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

Mirgita Frasheri, Branko Miloradović, Baran Çürüklü, Mikael Ekström, Alessandro V. Papadopoulos
Mälardalen University, Västerås, Sweden
{mirgita.frasheri,branko.miloradovic,baran.curuklu,mikael.ekstrom,alessandro.papadopoulos}@mdh.se

***Abstract*—Multi-robot systems can be prone to failures during plan execution, depending on the harshness of the environment they are deployed in. As a consequence, initially devised plans may no longer be feasible, and a re-planning process needs to take place to re-allocate any pending tasks. Two main approaches emerge as possible solutions, a global re-planning technique using a centralized planner that will redo the task allocation with the updated world state information, or a decentralized approach that will focus on the local plan reparation, i.e., the re-allocation of those tasks initially assigned to the failed robots. The former approach produces an overall better solution, while the latter is less computationally expensive. The goal of this paper is to exploit the benefits of both approaches, while minimizing their drawbacks. To this end, we propose a hybrid approach that combines a centralized planner with decentralized multi-agent planning. In case of an agent failure, the local plan reparation algorithm tries to repair the plan through agent negotiation. If it fails to re-allocate all of the pending tasks, the global re-planning algorithm is invoked, which re-allocates all unfinished tasks from all agents. The hybrid approach was compared to planner approach, and it was shown that it improves on the makespan of a mission in presence of different numbers of failures, as a consequence of the local plan reparation algorithm.**

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

I. Introduction

Automated planning is the process of defining a set of actions for an autonomous system to achieve a prescribed set of goals. The problem of automated planning has been a central research topic in artificial intelligence for the last decades [6], and it has become particularly relevant in the context of Multi-Agent Systems (MASs) [20].

Several approaches have been proposed to address the automated planning problem, and they are divided into two major categories: centralized and distributed planning. While there are clear advantages of centralized algorithms related to optimality of the computed plan with respect to an objective function, the curse of dimensionality has been the main limitation of such approaches. On the other hand, distributed algorithms provide a more robust alternative towards faults [7], [19], safety [1], [18], and security [8], [10].

This work was supported by the UNICORN project, the Aggregate Farming in the Cloud (AFarCloud) European project, with project number 783221 (Call: H2020-ECSEL-2017-2), the DPAC research profile funded by KKS (20150022), and the FIESTA project funded by KKS. Projects are supported by ECSEL JU and the VINNOVA.

The trade-off between the optimality of centralized solutions, and the flexibility and robustness to potential failures of distributed approaches is the main focus of this paper. In addition, this trade-off is studied in the context of agent or robot failures [1]. When a failure occurs, the centralized approach will compute a new plan based on the current conditions, referred to as re-planning, whereas the decentralized approach will rely on self-organization, which allows agents to perform a local plan reparation.

In this paper, a novel hybrid approach for multi-agent automated planning, GLocal, is proposed. GLocal exploits the advantages of both approaches, i.e., optimality and robustness, while limiting their inherent disadvantages. In particular, this paper investigates what is the effect of failures in MAS automated planning, and it shows the robustness of the proposed hybrid approach. When a failure occurs, agents in the MAS attempt to repair the plan locally, by negotiating with one another over the assignment of the pending tasks, i.e., tasks initially assigned to the failed agent. Agent collaboration is shaped by their willingness to interact, which captures the utility of being assigned to a given task. In case at least one task remains un-allocated, agents make a request to the centralized global planner for a re-plan, and as a result they switch from a local to a global strategy. More specifically, this paper focuses on the following research questions (RQs)

RQ1 How does the number of replans from a centralized planner impact on the overhead and quality of the solution of a MAS?

RQ2 How can the agents minimize the number of calls to the planner by collaborating with one another?

These questions are addressed through computer simulations, where the GLocal approach was compared to a planner-only approach, for different number of failures, as well as sizes of problem instances. The results reveal that GLocal produces solutions with shorter mission make-spans in the presence of failures, as a consequence of reduced calls to the centralized global planner.

The rest of the paper is organized as follows. Section II describes the background of the paper, and the problem addressed in this paper is presented in Section III. Section IV describes the design of the MAS that controls the behavior of the robots, while Section V presents the centralized global planner for the agents. Section VI describes the simulation

[1] In this paper the terms agent and robot will be used interchangeably.

setup, and Section VII presents the experimental results. Finally, Section VIII discusses the related work, and Section IX concludes the paper.

II. Background

In real world applications, the operation of agents or robots can be disrupted by environments 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. In order 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, and (ii) allow for planning to be an incremental process that happens continuously during the operation of agents, as well as (iii) combined approaches for distributed continual planning (DCP) [3]. Multi-agent planning 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) [2], or to negotiated distributed planning [21]. In the former view, the goal is to create a global plan, whereas in the latter the emphasis is on the agents ability to fulfil their own local objectives. While a CDP focuses on issues as 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 reactive planning systems, where an agent considers only the next step and does not look ahead further in the future; and flexible plan execution systems, which allow for some look ahead, and delay sketching out the detailed plan as much as possible.

Centralized planning implies that decisions are not made independently and locally, but rather holistically at a global level. Utilizing centralized planning to solve multi-agent planning problems is a widely accepted approach. Landa-Torres *et al.* [11] used a centralized planner, based on the 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.* [17]. The problem is modeled as a constraint satisfaction problem and solved using a multi-objective Genetic Algorithm (GA). A different approach to a similar problem is taken by Karaman Sertac *et al.* [9] where process algebra is used to model the problem that is later solved with the GA.

This paper is concerned with the investigation of methods that combine centralized and negotiated distributed planning approaches in order to optimize the execution of plans in a failure prone context, simultaneously increasing the robustness of the system by allowing agents to perform a local plan reparation online.

[2]A recent survey on cooperative and distributed MAP, referred to also as multi-agent coordination of actions in decentralized systems, provides a taxonomy of existing approaches in the literature based how they deal with issues such as agent distribution, computational process, plan synthesis schemes, communication mechanisms, heuristic search, and privacy preservation [20].

III. Problem Formulation

The problem that is being addressed in this paper is a relaxed version of the Extended Colored Traveling Salesperson Problem (ECTSP) [14]. 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, $\mathcal{A}_s \in \{0,1\}^{n \times n}$, can be defined as:

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

Agents are allowed to move in a 2D space Z, that is represented as a continuous map, and are able to communicate with one another with broadcast, without limitation in range.

The problem consists of 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 mission's duration, even in presence of one or more agent failures. Agent failures may be due to a physical failure of the robot performing a specific task, or of the equipment that is required to perform said task.

In MASs, a mission can involve optimization of many different parameters. Commonly, mission duration is minimized, however, a duration of a mission can be defined in various ways [13]. The objective function used in this work tends to minimize the makespan, i.e., 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 makespan over all agents.

IV. Agent Design

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.

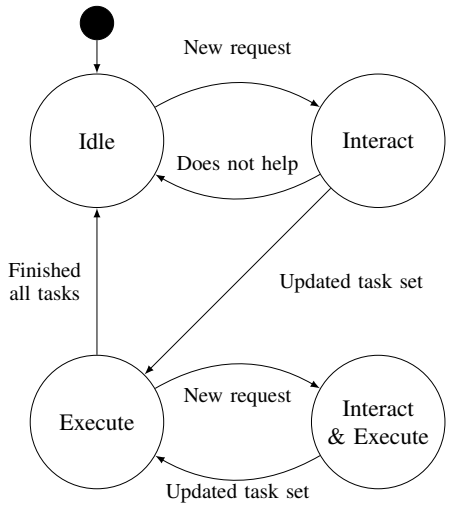

Fig. 1: Agent operation state machine.

A. Agent Architecture

Each agent is designed as a finite state machine composed of four states, *idle*, *interact*, *execute*, *interact & execute* (Fig. 1), and starts its operation in *idle*. Agents in *idle* are not committed to any task. Note that, depending on the application, other types of behaviours could be implemented in this state. Agents are able to create tasks on their own, e.g., reaching an object location after detecting its presence. Additionally, they are able to get a list of tasks by another entity, e.g., a centralized planner; this is the case studied in this paper. When an agent commits to performing a task, it switches to *execute* and proceeds with the task execution. Once an agent completes all its tasks, it reverts back to the *idle* state. As soon as a new task is detected or received, an agent, residing either in *idle* or *execute*, switches to *interact*, or *interact & execute*, respectively. Thereafter, a negotiation process is initiated with other agents in order to allocate every new task. The outcome of the negotiation is zero or more agents assigned to perform each task. An agent with no assignment at the end of the negotiation goes back to either *idle* or *execute*, respectively, i.e., the state it was in before the negotiation round.

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 $w_i(t) < 0$, with $w_i(t) = -1$ indicating 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 (explored in previous work [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 solely 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. 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:

$$u_{\tau_j}(t) = 1/d. \tag{2}$$

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:

$$d = \min(\{d_{kj}, \forall k \in L\}), \tag{3}$$

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

$$w_{i\tau_j}(t) = w_i(t) + u_{\tau_j}(t). \tag{4}$$

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.

C. Interaction Protocol

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, as well as the tasks that are completed. Thirdly, this knowledge is not necessarily available for every time-step, and can come to the knowledge of an agent with a certain delay. As a consequence 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. 2). Requests for help can be initiated by any agent in the event of the detection of a full failure of an agent [3]. The first agent to detect the failure of another agent initiates a negotiation with others to re-allocate the tasks of the failed agent. 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

[3]Such detection mechanisms are outside the scope of this paper. Further details on the implementation are given in Section VI.

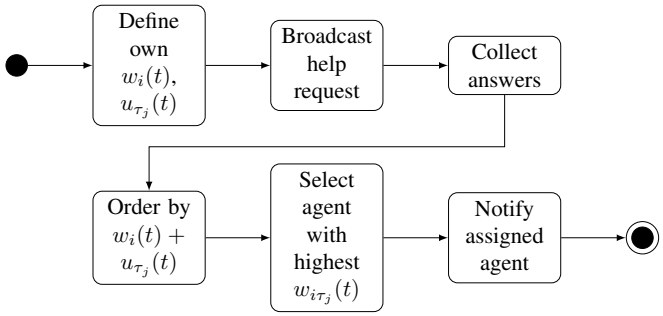

Fig. 2: Interaction Protocol.

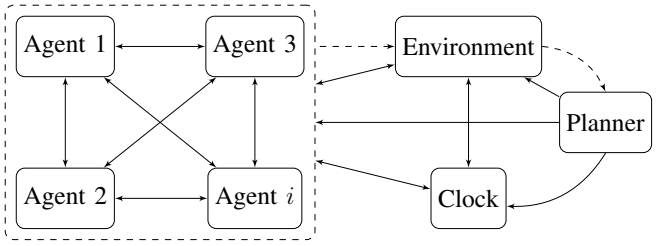

Fig. 3: Simulation components.

are ordered based on the combined value of willingness and utility, and the agent with the highest willingness will be allocated to the task.

V. Centralized Global Planner

The process of mission planning first starts with the creation of a mission. This is done by a human operator who defines the mission parameters (tasks to be done, available vehicles and the overall goal of the mission) in the Mission Management Tool (MMT) [14]. 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.

Algorithms used to solve this kind 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 situations where re-planning might be necessary. Although the initial plan making is not bounded by time, the re-planning is. Re-planning, in this case, can be seen as planning again with new initial conditions. Since multi-agent missions are usually costly and autonomy of agents is limited as well, the re-planning process should be very fast. For that reason, the algorithm behind the global planner, in this paper, is a Genetic Algorithm (GA), which is adapted to planning and scheduling problems. Chromosomes are encoded in the same way as in [14], thus two arrays of integers, representing the genes, are used. The first array consists of integers representing tasks and agents, whereas the second array represents task parameters (equipment requirements, task duration, and location). Chromosome length varies from $n + 1$ to a maximum of $n + m$ genes, depending on the number of agents used in a mission.

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

The crossover operator has not been used since it did not have positive effects on the convergence process. Mutation is the only source of variability as it allows genetic diversity in the population. Every individual has a low probability to be selected for mutation. In this paper, two types of mutation schemes are introduced. One operates on the task genes through swapping tasks and inserting new genes, whereas the other mutates agent genes through growing (adding agents) and shrinking (removing agents) from the chromosome.

A task swap mutation swaps two task genes in a chromosome, meaning that it can both swap tasks within a single agent or between two agents. An insert mutation chooses a task and inserts it in a new location in a chromosome, similarly to the previously explained mutation, the insertion can be within the same agent or different one.

An agent shrink mutation removes one agent from a chromosome, reallocating its tasks to other agents. Growth agent mutation adds a new agent to the plan. The new agent gene is randomly inserted, acquiring tasks from that location in the chromosome up to the next agent gene or end of the chromosome. If there are conflicting (a task not supported by assigned agent) tasks, they are randomly reallocated to other agents. Both algorithms take into account given constraints ensuring that the mutation process does not produce infeasible solutions.

VI. Simulation Design

The combined approach consisting of the integration of a centralized planner and decentralized agents was compared to the planner-only approach. Both approaches were evaluated and compared in simulations. This section initially provides a general description of the simulation design, followed by implementation details concerning each of the approaches separately.

A. Simulation Design

The agent simulation (Fig. 3) is built on top of the ROS (robot operating system) middle-ware [16]. Agents, called nodes in ROS terminology, are of two types, (i) operative agents described in Sect. IV, and (ii) special purpose agents such as the clock and the environment agents.

Operative agents have the goal of completing the tasks which are part of the mission. Agents are heterogeneous with respect to the set of capabilities they have. In addition, more than one agent can have the same capability and one agent can have more than one capability. Furthermore, they are homogeneous with respect to the implemented motion model with a maximum velocity v_{max} set to $10m/s$. Their behaviour in the *idle* state, i.e., remaining stationary, is also the same. Agents communicate with one another through the publish/subscribe broadcast mechanism, and are able to listen

to messages from each other, i.e., without a limitation in range. When an agent makes a request, it will wait until the first replies arrive or until a specified timeout of 5 seconds has passed. Consequently, not all responses from all other operative agents are necessarily considered. During the course of the simulation, any such operative agent can experience a full failure (determined in the environment node), which means that it has broken down and cannot do any task, or communicate with others.

The clock agent keeps track of the simulation time. After a simulation is initiated, the clock starts ticking when a plan has arrived from the planner. This is to ensure that across different runs of the simulations, the arrival time of the plan is the same and deterministic. The clock tick count is increased by 1 time unit every time all non broken down operative agents and the environment node have updated their state once. Also, it is assumed that any interaction between operative agents happens within the same time-step, i.e., the clock stops ticking when they are interacting, and resumes once the interaction has finished. The communication with the clock agent is realized through ROS one-to-one service calls.

The environment agent is used for three purposes: to keep track of information that concerns all agents, to determine which agent will fail at a given time-step, and as a locking mechanism. With respect to the first purpose, the information collected by the environment consists of the locations of agents and tasks in a $2D$-space, as well as lists of allocated and completed tasks by every operative agent. After the data is collected, it is broadcast as a whole – through the ROS publish-subscribe mechanism – to all operative agents, with the update taking place at every time-step. As a result, agents are aware of how tasks are allocated, and which tasks are completed at every time-step. Delays of a couple of time-steps might occur if some messages are lost or not received in time.

The second purpose of the environment is to simulate both the failure of an agent in the system, and the detection of such failure by other agents. Regarding the former, the environment initially calculates the time-step t_f in which to inject the failure as follows:

$$t_f = \mathrm{randint}(0.2 \cdot m_D, 0.8 \cdot m_D) + t_{rp}, \tag{5}$$

where the function $\mathrm{randint}$ generates a random integer that lies between the two given arguments, m_D is the estimated mission duration, and t_{rp} is the time the previous re-plan has taken place, whether by the planner or the agents. Additionally, the environment randomly selects an agent to fail from a list of agents that are currently executing. Regarding the latter, after the occurrence of a failure is simulated, the environment node informs all agents, one by one, through ROS one-to-one service calls. Note that, if an operative agent completes a task at time t_f, there is no time for such information to propagate to the rest. As a result, such a task will be reallocated and its execution will be repeated by another agent. The third purpose of the environment is to lock the tasks and the invocation of the planner. This means that, (i) two agents will not be able to initiate a negotiation for the same task at the same time-step, and (ii) agents cannot contact the planner after a call has already been placed and before a new plan has been received. In the meantime, operative agents drop their assigned tasks and change the state to *idle*. In order to lock, either the tasks or calls to the planner, agents communicate with the environment through ROS one-to-one service calls.

B. Simulation Scenarios

Every mission starts with a human operator defining a set of tasks to be executed in the MMT. After mission creation, it is forwarded to the automated high-level planner described in the Sect. V, where it is translated into ECTSP model [14]. The result of the planning process is a plan that is then forwarded to agents for execution. In both scenarios, the planner initiates the agent simulation by sending a plan with tasks allocated to agents and other mission relevant information such as the initial locations for agents and tasks, the required equipment, the task duration, and the physical capabilities and limitations of involved agents.

a) The planner-only approach scenario: In this approach, the planning and re-planning is only performed by the centralized planner. After the initial plan is generated, the planner goes into the idle state, waiting for re-planning requests, whereas agents begin executing the assigned tasks. When a failure is detected, the first agent that is able to lock the planner through the environment node, will immediately issue a re-planning request. Noticeably, there is no collaboration between operative agents. While waiting for the new plan, the clock node keeps ticking and all agents switch to the *idle* state. When the new plan arrives, t_{rp} is set to the current time t_n. It is important to emphasize that failures are induced in such a way that they cannot make a mission infeasible, i.e., the re-planning process will always be able to produce a feasible plan.

The GA planner is configured as follows. The population size is fixed to 500, and the number of generations is limited to 5000. The crossover operator was omitted, and the mutation probability is set to 10%, In addition, elitism is set to 5% in order to preserve the best candidate solutions from the previous generation.

b) The GLocal approach scenario: In this approach, the planner and agent approaches are combined, i.e., agents are able to collaborate with one another and re-allocate tasks in the case of a failure during mission execution. As in the previous approach, after the initial plan, the planner goes to the idle state, while the agents begin the execution of the assigned tasks. When a failure is detected, all other operative agents compute the list of tasks $\mathcal{V}_x$ assigned to the failed agent x, removing those tasks perceived as complete. Then, a negotiation round begins for every task $v \in \mathcal{V}_x$. The first agent that manages to lock the task through the environment node, will initiate the corresponding re-allocation, by sending out help requests to all other agents. An agent will wait for a

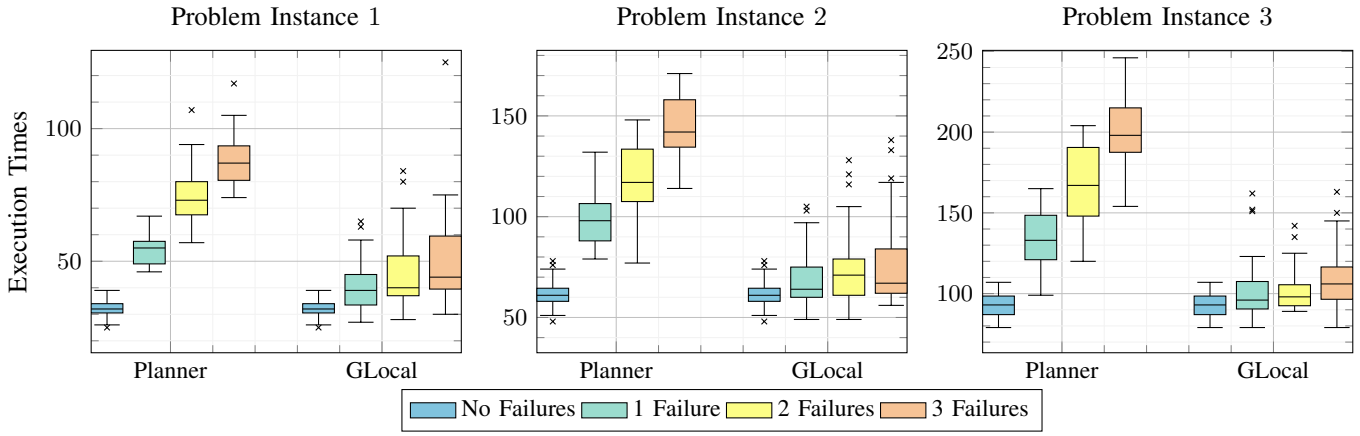

Fig. 4: Execution times for both planner and hybrid approach across problem instances 1-3.

maximum time of $t_w = 5$ seconds [4] for any replies, or until the some replies have arrived, i.e., if the list of replies is not empty when an agent makes the check, then the waiting stops, and the agent proceeds with the negotiation using the list of replies available at that moment. Thereafter, it will order the replies in descending order of the willingness to interact value, and will assign the task to the agent that has the highest value. In case the agent has the necessary equipment for completing the task, then it will itself be part of the candidates that could be assigned to the task. In case such negotiation fails, e.g., the agent does not get any positive replies, and if the agent itself is not capable of performing the task, then a re-planning request will be issued. This request will invoke the centralized planner with the updated world state information. Note that, one failed negotiation is sufficient for triggering a call to the centralized planner. On the other hand, if agents manage to re-allocate all the tasks by themselves, the planner is not engaged. In case the planner is called after an agent failure, then t_{rp} is set to the time of the arrival of the new plan. Otherwise, t_{rp} is calculated as $t_f + 3$, as it is assumed that most agents will have received such information within the 3 time-steps. A re-allocation that happens as a result of the negotiation between agents takes place within a t_f time-step. Three more extra time-steps are added to give enough time to agents to propagate the new information pertaining the newly allocated tasks, before a new failure is introduced.

VII. Results

The results of the comparison of the two aforementioned approaches are presented in this section. In order to gain an insight into the differences between the approaches, a series of statistical tests were conducted. We formulate the **null hypothesis** as *"A hybrid approach that combines a centralized and decentralized planner has no significant effect to the overall results compared to a centralized planner-only approach."*

[4]Note that, the length of the timeout is chosen arbitrarily. Depending on the needs of the application, a shorter or longer timeout could be selected. It is assumed that for the purposes of this paper $t_w = 5$ seconds is an adequate upper bound.

The data used for the comparison consist of execution times per each of 30 runs for every test case. The sample data has no normal distribution, i.e., data can be highly skewed and can have extended tail. This is the reason why the median value was chosen over the mean value and consequently the non-parametric Wilcoxon rank sum test is used. The results of the test are shown in Table I. Based on the results we can reject the null hypothesis since it is clear that the hybrid approach performs better with statistical significance. The agent only approach has been omitted from this comparison since it is generally accepted that decentralized planning results in worse sub-optimal plans compared to centralized planning in failure-free, non time-critical scenarios. This applies to the production of the initial plan in presented scenarios.

a) Benchmark Settings: There are four benchmark settings which differ on the number of agent failures f that can happen in the system, $f \in \{0, 1, 2, 3\}$. Each settings has 3 different problem instances varying in the number of tasks $n_T \in \{50, 100, 150\}$, where n_A is the number of agents. In total 36 distinct simulations are run. The number of repetitions for each case is $n_R = 30$. The number of agents in the system is fixed to $n_A = 10$. A failure cannot make the completion of the mission infeasible, i.e., there will always be at least one non-broken agent with the necessary equipment needed by any task. Additionally, the simulations run until all tasks have been completed. Therefore, all tasks will eventually be completed by the agents.

b) Scenario Comparison: The average execution times across the different number of failures, task instances, and the two approaches are given in Fig. 4. It can be observed that with the increase of the number of failures, the mission makespan increases as well in the planner-only approach. Whereas in the hybrid approach, due to the plan reparation performed by the agents, the makespan is on average lower than in the planner approach. Additionally, as the size of the problem instance increases, the average disparity between the two approaches also increases in most cases. Similarly, as the number of the failures increases in a problem instance, the p-value decreases (Tab. I). The distribution of the execution times for each of the

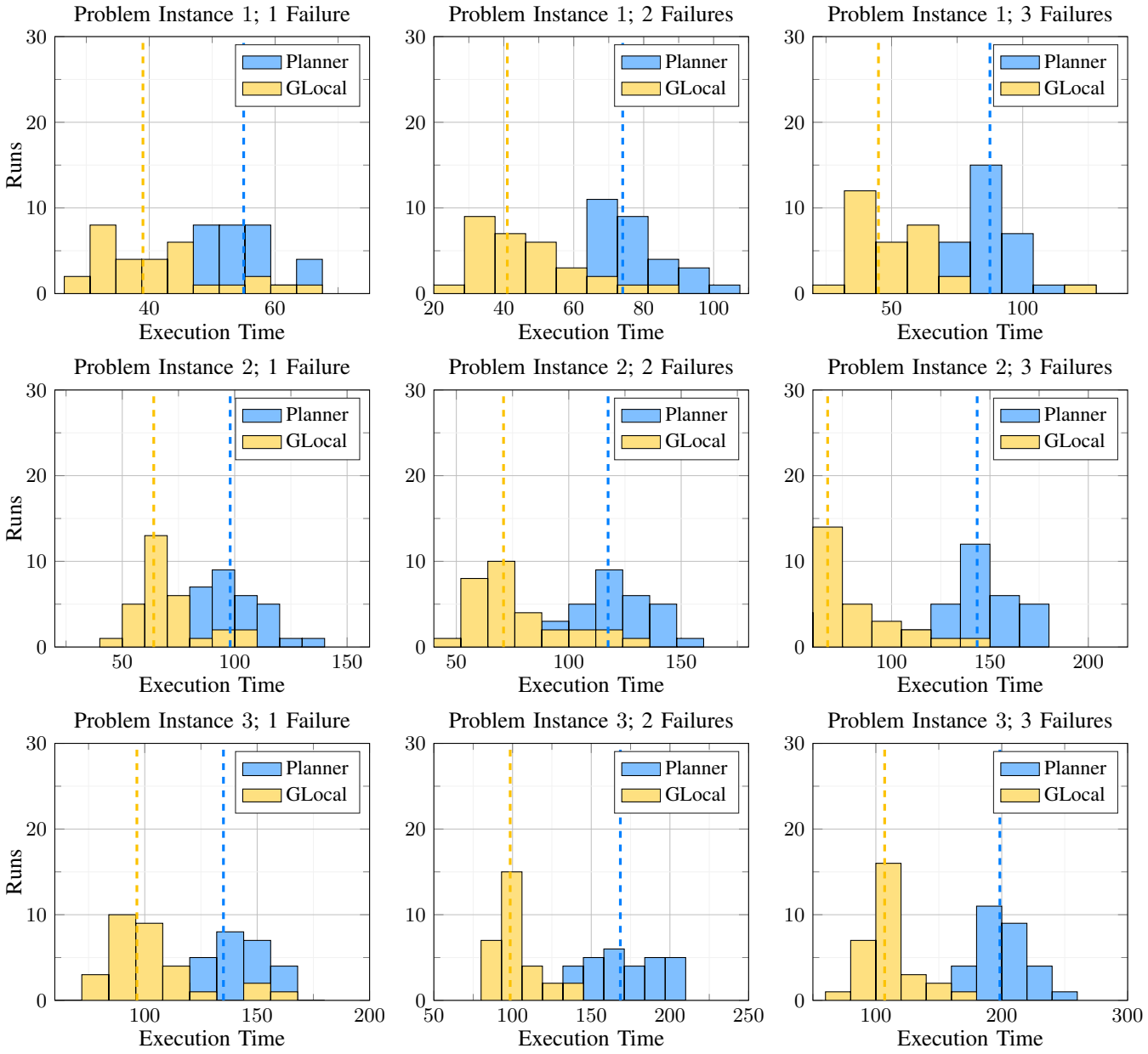

Fig. 5: The distribution of the execution times for each of the three failures in each problem instance.

TABLE I: Wilcoxon rank sum test applied on results obtained by planner only and hybrid approach.

	Problem Instance 1			**Problem Instance 2**			**Problem Instance 3**		
	1 failure	2 failures	3 failures	1 failure	2 failures	3 failures	1 failure	2 failures	3 failures
p-Value	1.3×10^{-6}	1.2×10^{-8}	6.1×10^{-10}	2.3×10^{-8}	1.4×10^{-8}	2.1×10^{-10}	6.5×10^{-7}	6.6×10^{-11}	3.3×10^{-11}

test cases can be seen in Fig. 5. The dashed lines represent median values. It can be observed that the median value of the hybrid approach is always better (in this measure lower is better) than the median value of the planner-only approach. Although, some of the solutions overlap, it is clear that the hybrid approach, in general, produces better solutions than the planner-only approach.

For the hybrid approach, the cumulative number of re-plans by the planner and the agents for each failure case across the three problem instances is calculated, and given in Fig. 6. Note that the total number of re-plans in a simulation run is equal to the number of failures f. As a result, the total number of re-plans can be calculated as $f \cdot n_R = \{0, 30, 60, 90\}$. Additionally, by inspecting Figs. 4 and 6, it is possible to note the influence of the number of times agents locally repair the plan over the execution times. In problem instance 1, failure

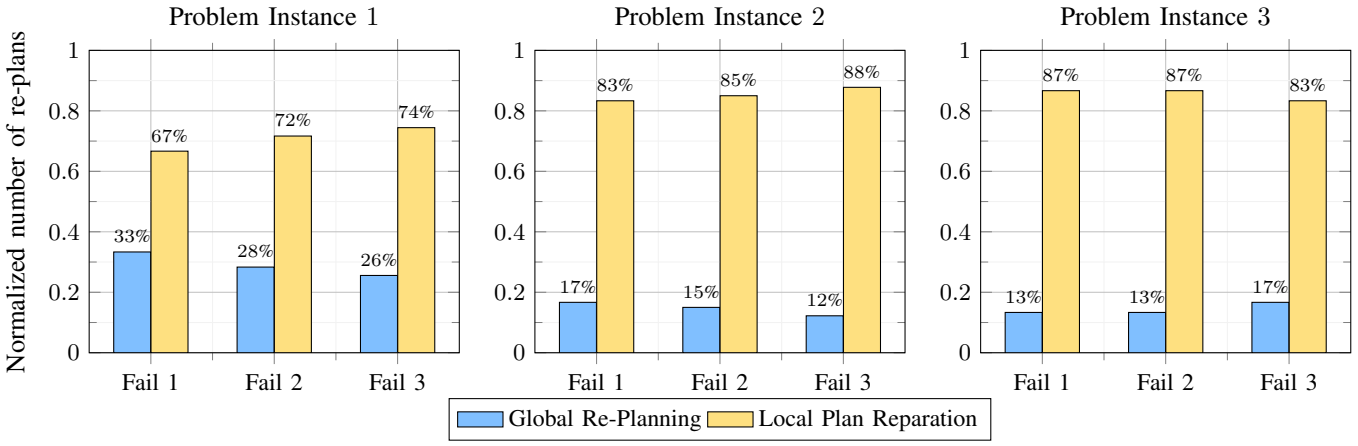

Fig. 6: Normalized cumulative number of re-plans over 30 runs for each problem instance.

case $f = 1$, agents do not invoke the planner 67% of the time a re-plan is needed (Fig. 6), thus impacting the variability of the execution times, which partly overlaps the results gained with the planner-only (Fig. 4). It can be noted that, depending on the problem instance, when a re-plan is needed the local plan reparation technique, involving only agents, is able to overcome the failure in the system and re-allocate tasks from the failed agent in 67–88% of the cases.

VIII. Related Work

To the best of authors' knowledge, only two works aim at addressing the problem of combining centralized and decentralized planning approaches in order to solve multi-robot task allocation problems. Le Pape has argued for the necessity of combining centralized and decentralized planning approaches to deal with uncertainties and unforeseeable events in dynamic environments [12]. Agents were given the option to create their own individual plans in a decentralized way. However, in the case of addition of new tasks into the system, e.g., as a result of the action taken by a human operator, agents invoke the centralized planner to make the task allocation of newly added tasks. Thereafter, agents proceed with the execution of the tasks. Duan *et al.* consider the online dispatching problem for dynamic autonomous taxi operations, the aim of which is to assign requests to the taxis in the system, while maintaining the feasibility of existing routes [4]. Requests can arrive at any time, and the time aspect is divided into short-term and long-term horizons. An immediate request is part of the short-term horizon, and is dealt with by a centralized planner that makes the assignment. Reservation requests, on the other hand, belong to the long-term horizon, and are dealt with by the autonomous taxis. In this approach, the taxis integrate these requests such that their routes remain feasible, until they are part of the short-term horizon and the planner makes the assignment. Their goal is to reduce the planner workload from requests that are too far in the future.

A more detailed comparison, from the problem definition viewpoint, is provided by expressing the problems addressed in related works, as well as the problem tackled in this paper, through the TAMER model proposed by Miloradović *et al.* [15] (Tab. II). TAMER is an entity relationship model that characterizes multi-robot allocation problems through four entities – *Robot*, *Environment*, *Mission*, *Task* and the relationships between these entities such as *teamed* (Team.), *communicate with* (Comm. with), *deployed* (Depl.), *includes robots* (Incl. R.), *includes tasks* (Incl. T.), *allocation* (Alloc.), *depend on* (Dep. on), and *decomposed off* (Dec. off) [5] *Teamed* and *Communicate with* capture the relationship between instances of the *Robot* entity, in terms of how robots collaborate with one another, and lower level concerns for communication, e.g., bandwidth, range etc. A *Mission* instance is *deployed* in an instance of Environment. Additionally, a *Mission includes* a set of *Robot* and *Task* entities. *Task* entities are connected with one another through the *depend on* and *decomposed off* relationships. Finally, the *Robot*, *Task*, and *Mission* entities are linked through the *allocation* relationship that captures properties such as the allocation type (instantaneous assignment versus time-extended) and utility function.

IX. Conclusion

In this work, a hybrid approach for multi-agent mission re-planning is proposed. This approach combines high-level global planning realized by a centralized planner, with a local plan reparation technique carried out in a decentralized manner by a group of agents. In the event of full failures, i.e., agent breakdowns, the remaining agents attempt to repair the plan locally by re-allocating any pending tasks among each other. If at least one task remains un-allocated, agents invoke a centralized planner which generates a new plan, for all agents and unfinished tasks, based on the updated information received from the agents. The hybrid approach is compared to a planner-only approach in simulations, and it is shown that on average it achieved better results, i.e., shorter mission make-span as compared to the latter, in the presence of different numbers of failures.

[5]The *act* relationship binding the *Robot* and *Environment* entities has been omitted from Table II, due to no specification in the given works.

TABLE II: Using the TAMER model [15] to classify the problems addressed in combined centralized and decentralized planning.

Ref. \ TAMER	Entities				Relationships							Approach
	Robot	**Env**	**Mission**	**Task**	**Team.**	**Comm. with**	**Depl.**	**Incl. (R&T)**	**Alloc.**	**Dep. on**	**Deco. of**	
Le Pape [12]	finite number of sensors; can determine action outcome; can plan; can generate goals	uncertainty (open);	tasks; capability constraints	single-robot task; capability needed (mobility)	–	communication with planner; communication between agents; asynchronous communication	Graph	n heterogeneous R;	minimize plan duration; wait or plan with current knowledge; 2 planning levels (planner & robot); time-extended	precedence	divisible into actions	Human/agent give planner a new goal; planner informs available agents; agents create individual plans for achieving the goal; planner collects the proposals and makes the final allocation; agents execute in a decentralized way
Duan *et al.* [4]	state (availability, location); able to estimate travel times; single-task robot	uncertainty (closed); discrete;	tasks; timing constraints;	single-robot task;	–	communication with: planner, clients, and other taxis	Graph	n functionally homogeneous R; constantly incoming T (requests)	minimize plan duration; minimize disjoint paths for traversing all nodes; time-extended	timing	atomic	Planner deals with requests in the short-term horizon; agents incorporate reservations far in the future in the long-term horizon, in order to maintain route feasibility;
Proposed Approach	state (location); finite number of capabilities; adaptive autonomy; determine who is broken; generate own goals	uncertainty (open); observability (fully); deterministic; discrete (with respect to time);	tasks; capability constraints; number of robots constraints	single-robot task; equipment needed	teams of size 1; no hierarchy between agents	broadcast	2D space	n heterogeneous R; m heterogeneous T	minimize plan duration; time-extended	no dependencies	atomic	Plan initially issued by the centralized planner; agents attempt execution; on failure agents attempt to repair the plan; in case of non-success in re-allocation agents ask for a re-plan.

There are three lines of inquiry for future work. First, it is of interest to investigate the performance of the hybrid approach in the presence of partial failures of agents. A partial failure could mean a faulty equipment for an agent that prevents the execution of some of the tasks allocated to the agent. Second, the proposed approach – with additional local strategies by agents for an efficient local re-planning – can be applied to solve the ECTSP with precedence constraints among others. Finally, different strategies for determining the re-planning threshold, i.e., when to call the planner, are to be investigated.

REFERENCES

[1] Beltrame, G., Merlo, E., Panerati, J., Pinciroli, C.: Engineering safety in swarm robotics. In: Proceedings of the 1st International Workshop on Robotics Software Engineering. pp. 36–39. ACM (2018)

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

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

[4] Duan, L., Wei, Y., Zhang, J., Xia, Y.: Centralized and decentralized autonomous dispatching strategy for dynamic autonomous taxi operation in hybrid request mode. Transportation Research Part C: Emerging Technologies **111**, 397–420 (2020)

[5] Frasheri, M., Cürüklü, B., Esktröm, M., Papadopoulos, A.V.: Adaptive autonomy in a search and rescue scenario. In: 2018 IEEE 12th International Conference on Self-Adaptive and Self-Organizing Systems (SASO). pp. 150–155. IEEE (2018)

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

[7] Hamann, H.: Swarm Robotics: A Formal Approach. Springer International Publishing (2018)

[8] Hedin, Y., Moradian, E.: Security in multi-agent systems. Procedia Computer Science **60**, 1604–1612 (2015). https://doi.org/10.1016/j.procs.2015.08.270

[9] Karaman, S., Shima, T., Frazzoli, E.: A process algebra genetic algorithm. IEEE Transactions on Evolutionary Computation **16**(4), 489–503 (2012)

[10] Laing, T.M., Ng, S.L., Tomlinson, A., Martin, K.M.: Security in swarm robotics. In: Handbook of Research on Design, Control, and Modeling of Swarm Robotics, pp. 42–66. IGI Global (2016)

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

[12] Le Pape, C.: A combination of centralized and distributed methods for multi-agent planning and scheduling. In: Proceedings., IEEE International Conference on Robotics and Automation. pp. 488–493. IEEE (1990)

[13] Maoudj, A., Bouzouia, B., Hentout, A., Toumi, R.: Multi-agent approach for task allocation and scheduling in cooperative heterogeneous multi-robot team: Simulation results. In: IEEE Int. Conf. on Industrial Informatics (INDIN). pp. 179–184 (2015). https://doi.org/10.1109/INDIN.2015.7281731

[14] Miloradović, B., Çürüklü, B., Ekström, M., Papadopoulos, A.V.: A genetic algorithm approach to multi-agent mission planning problems. In: International Conference on Operations Research and Enterprise Systems. pp. 109–134. Springer (2019)

[15] Miloradović, B., Frasheri, M., Cürüklü, B., Ekström, M., Papadopoulos, A.V.: TAMER: Task allocation in multi-robot systems through an entity-relationship model. In: International Conference on Principles and Practice of Multi-Agent Systems. pp. 478–486. Springer (2019)

[16] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. In: ICRA workshop on open source software. vol. 3, p. 5. Kobe (2009)

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

[18] Spears, D., Kerr, W., Spears, W.: Safety and security multi-agent systems. In: Safety and Security in Multiagent Systems. pp. 175–190. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)

[19] Stanković, R., Štula, M., Maras, J.: Evaluating fault tolerance approaches in multi-agent systems. Auton Agent Multi-Agent Syst **31**, 151–177 (2017). https://doi.org/10.1007/s10458-015-9320-6

[20] Torreño, A., Onaindia, E., Komenda, A., Štolba, M.: Cooperative multi-agent planning: A survey. ACM Comput. Surv. **50**(6), 84:1–84:32 (Nov 2017). https://doi.org/10.1145/3128584

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