Multi-Agent Task Allocation for Temporally Located Tasks in a Dynamic, Distributed Environment

Gialon Kasha

CMU-CS-23-123

August 2023

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Stephen F. Smith, Chair
Zachary Rubinstein

Submitted in partial fulfillment of the requirements for the degree of Master's of Science

Keywords: Multi-agent task allocation, auction, re-planning, execution monitoring

Abstract

The long-term maintenance of deep-space habitats requires the ability to dynamically allocate spatio-temporal tasks and robustly handle their execution. A centralized framework becomes infeasible in this setting, as the planning space can become intractable over long-term horizons, as well as the fact that single-point failures may be common and hard to recover from. Thus we utilize a decentralized, market-based approach to solve the multi-agent task allocation problem. To respect temporal constraints, we utilize STNs as a core component of each agent's execution monitoring and scheduling mechanism. We then conduct a comparison of single-agent versus multi-agent re-planning mechanisms, and identify the contexts in which each may outperform the other. We also introduce a novel positive re-planning technique that utilizes a reverse-allocation auction mechanism to match tasks to slots. Ultimately, we find that when optimizing for makespan, the multi-agent approach consistently beats the single-agent one. However, when optimizing for certain metrics such as travel time, the single-agent can fare better in specific environments. Therefore, we conclude that the multi-agent approach is better suited for the deep-space domain, suggesting that more work could be done by considering alternate objective functions.

Acknowledgments

This work would not have been possible without the help of my incredible mentors Robert Smith, Brian Coltin, J. Benton. I am extremely grateful for their support, as well as for the support of the larger NASA Ames community.

I would also like to thank my advisors Stephen Smith and Zachary Rubinstein. Their guidance greatly helped shaped my academic career, as well as my individual growth outside of work.

Finally, I am incredibly thankful for the encouragement and support of my friends and family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In deep-space exploration, it is likely that constructed habitats will experience long, un-crewed periods of time. Such periods can prove detrimental to the functionality of a habitat: leaks may occur, filters may need to be routinely replaced, and the system as a whole may degrade over time. Thus, in order to ensure the high-functioning capabilities of the system, the habitat must be able to maintain itself.

In particular, a team of robotic agents may be responsible for carrying out maintenance tasks such as conducting leak repairs, filter replacements, and cargo loading. These tasks may have deadlines (e.g. a leak must be repaired within an hour to maintain the current air pressure) and possible coordination constraints (e.g. two robots are needed to lift heavy cargo bags). Furthermore, due to the unpredictability of real-world environments, tasks may fail or get delayed. Thus, a task allocation system must be utilized that is able to efficiently determine allocations in a robust manner, so that it can handle the online assignment of tasks and the dynamic nature of the environment.

On the other hand, just as easily as tasks can get delayed due to unforseen changes in the environment, it is possible for tasks to complete earlier than expected. In particular, one might imagine that in order to build a more optimal system, heuristics might be used to determine upper and lower bounds for durations when scheduling tasks. Over time, as more data is collected, these bounds may grow tighter. However, due to certain tasks already being in a partially-completed state or improved capabilities of the agents, the actual durations may occasionally be shorter than expected. In such cases, the system must be able to re-allocate tasks to take advantage of the newly available free-time, a process which we shall call "positive" re-planning.

The above considerations are not necessarily specific to the space domain. In particular, in search-and-rescue missions, similar considerations must be made when designing software systems that can handle the dynamic environment. However, a more unique aspect of the deep-space setting is that the entire system operates in isolation. To be more specific, if something goes wrong, such as an agent becoming disconnected from the rest of the network, or a component failing, the rest of the system must be able to operate at a sufficient capacity to maintain the habitat. In other words, the system must robust in dealing with single-point failures.

Thus, it becomes unreliable to use a centralized architecture for planning and execution monitoring, since the system must be able to respond to new tasks and events at any time. It is therefore desirable to utilize a decentralized approach where any agent can be responsible for

determining task allocations and for monitoring their own task execution.

Our work is thus two-fold: First, we focus on the online assignment of temporally located tasks in a dynamic, distributed, multi-agent environment and secondly, we investigate the different contexts and replanning strategies for agents in this setup. Due to possible deadlines and coordination constraints of tasks, our allocation system must also take into account the temporal constraints of tasks and agent timelines. In order to account for all of these different factors, we opt for a market-based approach, utilizing multi-agent auctions to determine task allocations. This approach is particularly useful, as it allows agents to execute multi-agent re-planning strategies in a natural way.

Chapter 2

Related Work

Market-based methods have commonly been used to solve multi-agent task assignment problems [6]. The two main types of approaches tend to be consensus-based [5] and auction-based ones [7]. A recent survey analyzes the different approaches and notes that the current research direction favors consensus-based approaches and sequential/parallel single-item auctions, however they argue that in dynamic environments, traditional single-item auctions may be favorable [9].

Due to the problem setup, we adopt a single-item auction approach using a distributed architecture. This design and approach has been used in a variety of contexts. One paper uses this approach to determine task allocations for coverage tasks of radio spectrum use [10]. However, the paper deals with atomic tasks, and doesn't look at tasks that may have coordination requirements or dependencies. In general, these dependencies require a more complex bidding structure, so hierarchical auctions have been used in such settings [8]. While this solves the dependency problem, the proposed approach was not implemented in a dynamic environment, nor does it take into account temporal constraints. Being able to handle both of these factors is critical for our problem setting.

Recent work has looked at a combination of these factors in the context of law enforcement, using convex optimization to determine efficient allocation of tasks amongst agents [1]. However, the paper doesn't focus on the execution monitoring that occurs allocating tasks, which is an important aspect in building robust systems.

Closely related to our work, Barbulescu et al. focuses on the allocation and execution cycle, using Simple Temporal Networks to monitor task execution in combination with market-based approaches in the context of disaster-response missions [2]. Here, an auction approach is used in a distributed, dynamic environment. However, the proposed replanning mechanism only is triggered when tasks are delayed or become infeasible, not taking into account the possibility that agents may become available due to tasks finishing earlier than expected.

More generally, the notion of triggering replanning due to positive outcomes has received much less attention compared to its negative replanning counterpart. Some work has focused on triggering replanning when opportunities arise during autonomous underwater vehicle missions[4]. Recent work builds on this, by developing an algorithm to autonomously compute opportunities[3]. In both cases, opportunities are considered to be facts that are not initially known at the time of planning, but may arise during execution and trigger replanning due to their high utility. However, both of these papers only consider a single-agent context. Expanding this approach to a

multi-agent context is non-trivial, as a variety of questions then arise relating to how opportunities are shared. In particular, if one agent discovers a new opportunity, which agent should be able to use it? As we are not working within the PDDL planning framework, our work considers opportunities to be newly discovered slots of free time that result from tasks completing earlier than expected and from completing one's schedule. We use a reverse-allocation auction, where agents offer up slots, and agents propose tasks in the forms of bids.

Thus, our work attempts to synthesize previous approaches, by extending an auction-based approach in a distributed architecture to also trigger positive-replanning via reverse-allocation auctions.

Chapter 3

Methodology

This section describes the overall approach utilized in this work. It proceeds in the following way:

- High-level overview of the system
- Description of STNs and Timelines
- Auction Mechanism
- Replanning Strategies

3.1 High-level Overview

A high-level overview of our system is shown in figure 3.1.

It consists of 3 components (all ROS nodes): a Goal Manager, agent nodes, and an Execution Interface. The goal manager and execution interfaces mainly exist for the purpose of simulation. Specifically, the goal manager receives incoming goals from an external client and designates an agent to be an auctioneer. This is done by publishing a goal message on a ROS topic and choosing the first agent that responds. Alternative implementations could utilize multiple goal managers to allocate tasks to each agent, where each goal manager oversees a team of agents that are responsible for a particular component of the habitat. However, in our experiments, only one goal manager was used.

The execution interface allows agents to simulate the execution of their actions. In order to begin the execution of an action, agents send a ROS message to the interface describing information about the task. The execution interface then responds with either a confirmation or failure, depending on the execution outcome. Additionally, we can interact with the execution interface to create delays or trigger failures for specific actions to test the robustness of the system.

The agent nodes are the core components of the decentralized system. While they interact with the goal manager and execution interface nodes, they also interact with each other to determine task allocations. Essentially, after being designated an auctioneer by the goal manager, they host a single item auction for the given task. Agents then submit bids that represent their ability to achieve the auctioned task, and a winner is chosen based on some objective function (usually makespan). In order to understand the bid generation process, we now discuss Simple

Temporal Networks and their use in managing agent timelines.

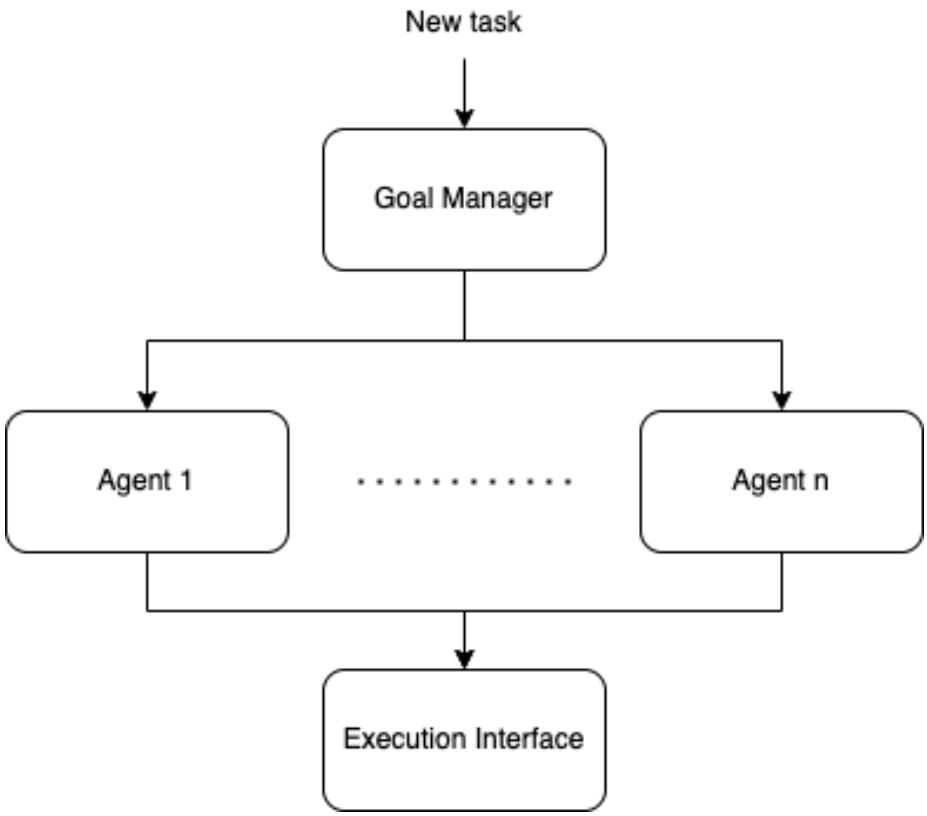

Figure 3.1: High-level system overview

3.2 Simple Temporal Networks and Agent Timelines

As mentioned, the problem setting involves deadlines and temporal constraints. Thus, it is necessary to utilize a mechanism that can track these constraints and ensure that the generated schedules are temporally consistent. To this end, we use Simple Temporal Networks (STNs).

Formally, an STN is a graph $G = (V, E)$, where V is a set of timepoints and E is a set of edges that correspond to temporal relations between timepoints. Temporal constraints are introduced to the graph via a tuple (x, y, lb, ub), indicating that timepoint y must occur after timepoint x within the interval $[lb, ub]$. Typically, we model a task by creating a start and end timepoint, and assigning them a lower bound for duration. It is also common to connect the start and end of a timeline with a "calendar zero" node via a $(0, \infty)$ sequencing constraint and a $(0, \text{deadline})$ constraint, respectively. Here, "calendar zero" refers to an absolute zero time reference point, indicating the beggining of time for the system. An example of this is shown in figure 3.2, where the task has a lower bound duration of 10, and the entire network has a deadline of 30. Solid edges indicate actual edges in the STN, while dashed lines indicate the temporal constraints that were given to the STN. Edges to the "cz" timepoint are omitted for simplicity.

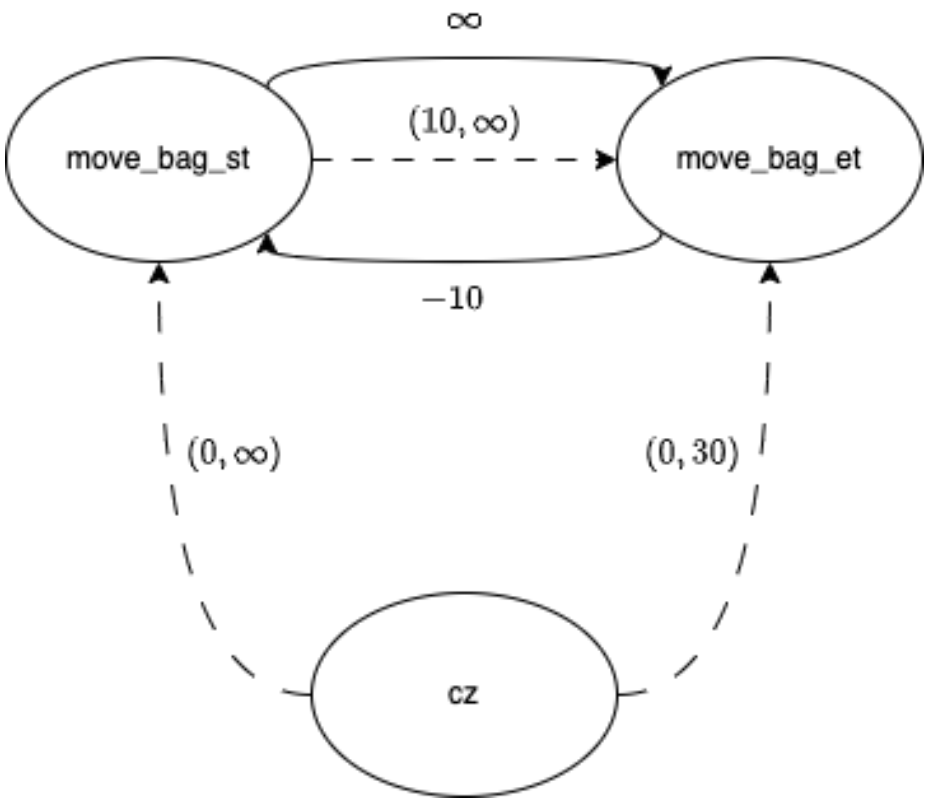

Figure 3.2: Example of STN for a move_bag task

Given an STN, one can extract the range of feasible values for a specific timepoint, giving us the interval in which it can occur. This is done by maintaining lower and upper bound shortest path trees, where the source node is "cz". In our figure 3.2 example, the move_bag_st must occur within the range $(0, 20)$, as any later would result in the move_bag_et timepoint occurring after the deadline. Furthermore, an STN is said to be temporally consistent if it contains no negative cycles. Looking at our example STN, we can see that this intuitively makes sense: If a negative cycle existed, we could traverse it a large number of times to lower the shortest path distances of the move_bag_et timepoint, making its range of feasible values come before that of the move_bag_st timepoint, which would violate our intial constraint that the end of the task must come after the start of it. Thus, by continually modifying our STN to represent the actual execution of tasks, we can track the temporal consistency of our schedules.

Each agent is therefore equipped with an STN to manage their task schedules. They track the execution of their tasks by creating a "now" timepoint that represents the current time, and connect it with the appropriate timepoint to track the execution of tasks. If a task takes too long, or violates the temporal consistency of a schedule, the STN will fail to add the constraint, and the appropriate replanning mechanism will be invoked. An example agent STN is shown in figure 3.3 and its corresponding timeline is shown in figure 3.4, where green nodes correspond to completed tasks, orange nodes correspond to tasks that are in execution, and grey nodes are tasks that have yet to be executed. With an understanding of how STNs work, we can now discuss the

auction and scheduling mechanisms that the agents use to allocate tasks and generate bids.

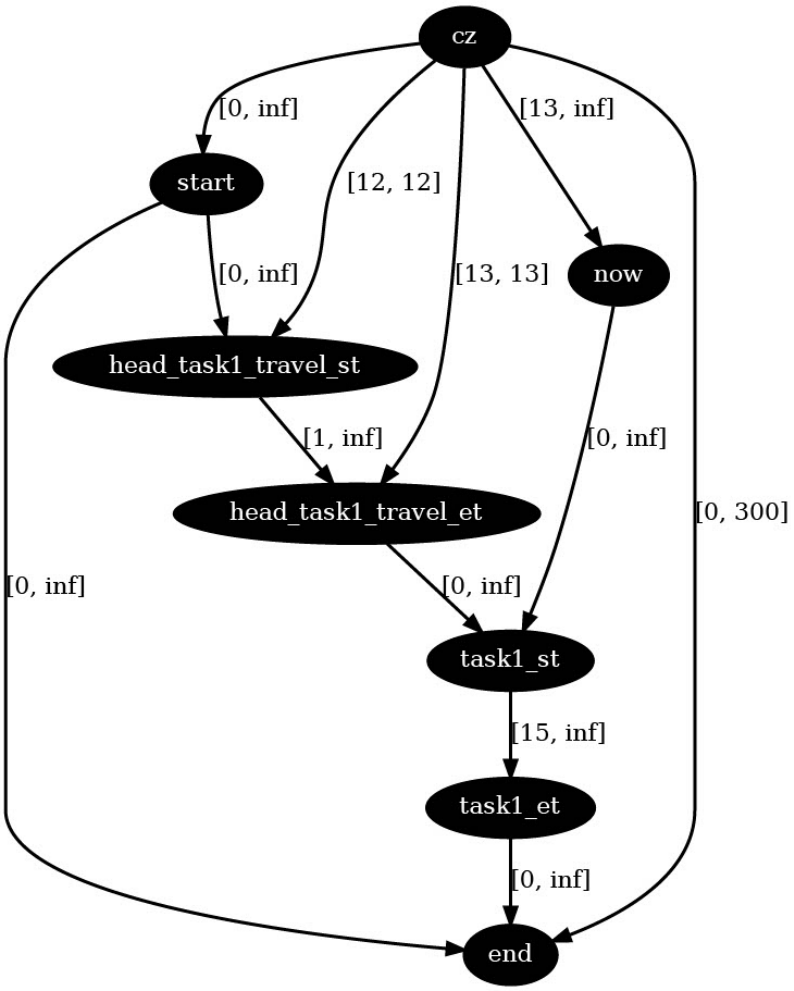

Figure 3.3: Example of an agent STN

Figure 3.4: Example of an agent timeline.

3.3 Auction Mechanism

This work uses a single item auction to allocate tasks. There are four phases to this auction scheme:

1. Auction announcement
2. Bid Generation
3. Winner Determination
4. Winner Confirmation

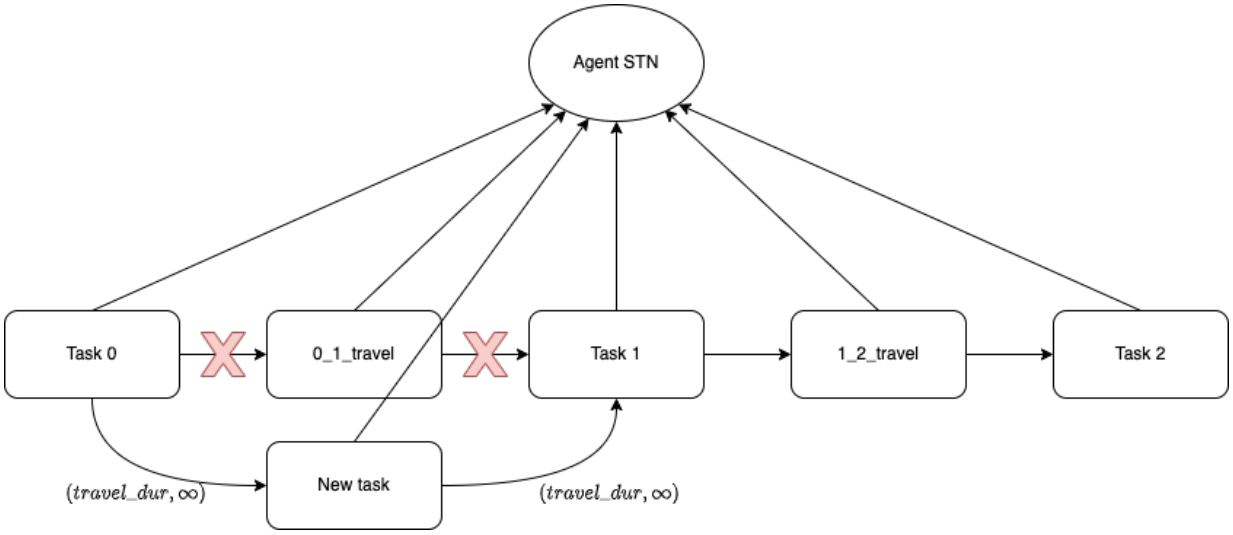

Figure 3.5: Bid generation illustration

3.3.1 Auction Announcement

To announce an auction, an agent takes the goal that they received from the Goal Manager, and construct a task message. Here, a task message consists of information such as the task id, the location of the task, the duration and expected range of the task, and other relevant information. They then publish this message on an auction topic, signaling the start of an auction. All agents will be constantly monitoring this topic, so once a task message is received, they begin the bid generation phase.

3.3.2 Bid Generation

In this phase, each agent scans their timeline, and attempts to schedule the task, taking into account travel durations. In particular, their timeline is a linked list of task nodes, where each node connects to the STN via a start and end timepoint. A visual example of this is shown in figure 3.5.

To schedule a task, they scan their timeline, jumping between non-travel task nodes. At each node, the travel task is temporarily unscheduled by removing the corresponding timepoints and

constraints from the STN. Then, a timepoint representing the new task is added to the STN, as well as constraints that correspond to the travel durations between tasks, and the task duration itself. Furthermore, if the task is a joint activity, then additional coordination constraints are added to the time point.

If all of the above can be done without violating the temporal consistency of the STN, the agent also modifies the start time to account for the auction duration. Once this has been done, and if the task can still be completed within the slot start and end times, then we have found a valid slot for which a bid is generated. Since the agent has not explicitly scheduled the task for a specific slot, this bid represents a weak commitment: the agent states that it can execute the task within the slot, but it is not guaranteed that upon winning a task, this slot will be available. Weak commitments allow the agent to participate in multiple auctions and execute its existing schedule, without having to wait for a particular auction to complete.

Finally, the bid is then sent to the auctioneer, describing the start and end time of the slot, as well as the makespan associated with scheduling the task for this slot. Doing so allows the auctioneer to consider multiple bids from an agent, which can be particularly useful when trying to allocate joint tasks.

3.3.3 Winner Determination

After some specified time, the auctioneer closes the auction, choosing to stop receiving bids. In the case of a task that only requires a singular agent, the winner determination process is fairly simple. In particular, the auctioneer first sorts the bids by makespan, and then by earliest start time. This ensures that the auctioneer is choosing the bid that minimizes the overall makespan of the agent schedules, and favors the bids that start at earlier times if the makespans are equal.

In the case of a joint activity, the auctioneer must do extra work. A joint activity is one that requires multiple agents. In order to determine the best set of agents to allocate the task to, the auctioneer takes the set of bids and generates all overlapping intervals which can support the temporal constraints of the task. It then sets the bid value of the overlapping interval to be the maximum makespan of the involved bids. Finally, it chooses the bid with the best bid value, which is the bid with the minimum makespan and earliest start time.

Since this is a decentralized execution framework, an additional measure must be taken to enforce coordination. To this end, when generating a winner announcement message, the auctioneer adds an additional start-time constraint for the task, requiring the task to start at the beginning of the interval. When scheduling the task, the agents must take this start time into account, and begin execution within some specified threshold.

3.3.4 Winner Announcement and Confirmation

Once a set of winners has been determined, the auctioneer sends out messages announcing the allocations. Each agent then attempts to schedule the task, based on the specified temporal constraints. Since agents made a weak commitment when generating bids, it is possible that they are unable to actually schedule the task after the auction has finished. Thus, they must publish a confirmation of whether or not they were able to successfully schedule the task. If not, then

the auctioneer chooses the next best set of winners, continuing until all agents of the winning set have reported a scheduling confirmation.

3.4 Replanning Strategies

As previously mentioned, the execution environment is dynamic, so actions may fail or get delayed. Thus, the agents need to be able to monitor their action execution and robustly manage their schedules, in the case that something goes wrong. There are two main strategies that we consider, single-agent replanning and multi-agent replanning. In general, we try to avoid rescheduling joint activities, as these have strict coordination constraints. Thus, agents will often attempt to re-optimize their schedule by re-sequencing single agent activities without hard start-time constraints.

In our simulated environment, all actions have a nominal duration, as well as a range in which they can occur. However, when scheduling tasks and incorporating them into STNs, we schedule them based on this nominal duration, considering a subset of this range. For example, if a task T has a duration of $dur(T)$ and an execution range of r, then the task would be incorporated into an STN via the constraint

$$< T_{st}, T_{et}, dur(T) - r \cdot k, dur(T) + r \cdot k >,$$

where $k \in [0, 1]$.

During execution, an agent will be constantly modifying their STN according to the feedback they receive from the execution interface, in order to accurately model the environment. Due to how we model tasks in our STN, there are three types of outcomes that can result from the execution of a task.

A negative outcome is one that results from a task taking longer than expected, either due to a delay or a failure. Complementary, a positive outcome is one that results from a task completing earlier than expected. In general, the appropriate response to one of these outcomes is to trigger replanning and try to optimize the scheduling by moving around future tasks.

A neutral outcome is one that results from a task completing within the expected range that we modeled in our STN. In this case, no replanning is necessary.

3.4.1 Single-agent replanning

In a single-agent replanning strategy, an agent is unable to communicate with other agents in an attempt to improve their schedule. Thus, the strategy is relatively limited. In the negative outcome case, an agent tries to move to-be-executed tasks down their timeline, creating space for the current task to finish executing. In the positive outcome case, an agent tries to move up tasks on their timeline. Both of these behaviors are conducted in similar ways, with the agent either looking for a slot that fits the upcoming task, or looking for a task that fits the newly created slot.

3.4.2 Multi-agent replanning

In the multi-agent replanning strategy, the behaviors are more complex. An agent can now take advantage of the existence of other agents, and can use them to optimize its schedule. In the negative outcome case, an agent tries to create more slack in their schedule by auctioning off upcoming tasks. To do this, they use the same auction mechanism as previously described, choosing the agent with the best bid as the new owner. Here, the best bid is the one with the lowest overall makespan and the earliest start time. Any existing coordination constraints are included in the new auction, to ensure that coordinated activities remain so. Notably, the agent itself can take part in this auction.

The positive outcome scenario operates in a similar, but reverse way. Here, the agent identifies the newly created slot that results from the previous task completing early. They then host a reverse auction by publishing this slot, indicating that they are a free agent that can take on tasks within this window. The other agents then look at their existing schedules, and try to find the best task that fits within the slot. In this case, the best task is the one that results in the greatest change in makespan after removing it from the schedule. The auctioneer then takes these tasks and chooses the best one such that the task results in the largest reduction in the overall makespan of the bidders, taking into account its own makespan. It then publishes a message, signaling the winning agent that they have scheduled the task.

Chapter 4

Performance Evaluation

4.1 Experimental Design

To evaluate the two replanning strategies, we conduct tests on a variety of scenarios. There are three types of environments that we will look at that represent different habitat configurations, all on the X-Y plane.

The first habitat is a 100x100 square, where task locations are chosen uniformly at random from within this square. This might correspond to a large habitat with a singular module, or main control center. The second habitat consists of a circle with a 25-unit diameter that is also enclosed by a 100-unit diameter circle. Tasks are chosen to be in the inner circle with a 70% probability, and are chosen to be on the outer circle with a 30% probability. Such an environment corresponds to a smaller habitat with nearby research stations that are used to conduct scientific experiments. The third setting consists of a series of 10x10 squares that are arranged in a grid-like formation. Tasks are assigned to each square uniformly at random, and their locations are chosen within each square uniformly at random, as well. This environment corresponds to a city-like habitat configuration, where a bunch of modules may be interconnected. A visual illustration of these different environments are shown in figure 4.1. For our experiments, the agents will begin at the center of each map upon initialization, but will be free to move throughout the map.

For each environment, 25 tasks will be allocated, with 20% of them being joint activities, requiring two agents to complete. Furthermore, the duration of each task will chosen uniformly at random from a range of 50-150 seconds, with each agent scheduling it to occur between 70-130 seconds. This allows tasks to finish earlier and later than expected, creating or removing slack from a possible schedule. Each setting will be run with 8 agents, that can move at a speed of 1 unit per second. There are no obstacles or restrictions on the movement of agents within each environment.

We will generate 10 problem instances to run on each environment. Both the single-agent and multi-agent approaches will be tested on the same problems. To ensure a fair comparison, the same random seed will be used to generate an list of execution times for all tasks at the beginning of the experiment.

The metrics that we will be measuring are the makespan (total time it takes to complete all tasks) of schedules across different envrionments, the average variance of individual agent

makespans, and the travel times associated with each test. Doing so will allow us to compare the efficiency and optimality of the allocations given by the single-agent versus multi-agent approaches.

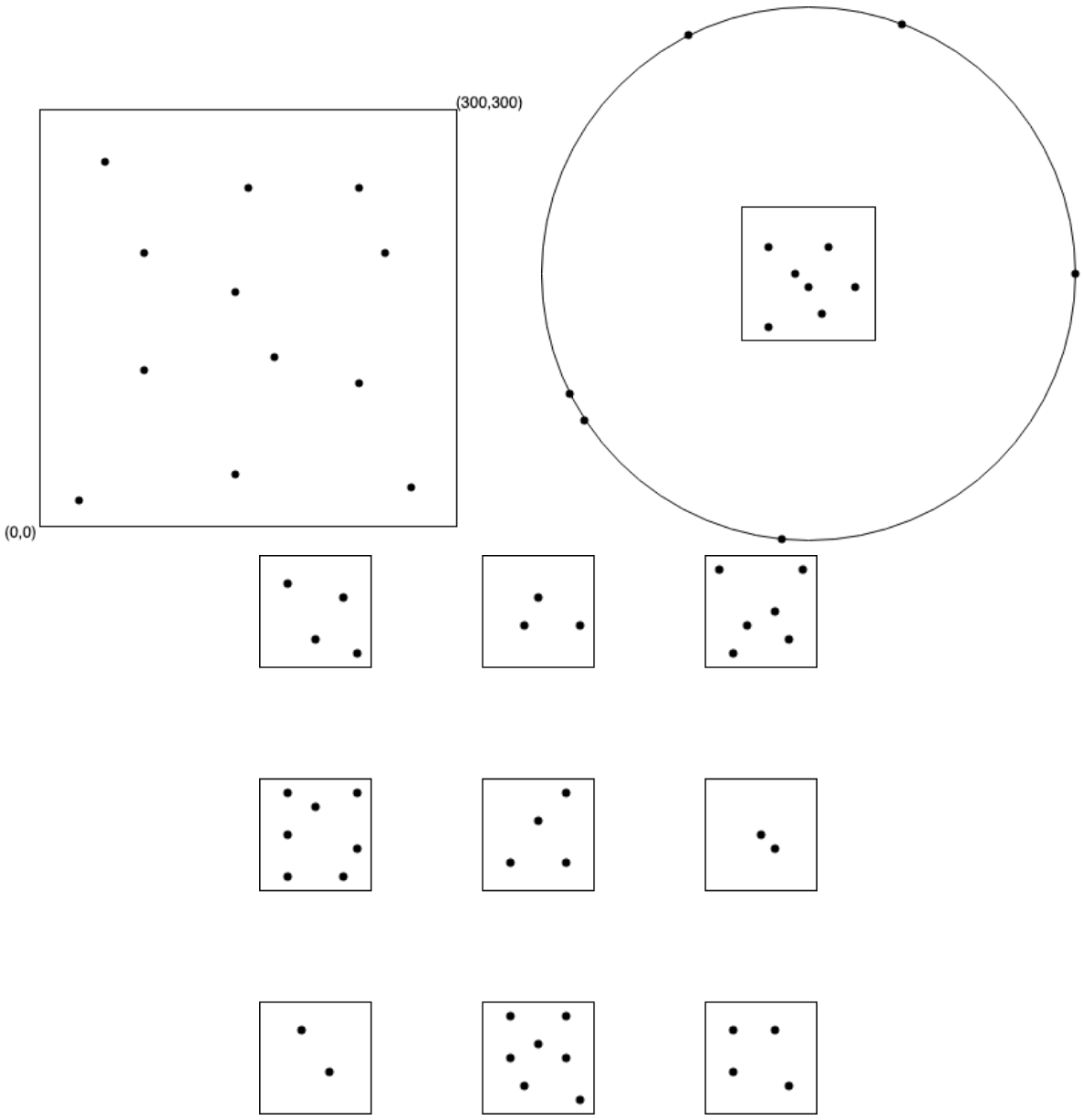

Figure 4.1: Square habitat, research station habitat, and grid-like habitat for experiments

4.2 Results

The data from our experiments is shown below.

Table 4.1: Average Makespan over 5 trials

	Multi-Agent	Single-Agent
Square	738.2	828
Circle	739.6	776
Grid	729.2	784.2

Overall, we see that the multi-agent approach tends to outperform the single-agent one, regardless of the environment setup.

On average, we see that in Table 4.1, the multi-agent approach had reported makespans that were 8% better than the single-agent approach. The largest improvement in makespan occurred in the square habitat environment. This larger improvement may be due to the fact that since tasks are uniformly distributed within the square, it is likely that in the case of delays, there are other nearby agents onto which future tasks can be offloaded, in order to re-optimize existing schedules. A similar argument can be made for the case where tasks complete earlier than expected.

On the other hand, the smallest improvement occurs in the circle habitat, where tasks occur either within an inner circle, or on an outer circle. Here, the multi-agent approach may yield a lesser improvement because the tasks are more spread out and isolated. In particular, due to the spatial distribution of tasks, the cost associated with taking on new tasks may be higher, resulting in fewer re-allocations of tasks.

We note that the grid habitat environment represents a hybrid of the other ones, as tasks are allocated in clusters that are spread out over a square region. Thus, it makes sense that the improvement in makespan falls in between the other two environments.

Looking at Table 4.2, we see that on average, the single-agent approach tended to have better travel times per task. Looking at the individual environments, one can understand how the distribution of tasks results in different travel times. For the square setting, tasks are uniformly distributed, so any clusters of agents that may form are likely to be irregular, if any clusters do exist. Thus, it is likely that agents are not heavily constrained to particular regions of the habitat, enabling the multi-agent approach to yield more efficient re-allocations.

However, in the the remaining environments, it is much more possible for agents to be constrained to smaller, more regular regions of the map. Therefore, should an agent complete its schedule earlier than expected, it may take on a task that is far away, reducing the overall makespan but increasing the the average travel time per task. As an example, in the circle setting, an agent may be behind schedule and seek to re-allocate a task that is on the outer circle. Since most agents will be in the inner circle, there is a higher chance that re-allocating the task will increase the average travel time. Comparatively, in the square setting, agents are more likely to be evenly distributed, so there is a higher chance that a nearby agent can take on a re-auctioned task. Finally, the grid setting slightly favors the single-agent approach in regards to travel time. Taking into consideration our previous assumptions, this makes sense, as agents are likely to constrained into several clusters. Thus, in the case of a re-allocation auction, an agent from a nearby cluster will travel a shorter distance to execute a task, resulting in a reduced makespan, but slightly longer average travel time when compared to the single-agent approach.

Table 4.2: Average Travel Times per Task

	Multi-Agent	Single-Agent
Square	53.70	56.65
Circle	59.11	51.12
Grid	64.74	61.52

Table 4.3: Average Variance of Agent Makespans

	Multi-Agent	Single-Agent
Square	6720.16	9981.38
Circle	6044.29	8938.65
Grid	3372.24	10074.52

Lastly, Table 4.3 provides us with the average variance of agent makespans across the different environments. Intuitively, one might imagine the multi-agent approach to yield a smaller average, since it is able to adjust to the real-time execution feedback. And in fact, this is the case. Across all environments, the multi-agent approach tends to yield smaller average variances when compared to the single-agent approach.

While the initial allocations given by both approaches may be similar in their ability to evenly distribute tasks and lower the makespan, the real-time execution feedback makes these allocations mostly serve as a guide upon which to generate actual allocations. In particular, the delays that occur during execution may result in agents falling very behind schedule. In a single-agent case, there is only so much that one agent can do to recoup their losses. On the other hand, the multi-agent case is able to react to the different outcomes and adjust the overall schedule through a series of auctions and reverse-allocation auctions, ensuring that the individual makespans remain relatively comparable. Thus, we see that the multi-agent approach is much better at reacting to the dynamic environment and ensuring that all agents are utilized more equally.

Chapter 5

Discussion

In this work, we developed a multi-agent task allocation and execution monitoring system that allows robotic agents to autonomously maintain a deep-space habitat. Our findings demonstrate that the multi-agent approach outperforms the single-agent version in responding to dynamic changes in the environment and task execution, resulting in an average reduction in makespan of 8%. However, when considering metrics such as average travel time per task, the single agent approach generally yielded better results.

To better understand the suitability of each approach, it is essential to consider the specific characteristics and constraints of deep-space habitats. In situations where resources are scarce and agents have limited capabilities, the single-agent approach may be preferred. For instance, if agents have limited battery capacities and computational capabilities, allocating resources for communication with other agents and traveling longer distances due to task re-allocation might be suboptimal. Furthermore, if a habitat is very large and spread out, such as in the circle experiment setting, it may be better to utilize a single-agent approach, since having agents move across the map to execute the tasks of others may be a waste of resources.

On the other hand, as seen in our experiments, the multi-agent approach tends to be more effective in environments that are less spread out and more highly connected, in terms of agent-to-agent communication. Assuming that agents are also able to replenish their individual resources efficiently (e.g. quickly replenishing batteries via charging stations), the moderate sub-optimality in travel time is more than made up for by the overall reduction in makespan, leading to more optimal schedules. Furthermore, the multi-agent approach can be fine-tuned for different objective functions, where the individual agents can take into account resource constraints when expressing their bids. Thus, it is possible for the multi-agent approach to be modified to support the environment in which the system is deployed.

While our work demonstrates the usefulness of multi-agent auctions for online re-planning, the primary contribution lies in the incorporation of positive re-planning. Positive re-planning is particularly well-suited for the maintenance of deep-space habitats due to their isolation and the challenges posed by the unique and unpredictable space environment. As an example, one challenge is determining accurate durations for the various tasks that agents may execute throughout their lifespan. Over long periods of time, data can be collected regarding these tasks, and the planning system can use the data to generate task schedules that more closely represent reality. Due to the completeness of our re-planning approach, the system can effectively handle negative,

neutral, and positive outcomes of execution.

Thus, our research demonstrates that the multi-agent approach offers significant advantages over the single-agent one, especially when responding to dynamic changes in the environment. However, there are various considerations that one must make when choosing an approach, and it is important to fine-tune each version and ensure that it is best suited for the environment.

Chapter 6

Future Work

One limitation for this paper was that we only were concerned with optimizing for makespan when determining allocations. As demonstrated in our experiments, while this allowed the multi-agent approach to yield much better makespans across all environments, it suffered from having longer average travel times. While we technically accounted for travel time by incorporating travel tasks into our schedules, it is possible that a better objective function exists that limits travel time more. Furthermore, we were only truly concerned with the resource of time, however as previously mentioned, there may exist other resources that must be considered. Thus, understanding what resource constraints may exist in the deep-space domain and determining the appropriate objective function requires future work.

Another limitation for this paper was that the tasks were relatively rudimentary in design. Joint activities only required two agents, and there wasn't really a notion of inter-task dependencies besides the coordination and sequencing constraints. In a real environment, tasks may require competing resources, and there may be safety constraints that prevent agents from operating in close proximity each other. Furthermore, joint activities may require more than two agents, and a more efficient algorithm may be needed to determine how to combine bids.

Thus, one area of future work would be to equip each individual agent with a local planner, such as POPF. The agents would then be able to track their local state in regards to resources and other environmenal factors, and generate plans based off of their current state of execution and schedules. These would then be expressed via bids in the same auction process as we have proposed in this paper. Furthermore, it would be interesting to see if one could use the ideas proposed by Milot et al. to use a HTN planner to solve the Winner Determination Problem for the auction allocations [8].

Another area of future work would be to use current techniques in multi-agent planning in distributed environments to limit the amount of communication that an agent must undergo. In particular, the current approach publishes message on a global topic that is shared by all agents. However, this may be unnecessary, as far away agents are less likely able to share tasks due to the larger travel times. Thus, limiting agents to only communicate within a certain range would be beneficial in reducing the number of messages exchanged between agents.

A third area of future work would be to integrate this implementation with the rest of the ongoing work done by Research Thrust 4 of the NASA HOME project. This would be done by connecting the timeline-based HTN planner that is currently under development to each agent's

scheduling mechanism, and also by connecting the goal manager to a higher level habitat maintenance system.

Chapter 7

Conclusion

In this work, we attempted to develop a full framework for multi-agent task allocation for spatio-temporal tasks in a distributed, dynamic environment. In doing so, we combined previous research in multi-agent task assignment using auctions with novel work in opportunistic re-planning. Our approach utilized a goal manager and execution interface in combination with a series of agent nodes to dynamically process tasks to simulate their execution. Notably, we developed re-planning mechanisms that utilized auctions and reverse-allocation auctions to robustly handle their execution. In doing so, we found that the multi-agent approach yields better results across all environments, resulting in an overall reduction in makespan of 8%. However, we note the importance of understanding the characteristics of a particular domain, as the connectedness and level of coordination affect the degree to which the multi-agent approach is better. In fact, if certain objectives are more desirable, such as minimizing travel time, the single-agent approach can be better. Despite this, we still argue that the multi-agent approach is better based on our experiments, and should be used in combination with positive re-planning as this provides a complete framework for dealing with multi-agent planning and execution.

Bibliography

[1] Sofia Amador, Steven Okamoto, and Roie Zivan. Dynamic multi-agent task allocation with spatial and temporal constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014. 2

[2] Laura Barbulescu, Zachary B Rubinstein, Stephen F Smith, and Terry L Zimmerman. Distributed coordination of mobile agent teams: the advantage of planning ahead. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 1331–1338, 2010. 2

[3] Daniel Borrajo and Manuela Veloso. Computing opportunities to augment plans for novel replanning during execution. 2021. 2

[4] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, and Bram Ridder. Opportunistic planning in autonomous underwater missions. *IEEE Transactions on Automation Science and Engineering*, 15(2):519–530, 2017. 2

[5] Han-Lim Choi, Luc Brunet, and Jonathan P How. Consensus-based decentralized auctions for robust task allocation. *IEEE transactions on robotics*, 25(4):912–926, 2009. 2

[6] M Bernardine Dias, Robert Zlot, Nidhi Kalra, and Anthony Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7):1257–1270, 2006. 2

[7] Matthew Hoeing, Prithviraj Dasgupta, Plamen Petrov, and Stephen O'Hara. Auction-based multi-robot task allocation in comstar. In *Proceedings of the 6th international joint conference on autonomous agents and multiagent systems*, pages 1–8, 2007. 2

[8] Antoine Milot, Estelle Chauveau, Simon Lacroix, and Charles Lesire. Solving hierarchical auctions with htn planning. In *4th ICAPS workshop on Hierarchical Planning (HPlan)*, 2021. 2, 6

[9] Félix Quinton, Christophe Grand, and Charles Lesire. Market approaches to the multi-robot task allocation problem: a survey. *Journal of Intelligent & Robotic Systems*, 107(2): 29, 2023. 2

[10] Stephen F Smith, Zachary B Rubinstein, David Shur, and John Chapin. Robust allocation of rf device capacity for distributed spectrum functions. *Autonomous Agents and Multi-Agent Systems*, 31:469–492, 2017. 2