A Field-based Approach for Runtime Replanning in Swarm Robotics Missions

Gianluca Aguzzi*, Martina Baiardi*, Angela Cortecchia*, Branko Miloradovic†, Alessandro Papadopoulos†, Danilo Pianini*, Mirko Viroli*

*Department of Computer Science and Engineering, University of Bologna, Cesena, Italy
{gianluca.aguzzi, m.baiardi, angela.cortecchia, danilo.pianini, mirko.viroli}@unibo.it

†Department of Electrical and Computer Engineering, Mälardalen University, Mälardalen, Sweden
{branko.miloradovic, alessandro.papadopoulos}@mdu.se

***Abstract*—Ensuring mission success for multi-robot systems operating in unpredictable environments requires robust mechanisms to react to unpredictable events, such as robot failures, by adapting plans in real-time. Adaptive mechanisms are especially needed for large teams deployed in areas with unreliable network infrastructure, for which centralized control is impractical and where network segmentation is frequent. This paper advances the state of the art by proposing a field-based runtime task replanning approach grounded in aggregate programming. Through this paradigm, the mission and the environment are represented by continuously evolving fields, enabling robots to make decentralized decisions and collectively adapt the ongoing plan. We compare our approach with a simple late-stage replanning strategy and an oracle centralized continuous replanner. We provide experimental evidence that the proposed approach achieves performance close to the oracle if the communication range is sufficient, while significantly outperforming the baseline even under sparse communication. Additionally, we show that the approach can scale well with the number of robots.**

***Index Terms*—multi-robot coordination, runtime adaptation, self-organization, runtime replanning,**

I. Introduction

Coordinating teams of autonomous robots to execute complex missions, such as environmental monitoring, search-and-rescue, and warehouse logistics, hinges on solving Multi-Robot Task Allocation (MRTA) and path-planning problems under spatial, temporal, and resource constraints [1], [2]. Classical MRTA methods compute task assignments and trajectories offline, assuming a static environment and perfect robot reliability; in practice, however, hardware malfunctions, energy depletion, and communication blackouts frequently invalidate precomputed plans when the mission runs [3]. To maintain mission performance in face of such disturbances, online planners must integrate fresh sensor data and updated robot states to trigger replanning whenever required. Two primary categories of runtime events demand this capability:

- *Robot failures.* Unexpected breakdowns or depleted batteries leave tasks uncompleted and disrupt planned routes [4].
- *Mission modifications.* Newly discovered obstacles, added tasks, or shifting priorities require rapid reallocation of robot efforts [5].

Absent timely replanning, these disturbances cascade, causing severe inefficiencies or outright mission failure. This type of event can lead to two main outcomes, which may also occur together: *task replanning* (reassigning tasks to different robots and consequently recomputing the path) and *path replanning* (recalculating a robot's route to a task, usually because the environment has changed). In this paper, we use the term *replanning* specifically to refer to *task replanning*.

Centralized replanning architectures leverage global information to compute high-quality plans, but exhibit single points of failure and suffer superlinear growth in computational and communication load as team size increases [5]. Semi-centralized schemes, such as rotating leaders or intermittent coordinators, mitigate some vulnerabilities but remain susceptible to network partitions and overload under sparse connectivity.

Distributed planning frameworks promise resilience and near-linear scalability by delegating decision-making across the robot team [6]. Existing distributed heuristics, however, typically separate replanning triggers from plan synthesis and degrade in solution quality as the number of robots or tasks grows, leading to brittle behavior under high failure rates and dynamic mission requirements.

This paper introduces a *fully runtime-based, field-guided replanning approach* grounded in Aggregate Computing (AC) [7], a functional macro-programming paradigm in which mission objectives and the environmental state are represented as continuously evolving computational fields over space and time [8], [7], [9]. In this approach, each robot executes the same aggregate program asynchronously and exchanges only local field values with its neighbors. Through this representation, fields continuously encode the distributed system state, enabling both replanning decisions and plan construction to determine:

- **When** to trigger replanning by monitoring fields that represent the global view of active robots and their positions—changes in these fields (e.g., due to robot

This work is funded by: The Knowledge Foundation (KKS), MARC project No. #20240011, The Swedish Research Council (VR), PSI project No. #2020-05094, "WOOD4.0 - Woodworking Machines for Industry 4.0", (CUP E69J22007520009) Emilia-Romagna regional project, call 2022, art. 6 L.R. N. 14/2014, and the Italian PRIN project "CommonWears" (2020 HCWWLP).

failures or new arrivals) automatically initiate distributed consensus on the need for replanning;
- **How** to construct new task assignments and path plans using fields that maintain collective knowledge of task completion status and robot capabilities across the network.

By unifying replanning triggers and plan synthesis within a single field-based abstraction, this mechanism eliminates centralized bottlenecks while ensuring that runtime complexity scales linearly with team size, because they leverage only the messages sent from their neighbors.

This paper makes three key contributions:

1) A novel runtime replanning mechanism based on computational fields.
2) An extensive simulation study demonstrating that the field-guided approach maintains near-optimal mission completion times under moderate communication ranges and high failure rates, outperforming late-stage and semi-centralized baselines.
3) A comparative evaluation against an oracle-supported centralized replanner that quantifies the trade-off between communication range and performance degradation.

By embedding both replanning triggers and plan generation within a single aggregate program, this field-guided strategy delivers robust, fully distributed coordination suitable for large-scale, unpredictable environments.

II. Background

A. Planning and Replanning in Multi-Robot Systems

Planning and task allocation are central challenges in multi-robot systems, where teams of robots must efficiently distribute and complete tasks despite communication, computation, and energy constraints. Early formulations modeled planning as a Travelling Salesman Problem (TSP), where a single robot visits a set of locations at minimal cost [10]. The problem was later extended to multiple agents through the Multiple Travelling Salesman Problem (mTSP) [11], capturing the need to partition and sequence tasks across several robots. In this work, our problem can be mapped into mTSP.

Solving the mTSP is considerably more complex than the single-agent case and has motivated a variety of approaches, including Genetic Algorithms (GAs) [12], [13], [14], auction-based methods [3], and consensus-based bundling algorithms for limited replanning [15]. However, many of these strategies assume perfect communication among agents, an assumption often violated in field deployments where network segmentation, delays, or losses are common. In certain scenarios, handling communication constraints is critical. Auction-based planners [3] typically require frequent exchanges, while rendezvous-based approaches [16] mitigate prolonged disconnections at the cost of detours and delays. Some recent methods [17] consider failures explicitly, but often assume disruptions can be rapidly detected and shared among agents—a nontrivial requirement in sparse or degraded networks.

In practical settings (such as search and rescue [18] and autonomous environmental monitoring [19]) task requirements, robot availability, and environmental conditions can change unpredictably during mission execution. Thus, initial plans must often be revised or rebuilt dynamically, motivating the development of robust runtime replanning strategies.

Runtime replanning enables multi-robot systems to adapt dynamically to failures, new tasks, and environmental changes during mission execution. In contrast to static offline planning, runtime replanning maintains mission effectiveness in non-stationary, unpredictable settings. While centralized replanning can achieve globally optimal plans in fully connected networks [20], [14], it becomes impractical in scenarios with limited bandwidth, mobile agents, or high failure rates. Thus, decentralized self-adaptive replanning, often inspired by aggregate computing [7] and gossip-based consensus protocols [21], offers a promising alternative for resilient multi-robot mission management.

Recent research has further explored the challenges of runtime replanning under uncertainty. Bramblett et al. [22] proposed an online epistemic replanning framework that enables multi-robot teams to adapt their plans based on evolving knowledge and beliefs about the environment and system state, improving resilience to information delays and failures. Similarly, Frasheri et al. [23] introduced GLocal, a hybrid approach that combines global and local planning strategies to reallocate tasks during mission execution dynamically, balancing the benefits of centralized optimization and distributed adaptation.

Multi-Agent Planning (MAP) frameworks [24] further structure the replanning process by distinguishing between cooperative distributed planning, where agents collectively synthesize a global plan, and negotiated distributed planning, where agents pursue individual goals while coordinating interactions [25]. Recent surveys [26], [27] classify MAP approaches based on agent autonomy, communication mechanisms, privacy preservation, and plan synthesis models.

In these settings, replanning must trade off between responsiveness, computational efficiency, and communication overhead, while maintaining a consistent and convergent view of the mission among participating robots.

B. Aggregate Computing in a nutshell

In this section, we will provide a brief introduction to AC and its execution model, covering the concepts necessary to understand the proposed approach.

AC [7] is a functional macro-programming approach for developing self-adaptive systems. It is based on the notion of *computational field* [8]: a distributed data structure that maps points in space and time to values. Building on the concept of computational fields, the Field Calculus (FC) [28], introduces higher-order functional programming abstractions that enable manipulation and composition of fields, relying on local observations of the fields' values. This paradigm has been successfully applied in the past in several domains, including crowd management [7], disaster detection and response [29], and multi-drone coordination [30].

a) Execution model: In AC, all agents share the same program, which is executed in asynchronous[1] rounds by every agent indefinitely[2]. Each iteration of the loop consists of three phases: *sense*, *compute*, and *share*. In the *sense* phase, the agent collects information from its local environment (including its previous state, if any) and from neighboring agents (by processing any received messages). During the *compute* phase, the agent executes the aggregate program using the collected information, producing a local value, an updated state, and messages to be sent to neighbors. Finally, in the *share* phase, the agent transmits these messages to its neighbors.

b) Evolution in time and space: To evolve a field in time and space, AC uses the **share** function [32], modeling a stateful communication among neighboring agents. **share** accepts a local value and a function that computes over a field of values of the same type, producing a new value which is returned and shared with neighbors:

```
fun <T> share(initial: T, op: (Field<T>) -> T): T
```

Share can be used, for instance, to build a self-healing version of the Bellman-Ford algorithm [33] to estimate the hop distance from the closest device where a condition holds [34]:

```
import Double.POSITIVE_INFINITY as infinity
fun bellmanFord(cond: Boolean): Double =
  share(infinity){ d ->
    val minDistance = (d + 1.0).minValue(infinity)
    if (cond) 0 else minDistance
  }
```

c) Computing distances: If we can estimate distances from each neighbor, we can extend the previous algorithm to estimate the *physical* distance between two situated agents. Given a `gps()` function (returning coordinates) and a `haversine()` function (computing great-circle distance), we can estimate distances as:

```
fun distanceTo(cond: Boolean) = share(gps() to inf) {
    val dists = it.first.mapValues {
      haversine(gps(), it.value)
    }
    val minDist = (dists + it.second).minValue(inf)
    if (cond) 0 else minDist
}
```

d) Gossiping values: The previous example is *self-stabilizing*—that is, they guarantee that after any transient fault or change in the system, the computed values will eventually converge to a correct and stable state [35]. Self-stabilization is a crucial property for distributed algorithms, especially in dynamic environments, as it ensures resilience to failures and network changes. However, in some scenarios, we may need to propagate information that does not self-correct if the underlying data changes or disappears—for example, when we want to quickly spread a value throughout the network, but do not require the system to retract or update that value if it becomes obsolete. In these cases, we use *non-self-stabilizing* gossip protocols that can be implemented as follows:

```
fun gossip<T>(local: T,  combine: (T, T) -> T) =
  share(local) { it.foldValues(local, combine) }
```

Where `local` is the initial value contributed by each node, For example, collecting all device identifiers in the system can be achieved as follows:

```
val allIds = gossip(setOf(localId), Set::plus)
```

Being the `gossip` function *non-self-stabilizing*, it does not adapt to devices leaving the system or retracting their previous information. For instance, if the temperature suddenly drops, the system will keep reporting the previous maximum value. This limitation of non-self-stabilizing gossip protocols is well-known; self-stabilizing variants have been proposed [36].

e) Self-stabilizing building blocks: Since self-stabilization is such an important property for distributed systems, in the context of AC a library of fundamental building blocks has been identified [34]. It includes functions to propagate information (`gradientCast`, a family of algorithms which includes the adaptive Bellman-Ford algorithm introduced before); to accumulate information (`convergeCast`), and to break the network symmetry (`sparseChoice`, typically used to elect leaders). Notably, the functional composition of these self-stabilizing blocks is in turn guaranteed to be self-stabilizing. As a consequence, versions of the gossip protocol built upon composition of `gradientCast` and `convergeCast` are self-stabilizing.

f) Network partitioning and leader election: Self-stabilizing leader election and network partitioning can be performed in a distributed fashion in AC using `boundedElection` [37]. Provided a measure of "strength" for each candidate device and a maximum diameter, `boundedElection` splits the network into partitions large at most as the diameter, automatically electing new leaders in case of failures or topology changes, with no central coordination.

III. Problem Formulation

The mission planning problem can be described as follows. A team of m robots $\mathcal{R} = \{r_1, \ldots, r_m\}$ must collectively service a set of n tasks $\mathcal{T} = \{t_1, \ldots, t_n\}$, each task being visited exactly once by exactly one robot. Every robot r departs from one of the designated source depots $\sigma \in \Sigma$ and must finish at one of the designated destination depots $\delta \in \Delta$.

The superset containing all of the tasks $\mathcal{T}$ and depots is defined as $\widetilde{\mathcal{T}} := \mathcal{T} \cup \{\Sigma, \Delta\}$. For simplicity, a superset containing all source depot and tasks is defined as $\mathcal{T}^{\Sigma} := \mathcal{T} \cup \Sigma$. Similarly, a superset containing all elements of the destination depot and cities is defined as $\mathcal{T}^{\Delta} := \mathcal{T} \cup \Delta$. Traveling between any two locations i and j (whether tasks or depots) by robot r incurs a nonnegative cost ω_{ijr}, and performing a task i requires a robot-specific service time ξ_{ir}.

The decision variable $x_{ijr} \in \{0, 1\}$ defines if a robot $r \in \mathcal{R}$ performs task $j \in \widetilde{\mathcal{T}}$ immediately after task $i \in \widetilde{\mathcal{T}}$ and in that case $x_{ijr} = 1$, otherwise $x_{ijr} = 0$.

[1]Though not strictly required, devices generally operate at similar frequencies.

[2]Reactive formulations of the execution model exist [31], which, albeit increasing efficiency, leave the foundations of the execution model unchanged.

The objective is to assign to each robot an ordered route, i.e., a Hamiltonian path (a path that visits each task exactly once) through its assigned tasks, to minimize the total sum of travel and service costs across all robots, while ensuring that

1) each robot's route starts and ends at a depot,
2) no robot revisits the same location, and
3) no subtours disconnected from a depot occur.

This can be formally written as an optimization problem that minimizes the following cost function:

$$J = \sum_{r \in \mathcal{R}} \sum_{i \in \mathcal{T}^{\Sigma}} \sum_{j \in \mathcal{T}^{\Delta}} (\omega_{ijr} + \xi_{ir}) x_{ijr}, \tag{1}$$

which represents the total sum of durations of the plans of all robots, and by the following constraints:

$$\sum_{r \in \mathcal{R}} \sum_{i \in \mathcal{T}^{\Sigma}} x_{ijr} = 1, \qquad \forall j \in \mathcal{T}, \tag{2}$$

$$\sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{T}^{\Delta}} x_{ijr} = 1, \qquad \forall i \in \mathcal{T}, \tag{3}$$

$$\sum_{i \in \mathcal{T}^{\Sigma}} \sum_{j \in \Delta} x_{ijr} = 1, \qquad \forall r \in \mathcal{R}, \tag{4}$$

$$\sum_{i \in \Sigma} \sum_{j \in \mathcal{T}^{\Delta}} x_{ijr} = 1, \qquad \forall r \in \mathcal{R}, \tag{5}$$

$$\sum_{i \in \mathcal{T}^{\Sigma}} x_{ijr} = \sum_{k \in \mathcal{T}^{\Delta}} x_{jks}, \quad \forall j \in \mathcal{T}, \forall r \in \mathcal{R}, \tag{6}$$

$$u_i - u_j + |T|\, x_{ijr} \leq |T| - 1, \ \forall r \in R, \forall i, j \in T, i \neq j. \tag{7}$$

$$x_{iir} = 0, \quad \forall i \in \widetilde{\mathcal{T}}, \forall r \in \mathcal{R}, \tag{8}$$

Only one robot $r \in \mathcal{R}$ can start (Eq. 2), and end (Eq. 3) each task, and it can do it exactly once. The final destination of a robot r must always be one of the destination depots (Eq. 4), while the starting location must always be at one of the source depots (Eq. 5). It is necessary to ensure that the same robot that starts a task is the one that finishes it. This constraint is given by (Eq. 6). We eliminate possible sub-tours with the Miller-Tucker-Zemlin (MTZ) formulation (Eq. 7), where the domain of variable u_i is defined as $1 \leq u_i \leq n$. Finally, it is forbidden for robot r to travel from a task i to the same task i (Eq. 8).

In real-time operation, robot failures and other runtime disturbances invalidate portions of this plan, triggering the field-based replanning mechanisms presented in this paper.

IV. Field-based Replanning

The challenge of runtime replanning in multi-robot systems can be divided into two main parts: the distributed construction of a new plan, and the distributed agreement on whether a new plan is necessary.

The latter operation is critical when robotic swarms are operating with uncertainties. In fact, distributed replanning is an expensive operation that should be performed solely when necessary, for instance, when the information upon which the previous plan was based is no longer valid.

Leveraging the principles of aggregate computing, we propose to model the collective system state in the form of fields, in such a way that its observation can be continually performed. Changes in the observed collective state trigger the distributed replanning. In short, the idea is to achieve *distributed consensus*, either implicitly or explicitly, on *whether* there is need to replan and on *which* plan to perform. Of these, deciding *when* to replan is the most challenging task, and the main focus of this paper. We explore two primary strategies grounded in aggregate programming: a gossip-based approach and a leader-based approach.

A. Replanning Algorithm

While our primary contribution focuses on distributed coordination mechanisms for triggering the replanning, we provide a formal description of the task allocation algorithm used when replanning.

Given the current state comprising active robots $\mathcal{R}_a \subseteq \mathcal{R}$ and remaining tasks $\mathcal{T}_r \subseteq \mathcal{T}$, we employ a greedy assignment algorithm that iteratively allocates tasks to robots by minimizing marginal costs.

For each robot $r_j \in \mathcal{R}_a$ and each unassigned task $t_i \in \mathcal{T}_r$, we compute the assignment cost:

$$C(t_i, r_j) = \omega_{\mathit{current}(r_j),i,r_j} + \xi_{ir_j} + \omega_{i,\mathit{next}(r_j),r_j} \tag{9}$$

where $\mathit{current}(r_j)$ denotes the location of robot r_j at replanning time (or its last assigned task), and $\mathit{next}(r_j)$ represents the robot's next destination (either its next assigned task or destination depot δ).

The replanning procedure, formalized in Algorithm 1, maintains individual robot paths while minimizing the global mission completion time. The algorithm iteratively:

1) Identifies the robot-task pair (r^*, t^*) with minimum marginal cost;
2) Updates the robot's path by inserting task t^*;
3) Removes t^* from the set of remaining tasks.

This process continues until either all tasks are assigned or no feasible assignments remain.

Algorithm 1 Task Allocation for Replanning

Require: $\mathcal{R}_a$ (active robots), $\mathcal{T}_r$ (remaining tasks)
Ensure: Assignment matrix $X = [x_{ijr}]$

$X \leftarrow \mathbf{0}$ ▷ Initialize assignment matrix
while $\mathcal{T}_r \neq \emptyset$ **do**
  $(r^*, t^*) \leftarrow \arg\min_{r_j \in \mathcal{R}_a, t_i \in \mathcal{T}_r} C(t_i, r_j)$
  if $C(t^*, r^*) < \infty$ **then** ▷ Feasible assignment exists
    Update path of robot r^* to include task t^*
    $x_{\mathit{current}(r^*),t^*,r^*} \leftarrow 1$ ▷ Update assignment matrix
    $\mathcal{T}_r \leftarrow \mathcal{T}_r \setminus \{t^*\}$
  else
    break ▷ No feasible assignments remain
  end if
end while
return X

B. Distributed Consensus via Gossiping

In the gossip based approach (summarized in Algorithm 2), each robot participates in a continuous gossiping protocol that maintains a collective view of the system state. The approach leverages the two distinct types of gossip mechanisms for different purposes:

- *Stabilizing gossip* (`gossip`) collects and maintains consistent information about the active nodes in the system (their positions and identifiers). This view should only change when the network topology changes, when robots fail, or when new robots join the system.
- *Non-self-stabilizing gossip* (`gossipNonStab`) tracks the completion status of tasks throughout the network (`tasksDone`). This mechanism ensures that robots maintain awareness of which tasks have already been completed, preventing redundant work even when the system topology changes. In this case, for each evaluation, the robots will share their local view of the tasks they have completed (`tasksState`).

In Algorithm 2, replanning is triggered whenever a robot detects a significant change in the stabilized global view. This global view represents the collective state of the system (namely the active robots and their positions), as gathered by the gossip protocol. Stabilization occurs when no changes are perceived for a certain number of rounds. Typically, replanning is needed when the set of active robots changes, such as when robots fail or new robots join the system. When `hasChanged`(*nodeInfo*) returns true, each node independently executes the `replan` algorithm based on their shared understanding of: the current set of active robots and their positions (*nodeInfo*); and the status (completed or remaining) of all tasks in the mission (*tasksDone*). This distributed computation effectively acts as a distributed state machine, with nodes implicitly converging towards a shared understanding of the new plan driven by the convergence of the underlying fields representing the system state. When a new plan is computed, each device checks if it is consistent with the current system state via `isConsistent`(*plan*).

A *plan* is considered consistent if:

1) Each robot's belief about the paths of all robots matches what those robots have computed for themselves (i.e., if robot A believes robot B will follow path P, then robot B has indeed computed P as its own path).
2) Each robot's belief about task assignments matches what other robots believe (i.e., if robot A believes it is assigned task T, then all other robots also believe robot A is assigned task T).

If the plan is consistent, the robot begins following this new plan via `followPlan`(*plan*). Otherwise, it waits until the system state stabilizes. Through this process, each robot obtains the plan segments relevant to its role in the mission while maintaining coordination with the entire swarm.

Pros: This method offers high resilience. The failure of any single node (barring catastrophic network partitions) does not halt the replanning capability of the remaining system, as the logic is fully replicated.

Cons: Significant computational overhead arises from multiple nodes redundantly computing large parts of the global plan (or even the entire one). Reaching consensus on the system state and converging on the resulting plan execution can also introduce latency, particularly in large or highly dynamic environments where the fields take time to stabilize.

Algorithm 2 Distributed Gossip-based Runtime Replanning

Require: *localId*, *localPosition*, *tasksState*
1: *nodeInfo* ← `gossip`(*localId* ↦ *localPosition*)
2: *tasksDone* ← `gossipNonStab`(*tasksState*)
3: **if** `hasChanged`(*nodeInfo*) **then**
4: *plan* ← `replan`(*nodeInfo*, *tasksDone*)
5: **if** `isConsistent`(*plan*) **then**
6: `followPlan`(*plan*) **else** `wait`()
7: **end if**
8: **else**
9: `followPlan`(*currentPlan*)
10: **end if**

C. Leader-Based Coordination

To reduce the computational effort required by the system, a strategy could involve the dynamic designation of a robot as leader. The leader is then responsible for initiating and computing the replan. This approach leverages standard aggregate computing patterns for robust leader election, network partitioning, and bidirectional communication within a partition (cf. Section II-B).

As shown in Algorithm 3, each robot maintains its own identifier ('*localId*') and position ('*localPosition*'), while the '*networkDiameter*' parameter defines the maximum number of hops within the communication network. The algorithm uses '*nodeInfo*' to track information about nodes in the network through gossiping or converge-casting, and '*tasksDone*' to maintain the status of completed tasks as in the previous algorithm. The '*leader*' variable identifies the currently elected leader node, while '*plan*' represents the computed replanning result when a change in network state is detected.

The replanning process typically unfolds as follows:

1. *Leader election:* A self-stabilizing distributed leader election algorithm (`boundedElection`) ensures that a unique leader is elected for the entire network (the maximum distance is unbounded). In case of network segmentation, every network partition will have an elected leader.
2. *State collection:* The elected leader gathers the necessary system state information. Data collection can be implemented using `convergeCast` (cf. Section II-B) directly, or any other self-stabilizing gossiping mechanism. In this discussion, without loss of generality, we call this *self-stabilizing* collection operation `gossip`.
3. *Replanning trigger:* The leader monitors the aggregated state. Similar to the gossip approach, replanning is trig-

gered when the leader detects a significant change in the collective state, most commonly, a modification in the set of active robot IDs it perceives through the collection process, indicating that either a robot malfunctioned or left the system.

④ *Plan computation:* Upon triggering, the leader executes the computationally intensive replanning algorithm to generate the new global or partial plan.

⑤ *Plan dissemination:* The leader disseminates the newly computed plan (or relevant assignments/directives) to all other robots using `gradientCast` (cf. Section II-B).

Pros: This approach significantly reduces the overall computational load on the system, as the replanning task is performed centrally by only one robot per network partition.

Cons: This approach introduces a dynamic centralization point, namely the leader. Although robust leader election can quickly replace failed leaders, transient periods without coordination may occur. Latency is a key drawback: information must traverse the network diameter to reach the leader, causing the leader to operate on potentially outdated data and resulting in suboptimal plans. Plan dissemination also incurs a diameter-length delay. Thus, the effectiveness of this approach depends on network topology, its dynamics, and the rate of environmental or system changes.

Algorithm 3 Leader-based Runtime Replanning

Require: *localId*, *localPosition*, *networkDiameter*, *tasksState*

```
tasksDone ← gossipNonStab(tasksState)
leader ← boundedElection(networkDiameter)        ▷ ①
nodeInfo ← gossip(localId ↦ localPosition)       ▷ ②
isLeader ← leader = localId
if isLeader and hasChanged(nodeInfo) then        ▷ ③
    plan ← replan(nodeInfo, tasksDone)           ▷ ④
    gradientCast(isLeader, plan)                 ▷ ⑤
end if
followPlan(getLatestPlan())
```

V. Evaluation

In this section, we describe the experimental evaluation conducted to validate the field-based runtime replanning approaches discussed previously. The code developed for this evaluation is based on the Alchemist simulator [38] and the experimental Collektive aggregate computing framework[3]. The full implementation, experimental data, reproduction scripts, and chart generation code have been open sourced with a permissive license [4] and archived on Zenodo for future reference[5].

A. Goals

The primary goals of this evaluation are:

- *Resilience Assessment:* verify whether the proposed field-based replanning approaches adapt to robot failures, ensuring mission completion despite unforeseen disruptions.
- *Scalability Analysis:* investigate how the number of robots and tasks affect the system performance.

B. Baseline Approaches

To benchmark the performance of our proposed field-based replanning strategies, we compare them against two baseline approaches, *Oracle-based Centralized Replanning* and *Late-Stage Replanning*.

The foster assumes a centralized controller with perfect, real-time view of the entire system state (robot locations, status, and task completion). Upon detecting any robot failure, the oracle immediately recomputes the optimal plan for the remaining active robots and tasks using the same underlying planning algorithm employed by the field-based approaches, but with *complete information*. This represents an ideal upper bound on performance achievable with immediate, informed replanning.

In *Late-Stage Replanning*, robots execute their initially assigned task sequence without adaptation during runtime. Only after a robot completes its assigned sequence (or reaches its final destination) it checks for any remaining uncompleted tasks in the mission. If any task is pending, the robots collectively replan to address it. This strategy minimizes runtime overhead but is expected to be inefficient in handling mid-mission failures.

Note that both strategies are *immune to network segmentation*, the former because we let the replanner know the state of the world without restrictions, and the latter because it performs no communication but at the late stage.

C. Experimental Parameters

Each simulation experiment involves a set of robots $\mathcal{R}$ and tasks $\mathcal{T}$ deployed in a square environment. Key parameters are summarized in Table I and detailed below:

Robots are initially placed randomly within a small starting area, representing the source depot (σ). The tasks are randomly distributed throughout the environment. The target destination (δ) is fixed. A task is considered completed when a robot

TABLE I
Simulation Parameters

Parameter	Value(s)
Number of Robots (m)	{5, 10, 20, 40}
Task-to-Robot Ratio (n/m)	{0.5, 1, 2, 4}
Resulting Task Counts (n)	Computed as mn/m
Environment Size	$[200 \times 200]$m
Robot Speed	Constant $0.5\,\mathrm{m\,s^{-1}}$
Communication Range (R)	$\{20\,,\ 50\,,\ 100\,,\ \infty\}$ m
Initial Robot Deployment Area	$[5 \times 5]$m, center (-95, -95)
Target Destination Area (δ)	Point (100, 100)
Mean time between failures (λ^{-1})	{1000, 2000, 5000, 50 000} s
Random Seeds	32
Simulation Termination Condition	Task completion stable for 1200 s

[3]https://github.com/Collektive/collektive
[4]https://github.com/angelacorte/experiments-2025-acsos-robots
[5]https://doi.org/10.5281/zenodo.16578273

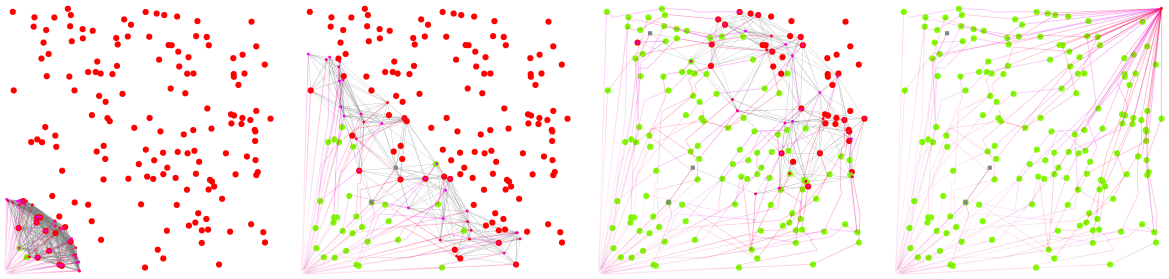

Fig. 1. Simulation snapshots. Tasks are represented by red dots, which turn green when completed. Pink lines show robot trajectories, gray boxes mark inactive/failed robots. Gray lines show direct communication lines between robots.

to remain within $\varepsilon = 10\,\mathrm{cm}$ of a task location for at least $\tau = 60\,\mathrm{s}$. Robots move at a constant speed of $0.5\,\mathrm{m\,s^{-1}}$.

Robots can communicate directly with neighbors within their radio range R, simulating technologies like Bluetooth or Wi-Fi in ad-hoc mode. Multi-hop communication is implicitly handled by aggregate computing. We test sparse ($20\,\mathrm{m}$) and medium ($50\,\mathrm{m}$, $100\,\mathrm{m}$) ranges and complete connection (∞).

Robot failures are modeled as a Poisson process, where each robot fails independently according to an exponential distribution. For each robot $r \in \mathcal{R}$, the time to failure T_r is a random variable:

$$\mathbb{P}(T_r > t) = e^{-\lambda t}, \quad t \geq 0 \tag{10}$$

where $\lambda > 0$ is the failure rate parameter, and λ^{-1} is the mean time between failures.

We repeat each experiment multiple times (32) using different random seeds. Different seeds change the initial placement of robots and tasks, the failing robots and failure times, and the scheduling of the aggregate computing rounds.

Figure 1 shows snapshots from a simulation run with n =40, n/m=4, and R =50.

D. Metrics

We evaluate the performance of the different replanning strategies using two key metrics:

- *Mission Stable Time (T_s):* the elapsed simulation time until all possible tasks are completed and the system reaches a steady state with no further robot movement. This metric captures the overall mission efficiency. Lower T_s values indicate superior performance, as it is a proxy for quicker mission completion despite disruptions.
- *Replanning Count (C):* the number of replanning events throughout the simulation, calculating the average number of replans triggered per robot. This metric quantifies the computational overhead of each strategy and its responsiveness to changing conditions.

E. Experimental Results

This section compares the proposed field-based runtime replanning approaches with the baselines from Section V-B. Figure 2 shows the *Mission Stable Time* (T_s). Figure 3 shows the average *Replanning Count* per robot.

a) Communication Range Impact on Performance: Data shows that the communication range is critical for both decentralized approaches. With extensive communication ranges, both approaches achieve performance comparable to the centralized Oracle, indicating that the replanning time detection is essentially correct. However, performance degrades as the communication range decreases. At $R = 100\,\mathrm{m}$, both approaches still significantly outperform the baseline, while at $R = 20\,\mathrm{m}$ performance deteriorates substantially: continuous network segmentation leads to frequent wrong assumptions of robot failures, causing a non-needed replanning, which results in tasks being assigned to multiple robots due to inconsistent views of the system state. With large teams and high task loads (bottom right corner of Figure 2 and Figure 3), performance degrades to the point where executing a late-stage replanning performs better than the field-based approaches. Sufficient connectivity is thus a fundamental requirement for decentralized field-based MRTA.

b) Resilience to failure: Data shows that the gossip-based approach (top in Figure 2) is significantly more resilient to robot failures compared to the leader-based (bottom in Figure 2) approach. Even with very high failure rates ($\lambda^{-1} = 1000\,\mathrm{s}$), it consistently outperforms the Baseline when connectivity is sufficient. Performance scales reasonably with increasing node count and task load, showing no signs of catastrophic degradation. The Leader-based approach performs well until failure rates rise (for each plot, the failure rate increases from left to right), then performance degrades significantly. This vulnerability stems from coordination gaps during leader failure and re-election periods. At lower failure rates ($\lambda^{-1} \geq 5000\,\mathrm{s}$), both approaches deliver comparable performance at the same communication ranges.

c) Scalability: Increasing m or n/m increases T_s across all approaches (as n increases). Both field-based methods scale better than the baseline in for moderate failure rates ($\lambda^{-1} \geq 5000\,\mathrm{s}$) and reasonable communication range ($R \geq 50\,\mathrm{m}$). In the extreme case ($m = 40$, $n/m = 4 \Rightarrow n = 160$, right bottom corner of Figure 2), both gossip- and leader-based approaches complete their mission in about $1400\,\mathrm{s}$, while the baseline takes longer than $2000\,\mathrm{s}$.

d) Replanning Overhead: The Gossip approach consistently generates significantly more replanning events compared to the leader-based approach, often by an order of magnitude.

Fig. 2. Mission completion time T_s over mean time between failures λ^{-1} For the gossip-based approach (top) and the leader-based approach (bottom). Differently colored bars represent different values of R (the baseline and the oracle ignore R). Each subplot frames a specific configuration of the robot count m and the task-to-robot ratio n/m. With sufficient communication range ($R \geq 50\,\mathrm{m}$), both field-based methods outperform the baseline and approach Oracle performance. gossip-based offers greater resilience under frequent failures, while leader-based is more efficient but less robust to leader transitions. Thus, gossip-based trades higher overhead for robustness; leader-based trades resilience for efficiency.

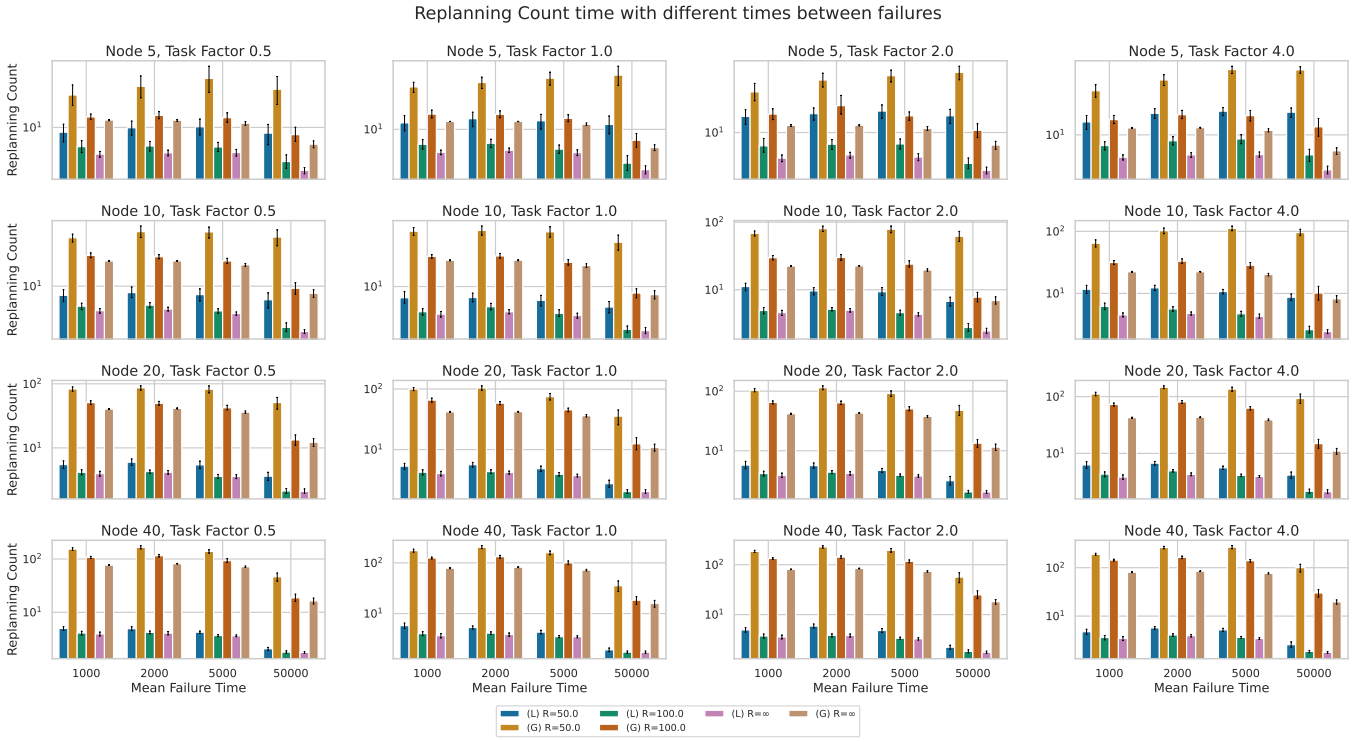

Fig. 3. Replanning events C over mean time between failures λ^{-1} comparing the gossip-based approach (G) and the leader-based approach (L) for a selected set of communication ranges R (50 m, 100 m, and fully connected). Each subplot frames a specific configuration of the robot count m and the task-to-robot ratio n/m.

In the Gossip approach, every robot independently monitors the collective state and may trigger replanning upon detecting changes, whereas the Leader approach centralizes the replanning management in a single elected node. The replication induced by the gossip-based approach contributes to its resilience, but also leads to higher computational overhead.

e) Corner Cases: Data reveals interesting corner cases where the field-based approaches underperform compared to the baseline. In scenarios with few tasks per robot ($n/m = 0.5$) and long robot mean failure times ($\lambda^{-1} = 50\,000$ s), where few tasks exist relative to robots and failures are rare, the gossip-based approach sometimes performs worse than the Baseline, even for long communication ranges. This counterintuitive result stems from unnecessary replanning overhead (see Figure 3): when robots rarely fail, the Baseline's single initial plan remains effective, while the Gossip approach continuously monitors for changes and occasionally triggers replanning due to transient field perturbations (e.g., temporary disconnections). Even in fully connected networks, the Gossip approach sometimes underperforms the baseline when failures are rare because achieving distributed consensus introduces latency as field values propagate and stabilize. During this consensus-building period, robots may temporarily follow inconsistent plans, creating inefficiencies that the simpler baseline approach avoids. Fundamentally, the baseline is optimized for failure-free scenarios, while field-based approaches are designed for early replanning upon failures. When failures are rare, continual monitoring leads to unnecessary computational overhead.

f) Trade offs: Data shows that both field-based replanning approaches effectively handle runtime robot failures, outperforming the late-stage replanning baseline under most conditions. The Gossip approach offers superior resilience, particularly under high failure rates, at the price of a substantial replanning overhead. Conversely, the Leader-based approach is computationally more efficient but exhibits greater sensitivity to leader failures and short communication ranges. Selecting between these approaches is a matter of balancing resilience against computational efficiency. Specific mission profiles and anticipated environmental conditions could steer the choice of approach. Both methods demonstrate the potential of AC for the design of robust and efficient decentralized multi-robot coordination in unpredictable environments.

VI. Conclusion

This paper presented a novel approach to runtime replanning for multi-robot systems based on Aggregate Computing. We introduced two alternative field-based coordination mechanisms, based respectively on gossip and distributed leader election, that enable decentralized adaptation to robot failures and mission changes with no central control.

Our experimental results demonstrate that both field-based approaches significantly outperform a late-stage replanning baseline when the communication range is sufficiently large and failures are a realistic concern. The gossip-based approach shows superior resilience under high failure rates, maintaining performance close to an oracle-supported centralized replanner when communication connectivity is adequate, while the leader-based approach is more computationally efficient but vulnerable during leader transitions.

The primary limitation of our approach is its dependency on sufficient communication connectivity. With sparse communication (20m range in our experiments), performance deteriorates substantially, sometimes falling below the baseline.

Future work will investigate hybrid approaches that adaptively switch between gossip and leader-based coordination based on network conditions and perceived failure rates. We also plan to investigate hierarchical coordination structures that could better balance computational efficiency and resilience. In particular, it would be worth exploring variants of the leader strategy with multiple leaders, rather than a single leader, to better handle network segmentation and leader failures. Finally, we plan to address the "reality gap" by validating our approaches on real robotic platforms, ensuring their robustness and effectiveness beyond simulation.

REFERENCES

[1] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.

[2] G. A. T. Korsah, A. Stentz, and M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," *International Journal of Robotics Research*, vol. 32, no. 12, pp. 1495–1512, 2013.

[3] M. B. Dias, R. Zlot, N. Kalra, and A. Stentz, "Market-based multirobot coordination: A survey and analysis," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1257–1270, 2006.

[4] J. Yu and S. M. LaValle, "Planning optimal paths for multiple robots on graphs," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 3612–3617.

[5] L. E. Parker, "Distributed intelligence: Overview of the field and its application in multi-robot systems," in *AAAI Fall Symposium on Regarding the Intelligence in Distributed Intelligent Systems*, 2007.

[6] Y. Cao, A. S. Fukunaga, and A. B. Kahng, "Cooperative mobile robotics: Antecedents and directions," *Autonomous Robots*, vol. 4, no. 1, pp. 7–27, 1997.

[7] J. Beal, D. Pianini, and M. Viroli, "Aggregate programming for the internet of things," *Computer*, vol. 48, no. 9, pp. 22–30, 2015.

[8] M. Mamei and F. Zambonelli, "Programming pervasive and mobile computing applications: The TOTA approach," *ACM Trans. Softw. Eng. Methodol.*, vol. 18, no. 4, pp. 15:1–15:56, 2009.

[9] M. Viroli, J. Beal, F. Damiani, G. Audrito, R. Casadei, and D. Pianini, "From distributed coordination to field calculus and aggregate computing," *Journal of Logical and Algebraic Methods in Programming*, vol. 100, p. 100429, 2019.

[10] E. L. Lawler, "The traveling salesman problem: a guided tour of combinatorial optimization," *Wiley-Interscience Series in Discrete Mathematics*, 1985.

[11] T. Bektas, "The multiple traveling salesman problem: an overview of formulations and solution procedures," *Omega*, vol. 34, no. 3, pp. 209–219, 2006.

[12] M. Gendreau, A. Hertz, and G. Laporte, "Parallel tabu search for the vehicle routing problem with time windows," *Transportation Science*, vol. 33, no. 1, pp. 107–122, 1999.

[13] J.-Y. Potvin, "Vehicle routing: a review of the problem and solution techniques," in *European Journal of Operational Research*, 1996.

[14] B. Miloradović, B. Çürüklü, M. Ekström, and A. V. Papadopoulos, "GMP: A genetic mission planner for heterogeneous multirobot system applications," *IEEE Transactions on Cybernetics*, vol. 52, no. 10, pp. 10 627–10 638, 2021.

[15] L. Bertuccelli and J. How, "A decentralized auction algorithm for multi-UAV task allocation," in *AIAA Guidance, Navigation and Control Conference*, 2009.

[16] S. L. Smith and D. Rus, "Multi-robot monitoring in dynamic environments with guaranteed currency of observations," in *49th IEEE conference on decision and control (CDC)*. IEEE, 2010, pp. 514–521.

[17] C. Zhang, J. Lee, and P. Tsiotras, "Multi-agent path planning with failure probabilities," in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2020.

[18] N. Michael, J. Fink, and V. Kumar, "Distributed multi-robot task assignment and formation control," *Proceedings of Robotics: Science and Systems (RSS)*, 2008.

[19] G. A. Hollinger, A. Pereira, J. Binney, and G. S. Sukhatme, "Autonomous data collection from underwater sensor networks using acoustic communication," *IEEE Journal of Oceanic Engineering*, vol. 38, no. 3, pp. 632–644, 2013.

[20] C. Ramirez-Atencia, J. Garcia-Nieto, and E. Alba, "A multi-objective evolutionary algorithm for multi-uav mission planning," *Applied Soft Computing*, vol. 61, pp. 1071–1088, 2017.

[21] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2508–2530, 2006.

[22] L. Bramblett, B. Miloradović, P. Sherman, A. V. Papadopoulos, and N. Bezzo, "Robust online epistemic replanning of multi-robot missions," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024, pp. 13 229–13 236.

[23] M. Frasheri, B. Miloradović, L. Esterle, and A. V. Papadopoulos, "GLocal: A hybrid approach to the multi-agent mission re-planning problem," in *2023 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2023, pp. 1696–1703.

[24] M. Crosby, M. Rovatsos, and R. Petrick, "Automated agent decomposition for classical planning," in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2013.

[25] D. Rotondi, F. Viti, and M. Gendreau, "Distributed negotiated multi-agent planning: A literature review," *Transportation Research Part C: Emerging Technologies*, vol. 57, pp. 142–160, 2015.

[26] J. Benton, A. Coles, and A. Coles, "Temporal planning with preferences and time-dependent continuous costs," *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 2–10, 2012.

[27] A. Torreno, E. Onaindia, and O. Sapena, "Cooperative multi-agent planning: A survey," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, pp. 1–32, 2017.

[28] G. Audrito, M. Viroli, F. Damiani, D. Pianini, and J. Beal, "A higher-order calculus of computational fields," *ACM Trans. Comput. Log.*, vol. 20, no. 1, pp. 5:1–5:55, 2019.

[29] G. Aguzzi, R. Casadei, D. Pianini, and M. Viroli, "Dynamic decentralization domains for the internet of things," *IEEE Internet Comput.*, vol. 26, no. 6, pp. 16–23, 2022.

[30] D. Grushchak, J. Kline, D. Pianini, N. Farabegoli, G. Aguzzi, M. Baiardi, and C. Stewart, "Decentralized multi-drone coordination for wildlife video acquisition," in *International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS*. IEEE, 2024, pp. 31–40.

[31] R. Casadei, F. Dente, G. Aguzzi, D. Pianini, and M. Viroli, "Self-organisation programming: A functional reactive macro approach," in *International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS*. IEEE, 2023, pp. 87–96.

[32] G. Audrito, J. Beal, F. Damiani, D. Pianini, and M. Viroli, "The share operator for field-based coordination," in *21st IFIP WG 6.1 International Conference on Coordination Models and Languages (COORDINATION)*, ser. Lecture Notes in Computer Science, vol. 11533. Springer, 2019, pp. 54–71.

[33] R. Bellman, "On a routing problem," *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[34] M. Viroli, G. Audrito, J. Beal, F. Damiani, and D. Pianini, "Engineering resilient collective adaptive systems by self-stabilisation," *ACM Trans. Model. Comput. Simul.*, vol. 28, no. 2, pp. 16:1–16:28, 2018.

[35] E. W. Dijkstra, "Self-stabilizing systems in spite of distributed control," *Commun. ACM*, vol. 17, no. 11, pp. 643–644, 1974.

[36] D. Pianini, J. Beal, and M. Viroli, "Improving gossip dynamics through overlapping replicates," in *18th IFIP WG 6.1 International Conference on Coordination Models and Languages (COORDINATION)*, vol. 9686. Springer, 2016, pp. 192–207.

[37] D. Pianini, R. Casadei, and M. Viroli, "Self-stabilising priority-based multi-leader election and network partitioning," in *International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS*. IEEE, 2022, pp. 81–90.

[38] D. Pianini, S. Montagna, and M. Viroli, "Chemical-oriented simulation of computational systems with alchemist," *Journal of Simulation*, vol. 7, no. 3, p. 202–215, Aug. 2013.