GLocal: A Hybrid Approach to the Multi-Agent Mission Re-Planning Problem

Mirgita Frasheri[1], Branko Miloradović[2], Lukas Esterle[1], Alessandro V. Papadopoulos[2]

***Abstract*—Multi-agent systems can be prone to failures during the execution of a mission, depending on different circumstances, such as the harshness of the environment they are deployed in. As a result, initially devised plans for completing a mission may no longer be feasible, and a re-planning process needs to take place to re-allocate any pending tasks. There are two main approaches to solve the re-planning problem (i) global re-planning techniques using a centralized planner that will redo the task allocation with the updated world state and (ii) decentralized approaches that will focus on the local plan reparation, i.e., the re-allocation of those tasks initially assigned to the failed robots, better suited to a dynamic environment and less computationally expensive. In this paper, we propose a hybrid approach, named GLocal, that combines both strategies to exploit the benefits of both, while limiting their respective drawbacks. GLocal was compared to a planner-only, and an agent-only approach, under different conditions. We show that GLocal produces shorter mission make-spans as the number of tasks and failed agents increases, while also balancing the trade-off between the number of messages exchanged and the number of requests to the planner.**

***Index Terms*—Multi-Agent Systems, Autonomous Agents, Centralized Planning, Decentralized Planning**

I. Introduction

Multi-Agent Systems (MASs) usually consist of a large number of agents assigned to perform certain tasks, that ultimately lead to the fulfillment of several competitive goals, or in the case of this paper, one common goal [16], otherwise referred to as a mission [11]. Should agents be embodied, the system is referred to as Multi-Robot System (MRS). In order for a MAS/MRS to achieve a given goal, a plan should be devised that prescribes what should be done and by whom. Such a plan should be optimal, or close to optimal, and to this end automated planning techniques can be used [6]. However, the search for optimality remains the main challenge, and is an NP-hard problem when dealing with MAS. While this is not problematic for the creation of an initial plan for a mission with a small number of tasks, for larger task-sets only sub-optimal solutions may exist.

It is common in certain applications for agents to be deployed in dynamic environments where they operate with limited amounts of information, and are prone to partial or full failures. When this is the case, the initial plan is no longer a viable option, and in order to adapt to the new circumstances a new plan (a re-plan) should be created. Re-planning is an activity that takes place online, during mission execution, thus the time it takes to produce it is crucial, even if only sub-optimal solutions are feasible. In other words, when it comes to re-planning time-limitations outweigh the need for optimality.

In general, mission planning techniques are either centralized or distributed. Centralized approaches make use of all of the information, thus leading to higher quality solution at the expense of being more computationally demanding compared to distributed algorithms. In dynamic environments, where changes are bound to happen often, a centralized planner would have to re-plan every time there is a change, thus amassing the time delays required for every single re-plan. Additionally, should the communication between robots and planner be obstructed, the re-plan cannot be delivered, potentially leaving said robots idle for indefinite amounts of time. In such circumstances, distributed approaches can be useful to mitigate time delays, by allowing agents to collaborate locally with each other in order to repair the plan.

The trade-off between the optimality of centralized approaches, and the flexibility and robustness to potential failures of distributed approaches, is the main focus of this paper. To this end we propose a hybrid approach for multi-agent mission planning called GLocal, that aims to exploit the advantages of both approaches, i.e., solution quality and robustness while limiting their inherent disadvantages. We assume agents that are either fully operational or fully broken, and enable agents to repair the plan locally through negotiation in case of an agent(s) failure. Should this fail, the centralized planner will be invoked, thus the system will switch from a local to a global strategy.

The main contribution of this paper is a comprehensive analysis of the impact of the number of re-plan calls to the centralized planner by agents on the overhead and the overall quality of the solution. This is done through simulations implemented in the Gama platform [15] where GLocal was compared to centralized and distributed planning approaches for a different number of failures, and problem sizes.

The rest of this paper is organised as follows. Section II provides the background, with the problem formulation given in Section III. The centralized and distributed approaches are described in Sections IV and V respectively. Sections VI and VII contain a detailed account on the simulation design and obtained results. A brief account on related work is given in Section VIII, with the paper concluding in Section IX.

We would like to thank the Poul Due Jensen Foundation that funds our basic research at Aarhus University, for the engineering of digital twins, and Innovation Fund Denmark for funding the AgroRobottiFleet project. This work was supported by the Swedish Research Council (VR) with the PSI project (No. #2020-05094), by the Knowledge Foundation (KKS) with the FIESTA project (No. #20190034).

[1]DIGIT, Dep. of Elect. and Comp. Eng., Aarhus University, Denmark.

[2]Dep. of Intelligent Future Technologies, Mälardalen University, Sweden.

II. Background

In real world applications, the operation of agents or robots can be disrupted by environment changes, which occur regardless of the agent's activities. Disruption can also occur due to unforeseen events such as faulty sensors or actuators, thus making an agent incapable of performing certain tasks. Additionally, the goals, toward which such agents are working for, can themselves be subject to change. To cope in such complex situations, distributed and continual planning approaches have been proposed, that (i) distribute the planning process among a group of agents, (ii) allow for an incremental planning process, happening continuously during the operation of agents, and (iii) combined approaches for distributed continual planning (DCP) [3]. Multi-agent planning (MAP) has been defined as the problem of creating a plan for and by a group of agents [2]. Furthermore, five stages of MAP have been identified such as goal allocation to agents, refinement of goals into sub-tasks, sub-task scheduling by considering other constraints, communication of planning decisions, and plan execution.

Depending on the perspective, distributed planning can refer to either cooperative distributed planning (CDP), also known as cooperative and distributed multi-agent planning (MAP), or to negotiated distributed planning (NDP) [16]. A recent survey on cooperative and distributed MAP, also referred to as multi-agent coordination of actions in decentralized systems, provides a taxonomy of existing approaches in the literature based on how they deal with issues such as agent distribution, computational process, plan synthesis schemes, communication mechanisms, heuristic search, and privacy preservation [16]. In the MAP view, the goal is to create a global plan, whereas in the NDP the emphasis is on the agents' ability to fulfil their own local objectives. While a CDP focuses on issues of plan representation and generation, task allocation, communication and coordination, in the scope of an NDP, the focus is on collaboration and cooperation between agents. Continual planning on the other hand allows agents to revise their plans during operation as unforeseen events occur. Examples include (i) reactive planning systems, where an agent considers only the next step and does not look ahead further in the future, (ii) flexible plan execution systems, which allow for some look ahead, and (iii) delaying sketching out the detailed plan as much as possible.

Centralized planning implies that decisions are not made independently and locally, but rather at a global level. Utilizing centralized planning to solve multi-agent planning problems is a widely accepted approach. Landa-Torres *et al.* [9] used a centralized planner, based on evolutionary algorithms, to solve an underwater multi-agent mission planning problem for a swarm of autonomous underwater vehicles. Similarly, the solution to the problem of mission planning for a swarm of unmanned aerial vehicles was presented by Ramirez-Atencia *et al.* [14].

This paper describes a method to combine centralized and negotiated distributed planning approaches in order to optimize the execution of plans in a failure prone context (we consider total agent breakdown), simultaneously increasing the robustness of the system by allowing agents to perform a local plan reparation at runtime.

III. Problem Formulation

The problem that is being addressed in this paper is a relaxed version of the Extended Colored Traveling Salesperson Problem (ECTSP) [11], [12]. The original problem is simplified by the removal of the precedence constraints among tasks.

Assume a set of n tasks, $v \in \mathcal{V} := \{v_1, v_2, \ldots, v_n\}$, m agents, $s \in \mathcal{S} := \{s_1, s_2, \ldots, s_m\}$, and k capabilities, $c \in \mathcal{C} := \{c_1, c_2, \ldots, c_k\}$ where $m, n, k \in \mathbb{N}$. Each agent $s \in \mathcal{S}$ has a set of capabilities $\mathcal{C}_s \subseteq \mathcal{C}$ assigned to it. Each task $v \in \mathcal{V}$ requires one capability in order to be successfully completed. A capability matrix of an agent s, $\mathcal{A}_s \in \{0,1\}^{n \times n}$, defines which cities allow visits from a salesperson s, and is defined as $\mathcal{A}_s := [a_{ijs}]$, with

$$a_{ijs} = \begin{cases} 1, & f_c(v_i) \in \mathcal{C}_s \wedge f_c(v_j) \in \mathcal{C}_s \\ 0, & \text{otherwise,} \end{cases}$$

The problem consists in allocating n tasks to m agents with respect to given constraints in the form of agent capabilities and task requirements for such capabilities in order to minimize the make-span of a mission.

Objective function: The goal is to complete all the tasks in the environment while minimizing the mission's duration, even in the presence of one or more agent failures. In MASs, a mission can involve optimization of different parameters. Commonly, mission makespan is minimized, however, the duration of a mission can be defined in various ways [10]. The objective function used in this work aims to minimize the makespan, i.e., the duration between the starting time of the first task and end time of the last task, over all agents in the mission. This objective function is also known as "minMax", as it minimizes the maximum duration of an agent's make-span over all agents.

IV. Centralised Global Planner

In a real-world scenario, the process of mission planning starts with a human operator defining the mission parameters, e.g., tasks to be performed, goal of the mission, etc. The information about available robots – or more generally, vehicles – and the environment would then be fetched from the database. After this step, the mission is sent to the planner, which solves the mission and produces the necessary set of actions (plan) for mission execution. For simplicity, in this work, this step is skipped, as we have a set of predefined missions, thus in this section the focus will be solely on the process of plan creation.

Algorithms used to solve these kinds of problems are usually divided into two groups, exact and meta-heuristic. While exact algorithms can guarantee that the produced solution is optimal, meta-heuristics usually have no guarantees at all. However, meta-heuristic algorithms can produce a reasonably good solution within a short period of time. This is sometimes more important than having an optimal plan, especially in time-limited situations where re-planning might be necessary. Although the initial plan making is not bounded by time, the re-planning usually is. Re-planning can be seen as planning again with new initial conditions. Since multi-agent missions are usually costly and agents have limited energy available, the re-planning process should be very fast. For this reason, the algorithm behind the global planner is a Genetic Algorithm (GA), which is adapted for mission planning problems.

a) Chromosome encoding: Chromosomes are encoded as a set of arrays, where each array encodes an agent plan. A graphical representation of a single chromosome is given in Fig. 1. The size of each array is equal to the sum of the number of agents and tasks, i.e., $n + m$. The elements of the array are integer task IDs. The agent's route can be extracted from the encoding by following the task chain. For example, in Fig. 1 if we look at Agent 1, we can see that the first task in its plan is 5, and the next task ID is then stored in column 5, which is task 7. This continues until a destination depot is reached, that has the ID of $n + m$, which is 10 in this example.

b) Initial population.: The initial population is randomly created with respect to given constraints, hence, initial candidate solutions are in the feasible region of the search space.

c) Variation operators: The crossover operator used is a modified version of Edge Recombination Crossover (ERX) [17]. The first step is to select two parents for crossover from the mating pool. The mating pool is created based on the crossover probability and ranking, which is based on the individual's fitness. The second step is to create the adjacency matrix. In the matrix, the information of the neighboring tasks for each task in the plan is created based on the two selected parents. The information of tasks that the plan starts with and ends with is also kept. Finally, we randomly select a starting task, and the chain of selection continues by always randomly picking a task from a neighboring list of a previously allocated task. In the case all neighboring tasks are already allocated, we select a new task randomly. The whole process keeps in check that the required equipment of allocated tasks match the capabilities of the agent they have been allocated to.

We have two types of mutation operators. The first one is the jump mutation, where a task is randomly selected and moved to a new randomly selected location. The second one is a swap mutation where two tasks are randomly selected, and their location in the plan is swapped. Both jump and swap mutation are invoked 2 times. One time to perform the mutation within a single agent plan, which is also randomly selected. The other time to perform the same mutations, but with the mutation happening between different agents. In a way, we have both a local mutation operation and a global mutation operation.

d) Local refinement: There are two local refinement methods implemented. The first one is Greedy Search (GS) based on nearest neighbor heuristics, and the second one is based on 2-opt heuristics. While the GA performs allocation of cities to salespersons, the GS and 2-opt only reorder cities within the agent's plan. In other words, local refinement methods perform exploitation of the candidate solution, by reordering the list of cities, governed by nearest-neighbor or 2-opt heuristics. There are no guarantees that local refinement methods will be able to improve the candidate solution. That is why the changes in the plan that do not improve the overall fitness are being rejected.

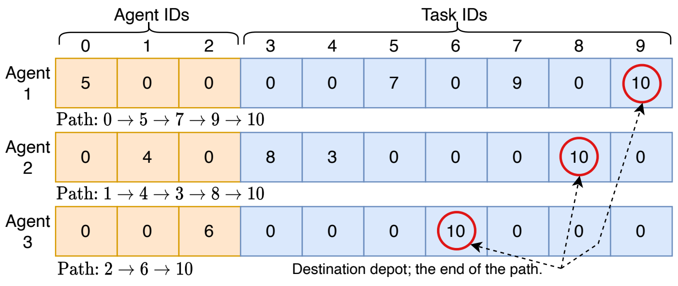
Agent IDs
Task IDs
0 1 2 3 4 5 6 7 8 9
Agent 1: 5 0 0 0 0 7 0 9 0 10
Path: 0 → 5 → 7 → 9 → 10
Agent 2: 0 4 0 8 3 0 0 0 10 0
Path: 1 → 4 → 3 → 8 → 10
Agent 3: 0 0 6 0 0 0 10 0 0 0
Path: 2 → 6 → 10
Destination depot; the end of the path.

Fig. 1: Graphical representation of chromosome encoding.

V. Decentralised Agent-based Planner

The agent design consists of three core aspects: (i) the architecture comprising of a finite state machine which captures the different behaviours of an agent, (ii) the willingness to interact abstraction and its role in shaping collaborative behaviour, and (iii) the interaction protocols used in any collaboration.

A. Architecture

The behaviour of each agent is captured by a finite state machine composed of four states, *idle*, *interact*, *execute*, *interact & execute* (Fig. 2). All agents start their operation in *idle*, where as the name suggests they are not committed to any task, but are rather waiting for a request to come. In principle, it is possible to make the agent engage in some activity, e.g., a random walk, while waiting in *idle*.

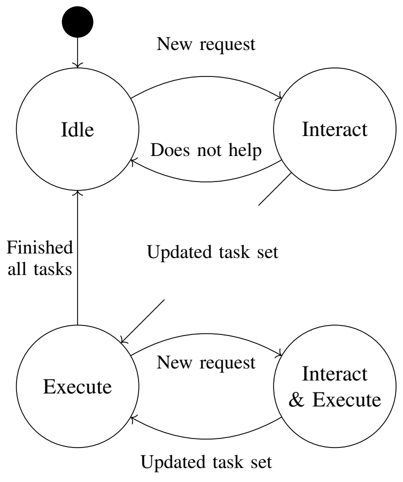

Fig. 2: Agent operation state machine.

Upon receiving a help request by another peer, an agent shifts to the interact state, where it evaluates the request. Such requests are of the form of "What is your willingness to perform task t_i?", and denote the start of a new negotiation round.

The request is broadcast to all agents within a given range, and all responses are collected. Should the agent receiving such requests not be impeded by the lack of equipment needed for performing t_i, it will respond with a positive willingness. Thereafter, if selected to perform the task, the agent will switch to *execute*, otherwise it will return to *idle*. At the end of each negotiation round, the task might be re-assigned to a new agent. If not, the agent handling the task, i.e., sending the help request, will attempt to re-assign it in future rounds.

The execution of any task is composed of two parts as follows: (i) reaching the location of the task, and (ii) its actual execution. A request to perform a task can also be received while the agent is already in *execute*. In this case, the agent will switch to *interact &execute*, and perform similar calculations as in *interact*. If selected to perform the new task, the agent will add said task to its list of tasks to execute.

Agents are also able to receive requests from other entities, such as a planner. In this paper, such requests override the existing task allocations that agents might have, thereafter providing new assignments. It would be possible in principle for agents to evaluate such requests in a similar way as they do requests from one another, as well as consider aspects such as trust. However, these are considered out-of-scope for the purpose of this paper.

B. Willingness to Interact

The collaborative behaviour of an agent a_i is determined by its willingness to interact $w_i(t) \in [-1, 1]$, i.e., the likelihood of asking and giving help to other agents at time t. A positive willingness indicates that a_i is able to help others $w_i(t) > 0$, whilst a negative willingness indicates that a_i needs help performing its tasks $w_i(t) < 0$. Where $w_i(t) = -1$ indicates that an agent must ask for help at time t, and $w_i(t) = 0$ denotes a neutral disposition. The willingness is affected by both the state of an agent, which captures the general attitude towards potential collaboration with others [5], and by the properties of the specific task considered during any negotiation, namely the utility of performing such task.

In this paper, the focus is on how the utility of performing a particular task τ_j affects the willingness to interact ($w_i(t) = 0$). Two factors are considered, the equipment required by τ_j and the distance d to the task. The case, in which agent a_i does not have the necessary equipment required by task τ_j, will reflect in a negative willingness to interact, specifically $w_i(t) = -1$. It is possible to distinguish two circumstances in which an agent a_i considers the allocation of τ_j, (i) a_i has no previous allocation, and (ii) a_i is already allocated to a set of tasks. In case (i), d is the distance between the agent's location and τ_j's location, with utility calculated as $u_{\tau_j}(t) = 1/d$. In case (ii), d is the minimum distance to τ_j considering the a_i's location, and the location of the other tasks allocated to a_i, given by $d = \min(\{d_{kj}, \forall k \in L\})$, where L is the set containing the locations of agent a_i and its tasks, and d_{kj} is the distance between the k^{th} element in L and task τ_j. The final value of the willingness to interact with respect to task τ_j is expressed by

$$w_{i\tau_j}(t) = w_i(t) + u_{\tau_j}(t). \quad (1)$$

Although the willingness to interact is itself an expression of utility, in this paper its notion and that of task utility are separated. This is done in order to have a clear distinction between what affects the general disposition to collaborate, and what affects the utility of a performing a single task. Note that, a negative willingness where $w_i(t) \in]-1, 0[$, can be used to express the confidence an agent has to completing a task τ_j, where the more negative the willingness, the less confidence in succeeding. When $w_i(t) = -1$ then the agent is absolutely sure it will fail, e.g., the required equipment is not functional any longer or is not present (the case studied in this paper). These aspects of negative willingness where $w_i(t) \in]-1, 0[$ are not explored in this paper, as aforementioned, it is assumed that $w_i(t) = 0$, thus agents are neutrally disposed to collaboration.

C. Interaction Protocols

Several assumptions hold concerning the interaction between agents. Firstly, no two agents can start the negotiation for a unique task at the same time. Secondly, agents can have the knowledge of each other's allocations, and the tasks that are completed. Thirdly, this knowledge is not necessarily available for every time-step, and can become available to an agent after a delay. As a result of the last two assumptions, it can happen that a task is repeated more than once.

The interaction protocol defines how agents negotiate with one another over the assignment of tasks which are to be completed (Fig. 3). Requests for help can be initiated by any agent in the event of the detection of a full failure of an agent[a]. The first agent to detect the failure of another agent initiates a negotiation with others to re-allocate the failed agent's tasks.

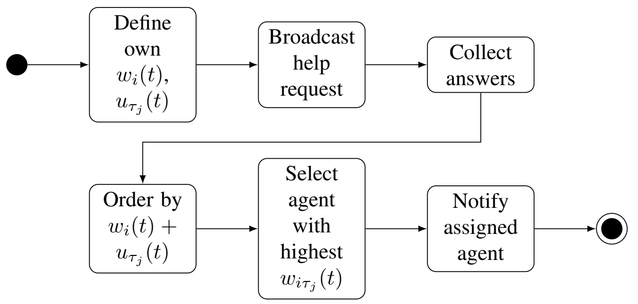
Define own $w_i(t)$, $u_{\tau_j}(t)$
Broadcast help request
Collect answers
Order by $w_i(t) + u_{\tau_j}(t)$
Select agent with highest $w_{i\tau_j}(t)$
Notify assigned agent

Fig. 3: Interaction Protocol.

For each task to be assigned, a request for help is broadcast. Afterwards, the responses of other agents, consisting of the respective willingness and utility values, are collected. Note that, replies from agents with negative willingness are ignored. Thereafter, the rest of the responses are ordered based on the combined value of willingness and utility, and the agent with the highest willingness will be allocated to the task.

[a]Such detection mechanisms are outside the scope of this paper.

VI. Simulation Design

In this section, we describe in detail the implementation of the MAS, as well as that of the planner. Given that these two are not contained within the same simulation environment, we also describe how their interaction is set up considering three strategies to tackle agent failures, namely planner-only, agent-only, and hybrid, focusing as well on time synchronisation.

A. MAS Implementation

The implementation of the MAS is done in the Gama platform [15] [b]. Gama is a simulation development environment, that allows users to quickly implement agents through a high-level agent-based language, as well as run large scale simulations of said agents. Additionally, in Gama the spatial component of an agent is part of the design from the get go, with the platform offering many macros for simulating the motion of agents, e.g., `goto(Point(x,y))`. It is possible to specify for a Gama agent a set of reflexes and actions. Reflexes are behaviours that get executed each tick of the clock (cycle), whereas actions are functions that can be called to perform some calculation at specific times. The communication between agents is also taken care by the platform. It is possible to broadcast messages to all agents by using `ask agentSpeciesName`, as well as to a group of agents within a defined range that can be returned using `agentSpeciesName at_distance(range)`. Additionally, it is possible to randomly select an agent from a group using `ask one_of(agentSpeciesName)`.

At the start of the simulation agents receive a plan (details to follow in the next subsection) that contains a list of allocations, mapping agents to tasks. This means that each agent gets a list of tasks it has to complete, but has as well knowledge about tasks assigned to other agents. In order to complete a task τ_j, and agent a_i is required to have the equipment needed for completing τ_j. More formally we can characterise an agent a_i by the equipment it holds, either one or two out of three given types, say A, B, C. Equipment is overlapping, i.e., more than one agent has the equipment of any type. Whereas, a task τ_j is defined by its location given as (x_{τ_j}, y_{τ_j}), and the equipment necessary (one of three given types) for its completion.

At every cycle, agent a_i will check if there is a task in its task set. If so, a_i checks whether it is on location, i.e., if within $0.5m$ distance to task τ_j. If on location, a_i proceeds with the execution of the task, simulated in one cycle. Otherwise, a_i moves towards the task using `goto(Point(`x_{τ_j}, y_{τ_j}`))`. When τ_j is complete, a_i removes it from the list, and continues in the same manner with the next task. Upon the completion of all tasks in the set, a_i proceeds towards a fixed depot point, which is the same for all agents (located at $(0, 0)$).

In every cycle, all agents engage in a gossip round, where they exchange relevant information with one agent picked randomly of the ones found within a defined range R. Such information includes: completed tasks, discovered failed agents, and pending tasks which have to still be assigned. When all tasks have been completed, and perceived as such by at least one operating agent, then the simulation ends.

[b]The code and instructions to reproduce the results in Section VII are publicly available at https://github.com/gitting-around/glocal

B. Planner Settings

The planner is implemented in C++ programming language, as a single threaded application. The algorithm behind the planner is described in Section IV. The algorithm parameters used in this paper are as follows. The population size is set to 200, mutation rate is 15%, while the crossover rate is at 70%. Only 5% of the population is kept and copied to the next generation. The stopping criterion of the algorithm is set to be 1000 generations.

C. Interaction between Planner and MAS

At the start of a simulation, agents receive a plan from the planner, irrespective of which strategy is adopted. Specifically, a server agent is setup able to communicate with entities external to Gama via TCP. The server does not perform any tasks, its only job is to serve as a bridge between the Gama MAS and planner. The cycle corresponding to when this first plan is received marks the beginning of the mission, and is used later on to determine the duration of the mission execution. Upon receiving the first plan, the server asks all agents to initialise their state (1 cycle), consisting in the initial location, equipment, and assigned tasks. Already in the next cycle, agents proceed with executing the assigned tasks. Every simulation cycle during execution is run faster than real-time, as fast as it can run in Gama. Assuming no agent failures, this process will continue smoothly until all tasks are completed.

In order to test the three strategies, we inject failures in the simulation by failing $\{1...n\}$ agents at random during the execution of the mission. Note that, should there still be tasks requiring equipment of some type X, but no agents left that has X, then the mission will become infeasible, thus remain incomplete, and the simulation will end. Failures are injected in the system at specific times, calculated based on the progress made with completed tasks. In this paper we define the progress based on the number of failures to be injected (f) and the number of tasks (τ), specifically a fail will occur every time the MAS has progressed by $\lfloor \tau/(f+2) \rfloor$ tasks. This choice is made such that there is enough space between failures to inject all desired failures while there are still tasks to be completed. When an agent a_i fails, the next step depends on the adopted strategy, each of which is described below.

a) Planner approach: Assume agent a_i fails at cycle t_k. The operation of the MAS will proceed as usual until a_i is detected as failed – here we assume full failures, where the robot is out. a_i can be detected as out of order during a gossip round, should it be picked by some other agent a_d in order to exchange information. It is assumed that the agent requesting the gossip is able to make this detection. As soon as a_i's failure is detected by a_d, the latter will initiate a call to the planner and request a re-plan. In such a request, a_d includes the list of tasks it knows to be complete (his own, and from others), as well as the list of active agents. Note that, if a_i has completed some of its tasks before failing, but was not able to spread this information through gossip, then those tasks will be re-assigned by the planner, and as such their execution will be duplicated. Additionally, the list of completed tasks depends on a_d's knowledge on what other agents have achieved by t_k. When a call to the planner is made, the execution of a simulation cycle is slowed down to real-time, i.e., 1 cycle takes 1 second to execute, as the agents are waiting for a new plan. During this time, all agents are in wait mode, and clear up their task lists. This process will continue until there are no more failures in the MAS, and the mission can either become infeasible, or complete. It might happen that the failure of a_i is never discovered, as other agents fail to go near it during their operation. To tackle these instances, a watchdog timer becomes active for every agent, once they complete all their tasks and move to the depot. When the timer expires, the first agent (the execution in Gama is set to sequential) that evaluates it will try to reassign all the tasks for which the status is unknown.

b) Agent approach: Assume that agent a_i fails at cycle t_k. As in the planner case the operation of the MAS will proceed as usual until a_i is detected as failed. When an agent a_d detects the failure of a_i then a_d starts a negotiation round with all agents within range R for each of a_i's tasks separately. Note that as before, if a_i has completed tasks, but failed to spread this information, or this information never reached a_d then the latter will include these tasks in the negotiation. All (not failed) agents in R will respond with their willingness to perform a_i's tasks. When all responses for task τ_j are collected, a_d will select the agent with the highest willingness and assign τ_j. Note that if the assigned agent knows τ_j to be complete, then it will drop the task, otherwise it will add it to its task list in the position with the highest utility. It can happen that no agent in R (including a_d itself) is able to perform τ_j. Should this happen, a_d places τ_j in a separate list with tasks to be dealt with after it has completed all its other tasks, and started moving toward the depot. As mentioned, the information on such tasks is shared during the gossip, in the eventuality that a_d fails itself. This ensures that a_d will eventually be able to assign an agent to τ_j, if there is one still with the necessary equipment. In case the failures are not discovered, the watchdog will ensure that the tasks with unknown status are assigned.

c) Hybrid approach: Assume that agent a_i fails at cycle t_k. Similarly to the other two cases, the operation of the MAS will proceed as usual until a_i is detected as failed. The hybrid strategy is similar to the agent approach up to the point whereupon detection of the a_i's failure, agent a_d will attempt to reallocate its tasks to those within R. Should this reallocation fail on at least one task, then a_d will initiate a call to the planner and request a global re-plan. If the reallocation succeeds for all tasks, then the MAS proceeds as usual. Note that for infinite communication range, the hybrid and agent approaches will produce the exact same results. In case a_i's failure is never discovered, then the watchdog will trigger a round of negotiation. If at least one task fails to be reallocated, then a new plan is requested from the planner.

VII. RESULTS

The evaluation of the three strategies, planner-only, agent-only, and hybrid, is performed by running simulations with Gama under different conditions with a population of 10 agents. In order to visualize the results, we plot 3D graphs for each metric over the number of fails and the task set sizes (Fig. 4). These metrics include: the size of the task set ψ, and the number of failures ϕ. Tasks are distributed over a 2D space of size $200 \times 200[m]$. In this paper, we run experiments over all combinations of values for the aforementioned parameters, where each takes values as following: $\psi \in \{50, 100, 150\}$, $\phi \in \{0, 1, 2, 3, 4, 5, 6, 7\}$, all with $r = 100[m]$ [c]. Each distinct configuration is repeated 30 times, with a different seed value generated using the random number generator function in Gama. Note that the planner and Gama simulation use the same seed per run. For each seed value, and for each failure case, we generate beforehand the list of agents to fail. This is to ensure that across the different approaches, the same agents fail, and should this lead to an infeasible mission, it will be the case for all three approaches. However, this means that in some cases during a re-plan, the agents that are planned to fail do not get any tasks allocated to them. If so, these agents will still fail, but there will be no impact on the simulation as a whole, i.e., there will be no need to re-plan since there are no tasks that need to be re-allocated. The watchdog trigger is equal to the longest execution over all agents, and is recalculated each time tasks are re-allocated (by the planner or through agent negotiation). Each simulation has a time limit, after which the mission is counted as failed. For every simulation, we use the recorded logs to calculate the following metrics: mission duration, number of exchanged messages between agents (counting both gossip and negotiation), relevant in cases of limited bandwidth, and number of requests to the planner.

Fig. 4a describes the mission duration in different mission conditions and settings. It can be noticed that for the case of 0 fails, all three approaches, for all three problem sizes, have the same mission duration. The reason behind this is that when there are no fails, the agents simply execute the initial plan provided from the centralized planner, thus there are no differences between the approaches. This changes as soon as agents fails are introduced, giving advantage to the planner approach when the number of fails is low (1–2 fails). In this range, the planner approach is able to outperform the other approaches, as the quality of the produced plan outweighs the time lost producing it. As the number of the fails increases, it can be observed that the hybrid approach starts performing the best. The planner approach performance keeps degrading until it becomes the worst approach in the scenario with 7 fails for problem sizes of 100 and 150 tasks, as the quality of the solution does not provide any benefits due to the time spent on re-planning. The agent approach suffers less from the increased number of fails, and that ultimately leads to

[c]Due to the page limit, the results we show in this paper correspond to one value of r. However, at https://gitting-around.github.io/glocal/ results for $r \in \{20, 50, \infty\}$ are reported, as well as an extensive list of metrics.

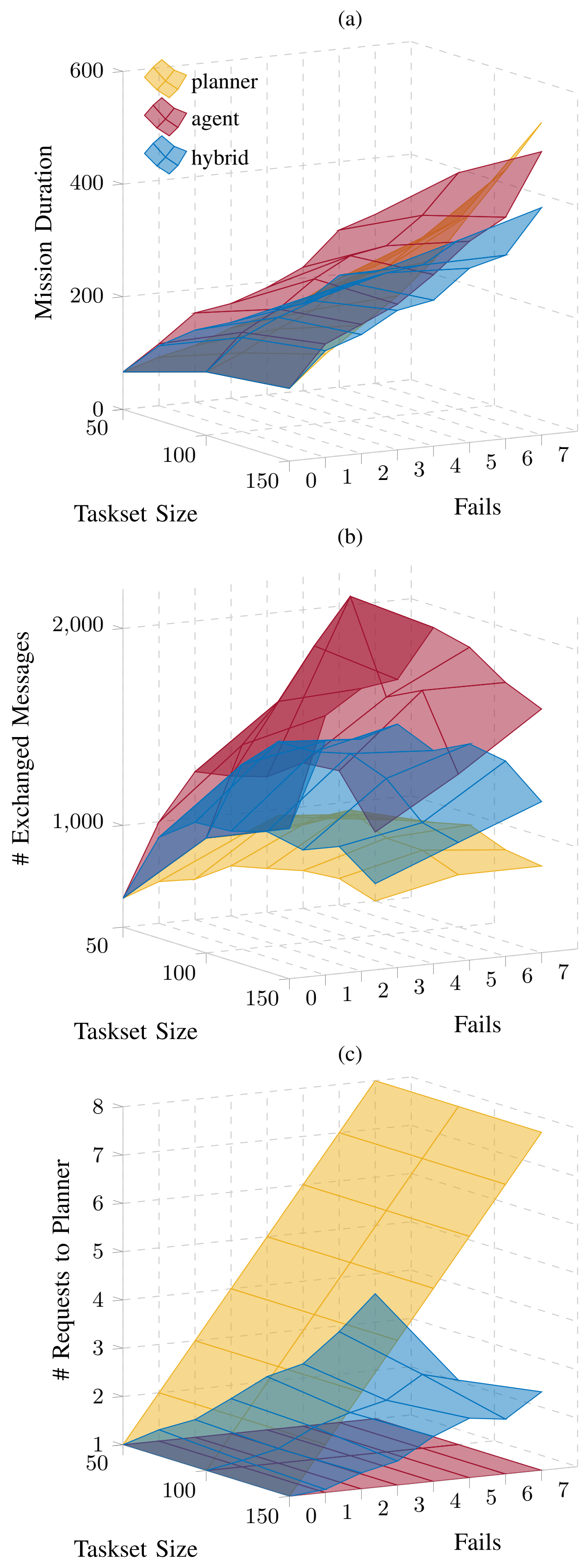

Fig. 4: Results for (a) mission duration, (b) average number of messages exchanged, and (c) average number of requests to the planner, over number of failures and task set sizes.

outperforming the planner approach in the case of 7 fails. The decentralized re-planning is not as computationally heavy as centralized re-planning, and that is the main reason for the agent approach performing better as the number of tasks and fails increases. The success of the hybrid approach can be better understood if we observe Fig. 4c. It can be noted that the number of requests to the planner in the hybrid approach is between 1 and 3.5 on average. That corresponds to the good performance of the planner approach in the cases of 1 and 2 fails shown in Fig. 4a. The hybrid approach is able to keep the number of re-plans relatively low, while not sacrificing too much of the overall solution quality. This is particularly apparent in the most complex cases with numerous fails, where the proposed hybrid approach outperforms the other two approaches, and the gap is widening with the increase in complexity.

In terms of messages exchanged (Fig. 4b), the planner-only approach is—as expected—more efficient, bearing only the cost of the gossip, and no negotiation overhead, and remains relatively low among all failures cases. The agent approach is consistently the worst, with the exception of 0–1 failure cases, where it is quite close to the hybrid approach. For a higher number of failures, the hybrid approach provides a middle-ground between the planner-only and agent-only. Additionally, we can observe that for 6–7 failures, there are dips in the number of messages exchanged even for the agent-only approach. We reason that, since there are fewer agents in the system taking part in the negotiation (and gossip), then it is normal to observe such decline, which remains consistently higher as compared to the other two approaches. Finally, with respect to the number of requests to the planner (Fig. 4c), the hybrid approach as expected results in lower values than the planner approach, given that plan reparation is performed locally. Comparing the number of exchanged messages we can observe that, for smaller task sets, there are more calls to the planner, which corresponds to fewer messages exchanged due to negotiation (hybrid approach), whereas for larger task sets the opposite holds. It is important to note that, the communication range has a non-negligible impact on these metrics. Low communication range would lead to less successful negotiation rounds, and therefore to more requests to the planner (in the hybrid approach).

VIII. Related Work

A recent survey on robotic systems in the domain of logistics [1], investigates the problem of task planning, and provides a historical overview, from the early days of PDDL [7], to current types of planners, from those employing exact methods to find optimal solutions, to those using heuristics and meta-heuristics, providing quickly good enough solutions but with no guarantees of optimality. The combination of such planners with mechanisms deployed at the agent level for plan reparation, are, to the best of our knowledge, not widely discussed in the literature. The need of combining what can be considered static approaches, i.e., approaches that compute solutions offline, with dynamic ones is relevant in other aspects of MASs/MRSs, e.g., for coordination and decision-making, bringing forth a need to explore hybrid mechanisms [4], [18]. Mohalik *et al.* [13] propose an approach for online plan repair in hierarchical MAS, while assuming only small local failures that do not affect the global situation, and allow managing agents to trigger re-planning for their underlings, thus adapting fragments of the plan. In our approach, we do not consider hierarchy, in fact any agent can try to fix the initial plan, only turning to the planner should the attempt fail.

Yang *et al.* [19] proposed the AutoRobot framework built on top of the JADE platform and ROS. They combine adaptive and reactive control strategies to create a hybrid architecture. The planner, which is in the adaptive layer of control, creates an initial plan that the robots execute. In case there is a deviation, re-planning is triggered. The reactive layer monitors and informs the planner of an impending danger. In this case, the planner picks one of the pre-planned strategies depending on the system state and executes it. That strategy has the highest priority until the danger is averted. The downside of this approach is that in case of a communication problem, the robot has no way of reacting. Our approach does not depend on pre-planned strategies to deploy, rather, agents attempt to fix the issue locally before requesting a new plan.

Hrabia *et al.* [8] proposed a ROS Hybrid Behaviour Planner that combines the advantages of a reactive and adaptive control with a symbolic planner. Their results show that the combination of a behaviour network with a centralized planner leads to more efficient solutions. In contrast to our work, the uncertainty comes from obstacles and not robot failures.

IX. Conclusion

In order to mitigate the effects of possible failures in Multi-Agent Systems, which may lead to prolonged mission execution time or in some cases complete mission failure, we have proposed a hybrid approach called GLocal. The common 2-approach (centralized vs. decentralized) paradigm to dealing with the planning in MASs does not always provide acceptable results. The centralized approach is computationally expensive and does not offer much flexibility, on the other hand, the decentralized approach offers more flexibility but at the cost of solution quality and communication bandwidth.

The proposed GLocal approach aims at exploiting the benefits of the two approaches while trying to minimize the drawbacks. This is done by incorporating both centralized and decentralized approaches into one framework, where in the case of system failure the system decides which action to take, i.e., should global re-planning be done, or the problem can be resolved locally. In addition to presenting the proposed architecture of the GLocal framework, we have devised a set of experiments to confirm the usability of the proposed approach. The results show that for a small number of the system fails, the centralized planner outperforms the other approaches. However, as the number of fails increases, GLocal is the top performer in terms of overall mission duration. Considering the number of messages that have to be exchanged in the system, GLocal sits in between decentralized and centralized approach. Meaning that the increased performance comes with a certain drawback of increased communication within the system.

In future work, we will investigate approaches able to further exploit the trade-off between local and global approaches. Specifically, we will study agents with learning capabilities and on-board decision mechanisms, able to choose whether to rely on local or global coordination.

References

[1] R. Bernardo, J. M. Sousa, and P. J. Gonçalves. Survey on robotic systems for internal logistics. *Journal of Manufacturing Systems*, 65:339–350, 2022.

[2] M. De Weerdt and B. Clement. Introduction to planning in multiagent systems. *Multiagent and Grid Systems*, 5(4):345–355, 2009.

[3] M. E. desJardins, E. H. Durfee, C. L. Ortiz Jr, and M. J. Wolverton. A survey of research in distributed, continual planning. *AI magazine*, 20(4):13–13, 1999.

[4] L. Esterle and D. W. King. Loosening control—a hybrid approach to controlling heterogeneous swarms. *ACM Trans. Auton. Adapt. Syst.*, 16(2), 2022.

[5] M. Frasheri, B. Cürüklü, M. Esktröm, and A. V. Papadopoulos. Adaptive autonomy in a search and rescue scenario. In *Proc. of the Int. Conf. on Self-Adaptive and Self-Organizing Systems*, pages 150–155. IEEE, 2018.

[6] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: theory and practice*. 2004.

[7] A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language. *Technical Report*, 1998.

[8] C.-E. Hrabia, S. Wypler, and S. Albayrak. Towards goal-driven behaviour control of multi-robot systems. In *Proc. of the Int. Conf. on Control, Automation and Robotics*, pages 166–173. IEEE, 2017.

[9] I. Landa-Torres, D. Manjarres, S. Bilbao, and J. Del Ser. Underwater robot task planning using multi-objective meta-heuristics. *Sensors*, 17(4):762, 2017.

[10] A. Maoudj, B. Bouzouia, A. Hentout, and R. Toumi. Multi-agent approach for task allocation and scheduling in cooperative heterogeneous multi-robot team: Simulation results. In *IEEE Int. Conf. on Industrial Informatics (INDIN)*, pages 179–184, 2015.

[11] B. Miloradović, B. Çürüklü, M. Ekström, and A. V. Papadopoulos. A genetic algorithm approach to multi-agent mission planning problems. In *Proc. of the Int. Conf. on Operations Research and Enterprise Systems*, pages 109–134. Springer, 2019.

[12] B. Miloradović, B. Çürüklü, M. Ekström, and A. V. Papadopoulos. GMP: A genetic mission planner for heterogeneous multirobot system applications. *IEEE Trans. on Cybernetics*, pages 1–12, 2021.

[13] S. K. Mohalik, M. B. Jayaraman, R. Badrinath, and A. V. Feljan. Hipr: An architecture for iterative plan repair in hierarchical multi-agent systems. *J. Comput.*, 13(3):351–359, 2018.

[14] C. Ramirez-Atencia, G. Bello-Orgaz, M. D. R-Moreno, and D. Camacho. Solving complex multi-UAV mission planning problems using multi-objective genetic algorithms. *Soft Computing*, 21(17):4883–4900, 2017.

[15] P. Taillandier, B. Gaudou, A. Grignard, Q.-N. Huynh, N. Marilleau, P. Caillou, D. Philippon, and A. Drogoul. Building, composing and experimenting complex spatial models with the gama platform. *GeoInformatica*, 23(2):299–322, 2019.

[16] A. Torreno, E. Onaindia, A. Komenda, and M. Štolba. Cooperative multi-agent planning: A survey. *ACM Computing Surveys*, 50(6):1–32, 2017.

[17] L. D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesmen: The genetic edge recombination operator. In *ICGA*, volume 89, pages 133–40, 1989.

[18] Z. Yan, N. Jouandeau, and A. A. Cherif. A survey and analysis of multi-robot coordination. *IJARS*, 10(12):399, 2013.

[19] S. Yang, X. Mao, S. Yang, and Z. Liu. Towards a hybrid software architecture and multi-agent approach for autonomous robot software. *IJARS*, 14(4):1729881417716088, 2017.