Plan Merging and Execution for Multirobot Coordination

Shivani Velapure[1], Christopher B. Nalty[2], Anisha Bontula[1]

Abstract— **The ability to replan efficiently is important for multirobot systems in dynamic, time-sensitive and mission critical domains. However, challenges arise when merging plans that involve conflict resolution and hierarchical tasks. In this paper, we address the issue of achieving both optimal makespan and low computational cost by proposing a novel plan merging strategy. We present the Selective Serial TCRA* approach that uses the Serial algorithm to selectively merge temporal orderings in individual agent plans prior to performing TCRA*, reducing the size of the search space. We conducted experiments simulating tightly-coupled problem files in the example setting of a first-response domain and evaluated the efficacy of our proposed algorithm with a comparative analysis. Our results showed that the Selective Serial TCRA* outperformed TCRA* with a smaller search space and in a majority of tests, resulted in faster run times. We further maintained the optimal makespan, indicating the scope of applying our proposed framework. Future implementations of plan merging can use these findings at a larger scale.**

I. Introduction

Planning for single agent systems is a well-researched domain, wherein the agent's properties and goals are informing factors in generating an efficient plan to reach the agent's destination. Replanning accounts for sudden changes in the states or goals that occur in real-world deployments, enabling the agent to accommodate these modifications. Planning and replanning however, become significantly more complex when multiple agents are introduced. Individual agents must now coordinate to complete their local goals, while optimizing the global results. Inherently, planning now must consider task allocation, conflict resolution, and coordination. In addition, the ability to replan is an important aspect of deploying multirobot systems. The introduction of the aforementioned new goals, unforeseen states, and unexpected agent actions with multirobot systems demand for the re-evaluation of objectives, and replanning to achieve the system's goals. In particular, the salient facet of replanning is to reduce the computational cost needed to create a conflict-free global plan by merging individual agents' plans.

Multirobot systems have to process new tasks dynamically, which may cause their original plan to become obsolete. In these cases, a new plan must be generated when there is a change in goal, state, or action of an agent in the system. Thus far, replanning and plan merging has broadly been studied on the basis of tasks and environments that are simple, loosely coupled, and of non-critical importance. Moreover, existing solutions assume apriori knowledge of new goals and introduce changes to the plan at informed junctions to adapt accordingly [1]. These approaches are non-generalizable and inapplicable to dynamic environments. Existing approaches to solving the problem of introducing a new task echo these limitations, focusing primarily on constraining assumptions and plan repair [2]. Other replanning methods that were previously presented cannot be scaled as they are restricted to forward state space searching, and consist of relatively simple tasks [3]–[5]. The limitations of the above approaches arise when dealing with multirobot systems that require merging individual agent plans and conflict resolution in time sensitive and dynamic domains.

The uncertainty that is introduced in complex, sensitive contexts, such as the first-response domain, make the subsystems of multirobot planning(i.e., task allocation, conflict resolution, and coordination) difficult to tackle. Our approach to plan merging is an extension of [6], wherein we focus specifically on multirobot replanning in the first-response domain. In [6], limitations with respect to existing plan merging algorithms in accordance with conflict resolution was addressed by proposing a new algorithm, TCRA*. Temporal Conflict Resolution A* (TCRA*) was designed to result in conflict free plans, such as those resulting from the Serial algorithm, while overcoming the impractical makespan of previous approaches. The resulting algorithm results in generation of temporal orderings of tasks within individual agents' plans, but is limited by the sub-optimal computational cost and run time.

In this paper, we introduce a new plan merging algorithm that leverages the temporal ordering of tasks from individual agents' plans to solve fewer conflicts, reducing the overall computational cost. Our approach is to create a hybrid algorithm (coined Selective Serial TCRA*) which merges the Serial Algorithm and the TCRA* to selectively serialize tasks before resolving conflicts. The serialization reduces the number of conflicts in the set of task plans before the merge occurs, enabling the algorithm to considerably reduce the run time while maintaining an optimal makespan.

Our hypotheses were tested by evaluating the search space, run time, and makespan of our algorithm against the baseline of TCRA*. We tested three problem files, simulated with the ActuPlan simulator [7] consisting of varied number and types of plans. Our results show that in all tests, Selective Serial TCRA* has a smaller search space than TCRA*. In ideal cases of selective serialization, we achieve a better run time as well. The makespan remained optimal and was not compromised, indicating the efficacy and scope of this approach. The primary contribution of this work is a new plan

[1] Collaborative Robotics and Intelligent Systems (CoRIS) Institute, Oregon State University, Corvallis, OR 97331, USA.
[2] Dynamic Robotics Laboratory, Oregon State University, Corvallis, OR 97331, USA.
{velapurs naltyc bontulaa}@oregonstate.edu

merging algorithm, Selective Serial TCRA*, that provides a conflict-free, scalable solution requiring low computation.

II. Background

Several real world applications involving multiagent systems require the ability to replan. Enabling a global plan to accommodate changes in the environment and changes in agents' profiles, is a well recognized problem since early work in multiagent systems [8]. Replanning however, has been conceptualized in previous work with several constraining assumptions. Furthermore, the introduction of plan repair as an alternative to replanning broadened this gap in understanding, and the overall applicability to highly dynamic domains. [1] presents an analysis evaluating stability of plan repair vs replanning. They experimentally conclude that plan repair is more stable, but is based on the key assumption of a stable environment where new goals or actions are known. Work presented in [2] serves as an extension, wherein the authors add search trees to the algorithm to explicitly define the actions that are a part of the repair. However, both approaches are limited by the overlying assumptions, and lack of clarity regarding the individual agents' profiles and plans. Replanning addresses some of these limitations with the inherent nature of involving conflict resolution and re-evaluation of both agent and system objectives, although limited by the time and computational power required.

A. Plan Merging

As a part of replanning, plan merging is key in applications and domains such as ours. Multirobot planning involves conflict resolution, task allocation and coordination between multiple individual robot plans, which enables the resulting global plan to be fairly robust.

1) Conflict Resolution: Foundational work in [9] defines planning as the abstract problem of making an agent reach a new state from its' current state. Multiagent planning therefore, innately involves plan merging, we now look at multiple plans of individual agents that must run concurrently. While [9] provides strong grounding, the algorithms discussed assume conflict-free plans. Work presented in [10] focuses on conflict-resolution of agents' goals, plans and beliefs, but define their agents and environment in a competitive setting. The difficulty lies in resolving conflicts and overcoming the classical categorization of agents as either competitive or grouped. In our problem setting, we consider individual agents that have multiple task-completion objectives and are not strictly competitive.

2) Task Allocation: Task allocation and priority recognition is elaborated on in [11]. The authors recognize that an important aspect to consider in plan merging is the ordering of actions within individual agent plans. They present a framework for including the hierarchy of tasks or actions that an agent must do while plan merging, essentially incorporating individual agent's objectives. Plans with temporal constraints are thoroughly discussed in [12], and provides essential guidance to including ordering in plan merging.

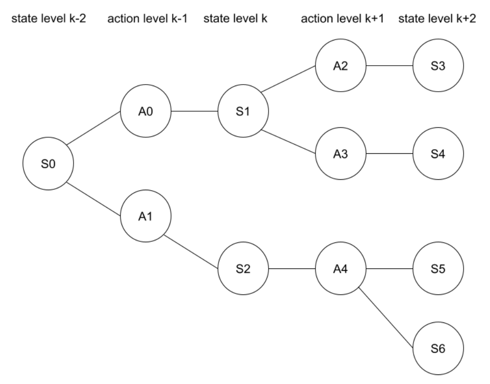

Fig. 1. In this figure is an example of a planning graph. The levels alternate between state nodes and action nodes. State nodes connect to possible action nodes to take in that state, and action nodes connect to possible resulting states from those actions.

Although the above approaches recognized deadlocks and addressed hierarchy, the computational time required was significantly higher with even a small increase in number of tasks.

3) Coordination: Graphing algorithms have significant backing in reducing makespan while coordinating, such as in [13]–[16]. The searching most often converges to A*, but is limited by scalability and state-spaces with fewer dimensions.

Many AI and multiagent applications require the agents to solve additional tasks while the current plan is being executed. The new tasks arrive asynchronously and are not known before arrival, hence strongly denying the assumption of apriori knowledge. To be the most efficient, the planner needs to incorporate the completion of the new task into the current plan without completely stopping current execution. This presents numerous challenges, estimating when the new plan's start state should be, accounting for the priority of the new task, minimizing the time to replan, and minimizing the makespan of the final plan. Here we will focus on minimizing the time to replan and the makespan of the final plan.

B. Planning Graphs

To elaborate, we succinctly describe the fundamental graphplans that our work is based on. As detailed in [3], [17], replanning is a special case of planning where there already exists some partial plan. Most modern planning techniques use Planning Graphs, introduced in [18]. A Planning graph is a directed, leveled graph constructed to encode the plan. Generally this graph will be made by a forward search with multiple possible paths of actions and states the agents could pursue [3], [18]. The graph alternates between state nodes and action nodes.

Graphplan [18] was the original algorithm that introduced Planning Graphs, and is what many state-of-the-art planning algorithms are based off. This algorithm consists of two stages, extending the Planning Graph and searching the Planning Graph for a valid plan. The Planning Graph begins with one level containing a node for each of the possible initial states. The nth level of the Planning Graph consists of all actions that are valid coming from the n-1 state level. the n+1 level contains the states that results from actions taken in level n. The graph is extended in this manner, alternating between actions that are valid, and resulting states. When a goal state is found, the plan can be searched backwards for a valid plan [18], or can be converted into a Constraint Satisfaction Problem (CSP) as in [19].

The subsequent graph planning algorithms that were proposed, such as A*, are difficult to scale and apply in scenarios that need to satisfy many conditions. The constraints of these algorithms prompted us to explore and expand on conflict-free solutions, thereby satisfying existing conditions while optimizing the merging.

```
Data: A plan $\pi_I$
Result: A conflict-free plan $\pi$
foreach plan $\pi_k$ in plan list $\pi$ do
    foreach action $a_i$ in plan $\pi_k$ do
        foreach action $a_j$ in plan $\pi_k + 1$, where plan $\pi_k + 1$ succeeds plan $\pi_k$ do
            Add temporal order $\prec_T = \langle a_i, a_j \rangle$ to plan $\pi$;
        end
    end
end
return conflict-free plan $\pi$;
```

Algorithm 1: Serial Algorithm

```
Data: A plan $\pi_I$
Result: A conflict-free plan $\pi$ or null
Add input plan $\pi_I$ to the queue;
while the queue is not empty do
    Pop plan $\pi$ from the queue;
    Identify conflicts K = $\{k_1, ...\} in\ \pi$;
    if there are conflicts in $\pi$ then
        foreach conflict $k \epsilon K$ do
            Identify Solutions $\Sigma = \{\sigma_1, ...\}$;
        end
        foreach solution $\sigma \epsilon \Sigma$ do
            Apply solution $\sigma$ to produce plan $\pi_\sigma$;
            Compute $f(\pi_{sigma}) = m(\pi_{sigma}) + \epsilon \dot{h}(\pi_{sigma})$;
            Enqueue plan $\pi_{sigma}$ with priority $f(\pi_{sigma})$;
        end
    else
        return conflict-free plan $\pi$;
    end
end
return null;
```

Algorithm 2: TCRA* Algorithm

III. Methods

We propose a new plan merging framework that combines the existing conflict-free solutions, i.e, Serial Algorithm and TCRA* [6]. To elaborate on our proposed framework, we will first provide a brief explanation of the respective algorithms that it comprises of:

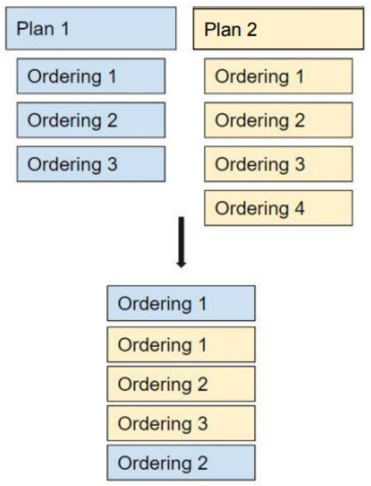

Fig. 2. A visual depiction of how elements of how plans are serialized. The first plan(L) is serialized with selective orderings from the second plan(R). The resulting plans are then merged with TCRA*

A. Temporal Conflict Resolution A*

The Temporal Optimal Conflict Resolution Algorithm (TCRA*) [6], is an algorithm for solving Temporal Multiagent Plan Coordination Problems (TMPCP). In a TMPCP, a plan π is a tuple $\langle A, \prec_T, \prec_C \rangle$, where A is a set of actions, $\langle a_1, a_2, ... \rangle$, and $\prec_T$ and $\prec_C$ are a set of temporal and casual orders over the actions, A. Temporal orders are a simple tuple of two actions in A, $\langle a_i, a_j \rangle$ where a_i must proceed a_j in the plan. A casual order, $\langle a_i, a_j$, c$\rangle$ is an extension of a temporal order, with the addition of a condition, c. c must arise from the effects of taking action a_i and is also a precondition to the action a_j. The TMPCP is represented by a tuple, $\langle$I, G, $\pi_I \rangle$, where I is the set of initial conditions, G is the set of goal conditions, an π_I is the initial plan consisting of a list, $\Pi = \langle \pi_1, ..., \pi_m \rangle$ for m multiagent plans. The Serial Algorithm, shown in algorithm 1, introduced by [6] as a baseline that merges each agents plan in a serial fashion.

The TCRA* algorithm bases off of the A* algorithm to guarantee completeness of the plan and to reduce the makespan [6]. It combines uniform-cost search and an admissible search heuristic $h(\pi)$ to minimize the makespan $m(\pi)$ or run time. The priority of a plan, $f(\pi)$, is the combination of $m(\pi)$ and $h(\pi)$ [6]. The full algorithm is shown in 2.

B. Serial Algorithm

The Serial Algorithm, a TMPCP solution, is an algorithm that introduces temporal orders in order to serialize the multiple robot plan execution. The Serial Algorithm strictly orders each of the successive plan's actions before the previous plan's actions, resulting in a conflict free plan. A drawback of this algorithm is that the global plan generated has a large makespan as none of the actions are executed concurrently. The complete algorithm is depicted in 1.

Both the above methods result in conflict-free solutions, but are limited by their computational costs. Our approach is to resolve this issue by taking advantage of the guaranteed conflict-free solutions from the Serial algorithm, and the optimal makespan from the temporal ordering of TCRA*. We explored this integration in two ways before formulating our final contribution, listed below:

- Naive Serial-TCRA*: In this method, we randomized the plan merging to be merged with either Serial or TCRA* for plans in the problem file, as shown in 3.
- Naive Parallel-TCRA*: In this method, the randomized selection of the plan merging algorithms ran concurrently, as shown in 4.
- **Selective Serial TCRA*:** The key method we aimed to explore is the selective serialization. Here, we selectively serialize parts of individual robot plans', that are subsequently merged with TCRA*.

To elaborate the working of selective serialization, we will provide a descriptive illustration with temporal orderings of plans A and B. Selectively serializing plan A with plan B implies that a fixed number of orderings from plan B will be added to plan A. When plan A and B are merged, the now included orderings from plan B allow for fewer conflicts to arise, and consequently fewer solutions to parse through. This process repeats when there are more plans, i.e., plan A serialized with plan B, plan B serialized with plan C, etc.

To determine the method of selecting elements that will be serialized, we decided to leverage the known temporal orderings in individual agents' plans. We chose an arbitrary ratio of 3:1 for serializing, i.e, 3 orderings from plan B will be serialized per 1 ordering of plan A. The orderings that were serialized were not repeated, and continued till all orderings from the second plan were added to the initial. An example of the selective serialization is depicted visually in Figure 2. This ratio was chosen from observing the input problem files at hand, as it ensured a consistent amount of serialization across the files.

*Modifications to Selective Serial TCRA**: However, we do acknowledge that the chosen ratio may be unrepresentative of the framework's performance, and evaluated ratios of 2:1, 4:1, and 5:1 serializing. We aimed to gain an understanding of the potential effects of "over-serialzing" or "under-serialzing" via this approach. The problem files we generated as inputs had plans of vastly varying sizes, inferring that some of these modifications could not be executed. Therefore, the modifications were tested only on one problem file that had plans of similar size. Due to this limited testing, our hypotheses are based only on the 3:1 ratio. The experimental results of all modifications are presented in Section V

C. Hypotheses

Our hypotheses are grounded in results of the past work we expanded upon [6]:

H1: The Naive Serial-TCRA* and Naive Parallel-TCRA* algorithms will not improve computational cost and makespan when compared to TCRA*.

H2: The Selective Serial TCRA* will improve computational cost and time when compared to TCRA*

H3: The Selective Serial TCRA* will not have a deteriorated makespan when compared to TCRA*.

The procedure of our testing and analysis are elaborated in Section IV.

```
Data: A plan $\Pi = \langle \pi_1, ..., \pi_m \rangle$ containing m robot
  plans, and and integer n, n $<$ m
Result: A conflict-free plan $\pi$ or null
plan groups = robot plans evenly split into n groups
merged plans = []
foreach plan group in plan groups do
    $\pi_{curr}$ = Serial(plan group) add $\pi_{curr}$ to merged
      plans
end
return TCRA*(merged plans)
```

Algorithm 3: Naive Serial TCRA*

```
Data: A plan $\Pi = \langle \pi_1, ..., \pi_m \rangle$ containing m robot
  plans
Result: A conflict-free plan $\pi$ or null
Add input plan $\pi_I$ to the shared queue;
Initialize Lock $l_q$
In parallel on all threads
while TerminateDetection() do
    if SharedQueue = $\emptyset$ then
        continue;
    end
    AcquireLock($l_q$);
    Pop plan $\pi$ from the queue;
    ReleaseLock($l_q$);
    Identify conflicts K = $\{k_1, ...\} in\ \pi$;
    if there are conflicts in $\pi$ then
        foreach conflict $k \epsilon K$ do
            Identify Solutions $\Sigma = \{\sigma_1, ...\}$;
        end
        foreach solution $\sigma \epsilon \Sigma$ do
            Apply solution $\sigma$ to produce plan $\pi_\sigma$;
            Compute
              $f(\pi_{sigma}) = m(\pi_{sigma}) + \epsilon \dot{h}(\pi_{sigma})$;
            Enqueue plan $\pi_{sigma}$ with priority
              $f(\pi_{sigma})$;
        end
    else
        return conflict-free plan $\pi$;
    end
end
return null;
```

Algorithm 4: Naive Parallel TCRA*

IV. Experiments

The Coalition Formation then Planning Framework [20] is used to allocate tasks to coalitions, create plans to

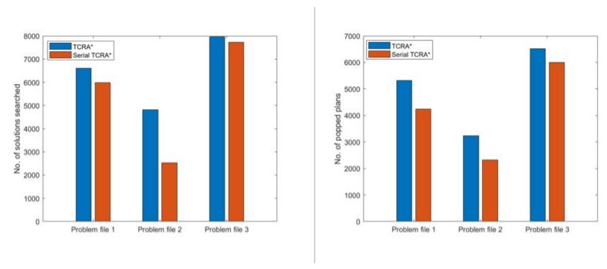

Fig. 3. Graph showing the comparison of solution search space between TCRA* and Selective Serial TCRA* via no. of solutions searched, and no. of plans popped from queue respectively.

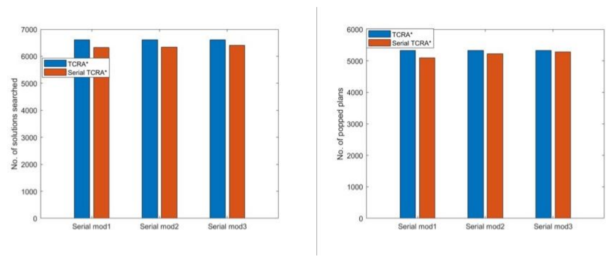

Fig. 5. Graph showing the comparison of solution search space between TCRA* and mods of Selective Serial TCRA* via no. of solutions searched, and no. of plans popped from queue respectively. The results shown are for Problem file 1.

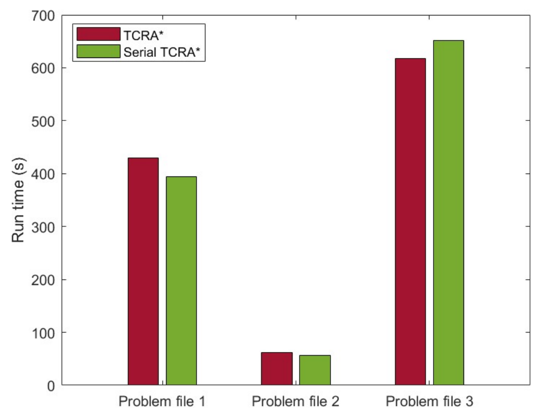

Fig. 4. Graph showing the comparison of run time between TCRA* and Selective Serial TCRA* for all problem files.

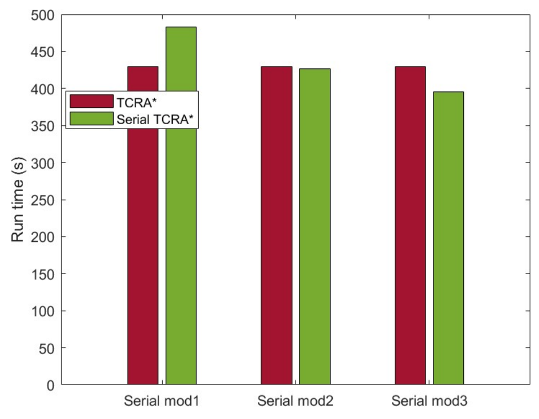

Fig. 6. Graph showing the comparison of run time between TCRA* and mods of Selective Serial TCRA* for Problem file 1.

solve each task and merge individual task plans. The first response domain [20] consists of five types of tasks- victim support, clearing blocked roads, sealing gas leaks, securing pharmacies and securing pawnshops. The number of robots, victims, pawn shops, pharmacies, road blocks, gas leaks, and waypoints was drawn from a uniform distribution. It is assumed that the action outcomes are deterministic and the action duration times are stochastic. Although, the tasks are loosely coupled, plans for separate tasks can become coupled when sharing the same robots. Additionally, the coupling between two tasks is made stronger when there is overlap between the locations the robots must traverse, due to the shared effects of cleared blocked roads.

Tasks are allocated to robots according to the robots' capabilities and the tasks' requirements using a dynamic programming coalition formation algorithm [21]. The Actions Concurrency and Time Uncertainty Planner (ActuPlan) [7] creates plans to solve each task. Each task plan consists of a list of actions that need be executed to complete the task, and the temporal orderings between those actions. The role of the plan merging algorithm is to resolve conflicts between these task plans to generate a conflict-free global plan. Two types of conflicts can exist between task plans- Open preconditions and Causal conflicts. Open preconditions arise when an action's preconditions are not satisfied. A causal conflict occurs if there exists an action a that negates the condition c needed to execute a causal order $\langle a_i, a_j \rangle$ such that a is not ordered away from the ordering. TCRA* is used as the baseline plan merging algorithm for our experiments.

The Naive Serial-TCRA* and Naive Parallel-TCRA* algorithms were tested on a problem file consisting of 4 task plans. Selective Serial-TCRA* was tested with three problem files, consisting of 2-3 plans each. A ratio of 3:1 was used for serializing the tasks. Each task in the first plan was serialized with three tasks in the second plan, and so on. To evaluate the effect of the degree of serialization, experiments were performed by merging plans with varying serialization ratios of 2:1, 4:1 and 5:1. The simulated experiments were performed using the ActuPlan simulator [7], a high-level plan simulator. To evaluate the performance, we used the following metrics: solution search space, run time, and makespan. TCRA* outputs the number of popped plans and number of solutions searched. The number of popped plans give us an estimate of how many conflicts were resolved. The number of plans visited gives us an estimate of the size of the solution space searched. ActuPlan simulator calculates the makespan of the global plan. The run time of the algorithms was evaluated by using the time module in python.

V. Results

The results of our experiments and the heuristic analysis are presented below. We found that for Naive Serial-TCRA* and Naive Parallel-TCRA*, both run time and makespan was much greater than TCRA*. Since the solution search space

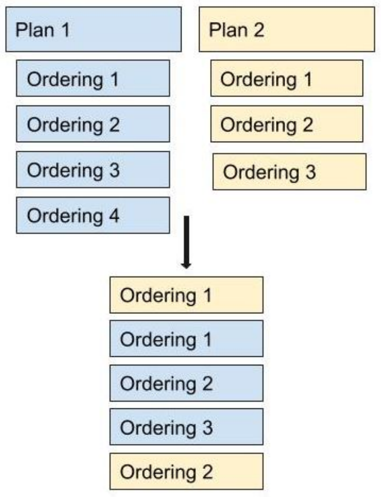

Fig. 7. A visual depiction of ”flipped” serialization. The second plan(R) is serialized with selective orderings from the first plan(L). The resulting plans would then be merged with TCRA*

was not explicitly manipulated, no changes were observed.

Selective Serial TCRA* however, had favorable results. We observed a reduction in solution search space as shown in Figures 3and 5. This result was replicated in all instances of Serial TCRA*, including all problem files and modifications. Interestingly for search space, the serialization ratio of 3:1 outperformed all modifications. To echo these findings, we checked the number of conflicts that arose by observing the number of popped plans. Figures 3 shows that along with fewer solutions searched, fewer conflicts also arose, pointing towards promising findings. For the serialization ratio of 3:1, we observed a reduction in run time for problem files 1 and 2, but an increase in run time for problem file 3, as shown in Figure 4. Depicted in Figure 6, for modified ratios of 4:1 and 5:1, we observed an even faster run time for problem file 1, but a longer run time for the modified 2:1 ratio. The makespan of all instances of Selective Serial TCRA* was the same as TCRA*, showing that the optimal result from [6] was not compromised.

VI. Discussion and Conclusions

In this paper, we present our approach to formulating, and the results of a new plan merging algorithm that leverages benefits of the Serial and TCRA* algorithms. The Serial algorithm is mainly beneficial due to it’s inherent nature of being conflict-free, as no conflicts arise to begin with. TCRA* solves conflicts with temporal orderings, but takes a significant amount of time and searches over a large solution space that can be reduced. First, we discuss the implications of combining the Serial and TCRA* algorithm in different ways. Experimental results fully support **H1**, showing that introducing the Serial algorithm in any form to plan merge, results in sub-optimal run time and poor makespan. Incorporating the benefits of Serial algorithm, must therefore be done in an efficient, selective manner.

Experimental results partially support **H2**, with a reduction in search space resulting in an overall lower computational cost. Run time however, was inconsistent across problem files; Problem file 3 had an increased run time with Selective Serial TCRA*. Problem file 3 consisted of three plans, with vastly varying sizes. From the serialization, the size of the previously smaller plan noticeably increased, which may have led to the longer run time. Furthermore, modifications of selective serializing all resulted in smaller solution search spaces. This indicates that while all amounts of serializing may not be beneficial, there exists a range, especially for smaller problem files, wherein the amount of serialization is beneficial regardless. However for search space, our initial selection of the ratio 3:1 performed better than all modification. There is therefore an extended implication that while most serialization is good, some serialization is better. The run times interestingly, were different for all three modifications. Ratios 4:1 and 5:1 resulted in lower run times than TCRA*, but the difference with the serialization of 4:1 was almost unnoticeable and not very compelling. The 5:1 serialization however had the most similar run time to 3:1, while 2:1 did not perform better than TCRA*. The above results further indicate that future work must be done to understand why and how these drastic differences occur, and how to optimize the solutions.

Finally, **H3** was fully supported, as makespan remained the same in all instances of selective serialization.

Strengths of this work include demonstrating that non-complex changes can be introduced to an intricate domain, without compromising the optimal makespan, and satisfying the domain property of being task-agnostic. Selective Serial TCRA* resulted in lower computational cost, and was replicated across assorted modifications, showing strong potential for future implementations of this framework.

There are several *limitations* to this approach. One key limitation is the problem files used in our simulations. While the problem files resulted in spanning over search spaces with thousands of solutions, they are still under-representative of how the algorithm would perform at scale. The second limitation is the method of selectively serializing. Our approach to the selection was based strongly on knowledge and observations of the problem files. However, the elements of plans could have commonalities in preconditions that are required to execute. Serializing based on these commonalities would further reduce the number of conflicts, and significantly improve the quality of the final solution. Finally, in this work we do not explore the effects of ”flipped” serializing, wherein the vice-versa, i.e., plan B serialized with plan A, occurs. A visualization of the working is provided in Figure 7. In the problem files, the varied sizes of plans showed that perhaps the ”flipped” serializing would always result in longer computational times, and was hence not explored. However, effects on solution search space and makespan remain unknown.

In the future, we hope to conduct experiments with larger and more complex problem files to evaluate the effect on makespan. Furthermore, we aim to incorporate a more informed method of selective serialization, to create a highly generalizable and holistic plan merging approach.

References

[1] M. Fox, A. Gerevini, D. Long, and I. Serina, "Plan stability: Replanning versus plan repair." in *ICAPs*, vol. 6, 2006, pp. 212–221.

[2] R. Van Der Krogt and M. De Weerdt, "Plan repair as an extension of planning." in *ICAPS*, vol. 5, 2005, pp. 161–170.

[3] J. Zhang, X. Nguyen, and R. Kowalczyk, "Graph-based multi-agent replanning algorithm," 01 2007, p. 122.

[4] N. Luis, S. Fernández, and D. Borrajo, "Plan merging by reuse for multi-agent planning," *Applied Intelligence*, vol. 50, no. 2, pp. 365–396, 2020.

[5] A. Dionne, J. Thayer, and W. Ruml, "Deadline-aware search using on-line measures of behavior," in *International Symposium on Combinatorial Search*, vol. 2, no. 1, 2011.

[6] G. Marcon dos Santos and J. Adams, "Optimal temporal plan merging," in *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2020.

[7] E. Beaudry, F. Kabanza, and F. Michaud, "Using a classical forward search to solve temporal planning problems under uncertainty," in *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[8] M. E. Bratman, D. J. Israel, and M. E. Pollack, "Plans and resource-bounded practical reasoning," *Computational intelligence*, vol. 4, no. 3, pp. 349–355, 1988.

[9] M. M. De Weerdt, "Plan merging in multi-agent systems," 2003.

[10] K. S. Barber, T.-H. Liu, and S. Ramaswamy, "Conflict detection during plan integration for multi-agent systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 31, no. 4, pp. 616–628, 2001.

[11] F. Gravot and R. Alami, "An extension of the plan-merging paradigm for multi-robot coordination," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, vol. 3. IEEE, 2001, pp. 2929–2934.

[12] I. Tsamardinos, M. E. Pollack, and J. F. Horty, "Merging plans with quantitative temporal constraints, temporally extended actions, and conditional branches." in *AIPS*, 2000, pp. 264–272.

[13] E. Burns, W. Ruml, and M. B. Do, "Heuristic search when time matters," *Journal of Artificial Intelligence Research*, vol. 47, pp. 697–740, 2013.

[14] Z. Zhang and Z. Zhao, "A multiple mobile robots path planning algorithm based on a-star and dijkstra algorithm," *International Journal of Smart Home*, vol. 8, no. 3, pp. 75–86, 2014.

[15] J. F. Zhang, X. T. Nguyen, and R. Kowalczyk, "Graph-based multiagent replanning algorithm," in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, 2007, pp. 1–8.

[16] W. Cushing, J. Benton, and S. Kambhampati, "Replanning as a deliberative re-selection of objectives," *Arizona State University CSE Department TR*, 2008.

[17] R. Krogt and M. Weerdt, "Plan repair as an extension of planning." 01 2005, pp. 161–170.

[18] A. Blum and M. L. Furst, "Fast planning through planning graph analysis," in *IJCAI*, 1995.

[19] M. Do and S. Kambhampati, "Solving planning-graph by compiling it into csp." 01 2000, pp. 82–91.

[20] G. Marcon dos Santos, "Coordination for scalable multiple robot planning under temporal uncertainty," Ph.D. dissertation, Oregon State University, 2020.

[21] L. Vig and J. A. Adams, "Multi-robot coalition formation," *IEEE transactions on robotics*, vol. 22, no. 4, pp. 637–649, 2006.

	Anisha	Shivani	Chris
Technical solution	33.3	33.3	33.3
Coding	33.3	33.3	33.3
Writing	33.3	33.3	33.3

TABLE I

CONTRIBUTIONS