentage lines). Thus, insufficient workers are not distributed proportionally across tasks, which may be undesirable.

VII. WORKFORCE EXCESS TEST

We now increase the number of agents in the system to test behavior given a workforce surplus. Task demands still add up to 1000 actions needed per step, but the team now has 1200 agents. We keep the more realistic conditions from the previous test: small errors in agents’ sensing of environmental cues and a 2-step task switching delay.

Average adaptation without and with resetting are presented in fig. 5. As we do not want wasted work, interference, and unnecessary wear-and-tear on system components, under conditions of excess workers, we ideally want to see some of them learning to idle for as long as they are not needed.

Without resetting, no system exhibits ideal behavior given excess agents. RTR performs the worst, unable to respecialize. ACO and L$_{R-P}$ average deviations fall quickly only to grow again, accompanied by large C.I. Looking at un-averaged data, we see that all task performances are above 100%, indicating that tasks are fulfilled, but excess workers cannot learn to idle. The cause is that ACO and L$_{R-P}$ update formulas do not allow probabilities to reach zero and the task with the highest probability, no matter how small, will be acted on. From un-averaged performances without resetting, we see ACO and L$_{R-P}$ performances approaching 100% from above, corresponding to excess work being done.

With resetting, we see a clear role reversal. Given excess agents, not only does RTR perform better with than without resetting, but it outperforms ACO and L$_{R-P}$, with deviation and task switch averages near 0% a third of the way through the adaptation period. While not as fast as the earlier observed cases of ideal behavior, it is still significantly better than the other configurations.

![Fig. 4: Average adaptation: agent deficit, with errors and delays](image)

![Fig. 5: Average adaptation: agent excess, with errors and delays](image)
VIII. CONCLUSIONS

We compare adaptable decentralized task allocation in approaches amenable to “ongoing” tasks: Response Threshold Reinforcement (RTR), a variant of Ant Colony Optimization (ACO), and a variant of Learning Automata called Linear Reward-Penalty ($L_{R-P}$). We assess their ability to adapt and respond to dynamic needs by comparing task performance and task switching rates over time. Results show that specialization and respecialization capabilities differ drastically, with ACO and $L_{R-P}$ vastly outperforming RTR under most conditions.

ACO and $L_{R-P}$ demonstrate better adaptability, characterized by performance fluctuations away from 100% diminishing over time. RTR shows persistent fluctuations, with agents continuing to switch tasks without attenuation. Without communication, adaptation is achieved by trial and error; agents making fewer task switches present a more stable environment for each other to adapt to over time. Under ACO and $L_{R-P}$, learning decreases as we get closer to specialization ($P_{a,t} = 0.0$ or 1.0), allowing for a gradual approach of the ideal task allocation. RTR employs statically-sized threshold updates, which may lead to continuously overshooting the target behavior. Altering RTR threshold updates to be dependent on the current value may help reduce performance fluctuations and task switching, but may help reduce performance fluctuations and task switching, but may also result in absorbing states (if $P_{a,t} = 0.0$ or 1.0) or an inability to idle (if $P_{a,t}$ is not allowed to reach 0).

The tested approaches are unable to proportionately distribute the workforce when there are insufficient agents to fulfill all demands. Resulting work deficits vary across tasks, with some tasks achieving ideal performance, and others being neglected, which could threaten system stability if the neglected tasks were crucial. It would then be of interest to incorporate relative importance for “ongoing” tasks (defined for the system or for the individual agents).

While having excess agents should increase robustness to agent failure, excess work causes interference and can, in fact, decrease performance. When excess workers cannot decide that their actions are not currently required and remain idle and out of the way, the group cannot settle on the correct allocation. While ACO generally outperformed RTR in the other tests, under excess workforce RTR was the only approach capable of finding the correct allocation, as ACO and $L_{R-P}$ probabilities cannot reach zero. Consequently, further research is needed into efficiently handling decentralized excess. Consider that while ACO and $L_{R-P}$ formulations could be altered to allow action probabilities to reach zero so as to allow idling, $P_{a,t} = 0$ would be an absorbing state, because probabilities are updated as a portion of their current values.

REFERENCES