HYBRID MISSION PLANNING WITH COALITION FORMATION

By

Anton Dukeman

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

May, 2017

Nashville, Tennessee

Approved:

Julie A. Adams, Ph.D.

Gautam Biswas, Ph.D.

Mark Ellingham, Ph.D.

Maithilee Kunda, Ph.D.

Eugene Vorobeychik, Ph.D.

ACKNOWLEDGMENTS

I want to thank my advisor, Dr. Julie Adams, for her guidance while performing this research as well as her many hours reading and editing my writing. She was a great source of encouragement through my years at Vanderbilt. I also want to thank my committee members, Dr. Gautam Biswas, Dr. Eugene Vorobeychik, Dr. Maithilee Kunda, and Dr. Mark Ellingham, for their time and support of my research. My discussions with them and their comments regarding my research were invaluable.

I am grateful to fellow lab members and friends, Jason, Jamison, Electa, and Musad. Our conversations during the day and at lunch have been great for relieving stress over upcoming deadlines. Outside the lab, I could not have asked for a better group of friends to have been with me through the past years. Their friendship is part of what makes life wonderful. There were events I had to miss these past years due to late nights and weekends working, but they were always understanding and supportive. The love and support from my friends is surpassed only by that from my family. I especially want to thank my parents, Greg and Margaret. They are a big reason I have been able to achieve all I have been able to achieve.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

Abbreviations

CFP	Coalition Formation then Planning
COLIN	COntinuous LINear planner
EHC	Enforced Hill Climbing
EMS	Emergency Medical Services
HMPCF	Hybrid Mission Planning with Coalition Formation
i-CiFHaR	Intelligent Coalition Formation for Humans and Robots
IPC	International Planning Competition
ITSAT	SAT-based temporal planner
MACPDDL	Multi-Agent Capabilities and Planning Domain Definition Language
PA	Planning Alone
POPF	State space planner used in experiments
RPCA	Coalition Formation then Planning with Relaxed Plan Coalition Augmentation
TF	Task Fusion
TFD	Temporal Fast Downward planner
UA	University of Alabama

Symbols

Φ	Grand coalition
Φ_i	Coalition i
ϕ_i	Agent i
π	Plan
A	Set of actions
a	An action
$A(i)$	Set of actions executable by agent or coalition i
Cap	Capability vector
Cap^i	Capability vector required (offered) by task (agent or coalition) i
Cap_i^j	Amount of capability i required (offered) by task (agent or coalition) j
$cond(v)$	State constraints to satisfy task v
$cond_{\dashv}(a)$	Conditions to end action a execution
$cond_{\leftrightarrow}(a)$	Conditions during action a execution

$cond_{\vdash}(a)$	Conditions to start action a execution
$dur(a)$	Duration constraint of action a
$\mathit{eff}_{\dashv}(a)$	State effects applied at end of action a execution
$\mathit{eff}_{\leftrightarrow}(a)$	Continuous state effects applied during action a execution
$\mathit{eff}_{\vdash}(a)$	State effects applied at start of action a execution
G	Set of goal states
I	Initial state
Q	Plan quality function
Res_i^j	Amount of resource i for entity j
Ser_i^j	Amount of service i for entity j
S	State space
$start(a)$	Start time of action execution a
V	Set of tasks
v	A task

CHAPTER I

Introduction

Robotic systems have proven effective in many domains. Some robotic domains, such as mass casualty response, require close coupling between the humans and robots that are able to adapt to the environment and tasks for successful completion of a mission. The planning problem uses the task requirements and the team members' capabilities to determine a set of actions that will accomplish the task. However, planning problem difficulty increases exponentially with the number of expressive features included in the mission description, the number of agents, and the number of tasks. Each additional agent and task makes a problem more difficult and the problem can quickly become intractable, especially for missions requiring rapid decisions.

A natural first step to address planning problem difficulty is assigning agents (humans or robots) to teams based on each agent's synergistic capabilities and the requirements of each task, a problem known as coalition formation. However, coalition formation is also a difficult problem to solve and does not produce a plan for completing the task. Once coalition formation allocates agents to tasks, the question that remains is how the agents will complete their assigned tasks.

The research goal is to quickly find approximate solutions to the coalition formation problem and use the results to focus the planning problem on individual tasks with coalitions of the available agents. Unfortunately, there are no guarantees that planning will be able to derive a plan for each coalition-task allocation produced by coalition formation, resulting in a nonexecutable coalition. Planning tools must be resilient to nonexecutable coalitions; the tool must be able to use information from planning to intelligently augment the generated coalitions in order to transition nonexecutable coalitions to executable coalitions with a plan to accomplish their assigned task.

Planning for each coalition and task individually will greatly reduce problem difficulty at the cost of being unable to consider the interactions between the plans for each task. Conversely, planning for multiple tasks simultaneously allows the interactions of the tasks to be considered in the resulting plan, but increases problem difficulty. Intelligently merging select tasks and coalitions that are likely to interact can balance problem difficulty with the need to derive plans by considering possible agent and task interactions.

I.1 Research Outline

Coalition formation lacks the details concerning how a coalition will accomplish a task. Planning without the higher level knowledge of which agents have the capabilities required to execute a task leads to large complex problems. Coupling the two problems creates tractable problems for which solutions can be found quickly.

Coupling coalition formation and planning can be accomplished with four tools: planning alone, coalition formation then planning, relaxed plan coalition augmentation, and task fusion. Each tool is implemented and evaluated. Planning alone produced plans, but requires large amounts of computational resources. Coalition formation then planning reduces the computational resource requirements, but can fail to produce plans due to nonexecutable coalitions. Relaxed plan coalition augmentation adds agents to nonexecutable coalitions in order to produce executable coalitions. Task fusion combines similar task and coalition allocations in order to maximally exploit the capabilities of each coalition during planning. Each tool is agnostic to the underlying coalition formation algorithms and planning algorithms. Multiple coalition formation algorithms and multiple planning algorithms are analyzed for use with the presented tools.

A set of four test domains and associated problems are presented to exploit the capabilities and reveal the weaknesses of each new tool. Three of the domains are existing planning research domains, modified for the purposes of this research. The Rovers domain demonstrates failures in coalition formation due to no plan existing for the allocated coalition to complete the assigned task. The Blocks World domain demonstrates coalition formation failures resulting from plans for previous tasks making a coalition nonexecutable. Zenotravel demonstrates the capabilities of task fusion by combining similar tasks. The fourth domain is First Response, a representation of the problems this research aims to address, in the form of immediate response to a natural disaster. Each domain includes heterogeneous agents and durative actions, with the last three including hybrid boolean-continuous state spaces. The new tools will use existing coalition formation algorithms and existing planning algorithms supporting the necessary expressive features of each domain. Plan makespan, number of plan actions, plan derivation time, and planning tool memory requirements are used to compare the performance of the tools for test cases in each domain.

I.2 Dissertation Roadmap

The subsequent chapters are organized as follows. Chapter II presents related prior research on task allocation, coalition formation, planning, and combinations of each field. A formal definition of the problem is presented in Chapter III, along with four domains and how they map to the problem definition. Chapter IV presents the tools developed to solve the problem: Planning Alone, Coalition Formation then Planning, Relaxed Plan Augmentation, and Task Fusion. Chapter V presents the experimental design used to analyze the problems in the test domains. Chapter VI present the results for each of the four tools. Finally, Chapter VII provides a conclusion and summarizes the dissertation contributions.

CHAPTER II

Literature Review

Multi-agent systems research covers many different areas. Research problems of importance for this dissertation include task allocation, coalition formation, and planning. Results from each of these problems interact with one another when designing multi-agent systems for the real world.

II.1 Task Allocation

The task allocation problem attempts to determine which agents to assign to each task. Formally, the task allocation problem assumes a set of n agents known as the grand coalition, $\Phi = \{\phi_1, \ldots, \phi_n\}$, and a set of m tasks, $V = \{v_1, \ldots, v_m\}$ and allocates agents to each task. Finding an optimal solution requires defining a utility function mapping agent-task allocations to a real value, $U : 2^{\Phi} \times V \rightarrow \mathbb{R}$.

Gerkey and Matarić (2004) proposed a taxonomy of multi-robot problems applicable to task allocations along three axes. The first axis distinguishes between robots capable of executing only a single task at a time versus those that are capable of executing multiple tasks at a time. The second axis distinguishes between tasks requiring a single robot versus those that require multiple robots. The final axis distinguishes between how the task allocation decision must be made: instantaneously or time-extended. Instantaneous assignment problems permit only an instantaneous allocation of tasks to the robots, whereas time-extended allocation problems have more information, such as the set of tasks that will need to be assigned after execution has commenced or a model for how the tasks are expected to arrive for allocation. This taxonomy assumes that tasks are independent, which is rarely the case in real world domains.

The iTax taxonomy extended the Gerkey and Matarić taxonomy to include varying levels of task dependence (Korsah et al., 2013). Four levels of task dependence are used: no dependencies, in-schedule dependencies, cross-schedule dependencies, and complex dependencies. In-schedule dependency tasks have constraints on a single agent's schedule. Cross-schedule dependency tasks have constraints across multiple agent's schedules, while complex dependency tasks have constraints across multiple agents' schedules that also depend on the plan an agent will execute to accomplish its task. Task dependency is also referred to as *task coupling*.

The ALLIANCE architecture for fault tolerant multi-robot cooperation was a behavior-based system with mathematically modeled motivations within each robot (Parker, 1998). Each robot had a set of behaviors from which one was dynamically selected, based on how the environment and the other agents in the system changed. The fully distributed system allowed robots to be added as desired by the system creators. Broadcast

of Local Eligibility was another behavior-based architecture (Werger and Matarić, 2000), in which robots communicate their eligibility for a task to other local robots. The robot with the highest eligibility for a task claims it by assuming the behavior to execute the task and actively inhibiting the behavior on the other robots. This approach is also fault-tolerant, because the inhibition is an active process; thus, other robots will acknowledge the task as unallocated upon agent failure.

A similar approach to ALLIANCE, the Automated Synthesis of Multi-robot Task solutions through software Reconfiguration (ASyMTRe) system represented robots as a set of schemas (Tang and Parker, 2005). Robots were defined by their environmental sensors and three different schema categories: perceptual, motor, and communication. Tasks were represented as a set of motor schema requirements and task specific parameters. An anytime algorithm connected robot schemas within and across robots to develop a joint schema capable of accomplishing the assigned task. Information Quality based ASyMTRe (IQ-ASyMTRe) addressed drawbacks of ASyMTRe and extended it to work with tightly-coupled tasks (Zhang and Parker, 2010). IQ-ASyMTRe considered both the quality and type of sensor information provided by a schema and moved from a centralized architecture to a distributed architecture to improve fault-tolerance. A similar approach, the Remote Object Control Interface framework, considered each robot as a node with several modules encapsulating specific functions, such as processing and sensing (Chaimowicz et al., 2003). Each robot has a kernel that tracks the functionality exported by each module in the network and dynamically connects modules to form a functioning application.

Social and biological systems have inspired task allocation algorithms as well. Vacancy chains are a social structure common in both human and animal groups. An example is a senior manager creating a vacancy at a company by retiring and the position being filled by a junior employee creating another vacancy and so on. Dahl et al. (2003) proposed a method of task allocation based on vacancy chains, focused on the subset of problems where tasks can be completed indefinitely, such as assembly lines. Tasks are assigned a value and the task completion rate is a function of the number of robots working on the task with diminishing returns. Robots allocate themselves to the task where they provide the highest marginal utility. Other biologically inspired approaches include artificial bee colony (TSai et al., 2009; Liu et al., 2015) and ant colony optimization (Pendharkar, 2015).

Market based approaches encourage individual robots to work towards the team goal by increasing their share of the team revenue based on their share of the work towards a goal. Stentz and Dias (1999) developed a distributed market-based approach in which robots use an ask-bid system to provide services. Cooperative interaction provides services that no robot can provide on its own, such as a grasp-and-lift robot working with a transport robot to provide a moving service. The other interaction mechanism is competitive, which requires multiple robots provide the same service and must outbid each other to perform the service. The bid-

ask system encodes all the information about availability of services and cost to perform those services in a single number; thus, limiting the communication overhead to the ask and bid prices. Market based approaches to task allocation have also been developed for optimal assignment in fault-tolerant networks (Liu and Shell, 2013), wireless sensor networks (Mezei et al., 2013; Haghighi, 2014), and sequentially interdependent tasks in swarm robotics (Brutschy et al., 2014).

Task allocation covers a wide range of problems. The next section focuses on multi-task robots working on multi-robot tasks, commonly referred to as coalition formation.

II.2 Coalition Formation

Coalition formation is a subclass of the task allocation problem without constraints on the number of agents allocated to each task or the number of coalitions to which an agent is a member. Coalition formation is an *NP*-complete problem (Sandholm et al., 1999) that is also difficult to approximate (Service and Adams, 2011). The goal is to form teams of agents that are together more capable than the team's individual agents and can accomplish a set of assigned tasks, while optimizing an objective function (e.g., utility, cost, or number of tasks completed). The general coalition formation problem assumes a set of n agents, $\Phi = \{\phi_1, \ldots, \phi_n\}$, and a set of m tasks, $V = \{v_1, \ldots, v_m\}$. A solution to the problem is a mapping of tasks, $v \in V$, to coalitions allocated to the task, $\Phi_v \subseteq \Phi$.

Two common models for coalition-task utility are the resource model and the service model. The *resource model* treats each agent as a set of resources (e.g., chemical sensor, camera, laser) and each task as a set of resource requirements. The resource vector for agents, Res_j^{ϕ}, is defined as

$$Res_j^{\phi} \geq 0 \mid \phi \in \Phi, 1 \leq j \leq n_r, \tag{II.1}$$

where n_r is the number of types of resources defined for the problem and Res_j^{ϕ} denotes the amount of resource j available to agent ϕ. Tasks also have a resource vector, defined as

$$Res_j^{v} \geq 0 \mid v \in V, 1 \leq j \leq n_r, \tag{II.2}$$

where Res_j^{t} is the amount of resource j required to complete task v.

The *service model* is a more constrained version of the resource model and it treats each agent as a set of functions that each agent can perform (e.g., box-pushing, mapping, sentry-duty). The service vector for agents, Ser_j^{ϕ}, is defined as

$$Ser_j^{\phi} \in \{0, 1\} \mid \phi \in \Phi, 1 \leq j \leq n_s, \tag{II.3}$$

where n_s is the number of service types defined for the problem, $Ser_j^\phi = 0$ if agent ϕ cannot perform service j and 1 if agent ϕ can perform service j. Tasks have a service requirement vector defined as

$$Ser_j^v \geq 0 \mid v \in V, 1 \leq j \leq n_s, \tag{II.4}$$

where Ser_j^v denotes the number of agents capable of performing service j, which is required to execute task t.

Definition II.1 (Candidate Coalition)**.** *A coalition, $\Phi_v \subseteq \Phi$, is a* candidate coalition *for task v, if and only if the sum of the services (or resources) that can be provided by the agents in the coalition is greater than the services (or resources) required by the task:*

$$\sum_{\phi \in \Phi_v} Ser_j^\phi \geq Ser_j^v \mid \forall j \in \{1, \ldots, n_s\},$$
$$\sum_{\phi \in \Phi_v} Res_j^\phi \geq Res_j^v \mid \forall j \in \{1, \ldots, n_r\}.$$

The additional constraints in the service model allow a coalition formation algorithm using the service model to make assumptions a resource model coalition formation algorithm cannot make (e.g., a candidate coalition of size at least three is required for a service model task requiring three units of a service). Throughout the remainder of this dissertation, the term *capability model* will be used to refer to a model applicable to coalition formation, whether that is the resource model, the service model, or some future model.

There are many approximation-based algorithms, each with its own strengths and weaknesses. Greedy algorithms can derive solutions quickly, but make no guarantees on the solution quality (Shehory and Kraus, 1998; Vig and Adams, 2006b; Ramchurn et al., 2010; Service and Adams, 2011; Sujit et al., 2014). Approximation algorithms provide solution quality guarantees, but suffer from poor worst-case run-time complexity, which can render them inappropriate for real-time applications (Dang and Jennings, 2004; Rahwan et al., 2009; Liemhetcharat and Veloso, 2014). Market-based techniques offer fault-tolerance for a distributed system, but have high communication processing requirements (Dias, 2004; Dias et al., 2005; Vig and Adams, 2006a; Shiroma and Campos, 2009; Service et al., 2014). Biologically inspired ant colony optimization algorithms have been applied to several NP-complete problems including coalition formation (Xia et al., 2004; Ren et al., 2008; Sen and Adams, 2015).

No single algorithm is appropriate for all real world scenarios; thus, algorithm selection must be performed intelligently to ensure the use of the best algorithm(s) for the situation. Using an algorithm that is ill-suited for the mission will negatively affect the resulting solution quality, potentially resulting in a brittle

Table II.1: Coalition Formation Algorithm Taxonomy

Category	Taxonomic Feature	Feature Domain Values
Agent	Agent Orientation	Group-Rational, Self-Interested
	Agent Type	Homogeneous, Heterogeneous
	Agent Capability Model	Resource, Service
	Agent Awareness	Aware, Partially, Unaware
	Agent Structure	Social Network, Organization Hierarchy, None
Task	Inter-Task Constraints	Yes, Prerequisite, No
	Task Preemption	Yes, No
	Task Resource Model	Resource, Service
	Intra-Task Constraints	Yes, No
	Task Coupling	Tightly, Loosely, Intermediate
Domain	Performance Criterion	Maximize Utility, Minimize Cost, Maximize Tasks
	Communication Overhead	High, Low
	Task Allocation	Instantaneous, Time-Extended
	Spatial Constraints	Yes, No
	Overlapping Coalitions	Yes, No
Algorithm	Algorithm Technique	Greedy, Auction-Based, Approximation
	Implementation	Centralized, Decentralized
	Coalition Size Constraint	Single, None, Fixed Upper Limit

team. The intelligent Coalition Formation for Humans and Robots framework (i-CiFHaR) uses mission scenario criteria to select the best subset of coalition formation algorithms from a library of nineteen coalition formation algorithms (Sen and Adams, 2013, 2015).

Each coalition formation algorithm in the i-CiFHaR library is classified using a taxonomy (see Table II.1) (Service, 2010). A link analysis uses the feature-value pairs associated with the coalition formation algorithms to compute base utility scores for each possible pair, which are incorporated into a utility table for each possible combination of feature-value pairs. Feature extraction, using principal component analysis, removes redundant or insignificant taxonomic features from the algorithm analysis. The remaining features are used as chance nodes in an influence diagram. Missions are defined in terms of required feature-value pairs that each have a probability value representing the confidence in the assignment. The probability values for the mission are represented by the chance nodes in the influence diagram. The chance nodes feed into a single decision node for selecting a coalition formation algorithm(s). The chance and decision nodes are used to calculate a utility for the selected coalition formation algorithm(s). The utility value for an algorithm is higher if the mission has matching feature-value pairs with the coalition formation algorithm and the feature-value assignment has a high confidence.

i-CiFHaR can provide decision support to human mission planners. The human operators provide the mission specifications, i-CiFHaR selects the coalition formation algorithm(s) and generates the coalition(s). If a coalition value is above the acceptable threshold, the coalition is automatically deployed. Otherwise, the

human is presented with the alternatives and can either select a coalition to deploy or modify the mission criteria. Once a coalition is formed, planning is necessary to determine how the coalition will accomplish its assigned task.

II.3 Planning

One of the first planning systems was the STanford Research Institute Problem Solver (STRIPS) (Fikes and Nilsson, 1971). The STRIPS system produced a plan consisting of actions to achieve a goal when executed sequentially from an initial state. A propositional STRIPS planning instance is defined by a tuple $\langle S, A, I, G \rangle$, where:

- S is a finite set of ground atomic formulas describing the world state,
- A is a set of operators (or actions), where each operator has a set of preconditions, $pre \subseteq S$, an add list, $add \subseteq S$, and a delete list, $del \subseteq S$,
- $I \subseteq S$ is the initial world state, and
- $G \subseteq S$ is the goal world state.

The STRIPS system produced total-ordered plans, in which the steps are ordered with respect to every other step in the plan. Metrics for comparing plans include makespan, the number of actions in the plan, and cost, the sum of the cost of the actions in the plan. A *satisficing plan* is one that achieves the goal when executed from the initial state. Plans can be compared using their cost or makespan, among other metrics. An *optimal plan* is the satisficing plan that maximizes an objective function.

The Nets of Action Hierarchies (NOAH) planning system avoided the strong ordering of actions produced by total-ordered plans (Sacerdoti, 1975). The problem formulation is identical to the STRIPS formulation. The NOAH system planned how to achieve each part of the goal state individually, before merging the plans together into a single partial-order plan. Constraints between actions enforce a least commitment ordering between the actions. For example, if action a produces a precondition for action b, then a is constrained to occur before b. Partial-order plans can be transformed into total-order plans by identifying a step order that satisfies the ordering constraints between the actions. Such a transformation is equivalent to the job-shop scheduling problem (Graham, 1966). Total-order plans can be transformed into partial-order plans by removing unnecessary orderings, which can be achieved in polynomial time (Bäckström, 1993).

STRIPS and NOAH both used state space search, which consists of nodes representing possible states and edges representing actions. Graphplan introduced the plan graph data structure (Blum and Furst, 1997) that has two types of nodes and two types of edges. The node types are proposition nodes and action nodes,

while the edge types are precondition edges and effect edges. Precondition edges connect a proposition node to a corresponding action node. Effect edges connect an action node to the produced proposition node. The Graphplan algorithm iteratively extends the length of plans in the graph until a valid plan is produced and outputs a partial-order plan.

Another popular technique in planning is boolean satisfiability. Plans are derived from the creation of and solution to the satisfiability problem. If a solution is not found, then a plan with the constraints does not exist. An example of satisfiability based planning is the SATPLAN classical planning algorithm (Kautz and Selman, 1992, 2006). State space search, plan space search, constraint satisfaction, and plan graph search are used extensively in planning algorithms and many algorithms combine techniques to take advantage of the strengths of each.

STRIPS, NOAH, Graphplan, and SATPLAN all produce plans consisting of instantaneous actions for a world modeled as a set of binary-valued fluents. Some domains will fit into these constraints, but many real world domains require more expressive planners. The remainder of this section reviews four different classifications of planning, how they expand the STRIPS formulation used in classical planning, and why they are important for real world planning. Classical planning assumes that the world is fully observable and that actions execute deterministically. Probabilistic planning addresses the uncertainty in real world domains by removing one or both of those assumptions. Classical planning assumes that actions are executed instantaneously. Temporal planning considers time by allowing durative actions with effects and conditions that can occur before, during, and after execution. The STRIPS' world model uses propositional variables that only allow for true or false values. Continuous planning introduces variables necessary for representing world model variables with continuous domains. Plans can be formed by distributed systems to accomplish their task, but these individual task plans may not work when executed concurrently. Plan merging algorithms address this problem and find coordination plans to accomplish the assigned tasks. Finally, multi-agent planning extends STRIPS by explicitly considering multiple agents working together.

II.3.1 Probabilistic

The classical planning formulation assumes that all aspects of the environment are known and actions are deterministic. Uncertainty arises from many different sources (e.g., noisy sensors, environment, action execution). Probabilistic planning introduces uncertainty by incorporating probabilities into the action effects and initial states. These problems are related to the Markov Decision Process (MDP), which is defined by a tuple $\langle S, A, T, R \rangle$ where:

- S is a set of states,

- A is a set of actions,
- $T : S \times A \rightarrow \Pr(S)$ is a Markovian transition function giving the probability of transitioning from one state to another when performing some action, and
- $R : S \times A \rightarrow \mathbb{R}$ is an real-valued immediate reward for taking an action in some state.

The function R is a reward function or a cost function, depending on the application (Kaelbling et al., 1998).

The first application of a MDP in planning used policy iteration to derive incrementally better plans (Koenig, 1992, 1994). The probabilistic planning problem was defined by the tuple $\langle S, A, T, C, B, G \rangle$ where:

- S is a finite set of states,
- A is a finite set of actions,
- $T : S \times A \rightarrow \Pr(S)$ is a Markovian transition function giving the probability of transitioning from one state to another when performing some action,
- $C : S \times S \rightarrow \mathbb{R}^+$ is a positive real-valued cost function for transitioning from one state to another state,
- $B : Pr(S)$ defines the probability distribution that a state is the initial state, and
- $G \subseteq S$ defines the set of goal states.

Modifications to the theoretical MDP included the state space, S, and action space, A, being finite and introduced the concept of an initial state and a goal state. Each goal state has an additional action, called the stop action, which deterministically transitions to itself (i.e., $T(s, a, s) = 1$, where $s \in G$ and a is the stop action).

Several methods exist for identifying MDP policies, including value iteration, policy iteration, and linear programming. Finding an exact solution through linear programming can be achieved in polynomial time with respect to the size of the state space and action space; however, in practice the iterative methods are generally more efficient and find good approximations (Littman et al., 1995b). Reducing the complexity of finding a policy necessarily reduces the complexity of the planning problem.

Dean et al. (1993) simplified the MDP problem by removing states that are unlikely to be reached. Instead of formulating a policy for the entire state space, one formulates a policy for the reachable states. States included in the analysis are in the envelope and all states not included are considered to be a single OUT state, with high costs for reaching it. Policy quality and envelope size can be balanced based on time constraints. A hill-climbing algorithm for expanding the envelope size has also been tested (Tash and Russell, 1994).

Another issue with MDPs is the need for a transition probability for every state to every other state for each action, requiring a table of size $|S| \times |A| \times |S|$. Specifying each of these values is prohibitive for all, but the smallest problems. A Bayesian network can be used to specify the proposition independencies and calculate each value in the transition table (Boutilier et al., 1995).

The MDP formulation assumes the entire world model is observable, which is not always the case. The partially observable MDP (POMDP) introduces uncertainty (Kaelbling et al., 1998) and is represented as a tuple $\langle S, A, T, R, \Omega, O \rangle$ where:

- $\langle S, A, T, R \rangle$ describe a MDP,
- Ω is a finite set of world observations, and
- $O : S \times A \rightarrow \Pr(\Omega)$ is the observation function, the probability of making an observation after executing an action and transitioning to a state.

An agent must estimate the current world state given the observations and actions it has taken thus far. POMDPs are computationally intractable; therefore, approximation methods, such as value iteration or policy iteration are used (Hauskrecht, 2000).

Point-based POMDP solvers seek to reduce the size of the state space by only considering a subset of the belief states (Hsu et al., 2008; Shani et al., 2007; Smith and Simmons, 2012; Spaan and Spaan, 2004). Point-based value iteration (PBVI) algorithms use value iteration and belief state expansion in an anytime algorithm to compute solutions (Pineau et al., 2003a). Each iteration performs value iteration and expands the set of belief states. Each expansion step doubles the number of belief states by adding the reachable belief states that are furthest from the current set. The reduction in the belief state size allowed PBVI to find solutions for POMDPs that were orders of magnitude larger than the existing results (Littman et al., 1995a; Brafman, 1997; Poon, 2001). PBVI can be further modified to only consider those belief states that can be reached, while executing an optimal policy (Kurniawati et al., 2008). Knowing the states reachable from the optimal policy entails knowing the optimal policy, so this set must be approximated. Concrete applications of POMDP planning systems include health care (Pineau et al., 2003c; Hsiao et al., 2007; Hoey et al., 2010) and robotic control (Pineau et al., 2003b; Pineau and Gordon, 2007; Kurniawati et al., 2011).

PGraphplan and TGraphplan represent two Graphplan extensions for probabilistic domains (Blum and Langford, 2000). PGraphplan performs a forward chaining search to produce an optimal contingent plan, with the plan graph used to prune the search. Contingent plans allow for unexpected outcomes by conditioning future actions on the outcomes of past actions. TGraphplan performs a backward chaining search, exactly as in Graphplan, but returns the optimal trajectory and the probability of achieving the goal state successfully.

A trajectory is a sequence of actions and outcomes. An optimal trajectory is the trajectory with the highest probability of reaching the goal state when executed from the initial state.

Another method, replanning executes a deterministic planner and generates a new plan based on observing an unexpected action effect. Fast Forward is a forward state space search deterministic planning algorithm that uses a relaxed Graphplan problem as a heuristic (Hoffmann and Nebel, 2001). Fast Forward Replan was a probabilistic replanning algorithm that used Fast Forward as its deterministic planner (Yoon et al., 2007). Input was compiled into a deterministic problem by selecting the most likely outcome from each action as the deterministic effect for each action. If a less likely outcome occurred, then the system found a new plan using the new initial state. The PRP planner also used determinization and replanning (Muise et al., 2012). Weak plans, policies that when executed from an initial state reach a goal with non-zero probability, were developed and combined to form a strong cyclic plan, a closed policy that when executed achieves the goal from every reachable state. MDP-based replanning algorithms must replan based on observing an unexpected effect from an action. Belief state can be used to determine when to replan in POMDP-based algorithms (Brafman et al., 2014). When observations are made that are unexpected or provide a great deal of new information to the belief state, replanning is performed.

The Upper Confidence bounds applied to Trees (UCT) algorithm selectively samples actions (Kocsis and Szepesvri, 2006). UCT treats inner nodes of the search tree as separate multi-armed bandit problems, where the arms correspond to actions and payoffs correspond to rewards from executing an action. It was proven that the probability of optimal action selection converges to 1 given infinite samples. UCT was found to solve much larger problems than either the Asynchronous Real Time Dynamic Programming system (Barto et al., 1991) or PG-ID (Péret and Garcia, 2004), two anytime algorithms that perform selective action expansion.

The 2014 International Planning Competition (IPC) required participants to submit solutions that were applied to a standard set of input domains and problems. The planners were given a limited amount of time to select actions for each problem instance. The comparison metric was the sum reward for each problem instance. The Discrete Probabilistic Planning Track was broken into MDP and POMDP subtracks. The PROST algorithm won the MDP track (Keller and Eyerich, 2012; Keller and Helmert, 2013). PROST improved upon UCT by using outcome determinization techniques to initialize heuristics and generate a two-step lookahead. GOURMAND (Kolobov et al., 2012a,b) is based on the Labeled Real Time Dynamic Programming approach (Bonet and Geffner, 2003) for solving MDP planning problems. No single planner dominated all other planners.

The PODMP track was won by the Approximate POMDP Planning Toolkit (APPL). The APPL planner combined the Regularized Determinized Sparse Partially Observable Tree algorithm (R-DESPOT) (Somani et al., 2013), with a partially observable Monte Carlo planning (POMCP) (Silver and Veness, 2010). The

R-DESPOT algorithm used sparse scenario sampling to produce a policy that balances execution time with optimal policy estimation accuracy. If R-DESPOT takes too long, then the APPL planner resorted to using the POMCP algorithm, which used Monte Carlo sampling and a black box POMDP simulator to efficiently solve large POMDPs. The Korean Advanced Institute of Science and Technology (KAIST) planner used an approach similar to APPL, in that there is a primary algorithm that resorts to POMCP when necessary. The KAIST planner primarily uses the symbolic heuristic search value iteration algorithm (Sim et al., 2008). Heuristic search value iteration is a point-based value iteration algorithm that approximates the optimal value function with an upper and lower bound. Symbolic heuristic search value iteration uses algebraic decision diagrams (Bahar et al., 1997) to compute the upper and lower bounds of the value function without explicitly enumerating the states and observations of the POMDP.

Uncertainty is inherent in real world domains due to noisy sensors, imperfect actuators, and an inability to control or measure all aspects of the environment. Probabilistic planning approaches can produce plans capable of executing in uncertain situations, but these approaches do not address executing actions in the real world over an extended time interval.

II.3.2 Temporal

The classical planning formulation assumes that actions execute instantaneously. Temporal planning problems drop this assumption by adding durative actions. Durative actions occur over a time interval and may have effects and conditions at any point during that interval. The Planning Domain Definition Language (PDDL) introduced two durative actions in version 2.1: discretized and continuous (Fox and Long, 2003). Discretized actions have effects at the start of the action, the end of the action, or both, while continuous actions have effects throughout the action's execution.

Temporal planning languages provide for much more expressive domains than STRIPS, but this expressiveness is not always necessary. Cushing et al. (2007a,b) introduced the notion of required concurrency. The solution sets for problems with required concurrency include only plans requiring at least one pair of concurrently executing actions. Problems lacking required concurrency can be reduced to a classical STRIPS planning problem, solved using a classical planning algorithm, and transformed to a temporal plan. An example of required concurrency is the *single hard envelope problem*. The single hard envelope problem arises when there is a resource-producing action in which a resource is generated at the start of action execution and is removed at the end of action execution (Coles et al., 2009b). An example of a single hard envelope is using a match for light; light is produced in the world model at the start of the action and deleted from the world model at the end of the action.

Temporal planning also changes the metrics when comparing plans. For example, makespan must be

redefined. Using the classical definition of makespan, a plan with two actions executed sequentially has a higher makespan than a plan with a single action. However, a single action plan can require a longer time to execute than the two action plan; thus, plan makespan must be redefined to consider the concurrent actions and action execution times in temporal planning. Makespan in temporal planning is a function of when each action starts and how long each action takes. Given a temporal plan π, the makespan is

$$\text{makespan}(\pi) = \max_{a \in \pi}(start(a) + dur(a)), \tag{II.5}$$

where $start(a)$ is the start time of action a and $dur(a)$ is the duration of action a.

Temporal planning encourages concurrent execution of actions when makespan is used as a metric. Temporal constraints between actions are used to determine when actions can be executed concurrently and when they must be executed sequentially. Plans can be total-ordered, with constraints placed on every pair of actions, or partial-ordered, with constraints between only some of the actions. Allen introduced seven primitive relationships between time intervals over which actions can execute (Allen, 1984):

- DURING(t_1, t_2): time interval t_1 is fully contained in t_2,
- STARTS(t_1, t_2): time interval t_1 has the same start point as t_2, but ends before t_1,
- FINISHES(t_1, t_2): time interval t_1 has the same end point as t_2, but begins after t_2 begins,
- BEFORE(t_1, t_2): time interval t_1 occurs prior to time interval t_2, and they do not overlap,
- OVERLAP(t_1, t_2): time interval t_1 starts before t_2, and they overlap,
- MEETS(t_1, t_2): time interval t_1 ends when t_2 starts, and
- EQUAL(t_1, t_2): t_1 and t_2 are the same interval.

These relationships can be expressed as constraints between actions in a plan. Not all actions will follow one of these relationships, because actions can have no temporal constraints between them.

One data structure common in planning is the *plan space graph*. A plan space graph consists of nodes representing plans and edges representing modifications to the plan (e.g., adding actions, removing actions, changing action temporal constraints). The Universal Conditional Partial Order Planner (UCPOP) used plan space search and allowed conditional effects and universal quantification, but not durative actions (Penberthy and Weld, 1992). The Versatile Heuristic Partial Order Planner (VHPOP) extended UCPOP to support durative actions (Younes and Simmons, 2003). VHPOP modified plans by finding and fixing flaws in the plan. A flaw in a plan is an action that makes the plan nonexecutable or an represents an inefficiency in the plan.

VHPOP implemented novel flaw selection strategies (heuristics for selecting refinements to make a plan that fixes a flaw) along with existing flaw selection strategies (Peot and Smith, 1993; Joslin and Pollack, 1994; Schubert and Gerevini, 1995). Each of the flaw selection strategies had strengths and weaknesses, so all were used concurrently. The additive heuristic (Bonet et al., 1997) was also adapted for use in VHPOP (Younes and Simmons, 2002).

Extensions to Graphplan have also been applied in temporal planning. Temporal Graphplan (TGP) exploited the mutual exclusion structure used in Graphplan (Smith and Weld, 1999). TGP compressed the plan graph structure to require a single proposition node for each proposition and extended the mutual exclusions to work for durative actions with overlapping execution. LPG is an anytime algorithm using stochastic local search over temporal action graphs, a variation of plan graphs (Gerevini and Serina, 2002; Gerevini et al., 2003). Action nodes in temporal action graphs are marked with the estimated earliest time that the corresponding action will terminate and have ordering constraints between them. Proposition nodes contain estimates of the earliest time that the proposition will become true. Linear Programming and Graphplan (LPGP) used Graphlan followed by linear programming to develop plans (Long and Fox, 2003b). Domains were modified to use only instantaneous actions with durative actions compiled into a start action, an end action, and a duration action. A linear program representing the duration of each action was built alongside the plan graph. If the plan graph and the linear program both had solutions, then a plan for the original planning problem was derived from the solutions.

Progression and regression planning through the state space are the most popular approaches in temporal planning. TALPlanner used forward-chaining with temporal action logic to represent domain-dependent knowledge for guiding the search (Kvarnström et al., 2000; Kvarnström and Doherty, 2000). Temporal action logic is a language specification for reasoning over actions and state changes over time (Doherty et al., 1998). The OPTIC (Optimizing Preferences and TIme-dependent Costs) planner performed forward search and used mixed integer linear programming to accommodate soft constraints and preferences when plan quality was not directly determined by makespan (Benton et al., 2012). FLAP performs forward search over the plan space with multiple weighted heuristics (Sapena et al., 2013), while FLAP2 intelligently weighs the heuristics based on the problem being solved (Sapena et al., 2014). TP4 performed heuristic regression search to develop plans in domains with resources, while minimizing makespan (Haslum and Geffner, 2001). TP4 automatically derived an admissible heuristic from the problem instance.

Yet Another Heuristic Search Planner (YAHSP) was an influential progression state space planner (Vidal, 2004b,a) that used a relaxed plan based on a planning graph as a heuristic. YAHSP2 and YASHP3 (Vidal, 2011, 2014) used a critical path heuristic to build the relaxed plan and modified the heuristic state value to use the additive heuristic. These changes increased the planner's search efficiency at the cost of no longer

being able to handle required concurrency.

The TEMPO algorithm was based on lifted temporal states and used heuristics from decision epoch planners (Cushing et al., 2007a). Lifted temporal states contain only temporal constraints between actions and not exact start times, where the decision regarding when actions are to be executed is delayed until it is determined which actions are to be executed. Decision epoch planners eagerly make decisions concerning when actions will execute. TP is a modified version of TEMPO that compiles some temporal actions into classical planning actions (Jiménez et al., 2015). TPSHE takes the approach of TP a step further and compiles all temporal actions into classical planning actions (Jiménez et al., 2015). TPSHE can handle required concurrency in the form of single hard envelopes, whereas TP cannot.

Temporal domains can also include uncertainty. Some durative actions, such as movement, can have durations not known during planning. Cimatti et al. (2015) developed a forward-chaining planner that can guarantee plan execution in domains where action duration is only known within some bounds.

Factored planning approaches divide the planning problem into smaller individually solved problems that are merged into a single solution (Brafman and Domshlak, 2006). SGPlan used a partition-and-resolve strategy with progression planning to generate plans (Chen et al., 2006; Hsu et al., 2006a,b, 2007). The partition-and-resolve strategy intelligently splits the problem goals and develops plans for each subproblem independently before resolving global constraints between the subplans. SGPlan also supported all the PDDL 3.0 features. A similar approach is taken by the Divide-and-Evolve planner (Bibaï et al., 2010a,b; Dréo et al., 2011), that can embed any existing planner to use for planning each of the smaller problems.

The deterministic temporal track of the 2014 IPC included six planners and was won by YAHSP3, an extension of YAHSP2 (Vidal, 2014). The Temporal Fast Downward planner (TFD) (Eyerich et al., 2012) is based on the Fast Downward classical planner (Helmert, 2006) and finished in second. The only SAT-based planner to participate was ITSAT (Rankooh and Ghassem-Sani, 2015) and it came in third. As with the probabilistic track, the winner did not beat the second place planner for every tested domain. YAHSP3 beat TFD in domains that did not require concurrency. All planners failed to solve a problem in at least one domain.

Real-world actions do not necessarily occur instantaneously as the classical planning model assumes. Temporal planning considers time by modeling durative actions and concurrent action execution. Real-world durative actions can also have effects throughout their execution, which requires continuous planning.

II.3.3 Continuous

The world state in the STRIPS planning formulation is based on boolean variables. Allowing variables to only take values from a finite domain prevents describing many real world values, such as fuel level, energy,

or distance. Continuous planning considers continuous fluents. The PDDL planning language (PDDL 2.1) introduced numeric fluents and durative actions with continuous effects over the execution of an action (Fox and Long, 2003). Examples of continuous effects include power usage, as provided by changing battery or fuel tank levels. The PDDL+ extension allows more expressive continuous effects in the domain by both agents and the environment (Fox and Long, 2006).

One of the first planners capable of handling continuous effects was Zeno, a plan space search least-commitment planner that supported continuous change, deadline goals, metric preconditions, and metric effects (Penberthy and Weld, 1994). At each step, a subgoal was selected and an action was either selected to support or added to support the subgoal. A drawback of Zeno was its inability to handle concurrent continuous change. Common real world actions, such as charging a battery through solar panels while simultaneously draining the battery to move, cannot be executed concurrently in Zeno.

State space search is a popular technique in continuous planning. The COntinuous LINear process planner (COLIN) (Coles et al., 2009a, 2012a) was based on CRIKEY3 (Halsey et al., 2004; Coles et al., 2008), a temporal planner applicable to domains and problems that required concurrency. COLIN extended CRIKEY3 to handle continuous linear effects from durative actions. COLIN used a linear program to express both temporal constraints and linear processes, whereas CRIKEY3 used a simple temporal network to express temporal constraints. After COLIN came POPF, a progression state space planner that handled durative actions and continuous linear change (Coles et al., 2010). POPF used a partial order approach to limit the commitments made early in the planning process; thus, reducing the need to backtrack due to reaching inconsistent states. Actions affecting numerical fluents remained totally ordered in order to reduce the problem complexity. POPF2 extended POPF by analyzing the problem structure for numerical behavior patterns to exploit during search (Coles et al., 2011).

IxTeT extends constraint satisfaction based planning to handle continuous linear change (Laborie and Ghallab, 1995; Lemai and Ingrand, 2004; Trinquart and Ghallab, 2014). A constraint satisfaction problem developed along with a candidate plan is checked for consistency when the candidate plan is found to accomplish the goal. If the constraint satisfaction problem is valid, then the candidate plan is a valid plan, otherwise the algorithm must backtrack, which can cause performance issues when the backtracking point is difficult to determine. The satisfiability-based dReal planner compiles problems from PDDL+, an extension of PDDL, into first-order logic (Bryce et al., 2015). dReal uses a boolean satisfiability solver to find literals satisfying each boolean constraint followed by an Interval Constraint Propagation based branch-and-bound solver to refine the intervals on the numeric constraints bounded by the chosen literals. A δ-satisfiability modulo theory solver determines a δ-plan tube based on the intervals for the numeric constraints. δ-plan tubes specify a plan in which action execution times can vary by as much as δ and remain valid (Fox et al., 2006b). Plan tubes

allow for physical systems that cannot execute plans exactly as prescribed to accomplish their goal. Unlike the other planners, dReal supports nonlinear continuous change to variables.

The combinations of planning techniques have resulted in continuous planning algorithms. Sapa was a forward-chaining planner that used domain-independent heuristics and handled continuous resource constraints and deadline goals (Do and Kambhampati, 2001, 2003). Plan graph techniques supplemented forward-chaining by generating heuristics for cost and makespan. After finding a plan, a linear time post processing step was applied to increase plan flexibility by removing unnecessary constraints on action execution times.

Continuous planning introduces action effects over the action duration and variables in a continuous domain. The expressiveness added to the planning language allows domains that more closely approximate real world problems. All of the discussed planning techniques handle linear variable change, but algorithms that handle nonlinear change are uncommon. The dReal planner is the only presented planner that supports nonlinear change. The next section reviews merging multiple plans into a single global plan.

II.3.4 Plan Merging

The plan merging problem integrates multiple plans into a single coordination plan. The problem consists of an initial state, I, a set of tasks, $v \in V$, and a set of plans to complete those tasks ($\Pi = \{\pi_1, \pi_2, \ldots, \pi_{|V|}\}$). A solution to the problem is a coordination plan, π_V, which completes all the tasks in V when executed from I. An agent's coordination plan does not interfere with the other agents' coordination plans.

Merging plans allows agents to work together and benefit from the unused product of other agents' actions, also called side products (de Weerdt et al., 2003). An example of a side product are the products produced by petroleum refining. The refining process produces many products, some of which will not be used by the agent performing the refining. Agents can also take advantage of products produced and used, but not consumed by other agents (Foulser et al., 1992), which is common in domains requiring setup and restore operations. An example is grocery shopping, where instead of driving from home to a store once for each item, an intelligent agent drives to the store once and purchases all of the desired items. Agents sacrifice independence in constructing plans that rely on side products or nonconsumed products, but gain efficiency by reducing the number of actions required to complete a plan.

One approach to plan merging is to treat it as a plan space search problem in which incremental changes are made to the plan until a valid plan is found (Cox and Durfee, 2005). The plan space search algorithm merged each agent's plan together and used a branch-and-bound search to resolve flaws and eliminate redundant steps in the combined plan. The algorithm optimized the solution for the number of steps in the plan shared by the agents. Performance increased with a decreasing degree of coupling between the agents and tasks.

Plan merging can also be interleaved with each agent's planning and not as a distinct step. The Plan-Merging Paradigm used a distributed global plan, where each agent only tracked its own coordination plan (Alami et al., 1995). When assigned a task, an agent used their own coordination plan as a starting point to develop a new plan to accomplish the newly assigned task, after which the agent requested permission to perform a plan merge. Upon receiving permission, the agent determined if its plan interfered with the coordination plans of the other agents and modified its plan, if needed, to ensure that the plan did not interfere with any of the other agents' plans. The new coordination plan replaced the existing plan for all robots.

Plan merging is necessary when distributed agents develop plans individually in order to ensure agents do not interfere with one another. Distributed constraint optimization can be used for plan merging after all the agents have developed their plan (Cox et al., 2005). A constraint satisfaction problem was created from the plans to be merged. Merge flaws, identical steps in multiple plans, were represented by constraint variables with a domain of *ignore* or *merge*. A value of *ignore* kept the identical steps in the plan, while a value of *merge* eliminated all, but one of the identical steps. Constraint variables for causal link threats in the merged plan had three possibilities in their domain: ignore, promote, and demote. Ignoring a causal link threat can create an invalid plan, but the constraint variable representing the threat can be assigned the ignore value, if the threat was eliminated by resolving a different threat or flaw.

Temporal fusion merges temporal plans and their associated temporal constraints (Allouche and Boukhtouta, 2010). Temporal plans were modeled as a simple temporal network with nodes for action start time points and end time points, while edges represented the temporal constraints between the time points. Plans with shared actions required temporal constraints be merged in the final plan. The merged constraint was the intersection of the two original constraints. If the intersection of the constraints was empty, then the two plans were not merged due to inconsistent constraints.

One of the benefits of performing plan merging is the independence of how the input plans are formed. The Plan Merger by Reuse algorithm performs plan concatenation followed by analysis in order to merge the plans into a single coordination plan (Luis and Borrajo, 2014). Agents are assigned goals and plan for their goals individually. These individual plans are merged into a combined plan. If the tasks are loosely coupled, then the combined plan is more likely to be a coordination plan, than if the tasks have tighter coupling. More tightly coupled domains require a separate planner, LPG-ADAPT, to resolve the flaws in the invalid merged plan (Fox et al., 2006a). Most of the relevant actions of the final coordination plan are present in the individual plans, so that the necessary actions to complete the plan are already known.

Plan merging algorithms allow agents to use their own planning algorithm, while maintaining a global coordination plan for the distributed system. The final section in planning for real world applications is multi-agent planning.

II.3.5 Multi-Agent

Classical planning does not consider agents that will execute a plan. Multi-agent planning considers the agents that will execute the plan and the action execution constraints on the agents. Multi-agent systems have several advantages over single agent systems, including redundancy and multiple action execution in different locations; however, these advantages come at the cost of increased problem complexity. Multi-agent planners must consider the interference, both negative and positive, that will occur when agents are executing the plan.

Multi-agent planning has applications in distributed sensor networks (Durfee and Lesser, 1991; Decker and Lesser, 1992). Sensor data processing can be distributed throughout the network, so that each individual processor is responsible for only a local area of sensors. Distributed sensor processors have advantages over a single centralized processor, including limited raw data transmission distance and redundancy. Capitalizing on these advantages requires collaboration and planning to determine the subset of data each agent will process and the level of abstraction of the processed data that will be transmitted to other agents.

Planning for multiple agents requires cooperation between the agents when developing a plan. Agents in cooperative refinement planning form a shared plan by iteratively performing local planning and global plan refinement (Torreño et al., 2012, 2014a,b). Each agent develops a local plan by expanding upon a centralized base plan. Each agent individually selects the most promising plan, based on their individual information before the agents jointly select the plan to be used as the next base plan. The algorithm terminates when every agent confirms that the centralized base plan accomplishes the assigned task.

Multi-agent planning can also be accomplished through constraint satisfaction techniques. Brafman and Domshlak (2008) applied constraint satisfaction and factored planning to produce plans for multi-agent problems, which was extended to use distributed constraint satisfaction solvers (Nissim et al., 2010). Another satisfiability based planner, μ-SATPLAN extended SATPLAN to multi-agent planning for two distinct classes of problems: multi-agent coordinated actions and multi-agent assistance actions (Dimopoulos et al., 2012). The multi-agent coordinated actions problem requires individual agents to coordinate their actions in order to avoid negative interactions and promote positive interactions in their individual plans. This problem can be solved individually, followed by plan merging to develop a single coordination plan to accomplish the tasks. The multi-agent assistance actions problem involves required cooperation between agents to accomplish a task. This problem requires that the agents required to cooperate are considered simultaneously when planning for the cooperative action.

Multi-agent planning can be approached from a game theory perspective. Non-cooperative planning treats each agent as self-interested and each agent uses a single-agent planning algorithm (Jonsson and Rovatsos, 2011). A global plan is developed iteratively as each agent selects its own best-response action to the current

plan. Agents are only limited in ensuring that other agents' plans remain executable.

Factored planning splits goals into subgoals and plans for each subgoal independently. Multi-agent planning is a natural fit when the tasks are independent and the set of agents assigned to the tasks are disjoint. Ephrati and Rosenschein (1993, 1994) proposed using sub-plans as heuristics for global planning. Agents rely on an a priori division of the goal into subgoals, from which the agents make their own plans. A heuristic function is derived from the subplans and is used in the global plan search.

Individual agents in multi-agent systems can have properties that they do not desire to broadcast to the other agents. Agents in privacy-preserving algorithms only advertise the properties and actions they want others to know. Nissim and Brafman (2014) apply distributed heuristic forward search techniques to privacy-preserving multi-agent planning. Each agent searches the state space and sends state information to the agents that have advertised actions executable from the state. An example application of this technique is supply chain management in which companies advertise their prices for parts, but not their costs.

Combinations of techniques can be applied in one algorithm to take advantages of each technique. The Threaded Forward-Chaining Partial Order Planner combines the performance of forward-chaining with the flexibility of partial-order planning (Kvarnström, 2011). Partial ordering between actions by different agents is more important for flexible execution than a partial ordering between actions executed by the same agent. Each agent can execute only a single action at a time, so each agents' set of actions can be totally ordered within the set of actions with minimal loss of flexibility. Totally ordering the actions that each agent executes allows for more descriptive states during planning.

Multi-agent planning considers how multiple agents will work together to accomplish goals. Redundancy and increased execution performance can be achieved by using multiple agents to solve a problem, in comparison to single agent problems.

II.3.6 Summary

Each of these planning categories represent an extension to the classical planning problem for application to real world domains. Probabilistic planning addresses the uncertainty from noisy sensors and actuators inherent in real world environments. Temporal planning introduces durative actions to the domain modeling, while continuous planning expands the possible domain of state variables to include continuous values. Distributed agents and coalitions can form their own plans to accomplish their goal and develop a coordination plan using plan merging algorithms. Multi-agent planning considers the agents that will execute the plan actions and any associated constraints. Planning systems must handle these attributes in order to be effective in real world domains and problems.

II.4 Task Allocation and Planning

Task allocation and planning are closely coupled problems, but current research into the interaction between the two problems is minimal. Planning affects task allocation through the developed plans, as the plan constrains the set of agents available by requiring agents to perform actions at specific times. Task allocation affects planning by determining which agents are available when developing a plan. If an agent is not allocated to a task, then the planning algorithm will not use the agent to develop a plan.

Flexible coordination and communication is important for addressing uncertainty. Shell for TEAMwork (STEAM) was a cooperative architecture based on joint intentions between the agents (Tambe, 1997). STEAM builds a hierarchical structure of joint and individual intentions for the agents. The architecture explicitly supports representation and reasoning with team goals, team plans, and team states. STEAM team members monitor the team and individual performance to autonomously determine if the team needs to reorganize. Teamcore was a successor to STEAM (Tambe et al., 2000) and focused on heterogeneous multi-agent teams. Adaptive autonomy, adaptive execution, adaptive monitoring, and adaptive information delivery were found to be crucial during execution for heterogeneous multi-agent systems.

Multi-agent Markov decision processes have been applied to heterogeneous agent task allocation in a grid world (Claes et al., 2015). The derived policy provided guidance regarding the tasks and locations to which each agent navigated. Mixed integer linear programs were applied to versions of the task allocation problem extended to include path planning and scheduling (Flushing et al., 2016), and task temporal constraints (Koes et al., 2005; Ramchurn et al., 2010). An agent did not contribute towards the task's required workload until it reached the task location. Aspects of planning considered by the algorithm included optimal travel time between tasks and time required for the coalition to actually complete the task. None of these algorithms conduct task planning from a traditional perspective. The tasks had a minimum required workload to be fulfilled by the agents for the task to be considered complete. Agents capable of reaching a task location were assumed to contribute to the task; thus, a sequence of actions to be executed by the agents for completing the task was not developed.

Chance-constrained task allocation introduces uncertainty into the task allocation and planning problem (Ponda et al., 2012). Agents develop plans for single agent tasks and estimate their plan execution duration. Agent allocation utility is a function of the agent allocated to the task, when the agent will execute the task, and a predefined model of problem uncertainty. Each agent solves their own version of the problem with their own risk bounds and communicates with the other agents to ensure that system constraints are satisfied, such as a single agent being allocated to each task. A similar probabilistic task allocation and planning problem encodes the probability of an agent's ability to apply a particular partial state transition (Zhang et al., 2015).

The partial state transitions can be applied sequentially to determine the agent(s) most likely to be capable of transitioning from the initial state to a desired partial state. The transitions can be used for coalition formation and as heuristic landmarks during planning.

Auction style methods are popular in coalition formation, but typically grant exclusive ownership of tasks. Exclusive task ownership can be detrimental when agents fail to complete their task and no method for informing the other agents of this failure exists. A method based on that used by bounty hunters and bail bondsmen allows for nonexclusive execution of tasks by agents (Wicke et al., 2015). Following the bail analogy, agents are termed as bounty hunters and auctioneers as bail bondsmen. The bail bondsmen increase the value of each task until it has been completed. Agents commit to a task and announce to the other agents that they have committed to it. Agents can plan how they can complete each of the available tasks, but they can only commit to a single task. Multiple agents may commit to a task, but they cannot collaborate. Agents receive the task value only upon completing the task. If an agent fails to complete a task, then the system easily adapts by incentivizing other agents to complete the task through increasing task value.

One combination that has been extensively studied in the literature is multi-robot task allocation and path planning to the task location (Barrientos et al., 2011; Woosley and Dasgupta, 2013; Moon et al., 2013; Turpin et al., 2013; Zhu et al., 2013; Turpin et al., 2014; Ma and Koenig, 2016). The task allocation and path planning problem is for single-agent tasks executed by single-task robots. Simultaneously considering the task allocation and the path to the task problem allows for collision-free trajectories to be developed more efficiently than if the two problems were considered separately. One application incorporates two agents that swap tasks when a collision is detected (Turpin et al., 2013, 2014). The two new trajectories for the agents are guaranteed not to collide with one another. These approaches work well for problems with a one-to-one mapping of agents to tasks, but multi-agent tasks will require more agents and more complex task allocation and path planning schemes.

The online version of multi-robot task allocation and path planning has been studied using a search and destroy problem with attack UAVs (Kim et al., 2014). A plan can be developed offline to determine an optimal search pattern for the UAVs to find their mobile targets, but the location of the targets is unknown and the decision of which UAVs will perform the attack must be made online. A distributed probabilistic approach considered the path each UAV needed to take to reach the target, the attack capability of the UAV, and the probability of destroying the target.

These algorithms and frameworks show that some aspects of task allocation and planning can be combined. Coupling task allocation with domain independent planning will improve planning results by allowing the two problems to inform each other.

CHAPTER III

Formal Definition and Experimental Domains

This chapter presents the formal problem definition, followed by the four experimental domains and how they map to the presented problem definition.

III.1 Formal Definition

The presented tools are for planning multi-task robot, multi-robot task, instantaneous allocation problems (Gerkey and Matarić, 2004). This Hybrid Mission Planning with Coalition Formation (HMPCF) problem couples coalition formation with planning to facilitate solving complex problem instances with heterogeneous multi-task robots executing multi-robot tasks.

Definition III.1 (Hybrid Mission Planning with Coalition Formation)**.** *The* hybrid mission planning with coalition formation *problem is defined as a tuple,* $\langle S, I, \Phi, A, V, C \rangle$*, where:*

- S *is the state space,*
- I *is the initial state,*
- $\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$ *is the grand coalition of agents,*
- $A = 2^{\Phi} \rightarrow 2^{Act}$ *is the coalition-action mapping, where* Act *is the set of all possible actions,*
- $V = \{v_1, v_2, \ldots, v_m\}$ *is the set of tasks, and*
- $C = \langle Cap, C_{\Phi}, C_V \rangle$ *is the capability vector, coalition capability mapping, and the task capability mapping.*

The hybrid state space, S, includes boolean, discrete, and continuous variables. A state, s, is an assignment of each state variable to a value in its associated domain. The initial state, I, is the environment state at the beginning of the mission.

The grand coalition, Φ, is the set of agents in the problem. A coalition, $\Phi_i \subseteq \Phi$, is any non-empty set of agents. The coalition-action mapping, A, maps each possible coalition, $\Phi\prime \subseteq \Phi$, to a set of actions, $\{a_1, a_2, \ldots\}$. An action is modeled as a tuple, $\langle \Phi_{exec}, \mathit{eff}, cond, dur \rangle$, where:

- Φ_{exec} is the executor coalition, the set of agents executing the action,
- $cond = \langle cond_{\vdash}, cond_{\leftrightarrow}, cond_{\dashv} \rangle$ is the action state constraints that must be satisfied at the beginning, during, and at the end of action execution, respectively, and

- $\mathit{eff} = \langle \mathit{eff}_{\vdash}, \mathit{eff}_{\leftrightarrow}, \mathit{eff}_{\dashv} \rangle$ is the action effects for state variable transitions applied to the state at the beginning of, during, and at the end of action execution, respectively,
- dur is a constraint on the duration of the time interval over which the action can execute.

The executor coalition, Φ_{exec}, for an action, a, is the set of agents that execute a. If Φ_{exec} is a singleton coalition consisting of a single agent, then a is a single-agent action. If Φ_{exec} includes more than one agent, then a is a joint action executed by multiple agents. Action state constraints on boolean and discrete variables specify the value the variable must hold, while constraints on continuous variables specify the interval to which the variable's value must belong. Each constraint must be a function of a single variable. Action state constraints can be specified as applying at the beginning, during, or end of action execution, with $cond_{\vdash}(a), cond_{\leftrightarrow}(a)$, and $cond_{\dashv}(a)$ being the set of all action state constraints that must be satisfied at the beginning, during, and end of executing a, respectively. Action effects can be applied to the state at the beginning of action execution, during action execution, or at the end of action execution. Instantaneous effects (those at the beginning or end of action execution) on boolean or discrete variables specify the new value of the variable, while instantaneous effects on continuous variables can specify a new value or an instantaneous addition or multiplication of the value. Durative effects (during action execution) are on continuous variables only and specify the linear change in variable value as a function of time. The sets of effects applied to the state as a result of executing an action a at the beginning, during, and at the end of executing a are $\mathit{eff}_{\vdash}(a), \mathit{eff}_{\leftrightarrow}(a)$, and $\mathit{eff}_{\dashv}(a)$, respectively. The action duration constraint, dur, is the interval to which action duration must belong, and must be positive.

The task set, V, is a set of tasks, all of which must be satisfied in a solution. Each task, $v \in V$, is modeled as a set of goal state constraints, $cond(v)$. A task, v, is satisfied in a state, s, if and only if all of v's goal state constraints are satisfied in s.

The capability vector, $Cap = [Cap_1\ Cap_2\ \ldots\ Cap_j]$, is the vector of coalition formation capabilities used in the problem. A *capability* is a high-level abstraction of the actions executable by agents and the goal constraints required by tasks, and it can follow either the resource model or service model of coalition formation, discussed previously in Chapter II.2. The coalition capability mapping, C_{Φ}, is a mapping of each agent to a capability available vector. The elements of a capabilities available vector are non-negative values, with at least one non-zero element. Each agent, ϕ, has a capabilities available vector, Cap^{ϕ}. For example, if Cap has five elements and ϕ has two of Cap_3 and three of Cap_5, then $Cap^{\phi} = [0\ 0\ 2\ 0\ 3]$, where Cap^i_j is the amount of Cap_j that entity i (agent or coalition) has at its disposal. Each coalition, Φ, has a capabilities available vector, Cap^{Φ}, equal to the sum of the capability available vectors of Φ's constituent agents, $Cap^{\Phi} = \sum_{\phi \in \Phi} Cap^{\phi}$. The capabilities are a function of the actions each coalition can execute. The

task capability mapping, Cap^V, is a mapping of each task to a capability required vector. The elements of a capability required vector are non-negative reals, with at least one non-zero element. For example, if Cap has five elements and v requires one of Cap_2 and two of Cap_3, then $Cap^v = [0\ 1\ 2\ 0\ 0]$, where Cap^i_j is the amount of Cap_j required to satisfy i. The task capabilities required vectors are a function of I and $cond(v)$.

A *plan*, π, is a set of *action execution steps*. An action execution step consists of an action, a start time to begin executing the associated action, and the duration of the action. An *executable plan* is a plan for which the action steps are executed validly. An action step is executed validly if all the associated action's state constraints are satisfied. Executing the action steps in an executable plan transitions the environment from the initial state, I, to an end state, s_{end}, achieved after all action steps have completed execution. A solution to the problem is a *satisficing plan*, a executable plan in which s_{end} satisfies the goal state constraints of each task, $v \in V$. A coalition is an *executable coalition* if a satisficing plan has been derived for the coalition to complete its task. A *nonexecutable coalition* is a coalition for which a satisficing plan has not been derived that allows the allocated coalition to complete its task. A *quality function*, $Q : \Pi \rightarrow \mathbf{R}$, provides an objective indicator of plan quality. The two quality functions used in this dissertation are temporal makespan and number of action executions, but any function mapping a plan to a real number can be used. Temporal makespan is the time required to execute the plan:

$$makespan(\pi) = \max_{s \in \pi}(start(s) + dur(s)),$$

where π is a plan and s is an action execution step in π. The *optimal plan* for a problem is the satisficing plan which maximizes Q.

III.2 Experimental Domains

Four experimental domains are presented. The first three, Rovers, Blocks World, and Zenotravel, are modified versions of existing popular planning domains. The fourth, First Response, is a novel domain presented as an example of a real world problem this research seeks to address. Each domain is implemented in the Multi-Agent Capabilities and Planning Domain Definition Language (MACPDDL), a new domain description language extending the PDDL 3.1 language and created specifically for this research. The features added by MACPDDL support coalition formation, task allocation, and explicit agent handling, and are detailed in Appendix A. The MACPDDL descriptions of each domain are presented in Appendix B.

```
1 (:durative-action communicate-soil-data
2   :executor (?r - rover)
3   :parameters (?l - lander ?s - waypoint ?x - waypoint ?y - waypoint)
4   :duration (= ?duration 0.5)
5   :condition
6     (and
7       (over all (at_rover ?r ?x))
8       (over all (at_lander ?l ?y))
9       (over all (have_soil_analysis ?r ?s))
10      (over all (visible ?x ?y))
11      (at start (channel_free ?l))
12    )
13  :effect
14    (and
15      (at start (not (channel_free ?l)))
16      (at end (channel_free ?l))
17      (at end (communicated_soil_data ?s))
18    )
19 )
```

Figure III.1: MACPDDL action implementation for a rover to communicate soil data to the central lander.

III.2.1 Rovers

The Rovers domain has been used for several iterations of the IPC (Long and Fox, 2003a). The domain models planetary rovers navigating between waypoints, collecting different classes of scientific data at a subset of waypoints, and communicating the data back to the central lander. The five classes of scientific data are soil analysis, rock analysis, high-resolution imagery, low-resolution imagery, and color imagery. Each rover can independently navigate a subsection of the environment and collect a subset of the classes of scientific data, but only one rover at a time can communicate data to the central lander. Rock analysis is required at a subset of waypoints and soil analysis is required at a subset of waypoints. Rovers must be at a waypoint in order to perform rock or soil analysis and must be equipped for the analysis. Up to three types of imagery data can be collected at each waypoint. A rover must have the correct camera type and the target waypoint must be visible in order for the rover to collect imagery data for the target waypoint.

The state space contains only boolean variables and describes waypoint connectivity, waypoint visibility, rover scientific tools, data collection types and location, central lander location, and communication channel capacity. Each action has a fixed duration. The domain's capability model corresponds to the classes of scientific data being collected. Each rover's capabilities offered vector is a function of the tools available to the rover. The goal is subdivided into a task for each class of scientific data, e.g., all the state constraints concerning rock analysis are grouped into a single task. The capabilities required vector for each task corresponds to the types of scientific data collected for the task.

An example action implementation for a rover communicating soil data is given in Figure III.1. The action is executed by a single rover (line 2) and has four parameters (line 3): the lander receiving the data

```
1  (:task soil_analysis
2    :services (
3      soil - 2
4    )
5    :goal (and
6      (communicated_soil_data waypoint5)
7      (communicated_soil_data waypoint44)
8      (communicated_soil_data waypoint54)
9      (communicated_soil_data waypoint29)
10     (communicated_soil_data waypoint45)
11     (communicated_soil_data waypoint18)
12     (communicated_soil_data waypoint67)
13   )
14 )
```

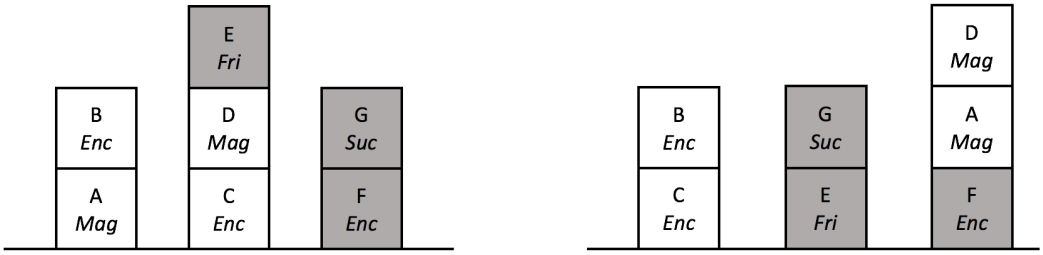
Figure III.2: MACPDDL description of a soil analysis task.

(?l), the waypoint from which the soil analysis was collected (?s), the rover's current location (?x), and the lander's location (?y). The action duration is half of a time unit (line 4). Executing the action requires the rover remain stationary during action execution (line 7), the lander is at the parameterized waypoint (line 8), the rover has the required soil analysis data (line 9), the rover has line of sight from its current location to the lander's location (line 10), and the communications channel is free (line 11). The effects of the action are that the communications channel is occupied for the action execution duration (line 15 is the rover occupying the channel and line 16 is the rover releasing the channel) and the soil data is communicated to the lander (line 17). An example task for soil analysis expressed in MACPDDL is presented in Figure III.2. The task uses the service model of coalition formation and requires two rovers capable of soil analysis (lines 2-4). The goal constraint requires soil data from different waypoints be communicated to the lander (lines 6-12). Each communicated soil data variable becomes true by executing an appropriate soil communication action (see line 17 in Figure III.1).

III.2.2 Blocks World

The modified blocks world domain requires that heterogeneous robotic arms manipulate stacks of heterogeneous blocks on a table of finite size. Each arm has a subset of available end effectors available and each block requires a specific end effector. A block can be manipulated by an arm if and only if the arm has the block's required end effector. While blocks have the same dimensions, blocks can be either single- or double-weight. Single-weight blocks can be manipulated by a single arm with the required end effector, while double-weight blocks require two arms, each with the required end effector, in order to be manipulated. The block stacks rest on a table with only enough space for a finite number of block stacks. The goal state is a rearrangement of the blocks from the initial state into a specified set of block stacks.

The state space, S, includes both boolean and continuous variables. The boolean variables describe the



(a) Initial state (b) Goal state

Figure III.3: Example states with the double-weight blocks shaded and required end effector for each block in italics.

block stacks, each block's required end effector type, which block each arm is holding, and each arm's available end effectors. The continuous variables describe the height of each arm and block, the number of blocks on the table, and the table capacity. The domain of the continuous variables is non-negative integers, which is not continuous; however, modeling the variables as continuous simplifies the state model by not requiring all possible values to be enumerated and ordered. The initial state, I, is an assignment of a value to each variable in the state space. As a partial example, the middle stack in the example initial state in Figure III.3a is expressed by assigning the value true to the following variables: $(onTable\ C)$, $(onBlock\ D\ C)$, $(onBlock\ E\ D)$, $(requires\ C\ encompass)$, $(requires\ D\ magnetic)$, and $(requires\ E\ friction)$. Each stack of blocks in the goal state corresponds to a task. The example goal state in Figure III.3b is divided into three tasks: v_C, v_E, and v_F. v_C is the stack with C on the bottom and the goal state constraints for v_C are satisfied when C is on the table and B is on C, i.e., when $(onBlock\ B\ C)$ and $(onTable\ C)$ are both true.

The grand coalition, Φ, is the set of arms executing actions. The actions are the up and down arm movement and block manipulation. A coalition, Φ_i, can execute any actions that are part of its action set, $A(\Phi_i)$. The action set is derived from the list of MACPDDL descriptions. An example MACPDDL action implementation of arm a picking up a single-weight block b_1 off of block b_2 using end effector e is presented in Figure III.4. An example element of a coalition action set derived from the MACPDDL description in Figure III.4 is $(pickupSingleBlockOnBlock\ arm_1\ block_B\ block_A\ magnetic)$. The action has a constant duration of 1 time unit. Conditions and effects are derived by the actions and parameters. The conditions on the start of the action, $cond_{\vdash}$, are that arm_1 is empty, $block_B$ is clear, $block_B$ is on $block_A$, $block_B$ requires a $magnetic$ end effector to move, and that arm_1 has a $magnetic$ end effector. The constraints during action execution, $cond_{\leftrightarrow}$, is that arm_1 remains at the same height as $block_B$. No end of action constraints, $cond_{\dashv}$, are given. The action has three beginning of action effects, $eff_{\vdash}$, and they are that arm_1 is no longer empty, $block_B$ is no longer clear, and $block_B$ is no longer on $block_A$. No continuous effects are applied as a result

```
(:durative-action pickup-single-block-on-block
  :executor (?a - arm)
  :parameters (?b_1 - single_block ?b_2 - block ?e - effector)
  :duration (= ?duration 1)
  :condition
    (and
      (at start (empty ?a))
      (at start (clear ?b_1))
      (at start (on_block ?b_1 ?b_2))
      (at start (requires ?b_1 ?e))
      (at start (has_effector ?a ?e))
      (over all (= (arm_height ?a) (block_height ?b_1)))
    )
  :effect
    (and
      (at start (not (empty ?a)))
      (at start (not (clear ?b_1)))
      (at start (not (on_block ?b_1 ?b_2)))
      (at end (clear ?b_2))
      (at end (holding_single ?a ?b_1))
    )
)
```

Figure III.4: MACPDDL action implementation for an arm to pick up a single-weight block off another block.

```
(:services
  arm_1 - encompass magnetic friction
  arm_2 - friction
  arm_3 - magnetic suction
  arm_4 - encompass suction
)
```

Figure III.5: MACPDDL agent capabilities specification.

of executing the action. Two end of action effects, $eff_{\dashv}$, are specified, $block_A$ is clear and arm_1 is holding $block_B$.

The capability vector for the blocks world domain corresponds to the end effector types: suction, friction, magnetic, and encompass. The capabilities offered vector for each arm is a function of the end effectors available to the arm. An example capability specification for agents is presented in Figure III.5. The example includes four agents using the service model of coalition formation. Each agent is listed followed by the services it offers. arm_1 has a friction end effector, a magnetic end effector, and an encompass end effector, corresponding to the capability vector $[0\ 1\ 1\ 1]$.

Double-weight blocks require twice the capabilities of single-weight blocks, because manipulating double-weight blocks requires two robotic arms. The capabilities for each stack are a function of two sets of blocks, the blocks in the goal stack and the blocks that must be manipulated to access the blocks in the goal stack. For example, the capabilities required vector for v_E is a function of E and G, because they are the blocks in the goal stack and there are no other blocks above E and G in the initial state. E requires two suction

```
(:task v_C
  :services (
    friction - 2
    magnetic - 1
    encompass - 1
  )
  :length 4
  :goal (and
    (onTable C)
    (onBlock B C)
  )
)
```

Figure III.6: MACPDDL action description of blocks world task in example goal state.

```
0.000: (pickup-double-block-on-block arm1 arm2 blockE blockD friction) [2.000]
2.001: (place-double-block-on-block arm1 arm2 blockE blockG) [2.000]
2.001: (pickup-single-block-on-block arm3 blockD magnetic) [1.000]
4.002: (move-down arm1) [1.000]
5.003: (pickup-single-block-on-block arm1 blockB blockA encompass) [1.000]
6.004: (place-single-block-on-block arm1 blockB blockC) [1.000]
```

Figure III.7: Example satisficing plan for v_C from example initial state.

capabilities and G requires the two friction capabilities; therefore, the capabilities required vector for v_E is $[2\ 2\ 0\ 0]$. The capabilities required vector for v_C is a function of B and C, as they are in the goal stack, and of D and E, because they are above C in the initial state. The capabilities required vector will be constructed iteratively as an example. E requires two friction capabilities, thus, $[0\ 2\ 0\ 0]$. D adds a requirement for a single magnetic capability, $[0\ 2\ 1\ 0]$. C adds a single encompass end effector, $[0\ 2\ 1\ 1]$. B requires a single encompass end effector, but an encompass end effector is already part of the capabilities required vector; therefore, the capabilities required vector is not modified. The final capabilities required vector for v_C is $[0\ 2\ 1\ 1]$.

An example MACPDDL description of the task to construct stack C in Figure III.3b is presented in Figure III.6. The services section implies that the task uses the service model of coalition formation. The number of each service required by the task is given, with the task assumed to require 0 of any unlisted services (the suction service in this example). The length is used by some coalition formation algorithms and is equal to the number of blocks used in deriving the capability vector. The goal is equivalent to the constraints on actions, but lacks temporal specifications. A task is complete in any state in which the task's goal constraint is satisfied.

The solution to the problem is a satisficing plan. An example plan to complete v_c when executed from the example initial state shown in Figure III.3a is given in Figure III.7. Each line corresponds to an action execution step. The example plan includes six action executions. The first action execution in the plan is for arm_1 and arm_2 to pickup E off of D using their friction end effectors. The action starts at $t = 0.000$ and

requires 2.000 time units to execute.

III.2.3 Zenotravel

The Zenotravel domain was originally created for testing the Zeno planner (Penberthy and Weld, 1994) and was augmented for this dissertation to include hub and spoke airports, passengers and cargo, and short-range and long-range planes. Spoke airports are airports in smaller cities, with each spoke connected to a single hub airport. Hub airports are located in larger cities and are connected to a set of spoke airports. Short-range planes fly only between a hub and its connected spokes. The sets of spoke airports for each pair of hubs are disjoint. All hubs are connected and only long-range planes can fly between them. Each plane has limited passenger and cargo capacity. The goal is satisfied when all passengers and cargo are at their destinations.

The state space includes both boolean and continuous variables. The boolean variables describe the location of each passenger, cargo, and plane. The continuous variables include the amount of passengers and cargo on each plane, each plane's passenger and cargo capacity, each plane's fuel level and capacity, and the distance between connected cities. The number of passengers, amount of cargo and their respective capacities for each plane are not continuous variables; however, similar to blocks world, modeling the values as continuous variables in PDDL facilitates the experiments and expression of the models by not requiring all possible values to be enumerated. The actions to load and unload passengers and cargo from a plane have fixed duration. Fuel use and the action duration for a plane to fly between two cities is a linear function of the distance traveled. The time required to refuel a plane is a linear function of the fuel level at the start of action execution and the fuel capacity. The capability model includes passenger and cargo capacity and the hub cities. For example, a short-range plane based out of the hub airport of ATL in Atlanta, Georgia has a capabilities offered vector corresponding to its passenger and cargo capacity and its ability to travel between ATL and ATL's spoke airports. A long-range plane has a capabilities offered vector corresponding to its passenger and cargo capacity and its ability to travel between any two hub airports, such as ATL and LAX in Los Angeles, California. The goal state is divided into tasks based on the origin and destination airports of the passengers and cargo. All passengers and cargo originating in a city and traveling to the same city are grouped into a single task. The capabilities required vector of each task is a function of the number of passengers and cargo included in the task, the origin, and the destination.

An example action implementation for a plane to fly between two cities is presented in Figure III.8. The action is executed by a single small plane and has two parameters: the origin city and destination city. The action duration is a function of the distance between the cities and the speed at which the plane flies. Executing the action requires that the plane start at the origin city and maintain a non-negative fuel level. The effect of the action is that the plane is no longer at the origin at the start of action execution and the plane

```
(:durative-action fly_small
  :executor (?p - small_plane)
  :parameters (?c_1 - city ?c_2 - city)
  :duration (= ?duration (/ (distance ?c_1 ?c_2) (speed ?p)))
  :condition
    (and
      (at start (plane_at_city ?p ?c_1))
      (over all (>= (fuel ?p) 0))
    )
  :effect
    (and
      (at start (not (plane_at_city ?p ?c_1)))
      (at end (plane_at_city ?p ?c_2))
      (decrease (fuel ?p) (* #t (fuel_rate ?p)))
    )
)
```

Figure III.8: MACPDDL action implementation for a small plane to fly between two cities.

```
(:task atl_regional
  :resources (
    small_cargo_atl - 4
    small_pass_atl - 3
  )
  :length 1
  :goal (and
    (transportable_at_city pass_52 atl)
    (transportable_at_city pass_53 atl)
    (transportable_at_city pass_54 atl)
    (transportable_at_city cargo_53 atl)
    (transportable_at_city cargo_54 atl)
    (transportable_at_city cargo_55 atl)
    (transportable_at_city cargo_56 atl)
  )
)
```

Figure III.9: MACPDDL description of a Zenotravel task to transport passengers and cargo near Atlanta.

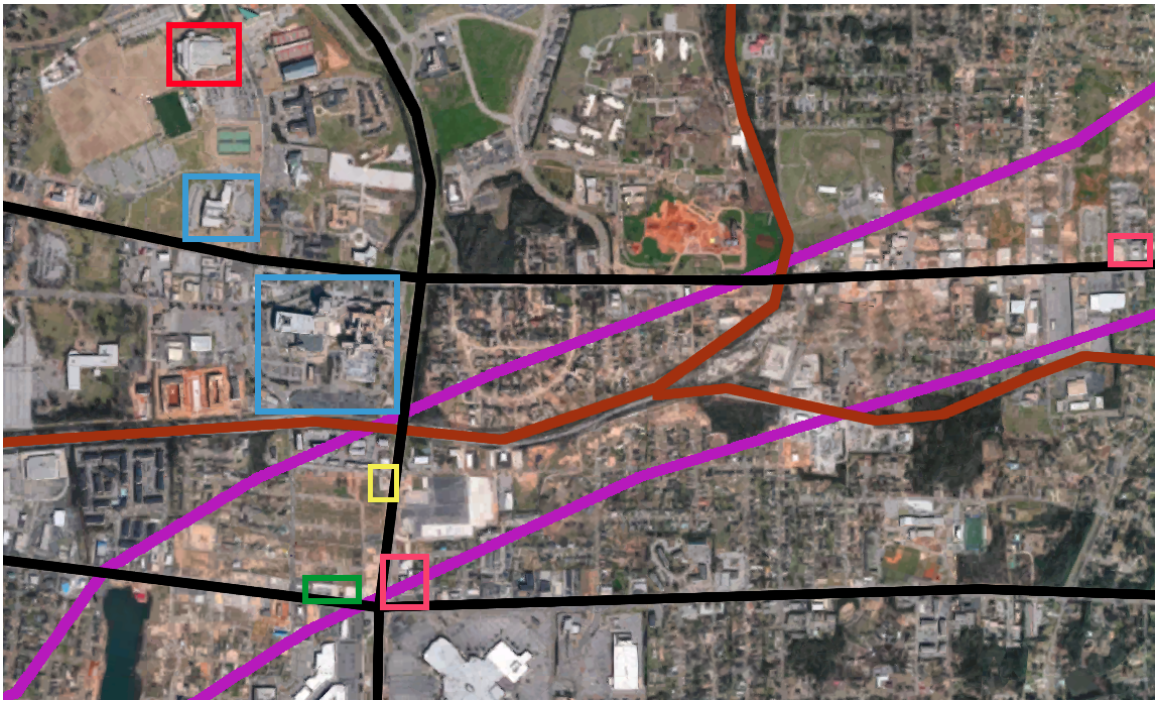


Figure III.10: Tuscaloosa, AL with selected locations

is at the destination at the end of action execution. The fuel level continuously decreases at the plane's fuel consumption rate throughout action execution. An example zenotravel task for transport around the Atlanta region is given in Figure III.9. The task uses the resource model of coalition formation and requires four units of cargo capacity from a small plane and three units of passenger capacity from a small plane. The goal is satisfied when the task passengers and task cargo (transportables) are in Atlanta.

III.2.4 First Response

The First Response task is based on a local government's immediate response to a tornado. Tuscaloosa, Alabama was hit by an EF4 multiple-vertex tornado in the early evening of April 27, 2011. The tornado passed within a half mile of The University of Alabama in Tuscaloosa (UA) and continued northeast to Birmingham, Alabama before dissipating. Sixty-four people were killed and thousands were injured and left without housing for weeks. UA and the surrounding area will be used as an example scenario.

A marked aerial view of Tuscaloosa is shown in Figure III.10. Businesses and houses were destroyed or severely damaged within a quarter mile of the tornado path (outlined in purple). The businesses include a pawn shop (green), two pharmacies (pink), and a gas station (red). The area roads were in good condition before the tornado, but destroyed cars, utility poles, and power lines made some roads impassable. Major roads are marked in black and rail lines in brown. Druid City Hospital (larger blue box) and the Student

Group	Agents
Police	humans, quadrotors, ground robots
Fire	humans, ground robots
EMS	humans, ground robots

Table III.1: Agents in the First Response Domain

Task	Group	Description
Victim Support	EMS Police	Triage victims
		Transport badly wounded victims to hospital
		Lead walking wounded victims to hospital
Roads	Fire	Clear downed power lines and debris from major roads
Gas Leaks	Fire	Seal gas leaks
Pharmacy	EMS	Secure controlled substances in destroyed pharmacy
Pawn Shop	Police	Secure weapons and munitions in destroyed pawn shop

Table III.2: Tasks in the First Response Domain

Health Center (smaller blue box) were both unaffected by the tornado. UA cancelled the remaining two weeks of the academic year, so the University Recreation Center (red) was available as a temporary shelter for displaced students and community members.

The state space for the First Response domain contains boolean and continuous variables. The environment is divided using a grid. Roads connect adjacent grid cells and each road is either traversable or blocked due to debris or power lines. Victims are located in a grid cell, as are human and robot first responders. Robot first responders have a power level. Hospitals, mobile police bases, shelters, pharmacies, gas stations, and pawn shops are located in a grid cell.

The agent model includes human and robotic first responders, summarized in Table III.1. All agents can move between adjacent grid cells, but human and ground robots require a traversable road in order to move to a new grid cell. Emergency Medical Services (EMS) responders include humans and ground robots. EMS human agents can triage victims and ground robots can transport victims. Police responders include humans, ground robots, and quadrotor aircraft. Police humans can search for victims, clear and secure pawn shops and pharmacies of restricted items (e.g., firearms at pawn shops and narcotic prescription drugs at pharmacies), and lead the walking wounded to hospitals or shelters. Police ground robots can transport restricted items to secure locations for storage. Police quadrotors can search for victims from the air, monitor pawn shop and pharmacy clearing operations, and lead victims to hospitals and shelters. The fire department personnel can clear power lines from roads and search for victims. The robotic first responders from the fire department can clear debris from roads.

Problems in the first response domain have goals subdivided into five task types, summarized in Table III.2. The first task is victim support. EMS agents must triage each victim in the disaster area. Each

```
(:durative-action bulldoze-edge-1
  :executor (?f - firebot)
  :parameters (?w_1 - waypoint ?w_2 -
    waypoint)
  :duration (= ?duration 2.0)
  :condition
    (and
      (over all (powered ?f))
      (over all (agent_at ?f ?w_1))
      (over all (edge ?w_1 ?w_2))
      (at start (debris ?w_1 ?w_2))
    )
  :effect
    (and
      (at start (not (debris ?w_1 ?w_2)))
      (at end (not_blocked ?w_1 ?w_2))
    )
)
```

Figure III.11: MACPDDL action implementation for a fire department robot to clear a blocked road.

```
(:task victim_triage
  :services (
    triage - 2
    victim_support - 1
    victim_transport - 1
    victim_search - 2
  )
  :goal (and
    (victim_at_hospital victim_1 dch)
    (victim_at_hospital victim_2 dch)
    (victim_at_hospital victim_3 dch)
    (victim_at_shelter victim_10 dch)
    (victim_at_shelter victim_11 dch)
    (victim_at_shelter victim_12 dch)
    (waypoint_searched waypoint_1)
    (waypoint_searched waypoint_2)
    (waypoint_searched waypoint_3)
  )
)
```

Figure III.12: MACPDDL description of a First Response task to get victims to a hospital.

victim must be transported or directed to a hospital depending on the victim's triage level and condition. Victim support is performed by EMS and Police responders. The second task is road clearing. Fire department agents must clear the major roads of debris and power lines. The third task involves sealing gas leaks. Gas leaks from gas stations and residential gas tanks must be sealed by the fire department. The fourth and fifth tasks are controlled item storage, which involves recovering and securing controlled items, such as prescription drugs and weapons located at the pharmacies and pawn shop, respectively. EMS are responsible for clearing the pharmacy and Police are responsible for clearing the pawn shop.

An example action for a fire department robot to clear road debris is presented in Figure III.11. The action duration is 2.0 time units. Executing the action requires the robot be powered, the robot be at the first

waypoint, a road (edge) exist between the two waypoints, and debris to be resident on the road. The effect of the action is that the road is cleared. An example victim triage task is presented in Figure III.12. The task requires four classes of services. The triage service is offered by EMS human agents, the victim support service is offered by police for leading victims to hospitals, the victim transport service is offered by the EMS robots, and the victim search service is offered by police and fire department humans and quadrotor aircraft.

III.3 Summary

The HMPCF problem formalization combines aspects of coalition formation and multi-agent planning. Four domains were presented: Rovers, Blocks World, Zenotravel, and First Response. Each domain was mapped to the HMPCF problem and implemented in MACPDDL. The next chapter presents tools developed to solve HMPCF problems, which balance computational resource requirements (time and memory) with the need to derive high quality plans.

CHAPTER IV

Planning Tools

Four centralized planning tools for solving HMPCF problems are presented: Planning Alone, Coalition Formation then Planning, Relaxed Plan Coalition Augmentation, and Task Fusion. Each tool is presented followed by an analysis and the motivation for the following tool. Planning Alone uses existing planning algorithms to solve HMCPF problems through a transformation to a single-agent planning problem. Coalition Formation then Planning addresses the computational difficulty of Planning Alone by factoring the problem along tasks and agents. Relaxed Plan Coalition Augmentation and Task Fusion address the nonexecutable coalition and suboptimal solution shortcomings of Coalition Formation then Planning, respectively.

IV.1 Planning Alone

Planning Alone (PA) is theoretically the simplest of the tools, applying a transformation to a single-agent planning problem and solving the resulting problem. The first transformation creates the set of actions that will be used during planning (Crosby et al., 2014). The set of actions available for planning, $Actions$, is a function of the agents in the grand coalition, $A(\Phi)$, shown in Line 1 of Algorithm IV.1. Assume a problem in the Blocks World domain and an action to pickup $block_m$ from the table. The single-agent version of the action is $(pickupSingleBlockFromTable\ block_m)$. Solving a multi-agent planning problem with a single-agent planner requires adding the executor coalition to the action. Let there be two agents, arm_i and arm_j. The example actions for arm_i and arm_j to pickup a single-weight block from the table are $(pickupSingleBlockFromTable\ arm_i\ block_m)$ and $(pickupSingleBlockFromTable\ arm_j\ block_m)$, respectively. The second required transformation combines all task goals into a single goal constraint, G, in Line 2. Let v_i and v_j be two tasks. The logical conjunction of the two tasks, $v_i \wedge v_j$, is a goal state constraint that is satisfied if and only if both v_i and v_j are completed. All tasks, $v \in V$, are completed if and only if G is satisfied. $Plan$ in Line 3 is a planning algorithm capable of reasoning over the action model, the state space (S), and the goal (G). Finally, translating the solution plan from the single-agent algorithm to a multi-agent plan reverses the action transformation used to derive the action set, i.e., the $(pickupSingleBlockFromTable\ arm_i\ block_m)$ action is converted to $(pickupSingleBlockFromTable\ block_m)$ action and assigned to arm_i.

The PA tool considers all agents solving all tasks simultaneously. The manner in which the tasks are divided has no effect on PA due to the tasks being combined into a single goal in the first step. All possible satisficing plans that the grand coalition can execute to solve the original problem can be derived using this method. Furthermore, the optimal plan to solve the problem must be among the plans that can be derived by

Algorithm IV.1: Planning Alone

Input : S - state space, A - coalition action mapping, Φ - grand coalition, I - initial state, V - tasks
Output: π - plan to complete all tasks

1 $Actions = A(\Phi)$;
2 $G = \bigwedge_{v \in V} cond(v)$;
3 $\pi = Plan(S, I, Actions, G)$;

this tool.

The limitation of the planning alone tool is computational complexity. The problem difficulty increases as the expressive features of the domain model, the number of agents, and the number of tasks increase. As the number of agents increases, so too does the number of actions that can be executed in any given state and the size of the state space. Increasing the number of actions increases the branching factor of the state space search, while increasing the size of the state space increases the time to perform operations on the state. Increasing the number of tasks decreases the number of satisficing plans by placing additional constraints on goal states.

Being able to generate all possible plans does have a drawback. PA is computationally expensive, which limits its applicability to real world problems with expressive domain models. The next tool, Coalition Formation then Planning, addresses computational complexity by considering subsets of agents to plan individual tasks.

IV.2 Coalition Formation then Planning

The Coalition Formation then Planning tool (CFP) produces several smaller planning instances focused on a subset of the goals, each using a subset of the agents to satisfy the goals. Algorithm IV.2 presents the CFP algorithm, which begins with an empty plan and no goals, Lines 1 and 2, respectively. The capabilities offered vector for each agent is identified using the capability mappings in Line 3. The capabilities required vector for each task is identified using the capability mappings in Line 4. Coalition formation is applied in Line 5 to allocate coalitions to tasks. Coalition formation's result is an assignment of candidate coalition to each task. The candidate coalitions can be overlapping, depending on the coalition formation algorithm applied. A coalition is a candidate coalition for a task if and only if the coalition has at least as many of each type of resource as required by task, i.e., Φ_i is a candidate coalition for v_i if and only if $\forall j, Cap_j^{\Phi_i} \geq Cap_j^{v_i}$. The planning loop, Line 6-12, is executed after coalition formation. The goals for v_i are combined with G in order to form the goals to be solved in the current iteration, Line 7. Combining the previous constraints with the current iteration constraints allows the iterative plan to break previous constraints in the course of planning, as long as the constraints are satisfied at the end of the iterative planning. The initial state, I_i, for the

Algorithm IV.2: Coalition Formation then Planning

Input : S - state space, A - coalition action mapping, Φ - grand coalition, I - initial state, V - tasks, C - capability mappings

Output: π - plan to satisfy all tasks

1 $\pi = \varnothing$;
2 $G = \varnothing$;
3 $\{Cap^{\phi}\}_{\phi \in \Phi} = C(\Phi)$;
4 $\{Cap^{v}\}_{v \in V} = C(V)$;
5 $\{\Phi_i, v_i\}_{i=1}^{|V|} = CF(\{Cap^{\phi}\}_{\phi \in \Phi}, \{Cap^{v}\}_{v \in V})$;
6 **foreach** $i \in \{1, \ldots, |V|\}$ **do**
7 $\quad G = G \wedge cond(v_i)$;
8 $\quad I_i = Simulate(I, \pi)$;
9 $\quad Actions = A(\Phi_i)$;
10 $\quad \pi_i = Plan(S, I_i, Actions, G)$;
11 $\quad \pi = PlanMerge(S, I, \pi, \pi_i, G)$;
12 **end**

current iteration is the end state achieved after simulating the current plan, π, from the initial state, I, using VAL, Line 8. The set of actions available, $Actions$, is a function of the coalition allocated to the current task, Φ_i, Line 9. An appropriate planning algorithm, $Plan$, finds an iterative plan, π_i, to satisfy G from the initial state, I_i, using the actions of the available coalition, $Actions$, in Line 10. CFP relies on coalition formation to produce executable coalitions. If Φ_i is a nonexecutable coalition, then CFP reports the problem as failed, else the iterative plan must be merged. π_i is merged with the current plan, π, to create a plan to satisfy G when executed from I, Line 11. The planning problem solved during each iteration in Line 10 is analogous to deriving a plan, executing the plan, and being given an additional planning goal to satisfy. The current goals, G, have been satisfied in the current state, I_i, but additional goals are given, so G is augmented with the additional goal constraints, $G = G \wedge v_i$, and a plan must be derived to transition from I_i to a state satisfying G. The plan merge step, Line 11, can be as simple as modifying the action execution steps in π_i, such that the action execution steps begin execution immediately when π ends; however, more complex scheduling can occur. A greedy scheduling approach, in which each action execution step in π_i is modified, one at a time in increasing original start time order, to occur as early as possible in the resulting plan, is applied. The overall approach is sound, if the underlying planning algorithm is sound, but is incomplete, even if the underlying planning algorithm is complete, due to greedy iterative task planning.

Performing coalition formation affects the problem in two ways by providing: a weakly smaller set of goal constraints to satisfy, and fewer agents with which to plan. First, the number of unsatisfied goal constraints that must be planned for is weakly reduced. The original problem has a set of constraints, G, to be satisfied. CFP partitions the constraints, $G = \{G_1, G_2, \ldots G_n\}$, where n is the number of tasks. The i^{th} planning loop iteration starts planning from an initial state, I_i, already satisfying $\{G_1, \ldots, G_{i-1}\}$. The i^{th} planning loop

iteration needs to satisfy G_i. The previously satisfied goal constraints can be violated if necessary, but they must be satisfied at at the end of executing π_i. The worst case is when planning for the n^{th} planning loop iteration, the resulting plan has to temporarily violate the constraints in $\{G_1, \ldots, G_{n-1}\}$ in order to satisfy G_n. The number of goal constraints that must be satisfied in any one planning iteration is not reduced in the worst case, but it is no greater than the number of goal constraints that must be satisfied during Planning Alone. Assume the Blocks World example from Figure III.3b on page 29. The set of goal constraints for PA addresses the seven blocks in the figure, whereas the goal constraints in CFP are divided into three sets of goal constraints, two of which address the locations of two blocks and one of which addresses the location of three blocks. Each of the stacks of blocks in the goal state description correspond to a task. When a task is completed, the goal constraints for the task never has to be unsatisfied in order to achieve a goal for subsequently planned tasks. Fewer incremental goal constraints must be satisfied during each CFP iteration than the number of goal constraints satisfied during PA.

Furthermore, the number of agents is reduced; thus, the number of actions is reduced and the number of variables in the state space is weakly reduced. Each of the agents in the experimental domains has a set of variables associated with it. For example, state variables are associated with each rover for its location and capabilities, the arms in Blocks World have a height above the table and a set of end effectors, and the planes in Zenotravel have a location and fuel level. The variables associated with the agents not included in planning are not part of the state space.

The composition and order of the task set affects the solution. A rearrangement of goal constraints among an equal number of tasks will result in a different solution. Greedily planning for each task places constraints on the subsequently planned tasks by affecting the iterative initial state; thus, the order in which the tasks are planned can affect all aspects of tool performance, including plan quality, computational resource usage, and whether or not a plan is found. The degree to which each task affects subsequent tasks is domain and problem dependent. The tasks in the Blocksworld domain greatly affect each other due to moving blocks from their initial state, possibly making coalitions nonexecutable. However, task planning in the Zenotravel and Rovers domains can only affect plan quality. The location of each plane in the iterative initial state is a function of prior task planning. Unlimited refueling allows each plane to get where it needs to go regardless of prior planning decisions. Task planning in the Rovers domain only affects when the rovers can use the single communications channel to transmit data to the central lander. Prior planning decisions affect the plan merge step and can require the communication actions to be delayed to prevent multiple rovers from using the single communications channel.

Reducing the number of actions creates a lower branching factor in the search tree. Every state, s, has a set of actions, A, which are executable from s when planning with Φ. If planning is performed with $\Phi_i \subset \Phi$

instead, then the actions in A which are only executable by agents in $\Phi \setminus \Phi_i$ can no longer be used to expand s. Reducing the action set at each state can eliminate reachable states, including some goal states; thus, reducing the search branching factor can force deeper searches to identify a goal state by increasing the minimum depth of goal states. The effects of the reduced action set and reduced state space combine to increase the number of states that can be searched per unit of computation time. Coalition formation identifies a coalitions to allocate to each task. Each identified coalition is selected based on the coalition's available capabilities and the task's required capabilities. If the grand coalition is not allocated to the task, then the action set and state space are reduced.

The capabilities required for each task in coalition formation is an estimate of the capabilities required to produce a satisficing plan for each task. Cap^{v_i} is determined based on v_i's goals and the problem initial state, but planning for v_i proceeds from an iterative initial state derived from prior planning decisions. Therefore, coalition formation makes decisions for coalitions and tasks using information that is likely to be inaccurate by the time the decision is applied. The information used in coalition formation is accurate for the first task, but the information used during coalition formation may have changed by the time the decisions are executed. The correct Cap^{v_i} derived from the iterative initial state and the task goals can deviate from the Cap^{v_i} used in coalition formation, such that additional capabilities are required and others are no longer needed. An example from the experimental domains is Blocks World may require additional capabilities to complete a task if blocks have moved. For example, the capabilities required for each task are a function of the blocks that must be moved to build the tower described in the task goal constraints. If a new block requiring a magnetic end effector is moved on top of one of the required blocks, then the coalition must have a magnetic end effector to move the new block and access the required block. If the required additional capabilities (an arm with a magnetic end effector in the example) are not included in the allocated coalition, then the coalition will be nonexecutable.

CFP is reliant on coalition formation to produce executable coalitions that can be used for planning. However, coalition formation can fail to produce executable coalitions. The next tool, Relaxed Plan Coalition Augmentation, modifies the CFP algorithm to account for nonexecutable coalitions.

IV.3 Relaxed Plan Coalition Augmentation

The Coalition Formation then Planning with Relaxed Plan Coalition Augmentation (RPCA) tool addresses the nonexecutable coalition limitation that arises with the CFP tool. RPCA adds logic to the planning loop in CFP (see Lines 12-17 in Algorithm IV.3). Planning for relaxed domains creates plans from lower fidelity models of the original domain, but the problem is easier; thus, relaxed plans are appropriate for use as a heuristic or, in this case, augmenting coalitions. Planning is attempted as in CFP, Line 11. If planning fails,

Algorithm IV.3: Coalition Formation then Planning with Relaxed Plan Coalition Augmentation

Input : S - state space, A - coalition action mapping, Φ - grand coalition, I - initial state, V - tasks, C - capability mappings

Output: π - plan to satisfy all tasks

1 $\pi = \varnothing$;
2 $G = \varnothing$;
3 $\{Cap^{\phi}\}_{\phi \in \Phi} = C(\Phi)$;
4 $\{Cap^{v}\}_{v \in V} = C(V)$;
5 $\{\Phi_i, v_i\}_{i=1}^{|V|} = CF(\{Cap^{\phi}\}_{\phi \in \Phi}, \{Cap^{v}\}_{v \in V})$;
6 $AllActions = A(\Phi)$;
7 **foreach** $i \in \{1, \ldots, |V|\}$ **do**
8 $\quad G = G \wedge cond(v_i)$;
9 $\quad I_i = Simulate(I, \pi)$;
10 $\quad Actions = A(\Phi_i)$;
11 $\quad \pi_i = Plan(S, I_i, Actions, G)$;
12 $\quad$ **while** Φ_i *is nonexecutable* **do**
13 $\quad\quad \pi_r = RelaxedPlan(S, I_i, AllActions, G)$;
14 $\quad\quad \Phi_i = \Phi_i \cup RelaxedPlanCoalitionAugmentation(\Phi, \Phi_i, \pi_r)$;
15 $\quad\quad Actions = A(\Phi_i)$;
16 $\quad\quad \pi_i = Plan(S, I_i, Actions, G)$;
17 $\quad$ **end**
18 $\quad \pi = PlanMerge(S, I, \pi, \pi_i, G)$;
19 **end**

then the algorithm enters the relaxed plan loop (Lines 12-17). A nonexecutable coalition, Φ_i, is modified in order to make it executable. Coalitions can be modified by adding agents, removing agents, or a combination thereof; however, allowing any modifications introduces an exponential number of possible coalitions. The tool is limited to adding agents in order to transform a nonexecutable coalition into an executable coalition. Limiting the coalition augmentation step to only adding agents to the coalition ensures that the loop runs a linear number of iterations, $|\Phi| - |\Phi_i|$, where Φ_i is the coalition originally allocated to the task. The set of actions available when planning with Φ, $AllActions$, is used in relaxed planning in Line 13. The grand coalition, Φ, the currently allocated coalition, Φ_i, and the generated relaxed plan, π_r, are analyzed to select additional agent(s) to allocate to the task, Line 14. Each action execution step in the relaxed plan is analyzed in execution order. If the action in the step can be executed by an agent in Φ_i or a subcoalition of Φ_i, then analysis continues to the next action execution step, otherwise, the agent or coalition assigned to the action execution step is added to Φ_i and relaxed plan analysis stops. At least one agent must be added to Φ_i before planning is attempted again, otherwise a nonexecutable coalition will be reported by the planning algorithm again. If all steps of the relaxed plan are analyzed and no agent has been identified to be added to Φ_i, then the agent, $\phi \notin \Phi_i$, allocated to the most action execution steps in π_r is added to Φ_i. Planning is attempted with the new coalition, Line 16, and the loop repeats if necessary.

The relaxed plan loop (Lines 12-17) has two potential completions. Either an executable plan is derived using the newly generated coalitions and planning attempts or the grand coalition is allocated to the task and the algorithm is unable to identify a plan. The latter case is inconclusive, since the grand coalition can be nonexecutable for multiple reasons. The grand coalition may be nonexecutable due to previous planning decisions that make the task nonexecutable. An example can be created from a finite fuel modification to Zenotravel. The version of Zenotravel used in this dissertation allows unlimited refueling, but if the amount of fuel available is limited, then prior planning decisions using excessive amounts of fuel can create situations in which no planes can reach their destination. The grand coalition may also be nonexecutable if the task cannot actually be executed, in which case the problem is unsolvable by any tool. An example of a nonexecutable grand coalition in the Blocks World domain is a grand coalition for which there are fewer than two arms with access to a required end effector type, in which case double-weight blocks with the missing end effector cannot be manipulated. If all the initially allocated coalitions are executable, then the relaxed plan loop starting at Line 12 is never entered; thus, RPCA reduces to CFP when all coalitions are executable.

Applying coalition formation and solving tasks iteratively does have a drawback. CFP has a reduced set of reachable states compared to PA. If all the goal states are eliminated, then the result is a nonexecutable coalition for a task, v. Selecting an agent, ϕ, to augment an allocated coalition, Φ_i, results in a new coalition, $\Phi_j = \Phi_i \cup \phi$. Planning for v with Φ_i is performed with $A(\Phi_i)$, the set of actions executable by the agents in Φ_i. Planning for v with Φ_j is performed with $A(\Phi_j) = A(\Phi_i) \cup A(\phi)$; thus, additional states are reachable. If any one of the additional reachable states is a goal state, then RPCA succeeds in correcting the nonexecutable coalition.

RPCA addresses the most important of the CFP drawbacks, nonexecutable coalitions. The remaining problem is low quality plans as a result of pruning the set of derivable plans. The Task Fusion tool addresses the low quality plans problem by selecting the pairs of tasks for which planning together maximizes marginal plan quality over RPCA, while minimizing marginal computational resource requirements.

IV.4 Task Fusion

The Task Fusion (TF) modification to RPCA addresses the limited ability of CFP to reason over task and agent interactions. Coalition formation is applied as in RPCA, Line 5 in Algorithm IV.4. Task Fusion reasons over the tasks, the task capabilities required, the coalition structure, and the coalition capabilities offered in order to predict which tasks and coalitions are most likely to benefit from joint planning, Line 6. Two coalition-task pairs, $\langle \Phi_a, v_a \rangle$ and $\langle \Phi_b, v_b \rangle$, are selected to be fused in order to create a single coalition-task pair, $\langle \Phi_a \cup \Phi_b, v_a \wedge v_b \rangle$. Fusing two tasks into a single task permits more potential interactions between the tasks to be considered during the planning step. Let S_i be the set of goal states satisfying v_i and S_j be the set

Algorithm IV.4: Coalition Formation and Task Fusion followed by Planning with Relaxed Plan Coalition Formation Augmentation

Input : S - state space, A - coalition action mapping, Φ - grand coalition, I - initial state, V - tasks, C - capability mappings
Output: π - plan to satisfy all tasks

$\pi = \varnothing$;
$G = \varnothing$;
$\{Cap^{\phi}\}_{\phi \in \Phi} = C(\Phi)$;
$\{Cap^{v}\}_{v \in V} = C(V)$;
$\{\Phi_i, v_i\}_{i=1}^{|V|} = CF(\{Cap^{\phi}\}_{\phi \in \Phi}, \{Cap^{v}\}_{v \in V})$;
$\{\Phi_j, v_j\}_{j=1}^{|V_f|} = TaskFusion(\{\Phi_i, v_i\}_{i=1}^{|V|})$;
$AllActions = A(\Phi)$;
foreach $j \in \{1, \ldots, |V_f|\}$ **do**
    $G = G \wedge cond(v_j)$;
    $I_j = Simulate(I, \pi)$;
    $Actions = A(\Phi_j)$;
    $\pi_j = Plan(S, I_j, Actions, G)$;
    while Φ_j *is nonexecutable* **do**
        $\pi_r = RelaxedPlan(S, I_j, AllActions, G)$;
        $\Phi_j = \Phi_j \cup RelaxedPlanCoalitionAugmentation(\Phi, \Phi_j, \pi_r)$;
        $Actions = A(\Phi_j)$;
        $\pi_j = Plan(S, I_j, Actions, G)$;
    end
    $\pi = PlanMerge(S, I, \pi, \pi_j, G)$;
end

of goal states satisfying v_j. Fusing v_i and v_j creates a new task with $S_i \cap S_j$ as the set of goal states. While the set of goal states is weakly smaller, the fusion allows the interactions between the tasks to be considered during planning.

The first heuristic for TF is *coalition assistance*, which uses the coalition capability offerings and task capability requirements to determine which tasks to fuse. The coalition assistance score is

$$\frac{\sum_{r \in req} \frac{Cap_r^{\Phi_i \cup \Phi_j}}{\max(Cap_r^{v_i}, Cap_r^{v_j})}}{|req|},$$

where v_i and v_j are tasks assigned to coalitions Φ_i and Φ_j, respectively, and req is the set of capability vector indices for which either v_i or v_j have non-zero requirements. Coalitions with overlapping capabilities are likely to be able to assist each other with their assigned tasks. If $Cap_r^{\Phi_i \cup \Phi_j} = Cap_r^{\Phi_i}$, then Φ_j cannot assist Φ_i with the parts of v_i requiring cap_r. Assume two coalitions, Φ_i and Φ_j assigned to tasks v_i and v_j, respectively. Let there be two capabilities, cap_m and cap_n, and each of the tasks require some amount of a single capability, $Cap_m^{v_i} = a$ and $Cap_n^{v_j} = b$, where the other elements of the capabilities required vector are 0. Φ_i and Φ_j must be candidate coalitions for v_i and v_j; therefore, the capability vectors for the two

coalitions must satisfy the following constraints:

$$Cap_m^{\Phi_i} \geq a,$$
$$Cap_m^{\Phi_j} \geq 0,$$
$$Cap_n^{\Phi_i} \geq 0,$$
$$Cap_n^{\Phi_j} \geq b.$$

If $Cap_n^{\Phi_i} = 0$, then no agent in Φ_i offers cap_n; thus, Φ_i cannot assist Φ_j on v_j. However, if $Cap_n^{\Phi_i} > 0$, then $\exists \phi \in \Phi_i$, such that $Cap_n^{\phi} > 0$. Combining Φ_i and Φ_j allows ϕ to offer cap_n to assist Φ_j with v_j. A similar argument as to whether or not Φ_j can assist Φ_i with v_i can be made by swapping the coalitions and capability indices in the appropriate locations. The drawback of this heuristic is that it assumes the coalition formation capability model of each task is complete and accurate, which is difficult to assure. Iteratively planning tasks and using a different initial state during planning than was used when deriving the capabilities required vectors makes it especially difficult to ensure the capabilities required vectors are accurate, as discussed in Chapter IV.2.

The second heuristic for TF is the *Jaccard similarity coefficient*, which is a function of the coalitions, but not of the tasks each coalition is assigned to complete. The Jaccard similarity coefficient is

$$\frac{|\Phi_i \cup \Phi_j|}{|\Phi_i \cap \Phi_j|},$$

where Φ_i and Φ_j are coalitions allocated to v_i and v_j, respectively. The score is in $[0, 1]$, where a score of 0 indicates no common agents between the coalitions and a score of 1 indicates identical coalitions. The similarity heuristic attempts to combine the coalitions most likely to have spatial and temporal constraints on their constituent agents. Planning with the two coalitions together allows the constraints to be considered. Planning for a single coalition allocated to two tasks is guaranteed to have spatial or temporal interactions due to the plans consisting of the same agents. Alternatively, with no common agents, an agent in one coalition is less likely to have spatial and temporal constraints with a different agent in the other coalition. The most interesting task fusion cases are those in which $\Phi_i \cap \Phi_j \neq \varnothing$ and the similarity score is less than 1. Such cases in real world problems are very likely to have spatial and temporal constraints on the actions each agent can execute simultaneously, but the question remains if the gain in plan quality is worth the increase in required computational resources. Assume the Zenotravel domain with an agent, $\phi \in \Phi_i \cap \Phi_j$, where v_i and v_j originate from Atlanta and Chicago, respectively, and the destination for both tasks is Seattle. If ϕ has enough cargo and passenger space for both v_i and v_j, then ϕ can fly from Atlanta to Chicago to Seattle,

instead of Atlanta to Seattle to Chicago to Seattle. The RPCA tool requires the current task to be completed before the next task can be considered, i.e., all passengers and cargo must be transported from Atlanta to Seattle before planning considers the passengers and cargo traveling from Chicago to Seattle.

The application of TF is identical regardless of which heuristic is selected. A minimum heuristic value for fusion is selected to prevent fusion of coalition-task pairs below a certain value. The heuristic value of each pair of two coalition-task pairs is calculated. The pair of coalition-task pairs with the highest heuristic value are fused if their heuristic value is greater than the minimum heuristic value. Each coalition-task pair can be fused with at most one other coalition-task pair. Pairs of coalition-task pairs continue to be fused until no more coalition-task pairs can be fused or the maximum heuristic value of a pair of coalition-task pairs is below the required minimum value.

Task composition affects TF. Each task can be as small as a single goal constraint, but such a composition constrains planning by totally ordering how the goals are satisfied. However, combining all the goals into a single task reduces the tool to PA. Somewhere between the two extremes exists a task composition balancing planning constraints and planning difficulty. The TF tool, by fusing coalition-task pairs, transitions the problems closer to the PA tool. However, the inverse operation, splitting coalition-task pairs is left as future work.

Each coalition-task allocation fusion decreases the number of tasks by one, as two tasks are replaced by a single task, but the average difficulty of the tasks is increased due to a single, much more difficult task replacing two component tasks. The goal of Task Fusion is to select the coalition-task pairs to fuse for which improved plan quality is worth the increase in computational resources.

IV.5 Summary

Four planning tools were presented: Planning Alone, Coalition Formation then Planning, Relaxed Plan Coalition Augmentation, and Task Fusion. These tools allow problem designers to balance planning difficulty, in terms of computational resource usage, with plan quality. The objective function for real world planning problems is a combination of multiple factors, including plan quality and computational resource usage. Solving real world problems requires finding quality plans, while considering the plan derivation time and required memory. The next chapter presents the experimental design used to analyze each tool.

CHAPTER V

Experimental Design

Each tool presented in Chapter IV was analyzed using the domains presented in Chapter III.2. This current chapter presents the common features of the experimental analysis shared between each tool.

V.1 Random Problem Generation

Grand Coalitions and Missions were generated for each domain. A Grand Coalition consists of a set of agents and their associated capabilities. A Mission consists of an initial state and a goal state description. Each Grand Coalition in each domain was paired with each Mission in the same domain to create *problems* to be solved. Ten Grand Coalitions and ten Missions were generated for each domain, for a total of 100 generated problems for each domain.

V.2 Metrics

A test case is a single problem attempted by a single planning tool. The dependent variables, as presented in Table V.1, were recorded during each test case. *Makespan* is the amount of time required to execute a plan:

$$makespan(\pi) = \max_{s \in \pi}(start(s) + dur(s)),$$

where π is a plan and s is an action execution step in π. The *number of action execution steps* in the generated plan was recorded. *Memory usage* was recorded using the Linux `getrusage` function. The `getrusage` function returns resource usage measures of the current process, including the maximum resident set size, which is an indicator of the amount of memory required by the planning tool. Reported memory values are in gigabytes. *Planning tool time* is the time for the planning tool to produce a solution in seconds.

Four potential outcomes exist for each test case. First, a satisficing plan for the grand coalition to achieve the goal is produced, in which case the metrics are reported. Second, no plan is produced due to a grand coalition being nonexecutable for a mission, which happens when an allocated coalition is confirmed by the

Dependent Variable	Units
Makespan	time
Action Execution Steps	number of actions executed
Memory Usage	gigabytes
Planning Tool Time	seconds

Table V.1: Dependent Variables

planning tool as unable to complete the assigned task. If a coalition is unable to complete its task, then the grand coalition cannot complete that mission. Third, the planning tool can exceed either the memory or computation time limits. All planning problems were limited to 48 GB of memory. All problems were limited to one hour of computation time, unless stated otherwise. Finally, a tool can fail to solve a problem due to using an incomplete planning algorithm.

VAL, the plan validator for PDDL (Howey et al., 2004), was used to confirm that the produced plans were satisficing. The experiments were run under Xubuntu 16.04 using an Intel Core i7-5820K CPU with 64 GB RAM. All source code is written in C++ and compiled with g++ 5.2.1.

V.3 Coalition Formation and Planning Algorithms

Five planning algorithms and three coalition formation algorithms were used with the presented tools. ITSAT (Rankooh and Ghassem-Sani, 2015), a SAT-based planner, and POPF2 (Coles et al., 2011) in enforced hill climbing (EHC) mode, a state space planner, were selected for planning in the Rovers domain and a service model approximation algorithm (Service and Adams, 2011) was selected for coalition formation. ITSAT was selected due to being open source after the 2014 International Planning Competition and a desire to test multiple classes of planning algorithms. POPF2 in EHC mode was selected due to being open source and easy to extend for use in this research. EHC is a greedy approach to planning that requires expanded states have monotonically decreasing heuristic values. The requirement can greatly improve search times, but sacrifices completeness. The service model approximation algorithm provides solutions quickly and supports the service model of coalition formation used in the Rovers domain. TFD (Eyerich et al., 2014), a state space search planner using the context-enhanced additive heuristic modified for continuous state variables and temporal planning, and COLIN (Coles et al., 2012a) in best-first search mode, a state space planner using a relaxed planning graph heuristic, was selected for the Blocks World domain and a dynamic programming coalition formation algorithm (Service and Adams, 2011) was selected for coalition formation. TFD was selected due to being open source, performing well in the temporal track of the 2014 International Planning Competition, and supporting the continuous state variables. COLIN was selected due to being open source and easily extended for use in this research. The dynamic programming algorithm finds solutions for the Blocks World problems quickly and works with either service or resource models. Coalition formation in the Blocks World domain uses the service model. The Zenotravel domain planner is the EHC version of the COLIN planner (Coles et al., 2012a) and was selected for its support of continuous linear effects. A greedy algorithm (Shehory and Kraus, 1998) was selected for coalition formation in the Zenotravel domain, because it works quickly by limiting potential coalition size. The same planner and coalition formation algorithm used with Zenotravel was also used for the First Response domain.

All three coalition formation algorithms can be applied to the Rovers, Blocks World, and First Response domains, but the service model approximation algorithm cannot be applied in the Zenotravel domain due to not supporting the resource model of coalition formation. Both versions of COLIN support all the necessary features to plan problems in all four domains. TFD and POPF2 can be used to solve problems in the Rovers and Blocks World domains, but do not support the continuous effects required for problems in the Zenotravel and First Response domains. ITSAT does not support continuous variables, so it can only be used with the Rovers domain.

CHAPTER VI

Results

Each of the domains presented in Chapter III was evaluated using the planning tools presented in Chapter IV. This chapter summarizes the results and provides a discussion, while the full results are presented in the cited appendices. The notation "Problem i-j" will be used to indicate the problem generated by the combination of Mission i and Grand Coalition j. The notation "X[Y]" will be used to indicate tool X with underlying planner Y, for example, CFP[COLIN] indicates the CFP tool using COLIN as the underlying planner.

VI.1 Rovers

The Grand Coalitions included ten randomly generated rovers. Each rover was allocated tools allowing it to collect an average of two of the five classes of scientific data, defined in Chapter III.2.1. The rovers were generated by drawing from a binomial distribution ($n = 5, p = 0.4$), with zero values modified to have a single random capability. The mission initial states included the connections between the waypoints, the waypoints between which the rovers navigated, each rover's starting location, scientific data source locations, and the central lander's location. The mission goal state description requires all the scientific data to be communicated to the central lander. The missions were generated using the random problem generator used in the IPC with the rover related variables removed. The goal state is divided into five tasks, with the goal for each task consisting of collecting all the data of a specific type.

Summary capability data for the ten generated Grand Coalitions are presented in Table VI.1. Rovers can have multiple capabilities; thus, the total capabilities for each Grand Coalition will be greater than the number of rovers. Grand Coalition 1, for example, has five rovers capable of taking low-resolution images. Each

Grand Coalition	Low-Res	High-Res	Color	Rock	Soil	Total
1	5	3	3	6	2	19
2	6	3	2	6	4	21
3	3	4	3	2	6	18
4	4	3	4	6	4	21
5	6	3	3	5	5	22
6	6	5	8	2	4	25
7	5	3	5	3	3	19
8	3	3	3	3	3	15
9	4	4	3	4	7	22
10	6	5	5	6	6	28

Table VI.1: Number of rovers capable of each capability by Grand Coalition

Mission	Low-Res	High-Res	Color	Rock	Soil	Total
1	8	23	14	46	25	116
2	6	16	9	28	42	101
3	18	23	8	22	21	92
4	16	15	14	30	34	109
5	20	19	23	24	29	115
6	7	11	8	42	29	97
7	13	7	4	44	26	94
8	18	19	12	40	28	117
9	22	13	21	28	38	112
10	20	17	24	35	39	135

Table VI.2: Objectives per data type by Mission

Grand Coalition averaged 4.2 rovers capable of collecting a given class of scientific data, with a minimum of two rovers in each Grand Coalition capable of collecting each class of scientific data. Grand Coalition 8 has the lowest total capabilities at 15, with three rovers offering each of the five capabilities. Grand Coalition 10 is the most capable, with at least five rovers offering each capability, while Grand Coalitions 3 and 6 each have two rovers capable of rock analysis and Grand Coalition 1 has two rovers capable of soil analysis. Rock and soil analysis data are the most difficult to collect, because they require the rover to be at a waypoint to collect data, while the three types of imaging analysis data only require that the rover be at one of several waypoints with line of sight to the target waypoint in order to collect data. A rover can collect imaging data for several waypoints without moving.

The ten generated Missions are summarized in Table VI.2. Each Mission had 100 waypoints, but each waypoint can require multiple different analyses. Mission 7, for example, has seven waypoints requiring high-resolution imaging. Each mission required collecting an average of 116.1 pieces of scientific data. Mission 10 requires the most data, with a total of 135 data pieces required, and the second highest soil analysis requirement, while Mission 3 requires the least data, with a total of 92 data pieces, and the fewest soil and rock analysis requirements.

VI.1.1 Planning Alone

The first set of results demonstrate the complexity of planning for all tasks with all agents simultaneously in Planning Alone (PA). PA[POPF] was unable to solve any of the generated Rovers problems due to exceeding

Planner	Plans Found	Time Fails
POPF	0	100
ITSAT	100	0

Table VI.3: Number of Rovers problems solved and failed with Planning Alone per planner

Planner	Makespan	Action Executions	Time (s)	Memory (GB)
POPF	N/A			
ITSAT	1545.3 (396.8)	736.5 (135.2)	2261.8 (588.2)	17.32 (3.15)

Table VI.4: Descriptive statistics for makespan, action executions, time required, and memory required for solved Rovers problem using Planning Alone with each planner.

Planner	Mission	Grand Coalition	Makespan	Action Executions	Time (s)	Memory (GB)
POPF	1	5	Time fail			
	3	10	Time fail			
	6	6	Time fail			
	10	3	Time fail			
ITSAT	1	5	1369	697	2047.1	16.62
	3	10	876	485	2016.8	16.74
	6	6	1817	869	3723.0	25.69
	10	3	1942	902	1928.9	17.49

Table VI.5: Makespan, action executions, time required, and memory required for four example Rovers problems using Planning Alone

the one hour time limit, as shown in Table VI.3. Ten of the problems were selected to run without the time limit, but all ten failed due to exceeding the 48 GB memory limit. PA[ITSAT] solved all 100 problems, but required on average 38 minutes and 17.32 GB of memory to solve each problem. Detailed results for the PA[ITSAT] are presented in Appendix C.1. The descriptive statistics for makespan, number of action executions, time, and memory is presented in Table VI.4. The solutions produced using PA[ITSAT] had a mean makespan of 1545.3 and 736.5 action executions.

The full results for a set of selected problems are provided in Table VI.5. These same problems will be used for comparison across and discussions of the next three tools. Problem 1-5 was selected for being representative of the trends seen in the results for the four tools. All the metrics collected for PA[ITSAT] solving Problem 1-5 are above the averages (between 0.05 and 0.65 standard deviations). Problem 3-10 was selected due to being created from the Mission requiring the least data and the Grand Coalition with the most capable rovers. The derived plan for Problem 3-10 solved by PA[ITSAT] is the shortest plans derived using this combination, as expected. The rovers of Grand Coalition 10 average more capabilities than the rovers in other Grand Coalitions. The majority of the makespan in the Rovers domain is the result of rovers navigating the environment. More capable rovers have more options regarding at which waypoints they can perform analysis actions and contribute to the mission. Problem 6-6 was selected due to requiring a large amount of rock data, but the Grand Coalition only includes two rovers capable of collecting rock data. However, the high rock analysis requirement in this Mission is countered by a lower than average requirement for the other four data types. Problem 10-3 is the combination of the Mission with the highest number of required data pieces executed by a Grand Coalition with a below average number of rovers equipped for most of the capabilities.

Planner	Plans Found	Total Failures	Nonexecutable Coalition	EHC
POPF	69	31	22	9
ITSAT	78	22	22	0

Table VI.6: Number of Rovers problems solved and failed with CFP per planner

Planner	Makespan	Action Executions	Time (s)	Memory (GB)
POPF	2915.8 (518.5)	741.8 (102.9)	297.8 (157.3)	7.96 (6.13)
ITSAT	1888.9 (336.4)	625.5 (72.6)	1653.6 (330.6)	1.69 (0.46)

Table VI.7: Descriptive statistics for makespan, action executions, time required, and memory required per solved Rovers problem using CFP with each planner

All metrics collected while solving the problem (with the exception of number of action executions) were less than a standard deviation above the average.

Planning Alone leaves much to be desired. The computational resources required to solve the problems with PA are significant. PA[POPF] failed to find any plans due to exceeding available computational resources and the memory required by PA[ITSAT] is high. If the problem size is increased (e.g., more agents, more data to be collected, or more waypoints), then the memory required will increase. The problems generated are solvable, but problems can be generated that PA[ITSAT] will be unable to solve due to exceeding the available memory. The next set of results are for CFP, the tool developed to address the problem of large computational resource requirements.

VI.1.2 Coalition Formation then Planning

Coalition Formation then Planning (CFP) resulted in a large improvement over Planning Alone in terms of required computational resources to solve a problem. CFP[POPF] was able to solve 69 of the 100 problems, while CFP[ITSAT] solved 78 of the 100 problems, as presented in Table VI.6. The 22 problems CFP[POPF] failed to solve are the same 22 problems CFP[ITSAT] failed to solve, all of which were due to coalition formation allocating nonexecutable coalitions. The nonexecutable coalition structures derived by coalition formation are due to rovers being unable to reach a necessary waypoint, i.e., the coalition being nonexecutable is independent of the plans derived for other tasks. CFP[POPF] failed to solve an additional nine problems due to using the incomplete EHC algorithm. Detailed results for CFP[POPF] and CFP[ITSAT] are presented in Appendices C.2 and C.3, respectively.

The makespan, action executions, and required derivation time and memory descriptive statistics are presented in Table VI.7. CFP[POPF] required, on average, 297.8 seconds and 7.96 GB of memory to derive plans with an average makespan of 2915.8 and 741.8 action executions, while CFP[ITSAT] required an average of 1653.6 seconds and 1.69 GB of memory to derive plans with an average makespan of 1888.9

Planner	Problems	Makespan	Action Executions	Time	Memory
POPF	0	N/A			
ITSAT	78	1.258 (0.306)	0.864 (0.108)	0.769 (0.175)	0.099 (0.020)

Table VI.8: Average (and standard deviation) ratio of CFP metric to PA metric for makespan, action executions, time required, and memory required for problems solved by both CFP and PA

Planner	Mission	Grand Coalition	Makespan	Action Executions	Time (s)	Memory (GB)
POPF	1	5	3372	775	645.1	27.51
	3	10	2621	651	357.7	5.68
	6	6	Nonexecutable Coalition			
	10	3	2962	852	378.4	5.13
ITSAT	1	5	2292	639	1770.5	1.66
	3	10	1194	506	1227.8	1.33
	6	6	Nonexecutable Coalition			
	10	3	1750	741	2102.4	1.41

Table VI.9: Makespan, action executions, time required, and memory required for four example Rovers problems with each planner using CFP

and 625.5 action executions. CFP[POPF] required only 18% of the time that CFP[ITSAT] required to derive plans, but CFP[ITSAT] required less memory, only 21% of that used by CFP[POPF].

The 78 problems solved by CFP[ITSAT] were also solved by PA[ITSAT]. Descriptive statistics of the ratios of the results from the set of problems solved by both CFP[ITSAT] and PA[ITSAT] are presented in Table VI.8. PA[POPF] did not solve any problems; therefore, no quantitative comparison with CFP[POPF] is provided. The makespan of the problems solved by CFP[ITSAT] are 25.8% higher, on average, than the makespan of the corresponding problems solved by PA[ITSAT], while the number of action executions was 13.6% lower. The key result is the large decrease in the computational resource requirements. CFP[ITSAT] used on average 76.9% of the time and 9.9% of the memory required to solve the same problems as PA[ITSAT].

The detailed results for the four example problems are presented in Table VI.9. Problem 1-5 was solved by both CFP[POPF] and CFP[ITSAT]. CFP[POPF] required 645.1 seconds and 27.51 GB of memory to derive a plan that included 775 action executions and had a makespan of 3372. While the memory requirements were high, it is much better to have a plan than no plan (as with PA[POPF]). CFP[ITSAT] derived a plan for Problem 1-5 with a makespan of 2292 and 639 action executions, while using 1.66 GB of memory and 1770.5 seconds. The plan is 67.4% longer in terms of makespan, but requires 8.3% fewer action executions than the plan derived using PA[ITSAT]. The most important difference is the difference in computational resources required; CFP[ITSAT] required 86.5% of the time and 10.0% of the memory required by PA[ITSAT]. Solving Problem 3-10 with CFP[POPF] produced a plan with a makespan of 2621 and 651 action executions, while using 5.68 GB of memory and 357.7 seconds of computation time. CFP[ITSAT] required 1227.8 seconds and

1.33 GB of memory to solve Problem 3-10, reducing the time required by 39.1% and memory required by 92.1% over PA[ITSAT]. The plan derived by CFP[ITSAT] was 2.99 times longer in terms of makespan, but 6.6% shorter in terms of action executions. Neither CFP[POPF] nor CFP[ITSAT] solved Problem 6-6 due to coalition formation allocating a nonexecutable coalition. Problem 10-3 was solved by both CFP[POPF] and CFP[ITSAT]. CFP[POPF] required 378.4 seconds and 5.13 GB of memory to derive a plan with a makespan of 2962 over 852 action executions. CFP[ITSAT] produced a plan with a makespan of 1750 and 741 action executions, while using 2102.4 seconds of computation time and 1.41 GB of memory. Compared with PA[ITSAT], solving Problem 10-3 with CFP[ITSAT] required 8.2% less time and 91.9% less memory to derive a plan with a 9.9% shorter makespan. However, the plan produced by CFP[ITSAT] required 21.7% more action executions than the plan produced by PA[ITSAT].

If coalition formation allocates a nonexecutable coalition to any of the tasks, then the CFP tool will fail due to being unable to derive a plan for the task assigned to the nonexecutable coalition. Reachability of waypoints is not considered in the capability model used in coalition formation, as such, coalition formation cannot consider the subsets of waypoints each rover can reach. Adding the reachable waypoints to the capability model guarantees coalition formation will allocate only executable coalitions, but doing so increases the complexity of the coalition formation problem by requiring knowledge of which waypoints each rover can reach. The next tool, RPCA, addresses the executable coalition structure problem.

VI.1.3 Relaxed Plan Coalition Augmentation

Relaxed Plan Coalition Augmentation (RPCA) increases the number of problems solved with each underlying planner over Coalition Formation then Planning, presented in Table VI.10. RPCA[POPF] solved 85 problems, versus CFP[POPF]'s 69 problems. The additional problems solved come exclusively from CFP[POPF]'s nonexecutable coalition cases. RPCA[POPF] augmented the coalition structures of and solved 16 of the 22 nonexecutable problems. The remaining six problems were unsolvable due to EHC failures or exceeding computational resource limits. Detailed results for RPCA[POPF] are presented in Appendix C.4. CFP[ITSAT] solved 78 problems, while RPCA[ITSAT] solved 100 problems; thus, bringing the number of problems solved by RPCA[ITSAT] up to the same level as PA[ITSAT]. Detailed results for RPCA[ITSAT] are presented in Appendix C.5.

Planner	Plans Found	Total Failures	Time Fails	Memory Fails	EHC
POPF	85	15	1	1	13
ITSAT	100	0	0	0	0

Table VI.10: Number of Rovers problems solved and failed with RPCA per planner

Planner	Makespan	Action Executions	Time (s)	Memory (GB)
POPF	2919.8 (605.8)	736.2 (95.7)	348.3 (342.1)	8.47 (7.43)
ITSAT	1951.0 (390.7)	623.1 (68.4)	1610.3 (331.3)	1.85 (0.62)

Table VI.11: Descriptive statistics for makespan, action executions, time required, and memory required per solved Rovers problem using RPCA with each planner. Standard deviations are in parentheses.

Planner	Problems	Makespan	Action Executions	Time	Memory
POPF	0	N/A			
ITSAT	100	1.322 (0.369)	0.862 (0.106)	0.739 (0.172)	0.107 (0.029)

Table VI.12: Average (and standard deviation) ratio of RPCA metric to PA metric for makespan, action executions, time required, and memory required for problems solved by both RPCA and PA.

The descriptive statistics for makespan, action executions, and the computational resources required are presented in Table VI.11. RPCA[POPF] produced plans with an average makespan of 2919.8 and 736.2 action executions, while using 348.3 seconds of computation time and 8.47 GB of memory. RPCA[ITSAT] performed better in all metrics, with the exception of computation time. RPCA[ITSAT] derived plans with an average makespan of 1951.0 and 623.1 action executions, while using 1610.3 seconds of computation time and 1.85 GB of memory.

Both PA[ITSAT] and RPCA[ITSAT] solved all 100 problems; thus, a comparison of the metrics can be performed. The average ratio of the metric value of RPCA[ITSAT] solving a problem versus PA[ITSAT] solving the same problem is presented in Table VI.12. RPCA[POPF] is excluded due to PA[POPF] not solving any problems. RPCA[ITSAT] averaged plans 32.2% longer in terms of makespan in comparison to PA[ITSAT], while action executions were reduced by 13.8%, computation time was reduced by 73.9%, and required memory was reduced by 89.3%.

Results for the four example problems are presented in Table VI.13. Problems 1-5, 3-10, and 10-3 were solved by both CFP[POPF] and CFP[ITSAT], thus the RPCA results are the same. A nonexecutable coalition structures was allocated for Problems 6-6 with the CFP tool, but RPCA[POPF] and RPCA[ITSAT] augmented

Planner	Mission	Grand Coalition	Makespan	Action Executions	Time (s)	Memory (GB)
POPF	1	5	3372	775	645.1	27.51
	3	10	2621	651	357.7	5.68
	6	6	4554	733	2666.1	31.89
	10	3	2962	852	378.4	5.13
ITSAT	1	5	2292	639	1770.5	1.66
	3	10	1194	506	1227.8	1.33
	6	6	3269	622	1797.3	3.92
	10	3	1750	741	2102.4	1.41

Table VI.13: Makespan, action executions, time required, and memory required for four example Rovers problems solved with RPCA

the coalitions in order to make them executable, which resulted in a solution. RPCA[POPF] derived a plan with a makespan of 4554 using 733 action executions, while requiring 2666.1 seconds of computation time and 31.89 GB of memory. The solution to Problem 6-6 derived by RPCA[ITSAT] had a makespan of 3269 using 622 action executions, while requiring 2102.4 seconds of computation time and only 1.41 GB of memory. RPCA[ITSAT]'s solution had a 79.9% higher makespan, but required 28.4% fewer action executions, 43.5% less computation time, and 94.5% less memory.

The key benefit to RPCA is its ability to use CFP and correct for nonexecutable coalitions, while maintaining lower memory and time requirements when compared to Planning Alone. RPCA derived a relaxed plan that was analyzed to select the rover(s) required for the coalition to collect the data for each task. A relaxed plan is not a solution to the original problem, but it is much easier to derive and provides information as to the agent(s) most likely to make the coalition executable when added to the coalition. The next set of results are for the Task Fusion tool used to address the lower plan quality problem that arises as a result of factoring the planning problem into tasks.

VI.1.4 Task Fusion

The coalition assistance score was used to select the tasks to fuse. Two eligible pairs of tasks with maximal score were selected for fusion. Tasks pairs had to had a score of at least 1.5 to be eligible. As an example, Φ_{rock} and Φ_{soil} are disjoint coalitions, with two rovers each, assigned to v_{rock} and v_{soil}, respectively. Both agents in Φ_{rock} can perform rock analysis and both agents in Φ_{soil} can perform soil analysis. Additionally, at least one agent in Φ_{rock} can perform soil analysis and one agent in Φ_{soil} can perform rock analysis. Fusing the coalition-task allocation creates a new coalition, $\Phi_{new} = \Phi_{rock} \cup \Phi_{soil}$, with at least three agents capable of rock analysis and three agents capable of soil analysis. The task for Φ_{new} is a combination of $v_{rock} \wedge v_{soil}$, to collect all of the soil and rock data.

The difficulty of planning for fused tasks is demonstrated by the TF[POPF] results, as shown in Table VI.14. TF[POPF] solved only 19 of the problems, with 48 problems exceeding the time limit and 33 problems exceeding the memory limit. TF[ITSAT] performed much better by solving all the problems. Detailed results for TF[POPF] and TF[ITSAT] are presented in Appendices C.6 and C.7, respectively.

The descriptive statistics for makespan, action executions, and the required computational resources for

Planner	Plans Found	Total Failures	Time Fails	Memory Fails
POPF	19	81	48	33
ITSAT	100	0	0	0

Table VI.14: Number of Rovers problems solved and failed with TF per planner

Planner	Makespan	Action Executions	Time (s)	Memory (GB)
POPF	2739.8 (579.3)	736.2 (90.2)	1114.6 (1064.2)	14.18 (11.70)
ITSAT	1886.8 (392.2)	636.1 (69.3)	1433.5 (307.6)	3.95 (1.42)

Table VI.15: Descriptive statistics for makespan, action executions, time required, and memory required per solved Rovers problem using TF with each planner

Tool	Planner	Problems	Makespan	Action Executions	Time	Memory
PA	POPF	0	N/A			
	ITSAT	100	1.279 (0.361)	0.883 (0.129)	0.651 (0.124)	0.229 (0.077)
RPCA	POPF	18	1.007 (0.026)	0.992 (0.330)	4.686 (4.793)	3.538 (4.948)
	ITSAT	100	0.980 (0.164)	1.022 (0.061)	0.900 (0.144)	2.203 (0.705)

Table VI.16: Average (and standard deviation) ratio of TF metric to PA and RPCA metrics for makespan, action executions, time required, and memory required for problems solved by both tools.

both TF[POPF] and TF[ITSAT] are presented in Table VI.15. TF[POPF] averaged plans with a makespan of 2739.8, 736.2 action executions, a computation time of 1114.6 seconds and memory usage of 14.18 GB. The plans derived by TF[ITSAT] averaged a makespan of 1886.8 with 636.1 action executions, while using 1433.5 seconds of computation time and 3.95 GB of memory.

A comparison of TF to the other tools is presented in Table VI.16. The Tool and Planner column represent the configuration being compared. The RPCA[POPF] row, for example, compares the plans derived by RPCA[POPF] and those derived by TF[POPF] for only those problems solved by both configurations. The number of Problems considered in the comparison is given in the Problems column. RPCA[POPF] and TF[POPF], for example, solved 18 Problems in common. TF[ITSAT] derived plans with a 28.9% longer makespan than those derived by PA[ITSAT], but the number of action executions was 11.7% less, computation time was 34.9% less, and 77.1% less memory was used. TF[POPF] performed similarly to RPCA[POPF] in terms of plan quality (0.7% higher makespan, but 0.8% fewer action executions), but performed much worse in terms of computational resource requirements, requiring 367% more time and 254% more memory. TF[ITSAT] produced 2.0% better plans by makespan than RPCA[ITSAT], but required 2.2% more action executions. TF[ITSAT] reduced computation time by 10.0% compared to RPCA[ITSAT], but increased memory usage by 120%.

Full results for the four example Problems are presented in Table VI.17. TF[POPF] failed to solve three of the example problems due to exceeding the time limit and failed to solved the fourth due to exceeding the memory limit, while TF[ITSAT] solved all the example Problems.

The plan TF[ITSAT] derived for Problem 1-5 had a makespan of 2051, used 680 action executions, while using 1573.9 seconds of computation time, and 4.07 GB of memory. The plan quality and memory metrics were midpoints between the solutions derived by PA[ITSAT] (33.3% lower makespan, 2.5% more action

Planner	Mission	Grand Coalition	Makespan	Action Executions	Time (s)	Memory (GB)
POPF	1	5	Time failure			
	3	10	Time failure			
	6	6	Time failure			
	10	3	Memory failure			
ITSAT	1	5	2051	680	1573.9	4.07
	3	10	1844	540	1366.9	5.41
	6	6	2885	645	1960.8	5.31
	10	3	1633	700	1638.6	2.89

Table VI.17: Makespan, action executions, time required, and memory required for four example Rovers problems solved with TF

executions, and 308% higher memory usage) and RPCA[ITSAT] (11.8% higher makespan and 6.0% fewer action executions, and 59.2% less memory usage). The time required to solve Problem 1-5 with TF[ITSAT] was 76.9% and 88.9% of that required by PA[ITSAT] and RPCA[ITSAT], respectively.

Plan makespan for Problem 3-10 solved by TF[ITSAT] was 211% and 154% of those plans derived by PA[ITSAT] and RPCA[ITSAT], respectively. The number of action executions were also worse, 11.3% higher compared to PA[ITSAT] and 6.7% higher compared to RPCA[ITSAT]. Both computational resource metrics were midpoints, with PA[ITSAT] requiring 47.5% more time and 209% more memory, while RPCA[ITSAT] required 10.2% less time and 75.4% less memory.

Problem 6-6 was a midpoint across all metrics. The makespan of the plan derived by TF[ITSAT] was 58.8% higher than that produced by PA[ITSAT], but 11.7% lower than that produced by RPCA[ITSAT]. The number of action executions in the plan derived by TF[ITSAT] was 34.5% lower with PA[ITSAT], but 3.6% higher than that produced by RPCA[ITSAT]. Computation time required by TF[ITSAT] was 52.7% that of PA[ITSAT], but 109% that of RPCA[ITSAT]. Memory usage with TF[ITSAT] was 20.7% that required by PA[ITSAT], but 135% that required by RPCA[ITSAT].

TF[ITSAT] performed the best in all metrics when solving Problem 10-3, with the exception of required memory. Plan makespan was 84.1% and 93.3% of those derived by PA[ITSAT] and RPCA[ITSAT], respectively, while the number of action executions was at 77.6% and 94.5%, respectively. Computation time was 84.9% and 77.9% of that required by PA[ITSAT] and RPCA[ITSAT], respectively, but memory usage was at 16.5% and 205%, respectively.

The results for TF were surprising in that the makespan was reduced by such a small value. Each rover is capable of collecting multiple data types in most cases; however, CFP and RPCA limit each rover to collecting a single data type, due to the goal constraints for a single task only addressing a single data type. Applying TF allows rovers in fused tasks to collect multiple data types. For example, fusing the rock and soil tasks allows a rover to collect both rock and soil data at a single waypoint. The majority of the plan makespan

is used to navigate the environment; therefore, allowing the rovers to collect multiple pieces of data without moving will reduce the navigation required to complete each task. The results for TF[ITSAT] show only a small decrease in makespan over RPCA[ITSAT] and the results for TF[POPF] show a small increase, which is likely due to using satisficing planners, rather than optimal planners. Satisficing planners do not consider aspects of plan quality, such as makespan, and are only concerned with finding a plan to satisfy the goal constraints.

VI.1.5 Discussion

The Rovers domain has been used for many years as a benchmark for planning algorithms. The domain includes multiple heterogeneous agents and was easily adaptable to the HMPCF problem, making it an excellent choice for testing the developed tools. Each of the rovers can operate independently for most of the plan execution, the only exception being communication with the central lander. The mostly independent operation of the rovers simplifies the plan merge steps in CFP, RPCA, and TF. Furthermore, the tasks are mostly independent; a plan for one task only affects the scheduling of the action executions in the plan for another task. The only time at which two plans will negatively interact occurs when rovers need to communicate with the lander.

Selecting the best option for the Rovers domain requires additional problem information. Rover energy level is ignored in the domain model used in the experiments, so plan makespan is likely to be the most important of the plan quality metrics. Changing the domain to include energy requirements will make action executions more important if energy is a limited resource. Such a problem description may make it desirable to sacrifice plan makespan to avoid additional action executions. A real world implementation of this domain is likely to balance the two plan quality metrics. A short makespan is meaningless if the rovers do not have the energy reserves to execute the plan. Computational resource requirements are also important. The most recent Mars rover, *Curiosity*, has only 2 GB of memory. Even if three-quarters of the system memory is available for autonomy (a poor assumption), many test problems will be unsolvable with the on-board memory.

The number of problems solved, average computation time for solved problems, and average required memory for solved problems are presented in Figures VI.1, VI.2, and VI.3, respectively. Error bars are 25^{th} and 75^{th} percentile of the respective data. The results demonstrate the complexity of PA, the benefits of CFP to reduce the required computational resources, and the ability for RPCA to correct CFP's nonexecutable coalition problem. The results for TF were mixed. About half the problems benefitted from TF, while the other half suffered. PA[ITSAT] was able to solve Rovers problems only with ITSAT as the underlying planner, while CFP and RPCA were able to solve problems using both ITSAT and POPF. The results supported the use of RPCA when plans must be developed quickly or by systems with limited memory, while PA[ITSAT]

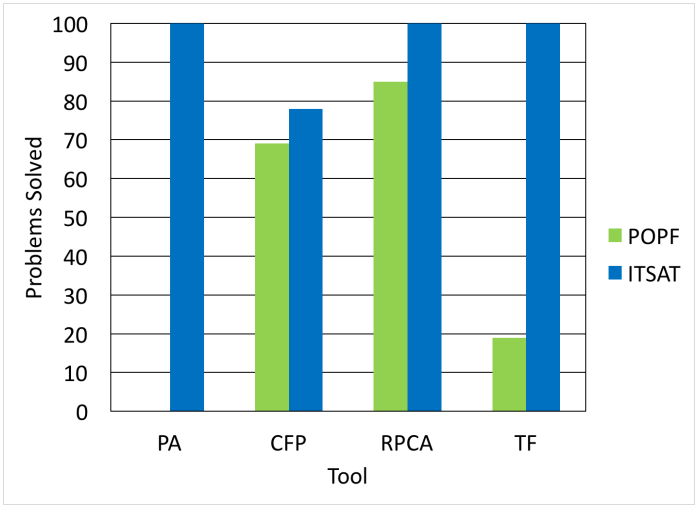


Figure VI.1: Total Number of Rovers Problems Solved by each Tool with each Planner

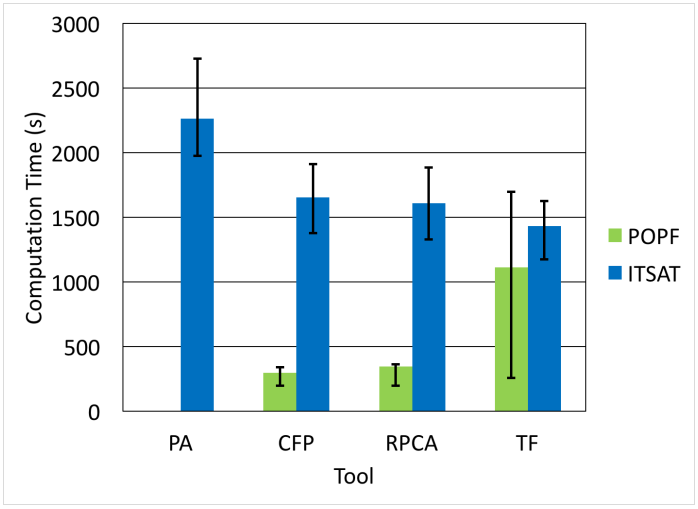


Figure VI.2: Average, 25th, and 75th Percentile Computation Time required for Solved Rovers Problems

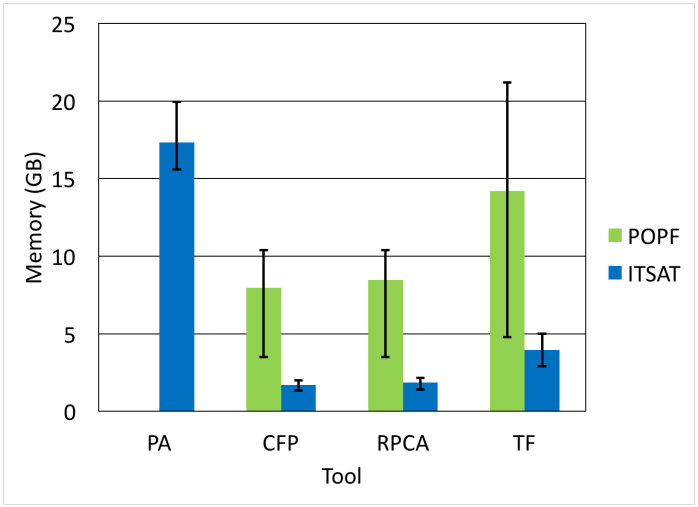


Figure VI.3: Average, 25th, and 75th Percentile Memory required for Solved Rovers Problems

is recommended when plan makespan is of higher importance than the amount of time and memory required to derive the plan. Finally, PA[POPF] is a poor choice for problems of the size used in the experiments. Applying TF to the problems averaged lower makespan plans, but higher average memory usage. Using two different classes of planners supports the planner agnostic nature of the tools developed in this dissertation.

VI.2 Blocks World

The Grand Coalitions in the Blocks World domain were a randomly generated set of robotic arms. Four types of end effectors were used: friction, suction, magnetic, and encompass. Each Grand Coalition had between four and eight arms, with each arm averaging two end effectors. The Grand Coalitions required at least two arms with each end effector to guarantee the ability to execute each mission. The generated Grand Coalitions were manually validated as possessing the required end effectors. If a Grand Coalition was deficient, then the least capable arm in the Grand Coalition was augmented with the missing end effector(s). Each mission's goal state description required a random rearrangement of the blocks from the initial block stacks into an equal number of block stacks. The problems generated from the same mission differ in the number of arms and the number and types of end effectors on the arms. The problems generated from the same Grand Coalition differ in the number of blocks and the goal state description.

Summary capability data for the ten generated Grand Coalitions is presented in Table VI.18. Each robotic arm can have multiple end effectors; thus, the total capabilities for each Grand Coalition will be greater than the number of arms. The minimum capabilities required to guarantee a Grand Coalition can complete a mission is two arms per end effector type. The Grand Coalitions ranged from 4 to 8 arms, with an average of 6.5 arms. Each arm averaged 2.6 end effectors. The utility of a Grand Coalition for any given Mission is hard to predict, given that it is dependent on the distribution of the capabilities and the blocks to be manipulated. Grand Coalition 5 may appear to be the least capable, because it has only one more capability (magnetic) than

Grand Coalition	Number of Arms	Friction	Suction	Encompass	Magnetic	Total
1	6	2	3	4	4	13
2	8	3	7	7	3	20
3	5	2	3	4	2	11
4	5	4	2	4	2	12
5	4	2	2	2	3	9
6	8	5	6	8	5	24
7	7	5	3	4	7	19
8	7	5	5	4	2	16
9	7	6	6	4	6	22
10	8	6	7	3	7	20

Table VI.18: Number of arms with each end effector per Grand Coalition

Mission	Friction	Suction	Encompass	Magnetic	Total Blocks
1	4 (3)	4 (2)	1 (0)	3 (1)	12 (6)
2	2 (0)	5 (1)	2 (1)	0 (0)	9 (2)
3	6 (2)	2 (1)	5 (0)	2 (1)	15 (4)
4	6 (3)	2 (2)	5 (3)	2 (1)	15 (9)
5	2 (0)	4 (1)	3 (1)	0 (0)	9 (2)
6	4 (1)	2 (0)	4 (1)	2 (1)	12 (3)
7	6 (1)	4 (1)	3 (0)	2 (0)	15 (2)
8	3 (2)	4 (2)	5 (1)	3 (2)	15 (7)
9	0 (0)	2 (0)	4 (2)	6 (2)	12 (4)
10	2 (1)	1 (1)	3 (0)	3 (1)	9 (3)

Table VI.19: Number of blocks requiring each type of end effector by Mission. The number of blocks that are double-weight blocks are indicated in parentheses.

the minimum required capabilities, but this assumption is not valid in the general case. A Mission consisting solely of blocks requiring magnetic end effectors will be easier for Grand Coalition 5, than Grand Coalitions 3, 4, and 9 due to having an additional arm with a magnetic end effector (Grand Coalition 5 has three arms with a magnetic end effector, while the other three only have two).

The ten generated Missions are summarized in Table VI.19. The number of blocks by type are presented, with the number of such blocks that are double-weight blocks in parentheses. Mission 1, for example, includes 12 blocks, six of which are double-weight blocks, and four of those 12 blocks require friction end effectors, with three of those four being double-weight blocks. The Missions averaged 4.1 stacks of blocks in the initial state. The Mission initial states included between three and five stacks of blocks, with each stack having three blocks, for a total of nine to fifteen blocks.

VI.2.1 Planning Alone

Both PA[TFD] and PA[COLIN] solved some Blocks World problems, but failed to solve most of the problems, shown in Table VI.20. PA[TFD] solved 40 problems, with 32 problems exceeding the time limit and 28 problems exceeding the memory limit. PA[COLIN] solved 30 problems, while 17 problems failed due to exceeding the time limit and 53 problems failed due to memory limitations. Both computational resource metrics were highly variable for both PA[TFD] and PA[COLIN]. Detailed results for PA[TFD] and PA[COLIN] are shown in Appendices D.1 and D.2, respectively.

Planner	Plans Found	Total Failures	Time Fails	Memory Fails
TFD	40	60	32	28
COLIN	30	70	17	53

Table VI.20: Number of Blocks World problems solved and failed with Planning Alone per planner

Planner	Makespan	Action Executions	Time (s)	Memory (GB)
TFD	35.0 (10.9)	71.3 (20.4)	342.9 (602.9)	5.37 (10.19)
COLIN	45.1 (11.3)	59.0 (13.2)	464.7 (533.4)	13.65 (11.88)

Table VI.21: Descriptive statistics for makespan, action executions, time required, and memory required per solved Blocks World problem using Planning Alone with each planner.

Planner	Mission	Grand Coalition	Makespan	Action Executions	Time (s)	Memory (GB)
TFD	2	4	39	75	569.1	11.21
	4	8	Time Failure			
	5	7	32	68	1666.7	23.33
	8	6	Time Failure			
COLIN	2	4	36	44	373.4	15.39
	4	8	Memory Failure			
	5	7	Memory Failure			
	8	6	Time Failure			

Table VI.22: Makespan, action executions, time required, and memory required for four example Blocks World problems with Planning Alone

The makespan, action executions, and computational resources required descriptive statistics are presented in Table VI.21. PA[TFD] derived plans averaged a makespan of 35.0 and 71.3 action executions, and required 342.9 seconds and 5.37 GB of memory to derive. The plans derived by PA[COLIN] had an average makespan of 45.1 and 59.0 action executions, while requiring 464.7 seconds of computation time and 13.65 GB of memory. The computational resource requirements for both configurations had high variability.

Full results for four selected problems for each underlying planner are provided in Table VI.22. Problem 2-4 was selected for the matching capabilities of the Mission and Grand Coalition. Mission 2 has five blocks requiring suction end effectors and Grand Coalition 4 has seven arms with access to magnetic end effectors. Both PA[TFD] and PA[COLIN] solved Problems 2-4. PA[TFD] derived a plan with a makespan of 39, required 569.1 seconds and 11.21 GB of memory, while PA[COLIN] derived a plan with a makespan of 36, required 373.4 seconds and 15.39 GB of memory. Problem 4-8 is the combination of a Mission with the most blocks and most double-weight blocks and a Grand Coalition with one fewer than the maximum number of arms. PA[TFD] and PA[COLIN] failed to solve the problem due to exceeding time and memory limits, respectively. Problem 5-7 was selected for opposing capabilities. Grand Coalition 7 has seven arms with magnetic end effectors, but Mission 5 does not have any blocks requiring magnetic end effectors. PA[TFD] solved the problem, requiring 1666.7 seconds and 23.33 GB of memory to produce a plan with a makespan of 32, while PA[COLIN] exceeded the memory limit. Problem 8-6 is the combination of a large Mission with the second highest number of double-weight blocks combined with the Grand Coalition with the most arms and most capabilities. Both PA[TFD] and PA[COLIN] failed to solve the problem due to exceeding the computation time constraint.

Planner	Plans Found	Total Failures	Nonexecutable Coalition	Time Fails	Memory Fails
TFD	27	73	58	15	0
COLIN	38	62	48	2	12

Table VI.23: Number of Blocks World problems solved and failed with CFP per planner

Planner	Makespan	Action Executions	Time (s)	Memory (GB)
TFD	50.2 (11.6)	71.3 (20.4)	490.0 (790.1)	6.32 (8.51)
COLIN	66.9 (30.9)	74.4 (23.2)	155.6 (407.7)	2.17 (3.63)

Table VI.24: Descriptive statistics for makespan, action executions, time required, and memory required per solved Blocks World problem using CFP with each planner.

PA provides varied performance on the Blocks World problems. Less than half the problems were solved, but some problems required large amounts of both time and memory. As with the Rovers Planning Alone results, there is room for improvement, especially in terms of the number of problems solved.

VI.2.2 Coalition Formation then Planning

CFP[TFD] solved fewer problems than PA[TFD], but CFP[COLIN] solved more problems than PA[COLIN], as presented in Table VI.23. Nonexecutable coalitions caused failures for many problems with both underlying planners, 58 for CFP[TFD] and 48 for CFP[COLIN]. The nonexecutable coalitions are a result of planning decisions made for prior tasks; thus, the set of nonexecutable tasks when using each underlying planner are different. Time limits were exceeded for 15 of the problems using CFP[TFD] and for two problems using CFP[COLIN]. Twelve additional problems failed due to exceeding the memory limit while using CFP[COLIN]. Detailed results for CFP[TFD] and CFP[COLIN] are presented in Appendices D.3 and D.4, respectively.

The average makespan, action executions, and computational resource usage for the problems solved with CFP are provided in Table VI.24. CFP[TFD] derived plans with an average makespan of 50.2 and 71.3 action executions, while requiring 490.0 seconds of computation time and 6.32 GB of memory. The plans derived by CFP[COLIN] averaged a makespan of 66.9, 74.4 action executions, and required a computation time of 155.6 seconds and 2.17 GB of memory. The computational resource requirements of both configurations were highly variable.

Planner	Problems	Makespan	Action Executions	Time (s)	Memory (GB)
TFD	17	1.374 (0.487)	1.041 (0.320)	5.066 (10.683)	12.557 (27.477)
COLIN	14	1.291 (0.681)	1.134 (0.466)	0.141 (0.181)	0.091 (0.140)

Table VI.25: Average (and standard deviation) ratio of CFP metric to PA metric for makespan, action executions, time required, and memory required for problems solved by both CFP and PA

Planner	Mission	Grand Coalition	Makespan	Action Executions	Time (s)	Memory (GB)
TFD	2	4	62	89	415.5	8.66
	4	8	Time Failure			
	5	7	Nonexecutable Coalition			
	8	6	Nonexecutable Coalition			
COLIN	2	4	51	53	4.4	0.07
	4	8	93	92	16.5	0.15
	5	7	25	46	6.8	0.16
	8	6	Nonexecutable Coalition			

Table VI.26: Makespan, action executions, time required, and memory required for four example Blocks World problems with CFP

PA[TFD] and CFP[TFD] solved 17 problems in common, while PA[COLIN] and CFP[COLIN] had 14 commonly solved problems. Descriptive statistics for the ratio of the plan metrics for corresponding plans are presented in Table VI.25. The problems solved by CFP[TFD] averaged 37.4% longer makespan with 4.1% more action executions, and required 406% more time and 1156% more memory than the same problem solved by PA[TFD]. CFP[COLIN] had similar results in terms of plan quality, 29.1% higher makespan plans and 13.4% more actions than the PA[COLIN] derived plans, but required only 14.1% of the memory and 9.1% of the time.

The results for the four example problems are presented in Table VI.26. CFP[TFD] solved one fewer problem than PA[TFD]. The plan derived by CFP[TFD] for Problem 2-4 had a 59.0% higher makespan, with 18.7% more action executions than the corresponding plan derived by PA[TFD], but used only 73.0% of the computation time and 77.3% of the memory. Both PA[TFD] and CFP[TFD] failed to solve Problem 4-8 due to exceeding the time limit. Problems 5-7 and 8-6 failed due to coalition formation allocating nonexecutable coalitions. CFP[COLIN] solved the same problem as PA[COLIN], as well as two additional problems. Problem 2-4 solved with CFP[COLIN] derived a plan with 41.7% higher makespan and 20.5% more action executions than the plan derived by PA[COLIN], but required only 1.2% of the time and 0.5% of the memory. CFP[COLIN] was able to solve Problems 4-8 and 5-7 that were not solved by PA[COLIN] due to memory limitations. Problem 8-6 was not solved by CFP[COLIN] due to a nonexecutable coalition. This same problem was also not solved by PA[COLIN] due to exceeding the time limit.

The nonexecutable coalitions in the Blocks World problems are the result of the plans for previous tasks transitioning the initial state used in planning that resulted in a nonexecutable coalition, which can be split into two different cases. The first case is a result of the finite table, which can prevent arms from placing blocks on the table and force arms to place blocks on top of other blocks. Let block a be placed on top of b as a result of planning for earlier tasks and let the task being planned require moving b. If the coalition allocated to the current task is incapable of moving a in order to access b, then the coalition is nonexecutable. The second case

Planner	Plans Found	Total Failures	Time Fails	Memory Fails
TFD	74	26	26	0
COLIN	83	17	3	14

Table VI.27: Number of Blocks World problems solved and failed with RPCA per planner

Planner	Makespan	Action Executions	Time (s)	Memory (GB)
TFD	53.8 (19.5)	83.3 (23.2)	386.7 (650.3)	4.65 (7.53)
COLIN	59.1 (26.6)	68.5 (21.4)	137.8 (363.1)	1.70 (3.23)

Table VI.28: Descriptive statistics for makespan, action executions, time required, and memory required per solved Blocks World problem using RPCA with each planner.

occurred due to arms holding blocks at the end of task planning. One option to avoid the second failure mode is to require agents to satisfy "cleanup goals" before planning each task. Cleanup goals in the Blocks World problems will require that all agents not hold any blocks at the end of each task planning loop. Requiring agents to drop blocks between tasks can detrimentally affect plan quality by requiring subsequent planning to pick up the block again. Correcting nonexecutable coalition failures in CFP requires an algorithmic change to augment the coalition structure with the agents that will make it executable.

VI.2.3 Relaxed Plan Coalition Augmentation

Relaxed Plan Coalition Augmentation solved 85% and 176% more problems than PA[TFD] and PA[COLIN], respectively, as shown in Table VI.27. RPCA[TFD] suffered 26 failures due to exceeding the time limit. Detailed results for RPCA[TFD] are presented in Appendix D.5. Three problems failed due exceeding the time limit with RPCA[COLIN] and fourteen failed due to exceeding the memory limit. See Appendix D.6 for detailed results.

The descriptive statistics for plan makespans, action executions, and the computational resources required are presented in Table VI.28. Plans derived using RPCA[TFD] averaged a makespan of 53.8 and 83.3 action executions, while requiring 386.7 seconds of computation time and 4.65 GB of memory. RPCA[COLIN] derived plans with an average makespan of 59.1 and 68.5 action executions, and required 137.8 seconds of computation time and 1.70 GB of memory to do so. As with the PA and CFP results, the computational resources were highly variable.

Planner	Problems	Makespan	Action Executions	Time	Memory
TFD	35	1.470 (0.502)	1.105 (0.297)	2.943 (7.664)	7.070 (19.767)
COLIN	28	1.145 (0.524)	1.086 (0.401)	0.383 (1.221)	0.286 (0.990)

Table VI.29: Average (and standard deviation) ratio of RPCA metric to PA metric for makespan, action executions, time required, and memory required for problems solved by both RPCA and PA

Planner	Mission	Grand Coalition	Makespan	Action Executions	Time (s)	Memory (GB)
TFD	2	4	62	89	415.5	8.66
	4	8	Time Failure			
	5	7	45	83	1509.2	25.70
	8	6	63	102	145.2	0.41
COLIN	2	4	51	53	4.4	0.07
	4	8	93	92	16.5	0.15
	5	7	25	46	6.8	0.16
	8	6	51	68	186.4	1.35

Table VI.30: Makespan, action executions, time required, and memory required for four example Blocks World problems with RPCA

A comparison of RPCA with the PA tool is presented in Table VI.29. A comparison of RPCA with CFP is not provided because every problem solved by CFP is solved in an identical manner by RPCA. RPCA[TFD] solved greater than 85% more problems than PA[TFD]. The problems solved by RPCA[TFD] averaged 47.0% higher makespans, with 10.5% more action executions and required 194% more time and 607% more memory than the same problem solved by PA[TFD]. RPCA[COLIN] solved 176% more problems than PA[COLIN]. RPCA[COLIN] also averaged worse plan quality (14.5% higher makespan and 8.6% more action executions), than the corresponding PA configuration, but required only 38.3% of the time and 28.6% of the memory.

Results for the four example problems are presented in Table VI.30. Generally, RPCA[TFD] solved three problems, including one problem PA[TFD] was unable to solve. CFP[TFD] derived a plan for Problem 2-4; thus, the RPCA[TFD] results are identical to CFP[TFD]. Problem 4-8 was not solved by PA[TFD], CFP[TFD], or RPCA[TFD], due to exceeding the time limit. Problem 5-7 was solved by RPCA[TFD] and PA[TFD]. RPCA[TFD] derived a plan with a makespan of 45 and 83 action executions, while requiring 1509.2 seconds of computation time and 25.70 GB of memory. The plan derived by PA[TFD] had a makespan 71.1% and action executions 81.9% that of RPCA[TFD], but required 10.4% more computation time and 90.8% the memory. The RPCA[TFD] derived plan for Problem 8-6 had a makespan of 63 and 102 action executions, and required 145.2 seconds of computation time and 0.41 GB of memory. PA[TFD] was unable to solve the problem.

RPCA[COLIN] was able to solve one more problem than CFP[COLIN]. Problem 6-8 was solved by RPCA[COLIN] with a plan makespan of 51, 68 action executions, with a required computation time of 186.4 seconds and 1.35 GB of memory. PA[COLIN] and CFP[COLIN] failed to solve the problem due to exceeding the time limit and a nonexecutable coalition, respectively.

RPCA[COLIN] demonstrated reduced computational resource usage comparable to PA[COLIN], but RPCA[TFD] required more computational resources than PA[TFD]. However, RPCA[TFD] did perform better than PA[TFD] in terms of total problems solved, by solving 85% more problems. RPCA[COLIN] solved

Planner	Plans Found	Total Failures	Time Fails	Memory Fails
TFD	54	46	46	0
COLIN	80	20	2	18

Table VI.31: Number of Blocks World problems solved and failed with TF per planner

Planner	Makespan	Action Executions	Time (s)	Memory (GB)
TFD	54.8 (19.2)	80.7 (23.9)	402.0 (605.8)	5.43 (8.38)
COLIN	60.3 (21.3)	71.1 (18.7)	150.8 (331.8)	2.83 (5.71)

Table VI.32: Descriptive statistics for makespan, action executions, time required, and memory required per solved Blocks World problem using TF with each planner.

more than twice as many problems as its PA counterpart. RPCA was able to exploit the factoring produced by CFP, while correcting for nonexecutable coalitions. RPCA selected the additional agent(s) to augment a coalition; thus, making it executable in both of the nonexecutable coalition cases. RPCA suffered from lower average plan quality relative to Planning Alone, but solved more than twice as many problems, while averaging less memory across all problems solved for both underlying planners and less time when using COLIN.

VI.2.4 Task Fusion

Task Fusion in the Blocks World domain was performed using the coalition similarity score. Coalitions that were executed by coalitions with the highest similarity were selected to merge. Merge candidates had to have at least half of their agents in common in order to be selected.

Fewer problems were solved with TF than with RPCA when using both underlying planners, see Table VI.31, with TF[TFD] resulting in the larger drop. TF[TFD] solved twenty fewer problems than RPCA[TFD] (54 and 74, respectively), while TF[COLIN] solved three fewer problems than RPCA[COLIN] (80 and 83, respectively). Detailed results for TF[TFD] and TF[COLIN] are presented in Appendices D.7 and D.8, respectively.

Descriptive statistics for makespan, action executions, and the computational resources for the solved problems are presented in Table VI.32. The plans derived by TF[TFD] averaged a makespan of 54.8 with 80.7 action executions and required 402.0 seconds of computation time and 5.43 GB of memory. TF[COLIN] produced plans with an average makespan of 60.3 and 71.1 action executions, while requiring 150.8 seconds of computation time and 2.83 GB of memory.

A comparison of TF to PA and RPCA is presented in Table VI.33. PA[TFD] solved 36 of the problems also solved by TF[TFD], but TF[TFD] performed worse on those 36 problems by all metrics. TF[COLIN] used 93.5% of the memory required by PA[COLIN] across the 29 problems, but performed worse in terms

Tool	Planner	Problems	Makespan	Action Executions	Time	Memory
PA	TFD	36	1.530 (0.572)	1.104 (0.323)	3.972 (8.883)	6.465 (12.875)
	COLIN	29	1.274 (0.372)	1.187 (0.306)	1.164 (3.859)	0.935 (3.265)
RPCA	TFD	48	1.078 (0.221)	1.027 (0.192)	4.061 (10.932)	8.469 (22.225)
	COLIN	75	1.093 (0.290)	1.081 (0.247)	4.566 (14.132)	6.617 (15.131)

Table VI.33: Average (and standard deviation) ratio of TF metric to PA and RPCA metrics for makespan, action executions, time required, and memory required for problems solved by both configurations.

Planner	Mission	Grand Coalition	Makespan	Action Executions	Time (s)	Memory (GB)
TFD	2	4	54	79	770.8	15.45
	4	8	Time Failure			
	5	7	48	83	1809.9	25.70
	8	6	99	142	159.3	0.47
COLIN	2	4	62	61	42.2	1.90
	4	8	69	75	37.5	0.53
	5	7	25	46	6.8	0.16
	8	6	82	94	44.9	0.49

Table VI.34: Makespan, action executions, time required, and memory required for four example Blocks World problems with TF

of all other metrics. TF[TFD] and RPCA[TFD] both produced solutions for 48 of the 100 problems, but TF[TFD] performed worse by all metrics for all problems. TF[COLIN] and RPCA[COLIN] solved a common set of 75 problems, with TF[COLIN] averaging worse performance than RPCA[COLIN] across all metrics.

Results for the four example problems are presented in Table VI.34. TF[TFD] solved three of the four problems, while PA[TFD] solved only two of the problems. PA[TFD], CFP[TFD], and TF[TFD] all solved Problem 2-4. TF[TFD] derived a plan with a makespan of 54 and 79 action executions (39 and 75 for PA[TFD], 62 and 89 for CFP[TFD]), and required 770.8 seconds and 15.45 GB of memory (569.1 and 11.21 for PA[TFD], 415.8 and 8.66 for CFP[TFD]). Problem 4-8 was not solved by TF[TFD] due to exceeding the time limit, as was the result with PA[TFD] and CFP[TFD]. The plan derived for Problem 5-7 by TF[TFD] had a makespan of 48 and 83 action executions, while requiring 1809.9 seconds of computation time and 25.70 GB of memory. Both PA[TFD] and RPCA[TFD] derived lower makespan plans (32 and 45, respectively), but only PA[TFD] derived a plan with fewer action executions (68). TF[TFD] required 8.6% more time and 10.2% more memory than PA[TFD] and 19.9% more time and the same amount of memory as RPCA[TFD]. TF[TFD] performed worse than RPCA[TFD] on Problem 8-6 across all metrics.

TF[COLIN] solved all four of the example problems. Plan quality for Problem 2-4 solved by TF[COLIN] was worse by both metrics, but TF[COLIN] required 11.4% the time and 12.3% the memory of PA[COLIN]. Problem 4-8 solved by TF[COLIN] produced a plan with 74.2% the makespan and 81.5% the action executions, but required 155% more time and 253% more memory. Both CFP[COLIN] and TF[COLIN] required less than 1 GB of memory to solve the problem, but PA[COLIN] was unable to solve it with 48 GB of mem-

ory. TF[COLIN] solved Problem 5-7 with the same metric values as CFP[COLIN], due to not fusing any tasks, as such, the problem was solved identically to CFP[COLIN]. TF[COLIN] solved Problem 8-6, but had worse plan quality than the plan derived by RPCA[COLIN] (60.8% higher makespan and 38.2% more action executions). However, TF[COLIN] solved the problem with 24.1% the time and 36.3% the memory required by RPCA[COLIN].

As with the Rovers domain, the TF results for Blocks World are not promising. Also similar to the Rovers domain, neither of the underlying planners are optimal, so the plans produced are not optimal. It is possible that an optimal planner will produce better results when used with TF, by balancing the quick solutions of RPCA with the higher plan quality of PA.

VI.2.5 Discussion

Blocks World in its classic form is a consistent benchmark for planning algorithms. The domain modifications to support this research include multiple heterogeneous agents. The tasks in Blocks World are highly dependent upon each other, as are the agents. The dependency is clear from the CFP results, as each coalition is executable from the problem's initial state, but can become nonexecutable as other tasks are planned, leaving blocks in arms or on other blocks.

Assume the Blocks World domain models a real world construction project. Plan makespan in the construction metaphor measures the amount of time between project start and project finish. The number of action executions roughly measures the number of worker-hours required to complete a project, although this is an inexact measure, because each action has different durations. Companies are interested in minimizing both project completion time and worker-hours. Computational resources are unlikely to be as important as plan quality in this metaphor. Construction timelines can be planned months in advance; therefore, plan computation time is not very constrained. Furthermore, computers with large amounts of memory can be used, as such, allotted memory limits can be increased to limit the degree to which memory is a constraint.

The number of problems solved, average computation time for solved problems, and average required memory for solved problems are presented in Figures VI.4, VI.5, and VI.6, respectively. Error bars are 25^{th} and 75^{th} percentile of the respective data. The results from Blocks World are similar to Rovers. PA solves some problems, but fails to solve with many of them. CFP solves problems quicker and with less memory than PA, but suffers from nonexecutable coalitions. RPCA provides the best performance by building on CFP and correcting the nonexecutable coalitions. As with Rovers, TF performs poorly with Blocks World.

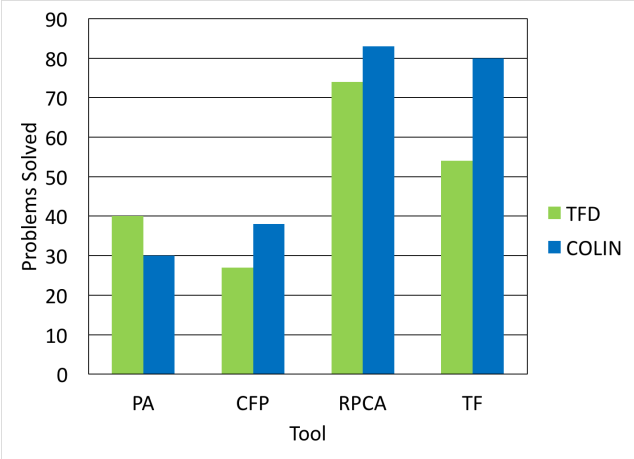


Figure VI.4: Total Number of Blocks World Problems Solved by each Tool with each Planner

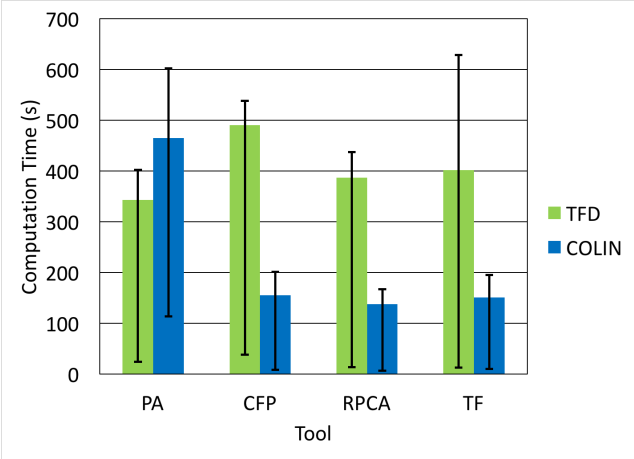


Figure VI.5: Average, 25th, and 75th Percentile Computation Time required for Solved Blocks World Problems

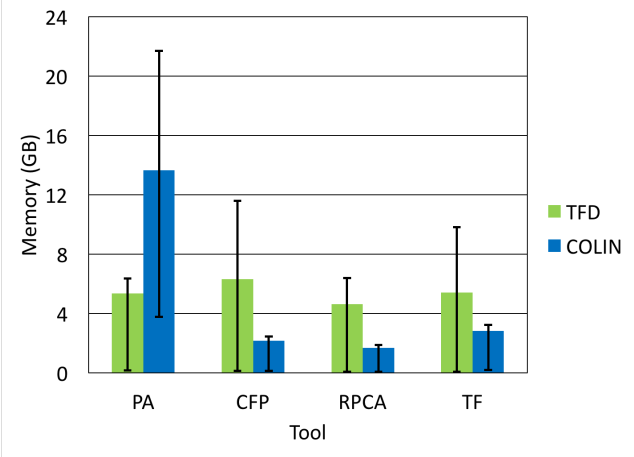


Figure VI.6: Average, 25th, and 75th Percentile Memory required for Solved Blocks World Problems

Grand Coalition	ATL	LAX	ORD	DFW	DEN	SEA	JFK	Total
1	4 (2)	2 (1)	7 (3)	3 (1)	2 (2)	2 (2)	4 (3)	24 (14)
2	4 (3)	1 (1)	2 (1)	2 (1)	4 (2)	2 (1)	3 (3)	18 (12)
3	2 (1)	6 (3)	5 (3)	5 (3)	1 (1)	1 (1)	2 (1)	22 (13)
4	8 (3)	4 (3)	3 (3)	5 (3)	4 (3)	4 (3)	2 (2)	30 (20)
5	2 (1)	2 (1)	4 (2)	5 (3)	3 (3)	4 (3)	5 (3)	25 (16)
6	2 (2)	4 (2)	3 (2)	3 (1)	6 (3)	3 (2)	2 (2)	23 (14)
7	1 (1)	2 (1)	2 (2)	2 (2)	1 (1)	4 (2)	6 (3)	18 (12)
8	5 (3)	4 (2)	3 (3)	1 (1)	4 (2)	3 (2)	3 (2)	23 (15)
9	1 (1)	2 (1)	6 (2)	4 (3)	3 (2)	3 (2)	3 (2)	22 (13)
10	3 (3)	4 (3)	3 (1)	2 (2)	2 (2)	3 (3)	5 (3)	21 (16)

Table VI.35: Number of planes in each hub and its associated spokes in the initial state. Number which are short-range planes is indicated in parentheses.

Mission	Tasks	Passengers	Cargo	Total
1	19	65	66	131
2	16	57	55	112
3	16	56	55	111
4	16	56	57	113
5	16	60	57	117
6	19	68	65	133
7	18	64	60	124
8	18	63	66	129
9	16	57	59	116
10	16	55	57	112

Table VI.36: Number of passengers and cargo to be transported for each task

VI.3 Zenotravel

The Grand Coalitions in the Zenotravel domain were a randomly generated set of long-range planes and short-range planes. Each hub city had between one and three short-range planes and five to ten long-range planes that were randomly distributed across the hubs in each mission's initial state. The generated missions use the same set of hubs and spokes, based on actual airports in the United States of America and the distances between each. Seven hub airports and forty-two spoke airports were selected, with each hub having between five and seven associated spokes. The short-range planes had a capacity of four passengers and four cargo units, while the long-range planes had a capacity of eight passengers and eight cargo units.

Summary data for the ten generated Grand Coalitions is presented in Table VI.35. The table lists the number of long-range planes in each hub and the total number of short-range planes in each hub or a spoke connected to the hub in the initial state. Grand Coalition 3, for example, has a total of five planes, two of which are short-range planes, at ORD in the initial state, with a total of 22 planes, thirteen of which are short-range, distributed around the country. The Grand Coalitions ranged between 5 and 10 long-range planes and between 12 and 20 short-range planes.

Planner	Plans Found	EHC Fails
COLIN	87	13

Table VI.37: Number of Zenotravel problems solved and failed with CFP

Planner	Makespan	Action Executions	Time (s)	Memory (GB)
COLIN	1560.5 (451.7)	505.4 (49.2)	499.7 (219.9)	0.16 (0.02)

Table VI.38: Average (and standard deviation) makespan, action executions, time required, and memory required for solved Zenotravel problem using CFP

The ten generated Missions are summarized in Table VI.36. Mission 8, for example, consists of 18 tasks transporting 63 passengers and 66 units of cargo. The missions ranged between a total of 111 and 133 passengers and cargo. The missions consisted of an average of 60.1 passengers and 59.7 units of cargo that were spread over 17 tasks.

VI.3.1 Planning Alone

The computational complexity of PA is further demonstrated by attempting to solve the Zenotravel problems. PA[COLIN] was unable to solve any of the problems within the time limit. Ten of the problems were attempted without a time limit, but all failed after about 90 minutes due to exceeding the memory limit.

VI.3.2 Coalition Formation then Planning

Thirteen of the 100 problems failed with CFP[COLIN] due to using the incomplete EHC-based version of COLIN, while 87 problems were solved, as presented in Table VI.37. The makespan, action executions, and the computational resources descriptive statistics are presented in Table VI.38. The plans derived by CFP[COLIN] averaged makespans of 1560.5 with 505.4 action executions, and required 499.7 seconds of computation time and 0.16 GB of memory. The low memory usage is a result of using an EHC-based algorithm; however, the EHC-based algorithm is the root of the failures to solve thirteen problems. Detailed results for CFP[COLIN] are presented in Appendix E.1.

The Zenotravel domain is the first domain for which no problems failed due to nonexecutable coalitions. Every city is reachable from every other city and the tasks form a partition of the sets of all passengers and cargo. The only way for a coalition to be nonexecutable is for an iterative task plan to transport a passenger or cargo to a city that the coalition cannot reach. Such a case is highly unlikely in EHC-based COLIN, as an action to transport a non-task cargo or passenger is likely to have the same or higher heuristic value as its parent node; thus, it will never be expanded.

Planner	Plans Found	EHC Fails
COLIN	85	15

Table VI.39: Number of Zenotravel problems solved and failed with TF

Planner	Makespan	Action Executions	Time (s)	Memory (GB)
COLIN	1472.6 (379.8)	505.3 (43.3)	319.4 (135.4)	0.17 (0.06)

Table VI.40: Average (and standard deviation) makespan, action executions, time required, and memory required for solved Zenotravel problem using TF

VI.3.3 Relaxed Plan Coalition Augmentation

Relaxed Plan Coalition Augmentation is only beneficial when nonexecutable coalitions exist. Due to no Zenotravel problems failing for that reason when using CFP[COLIN], no results are presented for RPCA, as the results are identical to CFP.

VI.3.4 Task Fusion

Zenotravel used the coalition similarity score to select the tasks to fuse. Fusing tasks in the Zenotravel domain maximizes the used capacity of each plane. For example, a coalition with a task to transport passengers from ATL to LAX and from DEN to LAX can fly a route from ATL to DEN to LAX, assuming the plane has the capacity to transport the required passengers and cargo. Fifteen problems failed due to using EHC and plans were found for 85 problems, as presented in Table VI.39. The number of successes and failures with TF[COLIN] is similar to the CFP[COLIN] results, with CFP[COLIN] solving two additional problems. Detailed results for TF[COLIN] are presented in Appendix E.2.

The descriptive statistics for the makespan, action executions, and required computational resources are shown in Table VI.40. TF[COLIN] derived plans with an average makespan of 1472.6 and 505.3 action executions. Average time for TF[COLIN] to derive a plan was 319.4 seconds and required 0.17 GB, on average.

TF[COLIN] solved 76 problems that were also solved by CFP[COLIN]. A comparison of the metrics for those 76 problems is presented in Table VI.41. The plans derived by TF[COLIN] averaged 2.2% lower makespan and had the same number of action executions, while requiring 32.1% less computation time and 5.1% more memory. Each Problem had at least 16 tasks, and reduced to as few as eight tasks after the fusion

Problems	Makespan	Action Executions	Time	Memory
76	0.978 (0.236)	1.000 (0.012)	0.679 (0.193)	1.051 (0.342)

Table VI.41: Average (and standard deviation) ratio of TF metric to CFP metric for makespan, action executions, time required, and memory required for problems solved by both TF and CFP

Mission	Grand Coalition	Tool	Makespan	Action Executions	Time (s)	Memory (GB)
1	1	CFP	1110	451	255.3	0.18
		TF	824	452	207.8	0.18
5	3	CFP	3673	532	502.8	0.15
		TF	2190	521	388.6	0.15
8	8	CFP	1136	519	405.6	0.18
		TF	1625	524	446.6	0.18
10	7	CFP	1227	439	344.8	0.14
		TF	1420	438	207.5	0.14

Table VI.42: Comparison of four problems solved with CFP[COLIN] and TF[COLIN]

step. The reduced computation time is partially due to the lower number of tasks. The time required to plan each individual task in the Zenotravel domain was much faster compared to the time required to merge the plans and the overhead necessary to initialize each planning step (e.g., action grounding).

Two examples of TF producing lower makespan plans and two examples of TF producing higher makespan plans are presented in Table VI.42. TF[COLIN] derived a plan for Problem 1-1 with 25.8% lower makespan and one additional action execution compared to CFP[COLIN]. The memory usage was the same, but TF[CFP] required 81.4% of the time required by CFP[COLIN]. Problem 5-3 was solved by TF[COLIN] with 59.6% of the makespan and 97.9% of the action executions in the plan derived using CFP[COLIN], while reducing computation time by 22.7%.

CFP[COLIN] derived a plan for Problem 8-8 with 69.9% of the makespan and 99.0% of the action executions of the plan derived using TF[COLIN]. CFP[COLIN] required 90.8% of the time required by TF[COLIN]. The plan CFP[COLIN] produced for Problem 10-7 had a 13.6% lower makespan and had one more action execution, when compared to the plan produced by TF[COLIN]. However, TF[COLIN] required 60.2% of the computation time required by CFP[COLIN].

VI.3.5 Discussion

Data for average computation time and memory for CFP and TF to derive solutions are presented in Figures VI.7 and VI.8, respectively. Error bars are 25^{th} and 75^{th} percentile of the respective data. TF averaged higher memory usage, but lower time requirements than CFP. Factoring the Zenotravel problems into tasks and limiting the number of agents considered at any one time allows the EHC algorithm to use very low amounts of memory when using both the TF and the CFP tool. The PA tool includes a much larger number of agents and possible actions each agent can perform; therefore, the EHC algorithm can easily become stuck in heuristic plateaus, as it did in attempting to solve each of the 100 problems.

The Zenotravel experiments were designed to model air travel in the US. While this is not the solution employed by major airlines, it is an interesting experiment for domain-independent planning. Air travel is

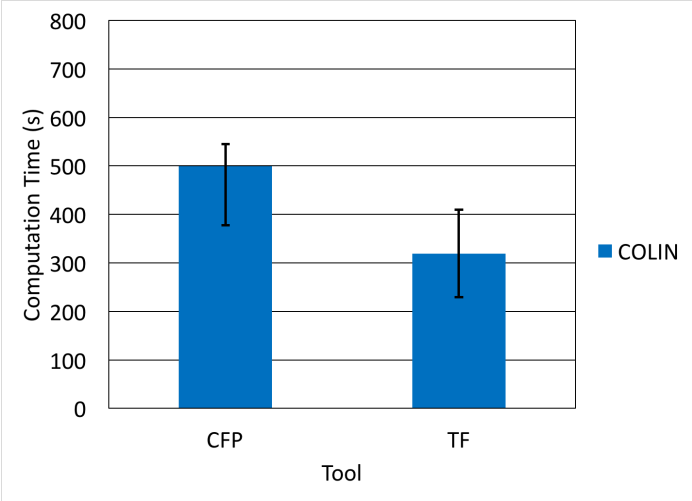


Figure VI.7: Average, 25th, and 75th Percentile Computation Time required for Solved Zenotravel Problems

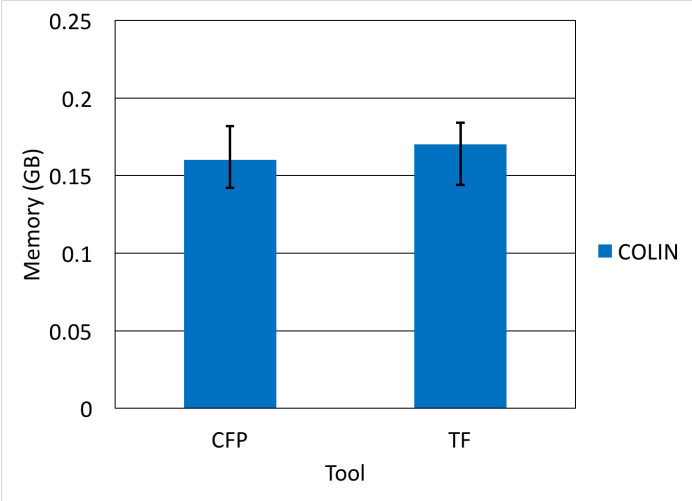


Figure VI.8: Average, 25th, and 75th Percentile Memory required for Solved Zenotravel Problems

a big business, and very large amounts of computation time and memory can be dedicated to solving the problem. Furthermore, plan schedules are created months ahead of time. Eliminating limited computational resources as a constraint leaves plan quality as the determining factor in selecting a solution method. A low makespan implies customers arrived at their destination faster. Fewer action executions imply fewer required flights and; thus, lower financial costs.

VI.4 First Response

The First Response domain is unique in that there is a single Mission and Grand Coalition. The Problem is based on the aftermath of the 2011 tornado in Tuscaloosa, AL. The Grand Coalition, summarized in Table VI.43, consists of the Police, Fire, and EMS agents, their starting locations, and each groups' bases of

operations. Two EMS human responders and three EMS ground robots are deployed, starting from the hospital. The Fire department includes two humans and a ground robot starting at the fire station. The Police responders include two humans, two quadrotor UAVs, and a ground robot that are all positioned at a mobile police command station setup immediately after the tornado cleared the area.

The Mission environment includes the location of ten victims, two gas leaks, four blocked roads (due to debris or downed power lines), a destroyed pharmacy, and two pawn shops. The designed Mission is split into seven tasks, summarized in Table VI.44. Two victim support tasks, each with five victims, must be completed by EMS and Police agents. The victims were divided into tasks based on their initial location. Four adjacent roadway sections must be cleared in one task. The two gas leaks are split into separate tasks due to being spatially distant. Pharmaceutical drugs must be cleared from the pharmacy store debris and secured at the local hospital in order to prevent the drugs from being looted. Finally, two pawn shops must be searched and the retrieved controlled items (e.g., weapons and ammunition) must be secured at the police base.

VI.4.1 Planning Alone

Results from the previous domains indicate that PA will take a large amount of time and memory relative to the other tools, and this result is valid in this domain. PA[COLIN] failed to solved the Problem due to exceeding the memory limit after 90 minutes of computation time.

VI.4.2 Coalition Formation then Planning

CFP[COLIN] was also unable to solve the Problem, due to a nonexecutable coalition being allocated to one of the victim support tasks. The victim support task was attempted by EMS and Police agents, but the Problem includes a blocked road preventing agents from reaching one of the victims. EMS agents cannot triage the victim and Police agents cannot lead the victim to the hospital unless they can reach the victim. The road must be cleared for the agents to reach the victim. Only the Fire department agents can execute the action that clears roads so that responders and ground robots can move about the environment, as such, the injured victim was unreachable by EMS agents and planning for the task failed.

One of the tasks for the Fire department is ensuring that main roads are clear of debris and downed power lines; clearing the side streets is intentionally not included. A real response to such a disaster will be

Group	Agents
Police	2 humans, 2 quadrotors, 1 ground robot
Fire Department	2 humans, 1 ground robot
EMS	2 humans, 3 ground robots

Table VI.43: Grand Coalition in the First Response Problem

Task Type	Tasks in Problem	Description
Victim Support	2	Ten victims, divided spatially into two tasks
Roads	1	Four adjacent blocked roadway sections in one task
Gas Leaks	2	Two spatially distant gas leaks
Pharmacy	1	One pharmacy with pharmaceutical drugs
Pawn Shop	1	Two spatially close pawn shops

Table VI.44: Tasks in the First Response Problem

concerned with victims incapable of helping themselves. Main roads are more likely to open up new or faster routes to severely injured victims. Fire department agents can be included in the victim support task, but the agents will be idle if no roads to victims are blocked; therefore, Fire department agents in the First Response domain focus on clearing the main roads and sealing gas leaks to prevent another disaster.

VI.4.3 Relaxed Plan Coalition Augmentation

Not including Fire department capabilities in the required capabilities of the victim support task and inserting blocked roads at specific side streets forced RPCA to be used. The relaxed plan generated for the task includes a Fire department agent clearing a road for EMS agents and the action to clear roads is executable only by Fire department agents. The clear roads action was recognized as one that the allocated coalition of EMS and Police agents was unable to execute; therefore, a Fire department robot was allocated to the coalition in order to execute the clear road action. Planning the victim support task was reattempted with the coalition augmented with the Fire department robot, and a plan was derived. The solution produced by RPCA[COLIN] had a makespan of 216.2, 186 action executions, required 23.2 seconds and 0.24 GB of memory.

VI.4.4 Task Fusion

The coalition assistance score was used to select tasks to fuse in First Response. Task Fusion selected four tasks to combine into two tasks. The first two tasks selected were the two gas leak tasks. Both tasks require Fire department humans to seal the leak. The coalition formation problem does not consider spatial constraints, but planning the two tasks together allows the planning algorithm to consider the distance each unit must travel to reach the gas leaks. The second set of tasks fused is one of the victim support tasks with the pharmacy clearing task. Both tasks require the EMS agents, but for different reasons. The victim support task requires a human EMS agent to triage the victim, and possibly an EMS robot to transport the victim to the hospital. The pharmacy clearing task requires a human EMS agent to identify the controlled prescription drugs in the pharmacy building debris, place the drugs in the robot, and send the robot to the hospital in order to store the drugs in a secure facility. Both tasks require EMS agents. When the triage portion of the victim support task is complete, EMS human agents and EMS robots not transporting victims are free to

immediately move on to the pharmacy clearing task, or vice-versa, and a higher quality plan.

The solution derived by TF[COLIN] had a makespan of 211.2, 187 action executions, and required 799.1 seconds and 4.90 GB of memory. The derived plan has a 2.3% lower makespan than RPCA[COLIN], but requires one additional action execution. However, TF[COLIN] required 34.4 times the computation time and 20.7 times the memory when compared to RPCA[COLIN]. The computational resources required to derive a plan with TF[COLIN] are within the computational resource limits of a laptop, or other portable computing device, that can be carried into the disaster area.

VI.4.5 Discussion

Applying planning to First Response domains requires many considerations. A shorter plan makespan means

- EMS agents get to victims sooner, which may mean more victims survive,
- roads become traversable earlier, which increases the ability of all agents to get where they need to be,
- less gas is leaked into the environment, which reduces the possibility of an explosion, and
- criminals have a shorter time interval during which to loot controlled drugs and weapons from pharmacies and pawn shops.

Additionally, fewer action executions can reduce the strain on the communications network, which is likely to have been hastily established if existing cellular networks were overloaded or otherwise disabled.

Computational resources are also an important consideration. Mobile devices are unlikely to have the memory and computing resources required to solve larger problems. Cellular networks, if available after a mass casualty incident, can be used to communicate problems and plans. Solving plans remotely and communicating the plans will allow computers with the necessary computational resources to solve the problems.

A more important (and unique) metric in the First Response domain was not measured: number of victims saved. Waiting to develop a better plan is meaningless if the victims are dead; however, quickly developing a poor quality plan is also not good. The computation time and the number of victims saved must be balanced.

VI.5 Summary

The developed planning tools were evaluated with four domains. The PA tool averaged the highest quality plans, but also consumed the highest amount of computational resources. Solving some of the problems with PA required large amounts of memory, and solving most of the problems required more memory than is readily available on mobile platforms. Applying PA for those problems requiring excessive amounts of memory will require deriving plans remotely before communicating the results to the coalition. CFP was

able to solve problems with much fewer computational resources than PA, but suffered from nonexecutable coalitions, resulting in no solution being derived when a solution was known to exist. CFP also produced plans of lower quality compared to those plans derived using PA. Two tools were designed to address the nonexecutable coalition problem and the suboptimal plan quality problems. RPCA dominated CFP, in that it performed identically to CFP in all the problems solved by CFP, but was also able to solve most of the problems that failed with CFP due to a nonexecutable coalition. RPCA solved many more problems than PA across all domains (with the exception of Zenotravel, for which RPCA was not required), all while using fewer computational resources than PA.

The TF results were mixed, resulting in most of the TF derived plans being of worse quality than those derived with PA or RPCA. Using satisficing planners, as opposed to optimal planners, is a possible explanation for the worse performance for some of the problems. Optimal planners, however, do not exist for the expressive domains used in this dissertation. Optimal planners do exist for classical planning problems and may provide better results. Different heuristic functions for predicting the utility of fusing a pair of tasks will provide differing results due to fusing different task pairs. Drawing from the results of RPCA, a relaxed plan for each coalition to complete its task from the problem initial state can provide more information as to whether or not a pair of coalitions will interact. Further analyses will be required to explore the effects of optimal planners and different fusion prediction functions.

CHAPTER VII

Conclusion

VII.1 Dissertation Summary

The novel Hybrid Mission Planning with Coalition Formation problem formalization combining aspects of coalition formation, multi-agent planning, temporal planning, and continuous planning was developed. The problem definition extends the coalition formation problem by introducing plans for how coalitions will complete their allocated tasks and addresses planning problem complexity by considering different levels of abstraction, from concrete actions and goal constraints to abstract capabilities. The more abstract results from coalition formation can guide the planning problem by selecting an appropriate subset of agents for each task.

Four domains, Rovers, Blocks World, Zenotravel, and First Response were used to explore the aspects of the HMPCF problem formalization and how they relate to one another. The Rovers, Blocks World and Zenotravel domains used in this dissertation were modified versions of existing planning domains used extensively in planning literature. The First Response domain is a new domain inspired by the 2011 tornado in Tuscaloosa, AL. Each domain includes expressive domain features (e.g., durative actions, continuous state variables) and heterogeneous agents. Problems from each domain were randomly generated for use in an extensive experimental analysis.

Four tools were developed to solve the HMPCF problem: Planning Alone, Coalition Formation then Planning, Relaxed Plan Coalition Augmentation, and Task Fusion. Planning Alone performed a translation from the HMPCF problem to a single-agent planning problem, the only known existing method for solving problems that have expressive domain models. The results produced by Planning Alone served as the baseline comparison for the results produced by the new tools. Coalition Formation then Planning applied coalition formation to the grand coalition, the task set, and the capability mappings in order to reduce the computational complexity of finding a solution to the HMPCF problem. Generally, Coalition Formation then Planning derived a plan with fewer computational resources than the corresponding problem solved by Planning Alone. However, Coalition Formation then Planning encountered numerous nonexecutable coalitions, coalitions allocated to tasks that they cannot complete, which inspired the Relaxed Plan Coalition Augmentation tool. Relaxed Plan Coalition Augmentation analyzed a relaxed plan for a failed task in order to select agent(s) to augment the nonexecutable coalition and make the coalition executable. Relaxed Plan Coalition Augmentation solved many more problems than Planning Alone, while requiring much lower computational resources; however, Relaxed Plan Coalition Augmentation produced plans of worse quality than Planning

Alone. The Task Fusion tool attempted to address the lower plan quality problem by selecting tasks to combine in order to support joint planning. The results for Task Fusion were mixed, with Task Fusion producing higher quality plans for some problems and lower quality plans for others. The computational resources required when solving problems with Task Fusion were also mixed.

VII.2 Contributions

This dissertation results in several novel contributions in the field of multi-agent systems and planning. The primary contributions are the Coalition Formation then Planning and the Relaxed Plan Coalition Augmentation tools. The secondary contributions include the Task Fusion tool, a new set of expressive test domains, and the extensions to PDDL required to create MACPDDL.

VII.2.1 Coalition Formation then Planning Tool

The Coalition Formation then Planning tool presents a major step forward in multi-agent planning. Existing research in multi-agent planning focuses on individual agents completing tasks or makes classical planning assumptions. Real-world planning problems, especially as robots become more capable and work alongside humans, will require humans and robots to cooperate on joint tasks to achieve their goals; restricting tasks to only those that a single agent can complete is far too large of a constraint for many real world situations. Existing multi-agent planners also make classical planning assumptions, including instantaneous actions and boolean state spaces. Real-world actions execute over a time interval and accurate models of the world must include continuous variables. The assumptions made by multi-agent planners greatly reduce the planning problem complexity, but the decrease in complexity comes at the cost of lower fidelity domain models and; thus, the plans are based on a low fidelity model of the environment and are more likely to fail when executed. The reduction in computational complexity as a result of applying Coalition Formation then Planning allows highly expressive domain models to be used in planning; thus, the derived plans are more likely to successfully execute.

VII.2.2 Relaxed Plan Coalition Augmentation Tool

The Relaxed Plan Coalition Augmentation modification greatly improves the performance of the Coalition Formation then Planning tool. Coalition Formation then Planning is reliant on coalition formation to produce an executable coalition structure, but coalition formation cannot guarantee each coalition it allocates for a task will be able to execute its assigned task. Relaxed planning is computationally simple relative to solving the original problem and it provides an intermediate level of abstraction between coalition formation and planning, allowing coalition formation to inform planning, and vice versa. A relaxed plan cannot be executed,

but it provides clues as to the set of actions a successful coalition must be able to execute in a satisficing plan. These clues inform how coalitions must be modified in order to produce a plan that is executable.

VII.2.3 Task Fusion Tool

Task Fusion can increase plan quality, while remaining within computational resource constraints. Coalition Formation then Planning solves many problems, while using far less computational resources than are available, while Planning Alone often requires more computational resources than are available. Task Fusion, applied properly, can use allotted computational resources to produce high quality plans. The fusion heuristics use the coalition makeup and coalition capabilities to predict the coalition-task pairs most likely to benefit from fusion. Coalition assistance attempts to maximize positive interactions by allowing coalition with similar capabilities to assist each other. Coalition similarity allows the planning algorithm to consider the spatial and temporal constraints that come about as a result of coalition overlap. Further research is required in order to provide automated guidance on when and how to apply Task Fusion.

VII.2.4 Planning Domains and Language Extensions

A challenge in this dissertation was identifying planning domains representative of the mission complexity needed for real world domains, such as first response domains. The existing research planning domains have been used for decades with little modification. The most popular domains are simple, which is excellent for purposes of describing and understanding algorithms, but are low-fidelity representations and are poor approximations of real world problems. The most egregious of their faults is the lack of heterogeneous multi-agent domains. The Rovers domain was the only domain that include heterogeneous agents that was applicable for analyzing the developed tools. Blocks World and Zenotravel were significantly modified as part of this dissertation in order to add continuous state variables and heterogeneous agents. The new First Response domain is the only known domain to represent the breadth of complexity that exists in this type of real world problem domain. The First Response domain was developed to incorporate multiple heterogeneous agents, durative actions, continuous state variables, continuous effects, and required concurrency. No existing domain combines this set of expressive modeling features into a single domain.

VII.2.5 Multi-Agent Capabilities and Planning Domain Definition Language

A final contribution of this dissertation is a new domain description language, MACPDDL, which has been designed to extend PDDL in order to support the features necessary to describe problems in the HMPCF formalization. MACPDDL extends PDDL to support an agent type hierarchy, agent capabilities, goals factored into tasks, task capability requirements, and explicit action execution. The agent type hierarchy is implicitly

included in PDDL alongside the objects declaration. Partitioning the PDDL objects into passive objects in the environment and active agents capable of changing the environment increases understanding of domain models. Agent action executions in PDDL are typically denoted by including the executing agent in the action's parameter list. Moving the executor declaration from the parameters to a separate field makes explicit which parameters are the agents executing the action and which parameters are the environment objects upon which the agents are executing. Each of the new language features is necessary to describe HMPCF problems and apply the newly developed tools to the problem.

VII.3 Future Work

This research can be extended along several directions.

VII.3.1 Prioritized Coalition Formation then Planning

Another modification to the tools framework is Prioritized Coalition Formation then Planning. Tasks are placed in a priority queue based on their capability requirements, goal state constraints, deadline, and the user defined task priority. While there are still tasks that have not been planned, the highest priority task is popped from the queue. The task deadline and estimated plan makespan are used to estimate the latest possible start time for the task. The agents available at the latest possible start time are used in coalition formation to derive the coalition to be allocated to the task. An agent is considered available at a time t when there are no actions for it to execute at or after t in the current plan. Planning continues as usual in Coalition Formation then Planning, with iterative task planning and plan merging.

VII.3.2 Intelligent Planner Library

No planning algorithm completely dominates all other planning algorithms. For example, ITSAT produced better plans than POPF, but POPF produced plans much faster. Different real world domains and missions will have different constraints. Problems in First Response will require plans to be derived and executed over hours, whereas problems in Zenotravel can take months to plan and hours to execute. This dissertation used ITSAT, POPF, TFD, and COLIN as underlying planners, but many more planners exist that are well suited for different domains. Knowing which planning algorithm to apply given a specific domain or constraints within a domain is difficult. A breadth of planners can be composed into an accessible library and an intelligent middleware level can be developed to decide which planner to apply, similar to the Intelligent Coalition Formation for Humans and Robots framework previously developed to facilitate selecting applicable coalition formation algorithms (Sen and Adams, 2015). The most obvious feature and easiest to consider when selecting a planner is the required domain features. For example, a domain with durative actions will require

a planner capable of reasoning with durative actions. Other features can be user defined, such as required concurrency and estimated plan length. MACPDDL will require extensions to support expressing the user defined features, whereas domain features can be extracted from the domain model.

VII.3.3 Uncertainty and Observability

Uncertainty and observability are two aspects of planning this dissertation did not address. Any reasonably complex domain model of the real world must include nondeterministic actions and partial observability. Acting in the real world is difficult (e.g., noisy sensors, broken actuators). The First Response domain used in the analysis assumed the location of victims was known. Real world responses to natural disasters cannot make such assumptions, as first responders must explore the environment to find victims. New underlying planning algorithms and adapted tools will be required to support uncertainty and partial observability.

VII.3.4 Anytime Planning

Anytime planners continue planning even after producing an initial plan, continuing until the search space is exhausted or computational resource limits are exceeded. Additional derived plans are output only if they are better than the best current plan. Two approaches to transforming the presented tools into anytime planners have been identified. First, order the tools in a prescribed manner and execute each tool, outputting a new plan only if it is better than the best available plan. The second method is more complex. Anytime planning produces successively better plans. Applying such a technique to the CFP, RPCA, and TF tools requires that the iterative initial state used for planning subsequent tasks changes with each newly derived iterative plan. Given that finding a new plan for a task requires replanning all subsequent tasks, the benefits from using the newly derived plan are uncertain.

VII.3.5 Task Formation

The order and composition of the tasks greatly impacts the performance of the CFP, RPCA, and TF tools. Investigating heuristics for ordering task planning and composition has the potential to significantly improve the performance of each tool. The TF tool is especially affected by the task composition and ordering. The TF tool proposed and evaluated in this dissertation was restricted to fusing each existing task with at most one other task. A more general task formation problem can be created by lifting the TF constraints and permitting splitting tasks into multiple tasks, fusing more than two tasks into a single task, and modifying the order in which the tasks are planned. The task formation problem can be modeled as a search problem to find an optimal task composition.

VII.3.6 Extensive Analysis

The presented analysis makes excellent in roads to understanding the implications and usage of the developed tools, but additional analyses are required, particularly analyses that incorporate real robots. However, there are additional near term analyses required as well. For example, the Task Fusion results were mixed, possibly due to using a satisficing planning algorithm, rather than an optimal planning algorithm. Numerous optimal planners exist for classical planning (Kissmann and Edelkamp, 2011; Barley et al., 2014; Torralba et al., 2016); therefore, analysis with domains making classical planning assumptions can be performed to investigate if Task Fusion can be expected to work reasonably with expressive domains. The Rovers domain is an excellent candidate for additional Task Fusion analysis, as a version of the Rovers domain already exists that can be used with the optimal planners.

Additional methods for estimating the utility of fusing a pair of tasks can be investigated. The two utility estimation functions are similar to coalition formation, in that the heuristic functions consider the members of the coalitions and the requirements for the tasks, but not how the coalitions will complete the tasks. Drawing from the results of Relaxed Plan Coalition Augmentation, a relaxed plan for the coalition to execute its assigned task can be derived easily and may provide information that will facilitate selecting the best pairs of tasks to fuse. A relaxed plan can inform which parts of the state space a coalition may affect while completing their task. If the affected parts of the state space overlap, then coalitions are likely to interact during task execution.

It is also necessary to develop a deeper understanding of the effect of coalition formation on the planning process. Factored planning reduces problem complexity by focusing on specific aspects of the problem, one at a time. Another set of analyses can isolate the effect of problem factorization applied by coalition formation. Factored planning is a popular method for addressing plan complexity. Relaxed Plan Coalition Augmentation factors the planning problem along two dimensions, goals into tasks and grand coalitions into coalitions. Relaxed Plan Coalition Augmentation can be run without coalition formation; instead of applying coalition formation to allocate subsets of the grand coalition to each task, a grand coalition can be allocated to every task. The effects of factoring the goal into tasks will be isolated from the effects of factoring the grand coalition into smaller coalitions.

Appendix A

MACPDDL

The most popular language for describing planning problems is PDDL. PDDL 3.1,[1] the most recent version, was created for IPC 2011 (Coles et al., 2012b). One aspect that PDDL does not handle well is agent-based modelling. MAPL (Brenner, 2003) and MA-PDDL (Kovacs, 2012) represent attempts to explicitly incorporate multiple agents, but neither language handles all aspects of multi-agent systems required for this dissertation. PDDL also lacks support for the coalition formation requirements, such as the capability model. The MACPDDL language was developed to explicitly handle agents, heterogeneous coalitions, agent capabilities, tasks, and task requirements. Each extension is detailed in this appendix.

PDDL partitions planning problems into two files, Domain files and Problem files. This problem file partitioning is also used with MACPDDL. The Domain file lists the domain's planning concept requirements, object types, predicates, functions, constraints, and the actions the agents can execute. The concept requirements are the modeling concepts that must be supported by a planning algorithm. Examples of planning concepts include durative actions and numeric fluents. The Problem file lists the problem domain, the objects in the environment, the initial state of the environment, the goal descriptor for the problem, the constraints on the environment, and the metric used to compare plans. The current PDDL specification implicitly handles agents, but it is not enforced in the language grammar.

The *agent type hierarchy* is modeled in the MACPDDL Domain file. The agent type hierarchy is similar to an inheritance hierarchy in object-oriented programming. The hierarchy allows a single definition of actions for agents in the hierarchy. The agent type hierarchy can be translated to a PDDL object type hierarchy to ensure compatibility with planners requiring standard PDDL input. The agent type hierarchy is declared similarly to the object types declaration in PDDL. The agent hierarchy from the First Response domain is presented in Figure A.1. The top level of the hierarchy, on Line 2, states that agents can be humans or robots. The next three lines (Line 3-5) present the robot side of the hierarchy. Robots can be ground robots or quadrotors, with three types of ground robots and one type of quadrotor. The human side of the hierarchy is on Line 6, with human agents being Fire, Police, or EMS agents.

The *action agent executor* specification makes explicit the agent that is executing each action. The powered action from the First Response domain is presented in Figure A.2). The agent executing the action is in Line 2, all agents of type robot or a type under robot in the hierarchy can execute the powered action.

[1]The full BNF grammar for PDDL 3.1 is presented in several unpublished papers by Daniel Kovacs, the most complete of which is hosted on the IPC 2014 website (https://helios.hud.ac.uk/scommv/IPC-14/repository/kovacs-pddl-3.1-2011.pdf).

```
1       (:agents
2         human robot - agent
3         ground quadrotor - robot
4         policebot firebot emsbot - ground
5         pquad - quadrotor
6         fire police ems - human
7       )
8
```

Figure A.1: Example agent types declaration in MACPDDL

```
1       (:durative-action powered
2         :executor (?r - robot)
3         :parameters ()
4         :duration (>= ?duration 0)
5         :condition
6           (and
7             (over all (>= (power ?r) 0))
8           )
9         :effect
10          (and
11            (at start (powered ?r))
12            (decrease (power ?r) #t)
13            (at end (not (powered ?r)))
14          )
15      )
16
```

Figure A.2: Example agent executor for actions in MACPDDL

The MACPDDL format is easily translated to the PDDL format by moving the parameterized agents in the executor line to the parameters in Line 3 and removing the executor line.

Agents must be treated as special objects, because they act in the environment and can change the environment, while objects are passive and do not act in the environment. Declaring the agents in the MACPDDL Problem file is similarly to declaring objects in the PDDL Problem file, but the agents declaration will be separated from the objects declaration, as presented in Figure A.3. MACPDDL can be translated to the current PDDL by moving the contents of the agents declaration to the objects declaration.

Agents have capabilities that must be considered for coalition formation. Declaring the services and

```
1       (:agents
2         pquad1 - pquad
3         pbot1 - policebot
4         emsbot1 emsbot2 emsbot3 emsbot4 emsbot5 emsbot6 - emsbot
5         ems1 ems2 - ems
6         firebot1 - firebot
7         fire1 fire2 - fire
8         police1 - police
9       )
10
```

Figure A.3: Agent declaration in MACPDDL

```
1       (:resources
2         (pbot1 pawn_transport - 1)
3         (emsbot1 victim_transport - 1 pharm_transport - 1)
4         (emsbot2 victim_transport - 1 pharm_transport - 1)
5         (emsbot3 victim_transport - 1 pharm_transport - 1)
6         (ems1 triage - 1 pharm_filter - 1)
7         (ems2 triage - 1 pharm_filter - 1)
8         (firebot1 road_clear_debris - 1)
9         (fire1 road_clear_lines - 1 seal_leak - 1)
10        (fire2 road_clear_lines - 1 seal_leak - 1)
11        (police1 pawn_clear - 1)
12      )
13
```

Figure A.4: Agent capabilities in MACPDDL

```
1       (:task victims
2         :resources (
3           triage - 1
4           victim_transport - 2
5         )
6         :length 5
7         :goal
8           (and
9             (victim_at_hosp victim6 dch)
10            (victim_at_hosp victim7 dch)
11            (victim_at_hosp victim8 dch)
12            (victim_at_hosp victim9 dch)
13            (victim_at_hosp victim10 dch)
14          )
15      )
16
```

Figure A.5: Tasks in MACPDDL

resources offered by each agent occurs in the Problem file. All agents declared in the Problem file must also be declared to offer a service or have a resource. Agent capabilities using the resource model is presented in Figure A.4. Agent capabilities have no meaning in PDDL; thus, the capabilities section is removed when translating to PDDL.

Tasks in coalition formation are modeled as the capabilities required to execute the task and in planning as the goal constraints that must be satisfied for the task to be completed. Both capabilities and goal constraints must be included in the task descriptions of the Problem file. An example task in the First Response domain is presented in Figure A.5. Each task has a name, "victims" in this example, on Line 1. The capability model used by the task is on Line 2, the resources model. The example task requires one triage resource and two victim transport resources, Lines 3 and 4. The length, Line 6, is an estimate of the makespan of the plan for the task, and is used by some coalition formation algorithms. The goal, Lines 7 - 14, are for planning; satisfying the constraints completes the task. The tasks can be translated to PDDL by combining the goal sections from all the tasks in a problem into the single goal statement used by PDDL.

Most planning algorithms use PDDL for parsing input. The coalition formation library, i-CiFHaR, uses XML. All the experiments in this dissertation used MACPDDL to generate the necessary PDDL for planning and XML for coalition formation.

Appendix B

Planning Domains

This appendix presents the MACPDDL version of each domain used in the experiments for this dissertation.

B.1 Rovers

```
(define (domain rover)
  (:requirements :fluents :durative-actions :typing)
  (:agents
    rover - agent
  )
  (:types
    waypoint store camera mode lander objective - object
  )
  (:predicates
    (at_rover ?x - rover ?y - waypoint)
    (at_lander ?x - lander ?y - waypoint)
    (can_traverse ?r - rover ?x - waypoint ?y - waypoint)
    (equipped_for_soil_analysis ?r - rover)
    (equipped_for_rock_analysis ?r - rover)
    (equipped_for_imaging ?r - rover)
    (empty ?s - store)
    (have_rock_analysis ?r - rover ?w - waypoint)
    (have_soil_analysis ?r - rover ?w - waypoint)
    (full ?s - store)
    (camera_available ?c - camera ?r - rover)
    (calibrated ?c - camera ?r - rover)
    (supports ?c - camera ?m - mode)
    (visible ?w - waypoint ?p - waypoint)
    (have_image ?r - rover ?o - objective ?m - mode)
    (communicated_soil_data ?w - waypoint)
    (communicated_rock_data ?w - waypoint)
    (communicated_image_data ?o - objective ?m - mode)
    (at_soil_sample ?w - waypoint)
    (at_rock_sample ?w - waypoint)
    (visible_from ?o - objective ?w - waypoint)
    (store_of ?s - store ?r - rover)
    (calibration_target ?i - camera ?o - objective)
```

```
    (on_board ?i - camera ?r - rover)
    (channel_free ?l - lander)
  )
  (:functions
  )
  (:durative-action navigate
    :executor (?x - rover)
    :parameters (?y - waypoint ?z - waypoint)
    :duration (= ?duration 10)
    :condition
      (and
        (over all (can_traverse ?x ?y ?z))
        (at start (at_rover ?x ?y))
        (over all (visible ?y ?z))
      )
    :effect
      (and
        (at start (not (at_rover ?x ?y)))
        (at end (at_rover ?x ?z))
      )
  )
  (:durative-action sample_soil
    :executor (?x - rover)
    :parameters (?s - store ?p - waypoint)
    :duration (= ?duration 1)
    :condition
      (and
        (over all (at_rover ?x ?p))
        (at start (at_soil_sample ?p))
        (over all (equipped_for_soil_analysis ?x))
        (over all (store_of ?s ?x))
        (at start (empty ?s))
      )
    :effect
      (and
        (at start (not (empty ?s)))
        (at start (not (at_soil_sample ?p)))
        (at end (full ?s))
        (at end (have_soil_analysis ?x ?p))
      )
  )
```

```
(:durative-action sample_rock
  :executor (?x - rover)
  :parameters (?s - store ?p - waypoint)
  :duration (= ?duration 1)
  :condition
    (and
      (over all (at_rover ?x ?p))
      (at start (at_rock_sample ?p))
      (at start (equipped_for_rock_analysis ?x))
      (at start (store_of ?s ?x))
      (at start (empty ?s))
    )
  :effect
    (and
      (at start (not (empty ?s)))
      (at end (full ?s))
      (at end (have_rock_analysis ?x ?p))
      (at start (not (at_rock_sample ?p)))
    )
)
(:durative-action drop
  :executor (?x - rover)
  :parameters (?y - store)
  :duration (= ?duration 0.1)
  :condition
    (and
      (at start (store_of ?y ?x))
      (at start (full ?y))
    )
  :effect
    (and
      (at start (not (full ?y)))
      (at end (empty ?y))
    )
)
(:durative-action calibrate
  :executor (?r - rover)
  :parameters (?i - camera ?t - objective ?w - waypoint)
  :duration (= ?duration 1)
  :condition
    (and
```

```
115         (over all (equipped_for_imaging ?r))
116         (over all (calibration_target ?i ?t))
117         (over all (at_rover ?r ?w))
118         (over all (visible_from ?t ?w))
119         (over all (on_board ?i ?r))
120         (at start (camera_available ?i ?r))
121       )
122     :effect
123       (and
124         (at start (not (camera_available ?i ?r)))
125         (at start (not (calibrated ?i ?r)))
126         (at end (calibrated ?i ?r))
127         (at end (camera_available ?i ?r))
128       )
129   )
130   (:durative-action take_image
131     :executor (?r - rover)
132     :parameters (?p - waypoint ?o - objective ?i - camera ?m - mode)
133     :duration (= ?duration 0.1)
134     :condition
135       (and
136         (over all (calibrated ?i ?r))
137         (over all (on_board ?i ?r))
138         (over all (equipped_for_imaging ?r))
139         (over all (supports ?i ?m))
140         (over all (visible_from ?o ?p))
141         (over all (at_rover ?r ?p))
142         (at start (camera_available ?i ?r))
143       )
144     :effect
145       (and
146         (at start (not (camera_available ?i ?r)))
147         (at end (have_image ?r ?o ?m))
148         (at end (not (calibrated ?i ?r)))
149         (at end (camera_available ?i ?r))
150       )
151   )
152   (:durative-action communicate_soil_data
153     :executor (?r - rover)
154     :parameters (?l - lander ?p - waypoint ?x - waypoint ?y - waypoint)
155     :duration (= ?duration 0.5)
```

```
    :condition
      (and
        (over all (at_rover ?r ?x))
        (at start (at_lander ?l ?y))
        (at start (have_soil_analysis ?r ?p))
        (at start (visible ?x ?y))
        (at start (channel_free ?l))
      )
    :effect
      (and
        (at start (not (channel_free ?l)))
        (at end (channel_free ?l))
        (at end (communicated_soil_data ?p))
      )
  )
  (:durative-action communicate_rock_data
    :executor (?r - rover)
    :parameters (?l - lander ?p - waypoint ?x - waypoint ?y - waypoint)
    :duration (= ?duration 0.5)
    :condition
      (and
        (over all (at_rover ?r ?x))
        (at start (at_lander ?l ?y))
        (at start (have_rock_analysis ?r ?p))
        (at start (visible ?x ?y))
        (at start (channel_free ?l))
      )
    :effect
      (and
        (at start (not (channel_free ?l)))
        (at end (channel_free ?l))
        (at end (communicated_rock_data ?p))
      )
  )
  (:durative-action communicate_image_data
    :executor (?r - rover)
    :parameters (?l - lander ?o - objective ?m - mode ?x - waypoint ?y - waypoint)
    :duration (= ?duration 0.5)
    :condition
      (and
        (over all (at_rover ?r ?x))
```

```
197         (over all (at_lander ?l ?y))
198         (at start (have_image ?r ?o ?m))
199         (at start (visible ?x ?y))
200         (at start (channel_free ?l))
201       )
202     :effect
203       (and
204         (at start (not (channel_free ?l)))
205         (at end (channel_free ?l))
206         (at end (communicated_image_data ?o ?m))
207       )
208   )
209 )
```

B.2 Blocks World

```
(define (domain blocksworld)
  (:requirements :fluents :durative-actions :typing)
  (:agents
    arm - agent
  )
  (:types
    block effector - object
    single_block double_block - block
  )
  (:predicates
    (has_effector ?a - arm ?e - effector)
    (empty ?a - arm)
    (not_moving ?a - arm)

    (requires ?b - block ?e - effector)

    (clear ?b - block)
    (on_block ?b_1 - block ?b_2 - block)
    (on_table ?b - block)
    (holding_single ?a - arm ?b - single_block)
    (holding_double ?a_1 - arm ?a_2 - arm ?b - double_block)
  )
  (:functions
    (block_height ?b - block)
    (arm_height ?a - arm)
    (arm_id ?a - arm)
    (max_on_table)
    (num_on_table)
  )
  (:durative-action move-up
    :executor (?a - arm)
    :parameters ()
    :duration (= ?duration 1)
    :condition
      (and
        (at start (not_moving ?a))
        (over all (empty ?a))
      )
    :effect
      (and
```

```
41          (at start (not (not_moving ?a)))
42          (at end (not_moving ?a))
43          (at end (increase (arm_height ?a) 1))
44        )
45    )
46    (:durative-action move-down
47      :executor (?a - arm)
48      :parameters ()
49      :duration (= ?duration 1)
50      :condition
51        (and
52          (at start (not_moving ?a))
53          (over all (empty ?a))
54          (at start (> (arm_height ?a) 0))
55        )
56      :effect
57        (and
58          (at start (not (not_moving ?a)))
59          (at end (not_moving ?a))
60          (at end (decrease (arm_height ?a) 1))
61        )
62    )
63    (:durative-action move-up-with-single-block
64      :executor (?a - arm)
65      :parameters (?b - single_block)
66      :duration (= ?duration 1)
67      :condition
68        (and
69          (over all (holding_single ?a ?b))
70          (at start (not_moving ?a))
71        )
72      :effect
73        (and
74          (at start (not (not_moving ?a)))
75          (at end (not_moving ?a))
76          (at end (increase (arm_height ?a) 1))
77          (at end (increase (block_height ?b) 1))
78        )
79    )
80    (:durative-action move-down-with-single-block
81      :executor (?a - arm)
```

```
    :parameters (?b - single_block)
    :duration (= ?duration 1)
    :condition
      (and
        (over all (holding_single ?a ?b))
        (at start (not_moving ?a))
        (at start (> (arm_height ?a) 0))
      )
    :effect
      (and
        (at start (not (not_moving ?a)))
        (at end (not_moving ?a))
        (at end (decrease (arm_height ?a) 1))
        (at end (decrease (block_height ?b) 1))
      )
  )
  (:durative-action move-up-with-double-block
    :executor (?a_1 - arm ?a_2 - arm)
    :parameters (?b - double_block)
    :duration (= ?duration 1)
    :condition
      (and
        (over all (holding_double ?a_1 ?a_2 ?b))
        (at start (not_moving ?a_1))
        (at start (not_moving ?a_2))
      )
    :effect
      (and
        (at start (not (not_moving ?a_1)))
        (at start (not (not_moving ?a_2)))
        (at end (not_moving ?a_1))
        (at end (not_moving ?a_2))
        (at end (increase (arm_height ?a_1) 1))
        (at end (increase (arm_height ?a_2) 1))
        (at end (increase (block_height ?b) 1))
      )
  )
  (:durative-action move-down-with-double-block
    :executor (?a_1 - arm ?a_2 - arm)
    :parameters (?b - double_block)
    :duration (= ?duration 1)
```

```
123     :condition
124       (and
125         (over all (holding_double ?a_1 ?a_2 ?b))
126         (at start (not_moving ?a_1))
127         (at start (not_moving ?a_2))
128       )
129     :effect
130       (and
131         (at start (not (not_moving ?a_1)))
132         (at start (not (not_moving ?a_2)))
133         (at end (not_moving ?a_1))
134         (at end (not_moving ?a_2))
135         (at end (decrease (arm_height ?a_1) 1))
136         (at end (decrease (arm_height ?a_2) 1))
137         (at end (decrease (block_height ?b) 1))
138       )
139   )
140   (:durative-action pick-single-block-on-block
141     :executor (?a - arm)
142     :parameters (?b_1 - single_block ?b_2 - block ?e - effector)
143     :duration (= ?duration 1)
144     :condition
145       (and
146         (at start (empty ?a))
147         (at start (= (arm_height ?a) (block_height ?b_1)))
148         (at start (clear ?b_1))
149         (at start (on_block ?b_1 ?b_2))
150         (at start (requires ?b_1 ?e))
151         (at start (has_effector ?a ?e))
152       )
153     :effect
154       (and
155         (at start (not (empty ?a)))
156         (at start (not (clear ?b_1)))
157         (at start (not (on_block ?b_1 ?b_2)))
158         (at end (clear ?b_2))
159         (at end (holding_single ?a ?b_1))
160       )
161   )
162   (:durative-action pick-single-block-on-table
163     :executor (?a - arm)
```

```
164     :parameters (?b - single_block ?e - effector)
165     :duration (= ?duration 1)
166     :condition
167       (and
168         (at start (empty ?a))
169         (at start (= (arm_height ?a) 0))
170         (at start (clear ?b))
171         (at start (on_table ?b))
172         (at start (requires ?b ?e))
173         (at start (has_effector ?a ?e))
174       )
175     :effect
176       (and
177         (at start (not (empty ?a)))
178         (at start (not (clear ?b)))
179         (at end (not (on_table ?b)))
180         (at end (holding_single ?a ?b))
181         (at end (decrease (num_on_table) 1))
182       )
183   )
184   (:durative-action place-single-block-on-block
185     :executor (?a - arm)
186     :parameters (?b_1 - single_block ?b_2 - block)
187     :duration (= ?duration 1)
188     :condition
189       (and
190         (at start (= (arm_height ?a) (+ (block_height ?b_2) 1)))
191         (at start (clear ?b_2))
192         (at start (holding_single ?a ?b_1))
193       )
194     :effect
195       (and
196         (at start (not (clear ?b_2)))
197         (at end (clear ?b_1))
198         (at end (on_block ?b_1 ?b_2))
199         (at start (not (holding_single ?a ?b_1)))
200         (at end (empty ?a))
201       )
202   )
203   (:durative-action place-single-block-on-table
204     :executor (?a - arm)
```

```
    :parameters (?b - single_block)
    :duration (= ?duration 1)
    :condition
      (and
        (at start (= (arm_height ?a) 0))
        (at start (holding_single ?a ?b))
        (at start (< (num_on_table) (max_on_table)))
      )
    :effect
      (and
        (at end (clear ?b))
        (at start (not (holding_single ?a ?b)))
        (at end (empty ?a))
        (at end (on_table ?b))
        (at start (increase (num_on_table) 1))
      )
  )
  (:durative-action pick-double-block-on-block
    :executor (?a_1 - arm ?a_2 - arm)
    :parameters (?b_1 - double_block ?b_2 - block ?e - effector)
    :duration (= ?duration 2)
    :condition
      (and
        (at start (empty ?a_1))
        (at start (empty ?a_2))
        (at start (= (arm_height ?a_1) (block_height ?b_1)))
        (at start (= (arm_height ?a_2) (block_height ?b_1)))
        (at start (clear ?b_1))
        (at start (on_block ?b_1 ?b_2))
        (at start (requires ?b_1 ?e))
        (at start (has_effector ?a_1 ?e))
        (at start (has_effector ?a_2 ?e))
        (at start (> (arm_id ?a_1) (arm_id ?a_2)))
      )
    :effect
      (and
        (at start (not (empty ?a_1)))
        (at start (not (empty ?a_2)))
        (at start (not (clear ?b_1)))
        (at start (not (on_block ?b_1 ?b_2)))
        (at end (clear ?b_2))
```

```
246         (at end (holding_double ?a_1 ?a_2 ?b_1))
247       )
248   )
249   (:durative-action pick-double-block-on-table
250     :executor (?a_1 - arm ?a_2 - arm)
251     :parameters (?b - double_block ?e - effector)
252     :duration (= ?duration 2)
253     :condition
254       (and
255         (at start (empty ?a_1))
256         (at start (empty ?a_2))
257         (at start (= (arm_height ?a_1) 0))
258         (at start (= (arm_height ?a_2) 0))
259         (at start (clear ?b))
260         (at start (on_table ?b))
261         (at start (requires ?b ?e))
262         (at start (has_effector ?a_1 ?e))
263         (at start (has_effector ?a_2 ?e))
264         (at start (> (arm_id ?a_1) (arm_id ?a_2)))
265       )
266     :effect
267       (and
268         (at start (not (empty ?a_1)))
269         (at start (not (empty ?a_2)))
270         (at start (not (clear ?b)))
271         (at end (not (on_table ?b)))
272         (at end (holding_double ?a_1 ?a_2 ?b))
273         (at end (decrease (num_on_table) 1))
274       )
275   )
276   (:durative-action place-double-block-on-block
277     :executor (?a_1 - arm ?a_2 - arm)
278     :parameters (?b_1 - double_block ?b_2 - block)
279     :duration (= ?duration 2)
280     :condition
281       (and
282         (at start (= (arm_height ?a_1) (+ (block_height ?b_2) 1)))
283         (at start (= (arm_height ?a_2) (+ (block_height ?b_2) 1)))
284         (at start (clear ?b_2))
285         (at start (holding_double ?a_1 ?a_2 ?b_1))
286       )
```

```
    :effect
      (and
        (at start (not (clear ?b_2)))
        (at end (clear ?b_1))
        (at start (not (holding_double ?a_1 ?a_2 ?b_1)))
        (at end (on_block ?b_1 ?b_2))
        (at end (empty ?a_1))
        (at end (empty ?a_2))
      )
  )
  (:durative-action place-double-block-on-table
    :executor (?a_1 - arm ?a_2 - arm)
    :parameters (?b - double_block)
    :duration (= ?duration 2)
    :condition
      (and
        (at start (= (arm_height ?a_1) 0))
        (at start (= (arm_height ?a_2) 0))
        (at start (holding_double ?a_1 ?a_2 ?b))
        (at start (< (num_on_table) (max_on_table)))
      )
    :effect
      (and
        (at end (clear ?b))
        (at start (not (holding_double ?a_1 ?a_2 ?b)))
        (at end (on_table ?b))
        (at end (empty ?a_1))
        (at end (empty ?a_2))
        (at start (increase (num_on_table) 1))
      )
  )
)
```

B.3 Zenotravel

```
(define (domain zenotravel)
  (:requirements :fluents :durative-actions :typing :action-costs)
  (:agents
    airplane - agent
    large_plane small_plane - airplane
  )
  (:types
    transportable city - object
    passenger cargo - transportable
    hub spoke - city
  )
  (:predicates
    (transportable_at_city ?t - transportable ?c - city)
    (transportable_on_plane ?t - transportable ?p - airplane)
    (plane_at_city ?a - airplane ?c - city)
    (not_managing_passenger ?a - airplane)
    (not_managing_cargo ?a - airplane)
  )
  (:functions
    (total-cost)
    (fuel ?a - airplane)
    (fuel_rate ?a - airplane)
    (max_fuel ?a - airplane)
    (speed ?a - airplane)
    (distance ?c_1 - city ?c_2 - city)
    (num_passengers ?a - airplane)
    (max_passengers ?a - airplane)
    (num_cargo ?a - airplane)
    (max_cargo ?a - airplane)
  )
  (:durative-action refuel
    :executor (?p - airplane)
    :parameters (?c - city)
    :duration (= ?duration (/ (- (max_fuel ?p) (fuel ?p)) (fuel_rate ?p)))
    :condition
      (and
        (over all (plane_at_city ?p ?c))
      )
    :effect
      (and
```

```
41          (at end (assign (fuel ?p) (max_fuel ?p)))
42        )
43    )
44    (:durative-action fly_small_1
45      :executor (?p - small_plane)
46      :parameters (?c_1 - city ?c_2 - city)
47      :duration (= ?duration (/ (distance ?c_1 ?c_2) (speed ?p)))
48      :condition
49        (and
50          (at start (plane_at_city ?p ?c_1))
51          (at start (>= (fuel ?p) (distance ?c_1 ?c_2)))
52        )
53      :effect
54        (and
55          (at start (increase (total-cost) 1))
56          (at start (not (plane_at_city ?p ?c_1)))
57          (at end (plane_at_city ?p ?c_2))
58          (decrease (fuel ?p) (* #t (speed ?p)))
59        )
60    )
61    (:durative-action fly_small_2
62      :executor (?p - small_plane)
63      :parameters (?c_2 - city ?c_1 - city)
64      :duration (= ?duration (/ (distance ?c_1 ?c_2) (speed ?p)))
65      :condition
66        (and
67          (at start (plane_at_city ?p ?c_2))
68          (at start (>= (fuel ?p) (distance ?c_1 ?c_2)))
69        )
70      :effect
71        (and
72          (at start (increase (total-cost) 1))
73          (at start (not (plane_at_city ?p ?c_2)))
74          (at end (plane_at_city ?p ?c_1))
75          (decrease (fuel ?p) (* #t (speed ?p)))
76        )
77    )
78    (:durative-action fly_large_1
79      :executor (?p - large_plane)
80      :parameters (?c_1 - hub ?c_2 - hub)
81      :duration (= ?duration (/ (distance ?c_1 ?c_2) (speed ?p)))
```

```
    :condition
      (and
        (at start (plane_at_city ?p ?c_1))
        (at start (>= (fuel ?p) (distance ?c_1 ?c_2)))
      )
    :effect
      (and
        (at start (increase (total-cost) 1))
        (at start (not (plane_at_city ?p ?c_1)))
        (at end (plane_at_city ?p ?c_2))
        (decrease (fuel ?p) (* #t (speed ?p)))
      )
  )
  (:durative-action fly_large_2
    :executor (?p - large_plane)
    :parameters (?c_2 - hub ?c_1 - hub)
    :duration (= ?duration (/ (distance ?c_1 ?c_2) (speed ?p)))
    :condition
      (and
        (at start (plane_at_city ?p ?c_2))
        (at start (>= (fuel ?p) (distance ?c_1 ?c_2)))
      )
    :effect
      (and
        (at start (increase (total-cost) 1))
        (at start (not (plane_at_city ?p ?c_2)))
        (at end (plane_at_city ?p ?c_1))
        (decrease (fuel ?p) (* #t (speed ?p)))
      )
  )
  (:durative-action load_passenger
    :executor (?a - airplane)
    :parameters (?c - city ?t - passenger)
    :duration (= ?duration 1)
    :condition
      (and
        (at start (transportable_at_city ?t ?c))
        (at start (< (num_passengers ?a) (max_passengers ?a)))
        (at start (not_managing_passenger ?a))
        (over all (plane_at_city ?a ?c))
      )
```

```
    :effect
      (and
        (at start (not (not_managing_passenger ?a)))
        (at start (not (transportable_at_city ?t ?c)))
        (at start (increase (num_passengers ?a) 1))
        (at end (transportable_on_plane ?t ?a))
        (at end (not_managing_passenger ?a))
      )
  )
  (:durative-action unload_passenger
    :executor (?a - airplane)
    :parameters (?c - city ?t - passenger)
    :duration (= ?duration 1)
    :condition
      (and
        (at start (transportable_on_plane ?t ?a))
        (at start (not_managing_passenger ?a))
        (over all (plane_at_city ?a ?c))
      )
    :effect
      (and
        (at start (not (not_managing_passenger ?a)))
        (at start (not (transportable_on_plane ?t ?a)))
        (at end (decrease (num_passengers ?a) 1))
        (at end (transportable_at_city ?t ?c))
        (at end (not_managing_passenger ?a))
      )
  )
  (:durative-action load_cargo
    :executor (?a - airplane)
    :parameters (?c - city ?t - cargo)
    :duration (= ?duration 1)
    :condition
      (and
        (at start (transportable_at_city ?t ?c))
        (at start (< (num_cargo ?a) (max_cargo ?a)))
        (at start (not_managing_cargo ?a))
        (over all (plane_at_city ?a ?c))
      )
    :effect
      (and
```

```
        (at start (not (not_managing_cargo ?a)))
        (at start (not (transportable_at_city ?t ?c)))
        (at start (increase (num_cargo ?a) 1))
        (at end (transportable_on_plane ?t ?a))
        (at end (not_managing_cargo ?a))
      )
  )
  (:durative-action unload_cargo
    :executor (?a - airplane)
    :parameters (?c - city ?t - cargo)
    :duration (= ?duration 1)
    :condition
      (and
        (at start (transportable_on_plane ?t ?a))
        (at start (not_managing_cargo ?a))
        (over all (plane_at_city ?a ?c))
      )
    :effect
      (and
        (at start (not (not_managing_cargo ?a)))
        (at start (not (transportable_on_plane ?t ?a)))
        (at end (decrease (num_cargo ?a) 1))
        (at end (transportable_at_city ?t ?c))
        (at end (not_managing_cargo ?a))
      )
  )
)
```

B.4 First Response

```
(define (domain first_response)
  (:requirements :fluents :durative-actions :typing :duration-inequalities)
  (:agents
    human robot - agent
    ground quadrotor - robot
    policebot firebot emsbot - ground
    pquad - quadrotor
    searching ems - human
    fire police - searching
  )
  (:types
    locatable waypoint - object
    base hospital hazardous victim batts - locatable
    pawn pharm - hazardous
  )
  (:predicates
    (not_moving ?a - agent)
    (victim_found ?v - victim)
    (victim_at_hosp ?v - victim ?h - hospital)
    (loc_at ?l - locatable ?w - waypoint)
    (agent_at ?a - agent ?w - waypoint)

    (carrying ?e - emsbot ?v - victim)
    (empty ?e - ground)

    (hazards_stored ?h - hazardous ?l - locatable)
    (carrying_hazardous ?g - ground ?h - hazardous)

    (not_blocked ?w_1 - waypoint ?w_2 - waypoint)
    (debris ?w_1 - waypoint ?w_2 - waypoint)
    (lines ?w_1 - waypoint ?w_2 - waypoint)

    (powered ?r - robot)

    (gas_leak ?w - waypoint)
    (gas_cleared ?w - waypoint)

    (edge ?w_1 - waypoint ?w_2 - waypoint)
    (victim_triaged ?v - victim)
  )
```

```
(:functions
  (power_max ?r - robot)
  (power ?r - robot)
  (victim_injury ?v - victim)
  (victim_state ?v - victim)
)
(:durative-action change-battery
  :executor (?r - robot)
  :parameters (?b - batts ?w - waypoint)
  :duration (= ?duration 2)
  :condition
    (and
      (over all (agent_at ?r ?w))
      (over all (loc_at ?b ?w))
    )
  :effect
    (and
      (at start (assign (power ?r) 0))
      (at end (assign (power ?r) (power_max ?r)))
    )
)
(:durative-action activate_robot
  :executor (?r - robot)
  :parameters ()
  :duration (>= ?duration 0)
  :condition
    (and
      (over all (>= (power ?r) 0))
    )
  :effect
    (and
      (at start (powered ?r))
      (decrease (power ?r) #t)
      (at end (not (powered ?r)))
    )
)
(:durative-action quad-move-1
  :executor (?q - quadrotor)
  :parameters (?w_1 - waypoint ?w_2 - waypoint)
  :duration (= ?duration 1.0)
  :condition
```

```
      (and
        (over all (powered ?q))
        (at start (not_moving ?q))
        (over all (edge ?w_1 ?w_2))
        (at start (agent_at ?q ?w_1))
      )
    :effect
      (and
        (at start (not (not_moving ?q)))
        (at start (not (agent_at ?q ?w_1)))
        (at end (agent_at ?q ?w_2))
        (at end (not_moving ?q))
      )
  )
  (:durative-action quad-move-2
    :executor (?q - quadrotor)
    :parameters (?w_1 - waypoint ?w_2 - waypoint)
    :duration (= ?duration 1.0)
    :condition
      (and
        (over all (powered ?q))
        (at start (not_moving ?q))
        (over all (edge ?w_2 ?w_1))
        (at start (agent_at ?q ?w_1))
      )
    :effect
      (and
        (at start (not (not_moving ?q)))
        (at start (not (agent_at ?q ?w_1)))
        (at end (agent_at ?q ?w_2))
        (at end (not_moving ?q))
      )
  )
  (:durative-action ground-move-1
    :executor (?g - ground)
    :parameters (?w_1 - waypoint ?w_2 - waypoint)
    :duration (= ?duration 1.0)
    :condition
      (and
        (over all (powered ?g))
        (over all (edge ?w_1 ?w_2))
```

```
        (over all (not_blocked ?w_1 ?w_2))
        (at start (not_moving ?g))
        (at start (agent_at ?g ?w_1))
      )
    :effect
      (and
        (at start (not (not_moving ?g)))
        (at start (not (agent_at ?g ?w_1)))
        (at end (agent_at ?g ?w_2))
        (at end (not_moving ?g))
      )
  )
  (:durative-action ground-move-2
    :executor (?g - ground)
    :parameters (?w_1 - waypoint ?w_2 - waypoint)
    :duration (= ?duration 1.0)
    :condition
      (and
        (over all (powered ?g))
        (over all (edge ?w_2 ?w_1))
        (over all (not_blocked ?w_2 ?w_1))
        (at start (not_moving ?g))
        (at start (agent_at ?g ?w_1))
      )
    :effect
      (and
        (at start (not (not_moving ?g)))
        (at start (not (agent_at ?g ?w_1)))
        (at end (agent_at ?g ?w_2))
        (at end (not_moving ?g))
      )
  )
  (:durative-action human-move-1
    :executor (?h - human)
    :parameters (?w_1 - waypoint ?w_2 - waypoint)
    :duration (= ?duration 1.0)
    :condition
      (and
        (over all (edge ?w_1 ?w_2))
        (over all (not_blocked ?w_1 ?w_2))
        (at start (not_moving ?h))
```

```
164         (at start (agent_at ?h ?w_1))
165       )
166     :effect
167       (and
168         (at start (not (not_moving ?h)))
169         (at start (not (agent_at ?h ?w_1)))
170         (at end (agent_at ?h ?w_2))
171         (at end (not_moving ?h))
172       )
173   )
174   (:durative-action human-move-2
175     :executor (?h - human)
176     :parameters (?w_1 - waypoint ?w_2 - waypoint)
177     :duration (= ?duration 1.0)
178     :condition
179       (and
180         (over all (edge ?w_2 ?w_1))
181         (over all (not_blocked ?w_2 ?w_1))
182         (at start (not_moving ?h))
183         (at start (agent_at ?h ?w_1))
184       )
185     :effect
186       (and
187         (at start (not (not_moving ?h)))
188         (at start (not (agent_at ?h ?w_1)))
189         (at end (agent_at ?h ?w_2))
190         (at end (not_moving ?h))
191       )
192   )
193   (:durative-action guide-victim-1
194     :executor (?q - pquad)
195     :parameters (?v - victim ?w_1 - waypoint ?w_2 - waypoint)
196     :duration (= ?duration 2.0)
197     :condition
198       (and
199         (over all (powered ?q))
200         (over all (edge ?w_1 ?w_2))
201         (over all (not_blocked ?w_1 ?w_2))
202         (at start (not_moving ?q))
203         (at start (loc_at ?v ?w_1))
204         (at start (agent_at ?q ?w_1))
```

```
205         (over all (>= (victim_state ?v) 4))
206       )
207     :effect
208       (and
209         (at start (not (not_moving ?q)))
210         (at start (not (agent_at ?q ?w_1)))
211         (at start (not (loc_at ?v ?w_1)))
212         (at end (agent_at ?q ?w_2))
213         (at end (loc_at ?v ?w_2))
214         (at end (not_moving ?q))
215       )
216   )
217   (:durative-action guide-victim-2
218     :executor (?q - pquad)
219     :parameters (?v - victim ?w_1 - waypoint ?w_2 - waypoint)
220     :duration (= ?duration 2.0)
221     :condition
222       (and
223         (over all (powered ?q))
224         (over all (edge ?w_2 ?w_1))
225         (over all (not_blocked ?w_2 ?w_1))
226         (at start (not_moving ?q))
227         (at start (loc_at ?v ?w_1))
228         (at start (agent_at ?q ?w_1))
229         (over all (>= (victim_state ?v) 4))
230       )
231     :effect
232       (and
233         (at start (not (not_moving ?q)))
234         (at start (not (agent_at ?q ?w_1)))
235         (at start (not (loc_at ?v ?w_1)))
236         (at end (agent_at ?q ?w_2))
237         (at end (loc_at ?v ?w_2))
238         (at end (not_moving ?q))
239       )
240   )
241   (:durative-action guide-victim-to-hospital
242     :executor (?q - pquad)
243     :parameters (?v - victim ?h - hospital ?w - waypoint)
244     :duration (= ?duration 0.5)
245     :condition
```

```
246       (and
247         (at start (loc_at ?v ?w))
248         (over all (powered ?q))
249         (over all (loc_at ?h ?w))
250         (over all (agent_at ?q ?w))
251         (over all (>= (victim_state ?v) 4))
252       )
253     :effect
254       (and
255         (at start (not (loc_at ?v ?w)))
256         (at end (victim_at_hosp ?v ?h))
257       )
258   )
259   (:durative-action bulldoze-edge-1
260     :executor (?f - firebot)
261     :parameters (?w_1 - waypoint ?w_2 - waypoint)
262     :duration (= ?duration 2.0)
263     :condition
264       (and
265         (over all (powered ?f))
266         (over all (agent_at ?f ?w_1))
267         (over all (edge ?w_1 ?w_2))
268         (at start (debris ?w_1 ?w_2))
269       )
270     :effect
271       (and
272         (at start (not (debris ?w_1 ?w_2)))
273         (at end (not_blocked ?w_1 ?w_2))
274       )
275   )
276   (:durative-action bulldoze-edge-2
277     :executor (?f - firebot)
278     :parameters (?w_1 - waypoint ?w_2 - waypoint)
279     :duration (= ?duration 2.0)
280     :condition
281       (and
282         (over all (powered ?f))
283         (over all (agent_at ?f ?w_1))
284         (over all (edge ?w_2 ?w_1))
285         (at start (debris ?w_2 ?w_1))
286       )
```

```
    :effect
      (and
        (at start (not (debris ?w_2 ?w_1)))
        (at end (not_blocked ?w_2 ?w_1))
      )
  )
  (:durative-action clear-lines-1
    :executor (?f - firebot)
    :parameters (?w_1 - waypoint ?w_2 - waypoint)
    :duration (= ?duration 2.0)
    :condition
      (and
        (over all (powered ?f))
        (over all (agent_at ?f ?w_1))
        (over all (edge ?w_1 ?w_2))
        (at start (lines ?w_1 ?w_2))
      )
    :effect
      (and
        (at start (not (lines ?w_1 ?w_2)))
        (at end (not_blocked ?w_1 ?w_2))
      )
  )
  (:durative-action clear-lines-2
    :executor (?f - firebot)
    :parameters (?w_1 - waypoint ?w_2 - waypoint)
    :duration (= ?duration 2.0)
    :condition
      (and
        (over all (powered ?f))
        (over all (agent_at ?f ?w_1))
        (over all (edge ?w_2 ?w_1))
        (at start (lines ?w_2 ?w_1))
      )
    :effect
      (and
        (at start (not (lines ?w_2 ?w_1)))
        (at end (not_blocked ?w_2 ?w_1))
      )
  )
```

```
(:durative-action seal-leak
  :executor (?f - fire)
  :parameters (?w - waypoint)
  :duration (= ?duration 3.0)
  :condition
    (and
      (over all (agent_at ?f ?w))
      (at start (gas_leak ?w))
    )
  :effect
    (and
      (at start (not (gas_leak ?w)))
      (at end (gas_cleared ?w))
    )
)
(:durative-action triage-bad
  :executor (?e - ems)
  :parameters (?v - victim ?w - waypoint)
  :duration (= ?duration 2.0)
  :condition
    (and
      (at start (= (victim_state ?v) 0))
      (over all (agent_at ?e ?w))
      (over all (loc_at ?v ?w))
      (over all (<= (victim_injury ?v) 4))
    )
  :effect
    (and
      (at end (assign (victim_state ?v) (victim_injury ?v)))
    )
)
(:durative-action triage-good
  :executor (?e - ems)
  :parameters (?v - victim ?w - waypoint ?h - hospital)
  :duration (= ?duration 1.0)
  :condition
    (and
      (over all (agent_at ?e ?w))
      (over all (loc_at ?v ?w))
      (over all (= (victim_state ?v) 0))
      (over all (= (victim_injury ?v) 5))
```

```
      )
    :effect
      (and
        (at end (victim_at_hosp ?v ?h))
      )
  )
  (:durative-action load-victim
    :executor (?e - ems ?eb - emsbot)
    :parameters (?v - victim ?w - waypoint)
    :duration (= ?duration 2.0)
    :condition
      (and
        (at start (loc_at ?v ?w))
        (at start (empty ?eb))
        (over all (powered ?eb))
        (over all (agent_at ?e ?w))
        (over all (agent_at ?eb ?w))
        (over all (> (victim_state ?v) 0))
      )
    :effect
      (and
        (at start (not (loc_at ?v ?w)))
        (at start (not (empty ?eb)))
        (at end (carrying ?eb ?v))
      )
  )
  (:durative-action unload-victim-at-hospital
    :executor (?eb - emsbot)
    :parameters (?v - victim ?w - waypoint ?h - hospital)
    :duration (= ?duration 2.0)
    :condition
      (and
        (at start (carrying ?eb ?v))
        (over all (powered ?eb))
        (over all (agent_at ?eb ?w))
        (over all (loc_at ?h ?w))
      )
    :effect
      (and
        (at start (not (carrying ?eb ?v)))
        (at end (victim_at_hosp ?v ?h))
```

```
        (at end (empty ?eb))
      )
  )
  (:durative-action clear-pawn
    :executor (?p - police ?pb - policebot)
    :parameters (?pa - pawn ?w - waypoint)
    :duration (= ?duration 5.0)
    :condition
      (and
        (at start (empty ?pb))
        (over all (powered ?pb))
        (over all (agent_at ?p ?w))
        (over all (agent_at ?pb ?w))
        (over all (loc_at ?pa ?w))
      )
    :effect
      (and
        (at start (not (empty ?pb)))
        (at end (carrying_hazardous ?pb ?pa))
      )
  )
  (:durative-action drop-pawn-hazards
    :executor (?pb - policebot)
    :parameters (?p - pawn ?b - base ?w - waypoint)
    :duration (= ?duration 1.0)
    :condition
      (and
        (at start (carrying_hazardous ?pb ?p))
        (over all (powered ?pb))
        (over all (loc_at ?b ?w))
        (over all (agent_at ?pb ?w))
      )
    :effect
      (and
        (at start (not (carrying_hazardous ?pb ?p)))
        (at end (empty ?pb))
        (at end (hazards_stored ?p ?b))
      )
  )
  (:durative-action clear-pharm
    :executor (?e - ems ?eb - emsbot)
```

```
    :parameters (?p - pharm ?w - waypoint)
    :duration (= ?duration 5.0)
    :condition
      (and
        (at start (empty ?eb))
        (over all (powered ?eb))
        (over all (agent_at ?e ?w))
        (over all (agent_at ?eb ?w))
        (over all (loc_at ?p ?w))
      )
    :effect
      (and
        (at start (not (empty ?eb)))
        (at end (carrying_hazardous ?eb ?p))
      )
  )
  (:durative-action drop-pharm-hazards
    :executor (?eb - emsbot)
    :parameters (?p - pharm ?h - hospital ?w - waypoint)
    :duration (= ?duration 1.0)
    :condition
      (and
        (at start (carrying_hazardous ?eb ?p))
        (over all (powered ?eb))
        (over all (loc_at ?h ?w))
        (over all (agent_at ?eb ?w))
      )
    :effect
      (and
        (at start (not (carrying_hazardous ?eb ?p)))
        (at end (empty ?eb))
        (at end (hazards_stored ?p ?h))
      )
  )
)
```

Appendix C

Detailed Rovers Results

This appendix presents the detailed results for the Rovers problems. Each table has a heatmap overlay, ranging from better results in green (lower values for all metrics) to worse results in red (higher values for all metrics). The range for each heatmap is identical for the same metrics (e.g., the makespan results for the problems solved using PA[ITSAT] have the same heatmap range as the makespan results for problems solved using RPCA[POPF]) to facilitate comparison across tool configurations, with specific range values presented in Table C.1. The overlay for time results ranges from 0 seconds (green) to 3600 seconds (red), and the overlay for the memory results ranges from 0 GB (green) to 48 GB (red). The range for action executions is from 400 (green) to 1200 (red) and the range for makespan is from 800 (green) to 4600 (red).

Metric	**Green**	**Red**
Makespan	800	4600
Action Executions	400	1200
Time	0 seconds	3600 seconds
Memory	0 GB	48 GB

Table C.1: Heatmap ranges for each metric in the Rovers domain

C.1 Detailed Results for PA[ITSAT] Applied to the Rovers Problems

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	1497	1280	2661	1357	1369	1867	1580	1850	1549	1256	1626.6
2	2174	1297	1751	1178	1096	1632	1528	1820	1062	1096	1463.4
3	1272	922	1244	919	1090	1320	1027	1243	905	876	1081.8
4	1894	1274	1696	1391	1237	1537	1391	1649	1162	1191	1442.2
5	1676	1340	1593	1253	1183	1479	1318	1640	1322	1012	1381.6
6	2483	1452	2294	1355	1377	1817	1742	1939	1257	1228	1694.4
7	1463	1364	2871	1214	1173	1735	1628	1860	1482	1170	1596.0
8	1969	1351	2139	1271	1404	1908	1626	1744	1349	1423	1618.4
9	2656	1542	1598	1678	1750	1779	1666	2219	1411	1236	1753.5
10	2512	1771	1942	1455	1471	2042	1727	1982	1711	1333	1794.6
Average	1959.6	1359.3	1978.9	1307.1	1315.0	1711.6	1523.3	1794.6	1321.0	1182.1	1545.3

Table C.2: Temporal makespan of derived plans using PA[ITSAT] for Rovers problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	708	659	1189	670	697	863	764	802	784	657	779.3
2	898	661	807	623	588	764	665	808	600	613	702.7
3	624	483	554	502	512	659	561	576	492	485	544.8
4	802	655	831	601	645	734	696	685	596	633	687.8
5	763	672	730	647	634	757	640	800	678	584	690.5
6	1024	725	931	659	733	869	849	796	605	672	786.3
7	704	624	1038	605	611	766	730	731	661	570	704.0
8	917	677	867	688	697	915	785	835	694	707	778.2
9	1004	782	728	817	832	821	817	846	702	697	804.6
10	1115	876	902	787	810	965	838	946	877	747	886.3
Average	855.9	681.4	857.7	659.9	675.9	811.3	734.5	782.5	668.9	636.5	736.5

Table C.3: Action execution steps in derived plans using PA[ITSAT] for Rovers problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	1717	1738	3371	1691	2047	3271	2168	2432	2355	2249	2303.7
2	2939	2015	1973	1888	1882	3240	2281	2352	1765	2215	2255.0
3	1533	1606	1594	1641	1805	3082	1797	1772	1532	2017	1837.9
4	1796	1564	1784	1558	1733	2614	1873	2003	1434	1968	1832.7
5	1866	1661	1429	1718	1736	2536	1536	2054	1805	1798	1813.9
6	3278	2259	3102	2065	2387	3723	2766	2873	1960	2672	2708.5
7	1437	1762	3634	1702	1897	3518	2546	2453	2020	2287	2325.7
8	2241	1916	2609	1736	2030	3420	2132	2364	1916	2423	2278.6
9	3254	2546	2024	2681	2880	3846	2669	3035	2371	2701	2800.6
10	2803	2294	1929	2019	2165	3590	2316	2609	2556	2330	2461.0
Average	2286.3	1936.0	2344.8	1870.1	2056.2	3283.9	2208.3	2394.6	1971.4	2265.9	2261.8

Table C.4: Computation time to derive plans using PA[ITSAT] for Rovers problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	14.13	14.54	22.18	14.49	16.62	23.22	15.20	14.83	18.12	19.07	17.24
2	17.66	15.64	15.88	15.23	14.72	19.67	15.19	14.19	14.35	18.43	16.10
3	13.21	13.25	14.19	13.54	14.15	19.59	13.62	12.39	13.93	16.74	14.46
4	14.58	14.68	15.59	14.56	15.42	19.00	14.66	13.79	14.79	18.55	15.56
5	14.41	14.16	15.12	14.16	14.55	19.39	12.71	12.89	15.72	17.26	15.04
6	20.98	17.58	21.47	17.27	18.94	25.69	17.03	17.18	17.81	21.56	19.55
7	14.02	14.98	23.98	14.93	15.97	23.80	15.90	15.48	17.18	19.55	17.58
8	17.38	17.15	19.63	16.07	18.35	25.92	16.62	15.99	18.04	21.79	18.69
9	20.98	18.43	16.53	19.05	20.01	25.47	16.60	18.99	18.33	21.07	19.55
10	19.45	18.27	17.49	18.27	19.22	25.43	16.56	16.91	19.59	22.66	19.39
Average	16.68	15.87	18.21	15.76	16.80	22.72	15.41	15.26	16.79	19.67	17.32

Table C.5: Memory required to derive plans using PA[ITSAT] for Rovers problems.

C.2 Detailed Results for CFP[POPF] Applied to the Rovers Problems

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	3983	3357	3062	3211	3372	3963	3812	2934	3059	EHC	3417.0
2	2916	3135	NE	3151	2965	NE	2579	3074	NE	NE	2970.0
3	1756	2232	1881	1588	2012	1821	2084	1946	NE	2621	1993.4
4	2362	NE	NE	2771	2451	NE	2432	2442	2429	NE	2481.2
5	2150	2153	EHC	2241	NE	NE	2269	1965	NE	NE	2155.6
6	3494	3792	NE	NE	2961	NE	NE	2856	2975	3311	3231.5
7	3103	3724	3573	3078	EHC	3392	3695	2615	2820	2907	3211.9
8	EHC	3201	4074	3281	2940	2536	3097	3621	3333	3335	3268.7
9	EHC	2791	3254	NE	EHC	NE	NE	EHC	NE	EHC	3022.5
10	EHC	3517	2962	3352	2961	2733	3197	2944	NE	3583	3156.1
Average	2823.4	3100.2	3134.3	2834.1	2808.9	2889.0	2895.6	2710.8	2923.2	3151.4	2915.8

Table C.6: Temporal makespan of derived plans using CFP[POPF] for Rovers problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	842	833	775	764	775	826	862	730	777	EHC	798.2
2	697	762	NE	766	710	NE	736	773	NE	NE	740.7
3	543	591	518	553	625	562	631	620	NE	651	588.2
4	673	NE	NE	749	672	NE	641	675	719	NE	688.2
5	656	670	EHC	661	NE	NE	666	643	NE	NE	659.2
6	784	855	NE	NE	723	NE	NE	713	751	725	758.5
7	625	675	687	657	EHC	678	707	625	657	646	661.9
8	EHC	820	919	819	820	795	880	882	867	890	854.7
9	EHC	814	778	NE	EHC	NE	NE	EHC	NE	EHC	796.0
10	EHC	933	852	868	880	805	921	870	NE	919	881.0
Average	688.6	772.6	754.8	729.6	743.6	733.2	755.5	725.7	754.2	766.2	741.8

Table C.7: Action execution steps in derived plans using CFP[POPF] for Rovers problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	267.7	298.1	339.1	262.9	645.1	341.8	442.8	252.8	1107.5	EHC	439.8
2	262.1	401.6	NE	241.1	237.4	NE	282.9	330.2	NE	NE	292.6
3	84.9	325.2	85.4	157.2	216.2	147.5	276.3	173.7	NE	357.7	202.7
4	127.5	NE	NE	200.4	251.4	NE	189.7	297.1	297.8	NE	227.3
5	104.8	175	EHC	172.4	NE	NE	245.4	144.5	NE	NE	168.4
6	231.1	331.6	NE	NE	184.2	NE	NE	217.7	268.5	263	249.4
7	113.8	162.5	216.6	162	EHC	286.3	341.6	124.7	144	209.1	195.6
8	EHC	342	406.3	227.8	321.8	423.8	517.5	431.4	475.8	332.7	386.6
9	EHC	318.8	219.7	NE	EHC	NE	NE	EHC	NE	EHC	269.3
10	EHC	466.2	378.4	422.7	440.9	434.4	478.5	265.2	NE	643.5	441.2
Average	170.3	313.4	274.3	230.8	328.1	326.8	346.8	248.6	458.7	361.2	297.8

Table C.8: Computation time to derive plans using CFP[POPF] for Rovers problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	12.4	17.33	3.44	7.62	27.51	6.99	22.24	8.35	24.79	EHC	14.52
2	12.8	14.33	NE	5.13	10.36	NE	4.11	10.38	NE	NE	9.52
3	1.19	11.69	1.33	0.83	1.78	0.93	3.49	2.97	NE	5.68	3.32
4	3.3	NE	NE	7.13	2.77	NE	4.41	2.77	9.33	NE	4.95
5	1.49	2.15	EHC	1.74	NE	NE	3.75	1.09	NE	NE	2.04
6	7.41	10.97	NE	NE	5.17	NE	NE	4.38	4.99	5.11	6.34
7	6.67	9.73	13.25	6.88	EHC	7.01	21.89	3.51	2.82	4.32	8.45
8	EHC	8.24	23.95	6.57	15.05	3.14	9.28	14.85	14.33	7.65	11.45
9	EHC	6.71	6.94	NE	EHC	NE	NE	EHC	NE	EHC	6.83
10	EHC	10.8	5.13	6.18	4.54	4.27	8.34	4.51	NE	17.04	7.60
Average	6.47	10.22	9.01	5.26	9.60	4.47	9.69	5.87	11.25	7.96	7.96

Table C.9: Memory required to derive plans using CFP[POPF] for Rovers problems.

C.3 Detailed Results for CFP[ITSAT] Applied to the Rovers Problems

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	2251	2282	2267	2266	2292	2231	2308	2291	2298	2166	2265.2
2	2055	2090	NE	2157	1905	NE	2004	1891	NE	NE	2017.0
3	1268	1112	1227	1147	1259	1202	1219	1179	NE	1194	1200.8
4	1728	NE	NE	1672	1714	NE	1526	1743	1699	NE	1680.3
5	1431	1438	1531	1474	NE	NE	1444	1382	NE	NE	1450.0
6	2108	2240	NE	NE	2095	NE	NE	1994	2132	2042	2101.8
7	2083	2274	2153	2192	2135	1958	2033	2182	2140	1853	2100.3
8	1958	1914	1798	2074	1995	2070	1923	1882	2024	1978	1961.6
9	2011	1988	2032	NE	1954	NE	NE	1942	NE	2307	2039.0
10	1961	1998	1750	1723	2050	1788	1805	1970	NE	2085	1903.3
Average	1885.4	1926.2	1822.6	1838.1	1933.2	1849.8	1782.8	1845.6	2058.6	1946.4	1888.9

Table C.10: Temporal makespan of derived plans using CFP[ITSAT] for Rovers problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	677	637	668	653	639	643	659	642	641	623	648.2
2	618	600	NE	583	560	NE	576	568	NE	NE	584.2
3	592	482	539	494	498	511	504	513	NE	506	515.4
4	651	NE	NE	610	585	NE	572	600	608	NE	604.3
5	659	606	643	618	NE	NE	615	622	NE	NE	627.2
6	673	642	NE	NE	649	NE	NE	624	637	635	643.3
7	562	554	566	552	539	526	526	537	553	530	544.5
8	722	635	685	654	674	674	640	638	659	659	664.0
9	732	630	711	NE	661	NE	NE	675	NE	714	687.2
10	825	739	741	724	752	717	719	733	NE	758	745.3
Average	671.1	613.9	650.4	611.0	617.4	614.2	601.4	615.2	619.6	632.1	625.5

Table C.11: Action execution steps in derived plans using CFP[ITSAT] for Rovers problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	1910	1801	1866	1838	1771	1886	1898	1809	1821	1755	1835.5
2	1606	1415	NE	1399	1313	NE	1376	1279	NE	NE	1397.9
3	1391	1165	1257	1168	1153	1243	1086	1148	NE	1228	1204.4
4	1258	NE	NE	1378	1478	NE	1145	1392	1326	NE	1329.5
5	1387	1471	1455	1443	NE	NE	1301	1378	NE	NE	1405.6
6	1933	1854	NE	NE	1970	NE	NE	1703	1925	1826	1868.7
7	1332	1409	1470	1440	1384	1478	1346	1395	1464	1330	1404.6
8	2028	1862	2077	1973	1919	2096	1854	1851	1959	1919	1953.8
9	1831	1633	1911	NE	1845	NE	NE	1742	NE	1892	1809.1
10	2354	2251	2102	1938	2304	2114	1941	1928	NE	2408	2148.7
Average	1703.1	1651.2	1734.1	1572.1	1681.7	1763.3	1493.3	1562.4	1699.0	1765.4	1653.6

Table C.12: Computation time to derive plans using CFP[ITSAT] for Rovers problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	1.54	1.18	1.53	1.43	1.66	2.27	1.58	1.69	2.11	1.99	1.70
2	1.56	1.88	NE	1.32	1.52	NE	1.91	1.17	NE	NE	1.56
3	0.99	1.21	1.19	1.29	1.18	1.75	1.20	1.07	NE	1.33	1.25
4	1.10	NE	NE	1.19	1.27	NE	1.51	1.42	1.40	NE	1.32
5	1.10	1.39	1.27	1.25	NE	NE	1.57	1.09	NE	NE	1.28
6	1.77	2.21	NE	NE	1.79	NE	NE	1.69	2.13	2.05	1.94
7	1.02	1.63	2.14	2.12	2.38	3.12	2.18	2.27	2.77	2.54	2.22
8	1.48	1.54	1.31	1.40	1.70	2.96	1.96	1.58	1.99	1.99	1.79
9	1.72	1.92	1.60	NE	1.37	NE	NE	1.79	NE	2.56	1.83
10	1.80	1.94	1.41	1.35	1.43	2.50	1.84	1.50	NE	2.03	1.76
Average	1.41	1.66	1.49	1.42	1.59	2.52	1.72	1.53	2.08	2.07	1.69

Table C.13: Memory required to derive plans using CFP[ITSAT] for Rovers problems.

C.4 Detailed Results for RPCA[POPF] Applied to the Rovers Problems

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	3983	3357	3062	3211	3372	3963	3812	2934	3059	EHC	3417.0
2	2916	3135	2841	3151	2965	3137	2579	3074	3085	3073	2995.6
3	1756	2232	1881	1588	2012	1821	2084	1946	3092	2621	2103.3
4	2362	2522	2608	2771	2451	2770	2432	2442	2429	4369	2715.6
5	2150	2153	EHC	2241	1929	EHC	2269	1965	2395	2175	2159.6
6	3494	3792	3086	TIME	2961	4554	2419	2856	2975	3311	3272.0
7	3103	3724	3573	3078	EHC	3392	3695	2615	2820	2907	3211.9
8	EHC	3201	4074	3281	2940	2536	3097	3621	3333	3335	3268.7
9	EHC	2791	3254	EHC	EHC	EHC	EHC	EHC	2940	EHC	2995.0
10	EHC	3517	2962	3352	2961	2733	3197	2944	MEM	3583	3156.1
Average	2823.4	3042.4	3037.9	2834.1	2698.9	3113.3	2842.7	2710.8	2903.1	3171.8	2919.8

Table C.14: Temporal makespan of derived plans using RPCA[POPF] for Rovers problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	842	833	775	764	775	826	862	730	777	EHC	798.2
2	697	762	672	766	710	722	736	773	744	777	735.9
3	543	591	518	553	625	562	631	620	592	651	588.6
4	673	747	680	749	672	688	641	675	719	743	698.7
5	656	670	EHC	661	654	EHC	666	643	668	697	664.4
6	784	855	749	TIME	723	733	752	713	751	725	753.9
7	625	675	687	657	EHC	678	707	625	657	646	661.9
8	EHC	820	919	819	820	795	880	882	867	890	854.7
9	EHC	814	778	EHC	EHC	EHC	EHC	EHC	772	EHC	788.0
10	EHC	933	852	868	880	805	921	870	MEM	919	881.0
Average	688.6	770.0	736.7	729.6	732.4	726.1	755.1	725.7	727.4	756.0	736.2

Table C.15: Action execution steps in derived plans using RPCA[POPF] for Rovers problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	267.7	298.1	339.1	262.9	645.1	341.8	442.8	252.8	1107.5	EHC	439.8
2	262.1	401.6	186.1	241.1	237.4	401.3	282.9	330.2	318.4	307	296.8
3	84.9	325.2	85.4	157.2	216.2	147.5	276.3	173.7	407.4	357.7	223.2
4	127.5	220.6	141	200.4	251.4	333.9	189.7	297.1	297.8	1217.5	327.7
5	104.8	175	EHC	172.4	174.6	EHC	245.4	144.5	216.1	184.2	177.1
6	231.1	331.6	281.7	TIME	184.2	2666.1	1639.9	217.7	268.5	263	676.0
7	113.8	162.5	216.6	162	EHC	286.3	341.6	124.7	144	209.1	195.6
8	EHC	342	406.3	227.8	321.8	423.8	517.5	431.4	475.8	332.7	386.6
9	EHC	318.8	219.7	EHC	EHC	EHC	EHC	EHC	364.4	EHC	301.0
10	EHC	466.2	378.4	422.7	440.9	434.4	478.5	265.2	MEM	643.5	441.2
Average	170.3	304.2	250.5	230.8	309.0	629.4	490.5	248.6	400.0	439.3	348.3

Table C.16: Computation time to derive plans using RPCA[POPF] for Rovers problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	12.4	17.33	3.44	7.62	27.51	6.99	22.24	8.35	24.79	EHC	14.52
2	12.8	14.33	4.78	5.13	10.36	5.5	4.11	10.38	9.34	4.27	8.10
3	1.19	11.69	1.33	0.83	1.78	0.93	3.49	2.97	6.34	5.68	3.62
4	3.3	6.04	3.28	7.13	2.77	7.47	4.41	2.77	9.33	34.81	8.13
5	1.49	2.15	EHC	1.74	1.21	EHC	3.75	1.09	1.74	1.45	1.83
6	7.41	10.97	10.78	TIME	5.17	31.89	32.8	4.38	4.99	5.11	12.61
7	6.67	9.73	13.25	6.88	EHC	7.01	21.89	3.51	2.82	4.32	8.45
8	EHC	8.24	23.95	6.57	15.05	3.14	9.28	14.85	14.33	7.65	11.45
9	EHC	6.71	6.94	EHC	EHC	EHC	EHC	EHC	8.79	EHC	7.48
10	EHC	10.8	5.13	6.18	4.54	4.27	8.34	4.51	MEM	17.04	7.60
Average	6.47	9.80	8.10	5.26	8.55	8.40	12.26	5.87	9.16	10.04	8.47

Table C.17: Memory required to derive plans using RPCA[POPF] for Rovers problems.

C.5 Detailed Results for RPCA[ITSAT] Applied to the Rovers Problems

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	2251	2282	2267	2266	2292	2231	2308	2291	2298	2166	2265.2
2	2055	2090	1553	2157	1905	2638	2004	1891	2455	2553	2130.1
3	1268	1112	1227	1147	1259	1202	1219	1179	2074	1194	1288.1
4	1728	1558	1738	1672	1714	2678	1526	1743	1699	2521	1857.7
5	1431	1438	1531	1474	2029	2149	1444	1382	1364	2255	1649.7
6	2108	2240	1950	1879	2095	3269	2097	1994	2132	2042	2180.6
7	2083	2274	2153	2192	2135	1958	2033	2182	2140	1853	2100.3
8	1958	1914	1798	2074	1995	2070	1923	1882	2024	1978	1961.6
9	2011	1988	2032	2180	1954	2087	2005	1942	1992	2307	2049.8
10	1961	1998	1750	1723	2050	1788	1805	1970	3138	2085	2026.8
Average	1885.4	1889.4	1799.9	1876.4	1942.8	2207.0	1836.4	1845.6	2131.6	2095.4	1951.0

Table C.18: Temporal makespan of derived plans using RPCA[ITSAT] for Rovers problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	677	637	668	653	639	643	659	642	641	623	648.2
2	618	600	575	583	560	570	576	568	568	571	578.9
3	592	482	539	494	498	511	504	513	497	506	513.6
4	651	587	612	610	585	585	572	600	608	589	599.9
5	659	606	643	618	591	626	615	622	597	593	617.0
6	673	642	637	643	649	622	648	624	637	635	641.0
7	562	554	566	552	539	526	526	537	553	530	544.5
8	722	635	685	654	674	674	640	638	659	659	664.0
9	732	630	711	675	661	665	652	675	674	714	678.9
10	825	739	741	724	752	717	719	733	742	758	745.0
Average	671.1	611.2	637.7	620.6	614.8	613.9	611.1	615.2	617.6	617.8	623.1

Table C.19: Action execution steps in derived plans using RPCA[ITSAT] for Rovers problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	1910	1801	1866	1838	1771	1886	1898	1809	1821	1755	1835.5
2	1606	1415	1083	1399	1313	1265	1376	1279	1031	1260	1302.6
3	1391	1165	1257	1168	1153	1243	1086	1148	1072	1228	1191.1
4	1258	1223	1226	1378	1478	1291	1145	1392	1326	1305	1302.3
5	1387	1471	1455	1443	1290	1505	1301	1378	1253	1351	1383.4
6	1933	1854	1656	1528	1970	1797	1760	1703	1925	1826	1795.3
7	1332	1409	1470	1440	1384	1478	1346	1395	1464	1330	1404.6
8	2028	1862	2077	1973	1919	2096	1854	1851	1959	1919	1953.8
9	1831	1633	1911	1808	1845	1978	1768	1742	1698	1892	1810.7
10	2354	2251	2102	1938	2304	2114	1941	1928	1903	2408	2124.2
Average	1703.1	1608.4	1610.3	1591.4	1642.5	1665.3	1547.4	1562.4	1545.2	1627.4	1610.3

Table C.20: Computation time to derive plans using RPCA[ITSAT] for Rovers problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	1.54	1.18	1.53	1.43	1.66	2.27	1.58	1.69	2.11	1.99	1.70
2	1.56	1.88	1.75	1.32	1.52	3.52	1.91	1.17	2.91	3.72	2.13
3	0.99	1.21	1.19	1.29	1.18	1.75	1.20	1.07	1.53	1.33	1.27
4	1.10	2.54	1.36	1.19	1.27	2.85	1.51	1.42	1.40	2.60	1.72
5	1.10	1.39	1.27	1.25	1.48	2.76	1.57	1.09	1.24	2.16	1.53
6	1.77	2.21	2.04	2.47	1.79	3.92	3.46	1.69	2.13	2.05	2.35
7	1.02	1.63	2.14	2.12	2.38	3.12	2.18	2.27	2.77	2.54	2.22
8	1.48	1.54	1.31	1.40	1.70	2.96	1.96	1.58	1.99	1.99	1.79
9	1.72	1.92	1.60	1.70	1.37	2.83	2.50	1.79	1.91	2.56	1.99
10	1.80	1.94	1.41	1.35	1.43	2.50	1.84	1.50	2.31	2.03	1.81
Average	1.41	1.74	1.56	1.55	1.58	2.85	1.97	1.53	2.03	2.30	1.85

Table C.21: Memory required to derive plans using RPCA[ITSAT] for Rovers problems.

C.6 Detailed Results for TF[POPF] Applied to the Rovers Problems

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	MEM	TIME	3067	MEM	TIME	TIME	MEM	2934	MEM	TIME	3000.5
2	2924	TIME	MEM	MEM	MEM	MEM	2864	3079	MEM	MEM	2955.7
3	TIME	TIME	1886	TIME	MEM	TIME	2068	1958	MEM	TIME	1970.7
4	MEM	MEM	TIME	MEM	MEM	TIME	TIME	2441	TIME	TIME	2441.0
5	2142	TIME	TIME	TIME	TIME	2251	TIME	1981	TIME	TIME	2124.7
6	MEM	MEM	MEM	TIME	MEM	TIME	TIME	2865	MEM	TIME	2865.0
7	MEM	3733	3576	MEM	MEM	TIME	TIME	2623	MEM	TIME	3310.7
8	TIME	MEM	TIME	MEM	TIME	TIME	3103	3631	MEM	MEM	3367.0
9	TIME	TIME	MEM	TIME	MEM	TIME	TIME	TIME	TIME	TIME	N/A
10	MEM	TIME	MEM	TIME	TIME	TIME	TIME	2931	TIME	TIME	2931.0
Average	2533.0	3733.0	2843.0	N/A	N/A	2251.0	2678.3	2715.9	N/A	N/A	2739.8

Table C.22: Temporal makespan of derived plans using TF[POPF] for Rovers problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	MEM	TIME	747	MEM	TIME	TIME	MEM	730	MEM	TIME	738.5
2	646	TIME	MEM	MEM	MEM	MEM	755	773	MEM	MEM	724.7
3	TIME	TIME	538	TIME	MEM	TIME	606	620	MEM	TIME	588.0
4	MEM	MEM	TIME	MEM	MEM	TIME	TIME	675	TIME	TIME	675.0
5	659	TIME	TIME	TIME	TIME	637	TIME	643	TIME	TIME	646.3
6	MEM	MEM	MEM	TIME	MEM	TIME	TIME	713	MEM	TIME	713.0
7	MEM	706	674	MEM	MEM	TIME	TIME	625	MEM	TIME	668.3
8	TIME	MEM	TIME	MEM	TIME	TIME	809	884	MEM	MEM	846.5
9	TIME	TIME	MEM	TIME	MEM	TIME	TIME	TIME	TIME	TIME	N/A
10	MEM	TIME	MEM	TIME	TIME	TIME	TIME	870	TIME	TIME	870.0
Average	652.5	706.0	653.0	N/A	N/A	637.0	723.3	725.9	N/A	N/A	700.5

Table C.23: Action execution steps in derived plans using TF[POPF] for Rovers problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	MEM	TIME	2094.7	MEM	TIME	TIME	MEM	254.1	MEM	TIME	1174.4
2	3461.5	TIME	MEM	MEM	MEM	MEM	932.6	338.1	MEM	MEM	1577.4
3	TIME	TIME	897.1	TIME	MEM	TIME	1899.3	175.2	MEM	TIME	990.5
4	MEM	MEM	TIME	MEM	MEM	TIME	TIME	308.3	TIME	TIME	308.3
5	1495.6	TIME	TIME	TIME	TIME	3452.9	TIME	150.0	TIME	TIME	1699.5
6	MEM	MEM	MEM	TIME	MEM	TIME	TIME	221.1	MEM	TIME	221.1
7	MEM	1986.4	1284.3	MEM	MEM	TIME	TIME	126.3	MEM	TIME	1132.3
8	TIME	MEM	TIME	MEM	TIME	TIME	1386.4	449.7	MEM	MEM	918.1
9	TIME	TIME	MEM	TIME	MEM	TIME	TIME	TIME	TIME	TIME	N/A
10	MEM	TIME	MEM	TIME	TIME	TIME	TIME	264.7	TIME	TIME	264.7
Average	2478.6	1986.4	1425.4	N/A	N/A	3452.9	1406.1	254.2	N/A	N/A	1114.6

Table C.24: Computation time to derive plans using TF[POPF] for Rovers problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	MEM	TIME	43.85	MEM	TIME	TIME	MEM	8.85	MEM	TIME	26.35
2	31.84	TIME	MEM	MEM	MEM	MEM	6.07	10.38	MEM	MEM	16.10
3	TIME	TIME	18.29	TIME	MEM	TIME	10.06	3.27	MEM	TIME	10.54
4	MEM	MEM	TIME	MEM	MEM	TIME	TIME	2.77	TIME	TIME	2.77
5	23.78	TIME	TIME	TIME	TIME	26.91	TIME	1.23	TIME	TIME	17.31
6	MEM	MEM	MEM	TIME	MEM	TIME	TIME	4.59	MEM	TIME	4.59
7	MEM	18.57	25.65	MEM	MEM	TIME	TIME	3.65	MEM	TIME	15.96
8	TIME	MEM	TIME	MEM	TIME	TIME	9.60	15.05	MEM	MEM	12.33
9	TIME	TIME	MEM	TIME	MEM	TIME	TIME	TIME	TIME	TIME	N/A
10	MEM	TIME	MEM	TIME	TIME	TIME	TIME	5.01	TIME	TIME	5.01
Average	27.81	18.57	29.26	N/A	N/A	26.91	8.58	6.09	N/A	N/A	14.18

Table C.25: Memory required to derive plans using TF[POPF] for Rovers problems.

C.7 Detailed Results for TF[ITSAT] Applied to the Rovers Problems

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	1270	2474	2267	1778	2051	1934	2308	2291	2517	2170	2106.0
2	2099	1652	1480	2075	1856	2135	2274	1891	1892	2398	1975.2
3	1117	1104	1213	1252	1226	1209	1266	1179	1483	1844	1289.3
4	1708	1448	1415	1871	1859	1615	1738	1743	1839	1264	1650.0
5	1497	2005	1353	1645	1276	1394	1725	1382	1283	1855	1541.5
6	2191	2590	1902	2264	2194	2885	2195	1994	2300	2620	2313.5
7	1275	2274	2153	1919	1974	1501	2033	2182	2315	1628	1925.4
8	1592	1920	1798	2074	1939	1672	1963	1882	2316	2397	1955.3
9	2011	2091	1834	2199	2010	2087	2257	1942	2115	2694	2124.0
10	1990	1755	1633	2029	2188	1702	2271	1966	2185	2161	1988.0
Average	1675.0	1931.3	1704.8	1910.6	1857.3	1813.4	2003.0	1845.2	2024.5	2103.1	1886.8

Table C.26: Temporal makespan of derived plans using TF[ITSAT] for Rovers problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	639	638	614	658	680	674	757	642	728	665	669.5
2	595	632	578	597	602	576	681	568	602	613	604.4
3	526	482	536	501	515	492	569	513	522	540	519.6
4	581	610	576	606	604	577	687	600	617	620	607.8
5	639	634	639	642	604	584	708	622	582	624	627.8
6	644	636	623	673	684	645	663	624	653	659	650.4
7	547	553	560	557	555	569	568	537	616	586	564.8
8	676	664	637	680	707	683	637	638	720	697	673.9
9	734	672	679	729	665	675	815	675	700	742	708.6
10	762	710	700	719	743	702	747	745	751	762	734.1
Average	634.3	623.1	614.2	636.2	635.9	617.7	683.2	616.4	649.1	650.8	636.1

Table C.27: Action execution steps in derived plans using TF[ITSAT] for Rovers problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	1319	1320	1610	1416	1574	1695	2182	1835	1778	1321	1604.9
2	1357	1317	1106	1094	1223	1553	1701	1295	1107	1405	1315.8
3	940	889	1121	951	1133	1273	1241	1164	1150	1367	1122.8
4	972	1059	1035	964	1083	1151	1220	1389	938	1209	1102.0
5	1167	1220	1125	1132	1123	1225	1402	1383	879	1096	1175.2
6	1560	1540	1543	1463	1425	1961	2111	1720	1443	1752	1651.7
7	957	1588	1382	1107	1178	1602	1561	1396	1310	1290	1337.1
8	1383	1639	1646	1403	1523	1863	2055	1893	1734	1520	1665.9
9	1657	1516	1342	1570	1521	1962	2040	1741	1320	1858	1652.7
10	1624	1814	1639	1443	1615	1581	1909	2010	1533	1905	1707.2
Average	1293.6	1390.2	1354.8	1254.2	1339.7	1586.6	1742.2	1582.6	1319.2	1472.2	1433.5

Table C.28: Computation time to derive plans using TF[ITSAT] for Rovers problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	2.73	3.57	2.30	3.49	4.07	5.32	3.92	1.76	5.57	7.01	3.97
2	2.31	4.59	3.74	5.08	3.76	3.95	5.46	1.35	4.57	5.13	3.99
3	2.06	3.42	2.59	3.45	3.12	4.30	3.68	1.07	3.56	5.41	3.27
4	2.29	3.78	3.44	3.39	3.54	3.93	3.93	1.21	3.88	4.36	3.38
5	2.39	3.72	2.80	3.38	3.03	3.74	4.15	1.08	3.47	4.28	3.20
6	2.82	4.94	4.23	6.85	6.30	5.31	7.15	1.68	5.53	6.04	5.09
7	2.73	3.52	2.24	5.23	3.87	5.25	3.95	1.75	5.08	4.55	3.82
8	2.77	4.13	2.92	4.16	3.81	5.37	4.86	1.58	5.74	5.44	4.08
9	2.83	4.63	3.62	4.33	4.19	4.72	7.03	1.52	5.29	5.87	4.40
10	2.88	4.91	2.89	3.83	3.66	4.99	5.93	1.50	5.05	7.10	4.27
Average	2.58	4.12	3.08	4.32	3.94	4.69	5.01	1.45	4.77	5.52	3.95

Table C.29: Memory required to derive plans using TF[ITSAT] for Rovers problems.

Appendix D

Detailed Blocksworld Results

This appendix presents the detailed results for the Blocks World problems. Each table has a heatmap overlay, ranging from better results in green (lower values for all metrics) to worse results in red (higher values for all metrics). The range for each heatmap is identical for the same metrics (e.g., the makespan results for the problems solved using PA[TFD] have the same heatmap range as the makespan results for problems solved using RPCA[COLIN]) to facilitate comparison across tool configurations, with specific values for ranges presented in Table D.1. The overlay for time results ranges from 0 seconds (green) to 3600 seconds (red), and the overlay for the memory results ranges from 0 GB (green) to 48 GB (red). The range for action executions is from 40 (green) to 170 (red) and for makespan is from 20 (green) to 170 (red).

Metric	**Green**	**Red**
Makespan	20	170
Action Executions	40	170
Time	0 seconds	3600 seconds
Memory	0 GB	48 GB

Table D.1: Heatmap ranges for each metric in the Blocks World domain

D.1 Detailed Results for PA[TFD] Applied to the Blocks World Problems

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	MEM	TIME	MEM	57	MEM	47	TIME	TIME	TIME	39	47.7
2	31	24	33	39	47	26	32	24	26	21	30.3
3	MEM	TIME	MEM	MEM	MEM	TIME	TIME	TIME	TIME	TIME	N/A
4	TIME	TIME	MEM	TIME	MEM	TIME	TIME	TIME	TIME	TIME	N/A
5	37	30	MEM	39	46	TIME	32	31	MEM	MEM	35.8
6	45	35	61	MEM	69	33	34	38	30	33	42.0
7	MEM	TIME	MEM	MEM	51	TIME	MEM	MEM	TIME	TIME	51.0
8	MEM	TIME	MEM	TIME	MEM	TIME	TIME	TIME	TIME	TIME	N/A
9	31	33	MEM	MEM	MEM	23	26	MEM	34	29	29.3
10	36	30	MEM	MEM	48	21	37	MEM	25	23	31.4
Average	36.0	30.4	47.0	45.0	52.2	30.0	32.2	31.0	28.8	29.0	35.4

Table D.2: Temporal makespan of derived plans using PA[TFD] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	MEM	TIME	MEM	99	MEM	105	TIME	TIME	TIME	80	94.7
2	65	47	64	75	93	52	74	61	54	51	63.6
3	MEM	TIME	MEM	MEM	MEM	TIME	TIME	TIME	TIME	TIME	N/A
4	TIME	TIME	MEM	TIME	MEM	TIME	TIME	TIME	TIME	TIME	N/A
5	63	73	MEM	68	87	TIME	68	68	MEM	MEM	71.2
6	95	68	139	MEM	127	73	91	87	70	66	90.7
7	MEM	TIME	MEM	MEM	95	TIME	MEM	MEM	TIME	TIME	95.0
8	MEM	TIME	MEM	TIME	MEM	TIME	TIME	TIME	TIME	TIME	N/A
9	58	60	MEM	MEM	MEM	52	55	MEM	77	72	62.3
10	69	59	MEM	MEM	82	46	52	MEM	49	48	57.9
Average	70.0	61.4	101.5	80.7	96.8	65.6	68.0	72.0	62.5	63.4	72.3

Table D.3: Action execution steps in derived plans using PA[TFD] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	MEM	TIME	MEM	2372.6	MEM	6489.1	TIME	TIME	TIME	5042.8	4634.8
2	382.7	138.4	105.7	569.1	1451.1	81.6	1779.7	44.9	250.1	34.2	483.8
3	MEM	TIME	MEM	MEM	MEM	TIME	TIME	TIME	TIME	TIME	N/A
4	TIME	TIME	MEM	TIME	MEM	TIME	TIME	TIME	TIME	TIME	N/A
5	15.2	722.6	MEM	62.8	152.9	TIME	1666.7	84.7	MEM	MEM	450.8
6	28.5	7.4	1850.2	MEM	234.7	36.3	24.5	113.5	24	33.2	261.4
7	MEM	TIME	MEM	MEM	478.3	TIME	MEM	MEM	TIME	TIME	478.3
8	MEM	TIME	MEM	TIME	MEM	TIME	TIME	TIME	TIME	TIME	N/A
9	10.3	54.3	MEM	MEM	MEM	50.4	18.2	MEM	35.9	174.9	57.3
10	7.8	7.1	MEM	MEM	551.4	6.1	13.8	MEM	9.5	31.8	89.6
Average	88.9	186.0	978.0	1001.5	573.7	1332.7	700.6	81.0	79.9	1063.4	601.2

Table D.4: Computation time to derive plans using PA[TFD] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	MEM	TIME	MEM	27.06	MEM	25.8	TIME	TIME	TIME	14.17	22.34
2	7.04	0.58	1.82	11.21	43.91	0.39	26.39	0.35	2.41	0.21	9.43
3	MEM	TIME	MEM	MEM	MEM	TIME	TIME	TIME	TIME	TIME	N/A
4	TIME	TIME	MEM	TIME	MEM	TIME	TIME	TIME	TIME	TIME	N/A
5	0.21	4.17	MEM	0.83	4.95	TIME	23.33	0.73	MEM	MEM	5.70
6	0.27	0.06	28.17	MEM	4.18	0.2	0.16	0.85	0.16	0.22	3.81
7	MEM	TIME	MEM	MEM	9.49	TIME	MEM	MEM	TIME	TIME	9.49
8	MEM	TIME	MEM	TIME	MEM	TIME	TIME	TIME	TIME	TIME	N/A
9	0.09	0.29	MEM	MEM	MEM	0.23	0.11	MEM	0.22	0.73	0.28
10	0.12	0.07	MEM	MEM	13.15	0.05	0.12	MEM	0.07	0.2	1.97
Average	1.55	1.03	15.00	13.03	15.14	5.33	10.02	0.64	0.72	3.11	6.07

Table D.5: Memory required to derive plans using PA[TFD] for Blocks World problems.

D.2 Detailed Results for PA[COLIN] Applied to the Blocks World Problems

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	MEM	MEM	MEM	58	58	MEM	MEM	TIME	TIME	TIME	58.0
2	33	MEM	32	36	39	MEM	MEM	MEM	MEM	MEM	35.0
3	MEM	TIME	MEM	59	MEM	MEM	MEM	MEM	MEM	TIME	59.0
4	MEM	TIME	61	MEM	MEM	TIME	MEM	MEM	49	TIME	55.0
5	41	MEM	35	62	47	MEM	MEM	MEM	MEM	MEM	46.3
6	34	MEM	MEM	37	44	MEM	MEM	MEM	MEM	39	38.5
7	MEM	MEM	52	64	67	MEM	TIME	52	MEM	41	55.2
8	MEM	TIME	MEM	MEM	MEM	TIME	TIME	TIME	TIME	TIME	N/A
9	38	37	MEM	40	59	32	TIME	MEM	TIME	28	39.0
10	42	MEM	MEM	MEM	36	MEM	MEM	MEM	MEM	MEM	39.0
Average	37.6	37.0	45.0	50.9	50.0	32.0	N/A	52.0	49.0	36.0	45.1

Table D.6: Temporal makespan of derived plans using PA[COLIN] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	MEM	MEM	MEM	61	79	MEM	MEM	TIME	TIME	TIME	70.0
2	44	MEM	46	44	46	MEM	MEM	MEM	MEM	MEM	45.0
3	MEM	TIME	MEM	67	MEM	MEM	MEM	MEM	MEM	TIME	67.0
4	MEM	TIME	60	MEM	MEM	TIME	MEM	MEM	55	TIME	57.5
5	57	MEM	61	70	65	MEM	MEM	MEM	MEM	MEM	63.3
6	71	MEM	MEM	61	60	MEM	MEM	MEM	MEM	52	61.0
7	MEM	MEM	64	88	84	MEM	TIME	71	MEM	51	71.6
8	MEM	TIME	MEM	MEM	MEM	TIME	TIME	TIME	TIME	TIME	N/A
9	49	47	MEM	55	60	40	TIME	MEM	TIME	32	47.2
10	77	MEM	MEM	MEM	52	MEM	MEM	MEM	MEM	MEM	64.5
Average	59.6	47.0	57.8	63.7	63.7	40.0	N/A	71.0	55.0	45.0	59.0

Table D.7: Action execution steps in derived plans using PA[COLIN] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	MEM	MEM	MEM	547	303	MEM	MEM	TIME	TIME	TIME	424.7
2	324	MEM	374	373	593	MEM	MEM	MEM	MEM	MEM	416.2
3	MEM	TIME	MEM	205	MEM	MEM	MEM	MEM	MEM	TIME	205.4
4	MEM	TIME	964	MEM	MEM	TIME	MEM	MEM	976	TIME	969.7
5	112	MEM	88	118	26	MEM	MEM	MEM	MEM	MEM	86.0
6	316	MEM	MEM	112	68	MEM	MEM	MEM	MEM	2277	693.4
7	MEM	MEM	147	786	176	MEM	TIME	789	MEM	432	466.0
8	MEM	TIME	MEM	MEM	MEM	TIME	TIME	TIME	TIME	TIME	N/A
9	250	1974	MEM	21	679	270	TIME	MEM	TIME	6	533.3
10	605	MEM	MEM	MEM	28	MEM	MEM	MEM	MEM	MEM	316.6
Average	321.5	1974.2	393.1	308.9	267.7	269.6	N/A	789.0	975.6	905.2	464.7

Table D.8: Computation time to derive plans using PA[COLIN] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	MEM	MEM	MEM	16.75	25.34	MEM	MEM	TIME	TIME	TIME	21.05
2	14.72	MEM	16.91	15.39	33.03	MEM	MEM	MEM	MEM	MEM	20.01
3	MEM	TIME	MEM	5.44	MEM	MEM	MEM	MEM	MEM	TIME	5.44
4	MEM	TIME	25.17	MEM	MEM	TIME	MEM	MEM	10.27	TIME	17.72
5	5.25	MEM	3.66	5.27	1.90	MEM	MEM	MEM	MEM	MEM	4.02
6	11.72	MEM	MEM	5.08	3.05	MEM	MEM	MEM	MEM	32.14	13.00
7	MEM	MEM	3.66	37.03	8.62	MEM	TIME	20.32	MEM	7.30	15.39
8	MEM	TIME	MEM	MEM	MEM	TIME	TIME	TIME	TIME	TIME	N/A
9	4.10	22.15	MEM	0.68	37.85	3.58	TIME	MEM	TIME	0.07	11.41
10	31.62	MEM	MEM	MEM	1.38	MEM	MEM	MEM	MEM	MEM	16.50
Average	13.48	22.15	12.35	12.23	15.88	3.58	N/A	20.32	10.27	13.17	13.65

Table D.9: Memory required to derive plans using PA[COLIN] for Blocks World problems.

D.3 Detailed Results for CFP[TFD] Applied to the Blocks World Problems

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	50	TIME	NE	55	56	47	65	NE	NE	NE	54.6
2	61	TIME	35	62	NE	47	53	NE	NE	NE	51.6
3	NE	NE	NE	NE	NE	NE	NE	72	NE	NE	72.0
4	TIME	71	TIME	NE	TIME	NE	TIME	TIME	NE	TIME	71.0
5	TIME	NE	NE	56	42	NE	NE	45	52	TIME	48.8
6	NE	NE	50	NE	NE	NE	67	NE	NE	NE	58.5
7	42	NE	49	46	37	NE	NE	NE	NE	NE	43.5
8	TIME	NE	TIME	TIME	TIME	NE	NE	NE	NE	NE	N/A
9	NE	NE	NE	NE	NE	NE	58	NE	NE	NE	58.0
10	35	NE	NE	NE	30	32	NE	TIME	41	NE	34.5
Average	47.0	71.0	44.7	54.8	41.3	42.0	60.8	58.5	46.5	N/A	50.2

Table D.10: Temporal makespan of derived plans using CFP[TFD] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	79	TIME	NE	79	85	94	86	NE	NE	NE	84.6
2	96	TIME	64	89	NE	76	77	NE	NE	NE	80.4
3	NE	NE	NE	NE	NE	NE	NE	116	NE	NE	116.0
4	TIME	120	TIME	NE	TIME	NE	TIME	TIME	NE	TIME	120.0
5	TIME	NE	NE	79	78	NE	NE	73	72	TIME	75.5
6	NE	NE	83	NE	NE	NE	108	NE	NE	NE	95.5
7	91	NE	80	85	62	NE	NE	NE	NE	NE	79.5
8	TIME	NE	TIME	TIME	TIME	NE	NE	NE	NE	NE	N/A
9	NE	NE	NE	NE	NE	NE	88	NE	NE	NE	88.0
10	52	NE	NE	NE	37	53	NE	TIME	59	NE	50.3
Average	79.5	120.0	75.7	83.0	65.5	74.3	89.8	94.5	65.5	N/A	80.0

Table D.11: Action execution steps in derived plans using CFP[TFD] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	44	TIME	NE	48	39	1044	82	NE	NE	NE	251.3
2	16	TIME	12	416	NE	514	562	NE	NE	NE	303.9
3	NE	NE	NE	NE	NE	NE	NE	3112	NE	NE	3111.9
4	TIME	389	TIME	NE	TIME	NE	TIME	TIME	NE	TIME	388.5
5	TIME	NE	NE	210	114	NE	NE	495	2567	TIME	846.3
6	NE	NE	1770	NE	NE	NE	654	NE	NE	NE	1211.6
7	38	NE	39	287	63	NE	NE	NE	NE	NE	106.8
8	TIME	NE	TIME	TIME	TIME	NE	NE	NE	NE	NE	N/A
9	NE	NE	NE	NE	NE	NE	696	NE	NE	NE	695.7
10	9	NE	NE	NE	3	6	NE	TIME	5	NE	5.7
Average	26.7	388.5	606.7	240.1	54.6	521.3	498.5	1803.6	1286.0	N/A	490.0

Table D.12: Computation time to derive plans using CFP[TFD] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	0.16	TIME	NE	0.15	0.15	15.47	0.18	NE	NE	NE	3.22
2	0.14	TIME	0.09	8.66	NE	11.64	14.17	NE	NE	NE	6.94
3	NE	NE	NE	NE	NE	NE	NE	17.45	NE	NE	17.45
4	TIME	0.48	TIME	NE	TIME	NE	TIME	TIME	NE	TIME	0.48
5	TIME	NE	NE	4.88	3.75	NE	NE	12.64	25.55	TIME	11.71
6	NE	NE	30.80	NE	NE	NE	8.01	NE	NE	NE	19.41
7	0.40	NE	0.24	3.30	0.65	NE	NE	NE	NE	NE	1.15
8	TIME	NE	TIME	TIME	TIME	NE	NE	NE	NE	NE	N/A
9	NE	NE	NE	NE	NE	NE	11.55	NE	NE	NE	11.55
10	0.09	NE	NE	NE	0.01	0.03	NE	TIME	0.05	NE	0.05
Average	0.20	0.48	10.38	4.25	1.14	9.05	8.48	15.05	12.80	N/A	6.32

Table D.13: Memory required to derive plans using CFP[TFD] for Blocks World problems.

D.4 Detailed Results for CFP[COLIN] Applied to the Blocks World Problems

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	MEM	65	70	63	77	NE	NE	NE	NE	NE	68.8
2	NE	41	39	51	65	51	70	NE	NE	NE	52.8
3	NE	NE	MEM	NE	80	NE	NE	77	NE	NE	78.5
4	120	TIME	MEM	106	MEM	MEM	115	93	171	82	114.5
5	NE	63	NE	61	45	62	25	NE	54	NE	51.7
6	NE	NE	32	35	59	NE	NE	35	97	NE	51.6
7	50	NE	39	NE	NE	NE	NE	NE	NE	MEM	44.5
8	MEM	NE	NE	69	117	NE	NE	NE	104	MEM	96.7
9	NE	NE	NE	44	NE	NE	NE	NE	NE	NE	44.0
10	40	MEM	TIME	MEM	30	MEM	NE	MEM	47	NE	39.0
Average	70.0	56.3	45.0	61.3	67.6	56.5	70.0	68.3	94.6	82.0	66.9

Table D.14: Temporal makespan of derived plans using CFP[COLIN] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	MEM	83	75	73	78	NE	NE	NE	NE	NE	77.3
2	NE	43	58	53	72	72	82	NE	NE	NE	63.3
3	NE	NE	MEM	NE	78	NE	NE	91	NE	NE	84.5
4	104	TIME	MEM	90	MEM	MEM	124	92	140	85	105.8
5	NE	81	NE	68	57	73	46	NE	63	NE	64.7
6	NE	NE	41	56	64	NE	NE	52	110	NE	64.6
7	68	NE	62	NE	NE	NE	NE	NE	NE	MEM	65.0
8	MEM	NE	NE	76	114	NE	NE	NE	97	MEM	95.7
9	NE	NE	NE	50	NE	NE	NE	NE	NE	NE	50.0
10	48	MEM	TIME	MEM	41	MEM	NE	MEM	66	NE	51.7
Average	73.3	69.0	59.0	66.6	72.0	72.5	84.0	78.3	95.2	85.0	74.4

Table D.15: Action execution steps in derived plans using CFP[COLIN] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	MEM	2377	527	6	38	NE	NE	NE	NE	NE	737.1
2	NE	7	27	4	11	12	417	NE	NE	NE	79.7
3	NE	NE	MEM	NE	35	NE	NE	114	NE	NE	74.1
4	872	TIME	MEM	124	MEM	MEM	145	17	112	108	229.4
5	NE	80	NE	43	7	87	7	NE	41	NE	44.1
6	NE	NE	4	3	11	NE	NE	21	11	NE	10.0
7	6	NE	4	NE	NE	NE	NE	NE	NE	MEM	5.0
8	MEM	NE	NE	110	103	NE	NE	NE	282	MEM	164.8
9	NE	NE	NE	13	NE	NE	NE	NE	NE	NE	13.4
10	36	MEM	TIME	MEM	2	MEM	NE	MEM	90	NE	42.5
Average	304.8	821.2	140.4	43.5	29.4	49.2	189.6	50.2	107.1	108.1	155.6

Table D.16: Computation time to derive plans using CFP[COLIN] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	MEM	18.51	9.81	0.05	1.18	NE	NE	NE	NE	NE	7.39
2	NE	0.13	0.64	0.07	0.21	0.22	8.97	NE	NE	NE	1.71
3	NE	NE	MEM	NE	0.57	NE	NE	1.00	NE	NE	0.79
4	4.83	TIME	MEM	0.96	MEM	MEM	1.45	0.15	1.08	2.27	1.79
5	NE	5.03	NE	2.46	0.31	5.50	0.16	NE	2.46	NE	2.65
6	NE	NE	0.05	0.04	0.18	NE	NE	0.27	0.15	NE	0.14
7	0.06	NE	0.02	NE	NE	NE	NE	NE	NE	MEM	0.04
8	MEM	NE	NE	2.03	2.87	NE	NE	NE	2.76	MEM	2.55
9	NE	NE	NE	0.22	NE	NE	NE	NE	NE	NE	0.22
10	0.86	MEM	TIME	MEM	0.02	MEM	NE	MEM	4.86	NE	1.91
Average	1.92	7.89	2.63	0.83	0.76	2.86	3.53	0.47	2.26	2.27	2.17

Table D.17: Memory required to derive plans using CFP[COLIN] for Blocks World problems.

D.5 Detailed Results for RPCA[TFD] Applied to the Blocks World Problems

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	50	TIME	TIME	55	56	47	65	49	57	58	54.6
2	61	TIME	35	62	TIME	47	53	45	40	22	45.6
3	TIME	TIME	65	74	TIME	148	73	72	70	94	85.1
4	TIME	71	TIME	TIME	TIME	60	TIME	TIME	TIME	TIME	65.5
5	TIME	73	39	56	42	61	45	45	52	TIME	51.6
6	39	63	50	42	TIME	90	67	45	69	TIME	58.1
7	42	TIME	49	46	37	47	38	41	42	64	45.1
8	TIME	62	TIME	TIME	TIME	63	TIME	TIME	114	TIME	79.7
9	45	34	47	41	59	38	58	47	56	45	47.0
10	35	33	TIME	TIME	30	32	45	TIME	41	29	35.0
Average	45.3	56.0	47.5	53.7	44.8	63.3	55.5	49.1	60.1	52.0	53.8

Table D.18: Temporal makespan of derived plans using RPCA[TFD] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	79	TIME	TIME	79	85	94	86	77	92	109	87.6
2	96	TIME	64	89	TIME	76	77	79	62	43	73.3
3	TIME	TIME	104	99	TIME	168	106	116	96	117	115.1
4	TIME	120	TIME	TIME	TIME	76	TIME	TIME	TIME	TIME	98.0
5	TIME	112	76	79	78	96	83	73	72	TIME	83.6
6	79	95	83	63	TIME	136	108	84	95	TIME	92.9
7	91	TIME	80	85	62	66	67	68	74	98	76.8
8	TIME	99	TIME	TIME	TIME	102	TIME	TIME	149	TIME	116.7
9	77	61	72	72	82	64	88	69	77	80	74.2
10	52	50	TIME	TIME	37	53	53	TIME	59	46	50.0
Average	79.0	89.5	79.8	80.9	68.8	93.1	83.5	80.9	86.2	82.2	83.3

Table D.19: Action execution steps in derived plans using RPCA[TFD] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	44	TIME	TIME	48	39	1044	82	77	170	1671	396.9
2	16	TIME	12	416	TIME	514	562	225	422	25	274.0
3	TIME	TIME	33	27	TIME	76	60	3112	77	1496	697.4
4	TIME	389	TIME	TIME	TIME	228	TIME	TIME	TIME	TIME	308.5
5	TIME	558	970	210	114	130	1509	495	2567	TIME	819.0
6	8	12	1770	14	TIME	53	654	13	12	TIME	316.9
7	38	TIME	39	287	63	91	64	443	107	1368	277.6
8	TIME	1593	TIME	TIME	TIME	145	TIME	TIME	1823	TIME	1187.0
9	8	13	204	8	8	17	696	9	40	11	101.3
10	9	5	TIME	TIME	3	6	4	TIME	5	10	6.0
Average	20.4	428.3	504.6	144.2	45.2	230.5	453.9	624.9	580.2	763.4	386.7

Table D.20: Computation time to derive plans using RPCA[TFD] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	0.16	TIME	TIME	0.15	0.15	15.47	0.18	0.20	0.30	22.28	4.86
2	0.14	TIME	0.09	8.66	TIME	11.64	14.17	5.20	8.66	0.17	6.09
3	TIME	TIME	0.17	0.12	TIME	0.23	0.23	17.45	0.17	21.56	5.70
4	TIME	0.48	TIME	TIME	TIME	1.48	TIME	TIME	TIME	TIME	0.98
5	TIME	14.65	18.76	4.88	3.75	3.28	25.70	12.64	25.55	TIME	13.65
6	0.03	0.04	30.80	0.10	TIME	0.87	8.01	0.05	0.04	TIME	4.99
7	0.40	TIME	0.24	3.30	0.65	0.51	0.23	4.69	1.47	6.63	2.01
8	TIME	5.72	TIME	TIME	TIME	0.41	TIME	TIME	7.48	TIME	4.54
9	0.03	0.07	2.49	0.03	0.03	0.09	11.55	0.05	0.53	0.04	1.49
10	0.09	0.02	TIME	TIME	0.01	0.03	0.03	TIME	0.05	0.14	0.05
Average	0.14	3.50	8.76	2.46	0.92	3.40	7.51	5.75	4.92	8.47	4.65

Table D.21: Memory required to derive plans using RPCA[TFD] for Blocks World problems.

D.6 Detailed Results for RPCA[COLIN] Applied to the Blocks World Problems

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	MEM	65	70	63	77	53	TIME	65	66	54	64.1
2	41	41	39	51	65	51	70	43	MEM	22	47.0
3	54	51	MEM	66	80	111	73	77	76	71	73.2
4	120	TIME	MEM	106	MEM	MEM	115	93	171	82	114.5
5	39	63	32	61	45	62	25	52	54	32	46.5
6	34	40	32	35	59	62	68	35	97	29	49.1
7	50	36	39	42	45	52	44	34	41	MEM	42.6
8	MEM	80	102	69	117	51	116	MEM	104	MEM	91.3
9	43	40	41	44	64	42	45	45	50	40	45.4
10	40	MEM	TIME	MEM	30	MEM	39	MEM	47	38	38.8
Average	52.6	52.0	50.7	59.7	64.7	60.5	66.1	55.5	78.4	46.0	59.1

Table D.22: Temporal makespan of derived plans using RPCA[COLIN] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	MEM	83	75	73	78	76	TIME	70	79	59	74.1
2	60	43	58	53	72	72	82	52	MEM	45	59.7
3	64	54	MEM	81	78	125	95	91	83	76	83.0
4	104	TIME	MEM	90	MEM	MEM	124	92	140	85	105.8
5	48	81	59	68	57	73	46	59	63	43	59.7
6	55	52	41	56	64	73	75	52	110	44	62.2
7	68	57	62	60	48	69	60	56	51	MEM	59.0
8	MEM	84	104	76	114	68	111	MEM	97	MEM	93.4
9	59	51	41	50	66	49	46	50	48	60	52.0
10	48	MEM	TIME	MEM	41	MEM	49	MEM	66	45	49.8
Average	63.3	63.1	62.9	67.4	68.7	75.6	76.4	65.3	81.9	57.1	68.5

Table D.23: Action execution steps in derived plans using RPCA[COLIN] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	MEM	2377	527	6	38	9	TIME	1589	108	120	596.7
2	31	7	27	4	11	12	417	15	MEM	109	70.4
3	12	10	MEM	17	35	31	129	114	43	38	47.4
4	872	TIME	MEM	124	MEM	MEM	145	17	112	108	229.4
5	4	80	4	43	7	87	7	15	41	19	30.6
6	5	4	4	3	11	6	12	21	11	57	13.4
7	6	6	4	5	3	9	7	7	4	MEM	5.6
8	MEM	1466	304	110	103	186	326	MEM	282	MEM	396.5
9	3	11	2	13	15	579	7	12	3	38	68.2
10	36	MEM	TIME	MEM	2	MEM	37	MEM	90	113	55.5
Average	121.1	495.0	124.4	36.2	24.8	114.7	120.8	223.6	77.0	75.2	137.8

Table D.24: Computation time to derive plans using RPCA[COLIN] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	MEM	18.51	9.81	0.05	1.18	0.15	TIME	16.77	2.78	1.78	6.38
2	0.98	0.13	0.64	0.07	0.21	0.22	8.97	0.44	MEM	2.91	1.62
3	0.14	0.10	MEM	0.24	0.57	0.18	2.19	1.00	0.82	1.27	0.72
4	4.83	TIME	MEM	0.96	MEM	MEM	1.45	0.15	1.08	2.27	1.79
5	0.08	5.03	0.08	2.46	0.31	5.50	0.16	0.73	2.46	0.43	1.72
6	0.06	0.03	0.05	0.04	0.18	0.06	0.31	0.27	0.15	0.78	0.19
7	0.06	0.04	0.02	0.03	0.02	0.10	0.07	0.11	0.03	MEM	0.05
8	MEM	7.22	5.23	2.03	2.87	1.35	2.90	MEM	2.76	MEM	3.48
9	0.03	0.18	0.02	0.22	0.14	4.94	0.08	0.17	0.02	0.36	0.62
10	0.86	MEM	TIME	MEM	0.02	MEM	0.88	MEM	4.86	1.55	1.63
Average	0.88	3.91	2.26	0.68	0.61	1.56	1.89	2.46	1.66	1.42	1.70

Table D.25: Memory required to derive plans using RPCA[COLIN] for Blocks World problems.

D.7 Detailed Results for TF[TFD] Applied to the Blocks World Problems

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	TIME	TIME	TIME	54	TIME	TIME	69	TIME	57	TIME	60.0
2	41	50	34	54	TIME	TIME	60	34	46	23	42.8
3	TIME	TIME	TIME	TIME	TIME	TIME	65	TIME	TIME	65	65.0
4	TIME	TIME	TIME	61	TIME	TIME	TIME	TIME	TIME	TIME	61.0
5	30	76	41	62	37	66	48	49	TIME	TIME	51.1
6	40	58	41	63	TIME	96	70	46	81	66	62.3
7	TIME	TIME	63	TIME	TIME	62	TIME	TIME	69	TIME	64.7
8	TIME	TIME	TIME	TIME	TIME	99	120	TIME	89	TIME	102.7
9	62	29	58	60	TIME	54	56	61	64	37	53.4
10	33	34	TIME	TIME	41	32	46	33	42	31	36.5
Average	41.2	49.4	47.4	59.0	39.0	68.2	66.8	44.6	64.0	44.4	54.8

Table D.26: Temporal makespan of derived plans using TF[TFD] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	TIME	TIME	TIME	68	TIME	TIME	76	TIME	99	TIME	81.0
2	63	69	63	79	TIME	TIME	82	61	92	43	69.0
3	TIME	TIME	TIME	TIME	TIME	TIME	102	TIME	TIME	94	98.0
4	TIME	TIME	TIME	75	TIME	TIME	TIME	TIME	TIME	TIME	75.0
5	53	112	74	76	59	96	83	73	TIME	TIME	78.3
6	84	86	64	91	TIME	136	108	73	110	101	94.8
7	TIME	TIME	96	TIME	TIME	72	TIME	TIME	96	TIME	88.0
8	TIME	TIME	TIME	TIME	TIME	142	152	TIME	116	TIME	136.7
9	88	51	87	93	TIME	78	68	87	85	77	79.3
10	48	50	TIME	TIME	53	52	57	61	59	46	53.3
Average	67.2	73.6	76.8	80.3	56.0	96.0	91.0	71.0	93.9	72.2	80.7

Table D.27: Action execution steps in derived plans using TF[TFD] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	TIME	TIME	TIME	554	TIME	TIME	432	TIME	452	TIME	479.3
2	15	664	13	771	TIME	TIME	80	321	1318	28	401.1
3	TIME	TIME	TIME	TIME	TIME	TIME	1866	TIME	TIME	160	1012.9
4	TIME	TIME	TIME	114	TIME	TIME	TIME	TIME	TIME	TIME	114.0
5	645	656	1171	1163	150	165	1810	580	TIME	TIME	792.4
6	14	11	24	14	TIME	62	761	10	35	16	105.1
7	TIME	TIME	2765	TIME	TIME	158	TIME	TIME	84	TIME	1002.4
8	TIME	TIME	TIME	TIME	TIME	159	1699	TIME	1295	TIME	1051.0
9	10	7	955	22	TIME	109	12	38	255	9	157.5
10	8	6	TIME	TIME	4	5	13	9	6	10	7.6
Average	138.2	268.9	985.5	439.6	77.0	109.7	834.2	191.6	492.0	44.5	402.0

Table D.28: Computation time to derive plans using TF[TFD] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	TIME	TIME	TIME	6.87	TIME	TIME	3.90	TIME	1.02	TIME	3.93
2	0.14	12.91	0.09	15.45	TIME	TIME	1.03	6.12	13.04	0.16	6.12
3	TIME	TIME	TIME	TIME	TIME	TIME	19.59	TIME	TIME	0.64	10.12
4	TIME	TIME	TIME	0.51	TIME	TIME	TIME	TIME	TIME	TIME	0.51
5	10.58	14.65	18.77	23.51	4.45	3.28	25.70	12.64	TIME	TIME	14.20
6	0.08	0.03	0.24	0.08	TIME	0.87	8.01	0.05	0.45	0.05	1.10
7	TIME	TIME	38.29	TIME	TIME	1.77	TIME	TIME	0.84	TIME	13.63
8	TIME	TIME	TIME	TIME	TIME	0.47	19.34	TIME	11.70	TIME	10.50
9	0.03	0.02	10.44	0.17	TIME	1.01	0.05	0.31	3.57	0.03	1.74
10	0.07	0.02	TIME	TIME	0.03	0.03	0.10	0.07	0.05	0.14	0.06
Average	2.18	5.53	13.57	7.77	2.24	1.24	9.72	3.84	4.38	0.20	5.43

Table D.29: Memory required to derive plans using TF[TFD] for Blocks World problems.

D.8 Detailed Results for TF[COLIN] Applied to the Blocks World Problems

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	64	MEM	MEM	68	71	50	67	MEM	66	54	62.9
2	41	41	39	62	65	51	70	43	MEM	22	48.2
3	63	77	62	98	78	64	46	60	105	91	74.4
4	MEM	TIME	MEM	MEM	MEM	MEM	68	69	129	TIME	88.7
5	39	63	32	65	45	62	25	52	54	32	46.9
6	43	40	42	67	72	62	68	40	75	33	54.2
7	62	MEM	58	70	81	89	55	53	88	58	68.2
8	MEM	69	78	91	96	82	98	85	118	MEM	89.6
9	43	40	34	44	72	42	45	MEM	45	40	45.0
10	40	MEM	MEM	MEM	32	MEM	39	MEM	47	38	39.2
Average	49.4	55.0	49.3	70.6	68.0	62.8	58.1	57.4	80.8	46.0	60.3

Table D.30: Temporal makespan of derived plans using TF[COLIN] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	78	MEM	MEM	74	71	61	71	MEM	79	59	70.4
2	60	43	58	61	72	72	82	52	MEM	45	60.6
3	67	82	67	98	86	75	68	85	109	94	83.1
4	MEM	TIME	MEM	MEM	MEM	MEM	65	75	110	TIME	83.3
5	48	81	59	66	53	73	46	59	63	43	59.1
6	71	52	51	90	76	73	75	59	76	58	68.1
7	91	MEM	75	84	93	95	71	72	104	68	83.7
8	MEM	97	88	115	104	94	97	97	107	MEM	99.9
9	59	51	39	58	82	49	46	MEM	50	60	54.9
10	48	MEM	MEM	MEM	43	MEM	49	MEM	66	45	50.2
Average	65.3	67.7	62.4	80.8	75.6	74.0	67.0	71.3	84.9	59.0	71.1

Table D.31: Action execution steps in derived plans using TF[COLIN] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	727	MEM	MEM	11	331	27	17	MEM	113	113	191.3
2	32	7	27	42	11	12	415	16	MEM	114	75.1
3	18	1119	8	148	105	200	16	1968	20	72	367.4
4	MEM	TIME	MEM	MEM	MEM	MEM	34	38	50	TIME	40.4
5	4	79	4	23	9	87	7	15	45	19	29.1
6	34	4	11	22	41	6	12	8	11	640	78.8
7	12	MEM	6	51	56	21	8	5	27	263	49.8
8	MEM	1563	764	569	159	45	100	13	156	MEM	421.1
9	3	10	2	418	11	578	8	MEM	4	37	118.9
10	37	MEM	MEM	MEM	5	MEM	38	MEM	90	117	57.4
Average	108.4	463.7	117.5	160.6	80.7	122.0	65.3	294.5	57.2	171.8	150.8

Table D.32: Computation time to derive plans using TF[COLIN] for Blocks World problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	16.01	MEM	MEM	0.19	13.86	0.38	0.24	MEM	2.78	1.78	5.03
2	0.98	0.13	0.64	1.90	0.21	0.22	8.97	0.44	MEM	2.91	1.82
3	0.17	10.12	0.09	2.42	2.61	2.24	0.21	36.28	0.26	0.87	5.53
4	MEM	TIME	MEM	MEM	MEM	MEM	0.81	0.53	1.37	TIME	0.90
5	0.08	5.03	0.09	1.46	0.41	5.50	0.16	0.73	2.46	0.43	1.64
6	1.05	0.03	0.25	0.75	0.77	0.06	0.31	0.09	0.35	8.88	1.25
7	0.08	MEM	0.07	0.75	1.05	0.48	0.11	0.04	0.34	2.60	0.61
8	MEM	13.85	21.01	15.34	2.93	0.49	1.16	0.11	1.52	MEM	7.05
9	0.03	0.18	0.02	11.67	0.20	4.94	0.08	MEM	0.06	0.36	1.95
10	0.86	MEM	MEM	MEM	0.25	MEM	0.88	MEM	4.86	1.55	1.68
Average	2.41	4.89	3.17	4.31	2.48	1.79	1.29	5.46	1.56	2.42	2.83

Table D.33: Memory required to derive plans using TF[COLIN] for Blocks World problems.

Appendix E

Detailed Zenotravel Results

This appendix presents the full results for the Zenotravel domain. Each page of results presents a single metric for the problems solved by each of the tools using a single underlying planning algorithm. The results are ordered from top to bottom of the page: Planning Alone, Coalition Formation then Planning, Relaxed Plan Coalition Augmentation, Task Fusion. Each table has a heatmap overlay, ranging from better results in green (lower values for all metrics) to worse results in red (higher values for all metrics), with specific values presented in Table E.1. The overlay for time results ranges from 0 seconds (green) to 3600 seconds (red), and the overlay for the memory results ranges from 0 GB (green) to 48 GB (red). The range for action executions is from 430 (green) to 630 (red) and for makespan is from 800 (green) to 3700 (red).

Metric	**Green**	**Red**
Makespan	800	3700
Action Executions	430	630
Time	0 seconds	3600 seconds
Memory	0 GB	48 GB

Table E.1: Heatmap ranges for each metric in the Zenotravel domain

E.1 Detailed Results for CFP[COLIN] Applied to the Zenotravel Problems

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	1110	1718	1507	841	1252	EHC	1896	1063	1192	1450	1336.6
2	EHC	1990	1113	EHC	1349	1106	1795	906	1244	EHC	1357.6
3	1968	2100	988	917	1705	EHC	1797	1497	1993	1630	1621.7
4	1983	1805	1644	2074	1348	2249	1629	1682	1609	2002	1802.5
5	EHC	2032	3673	1843	EHC	1897	EHC	1886	EHC	EHC	2266.2
6	1253	2440	EHC	1368	2573	1415	1825	1660	2193	2029	1861.8
7	944	1557	1793	1720	EHC	1493	1557	1350	1240	1963	1513.0
8	1348	2122	1056	964	2161	1040	1701	1136	1978	1406	1491.2
9	1301	1614	1376	1248	EHC	1283	1627	1269	1333	1561	1401.3
10	982	1351	1237	1012	1247	1069	1227	1465	1141	1650	1238.1
Average	1361.1	1872.9	1598.6	1331.9	1662.1	1444.0	1672.7	1391.4	1547.0	1711.4	1560.5

Table E.2: Temporal makespan of derived plans using CFP[COLIN] for Zenotravel problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	451	462	468	454	462	EHC	474	462	456	455	460.4
2	EHC	467	452	EHC	455	449	462	444	455	EHC	454.9
3	475	481	480	481	474	EHC	480	481	476	481	478.8
4	502	513	508	510	500	520	512	511	507	515	509.8
5	EHC	534	532	522	EHC	530	EHC	528	EHC	EHC	529.2
6	603	614	EHC	606	617	612	605	632	607	609	611.7
7	539	543	550	547	EHC	536	547	545	548	543	544.2
8	517	530	521	531	532	524	535	519	530	509	524.8
9	499	497	507	501	EHC	510	501	513	504	492	502.7
10	440	441	442	438	445	448	439	452	441	430	441.6
Average	503.3	508.2	495.6	510.0	497.9	516.1	506.1	508.7	502.7	504.3	505.4

Table E.3: Action execution steps in derived plans using CFP[COLIN] for Zenotravel problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	255	434	404	261	388	EHC	488	346	374	459	378.9
2	EHC	368	385	EHC	399	406	437	350	329	EHC	382.1
3	388	446	380	332	396	EHC	462	375	408	386	396.8
4	440	457	520	315	325	475	457	526	433	498	444.6
5	EHC	548	503	458	EHC	343	EHC	493	EHC	EHC	468.9
6	858	1226	EHC	873	1224	996	1063	1139	1247	928	1061.5
7	431	612	585	542	EHC	545	646	548	595	602	567.1
8	406	645	437	406	447	428	688	406	518	546	492.6
9	482	550	501	419	EHC	500	547	550	450	506	500.7
10	278	331	276	228	265	288	345	335	260	329	293.5
Average	442.3	561.7	443.4	426.0	492.0	497.6	570.2	506.7	512.6	531.6	499.7

Table E.4: Computation time to derive plans using CFP[COLIN] for Zenotravel problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	0.18	0.18	0.18	0.18	0.18	EHC	0.18	0.18	0.18	0.18	0.18
2	EHC	0.14	0.14	EHC	0.14	0.14	0.14	0.14	0.14	EHC	0.14
3	0.14	0.14	0.14	0.14	0.14	EHC	0.14	0.14	0.14	0.14	0.14
4	0.15	0.14	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14
5	EHC	0.15	0.15	0.15	EHC	0.15	EHC	0.15	EHC	EHC	0.15
6	0.18	0.18	EHC	0.18	0.18	0.18	0.18	0.18	0.18	0.18	0.18
7	0.17	0.17	0.17	0.17	EHC	0.17	0.17	0.17	0.17	0.17	0.17
8	0.18	0.18	0.18	0.18	0.18	0.18	0.18	0.18	0.18	0.18	0.18
9	0.15	0.15	0.15	0.15	EHC	0.15	0.15	0.15	0.15	0.15	0.15
10	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14
Average	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.16

Table E.5: Memory required to derive plans using CFP[COLIN] for Zenotravel problems.

E.2 Detailed Results for TF[COLIN] Applied to the Zenotravel Problems

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	824	1433	1878	889	1759	EHC	1518	1252	1417	2436	1489.6
2	1176	1880	EHC	1192	EHC	1751	EHC	1721	EHC	EHC	1544.0
3	1957	2035	960	929	1900	EHC	1914	1645	2124	1805	1696.6
4	1167	1588	1356	1360	1362	1865	1527	1957	1605	1838	1562.5
5	1276	1697	2190	1371	1306	880	1557	1431	1499	2202	1540.9
6	1183	EHC	EHC	EHC	2089	1322	2014	1218	EHC	2316	1690.3
7	944	1353	1314	1244	1488	942	1878	1169	1270	1646	1324.8
8	989	2056	981	978	1710	EHC	1568	1625	EHC	1614	1440.1
9	1090	2040	1374	1124	1224	1283	1233	1269	1220	1477	1333.4
10	1018	EHC	1087	1086	1285	1061	1420	EHC	1155	1389	1187.6
Average	1162.4	1760.3	1392.5	1130.3	1569.2	1300.6	1625.4	1476.3	1470.0	1858.1	1472.6

Table E.6: Temporal makespan of derived plans using TF[COLIN] for Zenotravel problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	452	459	470	458	459	EHC	463	457	451	466	459.4
2	458	471	EHC	444	EHC	450	EHC	457	EHC	EHC	456.0
3	480	485	488	495	488	EHC	482	483	485	485	485.7
4	496	512	504	512	497	521	504	512	506	523	508.7
5	525	536	521	520	531	518	520	528	523	531	525.3
6	603	EHC	EHC	EHC	622	608	599	606	EHC	604	607.0
7	537	541	545	548	538	542	548	548	547	541	543.5
8	521	541	523	533	533	EHC	535	524	EHC	517	528.4
9	493	492	501	504	502	505	497	513	504	493	500.4
10	439	EHC	437	440	446	445	438	EHC	435	436	439.5
Average	500.4	504.6	498.6	494.9	512.9	512.7	509.6	514.2	493.0	510.7	505.3

Table E.7: Action execution steps in derived plans using TF[COLIN] for Zenotravel problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	208	292	301	250	298	EHC	285	257	266	409	285.1
2	236	310	EHC	115	EHC	163	EHC	283	EHC	EHC	221.3
3	174	246	134	109	182	EHC	232	258	190	431	217.2
4	267	517	283	338	303	363	420	237	309	588	362.4
5	231	404	389	202	306	202	452	384	260	416	324.5
6	491	EHC	EHC	EHC	539	646	631	494	EHC	786	597.6
7	230	488	413	314	254	279	579	437	325	439	375.8
8	228	530	346	271	318	EHC	490	447	EHC	482	388.9
9	207	376	329	177	172	214	368	244	252	383	272.2
10	234	EHC	161	122	187	135	208	EHC	175	256	184.6
Average	250.5	395.4	294.3	210.7	284.3	286.1	407.0	337.8	253.9	465.4	319.4

Table E.8: Computation time to derive plans using TF[COLIN] for Zenotravel problems.

Mission \ Grand Coalition	1	2	3	4	5	6	7	8	9	10	Average
1	0.18	0.18	0.18	0.18	0.18	EHC	0.18	0.18	0.18	0.18	0.18
2	0.14	0.14	EHC	0.14	EHC	0.14	EHC	0.14	EHC	EHC	0.14
3	0.14	0.14	0.14	0.14	0.14	EHC	0.14	0.14	0.14	0.14	0.14
4	0.13	0.14	0.14	0.14	0.13	0.14	0.13	0.14	0.13	0.14	0.14
5	0.15	0.19	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15
6	0.18	EHC	EHC	EHC	0.18	0.18	0.18	0.18	EHC	0.31	0.20
7	0.17	0.17	0.17	0.17	0.22	0.24	0.17	0.17	0.23	0.17	0.19
8	0.17	0.69	0.17	0.17	0.18	EHC	0.17	0.18	EHC	0.17	0.24
9	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15
10	0.14	EHC	0.14	0.14	0.14	0.14	0.14	EHC	0.14	0.14	0.14
Average	0.16	0.23	0.16	0.15	0.16	0.16	0.16	0.16	0.16	0.17	0.17

Table E.9: Memory required to derive plans using TF[COLIN] for Zenotravel problems.

BIBLIOGRAPHY

Alami, R., Robert, F., Ingrand, F., and Suzuki, S. (1995). Multi-robot cooperation through incremental plan-merging. In *Proceedings of the 1995 IEEE International Conference on Robotics and Automation*, volume 3, pages 2573–2579.

Allen, J. F. (1984). Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123 – 154.

Allouche, M. K. and Boukhtouta, A. (2010). Multi-agent coordination by temporal plan fusion: Application to combat search and rescue. *Information Fusion*, 11(3):220–232.

Bäckström, C. (1993). Finding least constrained plans and optimal parallel executions is harder than we thought. In *Current Trends in AI Planning: Second European Workshop on Planning, Frontiers in AI and Applications*, pages 46–59.

Bahar, R., Frohm, E., Gaona, C., Hachtel, G., Macii, E., Pardo, A., and Somenzi, F. (1997). Algebraic decision diagrams and their applications. *Formal Methods in System Design*, 10:171–206.

Barley, M., Franco, S., and Riddle, P. (2014). Overcoming the utility problem in heuristic generation: Why time matters. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling*, pages 38–46.

Barrientos, A., Colorado, J., Cerro, J. d., Martinez, A., Rossi, C., Sanz, D., and Valente, J. (2011). Aerial remote sensing in agriculture: A practical approach to area coverage and path planning for fleets of mini aerial robots. *Journal of Field Robotics*, 28(5):667–689.

Barto, A. G., Bradtke, S. J., and Singh, S. P. (1991). Real-time learning and control using asynchronous dynamic programming. Technical Report 91-57, University of Massachusetts.

Benton, J., Coles, A. J., and Coles, A. (2012). Temporal planning with preferences and time-dependent continuous costs. In *International Conference on Automated Planning and Scheduling*, volume 77, pages 2–10.

Bibaï, J., Savéant, P., Schoenauer, M., and Vidal, V. (2010a). An evolutionary metaheuristic based on state decomposition for domain-independent satisficing planning. In *20th International Conference on Automated Planning and Scheduling*, pages 18–25.

Bibaï, J., Savéant, P., Schoenauer, M., and Vidal, V. (2010b). On the benefit of sub-optimality within the divide-and-evolve scheme. In Cowling, P. and Merz, P., editors, *Evolutionary Computation in Combinatorial Optimization*, volume 6022 of *Lecture Notes in Computer Science*, pages 23–34. Springer, Berlin Heidelberg.

Blum, A. and Langford, J. C. (2000). Probabilistic planning in the graphplan framework. In Biundo, S. and Fox, M., editors, *Recent Advances in AI Planning*, volume 1809 of *Lecture Notes in Computer Science*, pages 319–332. Springer Berlin Heidelberg.

Blum, A. L. and Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, 90(12):281 – 300.

Bonet, B. and Geffner, H. (2003). Labeled RTDP: Improving the convergence of real-time dynamic programming. In *International Conference on Automated Planning and Scheduling*, volume 3, pages 12–21.

Bonet, B., Loerincs, G., and Geffner, H. (1997). A robust and fast action selection mechanism for planning. In *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Conference on Innovative Applications of Artificial Intelligence*, pages 714–719.

Boutilier, C., Dearden, R., and Goldszmidt, M. (1995). Exploiting structure in policy construction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, volume 2, pages 1104–1111.

Brafman, R., Gorohovski, A., and Shani, G. (2014). A contingent planning-based POMDP replanner. *Proceedings of the 1st Workshop on Models and Paradigms for Planning under Uncertainty: A Broad Perspective*, pages 44–48.

Brafman, R. I. (1997). A heuristic variable grid solution method for POMDPs. In *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Conference on Innovative Applications of Artificial Intelligence*, pages 727–733.

Brafman, R. I. and Domshlak, C. (2006). Factored planning: How, when, and when not. In *Proceedings of the 21st National Conference on Artificial Intelligence*, pages 809–814.

Brafman, R. I. and Domshlak, C. (2008). From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the 18th International Conference on Autonomous Planning and Scheduling*, pages 28–35.

Brenner, M. (2003). A multiagent planning language. In *Proceedings of the Workshop on PDDL at the International Conference on Automated Planning and Scheduling*, volume 3, pages 33–38.

Brutschy, A., Pini, G., Pinciroli, C., Birattari, M., and Dorigo, M. (2014). Self-organized task allocation to sequentially interdependent tasks in swarm robotics. *Autonomous Agents and Multi-Agent Systems*, 28(1):101–125.

Bryce, D., Gao, S., Musliner, D., and Goldman, R. (2015). SMT-based nonlinear PDDL+ planning. In *Proceedings of the 29th Conference on Artificial Intelligence*, pages 3247–3253.

Chaimowicz, L., Cowley, A., Sabella, V., and Taylor, C. (2003). ROCI: a distributed framework for multi-robot perception and control. In *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 266–271.

Chen, Y., Wah, B. W., and Hsu, C.-W. (2006). Temporal planning using subgoal partitioning and resolution in SGPlan. *Journal of Artificial Intelligence Research*, 26(1):323–369.

Cimatti, A., Micheli, A., and Roveri, M. (2015). Strong temporal planning with uncontrollable durations: A state-space approach. *Association for the Advancement of Artificial Intelligence*, pages 3254–3260.

Claes, D., Robbel, P., Oliehoek, F. A., Tuyls, K., Hennes, D., and van der Hoek, W. (2015). Effective approximations for multi-robot coordination in spatially distributed tasks. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 881–890.

Coles, A., Coles, A., Fox, M., and Long, D. (2009a). Temporal planning in domains with linear processes. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence*, pages 1671–1676.

Coles, A., Coles, A., Fox, M., and Long, D. (2011). POPF2: A forward-chaining partial order planner. *The 2011 International Planning Competition*, pages 65–70.

Coles, A., Coles, A., Fox, M., and Long, D. (2012a). COLIN: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research*, 44(1):1–96.

Coles, A., Coles, A., Olaya, A. G., Jiménez, S., López, C. L., Sanner, S., and Yoon, S. (2012b). A survey of the seventh international planning competition. *AI Magazine*, 33(1):83–88.

Coles, A., Fox, M., Halsey, K., Long, D., and Smith, A. (2009b). Managing concurrency in temporal planning using planner-scheduler interaction. *Artificial Intelligence*, 173(1):1–44.

Coles, A., Fox, M., Long, D., and Smith, A. (2008). Planning with problems requiring temporal coordination. In *Proceedings of the 23rd National Conference on Artificial Intelligence*, volume 2, pages 892–897.

Coles, A. J., Coles, A., Fox, M., and Long, D. (2010). Forward-chaining partial-order planning. In *Proceedings of the 2010 International Conference on Automated Planning and Scheduling*, pages 42–49.

Cox, J. S. and Durfee, E. H. (2005). An efficient algorithm for multiagent plan coordination. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 828–835.

Cox, J. S., Durfee, E. H., and Bartold, T. (2005). A distributed framework for solving the multiagent plan coordination problem. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 821–827.

Crosby, M., Jonsson, A., and Rovatsos, M. (2014). A single-agent approach to multiagent planning. In *Proceedings of the 21st European Conference on Artificial Intelligence*, pages 237–242.

Cushing, W., Kambhampati, S., Mausam, and Weld, D. S. (2007a). When is temporal planning really temporal? In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, pages 1852–1859.

Cushing, W., Weld, D. S., Kambhampati, S., Mausam, and Talamadupula, K. (2007b). Evaluating temporal planning domains. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling*, pages 105–112.

Dahl, T., Matarić, M. J., and Sukhatme, G. (2003). Multi-robot task-allocation through vacancy chains. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, volume 2, pages 2293–2298.

Dang, V. D. and Jennings, N. R. (2004). Generating coalition structures with finite bound from the optimal guarantees. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, volume 2, pages 564–571.

de Weerdt, M., Bos, A., Tonino, H., and Witteveen, C. (2003). A resource logic for multi-agent plan merging. *Annals of Mathematics and Artificial Intelligence*, 37(1-2):93–130.

Dean, T., Kaelbling, L. P., Kirman, J., and Nicholson, A. (1993). Planning with deadlines in stochastic domains. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 574–579.

Decker, K. S. and Lesser, V. R. (1992). Generalizing the partial global planning algorithm. *International Journal of Intelligent and Cooperative Information Systems*, 1(2):319–346.

Dias, M., Ghanem, B., and Stentz, A. (2005). Improving cost estimation in market-based coordination of a distributed sensing task. In *Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3972–3977.

Dias, M. B. (2004). *Traderbots: A new paradigm for robust and efficient multirobot coordination in dynamic environments*. PhD thesis, Carnegie Mellon University.

Dimopoulos, Y., Hashmi, M. A., and Moraitis, P. (2012). µ-SATPLAN: Multi-agent planning as satisfiability. *Knowledge-Based Systems*, 29:54–62.

Do, M. and Kambhampati, S. (2001). Sapa: A domain-independent heuristic metric temporal planner. In *Proceedings of the 6th European Conference on Planning*, pages 57–68.

Do, M. B. and Kambhampati, S. (2003). Sapa: A multi-objective metric temporal planner. *Journal of Artificial Intelligence Research*, 20(1):155–194.

Doherty, P., Gustafsson, J., and Karlsson, L. (1998). TAL: Temporal action logics language specification and tutorial. *Computer and Information Science*, 3(015).

Dréo, J., Savéant, P., Schoenauer, M., and Vidal, V. (2011). Divide-and-Evolve: the marriage of Descartes and Darwin. *Proceedings of the 7th International Planning Competition*, pages 29–30.

Durfee, E. H. and Lesser, V. R. (1991). Partial global planning: a coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man and Cybernetics*, 21(5):1167–1183.

Ephrati, E. and Rosenschein, J. S. (1993). Multi-agent planning as the process of merging distributed sub-plans. In *Proceedings of the 12th International Workshop on Distributed Artificial Intelligence*, pages 115–129.

Ephrati, E. and Rosenschein, J. S. (1994). Divide and conquer in multi-agent planning. In *Proceedings of the 12th National Conference on Artificial Intelligence*, volume 1, pages 375–380.

Eyerich, P., Keller, T., Aldinger, J., and Dornhege, C. (2014). Preferring preferred operators in temporal fast downward. In *International Planning Competition*, pages 121–126.

Eyerich, P., Mattmüller, R., and Röger, G. (2012). Using the context-enhanced additive heuristic for temporal and numeric planning. In Prassler, E., Zöllner, M., Bischoff, R., Burgard, W., Haschke, R., Hägele, M., Lawitzky, G., Nebel, B., Plöger, P., and Reiser, U., editors, *Towards Service Robots for Everyday Environments*, volume 76 of *Springer Tracts in Advanced Robotics*, pages 49–64. Springer Berlin Heidelberg.

Fikes, R. E. and Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(34):189–208.

Flushing, E. F., Gambardella, L. M., and Di Caro, G. A. (2016). On decentralized coordination for spatial task allocation and scheduling in heterogeneous teams. In *Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems*, pages 988–996.

Foulser, D. E., Li, M., and Yang, Q. (1992). Theory and algorithms for plan merging. *Artificial Intelligence*, 57:143–181.

Fox, M., Gerevini, A., Long, D., and Serina, I. (2006a). Plan stability: Replanning versus plan repair. In *Proceedings of the 6th International Conference on Automated Planning and Scheduling*, pages 212–221.

Fox, M., Howey, R., and Long, D. (2006b). Exploration of the robustness of plans. In *Proceedings of the 21st National Conference on Artificial Intelligence*, pages 834–839.

Fox, M. and Long, D. (2003). PDDL 2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20(1):61–124.

Fox, M. and Long, D. (2006). Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research*, 27(1):235–297.

Gerevini, A., Saetti, A., and Serina, I. (2003). Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research*, 20(1):239–290.

Gerevini, A. and Serina, I. (2002). LPG: A planner based on local search for planning graphs with action costs. In *Proceedings of the 6th International Conference on AI Planning Systems*, volume 2, pages 13–22.

Gerkey, B. P. and Matarić, M. J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research*, 23(9):939–954.

Graham, R. L. (1966). Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581.

Haghighi, M. (2014). Market-based resource allocation for energy-efficient execution of multiple concurrent applications in wireless sensor networks. In Park, J. J., Adeli, H., Park, N., and Woungang, I., editors, *Mobile, Ubiquitous, and Intelligent Computing*, volume 274 of *Lecture Notes in Electrical Engineering*, pages 173–178. Springer Berlin Heidelberg.

Halsey, K., Long, D., and Fox, M. (2004). CRIKEY - a temporal planner looking at the integration of scheduling and planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling*, pages 46–52.

Haslum, P. and Geffner, H. (2001). Heuristic planning with time and resources. In *Proceedings of the 6th European Conference on Planning*, pages 107–112.

Hauskrecht, M. (2000). Value-function approximations for partially observable markov decision processes. *Journal of Artificial Intelligence Research*, 13(1):33–94.

Helmert, M. (2006). The fast downward planning system. *Journal of Artificial Intelligence Research*, 26(1):191–246.

Hoey, J., Poupart, P., von Bertoldi, A., Craig, T., Boutilier, C., and Mihailidis, A. (2010). Automated handwashing assistance for persons with dementia using video and a partially observable markov decision process. *Computer Vision and Image Understanding*, 114(5):503–519.

Hoffmann, J. and Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14(1):253–302.

Howey, R., Long, D., and Fox, M. (2004). VAL: automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *IEEE International Conference on Tools with Artificial Intelligence*, pages 294–301.

Hsiao, K., Kaelbling, L., and Lozano-Perez, T. (2007). Grasping POMDPs. In *Proceedings of the 2007 IEEE International Conference on Robotics and Automation*, pages 4685–4692.

Hsu, C.-W., Wah, B. W., Huang, R., and Chen, Y. (2006a). Handling soft constraints and goals preferences in sgplan. In *Proceedings of the 2006 International Conference on Automated Planning and Scheduling*, pages 54–57.

Hsu, C.-W., Wah, B. W., Huang, R., and Chen, Y. (2006b). New features in SGPlan for handling preferences and constraints in PDDL 3.0. In *Proceedings of the 5th International Planning Competition*, pages 39–42.

Hsu, C.-W., Wah, B. W., Huang, R., and Chen, Y. (2007). Constraint partitioning for solving planning problems with trajectory constraints and goal preferences. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, pages 1924–1929.

Hsu, D., Lee, W. S., and Rong, N. (2008). A point-based pomdp planner for target tracking. In *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*, pages 2644–2650.

Jiménez, S., Jonsson, A., and Palacios, H. (2015). Temporal planning with required concurrency using classical planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling*, pages 129–137.

Jonsson, A. and Rovatsos, M. (2011). Scaling up multiagent planning: A best-response approach. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling*, pages 114–121.

Joslin, D. and Pollack, M. E. (1994). Least-cost flaw repair: A plan refinement strategy for partial-order planning. In *Proceedings of the 12th National Conference on Artificial Intelligence*, volume 2, pages 1004–1009.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(12):99 – 134.

Kautz, H. and Selman, B. (1992). Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 359–363.

Kautz, H. and Selman, B. (2006). SATPLAN04: Planning as satisfiability. In *Booklet of the 2006 International Planning Competition*, pages 46–47.

Keller, T. and Eyerich, P. (2012). PROST: Probabilistic planning based on UCT. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling*, pages 119–127.

Keller, T. and Helmert, M. (2013). Trial-based heuristic tree search for finite horizon MDPs. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling*, pages 135–143.

Kim, M.-H., Baik, H., and Lee, S. (2014). Response threshold model based uav search planning and task allocation. *Journal of Intelligent and Robotic Systems*, 75(3–4):625–640.

Kissmann, P. and Edelkamp, S. (2011). Improving cost-optimal domain-independent symbolic planning. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, pages 992–997.

Kocsis, L. and Szepesvri, C. (2006). Bandit based Monte-Carlo planning. In Fürnkranz, J., Scheffer, T., and Spiliopoulou, M., editors, *Machine Learning: European Conference on Machine Learning 2006*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer Berlin Heidelberg.

Koenig, S. (1992). Optimal probabilistic and decision-theoretic planning using markovian decision theory. Master's thesis.

Koenig, S. (1994). Risk-sensitive planning with probabilistic decision graphs. In *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, pages 363–374.

Koes, M., Nourbakhsh, I., and Sycara, K. (2005). Heterogeneous multirobot coordination with spatial and temporal constraints. In *Proceedings of the 20th National Conference on Artificial Intelligence*, volume 3, pages 1292–1297.

Kolobov, A., Dai, P., Mausam, and Weld, D. S. (2012a). Reverse iterative deepening for finite-horizon MDPs with large branching factors. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling*, pages 146–154.

Kolobov, A., Mausam, and Weld, D. (2012b). LRTDP versus UCT for online probabilistic planning. In *Proceedings of AAAI Conference on Artificial Intelligence*, pages 1786–1792.

Korsah, G. A., Stentz, A., and Dias, M. B. (2013). A comprehensive taxonomy for multi-robot task allocation. *International Journal of Robotics Research*, 32(12):1495–1512.

Kovacs, D. L. (2012). A multi-agent extension of PDDL 3.1. In *Proceedings of the 3rd Workshop on the International Planning Competition at the International Conference on Automated Planning and Scheduling*, pages 19–27.

Kurniawati, H., Du, Y., Hsu, D., and Lee, W. S. (2011). Motion planning under uncertainty for robotic tasks with long time horizons. In Pradalier, C., Siegwart, R., and Hirzinger, G., editors, *Robotics Research*, volume 70 of *Springer Tracts in Advanced Robotics*, pages 151–168. Springer Berlin Heidelberg.

Kurniawati, H., Hsu, D., and Lee, W. S. (2008). SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proceedings of Robotics: Science and Systems*, volume 4, pages 65–72.

Kvarnström, J. (2011). Planning for loosely coupled agents using partial order forward-chaining. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling*, pages 138–145.

Kvarnström, J. and Doherty, P. (2000). TALplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence*, 30(1-4):119–169.

Kvarnström, J., Doherty, P., and Haslum, P. (2000). Extending TALplanner with concurrency and resources. In *Proceedings of the 14th European Conference on Artificial Intelligence*, pages 501–505.

Laborie, P. and Ghallab, M. (1995). IxTeT: An integrated approach for plan generation and scheduling. In *Proceedings of the 1995 INRIA/IEEE Symposium on Emerging Technologies and Factory Automation*, volume 1, pages 485–495.

Lemai, S. and Ingrand, F. (2004). Interleaving temporal planning and execution in robotics domains. In *Proceedings of the 19th National Conference on Artifical Intelligence*, pages 617–622.

Liemhetcharat, S. and Veloso, M. (2014). Weighted synergy graphs for effective team formation with heterogeneous ad hoc agents. *Artificial Intelligence*, 208:41 – 65.

Littman, M. L., Cassandra, A. R., and Kaelbling, L. P. (1995a). Learning policies for partially observable environments: Scaling up. In *Proceedings of the 12th International Conference on Machine Learning*, pages 362–370.

Littman, M. L., Dean, T. L., and Kaelbling, L. P. (1995b). On the complexity of solving Markov decision problems. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, pages 394–402.

Liu, H., Zhang, P., Hu, B., and Moore, P. (2015). A novel approach to task assignment in a cooperative multi-agent design system. *Applied Intelligence*, 43(1):162–175.

Liu, L. and Shell, D. A. (2013). Optimal market-based multi-robot task allocation via strategic pricing. In *Proceedings of the International Conference on Multi-Agent Systems*, pages 33–40.

Long, D. and Fox, M. (2003a). The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20(1):1–59.

Long, D. and Fox, M. (2003b). Exploiting a graphplan framework in temporal planning. In *Proceedings of the 2003 International Conference on Automated Planning and Scheduling*, pages 52–61.

Luis, N. and Borrajo, D. (2014). Plan merging by reuse for multi-agent planning. In *Proceedings of the 2nd International Conference on Automated Planning and Scheduling Distributed and Multi-Agent Planning Workshop*, pages 38–44.

Ma, H. and Koenig, S. (2016). Optimal target assignment and path finding for teams of agents. In *Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems*, pages 1144–1152.

Mezei, I., Lukic, M., Malbasa, V., and Stojmenovic, I. (2013). Auctions and iMesh based task assignment in wireless sensor and actuator networks. *Computer Communications*, 36(9):979–987.

Moon, S., Oh, E., and Shim, D. (2013). An integral framework of task assignment and path planning for multiple unmanned aerial vehicles in dynamic environments. *Journal of Intelligent and Robotic Systems*, 70:303–313.

Muise, C. J., McIlraith, S. A., and Beck, J. C. (2012). Improved non-deterministic planning by exploiting state relevance. In *Proceedings of the 22nd International Conference on Autonomous Planning and Scheduling*, pages 172–180.

Nissim, R. and Brafman, R. (2014). Distributed heuristic forward search for multi-agent planning. *Journal of Artificial Intelligence Research*, 51(1):293–332.

Nissim, R., Brafman, R. I., and Domshlak, C. (2010). A general, fully distributed multi-agent planning algorithm. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, volume 1, pages 1323–1330.

Parker, L. (1998). ALLIANCE: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240.

Penberthy, J. S. and Weld, D. S. (1992). UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, pages 103–114.

Penberthy, J. S. and Weld, D. S. (1994). Temporal planning with continuous change. In *Proceedings of the 1994 AAAI Conference on Artificial Intelligence*, pages 1010–1015.

Pendharkar, P. C. (2015). An ant colony optimization heuristic for constrained task allocation problem. *Journal of Computational Science*, 7:37–47.

Peot, M. A. and Smith, D. E. (1993). Threat-removal strategies for partial-order planning. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 492–499.

Péret, L. and Garcia, F. (2004). On-line search for solving markov decision processes via heuristic sampling. In *Proceedings of the 16th European Conference on Artificial Intelligence*, pages 530–534.

Pineau, J., Gordon, G., and Thrun, S. (2003a). Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 1025–1030.

Pineau, J., Gordon, G., and Thrun, S. (2003b). Policy-contingent abstraction for robust robot control. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence*, pages 477–484.

Pineau, J. and Gordon, G. J. (2007). POMDP planning for robust robot control. In Thrun, S., Brooks, R., and Durrant-Whyte, H., editors, *Robotics Research*, volume 28 of *Springer Tracts in Advanced Robotics*, pages 69–82. Springer Berlin Heidelberg.

Pineau, J., Montemerlo, M., Pollack, M., Roy, N., and Thrun, S. (2003c). Towards robotic assistants in nursing homes: Challenges and results. *Robotics and Autonomous Systems*, 42(34):271 – 281.

Ponda, S., Johnson, L., and How, J. (2012). Distributed chance-constrained task allocation for autonomous multi-agent teams. In *American Control Conference*, pages 4528–4533.

Poon, K. M. (2001). An intelligent and unified framework for multiple robot and human coalition formation. Master's thesis, The Hong Kong University of Science and Technology.

Rahwan, T., Ramchurn, S. D., Jennings, N. R., and Giovannucci, A. (2009). An anytime algorithm for optimal coalition structure generation. *Journal of Artificial Intelligence Research*, 34:521–567.

Ramchurn, S. D., Polukarov, M., Farinelli, A., Truong, C., and Jennings, N. R. (2010). Coalition formation with spatial and temporal constraints. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, volume 3, pages 1181–1188.

Rankooh, M. F. and Ghassem-Sani, G. (2015). ITSAT: an efficient sat-based temporal planner. *Journal of Artificial Intelligence Research*, 53:541–632.

Ren, Z., Feng, Z., and Wang, X. (2008). An efficient ant colony optimization approach to agent coalition formation problem. In *Proceedings of the 7th World Congress on Intelligent Control and Automation*, pages 7879–7882.

Sacerdoti, E. D. (1975). The nonlinear nature of plans. In *Proceedings of the 4th International Joint Conference on Artificial Intelligence*, volume 1, pages 206–214.

Sandholm, T., Larson, K., Andersson, M., Shehory, O., and Tohmé, F. (1999). Coalition structure generation with worst case guarantees. *Artificial Intelligence*, 111(1-2):209 – 238.

Sapena, O., Onaindía, E., and no, A. T. (2013). Forward-chaining planning with a flexible least-commitment strategy. In *Proceedings of the 16th International Conference of the Catalan Association for Artificial Intelligence*, volume 256, pages 41–50.

Sapena, O., Onaindıa, E., and Torreno, A. (2014). Combining heuristics to accelerate forward partial-order planning. *Proceedings of the 9th Workshop on Constraint Satisfaction Techniques for Planning and Scheduling*, pages 25–34.

Schubert, L. and Gerevini, A. (1995). Accelerating partial order planners by improving plan and goal choices. In *Proceedings of the 7th International Conference on Tools with Artificial Intelligence*, pages 442–450.

Sen, S. D. and Adams, J. A. (2013). A decision network based framework for multiagent coalition formation. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, pages 55–62.

Sen, S. D. and Adams, J. A. (2015). An influence diagram based multi-criteria decision making framework for multirobot coalition formation. *Autonomous Agents and Multi-Agent Systems*, 29(6):1061–1090.

Service, T. C. (2010). Coalition formation algorithm taxonomy. Technical Report HMT-10-03, Vanderbilt University.

Service, T. C. and Adams, J. A. (2011). Coalition formation for task allocation: Theory and algorithms. *Journal of Autonomous Agents and Multi-Agent Systems*, 22(2):225–248.

Service, T. C., Sen, S. D., and Adams, J. A. (2014). A simultaneous descending auction for task allocation. In *Proceedings of the 2014 IEEE International Conference on Systems, Man and Cybernetics*, pages 379–384. IEEE.

Shani, G., Brafman, R. I., and Shimony, S. E. (2007). Forward search value iteration for POMDPs. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, pages 2619–2624.

Shehory, O. and Kraus, S. (1998). Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101:165 – 200.

Shiroma, P. M. and Campos, M. F. M. (2009). Comutar: A framework for multi-robot coordination and task allocation. In *2009 IEEE/RSJ Internation Conference on Intelligent Robots and Systems*, pages 4817–4824.

Silver, D. and Veness, J. (2010). Monte-carlo planning in large pomdps. In *Advances in Neural Information Processing Systems*, volume 23, pages 2164–2172.

Sim, H. S., Kim, K.-E., Kim, J. H., Chang, D.-S., and Koo, M.-W. (2008). Symbolic heuristic search value iteration for factored POMDPs. In *Proceedings of the 23rd National Conference on Artificial Intelligence*, volume 2, pages 1088–1093.

Smith, D. E. and Weld, D. S. (1999). Temporal planning with mutual exclusion reasoning. In *Proceedings of the 16th International Joint Conference on Artifical Intelligence*, volume 1, pages 326–333.

Smith, T. and Simmons, R. (2012). Point-Based POMDP Algorithms: Improved Analysis and Implementation. *ArXiv e-prints*.

Somani, A., Ye, N., Hsu, D., and Lee, W. S. (2013). Despot: Online POMDP planning with regularization. In *Advances In Neural Information Processing Systems*, pages 1772–1780.

Spaan, M. and Spaan, M. (2004). A point-based pomdp algorithm for robot planning. In *2004 IEEE International Conference on Robotics and Automation*, volume 3, pages 2399–2404.

Stentz, A. and Dias, M. B. (1999). A free market architecture for coordinating multiple robots. Technical Report CMU-RI-TR-99-42, Carnegie Mellon University.

Sujit, P., Manathara, J., Ghose, D., and de Sousa, J. (2014). Decentralized multi-uav coalition formation with limited communication ranges. In Valavanis, K. P. and Vachtsevanos, G. J., editors, *Handbook of Unmanned Aerial Vehicles*, pages 2021–2048. Springer Netherlands.

Tambe, M. (1997). Towards flexible teamwork. *Journal of Artificial Intelligence Research*, pages 83–124.

Tambe, M., Pynadath, D., Chauvat, N., Das, A., and Kaminka, G. (2000). Adaptive agent integration architectures for heterogeneous team members. In *Proceedings of the International Conference on Multi-Agent Systems*, pages 301–308.

Tang, F. and Parker, L. E. (2005). ASyMTRe: Automated synthesis of multi-robot task solutions through software reconfiguration. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1513–1520.

Tash, J. and Russell, S. (1994). Control strategies for a stochastic planner. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 1079–1085.

Torralba, A., López, C. L., and Borrajo, D. (2016). Abstraction heuristics for symbolic bidirectional search. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 3272–3278.

Torreño, A., Onaindía, E., and Sapena, O. (2012). An approach to multi-agent planning with incomplete information. *European Conference on Artificial Intelligence*, 242:762–767.

Torreño, A., Onaindia, E., and Sapena, O. (2014a). A flexible coupling approach to multi-agent planning under incomplete information. *Knowledge and Information Systems*, 38(1):141–178.

Torreño, A., Onaindia, E., and Sapena, O. (2014b). FMAP: Distributed cooperative multi-agent planning. *Applied Intelligence*, 41(2):606–626.

Trinquart, R. and Ghallab, M. (2014). An extended functional representation in temporal planning: towards continuous change. In *Proceedings of the 6th European Conference on Planning*, pages 203–208.

TSai, P.-W., Pan, J.-S., Liao, B.-Y., and Chu, S.-C. (2009). Enhanced artificial bee colony optimization. *International Journal of Innovative Computing, Information and Control*, 5(12):5081–5092.

Turpin, M., Michael, N., and Kumar, V. (2013). Trajectory planning and assignment in multirobot systems. In Frazzoli, E., Lozano-Perez, T., Roy, N., and Rus, D., editors, *Algorithmic Foundations of Robotics X*, volume 86 of *Springer Tracts in Advanced Robotics*, pages 175–190. Springer Berlin Heidelberg.

Turpin, M., Michael, N., and Kumar, V. (2014). CAPT: Concurrent assignment and planning of trajectories for multiple robots. *The International Journal of Robotics Research*, 33(1):98–112.

Vidal, V. (2004a). A lookahead strategy for heuristic search planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling*, pages 150–159.

Vidal, V. (2004b). The yahsp planning system: Forward heuristic search with lookahead plans analysis. In *4th International Planning Competition*, pages 56–58. Citeseer.

Vidal, V. (2011). YAHSP2: Keep it simple, stupid. *Booklet of the 2011 International Planning Competition*, pages 83–90.

Vidal, V. (2014). YAHSP3 and YAHSP3-MT in the 8th international planning competition. *Proceedings of the 8th International Planning Competition*, pages 64–65.

Vig, L. and Adams, J. A. (2006a). Market-based multi-robot coalition formation. In *Proceedings of the 8th International Symposium on Distributed Autonomous Robotic Systems*, pages 227–236.

Vig, L. and Adams, J. A. (2006b). Multi-robot coalition formation. *IEEE Transactions on Robotics*, 22(4):637–649.

Werger, B. and Matarić, M. (2000). Broadcast of local eligibility for multi-target observation. In Parker, L. E., Bekey, G., and Barhen, J., editors, *Distributed Autonomous Robotic Systems 4*, pages 347–356. Springer Japan.

Wicke, D., Freelan, D., and Luke, S. (2015). Bounty hunters and multiagent task allocation. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 387–394.

Woosley, B. and Dasgupta, P. (2013). Multirobot task allocation with real-time path planning. In *Proceedings of the 26th International Florida Artificial Intelligence Research Society Conference*, pages 574–579.

Xia, N., Jiang, J., and Hu, Y. (2004). Solution to agent coalition problem using improved ant colony optimization algorithm. In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 475–478.

Yoon, S. W., Fern, A., and Givan, R. (2007). FF-Replan: A baseline for probabilistic planning. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling*, pages 352–359.

Younes, H. L. and Simmons, R. G. (2002). On the role of ground actions in refinement planning. In *Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling Systems*, pages 54–61.

Younes, H. L. S. and Simmons, R. G. (2003). VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research*, 20(1):405–430.

Zhang, Y. and Parker, L. (2010). IQ-ASyMTRe: Synthesizing coalition formation and execution for tightly-coupled multirobot tasks. In *2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5595–5602.

Zhang, Y., Sreedharan, S., and Kambhampati, S. (2015). Capability models and their applications in planning. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1151–1159.

Zhu, D., Huang, H., and Yang, S. (2013). Dynamic task assignment and path planning of multi-auv system based on an improved self-organizing map and velocity synthesis method in three-dimensional underwater workspace. *IEEE Transactions on Cybernetics*, 43(2):504–514.