Optimal Temporal Plan Merging

Gilberto Marcon dos Santos
Collaborative Robotics and Intelligent Systems (CoRIS)
Institute, Oregon State University
marcondg@oregonstate.edu

Julie A. Adams
Collaborative Robotics and Intelligent Systems (CoRIS)
Institute, Oregon State University
julie.a.adams@oregonstate.edu

ABSTRACT

Agents can individually devise plans and coordinate to achieve common goals. Methods exist to factor planning problems into separate tasks and distribute the plan synthesis process, while reducing the overall planning complexity. However, merging distributedly generated plans becomes computationally costly when task plans are tightly coupled, and conflicts arise due to dependencies between plan actions. Existing methods either scale poorly as the number of agents and tasks increases, or do not minimize makespan, the overall time necessary to execute all tasks. A new algorithm, the *Temporal Optimal Conflict Resolution Algorithm* (TCRA*), is introduced to merge independently generated plans and optimally minimize the resulting makespan. A proof of optimality is provided and the algorithm is empirically evaluated across two heterogeneous multiagent domains against two baseline algorithms. The TCRA* results in better makespan across the problems solved, and a search relaxation constant allows the TCRA* to generate better plans with competitive processing time and memory usage.

KEYWORDS

multiagent planning, plan merging, coalition formation

ACM Reference Format:
Gilberto Marcon dos Santos and Julie A. Adams. 2020. Optimal Temporal Plan Merging. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 9 pages.

1 INTRODUCTION

Multiagent planning allows agents to devise individual plans and coordinate to achieve common goals [12]. Multiagent plan synthesis distributes the planning process in order to reduce the computational cost and scale to larger problems [25]. Parallel plan execution allows agents to execute different parts of a plan and concurrently achieve shared goals. Parallel plan execution is required by physically embodied agent systems, such as multi-robot systems [1].

Multiagent plan synthesis factors the planning process across agents, decomposing problems into tasks before planning, and generates separate plans for each task [25]. However, real world tasks often are tightly coupled and tasks' plans conflict with each other. Solutions often assume loose coupling between tasks or serial plan execution to simplify the coordination process [25].

Tightly coupled tasks require plan merging methods to prevent conflicts during parallel plan execution [7]. Conflicts arise when the actions of one plan modify the environment and prevent another plans' actions from succeeding. Plan merging algorithms coordinate independently generated plans and eliminate conflicts by introducing temporal orders and removing redundant actions [7]. Algorithms minimize the number of plan actions optimally, but do not account for action durations and do not minimize makespan directly, the time required to execute the plan [13].

Accounting for durative actions can minimize makespan significantly, especially in real-world domains, where different actions require different completion times. The Logistics Domain [25] requires different vehicle types to deliver goods across their designated areas. Vehicle types are limited to subsets of an interconnected transportation network. Goods must be exchanged between vehicle types before reaching their destinations, requiring coordination. Real-world domains, such as Logistics, are physically distributed and require parallel plan execution to accomplish tasks concurrently. Tasks that share the same vehicles are coupled, as the shared vehicles' actions have mutual execution order constraints (e.g., a vehicle cannot deliver to two independent locations simultaneously). Durative actions are critical, as the travel time between locations varies significantly, directly impacting the makespan.

This manuscript presents a new problem formulation and an algorithm for merging loosely and tightly coupled task plans, while accounting for durative actions, and minimizing makespan for parallel execution. The algorithm can facilitate multiagent coordination and generate plans to solve complex heterogeneous multiagent problems efficiently. A proof of optimality is provided and the algorithm is empirically compared to baselines across two domains.

2 BACKGROUND AND RELATED WORK

Multiagent planning algorithms factor the problem and coordinate plans in order to improve scalability and solve more complex problems for larger numbers of agents [25]. Agents coordinate before planning, dividing the goal requirements and allocating tasks, generate task plans independently, and coordinate after planning, by merging task plans and resolving conflicts [9].

Agents are allocated to tasks according to their individual capabilities and plans are generated for each task [25]. Problem decomposition and task allocation can be performed using coordination graphs [15, 19], interaction graphs [5, 14], and coalition formation [11]. Tasks' plans are generated serially, where the planner assumes that its task's initial state is the final state from the prior task's plan [10]. The task goals are concatenated with the goals of the next task in order to guarantee that the next task's planner will not undo the goals achieved by the previous planner. Serial plan synthesis does not require serial plan execution, and the serially synthesized plans can be merged for parallel execution.

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand. © 2020 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

Serial plan synthesis addresses tightly coupled domains [25]; however, merging the independently generated plans can be a complex problem. Conflicts arise when the plan's actions can prevent the execution of another plan's actions. Most planners assume serial plan execution in order to simplify the merging process and prevent conflicts between plans. Plans are executed serially, in the same order as they were synthesized; however, serial plan execution prevents tasks from being executed in parallel, and results in long plan execution durations. Plan merging algorithms allow parallel task execution and can shorten the plan execution duration [7].

The Coalition Formation and Planning framework [11] employs coalition formation [23] to allocate tasks. Coalition formation algorithms derive agent teams, based on agents' capabilities and task requirements. A plan is generated for each task, using serial plan synthesis. However, the framework employs a greedy merging algorithm that can result in long plan execution.

A temporal plan merge algorithm distributes the plan synthesis before merging task plans, but the resulting plan is executed by a single agent [18]. The Task Coordination and Decomposition algorithm employs coordination before planning and imposes ordering constraints on tasks allocated to each agent, so that planning can be accomplished independently [24]. The method ignores the fact that one agent's plan can modify the environment and make another agent's plan invalid, limiting the algorithm to uncoupled tasks.

The Multi-Agent Planning by Plan Reuse algorithm performs task allocation before planning using relaxed reachability analysis after generating relaxed plans for all agent-task combinations [4]. However, the method requires homogeneous agents [4]. The Forward Multiagent Planning algorithm supports heterogeneous agents, but cannot scale due to communication limitations [26].

CSP+Planing introduces the agent interaction digraph in order to estimate task coupling [5]. Task coupling is defined as the level of interaction between agents, where tighter coupling leads to higher computational complexity. The algorithm seeks to minimize the coupling when allocating tasks in order to improve planning complexity. Constraint satisfaction factors the problem before planning; however, solving the constraint satisfaction problem dominates the plan synthesis computation, rendering the algorithm inefficient [8].

The Multiagent Plan Coordination Problem (MPCP) formulation assumes a set of agents derived plans independently to achieve their individual goals [7]. Agents can undermine each other when executing their plans and must coordinate to produce conflict-free plans that guarantee all agents achieve their individual goals. The solution to an MPCP is composed entirely of actions drawn from the original agents' plans; no actions are added. A plan π can be defined by a tuple $\langle A, \prec_T, \prec_C \rangle$, where $A = \{a_1, a_2, \ldots\}$ is a set of actions, $\prec_T$ and $\prec_C$ are sets of temporal and causal orders on actions A, respectively. Actions are defined by the tuple $\langle pre, eff \rangle$, where pre is a set of preconditions that must hold during action execution, and eff is the set of effects caused by the action execution. A temporal order, $\prec_T$, is a tuple $\langle a_i, a_j \rangle$, with a_i, $a_j \in A$, and establishes that action a_j cannot be executed before action a_i's execution is completed. A causal order is an extended temporal order, $\prec_C$, represented by the tuple $\langle a_i, a_j, c \rangle$, with a_i, $a_j \in A$, and c is a condition belonging to action a_i's effects ($c \in a_i.eff$) and to action a_j's preconditions ($c \in a_j.pre$). Causal orders help identify plan conflicts, but do not impact the plan execution, and are reduced to regular temporal orders after the merging process completes.

Two types of conflicts arise when merging coupled plans [7]. Open preconditions occur when a precondition c of an action a_j is not satisfied by the existence of a causal order $\prec_C = \langle a_i, a_j, c \rangle$, indicating that action a_i establishes condition c to satisfy action a_j's precondition. Causal conflicts occur when a causal order $\prec_C = \langle a_i, a_j, c \rangle$ is threatened by the effects of an action a. The causal order $\prec_C$ is threatened if there exists an action a whose effects establish the negative condition $\neg c$ and neither of the temporal orders $\langle a, a_i \rangle$ and $\langle a_j, a \rangle$ exist. A causal conflict implies that action a's effects can deny condition c and prevent action a_j to execute.

Open preconditions and causal conflicts are addressed by introducing causal and temporal orders, respectively. An open precondition of action a and condition c can be resolved by adding a causal order $\prec_T = \langle a_i, a, c \rangle$, such that condition c exists in the effects of action a_i, $c \in a_i.eff$ [7]. The solution forces action a_i to establish the condition c for action a. A causal conflict of action a and causal order $\prec_C = \langle a_i, a_j, c \rangle$ can be resolved by adding a temporal order $\langle a, a_i \rangle$ or $\langle a_j, a \rangle$ [7]. The solution forces the threatening action a to execute before action a_i, or after action a_j.

MPCP is NP-Complete and a tractable optimal algorithm is infeasible [7]. The Multiagent Plan Coordination by Plan Modification Algorithm (PMA) solves MPCP problems and minimizes the resulting number of actions [7]. The PMA allows a set of actions to replace a single redundant action collectively, resulting in merged plans with fewer actions. However, the PMA cannot scale to large numbers of agents executing complex and tightly coupled tasks [7].

The Solution Test Algorithm (STA) [7], a component of the PMA, solves plan conflicts iteratively and adds temporal and causal orders to derive a conflict-free plan. The STA, presented in Algorithm 1, was reformulated as a standalone merging algorithm. Each conflict resolved generates a new plan, which is added to a priority search queue for further refinement, and the first conflict-free plan encountered is returned. The STA uses the most-constrained conflicts first heuristic [20] and orders the priority queue using the number of conflict solutions [7]. Cyclical plans are discarded and a null plan is derived when the conflicts encountered cannot be resolved. Serial plan synthesis guarantees that the conflicts between plans can be resolved, and prevents the derivation of a null plan.

The STA scales better than the PMA, but does not minimize makespan [7]. The resulting plan's makespan determines the time taken to accomplish a plan's goals, a key factor in real-world problems. The reductions in plan execution time can make the difference between mission success or failure. This manuscript presents an algorithm that merges independently generated plans and optimally minimizes the resulting makespan, while accounting for durative actions. The following Section introduces a problem formulation that incorporates durative actions and the new algorithm.

3 TEMPORAL PLAN COORDINATION

The MPCP is extended to incorporate durative actions and minimize temporal makespan. The Temporal MPCP (TMPCP) is a tuple $\langle I, G, \pi_I \rangle$, where $I = \{i_1, \ldots\}$ is the set of initial conditions, $G = \{g_1, \ldots\}$ is the set of goal conditions, and π_I is the initial plan, consisted of a list $\Pi = \langle \pi_1, \ldots, \pi_m \rangle$ of m multiagent plans. TMPCP

```
Data: A (conflicted) plan $\pi_I$, consisted of a list of plans $\Pi$;
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 $K = \{\kappa_1, \ldots\}$ in plan $\pi$;
    if there are conflicts in plan $\pi$ then
        foreach conflict $\kappa \in K$ do
            Identify solutions $\Sigma = \{\sigma_1, \ldots\}$ to conflict $\kappa$;
            foreach solution $\sigma \in \Sigma$ do
                Apply solution $\sigma$ to produce plan $\sigma(\pi) = \pi_\sigma$;
                Compute priority $f(\pi_\sigma) = |\Sigma|$, the number
                  of solutions to conflict $\kappa$;
                Enqueue plan $\pi_\sigma$ with priority $f(\pi_\sigma)$;
    else return conflict-free plan $\pi$ ;
return null;
```

Algorithm 1: The Solution Test Algorithm (STA), extracted and reformulated from the Cox and Durfee's PMA [7].

plans' actions are defined by the tuple $\langle pre, eff, dur \rangle$, where pre is a set of preconditions that must hold during action execution, eff is the set of effects caused by the action execution, and dur is the action execution duration, the amount of time necessary to execute the action. A solution to a TMPCP is a conflict-free plan, π, that satisfies all goal conditions in G when executed from the set of initial conditions, I. All the actions of a solution plan π are drawn from the input plans, Π. No actions are added or removed.

A plan's path is a series of actions chained by temporal orders, $P = \langle \prec_{T1}, \ldots \rangle$. The length of a path, $l(P)$, is the sum of the temporal orders' action durations, $l(P) = \sum_{\prec_T \in P} \prec_T .a_i.dur$. The makespan, $m(\pi)$, is the length of π's longest path, $m(\pi) = \max_{P \in \pi} l(P)$, and represents the estimated time necessary to execute all of plan π's actions. An optimal solution plan, π^*, results in a makespan $m(\pi^*)$ less than or equal to the makespan of any conflict-free plan π generated by n successive conflict resolution iterations applied to the initial plan, $m(\pi^*) \le m(\pi)\ \forall\ \pi \in \sigma^n(\pi_I)$.

A Logistics example involves an autonomous semi-tractor-trailer truck (a), a manned truck (m), and two delivery tasks. Current autonomous semi-trucks are restricted to interstate highways and are not permitted in urban traffic [22]. The trailers are transferred to manned trucks before entering cities. Trailers 1 and 2 must be transported from a factory, via a highway, to an urban warehouse. Trucks can only transport one trailer at a time and exchange trailers at a transfer hub outside the city. Plans are generated individually for each delivery task, as shown in Figure 1. The plan for task 1 results in truck a bringing trailer 1 from the factory to the hub in 3 hours (1^a, 3h), followed by truck m moving trailer 1 to the warehouse (1^m, 1h). The plan for task 2 involves truck m returning to the hub (2^m_i, 1h), while truck a transports trailer 2 (2^a, 3h), followed by truck m moving trailer 2 to the warehouse (2^m_{ii}, 1h). The plans can be merged to accomplish both tasks.

The *Serial Algorithm*, Algorithm 2, a baseline TMPCP Solution, introduces temporal orders in order to strictly enforce the serial execution of the input plans Π. The Serial Algorithm assumes the input plans were synthesized serially, and can be merged into a

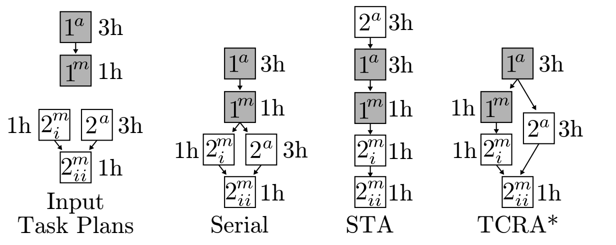

Figure 1: Illustrative Logistics problem and results for Serial, STA, and TCRA*. Task 1 actions shaded in gray.

conflict-free plan when executed serially. The Serial Algorithm strictly orders plan 2's actions after plan 1's, as shown in Figure 1, resulting in a 8 hour makespan. A second baseline for TMPCP is the *STA* (Algorithm 1). The STA can return a result rapidly due to its constraint satisfaction search, but does not necessarily minimize the resulting plans' makespan, making it suboptimal. STA results in truck a serially fetching both trailers to the hub, followed by truck m serially delivering both trailers, resulting in a 9 hour makespan.

```
Data: A (conflicted) plan $\pi_I$, consisted of a list of plans $\Pi$;
Result: A conflict-free plan $\pi$;
foreach plan $\pi_k$ in plan list $\Pi$ do
    Add temporal orders $\prec_T = \langle a_i, a_j \rangle$ between each action
      of plan $\pi_k$, $a_i$, and each action of plan $\pi_{k+1}$, $a_j$, where
      plan $\pi_{k+1}$ succeeds plan $\pi_k$ in the input plan list $\Pi$ ;
```

Algorithm 2: The Serial baseline algorithm.

The *Temporal Optimal Conflict Resolution Algorithm* (TCRA*), employs an A^* search to guarantee completeness and minimize makespan, as presented in Algorithm 3. The TCRA* is similar to the STA, in that both algorithms search the space of plans in order to solve conflicts iteratively and result in a conflict-free plan. STA and TCRA* have worst case complexity $O((|K| \cdot |\Sigma|)^n)$, where $|K|$ is the number of conflicts resolved per iteration, $|\Sigma|$ is the number of solutions to each conflict, and n is number of successive conflicts resolved on the resulting conflict-free plan π. The TCRA* performs a uniform-cost search and employs an admissible heuristic to minimize makespan and guarantee optimality. The priority of a plan π, $f(\pi)$, is defined by the plan's makespan, $m(\pi)$, and the admissible search heuristic, $h(\pi)$. The heuristic can be multiplied by a relaxation scalar $\epsilon > 1$ in order to generate approximate results and produce plans faster. The TCRA* algorithm is reduced to a uniform cost search when $\epsilon = 0$. TCRA* parallelizes both trucks' actions, as shown in Figure 1, resulting in a 7 hour makespan.

TCRA*'s admissible heuristic estimates the costs of a given plan π in order to guide the TCRA* search. A plan π has a set of conflicts $K = \{\kappa_1, \ldots\}$. All conflicts K must be addressed, so the cost of π is at least as high as the cost of the most costly conflict, and $f(\pi) = max(K) = max(\ f(\kappa_1), \ \ldots\ )$ is an underestimate. Each conflict $\kappa \in K$ has a set of solutions $\Sigma = \{\sigma_1, \ldots\}$. The cost of conflict κ can be as low as the cost of the least costly solution; thus, the cost of κ is underestimated by $f(\kappa) = min(\Sigma) = min(\ f(\sigma_1), \ \ldots\ )$. Each solution σ generates a new plan, π_σ, whose cost is greater than or equal to its makespan, $f(\pi_\sigma) \ge m(\pi_\sigma)$. The makespan $m(\pi_\sigma)$ is an

```
Data: A (conflicted) plan $\pi_I$, consisted of a list of plans $\Pi$;
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 $K = \{\kappa_1, \ldots\}$ in plan $\pi$;
    if there are conflicts in plan $\pi$ then
        foreach conflict $\kappa \in K$ do
            Identify solutions $\Sigma = \{\sigma_1, \ldots\}$ to conflict $\kappa$;
            foreach solution $\sigma \in \Sigma$ do
                Apply solution $\sigma$ to produce plan $\sigma(\pi) = \pi_\sigma$;
                Compute $f(\pi_\sigma) = m(\pi_\sigma) + \epsilon \cdot h(\pi_\sigma)$;
                Enqueue plan $\pi_\sigma$ with priority $f(\pi_\sigma)$;
    else return conflict-free plan $\pi$ ;
return null;
```
Algorithm 3: Temporal Optimal Conflict Resolution (TCRA*).

underestimate of the cost of σ, $f(\sigma) = m(\pi_\sigma)$. The estimate cost $f(\pi) = max(\ min(\ m(\pi_{\sigma_1}),\ \ldots),\ \ldots)$, is an underestimate, and the heuristic $h(\pi) = f(\pi)$ - $m(\pi)$ is admissible.

TCRA*'s optimality is supported by the additive property of conflict resolution, which adds, but does not remove temporal orders. TCRA*'s priority queue expands a plan π^* before a plan π, if $m(\pi^*) \leq m(\pi)$. Successive conflict resolution iterations can *increase*, but cannot *decrease* π's makespan. Thus, π's makespan and the makespan of all successors of π will have a makespan greater than or equal to the makespan of π^*. The makespan of the resulting plan, π^*, is better than the makespan of any other satisficing plan; thus, the TCRA* is optimal with respect to the resulting plan's makespan. A detailed proof of optimality follows.

The first Lemma establishes that adding a temporal order generates a new plan with greater than or equal makespan.

Lemma 1. *Let π denote a plan and let $\prec_T$ denote a temporal order added to π, producing a new plan π_1. Then $m(\pi) \leq m(\pi_1)$.*

Proof. Let π's longest path be P. Add a temporal order $\prec_T$ to π, forming a new path P_1 in the resulting plan, π_1.
If $l(P_1) > l(P)$, then π_1's longest path is P_1 and

$$\begin{aligned} m(\pi_1) &= l(P_1) \\ &> l(P) \\ &> m(\pi). \end{aligned}$$

If $l(P_1) \leq l(P)$, then π_1's longest path is P and

$$\begin{aligned} m(\pi_1) &= l(P) \\ &= m(\pi) \end{aligned}$$ □

Solving an open precondition adds a temporal order, generating a new plan with greater than or equal makespan.

Lemma 2. *Let κ denote an open precondition of plan π. Let σ denote a solution to κ that adds a temporal order $\prec_T$ to plan π, producing a new plan $\sigma(\pi) = \pi_1$. Then $m(\pi) \leq m(\pi_1)$.*

Proof. The proof follows from Lemma 1. □

Solving a causal conflict adds a causal order, which is an extended temporal order, and generates a new plan with greater than or equal makespan.

Lemma 3. *Let κ denote a causal conflict of plan π. Let σ denote a solution to κ that adds a causal order $\prec_C$ to plan π, producing a new plan $\sigma(\pi) = \pi_1$. Then $m(\pi) \leq m(\pi_1)$.*

Proof. A causal order $\prec_C$ is an extended temporal order; thus, the proof follows from Lemma 1. □

Solving a conflict generates a new plan with greater than or equal makespan.

Lemma 4. *Let κ denote a conflict of plan π. Let σ denote a solution to κ, producing a new plan $\sigma(\pi) = \pi_1$. Then $m(\pi) \leq m(\pi_1)$.*

Proof. The proof follows from Lemmas 2 and 3. □

Successive conflict resolution iterations to a plan π generate a plan $\sigma^n(\pi) = \pi_n$ with greater than or equal makespan.

Lemma 5. *Let plan π have a series of conflicts $\kappa_1, \kappa_2, \ldots, \kappa_n$, resolved by a series of solutions $\sigma_1, \sigma_2, \ldots, \sigma_n$, and generating a series of plans $\pi_1, \pi_2, \ldots, \pi_n$. Then $m(\pi) \leq m(\pi_n)$.*

Proof. Plan π_{i+1}, produced by applying solution σ_{i+1} to conflict κ_{i+1} in plan π_i, has makespan $m(\pi_{i+1})$, $m(\pi_i) \leq m(\pi_{i+1})$, from Lemma 4. Then $m(\pi) \leq \ldots \leq m(\pi_i) \leq m(\pi_{i+1}) \leq \ldots \leq m(\pi_n)$. □

A plan returned by TCRA* has makespan less than or equal than any plan generated by successive conflict resolution iterations applied to the initial plan, π_I.

Theorem 1. *Let π^* denote the first plan returned by TCRA*. Let π denote a plan generated by successive conflict resolution iterations applied to the initial plan, $\pi \in \sigma^n(\pi_I)$. Then $m(\pi^*) \leq m(\pi)$.*

Proof. Let π denote a plan in TCRA*'s priority queue, the product of a series of conflict resolution iterations applied to the initial plan, π_I. Let π_n denote a plan generated by n successive conflict resolutions applied to plan π, then $m(\pi) \leq m(\pi_n)$, from Lemma 5. Plan π^* is returned before plan π; thus

$$\begin{aligned} m(\pi^*) &\leq m(\pi) \\ &\leq m(\pi_n). \end{aligned}$$ □

TCRA*'s search queue is ordered according to makespan. The lowest makespan plan is expanded first. Other plans in the queue will produce plans with higher makespan, as successive conflict resolutions cannot reduce makespan. Thus, the fist plan returned has the lowest makespan.

4 METHODOLOGY

The baseline merging algorithms, Serial and STA, Algorithms 2 and 1, respectively, and TCRA*, Algorithm 3, were evaluated across two domains. Each TMPCP problem was extracted from randomly generated planning problems that each consists of a randomly generated group of agents, called a grand coalition, an initial state, and a goal state composed of randomly generated tasks. Each task has an individual set of requirements, and all tasks' requirements must be met in order to solve the planning problem. Tasks were allocated to agents according to the agents' capabilities and the tasks' requirements using a coalition formation algorithm [23]. Each task was individually solved using the Coalition Formation and Planning framework [11], and an external planner. Each task solution resulted in a task plan. A merging problem consists of merging all the task plans to solve the original planning problem.

The Actions Concurrency and Time Uncertainty Planner (ActuPlan) [3], adopted as the external planner, supports concurrent durative actions with stochastic action durations. Similar planners exist [17, 6], but ActuPlan is the only planner to support a high-level problem description language and offer a publicly available implementation. The stochastic action durations were determinized [29] by adopting the distribution's mean as the true action duration.

The Logistics Domain [25], a benchmark domain for multiagent planning, was extended in order to introduce continuous distances between locations, and model the travel time between locations, as a function of distance. Specific truck types cannot travel outside of their assigned districts, and different truck types must coordinate in order to deliver trailers across districts. The trailers' initial and goal locations are selected randomly and each trailer delivery is a separate task. Problems are generated by randomly positioning and connecting locations on a 2D coordinate system. Two districts and truck types were generated, requiring up to 2 trucks per task. The makespan, the time necessary to accomplish all tasks and deliver all trailers, is minimized. Coupling exists within each task, as trucks need to coordinate actions in order to deliver a trailer. There is no coupling between tasks, as each trailer delivery is independent and does not conflict with others. However, plans for separate tasks can become coupled when sharing the same trucks. Task plans that share common trucks have increased coupling, since each shared truck cannot execute all task actions simultaneously. Transport actions require the trucks to be positioned at specific locations in order to attach, detach, and transport trailers. An individual truck cannot be positioned in multiple locations simultaneously, and the task actions have inherent temporal constraints, creating a tight coupling between tasks.

The Blocks World Domain [16] was extended to allow multiple robot arms to cooperate and concurrently rearrange stacks of blocks [11]. Specific block types require robots with specific grippers. Magnetic blocks require magnetic grippers and suction blocks require suction grippers. Each block type requires a stochastic amount of time to be manipulated, defined by a Gaussian probability distribution. The initial and goal states are generated by stacking blocks randomly. Each goal state stack generates a separate task, but the component blocks of different goal stacks often originate from the same initial state stack, causing tight coupling between tasks.

Each planning problem consisted of 1-10 tasks and 2-10 agents. One hundred problems were generated for each task-agent combination, resulting in 9,000 problems per domain. Each experiment was allocated one CPU core, with planning time and memory limited to 1 hour and 120 GB, on an Intel Xeon 5115 Linux server. Algorithms were implemented in Python 2.7.

5 RESULTS

The dependent variables are the success rates, makespan, processing time, and memory usage. The success rates are the ratio of problems solved within the 1 hour time limit. The TCRA* algorithm's relaxation parameter, described in Section 3, defines the weight of the search heuristic to balance between optimality and greediness. The TCRA* algorithm was evaluated for relaxation values $\epsilon = \{ 0, 1, 10, 100, 1000 \}$. The results for $\epsilon = 1000$ were not significantly different from the results for 100, and were omitted. The number of agents (trucks and robots) for each domain was 2-10 and due to space limitations, results for 2, 6, and 10 agents are presented without loss of detail. Optimal makespan is guaranteed for $\epsilon \leq 1$, but lower computational cost can be achieved for $\epsilon > 1$.

The Serial Algorithm successfully solved all of the Logistics Domain problems. The STA solved 41.8% of the problems, while the uniform cost version of TCRA*, with $\epsilon = 0$, solved 39.7%. The TCRA* solved 41.7% of the problems with $\epsilon = 1$, 44.4% with $\epsilon = 10$ and 44.9% with the highest epsilon value, $\epsilon = 100$, but the success rates varied significantly per number of tasks and trucks.

Success decreased for higher numbers of tasks, as presented in Figure 2, for TCRA* ($\epsilon = 10$), and increased for higher numbers of trucks. The same pattern was observed for STA and TCRA* for all ϵ. The higher ratios of tasks per truck cause fewer trucks to be allocated multiple tasks. The tight coupling caused by a high ratio of tasks per truck has significant impact on all evaluated metrics.

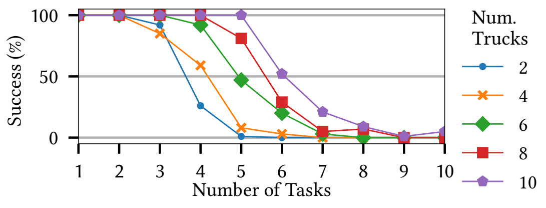

Figure 2: Success (%) per tasks and trucks for TCRA* ($\epsilon = 10$).

The makespan results, presented in Figure 3, are limited to the tasks that had at least 50% of the problems solved. Lower makespan values are better. One may assume, based on the success rate results, that the Serial Algorithm is the best solution; however, its makespan is linear as the number of tasks increase, see Figure 3 (a). The same pattern exists with additional trucks, as shown in Figure 3 (b-c).

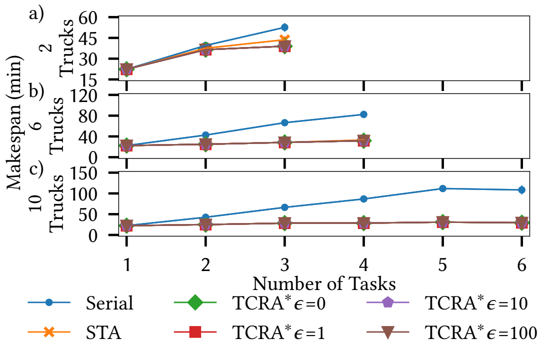

Figure 3: Makespan (min) per number of tasks for 2, 6, and 10 trucks. Note that the y-axes values vary.

The STA and TCRA* makespan was similar to the Serial Algorithm's makespan for 2 trucks, as shown in Figure 3 (a), but greater differences appeared with additional trucks. Increasing the number of trucks reduces the STA's and TCRA*'s makespan, see Figure 3 (b-c). A similar pattern is observed for 4 and 8 trucks, which are omitted. The STA resulted in a makespan less than, or equal to that of the Serial Algorithm across all numbers of tasks and trucks. TCRA* resulted in the same makespan for all ϵ values, which was less than or equal to the STA and the Serial Algorithm. STA's and TCRA*'s makespan benefit from the lower task per truck ratios, while the Serial Algorithm's was unaffected.

The Serial Algorithm solved all the problems in ≤ 0.5 min. The plan synthesis time for STA and TCRA* increased with the number of tasks per truck. The increased number of trucks, as presented across Figure 4 (a-c), resulted in less time, for each number of tasks for both algorithms. The processing time significantly increased when the ratio of tasks per truck is greater than 0.5, due to the more coupled merging problem, as multiple tasks must be allocated to each truck. Note that STA required 6.5 and 0.1 min to solve for 3 tasks using 2 and 6 trucks, respectively, while TCRA* ($\epsilon = 1$) required 5.7 and 0.1 min. Both STA and TCRA* required 0.5 min or less to solve problems with 3 tasks using 6 to 10 trucks. The same pattern was observed with more tasks. TCRA* ($\epsilon = 0$) required more processing time than STA for problems with 1-4 tasks, Figure 4 (a-b), but required less processing time for problems with 5 and 6 tasks, see Figure 4 (c). TCRA* ($\epsilon \geq 1$) required less processing time than STA for all numbers of trucks and tasks.

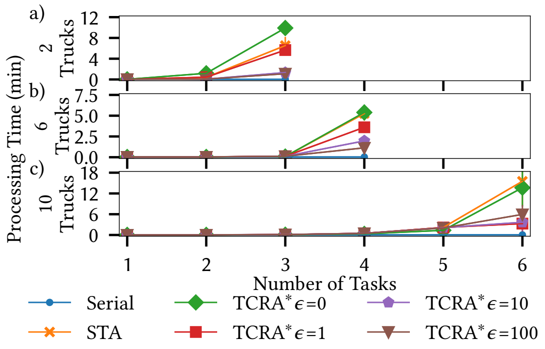

Figure 4: Processing time (min) per num. of tasks and trucks.

The Serial Algorithm memory usage was 2.1 and 1.6 GB for problems with 2 trucks and 1 and 2 tasks, respectively, but below 1.0 GB for problems with more trucks and tasks, as seen in Figure 5 (a-c). STA and TCRA* both required increasing memory with increasing numbers of tasks. STA required more memory than TCRA* when the number of tasks was greater than 3, see Figure 5 (b-c). STA and TCRA* ($\epsilon = 0$) resulted in a sudden increase in memory usage, as the number of tasks grew larger than 3, but TCRA* ($\epsilon = 100$), required memory usage similar to Serial.

The Serial Algorithm achieved the highest success rate, solved all problems, and required minimal processing time and memory usage. However, this algorithm's makespan increased linearly on the number of tasks, which was substantially higher than STA and TCRA*. Overall, STA and TCRA* required more processing time and memory usage, but resulted in substantially lower makespan as the number of tasks increased. These algorithms also resulted in an increased success rate and a decreased makespan as the number of trucks increased. TCRA* for all ϵ values, resulted in the lowest makespan, across all problems, and with $\epsilon = 100$, required less processing time and memory usage than the STA.

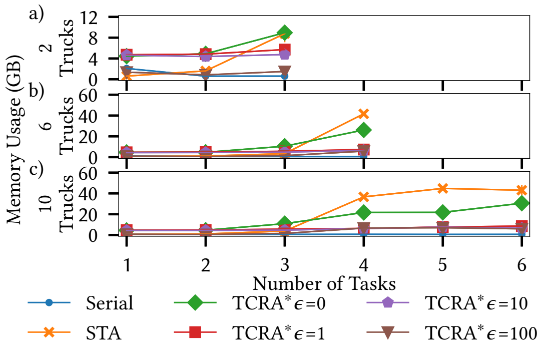

Figure 5: Memory usage (GB) per num. of tasks and trucks.

All of the Blocks World Domain problems were solved by the Serial Algorithm. STA solved 54.0% of the problems, whereas TCRA* solved 43.7% of the problems with $\epsilon = 0$, 48.9% with $\epsilon = 1$, 54.7% with $\epsilon = 10$, and 54.3% with $\epsilon = 100$. Similar to the Logistics Domain, the success rates varied per number of tasks and robots.

Additional tasks resulted in decreased success, as presented in Figure 6, for TCRA* ($\epsilon = 10$), but success increased with additional robots. The same pattern was observed for STA and TCRA* for all ϵ, but the increased success rate for higher numbers of robots is not as pronounced as in the Logistics Domain. The increased number of robots causes fewer tasks to be allocated to each robot, but the task coupling remains tight. The addition of robots does not reduce the problem complexity significantly, as the task coupling is mostly due to the same blocks being involved in multiple tasks.

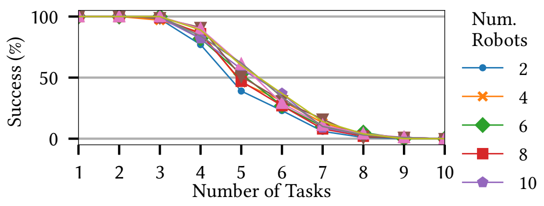

Figure 6: Success (%) per tasks and robots for TCRA* ($\epsilon = 10$).

The makespan results, presented in Figure 7, are limited to the tasks that had at least 50% of the problems solved. The number of tasks that had more than 50% of the problems solved did not vary significantly as more robots were added, and results are shown for 1-5 tasks across all numbers of robots.

The Serial Algorithm's makespan increases as the number of tasks increase, see Figure 7 (a). The same pattern exists with additional robots, as shown in Figure 7 (b-c). The STA and TCRA* makespan grew similarly to the Serial Algorithm's makespan, but

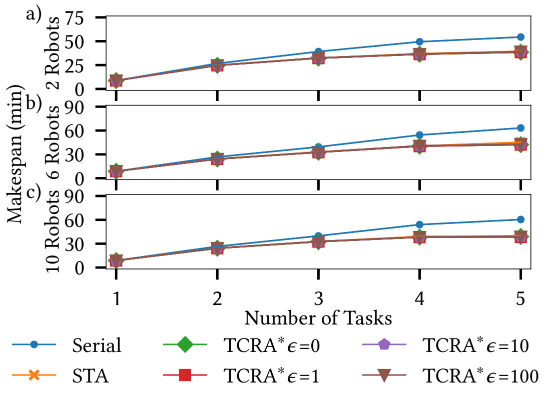

Figure 7: Makespan (min) per num. of tasks and robots.

the STA resulted in a makespan less than, or equal to that of the Serial Algorithm across all numbers of tasks and robots. TCRA*, for all ϵ values, resulted in strictly less than or equal makespan to STA and the Serial Algorithm. Contrary to the results in the Logistics Domain, the ratio of tasks per robot did not impact makespan.

All of the problems were solved by the Serial Algorithm in ≤ 0.5 min. The plan synthesis time for STA and TCRA* increased with the number of tasks. TCRA* ($\epsilon = 0$) required the highest processing time, followed by TCRA* ($\epsilon = 1$) and TCRA* for $\epsilon \geq 10$, as presented in Figure 8 (a). TCRA* ($\epsilon = 10$) required lower processing time than STA for 2 robots and 5 tasks, as shown in Figure 8 (a), but the STA required significantly lower processing time for fewer tasks. The same pattern occurred for more robots, but lower ϵ values required less processing time than STA with 5 tasks as more robots were added. TCRA* ($\epsilon \geq 1$) had lower processing time than STA for 6-10 robots and 5 tasks, see Figure 8 (b-c). STA's processing time, relative to TCRA*, was better than in the Logistics Domain.

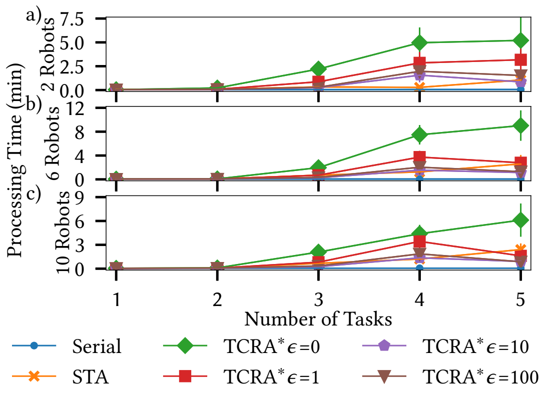

Figure 8: Processing time (min) per num. of tasks and robots.

Memory usage was approximately constant when adding up to 3 tasks, as seen in Figure 9 (a), across all algorithms. A sudden increase occurred for TCRA* ($\epsilon = 0$) with 4-5 tasks. This pattern repeats for more robots, and STA had the same increase in memory usage, as seen in Figure 9 (c). However, TCRA*'s memory usage remained low for $\epsilon \geq 1$. STA's memory usage, relative to TCRA*, was also better than in the Logistics Domain, as there were fewer problem instances where STA resulted in the worst memory usage.

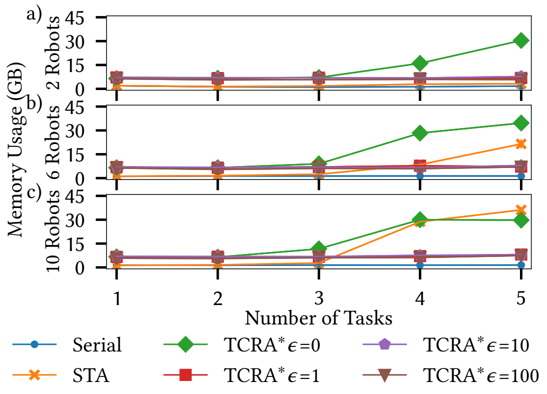

Figure 9: Memory usage (GB) per num. of tasks and robots.

TCRA* and STA offer similar makespan across all numbers of robots and tasks, but both are lower than the Serial Algorithm's makespan. TCRA*'s and STA's makespan also become smaller, relative to Serial's, as the number of tasks increase. TCRA* ($\epsilon \geq 10$) requires similar processing time as STA, but lower than STA for the largest numbers of robots and tasks. TCRA* ($\epsilon \geq 1$), similarly, requires approximately the same memory usage to that of STA for a low number of tasks, but requires less memory than STA when the number of robots and tasks is large. Among the algorithms offering the lowest makespan, TCRA* ($\epsilon \geq 10$) offers the lowest cost.

The Serial Algorithm solved all problems using the lowest processing time and memory, but had the highest makespan, across both domains. TCRA* and STA both resulted in a decaying success rate as the number of tasks increased. More tasks resulted in more plan actions to merge and more plan conflicts to resolve, which increased processing time and memory usage. Both algorithms resulted in better success rates with additional robots, although the impact was more pronounced in the Logistics Domain. TCRA* and STA resulted in the lowest makespan, and TCRA* resulted in lower makespan than STA for the most difficult problems across both domains. The TCRA* heuristic had a significant impact on the processing time and memory usage, as increased ϵ values required lower computational resources. TCRA* ($\epsilon \geq 10$) offered the second lowest processing time and memory usage, yet resulted in the best makespan across both domains.

6 DISCUSSION

Coupling between tasks was the main factor limiting the STA's and TCRA*'s ability to scale to an increasing number of tasks. Tighter task coupling required more computational resources, as more conflicts arise when merging task plans. Coupling varied per domain, as coupling is due to the allocation of multiple tasks to each agent and dependencies caused by shared objects.

The allocated computational power, 1-hour processing time limit, and graph search implementation impacted the success rates negatively. All the problems evaluated are solvable, as each task plan was synthesized serially, and the conflicts can be resolved. STA and TCRA* can generate successful results for all problems with additional processing power, a longer processing time, and by optimizing and parallelizing the graph search algorithm implementations.

6.1 The Logistics Domain

The Logistics Domain results were influenced by task coupling due to the ratio of tasks per truck. Problems with many tasks per truck were difficult to solve. The high ratio of tasks per truck resulted in multiple tasks being allocated to each truck and created a tighter coupling between each task. This tighter coupling increased the dependencies between task plans, causing more conflicts, and increasing the search space for STA and TCRA*. Adding trucks decreased the number of tasks allocated to each truck, which reduced the task coupling. The STA and TCRA* resulted in high quality plans that leverage the increased number of trucks to minimize makespan. This better makespan, compared to the Serial Algorithm, is more pronounced for larger numbers of trucks and tasks. TCRA* ($\epsilon > 1$) resulted in better plans than the STA, at a cost that was equal or lower than the STA. TCRA* ($\epsilon > 10$) also provided high quality plans at a cost comparable to that of the Serial Algorithm.

6.2 The Blocks World Domain

The Blocks World Domain results were largely independent of the number of robots, as the stacking of blocks causes task coupling. The addition of robots reduces the number of tasks allocated to each robot, but does not reduce the number of blocks that are involved in multiple tasks. Task plans sharing the same blocks are tightly coupled, as actions that move a common block have dependencies.

The Serial Algorithm required the lowest computational cost, but the highest makespan. TCRA* is the best solution when the fastest plan execution is required, at the cost of increased computational resources. TCRA* ($\epsilon \geq 10$) solved problems faster than STA for the largest problems solved, yet resulted in lower makespan.

6.3 Overall

The Serial Algorithm, a naive concatenation of plans, requires limited processing time and memory usage, but does not minimize the resulting makespan, and fails to benefit from additional agents. The serial plan execution, with a single task executed at a time, wastes agent-hours, as more agents and tasks are allocated, and causes a growing number of agents to wait or remain idle. The serial execution results in longer makespans as the number of tasks increases. However, the Serial Algorithm is the best choice for a single agent, when the opportunity to parallelize action execution is limited, or when the makespan does not need to be minimized.

STA and TCRA* account for action dependencies between task plans, which results in increased computational complexity and lower success rates. However, the algorithms leverage more agents to parallelize task execution and reduce the resulting makespan. The added computational complexity translates into lower makespan and faster plan execution. Fewer agents are idle as more tasks execute simultaneously. The benefits are more pronounced as tasks and agents are added, and more tasks can be parallelized. The main weakness of both algorithms is the increased computational complexity, which is worse than the serial concatenation of plans.

The TCRA* ($\epsilon = 1$) is guaranteed to result in the minimum makespan. TCRA*'s optimality guarantee defines lower bounds on solution quality and provides a conceptual benchmark. Knowing the lowest possible makespan helps quantify the quality of suboptimal solutions, such as those produced by Serial and STA.

The makespan-optimality of the input plans is not required, as TCRA* will minimize the resulting makespan, given a set of sub-optimal input plans. End-to-end makespan optimality is not guaranteed, since TCRA* preserves the input plans' actions and temporal orders. Algorithms that remove actions and temporal orders (i.e., PMA) can result in lower makespan, but at higher computational cost. TCRA* is the first merging algorithm to preserve the input actions and temporal orders while minimizing makespan.

TCRA*'s relaxation scalar, $\epsilon > 1$, resulted in a lower computational cost than STA for the largest problems solved. The admissible heuristic is simple, and future work can introduce more informed heuristics to improve the TCRA*'s scalability to a larger number of agents and tasks, while lowering the computational cost.

TCRA* is an especially relevant tool for large scale logistics environments, where each action can require hours to complete, and a reduction in makespan can offset the added computation time. Additionally, TCRA* is the best tool in scenarios where plans can be generated in advance, but the plan execution is time critical. An example of such a domains is extraplanetary robot missions [2], where the robots' operation is limited to day cycles due to solar power limitation and the plans can be generated overnight.

Complex multi-robot problems will require better task allocation algorithms, as Shehory and Kraus's [23] coalition formation algorithm does not account for certain aspects, such as the fact that some robot resources cannot be exchanged during task execution [28]. Multi-robot coalition formation algorithms can account for such constraints and result in better task allocation [21, 27].

STA's and TCRA*'s computational cost can also be improved by leveraging domain knowledge, such as task ordering [23]. A hybrid of TCRA* and the Serial Algorithm can selectively serialize tasks before resolving conflicts. Fewer conflicts will remain after the serialization, which will reduce the computational cost.

7 CONCLUSION

TCRA* was introduced to merge independently generated plans and minimize the resulting makespan, while accounting for durative actions. A proof of optimality was provided and the algorithm was empirically evaluated across two multiagent domains against two baseline algorithms, the Serial Algorithm, and the STA. STA was extracted from the PMA and used as a stand-alone algorithm for comparison to TCRA*. Serial and STA are the only algorithms comparable to TCRA*. STA's constraint satisfaction does not minimize makespan, but TCRA*'s A* search uses an admissible heuristic and a relaxation scalar to prune the search space and minimize makespan optimally. TCRA* results in better makespan, and a search relaxation constant allows it to generate better plans with lower processing time and memory usage than the STA. TCRA* provides the best solution when the lowest makespan is necessary.

8 ACKNOWLEDGMENTS

This work was partially supported by NSF grant #1723924.

REFERENCES

[1] Ron Alterovitz, Sven Koenig, and Maxim Likhachev. 2016. Robot planning in the real world: Research challenges and opportunities. *AI Magazine*, 37, 2, 76–84.

[2] Max Bajracharya, Mark W. Maimone, and Daniel M. Helmick. 2008. Autonomy for mars rovers: past, present, and future. *IEEE Computer*, 41, 12, (December 2008), 44–50.

[3] Eric Beaudry, Froduald Kabanza, and Francois Michaud. 2012. Using a Classical Forward Search to Solve Temporal Planning Problems under Uncertainty. In *Workshops at the AAAI Conference on Artificial Intelligence*, 2–8.

[4] Daniel Borrajo. 2013. Multi-agent planning by plan reuse. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems.* (May 2013), 1141–1142.

[5] Ronen I. Brafman and Carmel Domshlak. 2013. On the complexity of planning for agent teams and its implications for single agent planning. *Artificial Intelligence*, 198, (May 2013), 52–71.

[6] Olivier Buffet and Douglas Aberdeen. 2009. The factored policy-gradient planner. *Artificial Intelligence*, 173, 5-6, (April 2009), 722–747.

[7] Jeffrey S. Cox and Edmund H. Durfee. 2009. Efficient and distributable methods for solving the multiagent plan coordination problem. *Multiagent and Grid Systems*, 5, 4, (December 2009), 373–408.

[8] Matthew Crosby, Michael Rovatsos, and Ronald P. A. Petrick. 2013. Automated agent decomposition for classical planning. In *Proceedings of the International Conference on Automated Planning and Scheduling.* (June 2013), 46–54.

[9] Mathijs de Weerdt and Brad Clement. 2009. Introduction to planning in multiagent systems. *Multiagent and Grid Systems*, 5, 4, (December 2009), 345–355.

[10] Yannis Dimopoulos, Muhammad A. Hashmi, and Pavlos Moraitis. 2012. Mu-SATPLAN: multi-agent planning as satisfiability. *Knowledge-Based Systems*, 54–62.

[11] Anton Dukeman and Julie A. Adams. 2017. Hybrid mission planning with coalition formation. *Journal of Autonomous Agents and Multi-Agent Systems*, 31, 6, (November 2017), 1424–1466.

[12] Michael Luck, Vladimír Mařík, Olga Štěpánková, and Robert Trappl, (Eds.) 2001. *Distributed problem solving and planning. Multi-Agent Systems and Applications.* Springer Berlin Heidelberg, Berlin, Heidelberg, 118–149.

[13] Malik Ghallab, Dana S. Nau, and Paolo Traverso. 2016. *Automated planning and acting.* Cambridge University Press.

[14] Daniel Gnad and Jorg Hoffmann. 2018. Star-topology decoupled state space search. *Artificial Intelligence*, 257, (April 2018), 24–60.

[15] Carlos Guestrin and Geoffrey Gordon. 2002. Distributed planning in hierarchical factored MDPs. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 197–206.

[16] Naresh Gupta and Dana S. Nau. 1992. On the complexity of blocks-world planning. *Artificial Intelligence*, 56, 2-3, 223–254.

[17] Mausam and Daniel S. Weld. 2008. Planning with Durative Actions in Stochastic Domains. *Journal of Artificial Intelligence Research*, 31, (January 2008), 33–82.

[18] Lenka Mudrova, Bruno Lacerda, and Nick Hawes. 2016. Partial order temporal plan merging for mobile robot tasks. In *Proceedings of the European Conference on Artificial Intelligence.* (August 2016), 1537–1545.

[19] Ranjit Nair, Pradeep Varakantham, Milind Tambe, and Makoto Yokoo. 2005. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *Proceedings of the AAAI Conference on Artificial Intelligence.* (July 2005), 133–139.

[20] Martha E. Pollack, David Joslin, and Massimo Paolucci. 1997. Flaw selection strategies for partial-order planning. *Journal of Artificial Intelligence Research*, 6, 223–262.

[21] Sarvapali D. Ramchurn, Maria Polukarov, Alessandro Farinelli, Cuong Truong, and Nicholas R. Jennings. 2010. Coalition formation with spatial and temporal constraints. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems.* (May 2010), 1181–1188.

[22] Mohsen Shahandasht, Binaya Pudasaini, and Sean Logan McCauley. 2019. Autonomous Vehicles and Freight Transportation Analysis. Technical report. The University of Texas at Arlington, (August 2019).

[23] Onn Shehory and Sarit Kraus. 1998. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101, 1-2, 165–200.

[24] Adriaan W. ter Mors, Jeroen M. Valk, and Cees Witteveen. 2006. Task coordination and decomposition in multi-actor planning systems. In *Proceedings of the Workshop on Software-Agents in Information Systems and Industrial Applications.* (February 2006), 83–94.

[25] Alejandro Torreno, Eva Onaindia, Antonin Komenda, and Michal Stolba. 2017. Cooperative multi-agent planning. *ACM Computing Surveys*, 50, 6, (November 2017), 1–32.

[26] Alejandro Torreno, Eva Onaindia, and Oscar Sapena. 2014. FMAP: Distributed cooperative multi-agent planning. *Applied Intelligence*, 41, 2, (June 2014), 606–626.

[27] Lovekesh Vig and Julie A. Adams. 2007. Coalition formation: From software agents to robots. *Journal of Intelligent and Robotic Systems.*, 1, 85–118.

[28] Lovekesh Vig and Julie A. Adams. 2006. Multi-robot coalition formation. *IEEE Transactions on Robotics*, 4, 637–649.

[29] Sungwook Yoon, Alan Fern, Robert Givan, and Subbarao Kambhampati. 2008. Probabilistic planning via determinization in hindsight. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1010–1016.