AN ABSTRACT OF THE DISSERTATION OF

Gilberto Marcon dos Santos for the degree of Doctor of Philosophy in Robotics presented on June 8, 2020.

Title: Coordination for Scalable Multiple Robot Planning Under Temporal Uncertainty

Abstract approved: _____

Julie A. Adams

This dissertation incorporates coalition formation and probabilistic planning towards a domain-independent automated planning solution scalable to multiple heterogeneous robots in complex domains. The first research direction investigates the effectiveness of Task Fusion and introduces heuristics that improve task allocation and result in better quality plans, while requiring lower computational cost than the baseline approaches. The heuristics incorporate relaxed plans to estimate coupling and determine which tasks to fuse. As a result, larger temporal continuous planning problems involving multiple robots can be solved. The second research direction introduces new coordination methods to merge plans and resolve conflicts while extending the framework to domains with stochastic action duration. Merging distributedly generated plans becomes computationally costly when task plans are tightly coupled, and conflicts arise due to dependencies between plan actions.

Existing methods either scale poorly as the number of agents and tasks increases, or do not minimize makespan, the overall time necessary to execute all tasks. A new family of plan coordination and conflict resolution algorithms is introduced to merge independently generated plans, minimize the resulting makespan, and scale to a large number of tasks and agents in complex problems. A thorough algorithmic analysis and empirical evaluation demonstrates how the new conflict identification and resolution models can impact the resulting plan quality and computational cost across three heterogeneous multiagent domains and outperform the baseline algorithms.

Coordination for Scalable Multiple Robot Planning Under Temporal Uncertainty

by

Gilberto Marcon dos Santos

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented June 8, 2020
Commencement June 2020

Doctor of Philosophy dissertation of Gilberto Marcon dos Santos presented on June 8, 2020.

APPROVED:

Major Professor, representing Robotics

Director of the Robotics Program

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Gilberto Marcon dos Santos, Author

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my advisor, Dr. Julie A. Adams, for her guidance while performing this research as well as her thorough and attentive reading and editing of my writing. I would like to extend my sincere thanks to all my committee members, Drs. Geoffrey Hollinger, Kagan Tumer, Prasad Tadepalli, and Kipp Shearman, who inspired me and instilled confidence, either through the teaching of classes during my doctoral studies, or through mentorship and advice in the projects on which I collaborated. I would also like to thank all of my coworkers and colleagues, who supported me during my graduate studies, either through their direct feedback on my writing, or by providing me with advice on technical matters. I would also like to thank all of my friends and roomates, who offered their presence and emotional support. Ultimately, I would like to thank my family, my parents, and my brother, who continually supported me from a long distance. This work was partially supported by NSF grant #1723924.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF APPENDIX FIGURES

LIST OF APPENDIX FIGURES (Continued)

LIST OF APPENDIX FIGURES (Continued)

Chapter 1: Introduction

Advancements in sensing, processing, and actuation technologies are rapidly expanding robots' capabilities. The growing number of robot capabilities enables them to accomplish complex tasks and assist in first response to major disasters. Robots are navigating rough terrain, manipulating objects, and perceiving the environment, but integrating those individual capabilities into a coherent effort to accomplish specific goals requires complex decision making and planning. Real-world applications predominantly rely on a human operator for reasoning and decision making, because automated domain-independent planning methods do not meet real-world requirements [4]. Real-world problems require devising plans rapidly that account for uncertain environments, imprecise sensors, limited communication, and multiple robots. Current automated planners offer features targeting various subsets of these requirements, but no planner offers a complete set of features necessary to meet all requirements. This dissertation introduces an automated planning framework for multiple robot systems to accommodate the distributed, dynamic, and uncertainty requirements, while solving complex problems within real-world processing time and computational resource constraints.

One set of features required to meet real-world requirements relates to the dynamic and decentralized aspects of multiple robot systems. *Concurrency*, or concurrent action execution, allows robots to act simultaneously and take actions

in parallel. *Uncertain action outcomes* break the assumption that each action has a single deterministic result and requires contingency branching on each possible action outcome. *Partial observability* breaks the assumption that robots have an accurate perception of the world. The second set of required features relates to planning model expressiveness, and includes features that will permit more compact, expressive, and general representations of the planning problems. Many expressive features are supported by most modern deterministic planners, but are not supported by most probabilistic planners. *Durative actions* explicitly represents time for modeling actions that take different amounts of time to complete. *Continuous fluents* permit modeling continuous numerical values and constraints, such as battery power and payload weight. A comprehensive description of model and algorithm features is presented in Chapter 2.

Covering a wide range of features is one side of the equation for effective multiple robot planning. The other major limitation planners have is producing feasible plans within limited computational processing time and resources. Balancing planning features and computing power leads to the definition of scalability. *Scalability* is defined as a constraint over various factors, represented in Figure 1.1 by the area of the filled hexagon. A planner's scalability mutually constrains six factors, represented on the axes. *Robot heterogeneity* is determined by the breadth of robot capabilities across the available robots; where a larger variety of capabilities implies higher heterogeneity. The *time horizon* represents the timespan permissible for planning, where longer time horizons allow for planning actions over a larger number of time steps, or longer time period. *Domain model complexity* determines

the complexity of the world model (i.e., how many different objects with which the robots are assumed to interact). The *number of robots* defines how many robots will cooperate to execute the plan. A goal of this dissertation is to generate plans faster, while simultaneously using less memory in order to generate plans on the smaller, cheaper computers often found in robotic systems. The mutual constraint imposed by scalability requires trading-off one factor for the other when modeling a problem. Increasing the model complexity and the number of robots, for example, can require reducing the time horizon and the robot heterogeneity for any given planner, as presented by the dotted line in Figure 1.1.

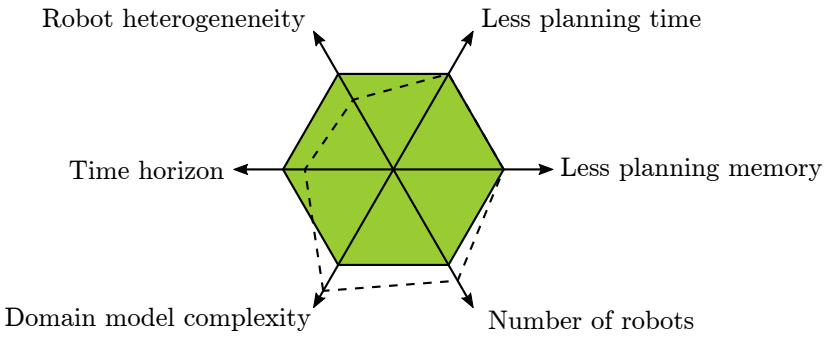


Figure 1.1: Scalability as a resource in multiple robot domain-independent planning. Domain-independent algorithms allow the arbitrary distribution of the available "scalability" across the multiple planning aspects when modeling a particular problem.

No single planner in the literature incorporates all the features deemed necessary for real-world planning. Further, the more expressive planners sacrifice the needed scalability. General domain-independent multiple robot planning necessitates facilitating a larger set of requirements and improved scalability. The primary contribution of this dissertation is a domain-independent planning approach

for real-world multiple robot systems. Dukeman and Adams [62] demonstrated that coalition formation can dramatically improve the scalability of deterministic multiple robot planners that incorporate concurrency and durative actions. This dissertation incorporates coalition formation into planning with probabilistic temporal domains. Integrating existing probabilistic planners and coalition formation algorithms improves the scalability of probabilistic planners, while preserving their features. The differences between deterministic and probabilistic planning created significant research questions related to key process of the Coalition Formation then Planning framework. The plan merging process, for example, is necessary to fuse plans of multiple robot coalitions into a single coherent plan. Deterministic plans can be optimally merged using existing plan merge algorithms, such as presented by Cox and Durfee [52]. However, the existing algorithms do not support durative actions or action duration uncertainty. The plans generated by probabilistic temporal planners with concurrent durative actions and action duration uncertainty, such as the Actions Concurrency and Time Uncertainty Planner [24], are stochastic task networks (STNs) [110] and cannot be optimally merged by existing algorithms.

Three research directions are presented towards incorporating coalition formation and probabilistic planning with the generation of a domain-independent automated planning solution scalable to multiple robot systems in complex uncertain domains.

The first research direction, presented in Chapter 4, is a new family of coordination heuristics that improve plan quality when solving continuous temporal

problems for dozens of heterogeneous robots. The new heuristics improve plan quality, while requiring comparable computational resources, relative to baseline heuristics across a First Response and an extended Blocks World domain.

The second research direction, presented in Chapter 5, adapts the framework to probabilistic domains with temporal uncertainty. Partial-order planning models allow decentralized plan execution with concurrency and action duration uncertainty. However, coordinating the distributedly generated plans becomes computationally costly when task plans are tightly coupled, and conflicts arise due to dependencies between plan actions. Existing coordination methods either scale poorly as the number of robots and tasks increases, or do not minimize makespan, the overall time necessary to execute all tasks. A new family of plan coordination and conflict resolution algorithms is introduced to merge independently generated plans, minimize the resulting makespan, and scale to a large number of tasks and robots in complex problems.

The third research direction, presented in Chapter 6, introduces a domain-independent approach to coordinate heterogeneous multiple robot systems, while executing complex plans. The system deploys the Coalition Formation then Planning framework in a physical real-world multiple robot system.

Chapter 2: Background

Planning for multiple robot systems leverages algorithms developed by the multiagent planning community. Multiagent planning algorithms extend the single-agent planning literature, which roots back to the origins of artificial intelligence research [142]. A comprehensive survey on planning models and algorithms is presented, covering the modern multiagent planners and summarizing their fundamental components that derive from the classical single-agent literature. Coalition formation, a superset of task allocation, can enhance multiple robot planning to improve scalability and solve complex problems. Coalition formation algorithms are presented, in addition to the hybrid Coalition Formation then Planning framework [61] that interleaves coalition formation and planning to solve temporal and continuous heterogeneous multiple robot planning problems that scale to dozens of agents in complex domains. The terms robot and agent are used interchangeably, in the sense that a robot is assumed to be driven by a decision-making agent.

Domain-independent multiple robot planning models and a generalized model taxonomy are presented in Chapter 2.1. Domain-independent multiple robot planning algorithms and a generalized algorithm taxonomy are presented in Chapter 2.2. Coalition formation models, algorithms, and a hybrid approach that applies coalition formation to accelerate domain-independent multiple robot planning are presented in Chapter 2.3.

2.1 Planning Models

The planning model is a very important aspect of automated planning and establishes the constraints and assumptions related to the robots and the environment. Different algorithms can yield different solutions that have different advantages and limitations based on the planning model [76]. The automated planning literature is segmented around different model families, which makes it challenging to establish a clear picture of the field. Models make varying assumptions about the environment and the robots, many of which are implicit to the specific literature segment. A planning model taxonomy is presented in order to provide a clearer picture of the various models and their limitations.

2.1.1 Planning Model Features

The interactions between robots and the environment are specified by a planning model [76]. Robot actions are the key element of the robot-environment interaction, and the core objective of planning is to establish a course of action. Plans determine actions that each robot must take to accomplish specific goals [76]. Robot actions are defined in terms of action requirements (i.e., conditions upon which actions are applicable) and action effects (i.e., changes caused to the environment and to the robots when a certain action is taken). Planning models determine which real-world features are considered by the planning process and also determine implicitly which solution algorithms can be applied [76]. Some features relate to the dynamic and decentralized aspects of multiple robot systems, while other features

relate to planning model expressiveness, permitting more compact, expressive, and general representations of the planning problems. Twenty-one features, considered the most relevant to multiple robot planning, are presented.

2.1.1.1 Centralized and Decentralized Plan Execution

A variety of definitions exist for multiagent planning [51]. Distributed plan synthesis for single agent execution employs multiple planning agents that cooperate to formulate a plan for a single agent to execute [85, 192]. Centralized plan synthesis for multiagent execution employs a single planning agent to formulate a plan for multiple executing agents [34]. Distributed plan synthesis for multiagent execution employs multiple planning agents to formulate a plan for multiple executing agents [52, 178]. Centralized and distributed plan synthesis are discussed in detail in Chapter 2.2. Multiple robot systems require multiple executing agents; thus, distributed plan synthesis for single agent execution is outside the scope of this dissertation. However, single agent execution is not to be confused with centralized plan execution for multiple agents.

Centralized plan execution models assume that a central decision-making authority will coordinate the plan execution and will command agents as a master puppeteer [142]. Centralized plan execution allows using the same planning solutions available for single-agent planning, but assumes perfect instantaneous communication between the decision-making authority and each agent. *Decentralized plan execution* models assume agents will independently make decisions, while ex-

ecuting the plan without using a centralized authority. Models for decentralized plan execution generate plans that each agent can follow independently to achieve the system goals [142]. Real-world multiple robot planning systems employ a hybrid approach that combines aspects of both centralized and decentralized plan execution [76]. Combined with a hierarchical problem decomposition, high level planning and execution monitoring can be employed in a centralized fashion, while low level planning and reactive control operate locally to give agents autonomy.

2.1.1.2 Concurrency

Sequential planning algorithms assume actions are taken one at a time. Multiple robot systems require action concurrency, or concurrent action execution. Concurrency breaks the sequential assumption and allows actions to be executed in parallel, potentially by a large number of robots. Some models permit concurrency, but impose other unrealistic assumptions. A common assumption is that of time discretization, which permits concurrency, but requires perfect synchronization across all robots during plan execution. Synchronization is often infeasible due to unreliable and delayed communication [104]. Another factor that can either facilitate or prevent effective automated planning for multiple robot systems is model abstractedness.

2.1.1.3 Model abstractedness

Planning models offer various levels of abstraction. More compact and abstract models have higher semantic value and facilitate modeling real-world problems. *Flat* models represent the lowest level of abstraction and describe the planning problem in terms of discrete states and actions. Transitions between states are represented as the cross product of all states and actions, which must be exhaustively enumerated, generating sparse and inefficient representations that have little semantic value [170]. *Factored* models, also known as state-variable models, represent the planning problem in terms of state variables and yield more compact representations than flat models [82]. Transitions between states are implicitly represented as conditional action effects on the state variables, which are succinctly described by propositional logic. Factored models can be directly converted into flat models by enumerating all combinations of valid state variable values. *Relational* models, also known as first-order models, represent the highest level of abstraction for existing planning models and describe planning problems in terms of objects, functions, relations, and predicates [143]. Relational models generate the most compact and semantically valuable problem representations and can be converted into factored models by instantiating all relationships into state variables. Converting relational models into factored models and factored models into flat models generates a combinatorial increase in representation, because of the combinatorial nature of the factored and relational representations [145]. Planners that operate directly on factored and relational models benefit from the compact-

ness and have significant efficiency gains [104]. Propositional and first-order logic are supported by most modern deterministic planners, but lack support by many probabilistic planners. Model compactness and expressiveness are further developed by representing time and temporal constraints.

2.1.1.4 Temporal Planning and Durative Actions

Time and temporal constraints are commonly represented using *durative actions*. Temporal constraints establish that the start, or the end of one action must occur before or after another action, respectively. Durative actions take a finite amount of time to execute and enhance model representation for multiple robot planning, because action execution time plays a fundamental role in most domains when multiple robots execute actions concurrently [76]. Durative actions incorporate start time and execution duration; thus, they require specific preconditions to be satisfied by the action start deadline. Indoor firefighting, for example, involves opening doors, which can take seconds, and putting down fires, which can take minutes. The time difference required to execute each action means that multiple short actions can be scheduled simultaneously with the execution of a longer action. Planners that ignore differences in action execution times result in idle robots during plan execution and longer execution duration [76].

Temporal models often go beyond durative actions by defining temporal constraints that are independent of each robots' actions. *Timed initial-literals* allow modeling arbitrarily timed conditions and effects, such as general deadlines [76].

Durative goals, a special case of timed initial-literals, allow specifying deadlines representative of when specific goals must be accomplished [76]. Timed initial-literals require advanced temporal models, which have been incorporated by a small set of planners [76]. Multi-valued, numeric, and continuous variable representations further extend abstractedness.

2.1.1.5 Boolean, Multi-Valued, Numeric, and Continuous Fluents

Early planning models rely on Boolean state variables, or fluents, for describing the robots and the environment. *Multi-valued fluents* simplify the problem description by fusing multiple Boolean variables into a concise multi-valued state variable [76]. *Numeric fluents* allow more compact representations by enabling quantified resources, such as the number of a robot's spare tires or payload slots. Numeric *continuous fluents* permit modeling continuous numeric values and constraints, such as a robot's battery power and payload weight [76]. Continuous fluents require factored or relational levels of abstractedness, because a flat model requires an infinite number of state transitions to represent continuous state variables [76]. *Continuous observations* allow agent percepts to be a real-valued reading, rather than a discrete reading. Continuous observations are often found in partially observable models, where the agent observation is a numeric value on a continuous range, such as the certainty output of an object classifier [45]. *Continuous actions* allow robots to take actions involving continuous parameters (e.g., the action determines how much torque the robot must apply on its motors from a continuous

range of valid torque values) [104]. *Continuous effects* allow durative actions to continuously change fluent values (e.g., the move action will continuously decrease the fuel level over time) [104]. *Continuous time* models explicitly represent time as a real-valued dimension and allow action-independent timed events, such as timed initial-literals. Continuous time is not necessary for continuous actions, and most planners implement continuous actions without continuous time using temporal dependency graphs that allow discrete reasoning about the continuous actions. Continuous effects and continuous actions are strictly different. Continuous actions can have continuous effects, but most models implement continuous effects for discrete actions [104]. Incorporating continuous fluents, continuous observations, and continuous actions generates a variety of model classes, such as continuous-state, continuous-action, hybrid-state, and hybrid-action models.

2.1.1.6 Hybrid and Continuous-State and Continuous-Action Models

Continuous fluents produce *continuous-state* models, where every state variable is continuous, and *hybrid-state* models, where some state variables are continuous and others are discrete [104]. Similarly, continuous actions produce *continuous-action* and *hybrid-action* models. Models that incorporate both continuous-states and continuous-actions are commonly studied in control theory. The complexity of continuous control restricts it to low level models of single-robot domains, which is beyond the scope of this dissertation [95]. Early approaches for solving nondeterministic continuous-state and continuous-action models include the linear-quadratic-

Gaussian model, which incorporates continuous noise, but does not incorporate piece-wise linear transition dynamics and rewards [19]. Recent approaches include the extended algebraic decision diagrams [146] that compactly represent general piece-wise linear functions, and the continuous-state and continuous-action Markov Decision Process (MDP), which incorporates piece-wise linear transition dynamics and rewards [202]. Uncertainty, typically incorporated into advanced control systems' models, also plays an important role in planning models for real-world problems.

2.1.1.7 Deterministic, Nondeterministic, and Probabilistic Models

Deterministic models assume actions have deterministic outcomes, duration, and resource requirements, which must be predictable at the time of planning. Real-world scenarios have multiple sources of uncertainty, including uncertain action outcomes, uncertain action duration, and uncertain action resource consumption. The various levels of uncertainty incorporated into the planning model require restrictive assumptions and determines the applicable solution algorithms. Russell and Norvig [142] divide planning for nondeterministic domains into three categories: Sensorless planning, contingent planning, as well as online planning and replanning. Sensorless planning, also known as conformant planning or coercive planning, occurs in the absence of observations and takes actions to reduce the belief space down to goal states [142]. Sensorless planning lacks feedback from the environment; thus, the resulting plan consists of a sequence of actions, as in

classical deterministic planning [142]. Contingent planning incorporates observations and satisfies environments with uncertain outcomes, generating conditional plans with branching based on percepts [142]. Online planning and replanning interleaves planning and execution in order to tackle the exponential growth in complexity caused by extended time horizon planning, and extends planning to unknown environments [142].

Sensorless planning is effective under strong assumptions about the robots' and the environments' attributes, which can be achievable in controlled manufacturing tasks [71]. However, in a broad range of environments such assumptions cannot be made and contingent planning is necessary. Online planning and replanning often use contingent planners with a fixed time horizon and replan according to heuristic conditions [75]. Multiple robot systems replanning requires a reliable communication infrastructure to share the updated plans across all robots. Both centralized plan synthesis, in which one robot replans and communicates the new plans to all other robots, and decentralized plan synthesis, were robots exchange messages to replan in a distributed manner, require synchronization and extensive message passing [76]. Faulty communications can cause inconsistencies among the plans each robot executes, generating suboptimal and potentially catastrophic results; thus, replanning has limited benefit for limited communication scenarios [76]. Contingent planning is a better alternative to replanning.

Contingent planning models can be separated into nondeterministic and probabilistic [75]. Probabilistic planning is a subset of nondeterministic planning where certain outcomes are more likely than others, which allows for leveraging proba-

bilistic patterns in the form of an estimated likelihood of action outcomes. The probabilistic approach to nondeterministic planning brings decision theory into planning by incorporating the concept of state-action utility as a substitute for the set of goal states [75]. The utility encapsulates state-action values as a reward function, which enhances the problem representation capability by allowing for a quantitative preference model for state-action sets. Probabilistic planning can be applied to domains in which the objective is not to reach a goal state, but rather to spend more time in high-valued states (i.e., preferred states), while avoiding low-valued states (i.e., undesirable states). The decision-theoretic approach also permits solving *infinite-horizon* problems, where the goal is not necessarily to reach an end state, but rather to harvest the most rewards per time step and achieve the most profitable behavior in the domain [76].

Models incorporate forms of nondeterminism, other than uncertain action outcomes, including *uncertain action execution duration* and *uncertain action resource requirements* [54]. Incorporating uncertain outcomes can be used to devise contingent plans, also known as policies, and enables decision making according to the environmental state, rather than a predefined sequence of actions [54]. Probabilistic models with uncertain outcomes account for the likelihood of action outcomes, specify a probability density function over a set of action effects, and allow the planner to benefit from action outcomes that are more likely than others [54]. *Uncertain duration* allows modeling durative actions with nondeterministic execution time and associates a probability density function with a range of possible action execution durations [54]. *Uncertain resource requirements* allow modeling

nondeterministic resource requirements for each action, and resource consumption is modeled probabilistically. Purely nondeterministic planners assume unknown action outcome likelihoods and build contingent plans for all possible action outcomes, while probabilistic planners account for probabilities in order to generate plans with higher expected value [76]. Modeling uncertainty permits incorporating partial observability, a fundamental concept for many deployed multiple robot systems.

2.1.1.8 Partial observability

Observability determines whether the robots can perceive perfectly all environmental aspects during plan execution, or if the robots make incomplete and inaccurate observations according to problem-specific conditions. *Full observability* occurs when all state aspects yield complete and accurate observations of their true value [142]. Partial observability assumes that robots do not have perfectly accurate perception, which captures models that include the uncertainties and limitations of robots' sensing. Partial observability accounts for the limited information robots possess regarding one another, allowing for decentralized execution, where each robot makes noisy and limited observations from its individual perspective [142]. Partial observability also enables planning for environments that require limited communication. Robots have to use costly actions and make noisy observations in order to communicate and share information. Deciding when to communicate becomes part of the planning process, which accounts for the cost and the un-

certainties of sending and receiving messages [76]. Domains with *no observability* require open-loop plans, where the world is either assumed to behave deterministically, or the plan is expected to funnel the initial belief state into the goal states [142].

2.1.1.9 Limited or Absent Communication

Limited or non-existing communication infrastructure is associated with partial observability, but not all partially observable models permit limited communications. Real-world settings often involve costly, delayed, and error-prone communication that is dependent upon distance, line-of-sight, and nondeterministic failures [37]. Communication may not exist due to constrained hardware and power or environmental features. Most planners generate solutions that accommodate multiple robot systems, but require perfect communications during plan execution. Few planners generate solutions executable within limited communication domains, and such planners tend to have poor scalability due to model complexity [124]. A more strict subset of models relaxes the communication requirement, but demands synchronization. Models with required synchronization do not demand data transmission or message sharing, other than broadcast pulses that synchronize time across all agents [157].

2.1.1.10 Explicit and Implicit Communication

Models that incorporate *explicit communication* make strong assumptions regarding the nature and availability of a communication infrastructure [124]. Models that incorporate *implicit communication* are applicable to a wider variety of environments and make no assumptions regarding the availability of a communication infrastructure [124]. Models with implicit communication do not describe communication as part of the framework, rather they incorporate communication by means of individual robot actions and observations [124]. For example, there can be a broadcast action that has specific requirements, cost, and execution time, and a received message observation, which has a conditional probability distribution on success and correctness. Implicit communication occurs via various mediums, beyond traditional radio-frequency networks, such as using gestures, lights, sounds, and any other actions that modify the environment. Models with implicit communication generate plans that adapt to the limitations of the medium, such as inaccuracies and delays endured when generating and receiving a message. However, implicit communication requires probabilistic models with partial observability [124]. The expressive features reviewed thus far allow for a more compact and accurate world model. The last model feature leverages the problem structure to reduce the overall computational complexity and accelerate the plan synthesis process.

2.1.1.11 Agent-Wise Independence

Real-world problems can often be simplified by factoring the model around certain aspects of the multiple robot system and the environment [124]. The actions of one robot may have no impact on the state of other robots, meaning that the state of a robot depends only on its own state and actions in *transition-independent* models. The state of a robot is independent of other robots' states and actions [157]. Multi-robot underwater ocean mapping is an example domain that can be modeled with transition-independence. Robots can be assumed to be independent, because collisions are unlikely. Urban and indoor mapping, on the other hand, cannot be assumed to be transition independent, because robots are capable of blocking each other. A robot's observations depend only on its own state and its own actions with *observation-independent* models (i.e., the observations are independent of other robots' states and actions). The underwater model can also be assumed to be observation-independent, because the sensors' relative ranges and the environment size make it unlikely that one robot will influence other robot's sensing. Urban and indoor mapping cannot be assumed to be observation independent, because robots may occlude each other's sensors. *Reward-independent* models assume that maximizing local robot rewards leads to maximizing the global system reward, such as with additive rewards. Exploration domains often cannot be assumed to be reward-independent, because the value of exploring a single area multiple times is less than the value of exploring multiple areas once. Models that assume independence allow for faster algorithms, but can only be applied to

domains where the assumptions are valid [157]. Model features often determine algorithm features, but a clear distinction is made in this Chapter to present features that distinctively belong to the algorithms alone.

The planning model features are summarized in Table 2.1. The feature domains are defined to accommodate planning models relevant for multiple robot systems, and can be extended into sub-sets. Plan execution, for example, can be broken into several levels of decentralization, but dividing it into centralized and decentralized is deemed sufficient for the scope of this Dissertation. Some features are found on a broad range of planning models, such as uncertain action outcomes, whereas others can only be found on a handful of models, such as timed initial-literals [75].

2.1.2 Concurrent Planning Models

A variety of mutually incompatible planning models that make different implicit assumptions and restrict the domain of applications in different ways have been developed [142]. Most deterministic temporal and continuous planning models evolved from discrete and non-temporal models and inherited their incompatibilities associated with probabilistic domains. Most probabilistic planning models assume time is discrete; thus, they cannot seamlessly incorporate concurrency or durative actions [76]. Markovian and non-Markovian planning models, which incorporate inherently different advantages and limitations, are reviewed with a particular focus on concurrent models, a minimal requirement for multiple robot planning.

Table 2.1: Planning model features, their domain values and descriptions.

Feature Name	**Feature Domain**	**Feature Description**
Concurrency	<Yes, No>	Parallel action execution.
Durative actions	<Yes, No>	Time-extended action execution.
Timed initial-literals	<Yes, No>	Exogenous timed events, independent of actions.
Durative goals	<Yes, No>	Goals with deadlines.
Multi-valued fluents	<Yes, No>	State variables have multi-valued domains.
Numeric fluents	<Yes, No>	State variables have numeric domains.
Continuous fluents	<Yes, No>	State variables have continuous domains.
Continuous Observation	<Yes, No>	Observation variables have continuous domains.
Continuous actions	<Yes, No>	Action variables have continuous domains.
Continuous effects	<Yes, No>	Actions have continuous time-varying effects.
Continuous time	<Yes, No>	Timeline modeled as a continuous variable.
Required communication	<Yes, No>	Communication must be part of the framework.
Required synchronization	<Yes, No>	Synchronization must be part of the framework.
Plan execution	<Centralized, Decentralized>	Whether a centralized authority coordinates execution communicating actions to each robot.
Model abstractedness	<Flat, Factored, Relational>	Level of model representation abstraction, from more grounded to more lifted.
Action outcomes	<Deterministic, Nondeterministic, Probabilistic>	Actions have predictable, unpredictable, or stochastically predictable outcomes.
Action duration	<Deterministic, Nondeterministic, Probabilistic>	Actions have predictable, unpredictable, or stochastically predictable durations.
Action resource required	<Deterministic, Nondeterministic, Probabilistic>	Actions have predictable, unpredictable, or stochastically predictable resource consumption.
Observability	<Full, Partial, No>	Robots' perception is perfect and complete, imperfect and partial, or have no perception.
Communication mode	<Explicit, Implicit>	Communication is a mandatory part of the framework, or is relegated to robot actions.
Agent independence	<Transition, Observation, Reward, None>	Model decoupling relative to the actions, observations and rewards of each robot.

2.1.2.1 Markovian Models

Most of the probabilistic planning literature assumes discrete time and Markovian dynamics [157]. The Markovian planning model family, summarized in Figure 2.1, incorporates deterministic discrete-time models as a subset. Markovian models assume that state transition and observation functions depend exclusively on the current world state and the robots' current actions [63]. The discrete time assumption makes basic Markovian models unfit for multiple robot planning, but generalizations to the model are promising, and extend to well-understood adversarial domains.

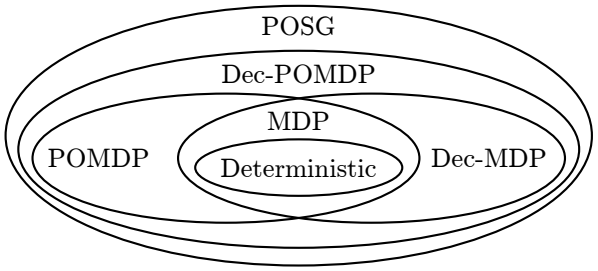


Figure 2.1: Discrete-time Markovian planning models, expanded from Amato [14].

Partially observable stochastic games include cooperative domains in which all robots receive the same reward, and adversarial domains in which robots have different reward functions, such as in zero-sum games [122]. Incorporating the Markovian assumption and decision-theoretic state-action values, the Markov Decision Process is a principled model for probabilistic planning [27]. Partially Observable Markov Decision Processes (POMDPs) incorporate partial observability over state variables and explicitly represent belief, which permits problem-solving to perform

information gathering actions [84].

Decentralized POMDPs allow each robot to maintain a separate set of actions and observations, and permit action execution concurrency [14]. Decentralized POMDP algorithms are solved in a centralized manner, but the policies generated are executed in a decentralized manner. Decentralized MDPs are a special case of Decentralized POMDPs in which the observation of all robots determines the global state (i.e., each robot perfectly observes part of the world state, and each part of the world state is observed by at least one robot) [11]. Decentralized MDPs apply to controlled environments in which robot sensors can be assumed to reliably cover all relevant world aspects, but do not generalize to large dynamic environments, where robots must balance gathering information and acting towards their goals. Solving Decentralized POMDPs generates individual policies for each robot. Executing Decentralized POMDPs' policies does not require explicit communication, because the robots are assumed to have individual belief spaces, do not share observations, and do not observe other robots' actions directly.

Two limitations make Decentralized POMDPs ill-suited for multiple robot domains [15]. Decentralized POMDPs require synchronized plan execution, because of the Markovian time discretization. Robots do not have to communicate during execution, but must take actions synchronously. Decentralized POMDP extensions permit real-world multiple robot implementations that circumvent this limitation [128]. The second drawback of Decentralized POMDPs is the plan synthesis computational complexity, which is $O(\mathbb{A}^{\frac{n(\mathbb{O}^h-1)}{\mathbb{O}-1}})$ [15], where $\mathbb{O}$ denotes the largest agent observation set, $\mathbb{A}$ denotes the largest agent action set, n is the number of agents,

and h is the time horizon. However, variations and extensions have proven successful for solving real-world problems, such as warehouse management, bar-tending, and delivering packages [14].

Multiagent POMDPs allow action execution concurrency and leverage the efficiency of centralized POMDP algorithms, but require centralized plan execution. Multiagent POMDPs have demonstrated applicability to complex multiple robot systems for environments with fast, stable, and freely available communications, such as in controlled indoor environments [107]. MDPs and POMDPs are inadequate for multiple robot systems for not allowing concurrency. Multiagent POMDPs require centralized execution and reliable communication infrastructure, whereas Decentralized POMDPs require synchronization during plan execution. None of the models support durative actions and uncertain durations, which can be addressed by some extended Markovian models.

Extended abstract MDP representations have generated compact models that are easy for humans to model and specify, while facilitating efficient algorithmic solutions. The Stochastic Planning Using Decision Diagrams model [82] extends binary decision diagrams into algebraic decision diagrams to represent MDPs with multivalued discrete state variables. Algebraic decision diagrams have been used to generate solutions for more compact and abstract MDP models, such as the factored MDP [35], relational POMDP [145], and relational continuous-state and continuous-action MDP [202], but none supports concurrency. Both Concurrent MDPs [103] and Generalized Semi MDPs [201] support concurrency, durative actions, and uncertain durations. However, neither supports partial observability,

which is addressed by Decentralized POMDPs.

Semi MDPs relax the synchronous time assumption [141] by permitting decision epochs to occur asynchronously at irregular intervals that can only be estimated probabilistically. Semi MDPs associate random variables with each time step, permit durative actions and asynchronous execution in multiple robot systems. Combined with a hierarchical domain decomposition, Semi MDPs involve temporally-extended macro-actions, which abstract away the continuous nature of time and the state space of the underlying low level actions [171]. Some of the extended Markovian models permit durative actions and uncertain durations, but do not permit continuous fluents or uncertain resource requirements, which are addressed by some non-Markovian models.

2.1.2.2 Non-Markovian Models

The Markovian planning models accommodate uncertain outcomes, but suffer from the time discretization and require synchronized action execution; thus, imposing restrictions on modeling concurrency and durative actions [37]. Non-Markovian models more naturally accommodate concurrency and durative actions [104].

Environments with uncertain outcomes lacking any statistics regarding the state transition probabilities have been attempted using Fully Observable Nondeterministic models [76]. The absence of probabilistic patterns from the state-action transition function requires plans to include contingency branches for all action outcomes [113]. The Fully Observable Nondeterministic models maximize

the probability of reaching goal states, rather than maximizing the expected rewards, but do not support concurrency [48].

The earliest non-Markovian planning model incorporating concurrency, durative actions, and uncertain duration was the Simple Temporal Network with Uncertainty, which separates scheduling and planning [183]. Contingent scheduling policies are generated at planning time, but the actual action scheduling is deferred until execution time. Suboptimal solutions are generated and a centralized execution authority capable of scheduling and communicating in real time is necessary in order to communicate scheduling decisions to all robots. The Simple Temporal Network with Uncertainty model is effective for controlled real-world environments, such as in harbor operations management [32].

The Resources and Time Uncertainty model is a non-Markovian continuous time model that supports uncertain resource requirements and uncertain action duration [23]. The hybrid of state-space models and dynamic Bayes networks lacks support for uncertain outcomes, which renders the planner incapable of generating contingency plans [23]. The Strong Temporal Planning with Uncontrollable Durations model incorporates durative actions, uncertain duration, multi-valued fluents, and timed initial-literals [43], but does not support uncertain outcomes or uncertain resource requirements [44].

None of the models support all types of uncertainty, which is addressed by some algorithms, presented in Chapter 2.2, by using their own models.

2.1.3 Planning Languages

Human-readable planning problem modeling languages facilitate modeling complex problems and establishing a common user interface across multiple planners [142]. Language definitions can specify model features and limitations, while permitting high-level problem decomposition. Standardization efforts attempted to establish a common denominator language across multiple planners, but the wide variety of models and the limitations of the proposed standard languages jeopardized the effort, as such no language satisfies all planning models and domains [144].

The Planning Domain Definition Language (PDDL) [105] is a deterministic relational problem modeling language based on first-order logic and inspired by the action description language [132]. PDDL's action schema represents actions in a compact manner by explicitly stating action preconditions and effects on state variables [99]. PDDL assumes that anything not explicitly mentioned is false, making domain descriptions compact. The Probabilistic PDDL language extends PDDL to incorporate uncertain action outcomes [199]. Problems described in Probabilistic PDDL can be efficiently solved using determinization, which translates a probabilistic domain into a deterministic domain by selecting the most likely action outcome as the deterministic action outcome and discarding all other outcomes; thus, allowing the use of deterministic planners [181].

The Relational Dynamic Influence Diagram Language [144] is a compact POMDP representation based on Dynamic Bayes Networks. The Relational Dynamic Influence Diagram Language permits representing domains where nonde-

terminism plays a stronger role and allows defining problems that are impossible to solve using trivial determinized translations [144]. The lack of continuous time and durative actions are the primary limitations with this language, which also lacks strict uncertainty (nondeterministic non-probabilistic action outcomes). The Relational Dynamic Influence Diagram Language permits exogenous stochasticity and concurrency for stronger probabilistic domains where determinization-based planners have had limited success. The prevalence of Probabilistic PDDL and the Relational Dynamic Influence Diagram Language have resulted in current state-of-the art planners adopting either language or incorporating translators, such as Stochastic Planning Using Decision Diagrams [82] and Symbolic Perseus [137]. The more recent planners, the Probabilistic Planning Based on Upper Confidence Bounds Applied to Trees [87] for MDPs and the Determinized Sparse Partially Observable Tree [164] for POMDPs, do not support durative actions.

The Action Notation Modeling Language extends PDDL to support hierarchical task decomposition and advanced temporal modeling features, such as timed initial-literals, and durative goals [163], but it does not support uncertainty [64]. The Action Notation Uncertain Modeling Language extends the Action Notation Modeling Language in order to bridge the gap between temporal and probabilistic planning languages, while incorporating uncertain duration [108]. The Probabilistic Graphical Model Extensive Markup Language is a factored Dynamic Bayes Networks description language used to model MDPs, POMDPs and Decentralized POMDPs [17]. The Probabilistic Graphical Model Extensive Markup Language is used to describe Decentralized POMDP and Decentralized MDP problems for the

Multiagent Decision Process Toolbox solvers [125].

2.2 Planning Algorithms

Planning algorithms are generally tied to a specific model type. Some algorithmic approaches have several adaptations into multiple models, but have not been generalized to all models. Dynamic programming, for example, has influenced algorithms across all MDP derivatives, but has been applied to a few Non-Markovian models with durative actions [76]. A general algorithm taxonomy is presented to classify algorithms across the wide range of presented models. Deterministic algorithms, while not directly applicable to real-world problems, represent the building blocks of modern planners [76]. Most probabilistic algorithms do not support concurrency; thus, the small subset of concurrent probabilistic planners are presented.

2.2.1 Planning Algorithm Features

A wide variety of algorithms can be applied to each type of planning model. The algorithms offer various trade-offs and significantly different features, which are summarized.

Decentralized plan synthesis distributes the planning effort across multiple agents in order to leverage parallel processing, eliminate points of failure, and preserve agents' privacy and autonomy [63, 179]. Preserving agents' privacy and autonomy while planning for a common goal is important, as the planning agents

can have undisclosed sub-goals and leverage private sensitive information they are unwilling to disclose [40, 109]. Robots from two separate teams, for example, can plan for achieving a shared goal, while each team is also trying to achieve private goals not to be disclosed to the other team, and each team can leverage private knowledge that must not be shared. However, the computational benefits of parallel processing do not pay off, and the decentralized plan synthesis is generally less efficient due to excessive communication requirements [52, 178].

Centralized plan synthesis, in which a single processing unit performs all the planning computations, regardless of how many robots will execute the plan, is more computationally efficient than decentralized plan synthesis [178]. Centralized plan synthesis permits the use of highly efficient optimization methods and is applied to multiple robot systems by assigning concurrent plans to each robot [142]. A centralized plan is devised, tasks are assigned, and agents determine locally how to execute actions designated by the centralized plan, and possibly devise local operational plans. Existing planners capable of satisfying real-world requirements and scalability constraints synthesize plans in a centralized manner, but allow plan execution to be decentralized [76].

Hybrid plan synthesis combines aspects of both centralized and decentralized plan synthesis. Weerdt and Clement [191] separate hybrid plan synthesis into coordination before planning and coordination after planning. Coordination before planning allocates tasks to agents followed by each agent individually devising plans to achieve its tasks. Coordination after planning involves each agent individually devising plans to achieve private goals followed by coordination to minimize con-

flicts and redundancies between each agents' plans. Torreno, Onaindia, Komenda, and Stolba [179] extended this taxonomy by dividing coordination before planning into two categories, according to how agents plan after coordination. Coordination before planning with parallel planning assumes that coordination will prevent planning conflicts, and robots can plan in parallel. Coordination before planning with serial planning, or iterative response planning, assumes that robots will coordinate and take turns planning. Robots generate plans iteratively, where each robot assumes that its initial state is the final state of the prior robot's plan.

This dissertation breaks Torreno, Onaindia, Komenda, and Stolba's hybrid plan synthesis taxonomy into two separate axis. Coordination that is carried out *before*, *after*, *before and after*, or *during* planning and plan synthesis that is completed either in *parallel* or *serial*. Note that coordination during planning is equivalent to decentralized plan synthesis.

Hybrid parallel plan synthesis assumes that robots simultaneously synthesize plans. A minimal example illustrates the result of parallel plan synthesis with two robots and two room cleaning tasks. Robot A and robot B begin in the middle room, as shown in Figure 2.2 (a.i), where there is a broom. Robot A is tasked with cleaning the left-side room and robot B is tasked with cleaning the right-side room, but they have to share the broom. The robots need to use the broom in order to clean either room. Parallel plan synthesis assumes that both robots will plan in parallel and that both planning processes will have the same initial state, which is presented in Figures 2.2 (a.i) and (b.i). Robot A's plan is to move the broom to the left-side room (Figure 2.2 a.ii); and clean the left-side room (Figure 2.2 a.iii).

Robot B's plan is to move the broom to the right-side room (Figure 2.2 b.ii); and cleans that room (Figure 2.2 b.iii).

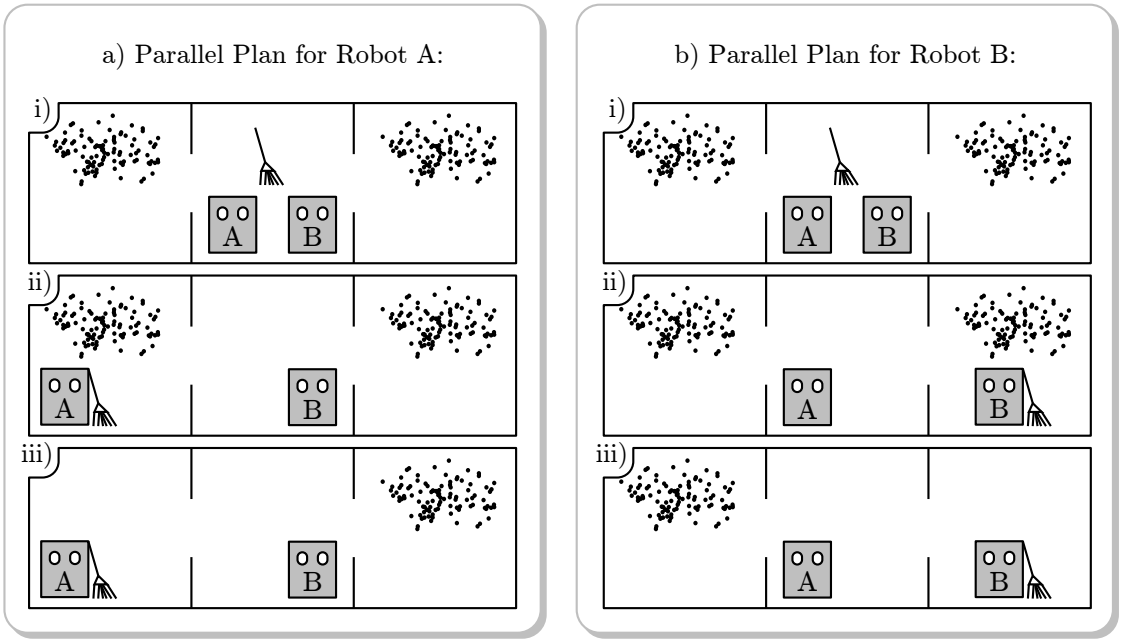


Figure 2.2: Hybrid parallel plan synthesis.

Merging the two plans is not straightforward, because the execution of one plan renders the other plan infeasible. The shared object, a broom, causes tight coupling between the two tasks, and generates a race condition. Whichever robot starts first will be able to complete its task; however, the other robot will encounter a modified initial state, where the broom is not in the middle room. Parallel plan synthesis requires loosely coupled or uncoupled tasks in order to avoid conflicting plans.

Hybrid serial plan synthesis, analogous to Torreno, Onaindia, Komenda, and Stolba's iterative response planning, assumes that robots will take turns during

planning. Serial plan synthesis assumes that robot A will plan first, generating the same plan as parallel planning did for robot A, presented Figure 2.2 (a). However, robot B will assume robot A's plan *final state* as its *initial state*, as shown in Figure 2.3 (b.i) and will move to the left-side room, where robot A is with the broom (Figure 2.3 b.ii); transport the broom to the right-side room (Figure 2.3 b.iii); and clean that room (Figure 2.3 b.iv).

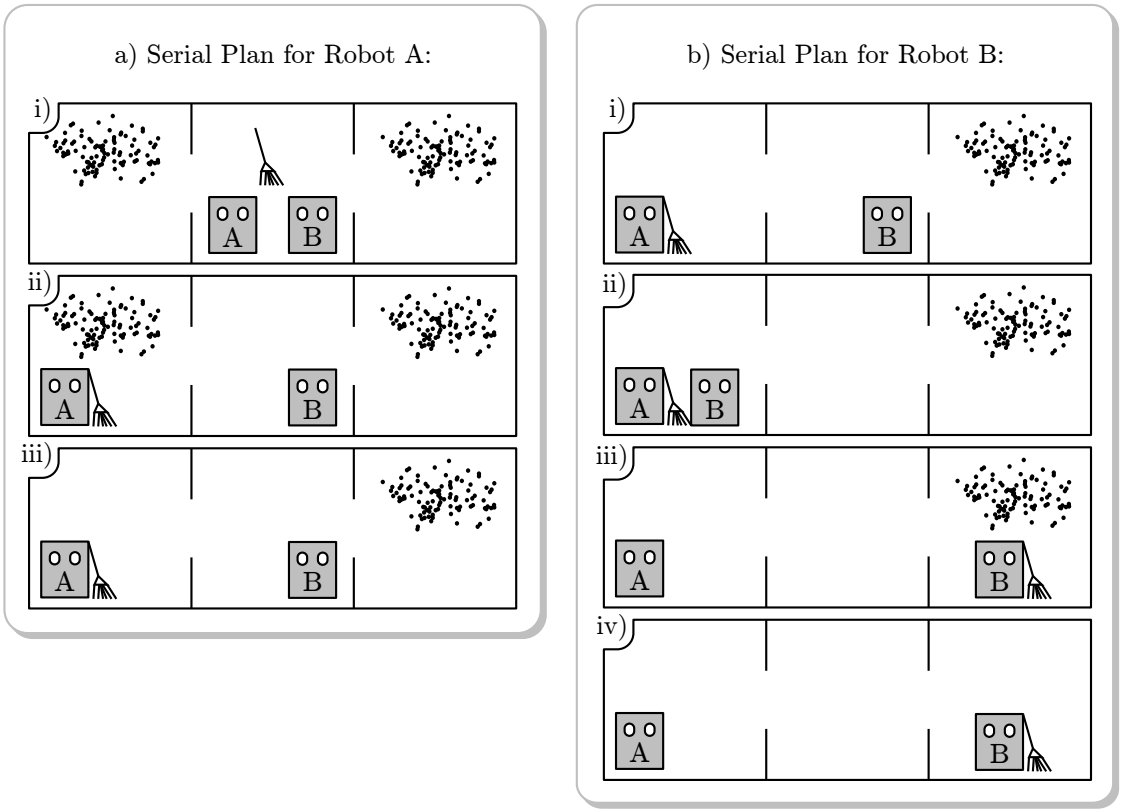


Figure 2.3: Hybrid serial plan synthesis.

The serial plans can be merged without any additional actions, and can be optimized by parallelizing actions. The shortest viable plan is obtained by merging

the two serial plans and eliminating redundant actions, producing the optimal plan presented in Figure 2.4. Robots A and B *simultaneously* move to the left-side room (Figure 2.4 ii); robot A cleans the left-side room (Figure 2.4 iii); robot B moves the broom to the right-side room (Figure 2.4 iv); and robot B cleans the right-side room (Figure 2.4 v). Note that each robot is essentially executing its own serial plan, as presented in Figure 2.3, and no extra actions are added. Serial plan synthesis supports tightly coupled tasks and permits applying merge algorithms that do not add actions, such as Cox and Durfee's [52].

The algorithm time horizon can be finite, infinite, or indefinite. Algorithms for problems with a *finite-horizon* develop a policy for a limited execution time, planning up to a specified time point in the future [157]. The *infinite-horizon* algorithms develop a policy for an unlimited execution time, allowing the robots to execute indefinitely [157]. Infinite-horizon problems often involve a discount factor for computing action values, and are often associated with models that incorporate uncertain outcomes [157]. *Indefinite-horizon* is a special case of the infinite-horizon, where the problem ends when terminal states are reached [124]. Some algorithms can operate in all horizons [124].

Algorithms offer different solution quality guarantees, which commonly fall into three categories. *Exact algorithms* have globally optimal guarantees on solution quality [157]. *Approximation algorithms*, also known as ϵ-optimal algorithms, have a provable bound on solution quality, a constant factor from the globally optimal solution [157]. *Heuristic algorithms*, also known as approximate algorithms, offer no bounds on solution quality [157]. The artificial intelligence community

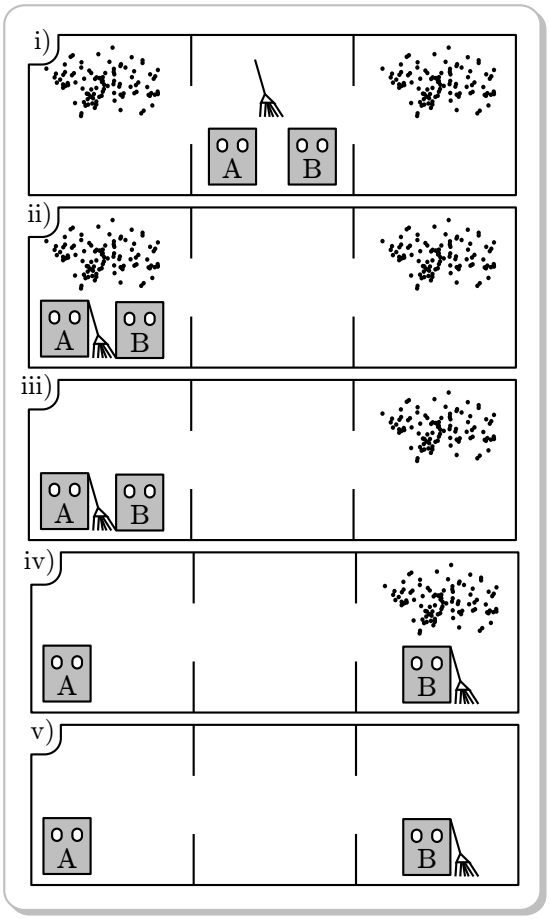


Figure 2.4: Hybrid serial plan synthesis followed by optimal plan merging.

refers to approximate algorithms as heuristic algorithms with no performance guarantees, while the theoretical computer science community defines approximation algorithms as providing provable bounds on its solution quality [124]. *Solution representation* is often tied to the algorithm adopted, but is commonly determined by the time horizon and the model observability.

Planning solutions can be divided into open-loop and closed-loop representations [76]. Open-loop solutions do not require sensing and assume the initial world

state is known. Closed-loop solutions, also known as policies and contingent plans, require observing the world state via sensor perception [76]. Closed-loop solutions have branches conditioned to robot observations and are robust to uncertain outcomes.

Open-loop solutions can be a *sequence* or a *schedule* of actions [76], which is determined by the model features. Non-observable algorithms produce open-loop action sequences, because no sensing is available. Temporal algorithms with deterministic action outcomes and deterministic action durations produce action schedules [76].

Closed-loop solutions for deterministic action outcomes and uncertain action durations can be a *partial-order* plan [34] or a *stochastic task network* (STN) [110]. *Partial-order* plans represent actions as nodes, while the edges represent action precedence requirements. Precedence requirements indicate which actions must complete prior to executing other actions. Precedence requirements are established between a subset of actions; not all actions are strictly ordered. The final action execution order is decided while executing the plan, as the action execution durations are uncertain. Partial-order plans do not incorporate action duration estimates. Algorithms with uncertain action duration estimates produce *stochastic task networks* (STNs) [110]. *Stochastic task networks* [110], also known as program evaluation and review technique networks [3, 100], extend partial-order plans with probability density functions over action durations.

Closed-loop solutions for uncertain action outcomes can be a *decision tree*, a deterministic finite-sate controller (FSC), or a stochastic finite-state controller

[124]. *Decision trees* interleave decision and outcome nodes, as in game trees [193], and have exponentially increasing memory complexity in the time horizon. At every decision point, the tree growth is combinatorial. FSCs offer a compact policy representation by allowing cyclic action outcomes, but unlike policy trees, states are mapped to actions in a one-to-one fashion that does not permit different action sequences for the same state. Partially observable planning generates belief-based or observation-based policies, depending on the planning approach [75]. Belief-based policies are mappings of belief states to actions, while observation-based policies are FSCs that require an internal state representation.

A decision tree assumes a known initial state [124]. Nodes represent actions and arcs represent observations. The root node action is executed and the observation received determines which arc to follow, leading to a child node. The child node determines the next action, and the cycle repeats. The time horizon T determines the tree depth, as the agent moves down a node with each time step t when executing the policy. Decision trees are commonly generated by finite-horizon algorithms.

Finite-state controllers represent a generalization of policy trees by permitting cycles in order to support more compact policy representations [124]. FSCs are also more appropriate for infinite-horizon problems and their associated algorithms [124]. An optimal infinite-horizon decision tree will have infinite depth, while an infinite-horizon FSCs can have finite size. *Stochastic finite-state controllers* have a probability distribution over their actions. *Deterministic finite-state controllers* represent a specific case of stochastic FSCs in which one action has likelihood 1

and the other actions have a likelihood 0. Stochastic FSCs provide higher value policies for the same controller size [157].

The probabilistic planning literature focuses on two FSC representations: Moore and Mealy [124]. Moore FSCs represent actions as nodes and observations as arcs, while Mealy machines represent both actions and observations as arcs. Moore and Mealy FSCs have equivalent representation capability, which allows Moore machines to be translated into Mealy machines and vice-versa. However, the Mealy representation is more compact and has fewer nodes and arcs than the equivalent Moore controller.

Planning algorithms optimize a variety of objectives [76]. Deterministic sequential planners often minimize the number of actions required by the plan. Deterministic temporal planners minimize the total time necessary to execute the plan, also known as plan *makespan* [76]. Goal-oriented probabilistic planners maximize the probability of successful goal achievement, whereas utility-oriented probabilistic planners maximize the utility, or expected net rewards, for a given time horizon, or for a discounted-rewards infinite time horizon. Some planners allow optimizing for multiple objectives, such as minimizing a specific numerical resource under constraints [168].

The planning algorithm features are summarized in Table 2.2. Analogous to the planning model feature domains, planning algorithm feature domains can be expanded into sub-domains. The solution representation feature domain, for example, can be expanded into a hierarchical domain by expanding FSCs into stochastic and deterministic, and into Moore and Mealy FSCs [124]. Most algorithms oper-

ate on finite and infinite time horizons, and only a handful operate on indefinite horizons [124].

Table 2.2: Planning algorithm features and feature domains.

Feature Name	**Feature Domain**	**Feature Description**
Plan synthesis	<Centralized, Decentralized, Hybrid>	Single or multiple processing units synthesize the plan.
Coordination	<Before, After, Before and After, During>	Hybrid plan synthesis scheme.
Planning Synchronization	<Serial, Parallel>	Hybrid plan synthesis mode.
Time horizon	<Finite, Infinite, Indefinite, Any>	Time extent accounted by the plan synthesis process.
Quality guarantees	<Exact, Approximation, Heuristic>	Plan quality guarantees offered by the planning algorithm.
Solution representation	<Sequence, Schedule, Partial-order, STN, Decision Tree, FSC>	Format of the generated plan.
Solution objectives	<Number of Actions, Makespan, Utility, Success Probability, Multiple>	Variables minimized or maximized by the plan synthesis process.

2.2.2 Deterministic Centralized Planning Algorithms

Early approaches to automated planning have inspired the more recent advanced planners [142]. Forward and backward search as well as planning-graph methods

are the simplest and most commonly used planning methods. *Forward search* is limited by exponential state space growth and requires heuristics to scale. *Backward search* reduces the branching factor, but requires state sets instead of individual states, which hinders derivation of well-informed heuristics [142].

Factored representations facilitate deriving domain-independent admissible heuristics that estimate the distance from a state to the goal [142]. Domain-independent heuristics are obtained by solving a relaxed version of the problem, adding edges, grouping node states, or using a planning graph. Adding more edges is achieved by ignoring action preconditions, so that every action becomes applicable at every state [142]. Grouping nodes reduces the number of states, which generates a state abstraction and ignores fluents. *Planning graphs* offer better informed admissible domain-independent heuristic estimates than relaxed plans and are the basis for graph-based planners [142]. Graphplan uses the planning graph directly and does not perform forward search [28].

Forward and backward state space search aided by admissible heuristics and Graphplan represent the majority of modern automated planners. Alternative approaches include Boolean satisfiability, situational calculus, and partially-ordered planning. Factored and relational problems can be solved as a Boolean satisfiability problem by converting action schemas into propositional ground actions [86]. Propositional logic approaches allow for domain-independent heuristics, but scale poorly [142]. Situation calculus has proven theoretical value, but lacks heuristics for planning and no scalable planner has been developed [94]. Partially-ordered planning searches for sub-goals using a least commitment approach and

merges sub-plans, while eliminating conflicts [75]. The promise of partially-ordered planning, including the ability to facilitate human understanding of the resulting plans, has been superseded by advancements incorporating forward search heuristics, such as those employed by the Fast Forward [83] and Fast Downward [81] planners. However, distributed state-space search planning can be enhanced with partially-ordered plan merging in order to permit scaling to larger problems, as demonstrated in the approach presented in Chapter 2.3.

Temporal planning permits durative actions, which allow multiple executors performing different actions at different execution times. Planning is intended to minimize makespan, the total execution time [142]. The simplest approach applies classical sequential planning and schedule plan actions, which fails when concurrency is required [61]. A common heuristic applies a classical planner to a relaxed version of the domain; thus, converting temporal actions into instantaneous actions [76]. Successful temporal planners include Temporal Graphplan [162] and the Flexible Acting and Planning Environment [64], none of which support numeric and continuous fluents.

The introduction of continuous fluents facilitates more concisely representing the world. Numerical fluents enhance the model's expressiveness by representing both continuous and discrete values, such as vehicle fuel, cargo load, or battery voltage levels [142]. Combining continuous fluents with durative actions enhances the model's fidelity by representing numerical quantities that change over time. The planners Zeno [133] and Yet Another Heuristic Search Planner [184] support continuous effects, but do not permit concurrency. The Temporal Fast Downward

[73] and the Forward-Chaining Partial-Order [46] planners support durative actions, concurrency, and continuous fluents, but do not support continuous effects [47]. The Continuous Linear [49] planner extends the Forward-Chaining Partial-Order planner to support linear continuous effects. Finally, dReal [38] supports nonlinear continuous effects, but suffers from limited scalability [61]. Most deterministic temporal planners support concurrency, durative actions, multi-valued, numeric, and continuous fluents, but lack support to any type of uncertainty and require centralized plan execution.

2.2.3 Deterministic Decentralized and Hybrid Planning Algorithms

Decentralized planning algorithms imply agents devise a plan cooperatively and coordinate throughout the planning process. The Divide and Conquer algorithm [70] leverages loose coupling in multiagent planning with multiple goals. The planning problem is resolved by distributing a graph search across all agents. The result is suboptimal in the general case and the algorithm does not scale well as the number of agents increases, even if the plans are loosely coupled [52]. The Forward Multiagent Planning algorithm supports heterogeneous agents and tightly coupled goals, but its intensive communication requirements stifle scalability to a large number of agents [178].

The Task Coordination and Decomposition algorithm [109] employs coordination before planning to preserve the privacy of self-interested planning agents. The algorithm imposes ordering constraints on tasks allocated to each agent, so that

planning can be accomplished independently. However, the method ignores the fact that one agent's plans can modify the environment in a manner that makes another agent's plans invalid.

CSP+Planing introduces the agent interaction digraph to estimate task coupling [36]. Task coupling is defined as the level of interaction between agents, where tighter coupling leads to higher computational complexity. The algorithm seeks to minimize the coupling when allocating tasks in order to improve planning complexity. Problem decomposition is formulated as a constraint satisfaction problem. Solving the constraint satisfaction problem dominates the plan synthesis time, rendering the algorithm inefficient [1].

Cox and Durfee's plan merging algorithm solves action redundancy and consistency flaws, while merging plans generated by multiple planning agents [52]. Action redundancy flaws are resolved to minimize the plans' number of actions, whereas consistency flaws must be resolved to generate valid plans. The merging mechanism operates in the plan space and allows for a set of actions to collectively replace a single redundant action, while attaining computational efficiency for loosely coupled and uncoupled tasks. However, the algorithm does not account for durative actions, does not minimize makespan, and does not scale to a large number of robots when tasks are tightly coupled. A recent temporal plan merge algorithm supports concurrent actions for single mobile robot tasks [111]. The algorithm generates relaxed plans for each task prior to merging. However, the method requires adding literals manually to each task's initial state and is not applicable to multiple robot tasks. Chapter 2.3.4 presents a formal definition

of the plan merging problem, and Chapter 5 introduces a plan merging problem formulation that incorporates durative actions and new algorithms.

Zhang, Sreedharan, and Kambhampati apply a model of capabilities as a heuristic for state-space forward search [204]. Capabilities are modeled as the likelihoods of an agent achieving a particular state from any other state. The method employs a Bayesian network to learn the likelihood capabilities from plan traces, but requires an arbitrarily large number of plan execution simulations covering initial and goal states.

Recent decentralized planning algorithms use hybrid serial plan synthesis, or iterative response planning, to improve scalability. The Multi-Agent Planning by Plan Reuse algorithm performs task allocation then planning using relaxed reachability analysis after generating relaxed plans for all agent-task combinations. However, the method requires homogeneous agents [31]. The algorithm can be applied to a heterogeneous mobile multiple robot system by using actuation maps instead of relaxed plans, but it does not generalize to complex tasks [134]. The Agent Decomposition-Based Planner uses causal graphs to decompose the planning problem [1], and has been extended to support concurrent actions in a real-world industrial mobile manipulator robot domain [53], but does not scale to a large number of robots. The Multiagent Planner for Required Cooperation uses m planning agents and n executing agents, where $m \leq n$ [166]. Tasks are allocated to the m planning agents, which devise plans for the n executing agents. The goals of a planner are concatenated with the goals of the next planning agent to guarantee that the next agent will not undo the goals achieved by the previous agent. The

μ-SATPLAN algorithm performs task allocation prior to planning [60]. None of the existing hybrid serial plan synthesis planners support concurrency.

2.2.4 Fully Observable Probabilistic Planning Algorithms

Multiple robot planning problems have multiple sources of uncertainty and require concurrency [103]. Many algorithms exist for solving centralized MDPs and POMDPs, but the need for concurrency renders them insufficient for the multiple robot systems; thus, the focus is on decentralized Markovian and non-Markovian algorithms [103].

Fully Observable MDP algorithms do not support concurrency without extensions, which imposes limitations [103]. The Probabilistic Planning Based on Upper Confidence Bounds Applied to Trees uses badit methods [88] to achieve high scalability on factored MDPs, but does not support concurrency [87]. Recent planners for Fully Observable Nondeterministic models include Fast-Forward Hindsight+ [198] and the Planner for Relevant Policies [112], which was further extended to support multiagent systems, do not support concurrency [114]. The Duration and Uncertainty on Resources planner family solves Concurrent MDPs, but the discretized model of time severely limits scalability, due to combinatorial explosion when all features are enabled [103]. The Tempastic planner [200] solves Generalized Semi MDPs and generates deterministic plans that are repaired on failure points to build a contingent plan, supporting durative actions and uncertain action outcomes, but not action duration uncertainty. Generalized Semi MDPs were

implemented in real-world multiple robot domains, but used centralized execution [106].

Forward state-space search extended with dynamically generated Bayesian networks is used to solve first-order problems by modeling continuous probability distributions on durative actions [22]. A Bayesian network is built and evaluated incrementally alongside the state-space search, outperforming concurrent planners that discretize time on multiple robot domains [22]. The method generates goal-oriented probabilistic plans with a lower bound on the probability of success, and facilitates decentralized asynchronous execution by producing STNs [110]. The Resources and Time Uncertainty planner extends the Bayesian networks approach to model uncertainty over the continuous fluents [23], while Actions Concurrency and Time Uncertainty Planner (ActuPlan) extends the Resources and Time Uncertainty planner to generate conditional plans [24]. ActuPlan does not support uncertain outcomes, and the generated conditional plans offer different trade-offs in plan quality and probability of success [24]. The Continuous Linear [49] planner was extended to support the Strong Temporal Planning with Uncontrollable Durations models, which generated schedules of actions that guarantee achieving the problem goals, but does not support uncertain outcomes [44].

Contingent non-Markovian probabilistic planners incorporate uncertain outcomes and generate policies. The Probabilistic Temporal Planner incorporates a non-Markovian model that addresses uncertain outcomes, concurrency, durative actions, and uncertain duration [97] using labeled real-time dynamic programming [29], plans for a finite-horizon, and generates decision trees that maximize the

probability of reaching a goal. The Factored Policy Gradient planner searches for plans directly using local optimization and uses the planning model only to build a plan execution simulator [39]. The Factored Policy Gradient model includes concurrency, durative actions, numeric fluents, uncertain outcomes, uncertain duration, and uncertain resource requirements. The Factored Policy Gradient uses policy-gradient methods to minimize the plan steps or maximize the probability of success. The Quantum Planner extends ActuPlan with labeled real-time dynamic programming in order to support uncertain outcomes, and generates results faster than the Duration and Uncertainty on Resources and the Factored Policy Gradient planners for a mars rovers domain [37]. The non-Markovian probabilistic planners offer the largest set of temporal, continuous, and uncertainty features, but do not support partial observability, which is provided by the Decentralized POMDP algorithms.

2.2.5 Decentralized Partially Observable Markovian Algorithms

Decentralized Partially Observable Markov Decision Processes (Decentralized POMDPs) algorithms incorporate concurrency, decentralized execution, partial observability, and limited communication. Three primary algorithmic approaches are presented: Heuristic search, dynamic programming, and optimization algorithms. Further, general enhancements that improve all the associated algorithms are presented.

2.2.5.1 Heuristic Search Algorithms

Heuristic search algorithms represent top-down approaches that start at the initial node at time t_0 and progress towards the time horizon T. All presented heuristic search algorithms produce an optimal decision tree policy, with the exception of Szer and Charpillet's algorithm [174], which produces sub-optimal deterministic FSCs for infinite-horizon Decentralized POMDPs [174]. The simplest top-down algorithm for solving Decentralized POMDPs is the best first search, also known as brute force search for Decentralized POMDPs [174]. The best first search algorithm is complete and offers an optimal solution, but takes a double exponential time ($O(\mathbb{A}^{\frac{n(\mathbb{O}^h-1)}{\mathbb{O}-1}})$), relative to the time horizon T. This algorithm is a degenerate heuristic search algorithm, because its heuristic value is a constant zero. The infinite-horizon best first search algorithm finds the optimal deterministic FSC for a given controller size with a discounted reward [174]; however, the algorithm requires the controller size to be specified, and finds the best controller for that size. Other heuristic search algorithms incorporate an admissible heuristic that does not compromise completeness and optimality [142].

The Multiagent A* (MAA*) is an optimal heuristic search algorithm for Decentralized POMDPs that ranks nodes according to the net expected value of arriving at the node, plus the expected value of a child node estimated by the admissible heuristic [175]. The initial MAA* incorporated three heuristics: The MDP heuristic, the POMDP heuristic, and the Recursive MAA* heuristic [175]. The heuristics provide increased accuracy. More elaborate heuristics can result in faster search-

ing, due to more aggressive search space pruning. MAA* can only expand one time step further into the time horizon, when compared to best first search algorithm [157]. Most of the expanded states for Decentralized POMDPs are nodes at the last time step t of the time horizon T.

The Search for Policies in Distributed Environments algorithm [182] incorporates an MDP-based branch and bound heuristic. The Search for Policies in Distributed Environments algorithm exploits the agents' network structure by organizing agents into a depth first search tree in order to capitalize on the independence of the different tree branches. The heuristic is only accurate when teammate interactions are loosely coupled, which provides no benefits in tightly coupled Decentralized POMDPs.

The Generalized MAA* algorithm [118] established the basis for a new family of heuristic search algorithms. The Exploiting the Last-Stage Independence Generalized MAA* algorithm [119] uses collaborative graphical Bayesian games and leverages locality of interaction to more quickly solve Decentralized POMDPs, while the Lossless Clustering of Observation Histories Generalized MAA* [120] generates optimal solutions orders of magnitude faster than other Generalized MAA* approaches. The Bayesian Game Branch and Bound algorithm [121] exploits the Bayesian game structure of Decentralized POMDPs in order to prune nodes more efficiently. Generalized MAA* with Incremental Clustering and Generalized MAA* with Incremental Clustering and Expansion [165] both greatly improve search scalability compared to previous exact heuristic-search methods. Refinements cluster equivalent histories and expand nodes incrementally [123].

All of the heuristic search methods offer exact solutions for finite-horizon Decentralized POMDPs, but suffer from worst case complexity of $O(\mathbb{A}^{\frac{n(\mathbb{O}^h-1)}{\mathbb{O}-1}})$ [15]. Execution times can be improved by a constant factor and generate solutions within a bound of the optimal by relaxing the heuristic. Recent admissible heuristics have improved scalability by orders of magnitude, but heuristic search approaches remain restricted to small time horizons for finite-horizon problems and are not applicable to infinite-horizon domains.

2.2.5.2 Dynamic Programming Algorithms

Dynamic programming algorithms perform a bottom up search [11]. The Joint Equilibrium-Based Search for Policies [115] applies dynamic programming and search for the locally optimal policy for finite-horizon Decentralized POMDPs. The Joint Equilibrium-Based Search for Policies algorithms employs an alternating maximization, or hill-climbing for Decentralized POMDPs, which optimizes one agent's policy at a time, while the others are held fixed. The Forward-Sweep Policy Computation [69] algorithm for finite-horizon Decentralized POMDPs efficiently solves each time step as a separate Bayesian Game, but offers no quality guarantees. The Bounded Policy Iteration algorithm [25] iteratively improves each agent's FSC by linear programming, while other FSCs are fixed. A node is chosen from one FSCs whose parameters are updated by solving a linear program on each iteration. The Bounded Policy Iteration algorithm solves infinite-horizon Decentralized POMDPs, but offers no quality bounds. The Generalized Bounded Policy

Iteration algorithm [26] provides an approximate solution with quality within a bound of the global optimum for infinite-horizon Decentralized POMDPs.

The first dynamic programming algorithm to optimally solve Decentralized POMDPs has a very high memory complexity, which exhausts the memory prior to exceeding the computational limits [79]. The ϵ-pruning Dynamic Programming algorithm reduces the memory requirements and produces solutions within a bound of the optimal [5]. The error bound ϵ determines a value margin by which policies are pruned. ϵ-pruning guarantees a bound of the optimal, but other solutions with no guarantees have been shown to provide better quality results for benchmark problems [172].

The Memory Bounded Dynamic Programming algorithm [172] and the Improved Memory Bounded Dynamic Programming algorithm [173] address the memory complexity and provide better quality policies than the ϵ-pruning Dynamic Programming. The Memory Bounded Dynamic Programming and the Improved Memory Bounded Dynamic Programming algorithms impose limits on the number of decision trees and the number of observations evaluated at each time horizon, bounding memory demand growth, and expanding the size of the Decentralized POMDPs that can be solved.

The dynamic programming backup operation is the most influential in terms of complexity [11]. Several exact algorithms incorporate different pruning approaches. A heuristic search prunes unreachable points in the Point-Based Dynamic Programming algorithm [176]. The Point-Based Incremental Pruning algorithm [55] replaces the dynamic programming backup step with a branch-and-bound search in

the space of the joint policies. The Point-Based Incremental Pruning Incremental Policy Generation algorithm [7] incorporates reachability analysis pruning, while the Constraint Based Point Backup algorithm [90] uses a weighted constraint satisfaction formulation to reduce the complexity of the dynamic programming backup operation. The Point-Based Policy Generation algorithm [194] is a linear program incorporating an the Improved Memory Bounded Dynamic Programming that increases the efficiency of selecting the best trees to expand. The Point-Based Policy Generation algorithm offers solutions within a bound of the global optimal, and performs as well as the original Improved Memory Bounded Dynamic Programming algorithm with the pruning technique removed.

Other algorithms improve scalability by compressing action and observation histories. The Lossless Policy Compression Dynamic Programming algorithm [33] employs a compressed policy representation and uses a mixed integer linear program to find the optimal policies. The Memory Bounded Dynamic Programming Observation Compression algorithm [41] provides higher quality policies and improved scalability relative to the original Improved Memory Bounded Dynamic Programming algorithm by compressing observations. The Feature-Based Heuristic Search Value Iteration algorithm [56] solves Decentralized POMDPs represented as continuous-state MDPs with piecewise-linear convex value functions.

2.2.5.3 Optimization Algorithms

Several promising algorithms have framed the Decentralized POMDP problem as a combinatorial optimization problem and applied various optimization methods [11]. A mixed integer linear program formulation was adopted to optimally solve finite-horizon Decentralized POMDPs using off-the-shelf solvers [16]. The mixed integer linear program approach solves Decentralized POMDPs faster than previous exact algorithms, but runtime remains orders of magnitude slower than heuristic algorithms. Nonlinear programming algorithms search for stochastic controllers for Decentralized POMDP problems with an infinite-horizon [6, 9], but cannot guarantee global optimality.

Various forms of Monte-Carlo sampling algorithms have been applied to solve infinite-horizon Decentralized POMDPs [67]. Evolution strategies [65] and genetic algorithms [68] can solve Decentralized POMDPs for real-world multiple robot systems [66]. The Direct Cross-Entropy Method is a Monte-Carlo sampling algorithm similar to evolutionary algorithms [117] that offers comparable performance to heuristic approaches, such as the Joint Equilibrium-Based Search for Policies algorithms [115], but does not offer quality guarantees. An Expectation-Maximization sampling algorithm is applied to the infinite-horizon Decentralized POMDPs planning problem transformed into its equivalent mixture of dynamic Bayes nets [91]. The resulting Expectation-Maximization algorithm has the anytime property. The Expectation-Maximization algorithm achieves similar quality and scalability to the nonlinear programming approach, but does not offer bounds on global optimality.

Periodic FSCs are applied to the Expectation-Maximization sampling algorithms in order to solve larger infinite-horizon Decentralized POMDPs [129]. Scalability is improved by applying Expectation-Maximization to factored Decentralized POMDPs [130]. Average rewards are optimized instead of discounted cumulative rewards to achieve better quality results with Expectation-Maximization [131]. The Monte-Carlo Expectation Maximization algorithm [195] allows solving Decentralized POMDPs without full prior knowledge of the model parameters. The method employs sampling strategies from reinforcement learning to efficiently explore the policy space, such as upper confidence bounds. The model-free Monte-Carlo Expectation Maximization converges faster to local optima, but can produce worse quality solutions. None of the Expectation-Maximization methods offer quality guarantees [11].

The Direct Cross-Entropy Method is the most directly applicable to multiple robot systems, because it supports an infinite time horizon and has superior scalability, yet Decentralized POMDPs do not support temporal and continuous features, and uncertainty other than action outcomes. Enhancements to Decentralized POMDPs enabled real-world multiple robot proofs of concept.

2.2.5.4 Enhancements

A number of enhancements are applicable to the existing Decentralized POMDP algorithms [107]. The enhancements include different forms of FSCs, such as Mealy FSCs [10], applying hierarchical task decomposition to Decentralized

POMDPs, and transforming Decentralized POMDPs into continuous-state MDPs. Some infinite-horizon Decentralized POMDP problems can be reformulated into indefinite-horizon Decentralized POMDPs and can be solved faster [8]. A sample-based approach can exploit the goal structure effectively, while more quickly generating higher quality solutions [8].

Decentralized POMDPs are transformed into Occupancy MDPs and can be solved faster as continuous-state MDPs [57]. The Feature-Based Heuristic Search Value Iteration algorithm solves Decentralized POMDPs represented as continuous-state MDPs with piecewise-linear convex value functions [58]. The Feature-Based Heuristic Search Value Iteration algorithm improves scalability, due to its compact feature-based representation of occupancy states and decision rules [59].

Recent approaches incorporate hierarchical decomposition of actions for solving Decentralized POMDPs [124], which was proven feasible for multiple robot planning in complex environments under uncertainties. Macro-Action Decentralized POMDPs incorporate temporally-extended macro-actions to more quickly solve the underlying Decentralized POMDPs problem [12]. The Memory Bounded Dynamic Programming method was extended to solve Macro-Action Decentralized POMDPs, but requires full specification of the underlying Decentralized POMDP model, domain-specific handcrafted Macro-Actions, and imposes a discrete model of time [12].

The Macro-Action Decentralized POMDP Heuristic Search algorithm relaxes the requirements of the Memory Bounded Dynamic Programming method by in-

corporating the Decentralized Partially Observable Semi MDP model and models continuous time [15]. The Macro-Action Decentralized POMDP Heuristic Search algorithm breaks the problem into a high level discrete Decentralized Partially Observable Semi MDP and a low level continuous-time and continuous-state control problem. The decomposition permits incorporating existing discrete Decentralized POMDP solvers, while abstracting away the continuous time and continuous-state of the underlying control problem to low level controllers. The low level controllers can be either handcrafted or automatically synthesized, depending on the domain. Each low level controller is treated as a macro-action by the higher level Decentralized Partially Observable Semi MDP. However, the Macro-Action Decentralized POMDP Heuristic Search algorithm requires estimating macro-actions' values, transition times, and terminal conditions using a domain-specific model or simulator [15]. Nevertheless, the Macro-Action Decentralized POMDP Heuristic Search algorithm generates effective policies for coordinating multiple heterogeneous robots in a partially observable environment with limited communications [15]. The Graph-Based Direct Cross-Entropy method extends the Direct Cross-Entropy Method to generate better control policies more quickly than the Macro-Action Decentralized POMDP Heuristic Search algorithm [127].

Macro-actions are temporally extended actions that can require different amounts of time to execute, analogous to *durative-actions* [124]. Macro-Action Decentralized POMDPs solve larger problems by planning at a higher abstraction level. Macro-Action Decentralized POMDPs have solved a decentralized asynchronous real-world multiple robot box pushing problem, without explicit com-

munication [13]. Macro-Action Decentralized POMDPs can be solved faster with Expectation-Maximization [98] and dynamic programming [15]. The Decentralized Partially Observable Semi MDP model [126], constructed upon Macro-Action Decentralized POMDPs, allows an abstract problem representation and extends the solutions to larger continuous-state problems.

Promising results for real-world multiple robot systems exist [128], but they are limited to a small number of research groups and have been difficult to replicate. Coalition formation is an extension to task allocation [159]. Coalition formation provides a principled approach towards leveraging the varying levels of coupling encountered in heterogeneous multiple robot planning to improve planning scalability and solve real-world problems without ignoring a broader set of model and algorithm features.

2.3 Coalition Formation for Scalable Multiple Robot Planning

Planning for multiple agents is largely intractable, but assigning tasks to the most appropriate agent coalitions is accomplished with reasonable scalability. The coalition formation problem is NP-hard [159], but domain-independent planning is EXPSPACE-complete [72]. The planning process can require orders of magnitude longer than the corresponding coalition formation problems; thus, the overhead created by coalition formation is minimal. The developments that led to integrating coalition formation and planning to achieve computationally feasible solutions for planning and allocating tasks in complex, large scale, and heterogeneous mul-

tiple robot systems are presented.

2.3.1 Definitions

Coalition formation is a generalization of task allocation where coalitions, as opposed to individual agents, are assigned tasks. Agents are grouped into coalitions, and each coalition is assigned a task, according to the capabilities offered by the coalitions and the capabilities required by the tasks. The coalition formation problem takes a set of n agents, $\Phi = \{\phi_1, \ldots, \phi_n\}$, modeled as an agent capability vector set C^{Φ}, a set of m tasks, $V = \{v_1, \ldots, v_m\}$, modeled as a task capability vector set C^{V}, and yields a solution that is a mapping of tasks to coalitions, $Coalitionformation : V \rightarrow \Phi_v \mid \Phi_v \subseteq \Phi$, yielding a set of m coalition-task pairs $P = \{p_1, p_2, \ldots, p_m\} \mid p_i = \langle \Phi_i, v_i \rangle \mid \forall i \in [1, m]$. Agent and task capability vector sets represent resources, services, or a hybrid of both resources and services, as defined in Chapters 2.3.1.1, 2.3.1.2, and 2.3.1.3, respectively.

2.3.1.1 The Resource Model

Resource-oriented coalition formation problems model agents and tasks as vectors of real values that represent numerical features, such as distance sensor range, camera field of view, or battery autonomy. A resource is a nonnegative real number r_j, and the resources the agents possess are represented as a vector Res_{ϕ}, $Res_{\phi} = \{r_1^{\phi}, r_2^{\phi}, \ldots, r_k^{\phi}\}$, where each vector entry r_j^{ϕ} determines how much of resource j

agent ϕ offers and k is the number of resources modeled. The agent resource vector set Res^{Φ} associates a resource vector with each of the n agents and is represented by $Res^{\Phi} = \{Res_1, Res_2, \ldots, Res_n\}$.

Tasks are defined as a vector of necessary resources Res_v, $Res_v = \{r_1^v, r_2^v, \ldots, r_k^v\}$, where each vector entry r_j^v specifies the amount of resource j required to complete task v. The task resource vector set Res^V associates a resource vector with each of the m tasks, and is defined as $Res^V = \{Res_1, Res_2, \ldots, Res_m\}$.

A coalition $\Phi_v \subseteq \Phi$ is a subset of agents capable of executing a task v, if $\left(\sum_{\phi \in \Phi_v} r_j^{\phi}\right) \geq r_j^v \mid \forall j \in \{1, 2, \ldots, k\}$, meaning that the sum of resources j offered by each agent ϕ in coalition Φ_v is greater than or equal to the amount of resource j required by task v for all k resource types. More compactly: $\left(\sum_{\phi \in \Phi_v} Res_{\phi}\right) \geq Res_v$.

2.3.1.2 The Service Model

Service-oriented coalition formation problems model agents and tasks as vectors of Boolean values that represent actions, such as obstacle avoidance, identifying targets, or manipulating certain types of objects. A service is defined as a Boolean value s_j, and the services the agents offer are represented as a vector Ser_{ϕ}, $Ser_{\phi} = \{s_1^{\phi}, s_2^{\phi}, \ldots, s_k^{\phi}\}$, where each vector entry s_j^{ϕ} determines if agent ϕ offers service j, and k is the number of services modeled. The agent service vector set Ser^{Φ} associates a service vector with each of the n agents and is represented by $Ser^{\Phi} = \{Ser_1, Ser_2, \ldots, Ser_n\}$.

Tasks are defined as a vector of necessary services Ser_v, $Ser_v = \{s_1^v, s_2^v, \ldots, s_k^v\}$, where each vector entry s_j^v specifies the number of agents offering service j required to complete task v. The task service vector set Ser^V associates a service vector with each of the m tasks, and is defined as $Ser^V = \{Ser_1, Ser_2, \ldots, Ser_m\}$.

A coalition $\Phi_v \subseteq \Phi$ is a subset of agents capable of executing a task v if $\left(\sum_{\phi \in \Phi_v} s_j^\phi\right) \geq s_j^v \mid \forall j \in \{1, 2, \ldots, k\}$, meaning that the number of agents offering service j in coalition Φ_v is greater than or equal to the number of agents offering service j required by task v for all k service types. More compactly: $\left(\sum_{\phi \in \Phi_v} Ser_\phi\right) \geq Ser_v$

2.3.1.3 The Capability Model

The capability model is an abstract problem formulation that is more general and incorporates both the resource and service models. A capability c_j is a non-negative real number, or Boolean value that represents a resource or a service. The capabilities the agents offer are represented as a vector of capabilities C_ϕ, $C_\phi = \{c_1^\phi, c_2^\phi, \ldots, c_k^\phi\}$, where each vector entry c_j^ϕ represents a capability j offered by agent ϕ (i.e., c_j^ϕ determines if agent ϕ offers service j or how much of resource j agent ϕ offers). k represents the number of capabilities modeled. The agent capability vector set C^Φ associates a capability vector with each of the n agents and is represented by $C^\Phi = \{C_1, C_2, \ldots, C_n\}$.

Tasks are defined as a vector of necessary capabilities C_v, $C_v = \{c_1^v, c_2^v, \ldots, c_k^v\}$, where each vector entry c_j^v specifies a capability j required by task v (i.e., c_j^v

specifies the number of agents offering service j required to complete task v or the amount of resource j required by task v). The task capability vector set C^V associates a capability vector with each of the m tasks and is defined as $C^V = \{C_1, C_2, \ldots, C_m\}$.

A coalition $\Phi_v \subseteq \Phi$ is a subset of agents capable of executing a task v if $\left(\sum_{\phi \in \Phi_v} c_j^{\phi}\right) \geq c_j^v \mid \forall j \in \{1, 2, \ldots, k\}$, meaning that the number of agents offering service j in coalition Φ_v is greater than or equal to the number of agents offering service j required by task v for all k service types, or that the sum of resources j offered by each agent ϕ in coalition Φ_v is greater than or equal to the amount of resources j required by task v for all k resource types. More compactly: $\left(\sum_{\phi \in \Phi_v} C_{\phi}\right) \geq C_v$.

2.3.2 Coalition Formation Algorithms for Multiple Robot Systems

Coalition formation algorithms for software agents, such as Shehory and Kraus' [159], cannot be directly applied to multiple robot systems, due to the physical limitations on resources sharing and communication. Vig and Adams [185] modified Shehory and Kraus' algorithm to adapt it to physical robot systems [187, 188]. Algorithms that incorporate greedy, optimization, and market-based approaches have since been developed. Distributed hierarchical multiagent structures [2], graph-based agent communication models [180], market-based algorithms [139, 147, 156, 160], and multiagent social networks [74, 169, 190] distribute the processing burden across the multiple robots, but have high message passing requirements. Central-

ized algorithms have a single point of failure, but do not require significant message passing. Centralized mixed integer linear program formulations [89, 148] and hybrid factor approximation algorithms [155] offer quality guarantees, but have poor scalability to larger problems. Centralized stochastic and biologically-inspired optimization algorithms, including particle swarm optimization [196, 203], ant colony optimization [140], evolutionary [153], and socially inspired algorithms [80] have no quality gurantees, but scale to larger problems.

Given such a diverse pool of multiple robot coalition formation algorithms, each having its own benefits and deficiencies, a single algorithm cannot generalize to different problems and domains [149]. The intelligent Coalition Formation for Humans and Robots (i-CiFHaR) [150] system was developed to dynamically reason over mission requirements and decide autonomously which algorithm from a library of multiple robot coalition formation algorithms to apply in order to best address the mission requirements and render the most appropriate coalition [151, 152].

2.3.3 Planning and Coalition Formation

Coalition formation and automated planning have no common literature. While the coalition formation domains include agents and tasks, planning domains include states and actions. Dukeman [61] defined a model for unifying both fields and presented approaches that address the combinatorial complexity in planning for heterogeneous multiagent systems. The Hybrid Mission Planning with Coalition Formation model is a mix of the coalition formation capability model, presented

in Chapter 2.3.1, and the temporal planning with continuous fluents model. The model allows using existing single-agent planners to solve multiagent planning problems that current multiagent planners do not have the necessary expressiveness to solve, such as that required by durative actions.

2.3.3.1 Hybrid Mission Planning with Coalition Formation

The Hybrid Mission Planning with Coalition Formation (HMPCF) [62] model is represented as a tuple $\langle S, I, A, \Phi, V, M, C \rangle$, where $S = \{s_1, s_2, \ldots\}$ is the state space, $I \subseteq S$ is the initial state, $A = \{a_1, a_2, \ldots\}$ is the action space, $\Phi = \{\phi_1, \ldots, \phi_n\}$ is the set of n robots, the grand coalition, $V = \{v_1, \ldots, v_m\}$ is the set of m tasks, $M : \Phi \rightarrow A_\Phi \mid A_\Phi \subseteq A$ is the robot-action mapping function, and C is the tuple $\langle C^\Phi, C^V \rangle$, where $C^\Phi = \{C_1, C_2, \ldots, C_n\}$ is the robot capability vector set and $C^V = \{C_1, C_2, \ldots, C_m\}$ is the task requirement capability vector set. The conjunction of all task conditions defines the goal states $G \subseteq S \mid G = \{s_1, s_2, \ldots, \} \mid s \vdash \bigwedge_{v \in V} v \mid \forall s \in G$. A solution to a HMPCF problem is a plan, π, consisting of a set of scheduled actions assigned to each robot $\phi \in \Phi$. HMPCF uses existing coalition formation and planning algorithms to solve large multiple robot planning problems that incorporate temporal constraints and continuous numerical fluents in a more tractable, albeit centralized manner. The Multiagent Capabilities and Planning Domain Definition Language extends PDDL to describe Hybrid Mission Planning with Coalition Formation problems [61]. Dukeman [61] presents two methods for solving Hybrid Mission Planning with Coalition

Formation problems, Planning Alone and Coalition Formation then Planning.

2.3.3.2 Planning Alone

Planning Alone, a baseline planning method, groups all robots and tasks into a fully centralized planning problem. Planning Alone synthesizes a goal set G as the conjunction of all task requirements and invokes a domain-independent planner. The external planner receives the grand coalition's combined action space and attempts to satisfy all task constraints embedded in the goal state set. Planning Alone generates high-quality plans, but scales poorly as the number of robots or the domain complexity increase [62]. The combinatorial complexity of centralized planning limits the problems that can be solved.

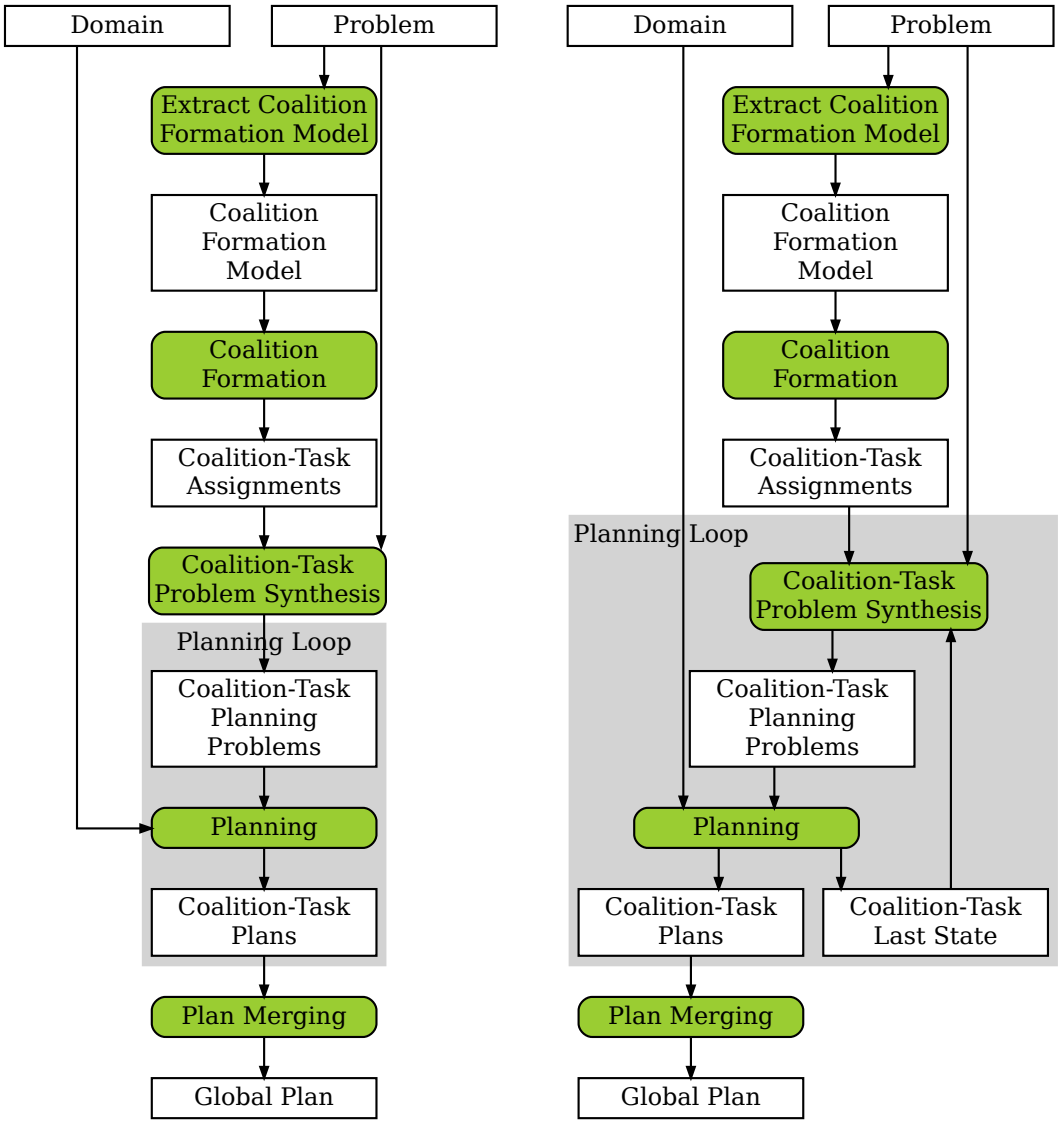
2.3.3.3 Coalition Formation then Planning

Coalition Formation then Planning (CFP) is a hybrid method that leverages coalition formation to minimize the planning combinatorial complexity and the overall computational cost. The robot and task capability vector sets and coalition formation algorithms are used to generate the coalitions and assign tasks, invoking planning algorithms for each coalition-task pair, as presented in Figure 2.5 (a). The *Extract Coalition Formation Model* process derives robot and task capability vectors from the problem description [62]. Coalition formation generates coalition-task pairs and the *Coalition-Task Problem Synthesis* process uses the coalition-task

pairs and the problem description to generate separate planning problems for each coalition-task pair. The *Coalition-Task Planning Problems* are solved separately by external planners, such as the Continuous Linear Planner (COLIN) [49] or the Temporal Fast Downward Planner (TFD) [73]. The planner produces a plan for each coalition-task pair, and the resulting plans are merged in order to prevent conflicts and generate a global plan [52].

Uncoupled tasks are resolved using parallel plan synthesis, as described in Chapter 2.2, and shown in Figure 2.5 (a). The Coalition-Task Problem Synthesis process assumes that tasks begin at the same initial state; thus, generating the task planning problem for one coalition does not require evaluating the plans generated for the tasks of other coalitions, and task plans can be synthesized simultaneously. Note on Figure 2.5 (a) that the Coalition-Task Problem Synthesis process is outside the planning loop and does not require input from the Coalition-Task Plans. Loosely and tightly coupled tasks require serial plan synthesis, as described in Chapter 2.2. Robot coalitions take turns when planning, as shown in Figure 2.5 (b). The task planning problem's goals are concatenated with the goals of the next task planning problem in order to guarantee that the following task plan will not undo the goals achieved by the prior task plan. The *Coalition-Task Problem Synthesis* process incorporates the final state achieved by the latest coalition-task plan to generate the following coalition-task problem.

Multiple robot planning is largely an intractable problem, but assigning tasks to the most appropriate robot coalitions scales significantly better. The plan synthesis time can be orders of magnitude longer than the corresponding coalition



a) Coalition Formation then Planning for uncoupled tasks.

b) Coalition Formation then Planning for loosely and tightly coupled tasks.

Figure 2.5: Coalition Formation then Planning for (a) uncoupled and for (b) coupled tasks. Rounded filled shapes represent processes and rectangles represent data [62].

formation problems; thus, the overhead created by coalition formation is minimal. The problem complexity is reduced by generating multiple small-action-set plans. The reduced search branching factor permits derivation of plans for significantly larger problems [62]. The Coalition Formation then Planning framework is agnostic to the coalition formation algorithm adopted and uses external algorithms [151].

Coalition formation can scale planning to larger numbers of robots and more complex tasks, but results in poor quality plans, that have longer makespan than centralized planning (i.e., Planning Alone) [62]. The model of capabilities used by coalition formation does not reveal whether tasks are tightly coupled, limiting cooperation between coalitions allocated to different tasks and results in lower quality plans that require more actions and time to complete. Coalition formation does not guarantee a coalition will be able to accomplish assigned tasks. A coalition not capable of executing its assigned task is called *nonexecutable* and planning will fail because the actions of the allocated agents are not sufficient. Plan quality can be improved by partitioning the planning problem along coalition and task coupling lines [36]. The most tightly coupled coalition-tasks pairs are fused, whereas the most loosely coupled remain separate. When two coupled coalition-task planning problems are solved separately, the planner considers each tasks' goals individually, and produces potentially redundant action sequences [179]. However, when two coupled coalition-tasks are fused, the actions for one task can contribute to achieving states necessary to achieve another task and can generate higher quality plans. Planning uncoupled coalition-task planning problems together does not im-

prove plan quality, and often increases planning complexity. Task Fusion leverages coupling to improve coalition formation and planning.

2.3.3.4 Task Fusion

Coalition formation was enhanced with *Task Fusion* in order to account for tightly coupled tasks and generate higher quality plans at a lower computational cost [62]. After coalition formation, tightly coupled coalition-task pairs are fused into larger coalition-task pairs. The fused coalition-tasks plans can be synthesized faster and result in shorter makespan with fewer actions. Fusing allows the planner to address the tasks' mutual dependencies and facilitates cooperation between the fused coalitions. The result of Task Fusion over coalition-task pairs $p_i = \langle \Phi_i, v_i \rangle$ and $p_j = \langle \Phi_j, v_j \rangle$ is a fused coalition-task pair p_f, $p_f = \langle \Phi_f, v_f \rangle = F(p_i, p_j)$, where F is a mapping of pairs of coalition-task pairs $F : p_i \times p_j \rightarrow p_f \mid p_i, p_j \in P_m$, Φ_f is the union of robots $\phi \subseteq \Phi_i$ and $\phi \subseteq \Phi_j$, $\Phi_f = \Phi_i \cup \Phi_j$, and $v_f = v_i \wedge v_j$ is the conjunction of task requirements from v_i and v_j.

Task Fusion is the fusion of tasks and the assigned coalitions. Both coalitions and tasks are fused. Tasks are fused by concatenating the goals of the original tasks. Coalitions are fused by combining or taking the union of the members of the original coalitions. A coalition-task pair consists of a task and a coalition. Fusing a coalition results in a new coalition where the members of the original coalitions are combined. Fusing tasks results in a new task, where the goals of the original tasks are concatenated. Members from both coalitions will be considered

during plan generation, as the goals of both tasks must be satisfied by the resulting plan.

Coalition-task coupling is estimated by a heuristic that maps two coalition-task pairs p_i and p_j, to a coupling estimate, $H(p_i, p_j) : p_i \times p_j \rightarrow [0, 1]$, where $H(p_i, p_j) = 0$ indicates that p_i and p_j are uncoupled and $H(p_i, p_j) = 1$ indicates that p_i and p_j are tightly coupled. The Task Fusion algorithm stops when the ratio of fused coalitions, relative to the original number of coalitions, m, becomes greater than a user-defined threshold f_{max}, the fusion ratio, as presented in Algorithm 2.1. No coalition is fused when $f_{max} = 0$, and all coalitions are fused when $f_{max} = 1$. A zero fusion ratio, $f_{max} = 0$, is equivalent to the baseline Coalition Formation then Planning (i.e., no Task Fusion). Fusing all coalitions using $f_{max} = 1$ does not produce the grand coalition, because the algorithm is restricted to pair wise coalition fusion only, in order to avoid the combinatorial complexity of evaluating all possible coalition-task subsets. Fusing all coalitions can only produce the grand coalition when there are only two initial coalitions.

The previously developed heuristics estimate coalition-task coupling based on the coalition formation model of capabilities [62]. The *Coalition Similarity* (CS) heuristic, $\frac{|\Phi_i \cap \Phi_j|}{|\Phi_i \cup \Phi_j|}$, operates on coalition-task pairs that share common robots. Coalition-task pairs that have no common robots score 0 and coalition-task pairs that share all robots score 1. The *Coalition Assistance* (CA) heuristic, $\sum_{r=1}^{k} \frac{c_r^{\Phi_i \cup \Phi_j}}{max(c_r^{v_i}, c_r^{v_j})}$, estimates the ratio of coalition capabilities over task requirement capabilities after fusion, and prioritizes coalition-task pairs that share the same task requirement capabilities. These heuristics do not consider planning-related

Data: $P_m = \{p_1, p_2, \ldots, p_m\}$, a set of m coalition-task pairs;
$H(p_i, p_j) : p_i \times p_j \rightarrow [0, 1]$;
Result: A set of o coalition-task pairs $P_o = \{p_1, p_2, \ldots, p_o\}$.

```
1  Initialize empty set P_o = {∅};
2  Populate list l with all (m choose 2) pairs of coalition-task pairs ⟨p_i, p_j⟩, p_i, p_j ∈ P_m;
3  foreach pair ⟨p_i, p_j⟩ in list l do
4  |  Compute the heuristic value h_ij = H(p_i, p_j);
5  Sort list l relative to the heuristic value h_ij;
6  foreach pair ⟨p_i, p_j⟩ in list l do
7  |  Remove pair ⟨p_i, p_j⟩ from list l;
8  |  Remove all pairs containing p_i or p_j from list l and from set P_m;
9  |  Fuse pair ⟨p_i, p_j⟩ into coalition-task pair p_f = F(p_i, p_j);
10 |  Insert coalition-task pair p_f into set P_o;
11 |  if 2 · ||P_o|| > m · f_max then
12 |  |  break;
13 Return P_o = P_m ∪ P_o;
```

Algorithm 2.1: The Task Fusion Algorithm.

information, such as robots handling the same logical objects, or sharing the same physical location. Ignoring planning-related information limits the heuristic's accuracy, which can produce plans that take longer to execute and require a larger number of actions. Chapter 4 introduces a new family of Task Fusion heuristics that improve plan quality and cost.

2.3.4 Plan Merging

Serial plan synthesis, described in Chapter 2.2.1, addresses tightly coupled domains [179]; however, merging the independently generated plans can be a complex problem. Conflicts arise when one plan's actions can prevent the execution of another plan's actions. Most planners assume serial plan execution in order to simplify

the merging process and prevent conflicts between plans. Plans are executed serially, in the same order as they were synthesized; however, tasks cannot execute in parallel, resulting in long plan execution durations. Plan merging algorithms allow parallel task execution and can shorten the plan execution duration [52]. The Coalition Formation then Planning framework [62] employs a greedy merging algorithm that can result in long plan execution. The effectiveness of the Coalition Formation then Planning framework can be enhanced by employing more efficient plan merging algorithms.

The Multiagent Plan Coordination Problem (MPCP) formulation assumes a set of agents derived plans independently to achieve their individual goals [52]. Agents can undermine each other when executing their plans and must coordinate to produce conflict-free plans that guarantee all agents achieve their individual goals. The solution to an MPCP is composed entirely of actions drawn from the original agents' plans; no actions are added. A plan π can be defined by a tuple $\langle A, \prec_T, \prec_C \rangle$, where $A = \{a_1, a_2, \ldots\}$ is a set of actions, $\prec_T = \{\prec_1, \ldots\}$ and $\prec_C = \{\prec_1^c, \ldots\}$ are sets of temporal and causal orders on actions A, respectively. Actions are defined by the tuple $\langle pre(\cdot), eff(\cdot) \rangle$, where $pre(a)$ defines a set of preconditions that must hold during action a's execution, and $eff(a)$ defines the set of effects caused by action a's execution. A temporal order, $\prec$, is a binary relation $a_i \prec a_j$, with $a_i, a_j \in A$, and establishes that action a_j cannot be executed before action a_i's execution is completed. A causal order, $\prec^c$, is an extended temporal order that defines a ternary relation $a_i \prec^c a_j$, with $a_i, a_j \in A$, and c is a condition belonging to action a_i's effects ($c \in eff(a_i)$) and to action a_j's

preconditions ($c \in pre(a_j)$). Causal orders help identify plan conflicts, but do not impact the plan execution, and are reduced to regular temporal orders after the merging process completes.

Plan merging results in two types of conflicts [52]. Open preconditions occur when a precondition c of an action a is not satisfied by the existence of a causal order $a_0 \prec^c a$, indicating that an action a_0 establishes condition c to satisfy action a's precondition. An open precondition κ_o is denoted by the tuple $\langle a, c \rangle$, where a is an action and c is a precondition of action a, ($c \in pre(a)$). Causal conflicts occur when a causal order $a_i \prec^c a_j$ is threatened by the effects of an action a. The causal order $\prec^c$ is threatened if there exists an action a whose effects establish the negative condition $\neg c$, ($\neg c \in eff(a)$), and neither of the temporal orders $a \prec a_i$ and $a_j \prec a$ exist. A causal conflict implies that action a's effects can deny condition c and prevent action a_j to execute. A causal conflict κ_c is denoted by the tuple $\langle a, \prec^c \rangle$, where a is an action and $\prec^c$ is a causal order.

Open preconditions and causal conflicts are addressed by inserting causal and temporal orders, respectively. An open precondition of action a and condition c can be resolved by adding a causal order $a_0 \prec^c a$, such that condition c exists in the effects of action a_0 ($c \in eff(a_0)$) [52]. The solution forces action a_0 to establish the condition c for action a. A causal conflict of action a and causal order $a_i \prec^c a_j$ is resolved by adding temporal orders $a \prec a_i$ or $a_j \prec a$ and execute action a before action a_i, or after action a_j [52].

MPCP is NP-Complete and a tractable optimal algorithm is infeasible [52]. The Multiagent Plan Coordination by Plan Modification Algorithm (PMA) solves

MPCP problems and minimizes the resulting number of actions [52]. The PMA allows a set of actions to replace a single redundant action collectively, resulting in merged plans with fewer actions, but cannot scale to large numbers of agents and tightly coupled tasks.

2.3.5 Summary

The existing planning algorithms incorporate various features deemed necessary for real-world planning, but no single algorithm incorporates all the necessary features. The hybrid approaches, such as the Coalition Formation then Planning framework, improve the algorithm's ability to scale. However, these methods do not support uncertain action durations, fail to minimize the makespan, or cannot scale to large numbers of agents and tightly coupled tasks.

Chapter 3: Outline

The Coalition Formation then Planning framework relies on two key coordination processes, highlighted in Figure 3.1 by the dashed red outline. The two key processes, coalition formation and plan merging, allow agents to coordinate before and after the distributed planning process, respectively. However, these two processes suffer from limitations that hinder the framework's ability to scale and solve complex problems.

Coordination before planning, the first process highlighted in Figure 3.1, is reliant on the coalition formation models, which ignore planning-related knowledge, producing lower quality plans that take longer to execute and require a larger number of actions. New Task Fusion Heuristics (Chapter 4) incorporate planning-related knowledge in order to enhance coordination before planning and allow the Coalition Formation then Planning framework to produce higher quality plans.

Existing methods for coordination after planning, the second process highlighted in Figure 3.1, either fail to minimize makespan, or do not scale to large numbers of robots and tightly coupled tasks. New plan merging algorithms (Chapter 5) incorporate new conflict resolution models in order to enhance coordination after planning and allow the Coalition Formation then Planning framework to minimize the resulting makespan with formal quality guarantees, while scaling to a larger number of robots and tasks.

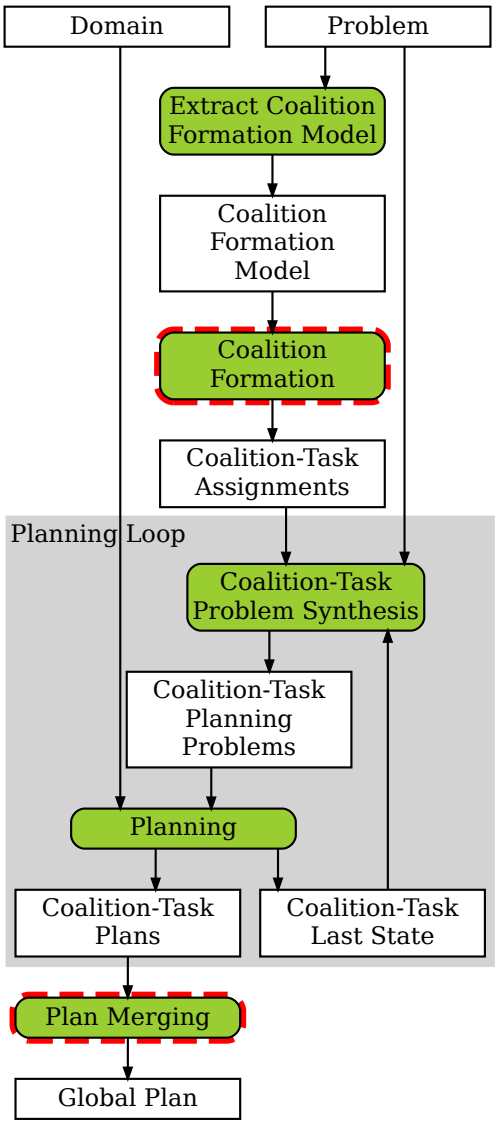


Figure 3.1: Coalition Formation then Planning Overview, where the key coordination processes are highlighted in dashed red, from Figure 2.5.

The original Coalition Formation then Planning framework incorporated durative actions in order to model temporal constraints and more accurately represent real-world domains; however, it did not address dynamic situations in which uncer-

tain action durations are required. Addressing this issue requires the incorporation of planners and planning description languages that accommodate uncertain action durations. Existing planning description languages cannot accommodate the needs of this research; thus, a new planning description language, capable to represent both the coalition formation models and the planning models, while supporting uncertain action durations, is needed.

3.1 Multiagent Actions Concurrency and Time Uncertainty Planning Language

The original Coalition Formation then Planning [62] framework was developed for deterministic planning models. Dukeman [61] extended PDDL to incorporate a coalition formation model into deterministic temporal planning and introduced the Multiagent Capabilities and Planning Domain Definition Language (MACPDDL). MACPDDL supports durative actions, but does not support uncertain action durations. Incorporating models with uncertain durations requires a new probabilistic problem description language that includes a coalition formation model. This dissertation introduces a new planning language, the Multiagent Actions Concurrency and Time Uncertainty Planning Language (MAPL) that incorporates a coalition formation model into stochastic temporal planning. MAPL, presented in Extended Backus-Naur Form in Appendix A, extends the Actions Concurrency and Time Uncertainty Planning Language (APL) [24]. Both APL and MAPL feature a relational C-Style syntax that is more readable than PDDL's parenthesized

Lisp-style syntax [105]. APL's "Problem" section was extended to include

i. A list of agents, independent from the list of non-agent objects;

ii. Tasks, to explicitly subdivide the goal requirements into task requirements;

iii. Agent capabilities, to declare the capabilities of each agent; and

iv. Task capabilities, to declare the capabilities required by each task.

The extensions permit integrating a full model of capabilities for coalition formation, and allow for deriving separate planning problems for each task and robot coalition. Deriving separate planning problems for each task is a necessary step of the Coalition Formation then Planning framework, in order to generate plans for each task separately and reduce computational complexity. Dukeman's [61] First Response Domain models problems where human-robot teams cooperate to rescue victims, collect hazardous objects, clear gas leaks, and clear blocked roads after a natural disaster, as described in Chapter 3.2.3. MACPDDL is restricted to deterministic domains and does not support uncertain action durations. The action durations are deterministic; thus, the rescue domain was simplified when represented using MACPDDL. Driving between locations was assumed to be a deterministic function of distance and the robot's velocity, as presented on line 4 of Listing 3.1.

Listing 3.1: A subset of the Rescue Domain in MACPDDL.

```
1  (:durative-action Drive
2      :executor (?r - Robot)
3      :parameters (?orig - Location ?dest - Location)
4      :duration (= ?duration (/ (Distance ?orig ?dest) (RobotVelocity ?r)))
5      :condition
6      (and
7          (at start (PosRobot ?r ?orig))
8          (at start (Road ?orig ?dest))
9      )
10     :effect
11     (and
12         (at end (PosRobot ?r ?dest))
13     )
14 )
```

The MAPL language supports uncertain action durations. Driving between locations is modeled as a probabilistic Gaussian function of distance and the robot's velocity, as presented on line 3 of Listing 3.2. A compiler, decompiler, and translator MAPL suite was implemented using the Earley algorithm [93], which can parse context-free grammars.

Listing 3.2: A subset of the Rescue Domain in MAPL.

```
1  action Drive(Robot r, Location orig, Location dest){
2      duration:
3          normal (Distance(orig,dest)/RobotVelocity(r)) DriveStd();
4      conditions:
5          @start: PosRobot(r) = orig;
6          @start: Road(orig,dest);
7      effects:
8          @end: PosRobot(r) = dest;
9  }
```

MACPDDL was used to support the Task Fusion experiments (Chapter 4) and MAPL was used to represent all domains used in the plan merging and conflict resolution experiments (Chapters 5 and 6), which incorporate uncertain action durations.

3.2 Experimental Domains

Benchmark problem domains are common in the automated planning literature for empirical algorithm evaluation and comparison. Three domains, the Blocks World Domain [78], the Logistics Domain [179], and the First Response Domain [62], were extended for this dissertation in order to model complex real world aspects more accurately. Two major types of extensions were made. The first set of major extensions introduced durative actions and continuous fluents, in order to model action durations and quantifiable aspects of the real world. The second set of major extensions introduced action duration uncertainty, in order to model the more dynamic and uncertain aspects of the real world.

3.2.1 Blocks World Domain

The Blocks World Domain is a common benchmark in the automated planning literature [78]. The original Blocks World Domain consists of one robot arm that manipulates and stacks piles of blocks. Problems consist of unstacking blocks onto an infinite size table, and re-stacking them into a goal piling configuration.

The original domain does not account for action durations and the blocks are homogeneous. Two major extensions were made to the Blocks World Domain in order to more accurately model complex real world aspects.

The Dukeman and Adams's Blocks World Domain extension [62], used for Task Fusion experiments (Chapter 4), introduced multiple robot arms, modeled a variety of end-effectors, and incorporated temporal constraints. A finite sized table holds stacks of blocks that require specific end-effectors. The start and goal states are represented via specific block stacks and block orders within the stacks. Each arm has a subset of available end effectors and each block requires a specific type of end effector. Blocks can be either single- or double-weight. Single-weight blocks can be manipulated by a single arm, while double-weight blocks require two arms. Each goal state stack generates a separate task, but the component blocks of different goal stacks can originate from a common stack, causing tight coupling between tasks. Each arm and end effector require different amounts of time to grasp, manipulate, and release blocks; thus, introducing durative actions. The time to stack and unstack blocks is also dependent on the end effector and the block's initial and final position positions, modeled with continuous fluents.

The second Blocks World Domain extension, developed for this dissertation, introduces action duration uncertainty, and was used for the experiments of Chapter 5. Each block type requires a stochastic amount of time to be manipulated, defined by a Gaussian distribution. This second extension builds upon the first extension and includes all of its features.

3.2.2 The Logistics Domain

The Logistics Domain [179], a benchmark multiagent domain, requires different truck types when delivering trailers to their destinations. Trailers must be exchanged between truck types before reaching their destinations, which requires coordination, as truck types are limited to subsets of a road system. Specific truck types cannot travel outside of their assigned districts, and different truck types must coordinate in order to deliver trailers across districts. The trailers' initial and goal locations are selected randomly and each trailer delivery is a separate task. Coupling exists within each task, as trucks need to coordinate actions in order to deliver a trailer. There is loose coupling between tasks, as each trailer delivery is independent. However, plans for separate tasks can become coupled when sharing the same trucks. Task plans that share common trucks have increased coupling, since each shared truck cannot execute all task actions simultaneously. Transport actions require the trucks to be positioned at specific locations in order to attach, detach, and transport trailers.

The Logistics Domain was extended for this dissertation in order to introduce durative actions, continuous distances, and model the travel time between locations, as a probabilistic function of distance. The extended domain was used for the experiments of Chapter 5. Durative actions are critical, as the travel time between locations varies significantly and impacts makespan.

3.2.3 First Response Domain

This First Response Domain [62] models disaster response problems that require coordinating heterogeneous human-robot teams. Human-robot teams cooperate to rescue victims, collect hazardous objects, clear gas leaks, and clear blocked roads after a natural disaster. Prescription drugs inside pharmacies and weapons at pawn shops must be secured to prevent looting and ensure civilian safety. Victim rescue tasks require a human to triage the victim. The resulting triage level determines how the victim is taken to a hospital, either guided an aerial robot or transported by a ground robot. The pawn shop cleanup tasks require a police officer to locate, clear, secure, and load the weapons into the police robot for transport to the police base. The pharmacy cleanup tasks require personnel to locate, clear, and secure all prescription drugs, including loading the drugs into a robot for transport to a hospital. The number of robots, victims, pawn shops, pharmacies, road blocks, gas leaks, and waypoints was drawn from a uniform distribution. Plans are generated for both robots and humans. Traveling across the environment and performing each task requires significantly different amounts of time, making continuous and temporal constraints critical to a plan's successful execution. Similar to the Logistics Domain, there is loose coupling between tasks. However, plans for separate tasks can become coupled when sharing the same robots, and the coupling between two tasks becomes stronger when there is overlap between the locations the robots must traverse, due to the shared effects of cleared blocked roads.

Two major extensions were made to the First Response Domain for this disser-

tation in order to permit more complex, but realistic problems. The first extension introduces additional numerical fluents that more accurately model the complexities of the domain, and was used for the Task Fusion experiments (Chapter 4). The robot batteries drain as a function of robot activity over time. As well, the robot load is a numerical fluent; thus, allowing robots to carry a varying number of objects, dependent on the individual robot load capacity.

The second extension introduces action duration uncertainty that model the travel time between locations, as a probabilistic Gaussian function of distance and the individual robot velocities, and was used for the plan merging experiments (Chapter 5). Further, the triaging of victims and the pawn shop and pharmacy cleanup tasks require using a shared aerial support service, which introduces tighter coupling between tasks.

The First Response Domain was reduced to victim rescue and hazardous material removal tasks for the multiple robot experiments of Chapter 6, as the limited number of robots and the limited robots' capabilities prevented the use of the full domain. Three types of robots cooperate to achieve these tasks: rescuer, ambulance, and hazard collector. Action duration uncertainty, the continuous model of distances, and the individual robot velocities remained. The rescuer robots, together with the ambulance and hazard collector robots, are responsible for the respective victim rescue and hazardous material removal tasks. The victim rescue tasks require robots to "load" victims for transport to a rescue base. The hazardous materials collection tasks require robots to clean up and collect hazardous materials, which requires "loading", transporting, and "unloading" hazards into a

safe container.

The rescuer robot is responsible for inspecting and clearing each area in order to ensure the area is safe for of the ambulance and hazard collection robots. Victim rescue tasks require a rescuer robot to "load" a victim onto an ambulance robot that transports the victim to a rescue base and "unloads" the victim at that location. Hazardous materials collection tasks require a rescuer robot to "load" hazardous materials onto a hazard collector robot that transports the hazardous materials to a safe container near the rescue base and "unloads" the hazardous materials onto a safe container.

Chapter 4: Plan Distance Heuristics for Task Fusion in Distributed Temporal Planning

The Coalition Formation then Planning framework employs coalition formation and Task Fusion algorithms in order to coordinates robots' actions before planning. The existing Task Fusion heuristics are ineffective and result in limited quality plans that take longer to execute and require a larger number of actions.

Heuristics for Task Fusion can become more accurate and achieve better planning results by incorporating an estimation of plan distance. Plan distance metrics were developed to quantify solution diversity in plan synthesis and can estimate the level of similarity between two plans [167]. Nguyen et al. [116] formulate a distance function between two plans π_i and π_j, that maps to a real-valued distance metric $\delta(\pi_i, \pi_j) : \pi_i \times \pi_j \rightarrow [0, 1]$. The action plan distance metric is defined by $1 - \frac{|A(\pi_i) \cap A(\pi_j)|}{|A(\pi_i) \cup A(\pi_j)|}$, where $A(\pi)$ is the set of actions in plan π [116]. The opposite of plan distance, plan similarity, can be approximated by $1 - \delta(\pi_i, \pi_j)$, changing the action plan distance metric to $\frac{|A(\pi_i) \cap A(\pi_j)|}{|A(\pi_i) \cup A(\pi_j)|}$. The similarity between plans can be a proxy for estimating the level of coupling between two coalition-task planning problems. Problems that produce similar plans can be considered more tightly coupled.

Plan distance heuristics can use the plan's logical objects, in addition to the plan's actions. Logical object instances are extracted from the plan actions' ar-

gument lists in order to reveal problem details that otherwise are ignored when only actions are considered. The overlap of actions and logical objects between two plans indicates the plans' level of similarity and coupling. Higher overlap of actions and logical objects indicates that the robots interact with common objects and navigate through common locations, which are represented as logical objects. The use of action *sets* makes Nguyen et al.'s heuristic unaware of repeated action instances. *Lists* allow and account for repeated actions, revealing nuances that are otherwise omitted. This dissertation introduces a family of plan distance heuristics that use lists of plan's actions and logical objects to estimate coupling and generate better planning results with Task Fusion.

The introduced plan distance heuristics consider the overlap of actions and logical object occurrences between two plans in order to estimate coupling [101]. The *Object heuristic* (O), the *Action heuristic* (A), and the *Action-Object heuristic* (AO), are based on overlaps in the action, object, and both action and object occurrences, respectively. The time at which each action is scheduled to occur is used to extend each heuristic into three temporal variants: the *Object-Temporal heuristic* (OT), the *Action-Temporal heuristic* (AT), and the *Action-Object-Temporal heuristic* (AOT).

A plan π consists of a list of actions, where each action entry contains a start time τ, robots $\Phi = \{\phi_1, \ldots\}$, and planning-model first-order logic objects $O = \{o_1, \ldots\}$. Plan distance heuristics compile a list of logical object and action occurrences, extracted from each plan action entry. Each action-object occurrence, tagged with the associated action start time, τ, populates the action-object list,

$L = \{\langle l_1, \tau_1 \rangle, \ldots\}$. The similarities between plans π_i and π_j result in an estimate for the utility of fusing coalition-task pairs p_i and p_j. Let π_i and π_j represent the plans for coalition-task pairs p_i and p_j, respectively. A plan distance heuristic is a function $H(\pi_i, \pi_j) : \pi_i \times \pi_j \rightarrow [0, 1]$ that maps to a utility value. Plan distance heuristics require synthesizing plans π for all m coalition-task pairs p, but can leverage the details from plans that are unavailable via the capabilities or coalition structures from the coalition formation model. The heuristics are agnostic to the origin of the plans adopted and leverage existing planners.

4.1 Object, Action, and Action-Object Heuristics

The *Object* (O), *Action* (A), and *Action-Object* (AO) heuristics represent the level of overlap between the logical object and action occurrences in plans π_i and π_j for coalition-task pairs p_i and p_j, respectively: $H(p_i, p_j) = \frac{1}{|L_i| \cdot |L_j|} \cdot \sum_{l_i \in L_i} \sum_{l_j \in L_j} (l_i = l_j)$, where $|L_i|$ and $|L_j|$ are list sizes for action-object lists L_i and L_j, respectively. The list elements l represent objects for the Object heuristic, actions for the Action heuristic, and both objects and actions for the Action-Object heuristic. All pairs of entries from both action-object lists are compared. Each heuristic variant populates the plan lists, L_i and L_j. The *Object heuristic* populates lists with logical object occurrences; the *Action heuristic* populates lists with action occurrences; and the *Action-Object heuristic* populates lists with both action and logical object occurrences. The normalizing fraction ensures that the heuristic values are between $[0, 1]$, where 1 indicates maximal task coupling.

A simple first response example is provided. Assume two coalition-task pairs, p_A and p_B, have plans π_A and π_B, respectively. The *move* (w_x, w_y) action moves a robot from a location w_x to a location w_y, whereas the *triage* (v, w) action triages a victim v in location w. The respective plan actions are $\{$*move* (w_0, w_1), *triage* $(v_1, w_1)\}$ and $\{$*move* (w_0, w_1), *move* (w_1, w_2), *triage* $(v_2, w_2)\}$. The Action lists are $L_A = \{$*move*, *triage*$\}$ and $L_B = \{$*move*, *move*, *triage*$\}$, resulting in three matches and producing an Action heuristic value of $H\ (p_A, p_B) = 0.500$, due to the normalization factor ($|L_A| = 2$, $|L_B| = 3$, and $|L_A| \cdot |L_B| = 6$). The Object lists are $L_A = \{w_0,\ w_1,\ v_1,\ w_1\}$ and $L_B = \{w_0,\ w_1,\ w_1,\ w_2,\ v_2,\ w_2\}$, resulting in five matches and producing an Object heuristic value of $H\ (p_A, p_B) = 0.208$, due to the normalization factor ($|L_A| = 4$, $|L_B| = 6$, and $|L_A| \cdot |L_B| = 24$). The Action-Object lists are $L_A = \{$*move*, *triage*, $w_0,\ w_1,\ v_1,\ w_1\}$ and $L_B = \{$*move*, *move*, *triage*, $w_0,\ w_1,\ w_1,\ w_2,\ v_2,\ w_2\}$, resulting in six matches and producing an Action-Object heuristic value of $H\ (p_A, p_B) = 0.111$, due to the normalization factor ($|L_A| = 6$, $|L_B| = 9$, and $|L_A| \cdot |L_B| = 54$). Lists allow repeated entries and account for higher coupling, as demonstrated by the repeated use of the action *move* by plan π_B.

4.2 Object-Temporal, Action-Temporal, and Action-Object-Temporal Heuristics

The *Object-Temporal* (OT), *Action-Temporal* (AT), and *Action-Object-Temporal* (AOT) heuristics integrate temporal dependencies in order to account for action

and object interactions at different times throughout the plan. Each heuristic variant populates the plan lists, L_i and L_j, with object occurrences, action occurrences, or both, as was the case in Chapter 4.1. The temporal heuristics weight each matching list entry with a decaying exponential weighting factor. The weighting ranks pairs that interact with the same objects at similar times higher than pairs that interact with the same objects at different times. The weighting factor is a function of the time difference between each matching list entry: $H(p_i, p_j) = \frac{1}{|L_i| \cdot |L_j|} \cdot \sum_{l_i \in L_i} \sum_{l_j \in L_j} (l_i = l_j) \cdot e^{-|\tau_i - \tau_j|}$, where τ_i and τ_j are temporal timestamps for list entries l_i and l_j, respectively. If $\Delta\tau = |\tau_i - \tau_j| = 0$, (i.e., the object matching occurs at the same time), the weighting factor is 1. If $\Delta\tau \to \infty$, (i.e., the object matching occurs at different times), the weighting factor is 0.

Drawing from the example in Chapter 4.1, assume the action *triage* (v_1, w_1) was scheduled to execute in plan π_A at time $\tau_i = 10$ minutes, whereas the action *triage* (v_2, w_2) was scheduled to execute in plan π_B at time $\tau_j = 12$ minutes. The time difference between the two actions is $|\tau_i - \tau_j| = 2$ minutes and the temporal weighting factor is $e^{-|\tau_i - \tau_j|} = e^{-2} = 0.607$, causing a 39.3% reduction in the action match contribution.

Generating full plans to estimate coalition-task coupling can be prohibitively costly, and defeat the purpose of Task Fusion; however, relaxed plans can replace full plans for coalition-task coupling estimation. A relaxation of the problem model, such as to ignore actions' negative effects, can significantly reduce the computational complexity [83]. Relaxed plans offer a rough approximation of the actual plans and are used to inform forward search [83], reachability analysis [31], and

distance metrics [50]. Relaxed plans can provide an estimate of the actions and the involved logical objects required by the full plan, yet require significantly less computation.

The heuristics use plan distance to estimate coupling and inform coalition formation. Relaxed plans allow evaluating efficiently the planning elements of coalition-task pairs before planning. Coupling across coalition-task pairs is estimated by the relaxed plans' actions and logical objects, and the most coupled coalition-task pairs are fused.

4.3 Empirical Evaluation

The heuristics were evaluated for two different domains chosen to model the complexity of planning for multiple heterogeneous robot systems. Continuous fluents and temporal constraints allow modeling the numerical and temporal constraints necessary for each domain. Heterogeneous multiple robot planning problems with continuous fluents and temporal constraints are yet unavailable in existing standard planning problem benchmarks. Heterogeneous robot systems contain robots with subsets of the capabilities necessary to accomplish each task, and require robots to cooperate. Complex problems with heterogeneous robot capabilities generate tightly coupled tasks. Ten coalitions of robots and ten missions were randomly generated and combined to form 100 problems per domain. Each coalition generated ten problems, one for each mission, and each mission generated ten problems, one for each coalition. The resulting plans were evaluated based on the plan outcome,

makespan, number of actions, processing time, and memory usage.

4.3.1 Domains

4.3.1.1 Blocks World Domain

The first Blocks World Domain extension, which introduces numerical fluents, multiple robot arms, and multiple block types, described in Chapter 3.2.1, was used with four types of end effectors: friction, suction, magnetic, and encompass. Ten coalitions with a minimum of four robot arms and a maximum of eight robot arms were generated, as shown in Table 4.1. Ten missions with a minimum of eleven tasks and a maximum of 24 tasks were generated, as presented in Table 4.2.

Table 4.1: Coalition Composition for the Blocks World Domain [62].

	Grand coalition									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Robot arms	6	8	5	5	4	8	7	7	7	8
Friction end effector	2	3	2	4	2	5	5	5	6	6
Suction end effector	3	7	3	2	2	6	3	5	6	7
Encompass end effector	4	7	4	4	2	8	4	4	4	3
Magnetic end effector	4	3	2	2	3	5	7	2	6	7
Total end effectors	13	20	11	12	9	24	19	16	22	23

4.3.1.2 First Response Domain

The first First Response Domain extension, which introduced additional numerical fluents, described in Chapter 3.2.3, was used. Ten coalitions with a minimum of

Table 4.2: Tasks per mission for the Blocks World Domain [62].

	Mission									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Position Friction Block	7	2	8	9	2	5	7	5	0	3
Position Suction Block	6	6	3	4	5	2	5	6	2	2
Position Encompass Block	1	3	5	8	4	5	3	6	6	3
Position Magnetic Block	4	0	3	3	0	3	2	5	8	4
Total Tasks	18	11	19	24	11	15	17	22	16	12

15 robots and a maximum of 21 robots were generated, with each coalition having a minimum of 1 robot and a maximum of 6 robots per robot type, as shown in Table 4.3. Ten missions with a minimum of 13 tasks and a maximum of 24 tasks were generated, with each mission having a minimum of 1 task and a maximum of 15 tasks per task type, as presented in Table 4.4.

Table 4.3: Coalition Composition for the First Response Domain.

		Coalition									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Human	**Rescuer**	2	4	2	2	3	4	2	2	3	3
	Police	2	1	1	1	2	1	2	1	1	2
	Firefighter	1	2	1	1	2	2	1	1	2	2
	Total Humans	5	7	4	4	7	7	5	4	6	7
Robot	**Rescuer**	6	6	5	4	6	6	4	6	5	5
	Police	1	1	2	1	2	1	1	2	2	1
	Firefighter	4	5	3	5	3	4	3	5	5	4
	Quadrotor	1	2	2	1	2	1	2	1	1	2
	Total Robots	12	14	12	11	13	12	10	14	13	12
Total Agents		17	21	16	15	20	19	15	18	19	19

Table 4.4: Tasks per mission for the First Response Domain.

	Mission									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Clear gas leaks	1	2	2	1	2	1	2	1	1	2
Clear road blocks	4	5	3	4	5	4	4	3	4	5
Rescue victims	12	12	5	15	5	10	5	15	14	12
Clear pawn shops	2	1	2	2	1	2	2	1	1	1
Clear pharmacies	1	2	1	2	2	1	1	2	2	2
Total Tasks	20	22	13	24	15	18	14	22	22	22

4.3.2 Experimental Design

The experiment's independent variables are the specific planning methods: Planning Alone (PA), Coalition Formation then Planning (CFP), and Coalition Formation then Planning with Task Fusion, using the plan distance heuristics: Object (O), Action (A), Action-Object (AO), Object-Temporal (OT), Action-Temporal (AT), and Action-Object-Temporal (AOT), and the baseline heuristics: Coalition Assistance (CA) and Coalition Similarity (CS). The planning outcomes are: Success, a valid plan is produced; Nonexecutable, no plan can be derived for the task given the coalition's composition and allocated tasks; Time Fail, the time limit is exceeded; and Memory Fail, the memory limit is exceeded. The fusion ratio, f_{max}, limits the number of fused coalition-task pairs and can impact the effectiveness of Task Fusion. Fusion ratio values were chosen to uniformly cover the valid $[0, 1]$ range, $f_{max} = \{0.25, 0.50, 0.75, 1.00\}$. Each experiment's planning time was capped at one hour and memory usage was limited to 120 GB. These processing time and memory limits emulate the demands for real-world systems that require timely solutions on computers with limited memory.

The dependent variables are the planning outcome, makespan, number of actions, processing time, and memory usage. Makespan represents plan length, measured in seconds [142]. The number of actions is the total number of actions required by the plan to accomplish a task [142]. The processing time, measured in minutes, is the time required to solve a problem, which includes the coalition formation, processing heuristics, planning for all tasks, and merging each task plan into a final plan. The memory usage is the limit on memory allocated, in GB. The makespan and the number of actions metrics indicate plan quality. Higher quality plans have lower makespan and fewer of actions; thus, higher quality plans achieve their goals faster and require fewer actions. Plans with lower processing time and memory usage require fewer computing resources.

The TFD [73] and COLIN [49] planners support temporal constraints and continuous fluents, and were adopted for the Blocks World Domain experiment. A dynamic programming coalition formation algorithm was used [154]. COLIN [49] is the only continuous planner that accommodates the time-varying continuous fluents required for the First Response Domain. Robot Allocation through Coalitions using Heterogeneous Non-Cooperative Agents (RACHNA) [187], a market-based coalition formation algorithm, was used for the First Response Domain experiment. The relaxed plans were generated by a relaxed COLIN planner, which removes actions' delete effects [49].

The experiments were performed on an Intel Xeon CPU E5-1630 v4 @ 3.70 GHz $\times$ 8 workstation with 128 GB memory, running Ubuntu 14.04.5 LTS with the 4.4.0-89-generic Linux kernel. Third party coalition formation and planning

systems were compiled using the gcc/g++ compiler version 5.4.0.

Plan distance heuristics aim to estimate coupling and provide higher quality plans requiring fewer computational resources. The first hypothesis (H_1) is that the effectiveness of Task Fusion is affected by the heuristics used. The second hypothesis (H_2) is that the object oriented plan distance heuristics: Object, Action-Object, Object-Temporal, and Action-Object-Temporal, will outperform the baselines: the Coalition Assistance and Coalition Similarity heuristics, CFP, and Planning Alone. The third and fourth hypothesis are that the object oriented plan distance heuristics will result in better quality plans (H_3) and will require lower computational cost than the baseline approaches (H_4).

4.4 Results

The results are presented by problem domain and planner. Method quality and cost are represented for multiple metrics. High quality methods minimize the plans' makespan and number of actions, while low cost methods minimize processing time and memory usage. The concepts of Pareto Dominance and Pareto Strength [205] were adopted for comparing methods across these metrics. Method t_1 *dominates* method t_2 if all of t_1's metrics' means are better than t_2's. Specifically, t_1's quality dominates t_2's quality if both t_1's mean makespan and mean number of actions are better than the mean makespan and mean number of actions for t_2. The *Pareto Strength* of a method t_i is determined by the number n of methods $t_1, t_2, \cdots, t_n$ that t_i dominates. Methods with higher *Pareto Strength* dominate many other methods.

4.4.1 The Blocks World Domain with TFD

The Blocks World Domain with TFD planning was characterized by a positive relationship between the evaluated metrics and the Task Fusion ratio f_{max}. Most heuristics offered increasingly better success rates, plan quality, and computational cost for larger f_{max} values. The improved performance saturates at high f_{max} values, with virtually equivalent results being obtained for $f_{max} = 0.75$ and 1.00 across the heuristics.

The Coalition Assistance heuristic ($f_{max} = 0.75$ and 1.00) had the best planning success rates (42% and 41%, respectively) followed by Planning Alone (40%), and the Object heuristic ($f_{max} = 0.75$ and 1.00, both 39%), as shown in Table 4.5. Planning Alone had the highest rate of time failures (60%), followed by the Coalition Similarity ($f_{max} = 0.75$ and 1.00, both 53%) and Coalition Assistance ($f_{max} = 0.75$ and 1.00, both 52%) heuristics. CFP produced the highest rate of nonexecutable coalitions (34%). No method exceeded the 120 GB memory limit.

Planning Alone (PA), the Object (O) and Coalition Assistance (CA) heuristics produced the highest success rates, as shown in Figure 4.1 (a). The Object heuristic produced the second highest success rates for $f_{max} = 0.25$ and 0.50, whereas the Coalition Assistance heuristic produced the highest success rates for $f_{max} = 0.75$ and 1.00. The Coalition Assistance heuristic, however, resulted in mediocre makespan, number of actions, and memory usage results for all f_{max} values, as presented in Figure 4.1 (b), (c), and (e), respectively. Planning Alone resulted in the

Table 4.5: Blocks World with TFD planning results by method, f_{max}, and percentage for successfully generating a plan, nonexecutable coalition, and no plan generated due to time failure or memory failure.

Method	f_{max}	**Success**	**Nonexecutable**	**Time Fail**
Object	0.25	38	26	36
	0.50	33	29	38
	0.75	39	16	45
	1.00	39	16	45
Action	0.25	29	29	42
	0.50	32	25	43
	0.75	37	13	50
	1.00	37	13	50
Action-Object	0.25	32	29	39
	0.50	28	28	44
	0.75	34	15	51
	1.00	34	15	51
Object-Temporal	0.25	35	28	37
	0.50	31	28	41
	0.75	36	17	47
	1.00	36	17	47
Action-Temporal	0.25	23	32	45
	0.50	31	23	46
	0.75	36	15	49
	1.00	35	16	49
Action-Object-Temporal	0.25	28	32	40
	0.50	30	25	45
	0.75	36	14	50
	1.00	35	15	50
Coalition Similarity	0.25	23	33	44
	0.50	20	31	49
	0.75	27	20	53
	1.00	27	20	53
Coalition Assistance	0.25	36	22	42
	0.50	28	23	49
	0.75	42	6	52
	1.00	41	7	52
CFP	N/A	24	34	42
Planning Alone	N/A	40	0	60

best makespan (Figure 4.1 b), but among the worst number of actions (Figure 4.1 c), processing time (Figure 4.1 d), and the worst memory usage (Figure 4.1 e).

The Action-Object (AO) heuristic produced the second best makespan and the best number of actions for f_{max} = 0.50-1.00, as presented in Figure 4.1 (b) and (c), respectively. The Action-Object-Temporal (AOT) heuristic produced the second best number of actions for f_{max} = 0.50-1.00 (Figure 4.1 c) and resulted in the best processing time and memory usage across all f_{max} values, as shown in Figure 4.1 (d) and (e), respectively. The Coalition Similarity (CS) heuristic resulted in the second worst success rates and makespan for f_{max} = 0.75 and 1.00 (Figure 4.1 a and b), and third worst memory usage for all f_{max} values (Figure 4.1 e). CFP produced among the worst success rates, makespan, number of actions, memory usage, and among the worst processing time.

The Pareto Strength *quality*, which minimizes plans' makespan and number of actions, was evaluated across all methods. The Action-Object heuristic (f_{max} = 1.00 and 0.75) produced the best and second best plan quality (Pareto Strengths 32 and 31, respectively), followed by the Action-Object-Temporal heuristic (f_{max} = 1.00 and 0.75), which had the third and fourth best quality (Pareto Strengths 30 and 29, respectively). The Action (f_{max} = 0.25), Object (f_{max} = 0.25), and Coalition Similarity (f_{max} = 0.50) heuristics produced the lowest quality (Pareto Strength 0). The Pareto Strength *cost* minimizes processing time and memory usage. The Action-Object-Temporal heuristic (f_{max} = 0.75 and 1.00) resulted in the two lowest costs (Pareto Strengths 33 and 32). Planning Alone (PA), CFP, and the Coalition Similarity heuristic (f_{max} = 0.50) had the highest cost (Pareto Strength 0). The complete Blocksworld with TFD Pareto Analysis results for quality and cost are presented in Appendix Chapter B.1.

The Action-Object-Temporal heuristic is the best solution to the Blocks World Domain with TFD, as it resulted in among the best makespan and number of actions; and the best processing time, and memory usage. The Action-Object heuristic is the second best, as it resulted in the best quality, but mediocre processing time and memory usage. CFP is the worst solution, resulting in the worst metrics.

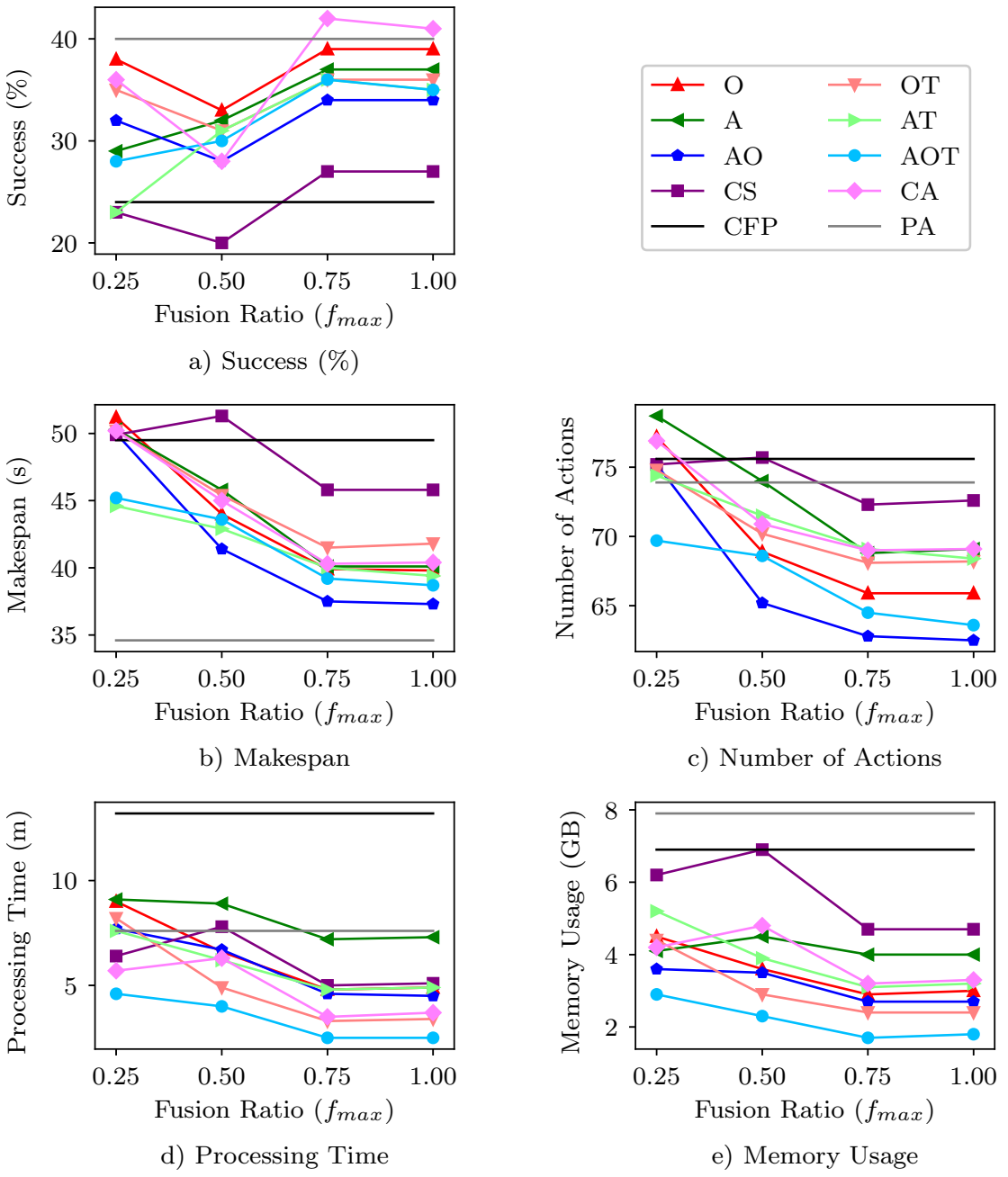


Figure 4.1: Blocks World with TFD (a) success, (b) makespan, (c) number of actions, (d) processing time, and (e) memory usage by fusion ratio (f_{max}). Samples were connected to facilitate visualization.

4.4.2 The Blocks World Domain with COLIN

The object oriented plan distance heuristics also offered the best solution to the Blocks World Domain with the COLIN planner. The top five success rates were achieved by the plan distance heuristics, whereas only two of the top five best success rates were plan distance heuristics when using TFD. The Object heuristic presented better results with increasing f_{max} values.

The Object heuristic (f_{max} = 0.75 and 1.00) had the best success rate (both 55%), as presented in Table 4.6. The Object (f_{max} = 0.25, 53%) and Object-Temporal (f_{max} = 0.75 and 1.00, both 52%) heuristics were second and the third best, respectively. Planning Alone had zero nonexecutable coalitions, but had the worst success (28%), time failure (42%), and memory failure (30%) rates. CFP produced the most nonexecutable coalitions (34%).

The Object (O) heuristic resulted in the highest success rates across all f_{max} values, as presented in Figure 4.2 (a), whereas Planning Alone (PA) produced the worst. PA produced the best makespan (Figure 4.2 b), but among the worst number of actions (Figure 4.2 c). The Object heuristic produced the second best makespan, the best number of actions, and the second best processing time and memory usage for f_{max} = 0.50 through 1.00, as shown in Figure 4.2 (b-e). The Coalition Similarity (CS) heuristic generated the lowest cost (Figure 4.2 d and e), but also produced among the worst success rates and the worst makespan, across all f_{max} values (Figure 4.2 a-c).

All heuristics produced their maximum success rates for the highest f_{max} values,

Table 4.6: Blocks World with COLIN planning results.

Method	f_{max}	**Success**	**Nonexecutable**	**Time Fail**	**Mem Fail**
Object	0.25	53	21	10	16
	0.50	48	16	13	23
	0.75	55	10	17	18
	1.00	55	10	17	18
Action	0.25	38	29	17	16
	0.50	41	22	13	24
	0.75	47	15	22	16
	1.00	47	15	22	16
Action-Object	0.25	49	23	12	16
	0.50	46	19	13	22
	0.75	50	13	21	16
	1.00	50	13	20	17
Object-Temporal	0.25	48	24	14	14
	0.50	47	16	14	23
	0.75	52	11	20	17
	1.00	52	11	19	18
Action-Temporal	0.25	43	22	18	17
	0.50	43	16	17	24
	0.75	45	10	29	16
	1.00	45	10	29	16
Action-Object-Temporal	0.25	46	25	13	16
	0.50	45	19	11	25
	0.75	47	12	23	18
	1.00	47	12	22	19
Coalition Similarity	0.25	37	27	17	19
	0.50	38	21	17	24
	0.75	43	16	22	19
	1.00	43	16	22	19
Coalition Assistance	0.25	44	27	17	12
	0.50	39	25	20	16
	0.75	50	15	22	13
	1.00	49	15	23	13
CFP	N/A	38	34	12	16
Planning Alone	N/A	28	0	42	30

0.75 and 1.00, as presented in Figure 4.2 (a), and resulted in monotonically better makespan for larger f_{max} values, as presented in Figure 4.2 (b), meaning that greater f_{max} values resulted in higher success rates for all f_{max} values evaluated.

The Object heuristic produced monotonically better makespan, number of actions, processing time, and memory usage for greater f_{max} values, as presented in Figures 4.2 (b-e). The Object heuristic is the best solution to the Blocks World Domain with COLIN, as it resulted in the best quality and second lowest cost across $f_{max} = 0.50, 0.75$, and 1.00.

The Object heuristic ($f_{max} = 0.75$ and 1.00) produced the best and second best plan quality (both Pareto Strength 30). The Action-Object heuristic ($f_{max} = 0.75$ and 1.00) produced the third and fourth best plan quality results (both Pareto Strength 28), followed by the Object ($f_{max} = 0.50$) and Object-Temporal ($f_{max} = 0.75$ and 1.00) heuristics (all Pareto Strength 24). The Coalition Similarity heuristic ($f_{max} = 0.25$) produced the lowest plan quality (Pareto Strength 0), while the Coalition Similarity heuristic ($f_{max} = 0.50$) was slightly better (Pareto Strength 1). The Coalition Similarity heuristic produced the three lowest cost results (Pareto Strengths 33, 31, and 30), with the best result being for the lowest f_{max} value. Planning Alone and the Action heuristic ($f_{max} = 0.50$) produced the worst cost results (both Pareto Strength 0). The complete Blocksworld with COLIN Pareto Analysis results for quality and cost are presented in Appendix Chapter B.2.

The Object heuristic is the best solution to the Blocks World Domain with COLIN, as it resulted in the best overall quality and second lowest cost results across $f_{max} = 0.50, 0.75$, and 1.00. The Object heuristic offered the best success rate and the best plan quality at the fourth lowest cost. The Action heuristic offered the worst solution, with the worst cost, and among the worst success rates.

The Coalition Similarity heuristic had the lowest cost across all f_{max} values, but produced the worst quality plans with among the lowest success rates. Planning Alone had the best makespan, but resulted in the worst success rate and required the highest cost.

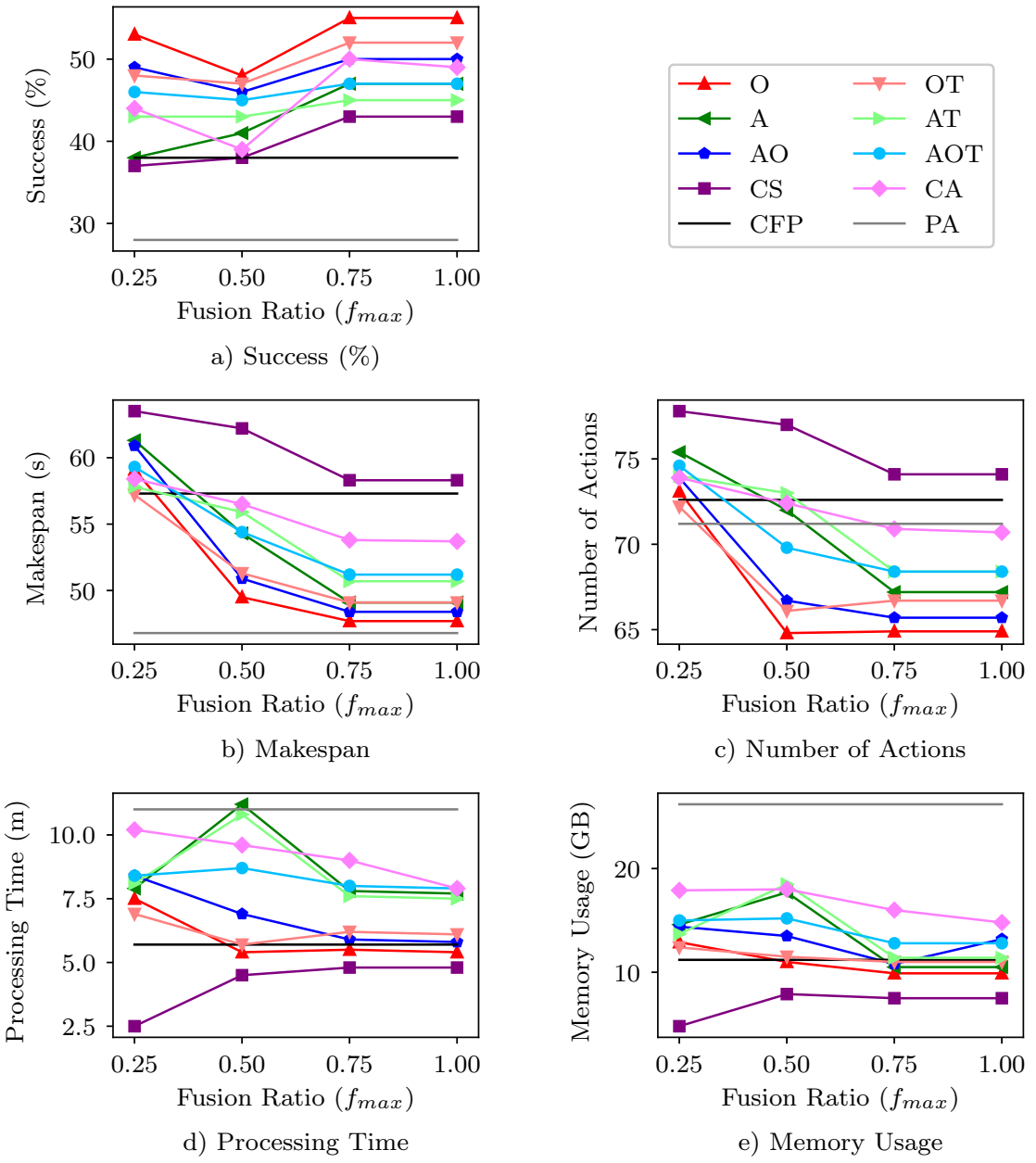


Figure 4.2: Blocks World with COLIN (a) success, (b) makespan, (c) number of actions, (d) processing time, and (e) memory usage by fusion ratio (f_{max}). Samples were connected to facilitate visualization.

4.4.3 The First Response Domain

The more complex First Response Domain presented noisier results, compared to the Blocks World Domain. Planning Alone exceeded the processing time limit for all problems, resulting in no plans. The plan distance heuristics produced better results for intermediary f_{max} values, whereas the baseline heuristics, Coalition Similarity and Coalition Assistance, performed better for lower f_{max} values. Monotonically worsening success, makespan, number of actions, and memory usage were observed for larger f_{max} values for most baseline methods, while most plan distance heuristics presented convex curves.

The Coalition Similarity heuristic ($f_{max} = 0.25$) produced the best planning success rate (73%), as shown in Table 4.7. The Coalition Similarity ($f_{max} = 0.50$) and Action-Object-Temporal ($f_{max} = 0.25$) heuristics were the second best (both 66%), followed closely by the Object-Temporal heuristic ($f_{max} = 0.50$, 65%). The Object heuristic ($f_{max} = 0.25$) produced the highest rate of nonexecutable coalitions (35%), the Coalition Assistance heuristic ($f_{max} = 1.00$) produced the highest rate of time failures (56%), and the Object heuristic ($f_{max} = 0.75$) produced the most memory failures (10%).

Many methods performed best for the intermediary f_{max} values, 0.50 and 0.75, in the First Response Domain, whereas most methods generated best results for the boundary f_{max} values, 0.25 and 1.00, in the Blocks World Domain. The Object (O) and Object-Temporal (OT) heuristics produced their best success rates for $f_{max} = 0.50$, as shown in Figure 4.3 (a), and produced their best makespan,

Table 4.7: First Response planning results.

Method	f_{max}	**Success**	**Nonexecutable**	**Time Fail**	**Mem Fail**
Object	0.25	56	35	8	1
	0.50	61	25	10	4
	0.75	33	24	33	10
	1.00	44	10	45	1
Action	0.25	51	20	29	0
	0.50	33	20	42	5
	0.75	29	20	51	0
	1.00	33	12	52	3
Action-Object	0.25	57	30	10	3
	0.50	59	20	21	0
	0.75	42	20	34	4
	1.00	59	10	29	2
Object-Temporal	0.25	59	24	13	4
	0.50	65	22	12	1
	0.75	43	28	23	6
	1.00	43	10	47	0
Action-Temporal	0.25	58	30	9	3
	0.50	57	22	19	2
	0.75	44	20	36	0
	1.00	39	10	51	0
Action-Object-Temporal	0.25	66	20	11	3
	0.50	62	23	13	2
	0.75	48	17	35	0
	1.00	34	11	55	0
Coalition Similarity	0.25	73	20	4	3
	0.50	66	16	16	2
	0.75	44	26	28	2
	1.00	47	18	32	3
Coalition Assistance	0.25	46	23	31	0
	0.50	41	18	40	1
	0.75	31	15	54	0
	1.00	31	13	56	0
CFP	N/A	62	32	5	1

number of actions, processing time, and memory usage for $f_{max} = 0.75$, as indicated in Figures 4.3 (b-e), respectively. The Action-Object (AO) heuristic produced its best success rate, processing time, and memory usage for $f_{max} = 0.50$, as shown in Figures 4.3 (a), (d), and (e), respectively, and produced its best

makespan and number of actions for $f_{max} = 0.75$, as shown in Figures 4.3 (b) and (c), respectively. The success rates produced by the Action-Temporal (AT), Action-Object-Temporal (AOT), and Coalition Assistance (CA) heuristics monotonically decreased for greater f_{max} values, as presented in Figure 4.3 (a). The Action-Object-Temporal (AOT) heuristics produced monotonically lower (better) makespan and fewer actions for larger f_{max} values, as shown in Figure 4.3 (b and c, respectively). The Action-Temporal (AT) and Coalition Similarity (CS) heuristics produced monotonically worse processing times and memory usage for greater f_{max} values, as indicated in Figure 4.3 (d and e, respectively). CFP resulted in among the best success rates, the best processing time and memory usage, but among the worst makespan and number of actions.

The Object heuristic ($f_{max} = 0.75$) produced the overall best plan quality (Pareto Strength 32), dominating all methods. The Object-Temporal heuristic ($f_{max} = 0.75$) had the second best plan quality (Pareto Strength 30), followed by the Action-Object ($f_{max} = 0.75$), Action-Object-Temporal ($f_{max} = 1.00$) and Object-Temporal ($f_{max} = 1.00$) heuristics (all Pareto Strength 28). The Coalition Similarity ($f_{max} = 0.50$ and 1.00), Coalition Assistance ($f_{max} = 1.00$), and Action-Temporal ($f_{max} = 0.75$) heuristics had the lowest plan quality (all Pareto Strength 0). CFP produced the lowest cost (Pareto Strength 32), dominating all other methods. The Coalition Similarity heuristic ($f_{max} = 0.25$) had the second lowest cost (Pareto Strength 31), followed by the Object heuristic ($f_{max} = 0.75$, Pareto Strength 30). The complete First Response Pareto Analysis results for quality and cost are presented in Appendix Chapter B.3.

The Object (O) heuristic with $f_{max} = 0.75$ was the best solution to the First Response Domain, which provided among the lowest success rates, but the best makespan, number of actions, and processing time. The action oriented plan distance heuristics, Action and Action-Temporal, offered mediocre results and did not provide the best solution to any of the domains and planners evaluated. The Coalition Assistance heuristic with $f_{max} = 0.75$ was the worst solution, with the second lowest success rate, and among the quality and cost results.

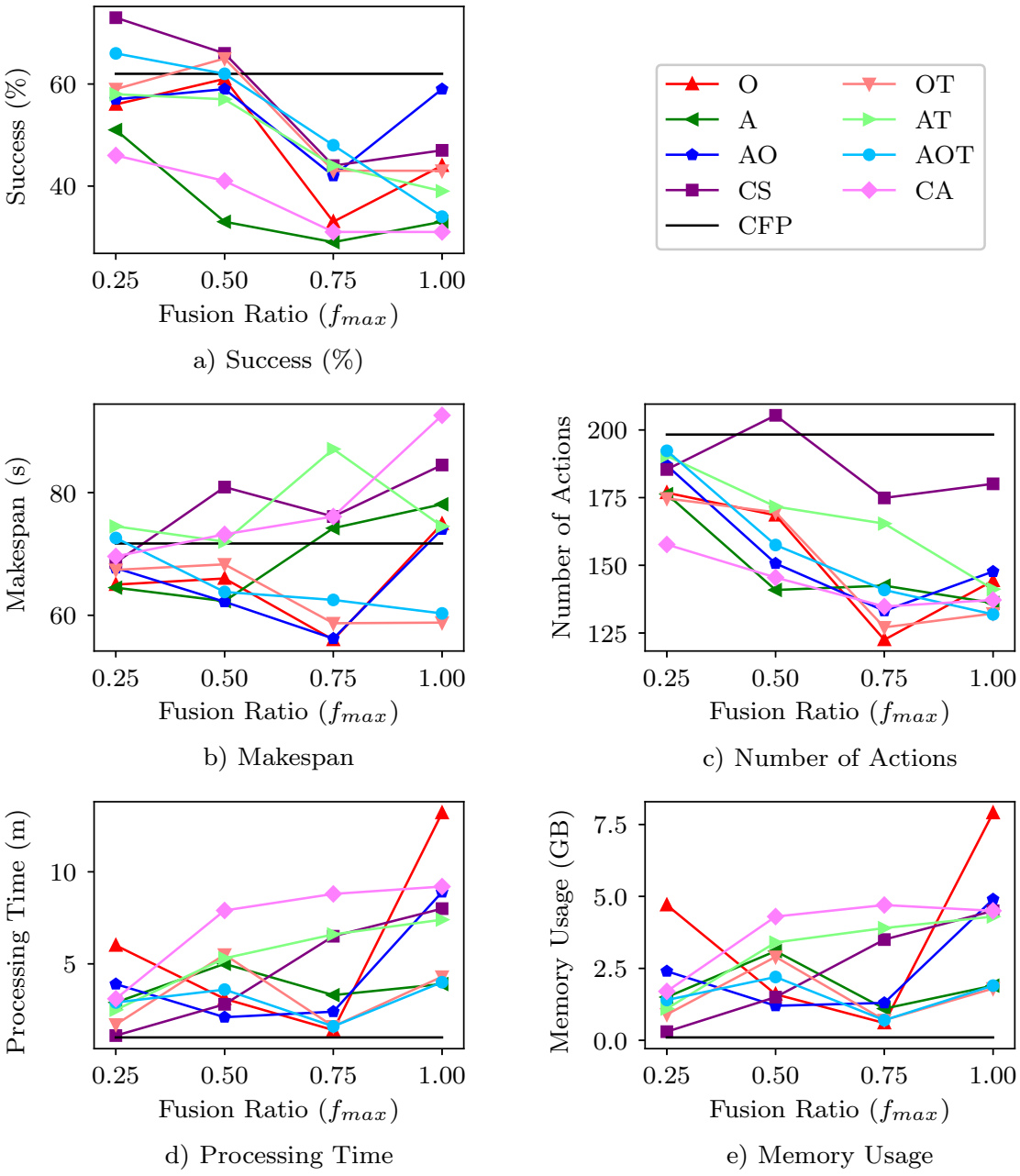


Figure 4.3: First Response (a) success, (b) makespan, (c) number of actions, (d) processing time, and (e) memory usage by fusion ratio (f_{max}). Samples were connected to facilitate visualization.

4.5 Discussion

The proposed heuristics significantly outperform the baseline methods in the resulting plans' quality and offers a better trade-off between quality and processing cost. While other existing methods make constraining assumptions, such as requiring serial plan execution, or requiring a specific planning algorithm, the framework is demonstrated to outperform baselines on both planning algorithms used.

The First Response Domain has an underlying routing problem, in that robots must travel across locations in order to perform their location-dependent tasks, such as navigating to a victim before triaging said victim. The randomly distributed victim locations result in a wide variety of complex routing problems, which is a possible cause for the larger variance across the various metrics when compared to the Blocks World Domain. Faster routes involving multiple short hops generate more actions than slower routes with fewer long hops. The number of possible alternative routes across the locations' graph connecting the multiple points of interest is larger. Allocating different tasks to different robots can result in plans with a wider variety of makespans and number of actions, due to the fact that the allocated robots perform different paths to achieve their tasks, depending on the robots' initial locations. The distances traveled by robot arms are more uniform in the Blocks World Domain, as there are only two block sizes.

The hypothesis H_1 stated that the effectiveness of Task Fusion is affected by the heuristics used, which was supported across all experiments. The heuristics had a profound impact on the Task Fusion effectiveness, with the choice of heuristic

resulting in success rates ranging from the lowest to the highest. The fusion ratio, f_{max}, also impacted Task Fusion by limiting the number of fused coalition-task pairs. The lower f_{max} values attenuated the negative impacts of the bad heuristics, whereas the higher values enhanced the effectiveness of the good heuristics. The best-performing heuristics had a positive relationship with increasing f_{max}. The Action-Object-Temporal heuristic performed generally better for larger f_{max} values, whereas the Coalition Similarity heuristic performed the worst.

Hypothesis H_2 stated that the object oriented plan distance heuristics, Object, Action-Object, Object-Temporal, and Action-Object-Temporal, outperform the baselines: Coalition Assistance and Coalition Similarity heuristics, CFP, and Planning Alone, which was supported. The best solution for each domain and planner was produced by object oriented plan distance heuristics, which account for logical objects common across the task plans to identify and fuse tightly coupled tasks. Task Fusion increases coalition size, which increases the search space; thus, increasing the computational costs. However, when two tightly coupled tasks are fused, the actions accomplishing one task often contribute to achieving states necessary to achieve the other task.

The third hypothesis, H_3, was supported, as the Object and Action-Object heuristics offered the best quality plans across the evaluated domains and planners. The relaxed plan logical objects provide an accurate estimate of the utility of fusing coalition-task pairs. Detecting and fusing tightly coupled coalition-task pairs facilitates planning for tasks that require explicit cooperation between robots. Robots explicitly cooperate and accomplish tasks faster when tightly coupled tasks

are fused. The Coalition Assistance and Coalition Similarity heuristics fuse tasks based on coalition formation capabilities and fail to account for task planning elements, such as plan actions and logical objects. Tightly coupled tasks are planned separately and the outcomes of one task increase the planning complexity for other tasks, resulting in worse plan quality.

The final hypothesis, H_4, was not supported, as no method dominated costs across all experiments. The object oriented plan distance heuristics resulted in lower costs compared to the baseline methods for the Blocks World Domain with TFD, but were superseded by the Coalition Similarity heuristic for the same domain with COLIN. Further, the Coalition Similarity heuristics' low cost results are associated with the worst plan quality. The object oriented plan distance heuristics resulted in better costs compared to the Coalition Assistance heuristic for all domains and planners, which provides some support for hypothesis H_4.

The dissertation extended and applied distance heuristics $\delta(\pi_i, \pi_j)$ as Task Fusion heuristics $H(\pi_i, \pi_j)$, related by the formula $H = 1 - \delta$. The existing action oriented plan distance heuristic [116] was extended to include plans' logical objects and used lists instead of sets to account for repeated instances. Plans often include repeated instances of actions and logical objects (i.e., the same block is handled multiple times to achieve a task in the Blocks World Domain, or the same location is visited to rescue victims in the First Response Domain). Accounting for the repeated instances of actions and logical objects allows more accurate coupling estimation. Tightly coupled tasks require robots to handle the same blocks and transition through the same locations more often, increasing the likelihood for

dependencies and conflicts. The object oriented plan distance heuristics outperformed the action oriented plan distance heuristics across most evaluated metrics, domains, and planners. Using lists of logical objects represents a potential contribution to diverse planning, which needs to be evaluated as future work.

The heuristics contribute to a more informed task allocation. Coalition formation models operate on robot and task capabilities, but lack planning domain information. The plan distance heuristics use relaxed plans in order to introduce planning domain information into the task allocation process. The added planning domain information supports more accurately estimating the value of fusing coalition-task pairs and results in improved task allocation. The heuristics also contribute to estimating coupling between planning problems. Determining the exact problem coupling by computing the treewidth of the agent interaction graph is an NP-hard problem [18]. The heuristics offer an approximate alternative, which is polynomial on the number of actions and objects in the problem's plan: $O(|L_i| \cdot |L_j|)$, where $|L_i|$ and $|L_j|$ are list sizes for action-object lists L_i and L_j, respectively.

4.6 Conclusion

The plan distance heuristics were introduced to provide a better balance between plan quality and the required processing resources, when planning for multiple heterogeneous robots in complex real-world time-sensitive domains. The heuristics estimate plan distance as a proxy for estimating coalition-tasks coupling. The level of coupling determines which coalition-tasks pairs to fuse, after robots are grouped

into coalitions and allocated tasks. Fusing coupled tasks improves plan quality by increasing cooperation between robots, while separating loosely coupled tasks reduces plan synthesis cost. The heuristics use lists of logical object instances, extracted from the plans' action description arguments, to reveal nuances ignored by existing Task Fusion heuristics.

The plan distance heuristics combine aspects of problem coupling and plan distance estimation to improve task allocation. The heuristics generally outperform baselines in both plan quality and computational costs. The cases in which the heuristics do not perform strictly better still offer a better balance between plan quality and computational cost. As a result, larger planning problems, which involve more tasks, robots, and logical objects, can be solved.

The Task Fusion heuristics faciliated better coordination before planning, while using planners that support temporal constraints and continuous fluents. However, better coordination after planning is necessary to further improve plan quality, minmize makespan, and scale to large numbers of agents and tightly coupled tasks.

Chapter 5: Scalable Temporal Plan Merging

Coordination before planning facilitates decoupling tasks; however, complex real-world domains prevent complete task decoupling and can result in tightly coupled tasks, causing conflicts between individual plans. Tightly coupled tasks require plan merging methods to mitigate conflicts that arise when the actions of one plan modify the environment and prevent another plans' actions from succeeding [52].

Previous plan merging algorithms, such as the PMA, described in Chapter 2.3.4, fail to scale and solve complex problems for a growing number of robots and tightly coupled tasks. The Solution Test Algorithm (STA) [52], Algorithm 5.1, is a component of the PMA and was reformulated for this dissertation as an independent merging algorithm.

The STA solves plan conflicts iteratively and adds temporal and causal orders to derive a conflict-free plan. Each conflict resolved generates a new plan, which is added to a priority search queue for further refinement, and the first conflict-free plan encountered is returned.

The STA uses the most-constrained conflicts first heuristic [136] and orders the priority queue using the number of conflict solutions [52]. Cyclical plans are discarded and a null (empty) plan is derived when the conflicts encountered cannot be resolved. Serial plan synthesis guarantees that the conflicts between plans can be resolved, and prevents the derivation of a null plan. STA requires supporting algorithms to identify and resolve conflicts. STA invokes conflict identification

Data: A (conflicted) plan π, consisted of a list of m multiagent plans, $\Pi = \langle \pi_1, \ldots, \pi_m \rangle$;
Result: A conflict-free plan π or null;

```
1  Add multiagent plan π to the queue;
2  while the queue is not empty do
3  |  Pop plan π from the queue;
4  |  if plan π is not cyclical then
5  |  |  Identify conflicts K = {κ_1, ...} in plan π;
6  |  |  if there are conflicts in plan π then
7  |  |  |  foreach conflict κ ∈ K do
8  |  |  |  |  Identify solutions S = {s_1, ...} to conflict κ;
9  |  |  |  |  foreach solution s ∈ S do
10 |  |  |  |  |  Apply solution s to produce plan s(π) = π_s;
11 |  |  |  |  |  Compute priority f(π_s) = |S|, the number of solutions to
   |  |  |  |  |   conflict κ;
12 |  |  |  |  |  Enqueue plan π_s with priority f(π_s);
13 |  |  |  else return conflict-free plan π ;
14 return null;
```

Algorithm 5.1: The Solution Test Algorithm, a component of PMA [52], extracted and reformulated as a standalone plan merging algorithm.

algorithms (line 5) to identify open preconditions and causal conflicts. STA later invokes conflict resolution algorithms (line 8) to identify solutions for each conflict. The worst case time complexity is $O((|K| \cdot |S|)^n)$, where $|K|$ is the number of conflicts identified per iteration, $|S|$ is the number of solutions to each conflict, and n is number of successive conflicts resolved in the resulting conflict-free plan π. The number of conflicts and solutions identified is dependent on the conflict models employed by the conflict identification and resolution algorithms, which are formalized in Chapter 5.1. The STA scales better than the PMA, but does not minimize makespan [52]. Reductions in makespan can make the difference between mission success or failure. The resulting plan's makespan determines the

time taken to accomplish a plan's goals, a key factor in real-world problems.

The *Temporal Optimal Conflict Resolution Algorithm* (TCRA*), developed for this dissertation, optimally minimizes makespan, but early trials did not scale to a large number of tasks in complex problems [102]. The success rates were significantly impaired and most experiments failed due to the high computational costs. The conflict identification and resolution algorithms adopted were the limiting factor and severely impacted performance. New algorithms, complexity analysis, and experiments, show that STA and TCRA* can scale to large problems using the adequate conflict resolution models. The results show that the original algorithms' performance was hindered by the previous conflict identification algorithms. The new algorithms allow STA and TCRA* to scale and solve problems with an increasing number of agents and tasks.

5.1 Conflict Identification and Resolution

The conflict identification and resolution algorithms have a profound impact on the overall plan merging's computational cost. Search algorithms perform conflict identification and resolution many times when merging plans. The computational cost of identifying conflicts and solutions depends on the underlying assumptions made about conflicts. Such underlying assumptions are formalized and two individual conflict models are presented. The models and algorithms that identify conflicts and conflict solutions are summarized in Table 5.1. The assumptions necessary to identify open preconditions are formalized, followed by a presentation of the Open Precondition Identification Algorithm.

Table 5.1: Conflict Identification and Resolution Overview.

	Open Preconditions	**Causal Conflicts**
Identify Conflicts	Chapter 5.1.1, Algorithm 5.2	Chapter 5.1.3, Algorithm 5.4
Identify Solutions	Chapter 5.1.2, Algorithm 5.3	Chapter 5.1.4, Algorithm 5.5

5.1.1 Open Precondition Identification

Open precondition identification, performed by Algorithm 5.2, requires evaluating each plan action precondition. Each action, $a \in A$ (line 2), and each action precondition, $c \in pre(a)$ (line 3), is evaluated. Action preconditions not satisfied by a causal order (line 4) generate an open precondition (line 5).

Data: A plan $\pi = \langle A, \prec_T, \prec_C \rangle$;
Result: The set of open preconditions $K_o = \{\kappa_{o1}, \ \ldots\}$ in plan π;
1 Initialize an empty set of open preconditions $K_o = \{\}$;
2 **foreach** *action* $a \in A$ **do**
3 | **foreach** *condition* $c \in pre(a)$ **do**
4 | | **if** $a_0 \prec^c a \notin \prec_C$ **then**
5 | | | Add open precondition $\kappa_o = \langle a, c \rangle$ to the set K_o;
6 **return** the set of open preconditions K_o;

Algorithm 5.2: Open Precondition Identification.

The worst case time complexity is $O(|A| \cdot \max_{a \in A} |pre(a)|)$, where $|A|$ is the number of plan actions and $\max_{a \in A} |pre(a)|$ is the maximum number of preconditions per action.

5.1.2 Open Precondition Resolution

The Open Precondition Resolution Algorithm, Algorithm 5.3, evaluates all viable solutions and produces an independent new plan for each solution. The algorithm's

inputs are a plan π and an open precondition $\kappa_o = \langle a, c \rangle$, which consists of an action a and a condition c.

Data: A plan $\pi = \langle A, \prec_T, \prec_C \rangle$ and an open precondition $\kappa_o = \langle a, c \rangle$;
Result: A set of plans $\Pi = \{\pi_1, \ldots\}$ that implement all solutions to κ_o;
1 Initialize an empty set of plans $\Pi = \{\}$;
2 **foreach** *action* $a_i \in A$ **do**
3 | **if** *condition* $c \in eff(a_i)$ **then**
4 | | Add a causal order $a_i \prec^c a$ to $\prec_C$, generating plan π_i;
5 | | Add plan π_i to the set Π;
6 **return** the set of plans Π;

Algorithm 5.3: The Open Precondition Resolution Algorithm.

A causal order $a_i \prec^c a$ (line 4) is introduced for each action a_i (line 2), if action a_i produces condition c, $c \in eff(a_i)$ (line 3). Each introduced causal order produces a new solution plan π_i, as presented in Algorithm 5.3 lines 4 and 5. Algorithm 5.3's worst case time complexity is $O(|A|)$, where $|A|$ is the number of plan actions. Causal conflicts require more intricate models and assumptions than open preconditions. Two distinct models that identify causal conflicts are presented.

5.1.3 Causal Conflict Identification

Causal Conflict Identification requires defining *ordered away* actions. Causal conflicts occur when a causal order's actions, a_i and a_j, are threatened by the effects of another action a. A threatening action, a, is *ordered away* from the threatened actions, a_i and a_j, if there are temporal orders ensuring that action a is executed before action a_i or after action a_j. Two causal conflict models, *direct* and *transitive*,

exist to determine if an action is ordered away.

The *transitive model* accounts for temporal order transitivity, while the *direct model* does not. Transitivity implies that if action $a_i \prec a_j$ and $a_j \prec a_k$, then $a_i \prec a_k$. The transitive model requires transitive ordering between actions a and a_i, $a \prec \ldots \prec a_i$, or between actions a_j and a, $a_j \prec \ldots \prec a$. Multiple temporal orders can be introduced to satisfy the transitive model ordering criteria, as there can be many temporal orders between actions a and a_i that satisfy $a \prec \ldots \prec a_i$. The direct model is more strict and requires a direct temporal order between actions a and a_i, $a \prec a_i \in \prec_T$, or between actions a_j and a, $a_j \prec a \in \prec_T$.

The Boolean function $orderedAway(a, a_i, a_j)$ encapsulates each model's assumptions in order to indicate whether action a is ordered away from actions a_i and a_j. The transitive model $orderedAway(\cdot)$ function is $orderedAway(a, a_i, a_j) = a \prec \ldots \prec a_i \vee a_j \prec \ldots \prec a$ and accounts for the transitive orders between the threatening and the threatened actions. For example, $a \prec a_i$, $a \prec a_x \prec a_i$, and $a \prec a_x \prec a_y \prec a_i$, all satisfy $orderedAway(\cdot)$.

The direct model $orderedAway(\cdot)$ function is $orderedAway(a, a_i, a_j) = a \prec a_i \in \prec_T \vee a_j \prec a \in \prec_T$ and requires the set of temporal orders, $\prec_T$, to include orders directly between the threatening and the threatened actions. For example, $a \prec a_i$ satisfies $orderedAway(\cdot)$, but $a \prec a_x \prec a_i$ and $a \prec a_x \prec a_y \prec a_i$ do not. The differences between the two models result in a different number of identified conflicts.

A causal conflict is identified between an action a and a causal order $a_i \prec^c a_j$ if two criteria are met: Action a negates condition c, $\neg c \in eff(a)$, and action

a is not ordered away from the causal order's actions a_i and a_j. The Causal Conflict Identification Algorithm, Algorithm 5.4, evaluates each causal order $a_i \prec^c a_j$ (line 2) and each action a of plan π (line 3). Actions a that negate condition c, $\neg c \in eff(a)$, and are not ordered away from the causal order's actions (line 4) generate a causal conflict (line 5).

Data: A plan $\pi = \langle A, \prec_T, \prec_C \rangle$;
Result: The set of causal conflicts $K_c = \{\kappa_{c1}, \ldots\}$ in plan π;
1 Initialize an empty set of causal conflicts $K_c = \{\}$;
2 **foreach** *causal order* $a_i \prec^c a_j \in \prec_C$ **do**
3 | **foreach** *action* $a \in A$, **do**
4 | | **if** $\neg c \in eff(a)$ ***and*** $\neg$ $orderedAway(a, a_i, a_j)$ **then**
5 | | | Add causal conflict $\kappa_c = \langle a, a_i \prec^c a_j \rangle$ to the set K_c;
6 **return** the set of causal conflicts K_c;

Algorithm 5.4: The Causal Conflict Identification Algorithm.

Algorithm 5.4's worst case time complexity is $O(\mid \prec^C \mid \cdot |A| \cdot O(orderedAway(\cdot)))$, where $\mid \prec^C \mid$ is number of causal orders and $|A|$ is the number of plan actions. The $orderedAway(\cdot)$ function's worst case time complexity is dependent on the conflict model.

The transitive model $orderedAway(\cdot)$ function's worst case time complexity is $O(|A| \cdot \mid \prec_T \mid)$, where $|A|$ is the number of actions and $\mid \prec_T \mid$ is the number of temporal orders, as a reachability search is necessary to determine if two actions are ordered transitively [161]. Thus, the overall causal conflict identification (Algorithm 5.4) worst case time complexity is $O(\mid \prec^C \mid \cdot |A|^2 \cdot \mid \prec_T \mid)$ using the transitive model. The direct model $orderedAway(\cdot)$ function's worst case time complexity is constant and the overall causal conflict identification's worst case

time complexity is $O(|\prec^C| \cdot |A|)$. The two causal conflict models also impact the number of identified solutions for each conflict.

5.1.4 Causal Conflict Resolution

Causal conflict solutions introduce two sets of temporal orders to order action a away from actions a_i and a_j. The first set orders action a before action a_i. The direct model introduces the temporal order $a \prec a_i$, whereas the transitive model introduces all the temporal orders that satisfy $a \prec \ldots \prec a_i$. The second set of temporal orders place action a after action a_j. The direct model introduces the temporal order $a_j \prec a$, whereas the transitive model introduces all the temporal orders that satisfy $a_j \prec \ldots \prec a$. The functions $prefix(a, a_i)$ and $posfix(a, a_j)$ encapsulate the described model assumptions in order to generate the actions that compose each set of temporal orders.

The Causal Conflict Resolution Algorithm, Algorithm 5.5, introduces the two sets of temporal orders and generates independent solution plans. Each action a_k produced by the function $prefix(a, a_i)$ (line 2) introduces a temporal order $a \prec a_k$ and generates a new solution plan (line 3). Each action a_k produced by the function $posfix(a, a_j)$ (line 5) introduces a temporal order $a_k \prec a$ and generates a new solution plan (line 6).

The transitive model $prefix(a, a_i)$ function produces action a_i and all the actions ordered before action a_i that are not ordered before action a, $\{a_i, a_k\}$, $\forall\ a_k \in \{a_k \prec a_i\} - \{a_k \prec a\}$. The temporal orders are introduced between action a and the actions that precede action a_i, $\{a_k \prec a_i\}$, causing action a to be ordered

Data: A plan $\pi = \langle A, \prec_T, \prec_C \rangle$ and a causal conflict $\kappa_c = \langle a, a_i \prec^c a_j \rangle$;
Result: A set of plans $\Pi = \{\pi_i, \pi_j\}$ that implement all solutions to κ_c;
1 Initialize an empty set of plans $\Pi = \{\}$;
2 **foreach** *action* $a_k \in prefix(a, a_i)$ **do**
3 | Add a temporal order $a \prec a_k$ to $\prec_T$, generating plan π_k;
4 | Add plan π_k to the set Π;
5 **foreach** *action* $a_k \in posfix(a, a_j)$ **do**
6 | Add a temporal order $a_k \prec a$ to $\prec_T$, generating plan π_k;
7 | Add plan π_k to the set Π;
8 **return** the set of plans Π;

Algorithm 5.5: The Causal Conflict Resolution Algorithm.

away from actions a_i and a_j. Cyclic ordering is prevented by excluding actions that precede action a, $\{a_k \prec a\}$.

The transitive model $posfix(a, a_j)$ function produces action a_j and all the actions ordered after action a_j that are not ordered after action a, $\{a_j, a_k\}$, $\forall\ a_k \in \{a_j \prec a_k\} - \{a \prec a_k\}$. The temporal orders introduced between action a and the actions that succeed action a_j, $\{a_j \prec a_k\}$, resolve the causal conflict due to the transitive property. The exclusion of actions a_k that succeed the threatening action a, $\{a \prec a_k\}$, prevents the introduction of cycles. The resulting direct model algorithm's worst case time complexity is given by $O(|A|)$, where $|A|$ is the number of actions of the input plan π.

The direct model $prefix(\cdot)$ and $posfix(\cdot)$ functions produce actions a_i and a_j, respectively. The worst case time complexity is constant for both functions, and the resulting conflict resolution algorithm's worst case time complexity is constant ($O(1)$).

5.1.5 Conflict Models and the Overall Plan Merging Computational Cost

The conflict identification and resolution algorithms' worst case time complexities, presented in Chapters 5.1.1-5.1.4, are summarized in Table 5.2. The direct model's causal conflict worst case time complexity is better than the transitive model's, both for conflict identification and resolution. However, the impact of each model on the overall plan merging computational cost is dependent on an additional factor. The number of conflicts identified, $|K|$, and the number of solutions to each conflict, $|S|$, impact the plan merging worst case time complexity. The causal conflict models identify different numbers of conflicts and solutions. The direct model identifies more conflicts than the transitive model, as fewer threatening actions are considered to be ordered away. The direct model identifies fewer solutions for each conflict, when compared to the transitive model.

Table 5.2: Conflict Identification and Resolution Worst Case Time Complexity per Conflict Model.

	Conflict Model	**Identification**	**Resolution**
Open Precondition	**N/A**	$O(\|A\| \cdot \max_{a \in A} \|pre(a)\|)$	$O(\|A\|)$
Causal Conflict	**Transitive Model**	$O(\| \prec^C \| \cdot \|A\|^2 \cdot \| \prec_T \|)$	$O(\|A\|)$
	Direct Model	$O(\| \prec^C \| \cdot \|A\|)$	$O(1)$

The balance between the number of conflicts and solutions identified varies per conflict model and problem domain. The overall plan merging computational cost is an emergent property that results from the complex interactions between the conflict models and the merging algorithms' computational cost. An empirical evaluation, presented in Chapters 5.3 and 5.4, compared each method across ran-

domly generated problems. The overall plan merging computational cost can also be reduced by applying a transitive closure, which discards redundant plans.

5.1.6 Transitive Closure

Plan merging algorithms, such as STA (Algorithm 5.1), search the space of plans, while identifying plans' conflicts and conflict solutions. Each conflict solution generates a new plan, which is added to the algorithms' search queue. Many plans are transitively equivalent and can be discarded, reducing the search space, and minimizing the overall computational cost. The transitive closure [96] can be applied before plans are enqueued in order to discard redundant plans, as transitively equivalent plans result in the same plan after closure.

The transitive closure introduces temporal orders $\prec$ between all transitively connected plan actions. Plan actions a_i and a_j are transitively connected if there is a path between action a_i and a_j, $a_i \prec \ldots \prec a_j$ [20]. The transitive closure can be performed with $O(|A| \cdot | \prec_T |)$ worst case computational complexity, where $|A|$ is the number of actions and $| \prec_T |$ is the number of temporal orders [92].

The empirical evaluation, presented in Chapters 5.3 and 5.4, compares the effects of the transitive closure on the resulting plans. Chapter 5.2 introduces the *Temporal Optimal Conflict Resolution Algorithm* (TCRA*) [102], which merges multiagent plans while minimizing the resulting makespan with quality guarantees.

5.2 Temporal Plan Coordination

Accounting for durative actions, while merging loosely and tightly coupled multiple robot plans, is necessary to minimize makespan. The Multiagent Plan Coordination Problem (MPCP) is extended to incorporate durative actions and minimize makespan. The Temporal MPCP (TMPCP) is a tuple $\langle I, G, \pi_I \rangle$, where $I = \{i_1, \ \ldots\}$ is the set of initial conditions, $G = \{g_1, \ \ldots\}$ is the set of goal conditions, and π_I is the initial plan, which consists of a list $\Pi = \langle \pi_1, \ \ldots, \pi_m \rangle$ of m multiple robot plans. TMPCP plans' actions are defined by the tuple $\langle pre(\cdot), eff(\cdot), dur(\cdot) \rangle$, where $pre(a)$ is a set of preconditions that must hold during action a's execution, $eff(a)$ is the set of effects caused by action a's execution, and $dur(a)$ is action a's execution duration, the amount of time necessary to execute the action. A solution to a TMPCP is a conflict-free plan, π, that satisfies all goal conditions in G when executed from the set of initial conditions, I. All the actions of a solution plan π are drawn from the multiple robot plans, Π. No actions are added or removed.

A plan's path is a series of actions chained by temporal orders, $P = \langle \ a_1, \ \ldots \ \rangle$. A path's length, $l(P)$, is the sum of the action durations, $l(P) = \sum_{a \in P} dur(a)$. The makespan, $m(\pi)$, is the length of π's longest path, $m(\pi) = \max_{P \in \pi} l(P)$, that represents the time necessary to execute all of plan π's actions. An optimal solution plan, π^*, results in a makespan $m(\pi^*)$ less than, or equal to the makespan of any conflict-free plan π generated after solving n successive conflicts in the initial plan, $m(\pi^*) \leq m(\pi) \ \forall \ \pi \in s^n(\pi_I)$.

A Logistics example involves an autonomous semi-tractor-trailer truck (a), a human driven truck (m), and two delivery tasks. Current autonomous semi-trucks are restricted to interstate highways and are not permitted in urban traffic [158]. The trailers are transferred to human driven trucks before entering cities. Trailers 1 and 2 must be transported from a factory, via a highway, to an urban warehouse. Trucks can only transport one trailer at a time and exchange trailers at a transfer hub outside the city. Plans are generated individually for each delivery task, as shown in Figure 5.1. The plan for task 1 results in truck a bringing trailer 1 from the factory to the hub in 3 hours (1^a, 3h), followed by truck m moving trailer 1 to the warehouse (1^m, 1h). The plan for task 2 involves truck m returning to the hub (2_i^m, 1h), while truck a transports trailer 2 (2^a, 3h), followed by truck m moving trailer 2 to the warehouse (2_{ii}^m, 1h). The plans can be merged to accomplish both tasks.

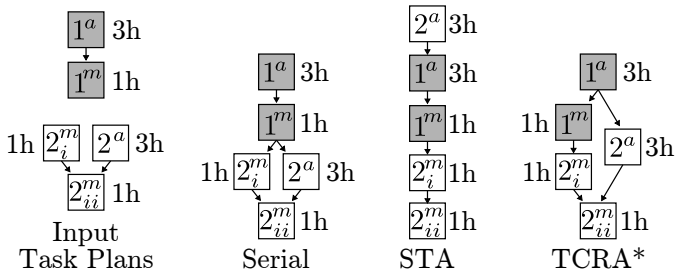


Figure 5.1: Illustrative Logistics problem and results for Serial, STA, and TCRA*. Task 1 actions shaded.

The *Serial Algorithm*, Algorithm 5.6, a TMPCP Solution, introduces temporal orders in order to serialize the multiple robot plan execution. The Serial Algorithm assumes the multiple robot plans were synthesized serially, and can be merged into

a conflict-free plan when executed serially. The worst case computational time complexity is $O(m \cdot |A_m|^2)$, where m is the number of multiple robot plans in the multiple robot plan list, Π, and $|A_m|$ is the maximum number of actions in each multiple robot plan. The Serial Algorithm strictly orders plan π_2's actions after plan π_1's, as shown in Figure 5.1, resulting in a 8 hour makespan.

A second solution for TMPCP is the *STA* (Algorithm 5.1), which does not necessarily minimize the resulting plans' makespan, making it suboptimal. STA results in truck a serially fetching both trailers to the hub, followed by truck m serially delivering both trailers, resulting in a 9 hour makespan.

Data: A (conflicted) plan π, consisted of a list of m multiple robot plans, $\Pi = \langle \pi_1, \ldots, \pi_m \rangle$;
Result: A conflict-free plan π;
1 **foreach** *plan* π_k *in plan list* Π **do**
2 **foreach** *action* a_i *in plan* π_k **do**
3 **foreach** *action* a_j *in plan* π_{k+1}, *where plan* π_{k+1} *succeeds plan* π_k *in the multiple robot plan list* Π **do**
4 Add temporal order $a_i \prec a_j$ to plan π;
5 **return** the resulting plan π;

Algorithm 5.6: The Serial algorithm.

The *Temporal Optimal Conflict Resolution Algorithm* (TCRA*) employs an A^* search to guarantee completeness and minimize makespan, as presented in Algorithm 5.7. TCRA* is similar to STA, in that both algorithms search the space of plans in order to solve conflicts iteratively and result in a conflict-free plan. STA and TCRA* have worst case computational time complexity $O((|K| \cdot |S|)^n)$, where $|K|$ is the number of conflicts identified per iteration, $|S|$ is the number of solutions to each conflict, and n is number of successive conflicts resolved in the

resulting conflict-free plan π.

Data: A (conflicted) plan π, consisted of a list of m multiple robot plans, $\Pi = \langle \pi_1, \ldots, \pi_m \rangle$;
Result: A conflict-free plan π or null;

```
1  Add input plan π to the queue;
2  while the queue is not empty do
3  |  Pop plan π from the queue;
4  |  if plan π is not cyclical then
5  |  |  Identify conflicts K = {κ_1, ...} in plan π;
6  |  |  if there are conflicts in plan π then
7  |  |  |  foreach conflict κ ∈ K do
8  |  |  |  |  Identify solutions S = {s_1, ...} to conflict κ;
9  |  |  |  |  foreach solution s ∈ S do
10 |  |  |  |  |  Apply solution s to produce plan s(π) = π_s;
11 |  |  |  |  |  Compute f(π_s) = m(π_s) + ε · h(π_s);
12 |  |  |  |  |  Enqueue plan π_s with priority f(π_s);
13 |  |  |  else return conflict-free plan π ;
14 return null;
```

Algorithm 5.7: The Temporal Optimal Conflict Resolution Algorithm (TCRA*).

TCRA* performs a uniform-cost search and employs an admissible heuristic to minimize makespan and guarantee optimality. Plan π's priority, $f(\pi)$, is defined by the plan's makespan, $m(\pi)$, and the admissible search heuristic, $h(\pi)$. The heuristic can be multiplied by a relaxation scalar $\epsilon > 1$ in order to generate approximate results and produce plans faster. The TCRA* algorithm is reduced to a uniform cost search when $\epsilon = 0$. The TCRA* search space can be minimized by performing the transitive closure before plans are enqueued, in the same way as STA. TCRA* parallelizes both trucks' actions, as shown in Figure 5.1, resulting in a 7 hour makespan.

TCRA*'s admissible heuristic estimates the costs of a given plan π in order to guide the TCRA* search. A plan π has a set of conflicts $K = \{\kappa_1, \ldots\}$. All conflicts K must be addressed, so the cost of π is at least as high as the cost of the most costly conflict; therefore, $f(\pi) = max(K) = max(\ f(\kappa_1), \ \ldots\ )$ is an underestimate. Each conflict $\kappa \in K$ has a set of solutions $S = \{s_1, \ldots\}$. The cost of conflict κ can be as low as the cost of the least costly solution; thus, the cost of κ is underestimated by $f(\kappa) = min(S) = min(\ f(s_1), \ \ldots\ )$. Each solution s generates a new plan, π_s, whose cost is greater than or equal to its makespan, $f(\pi_s) \geq m(\pi_s)$. The makespan $m(\pi_s)$ is an underestimate of the cost of s, $f(s) = m(\pi_s)$. The estimate cost $f(\pi) = max(\ min(\ m(\pi_{s_1}), \ \ldots\ ), \ \ldots\ )$, is an underestimate, and the heuristic $h(\pi) = f(\pi)$ - $m(\pi)$ is admissible.

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

Lemma 1 establishes that adding a temporal order generates a new plan with greater than or equal makespan.

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

Proof. Let π's longest path be P. Add a temporal order $\prec$ to π, forming a new

path P_1 in the resulting plan, π_1.
If $l(P_1) > l(P)$, then π_1's longest path is P_1 and

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

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

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

□

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

Lemma 2. Let κ_o denote an open precondition of plan π. Let s denote a solution to κ_o that adds a temporal order $\prec$ to plan π, producing a new plan $s(\pi) = \pi_1$. Then $m(\pi) \leq m(\pi_1)$.

Proof. The proof follows from Lemma 1. □

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

Lemma 3. Let κ_c denote a causal conflict of plan π. Let s denote a solution to κ_c that adds a causal order $\prec^c$ to plan π, producing a new plan $s(\pi) = \pi_1$. Then $m(\pi) \leq m(\pi_1)$.

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

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

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

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

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

Lemma 5. Let plan π have a series of conflicts κ_1, κ_2, ..., κ_n, resolved by a series of solutions s_1, s_2, ..., s_n, which generates a series of plans π_1, π_2, ..., π_n. Then $m(\pi) \leq m(\pi_n)$.

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

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

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

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

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

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

The original empirical trials for TCRA* [102] used the direct causal conflict model and did not use the transitive closure, both introduced in Chapter 5.1.

The dissertation experiment in which STA and TCRA* are compared using both the direct and the transitive causal conflict models, with and without transitive closure, is presented. The original empirical trials, using the direct causal conflict model, in addition to the new results, are presented.

5.3 Methodology

The plan merging algorithms, STA, Serial, and TCRA*, Algorithms 5.1, 5.6, and 5.7, respectively, were evaluated using the direct and transitive causal conflict models, with and without transitive closure, across three domains. Each TMPCP problem was extracted from randomly generated planning problems that each consists of a randomly generated group of robots, an initial state, and a goal state composed of randomly generated tasks. Each task has an individual set of requirements, and all tasks' requirements must be met in order to solve the planning problem. Tasks were allocated to robots according to the robots' capabilities and the tasks' requirements using a coalition formation algorithm [186]. Each task was individually solved using the Coalition Formation then Planning framework [62], and an external planner. Each task solution resulted in a task plan. A merging problem must merge all the task plans in order to solve the original planning problem.

The Actions Concurrency and Time Uncertainty Planner (ActuPlan) [24], adopted as the external planner, supports concurrent durative actions with uncertain action durations. Similar planners exist [39, 103], but ActuPlan is the only planner to support a high-level problem description language and offer a publicly

available implementation. The uncertain action durations were determinized [197] by adopting the distribution's mean as the action duration.

The uncertain action durations' extensions of each domain described in Chapter 3.2 were used. The extended Logistics Domain, described in Chapter 3.2.2, was used with two districts and truck types, requiring up to two trucks per task. The second Blocks World Domain extension, described in Chapter 3.2.1, was used with two types of end effectors: suction and magnetic. The second First Response Domain extension, described in Chapter 3.2.3, was used.

Each planning problem consisted of 1-10 tasks and up to 10 robots. The Logistics Domain and the Blocks World Domain had 2-10 robots. The First Response Domain had 5-10 robots, as the domain's tasks require a broader range of robot-specific capabilities. One hundred problems were generated for each task-robot combination.

Each experiment was allocated one CPU core on an Intel Xeon 5115 Linux server. The plan processing time was limited to 1 hour, and the maximum allocated memory was 256 GB. These processing time and memory limits emulate the demands for real-world systems that require timely solutions on computers with limited memory. All algorithms were implemented in Python 2.7. All the Logistics and First Response Domain problems were successfully generated. All Blocks World Domain problems with 1-3 tasks were successfully generated. The ratio of Blocks World Domain problems successfully generated with 4-10 tasks was 99%, 82%, 67%, 43%, 15%, 8%, and 1%, respectively, as the ActuPlan planner failed to generate plans within the time and memory limits for an increasing number of

tasks.

5.4 Results

The independent variables are the Serial, STA, and TCRA* merging algorithms, the direct and transitive causal conflict models, and the presence and absence of the transitive closure. The TCRA* algorithm's relaxation parameter, described in Chapter 5.2, defines the weight of the search heuristic that balances optimality and greediness. The TCRA* algorithm was evaluated for relaxation values $\epsilon = \{0, 1, 10, 100\}$. Optimal makespan is guaranteed when $\epsilon \leq 1$, but lower computational cost can be achieved when $\epsilon > 1$.

The dependent variables are the success rates and makespan. The success rates are the ratio of problems solved within the 1 hour processing time limit, as no failure occurred due to exceeding the maximum allocated memory. Aggregated makespan results are presented across problems solved by all algorithms, in order to avoid problem-specific biases. Thus, instances where algorithms failed all problems are omitted. Aggregates are presented with the standard error of the mean [189]. The transitive closure is referred to as closure for brevity. The full results for the Logistics, Blocksworld, and First Response Domains are presented in Appendices C.1, C.2, and C.3, respectivelly.

5.4.1 The Logistics Domain

The Serial Algorithm solved all problems for all numbers of robots and tasks successfully. STA's best success rates were achieved when using the direct model with closure, for all numbers of robots and tasks, as shown for two and four robots in Figures 5.2 and 5.3, respectively. Since success rates are not aggregated, there are no error bars. The worst STA success rates resulted from STA using the transitive model with closure, for all numbers of robots and tasks. The STA success rates were higher when the number of robots increased, for all numbers of tasks. All STA instances solved all ten robot problems for all numbers of tasks.

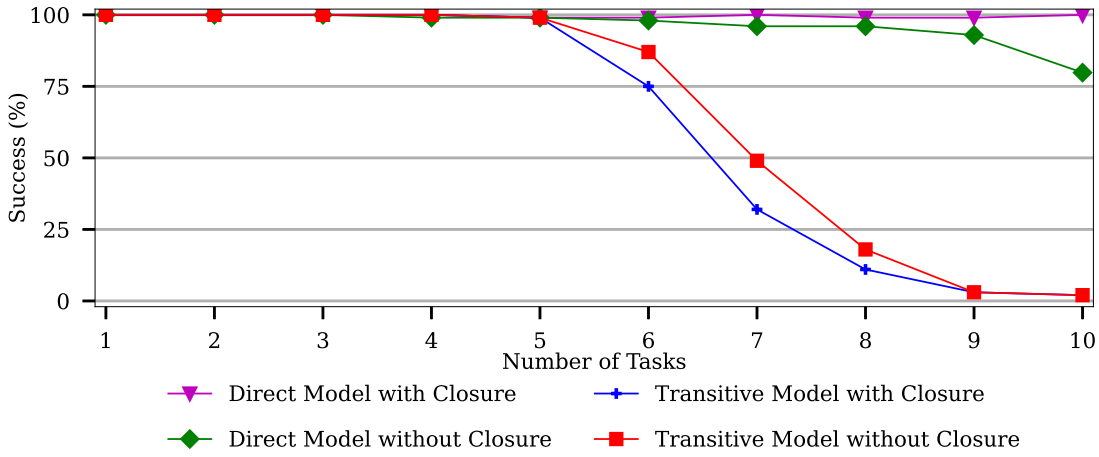


Figure 5.2: Logistics STA success (%) per number of tasks for two robots.

All TCRA* instances' success rates were better for $\epsilon = 1$ than $\epsilon = 0$, across all numbers of robots and tasks. The differences between TCRA*'s success rates for $\epsilon = 1$ and $\epsilon > 1$ were not noticeably different, across all numbers of robots and tasks; therefore, only the $\epsilon \leq 1$ results are provided.

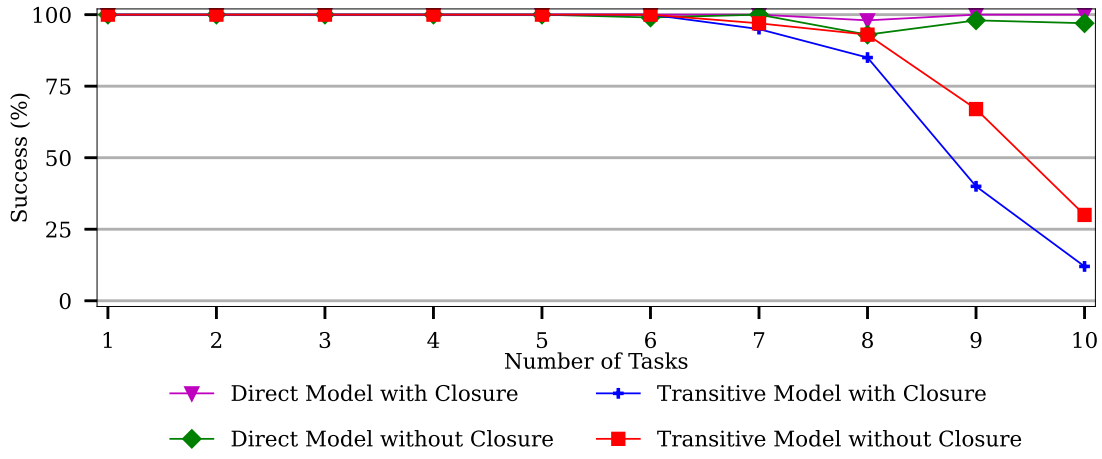


Figure 5.3: Logistics STA success (%) per number of tasks for four robots.

TCRA*'s ($\epsilon = 1$) best success rates were achieved when using the direct model with closure, as shown for two and four robots in Figures 5.4 and 5.5, respectively. TCRA*'s ($\epsilon = 1$) worst success rates were achieved when using the direct model without closure, for all numbers of robots and tasks. A similar pattern was found for $\epsilon = 0$, but the success rates were lower overall, for all numbers of robots and tasks. Ten robot problems resulted in TCRA*'s best success rates for all TCRA* instances and all numbers of tasks.

The Serial Algorithm resulted in the worst makespan for all numbers of robots and tasks. All TCRA* instances resulted in the best makespan for all numbers of robots and tasks, for all conflict models, with and without closure, and for all ϵ values, as presented for two and ten robots in Figures 5.6 and 5.7, respectively. STA using the direct model with, and without, closure also achieved the best makespan, for all numbers of robots and tasks. STA using the transitive model

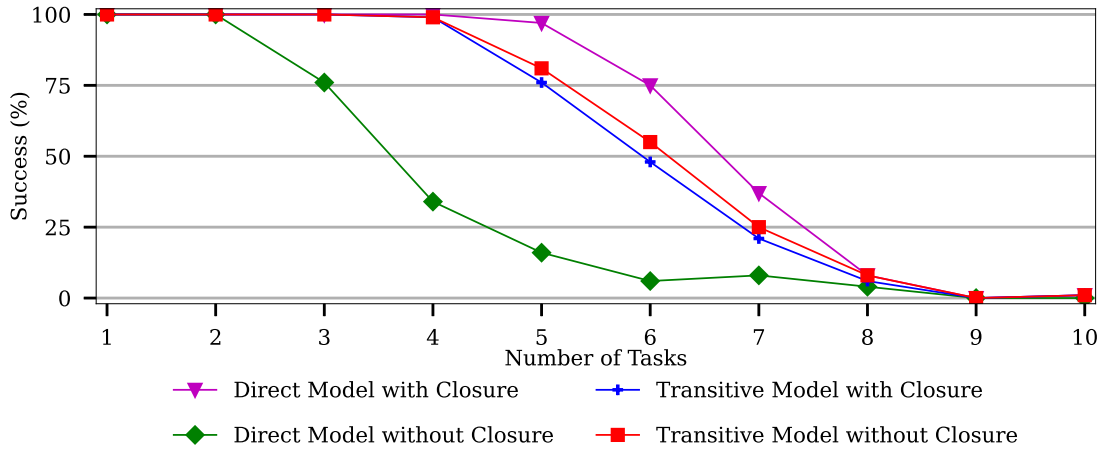


Figure 5.4: Logistics TCRA* ($\epsilon = 1$) success (%) per number of tasks for two robots.

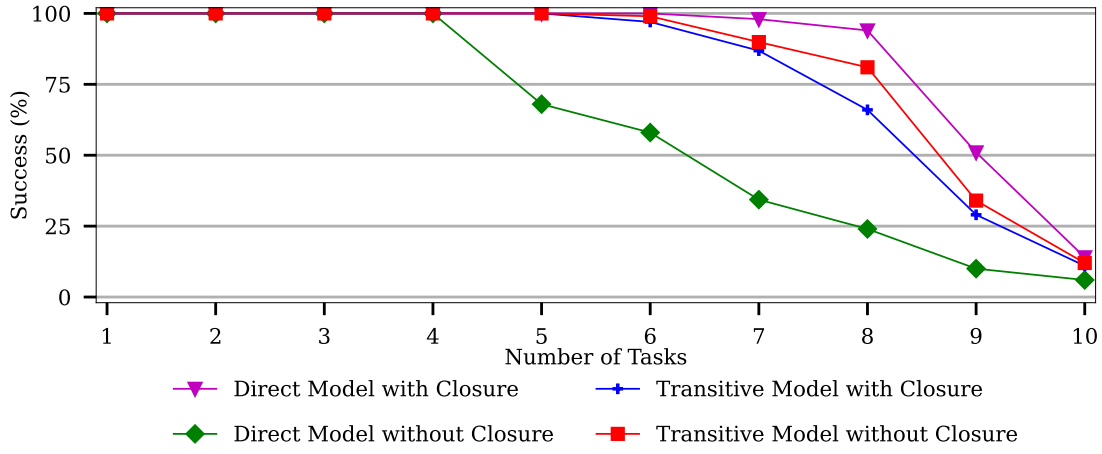


Figure 5.5: Logistics TCRA* ($\epsilon = 1$) success (%) per number of tasks for four robots.

with, and without, closure resulted in slightly worse makespan, for all numbers of robots and tasks. Results for two robots and ten tasks are missing because STA using the transitive model without closure failed to solve all of those problems.

The Serial Algorithm makespan was higher overall when the number of robots increased, for all numbers of tasks, and was the highest for ten robots.

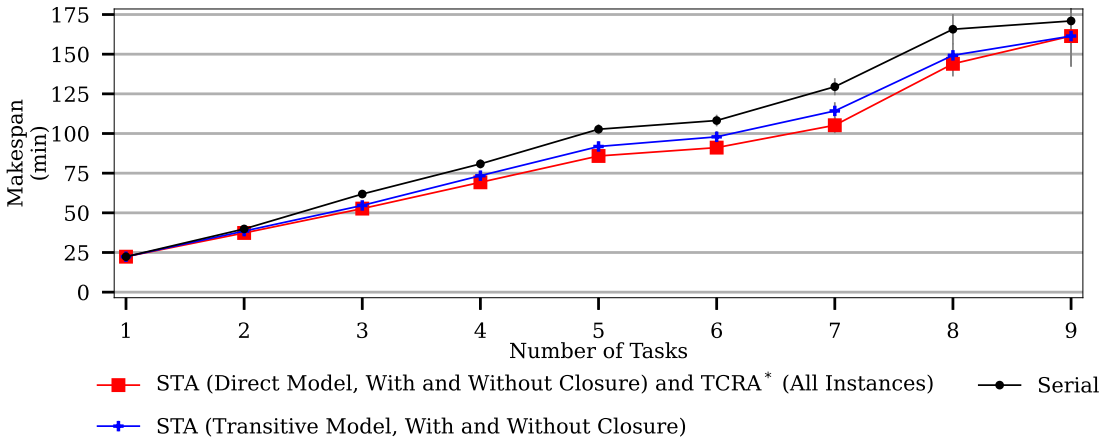


Figure 5.6: Logistics TCRA* (all ϵ values), STA, and Serial makespan (min) for two robots. The maximum makespan was 175 min. TCRA* (all ϵ values) and STA using the direct model with and without closure resulted in the same makespan.

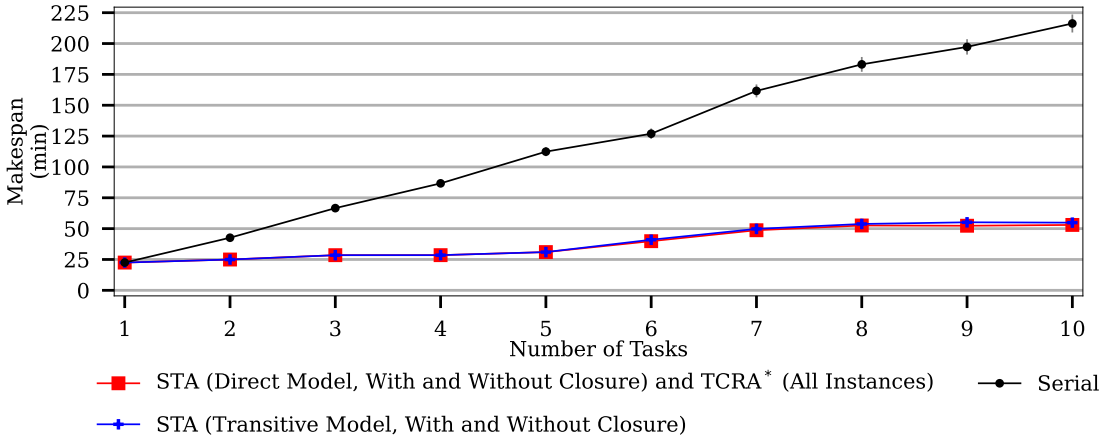


Figure 5.7: Logistics TCRA* (all ϵ values), STA, and Serial makespan (min) for ten robots. The maximum makespan was 225 min.

STA using the direct model with closure was the best solution to the Logistics

Domain, as most problems were solved and the best makespan was achieved, for all numbers of robots and tasks. However, there are no quality guarantees, and STA can result in worse makespan. The second best method was STA with the direct model without closure, which achieved the best makespan with a worse success rate, for all numbers of robots and tasks. The third best method was TCRA* ($\epsilon \geq 1$) using the direct model with closure, that achieved the best makespan, but at a lower success rate, for all numbers of robots and tasks. TCRA* ($\epsilon = 0$) using the direct model without the transitive closure was the worst solution to the Logistics Domain, for all numbers of robots and tasks, as the best makespan was achieved, but with the worst success rates, for all numbers of robots and tasks.

5.4.2 The Blocks World Domain

The Serial Algorithm solved all problems for all numbers of robots and tasks successfully. STA and TCRA* (all ϵ values) using the direct model without closure did not solve all ten robots problems, as shown in Figure 5.8. All other STA and TCRA* instances solved all problems for two to nine tasks, for all ϵ values. The differences between TCRA*'s success rates for $\epsilon = 1$ and $\epsilon > 1$ were not noticeably different, across all numbers of robots and tasks; therefore the presented results use $\epsilon \leq 1$.

The Serial Algorithm resulted in the worst makespan for all numbers of robots and tasks. All TCRA* instances resulted in the best makespan for all numbers of robots and tasks, for all conflict models, with and without closure, and for all ϵ values, as presented for ten robots in Figure 5.9. STA using the direct model with,

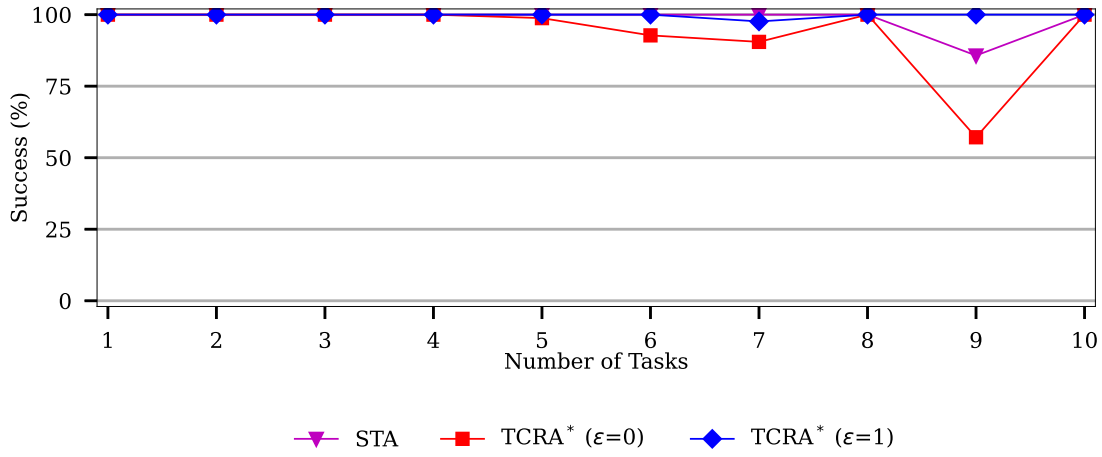


Figure 5.8: Blocks World STA and TCRA* using the direct model without closure success (%) for ten robots.

and without, closure also achieved the best makespan, for all numbers of robots and tasks. STA using the transitive model with, and without, closure resulted in slightly worse makespan, for all numbers of robots and tasks. Results for ten tasks are missing because STA using the transitive model without closure failed to solve all ten task problems. A similar pattern was found for fewer robots, for all numbers of tasks.

Most STA and TCRA* instances provided the best solution to the Blocks World Domain, solving all problems, and resulting in the best makespan for all numbers of robots and tasks. TCRA* ($\epsilon = 0$) using the direct model without closure was the worst solution, with the worst success rates for all numbers of robots and tasks. STA using the transitive model with, and without closure, were the second and third worst solutions, respectively, with an intermediary makespan for all robots and tasks.

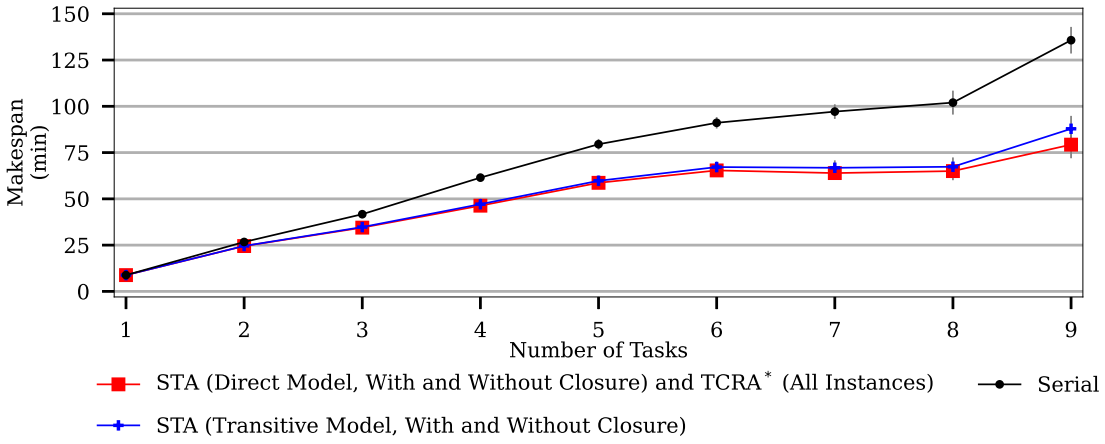


Figure 5.9: Blocks World TCRA* (all ϵ values), STA, and Serial makespan (min) for ten robots. The maximum makespan was 150 min.

5.4.3 The First Response Domain

The Serial Algorithm solved all problems for all numbers of robots and tasks successfully. STA's best success rates were achieved when using the direct model with closure, as shown for ten robots in Figure 5.10. The worst ten robot STA success rates resulted from STA using the direct model without closure for one to seven tasks and STA using the transitive model with closure for eight to ten tasks. Fewer robots resulted in lower success rates overall, for all numbers of tasks.

All TCRA* instances' success rates were better for $\epsilon = 1$ than $\epsilon = 0$, across all numbers of robots and tasks. The differences between TCRA*'s success rates for $\epsilon = 1$ and $\epsilon > 1$ were not noticeably different, across all numbers of robots and tasks; therefore, only the $\epsilon \leq 1$ results are provided.

TCRA*'s ($\epsilon = 1$) best success rates were achieved when using the direct model with closure, for all numbers of robots and tasks, as shown for ten robots in

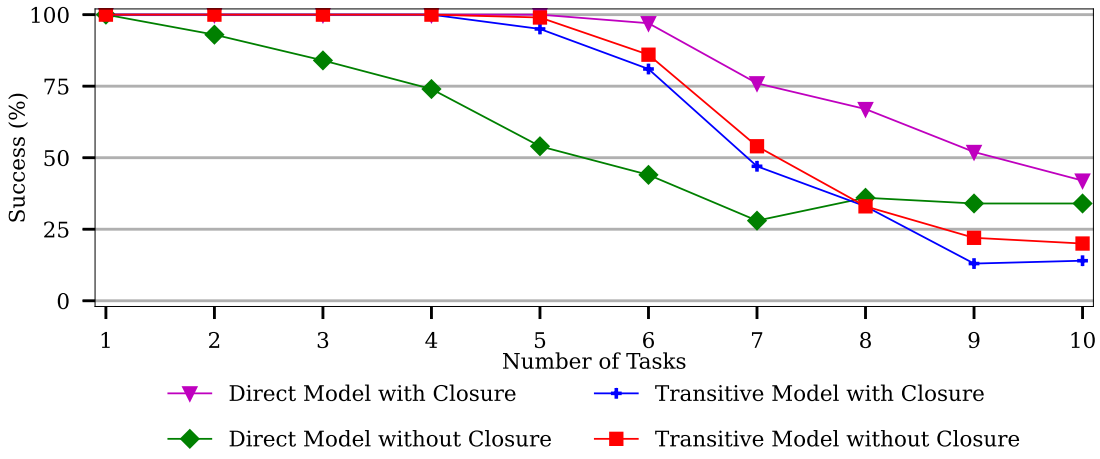


Figure 5.10: First Response STA success (%) per number of tasks for ten robots.

Figure 5.11. TCRA*'s ($\epsilon = 1$) worst success rates were achieved when using the direct model without closure, for all numbers of robots and tasks. A similar pattern was found for $\epsilon = 0$, but the success rates were lower, for all numbers of robots and tasks.

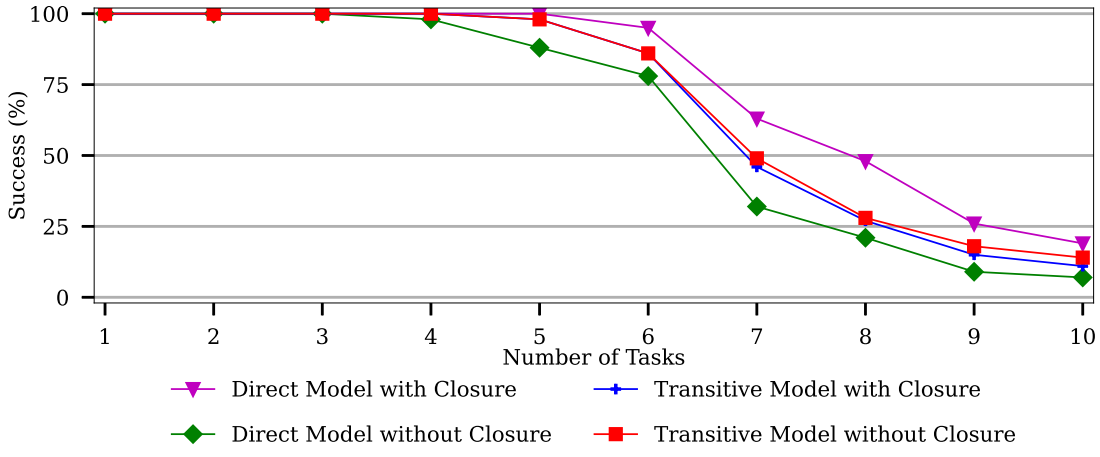


Figure 5.11: First Response TCRA* ($\epsilon = 1$) success (%) per number of tasks for ten robots.

The Serial Algorithm resulted in the worst makespan for all numbers of robots and tasks. All TCRA* instances resulted in the best makespan for all numbers of robots and tasks, for all conflict models, with and without closure, and for all ϵ values, as presented for ten robots in Figure 5.12. STA using the direct model with, and without, closure also achieved the best makespan, for all numbers of robots and tasks. STA using the transitive model with, and without, closure resulted in slightly worse makespan, for all numbers of robots and tasks. Fewer robots resulted in lower success rates overall, for all numbers of tasks.

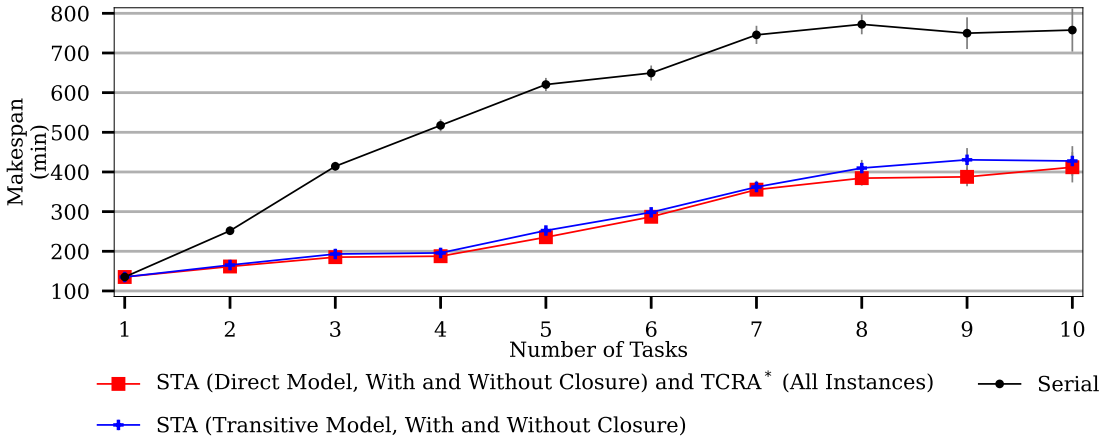


Figure 5.12: First Response TCRA* (all ϵ values), STA, and Serial makespan (min) for ten robots. The maximum makespan was 800 min.

STA using the direct model with closure was the best solution to the First Response Domain, as most problems were solved with the best makespan, for all numbers of robots and tasks. However, there are no quality guarantees, and STA can result in worse makespan. The second best solution was TCRA* ($\epsilon = 1$) using the direct model with closure, which achieved the best makespan and with a worse

success rate, for all numbers of robots and tasks. TCRA* ($\epsilon = 0$) using the direct model without the transitive closure was the worst solution, for all numbers of robots and tasks, as the best makespan was achieved, but with the worst success rates, for all numbers of robots and tasks.

5.4.4 Overall Results Across All Domains

The Serial Algorithm solved all problems across all domains for all numbers of robots and tasks, but always had the worst makespan. The makespan delta between the Serial and other algorithms grew with the number of tasks and robots, across all domains for all numbers of robots. TCRA*'s quality guarantees ensured the best makespan for all conflict models, with and without closure, across all domains, for all numbers of robots and tasks, and for all ϵ values. STA does not offer quality guarantees, and STA failed to achieve the best makespan using the transitive causal conflict model, with and without closure, across all domains, for all numbers of robots and tasks.

All TCRA* ($\epsilon = 0$) instances resulted in the worst success rates, for all conflict models, with and without closure, across all domains, for all numbers of robots and tasks. Higher TCRA* ϵ values were better overall, but $\epsilon > 1$ did not produce noticeably different results, for all conflict models, with and without closure, across all domains, for all numbers of robots and tasks. TCRA*'s and STA's success rates fell as the number of tasks increased, for all conflict models, with and without closure, across all domains, for all numbers of robots, and for all TCRA* ϵ values. More tasks resulted in more actions to merge and more conflicts to resolve, which

increased the computational complexity.

The direct model with closure produced the best results for both STA and TCRA* across all domains, for all numbers of robots and tasks, and for all TCRA* ϵ values. The direct model without closure performed the worst for both STA and TCRA*, across all domains, for all numbers of robots and tasks, and for all TCRA* ϵ values. The transitive model resulted in worse STA results, with and without closure, across all domains, for all numbers of robots and tasks. The transitive model improved TCRA*'s metrics, relative to the direct model without closure, across all domains, for all numbers of robots and tasks.

5.5 Discussion

The Serial Algorithm offers the best success rates, but does not minimize the resulting makespan, and fails to make effective use of additional robots. The serial plan execution causes a growing number of robots to wait and remain idle, as more robots and tasks are allocated, which results in longer makespans as the number of tasks increases. However, the Serial Algorithm is the best choice for a single robot, when the opportunity to parallelize action execution is limited, or when the makespan does not need to be minimized.

STA and TCRA*'s algorithmic complexity, presented in Chapter 5.2, is not a direct function of the number of tasks or the number of robots involved, but rather a function of the coupling between individual tasks and the overall problem model complexity. STA and TCRA* depend on the conflict identification and resolution models, in addition to the transitive closure. Thus, the STA and TCRA*'s com-

putational complexity, with respect to the number of robots and tasks, cannot be concisely formulated, and was evaluated empirically, through experiments.

STA and TCRA* account for action dependencies between task plans, which resulted in increased computational complexity and lower success rates. However, both the algorithms can leverage more robots to parallelize more action executions and reduce the resulting makespan. This added complexity translates into lower makespan and faster plan execution, as fewer robots are idle and more tasks execute simultaneously. The benefits are more pronounced as tasks and robots are added, and more actions can be parallelized.

TCRA* accounts for durative actions in order to minimize makespan and guarantees the minimum makespan when using $\epsilon \leq 1$. TCRA*'s optimality guarantee defines lower bounds on solution quality and provides a conceptual benchmark. Knowing the lowest possible makespan helps quantify the quality of suboptimal solutions, such as those produced by STA and the Serial Algorithm.

Early TCRA* experiments used the direct model without transitive closure, and failed to scale with a growing number of tasks [102]. The transitive model improved TCRA*'s ability to scale and solve a problems with a larger number robots and tasks. The transitive model identified more solutions for each causal conflict, but most solutions resulted in longer makespan and were pruned by TCRA*'s priority search queue, reducing the overall TCRA* branching factor. The transitive model conflict and solution identification algorithms have higher computational complexity than the direct model's; however, the net result across all domains was a higher success rate. The transitive causal conflict model impacted STA nega-

tively. The various solutions identified by the transitive model cause STA to waste computational resources, as STA does not prune solutions based on makespan.

The transitive closure introduces temporal orders that minimize the number of conflicts identified and benefited both TCRA* and STA. The transitive closure reduces the branching factor and does not increase the number of causal conflict solutions. Additional computational cost is necessary to perform the transitive closure, but the net results were better than the transitive model on all metrics for both TCRA* and STA.

The growing number of robots in the Logistics and First Response Domain reduced task coupling, benefiting STA and TCRA*. The high ratio of tasks per robot resulted in multiple tasks being allocated to each robot and created a tighter coupling between each task. Dependencies between task plans increased, causing more conflicts, and increasing the plan merging search space. The number of tasks allocated to each robot decreased as more robots were added. The additional robots reduced the coupling between tasks and the computational cost. Coupling between tasks was independent of the number of robots in the Blocks World Domain, as the stacking of blocks caused tight task coupling. The addition of robots reduced the number of tasks allocated to each robot, but did not reduce the number of blocks that were involved in multiple tasks. Tasks sharing the same blocks were tightly coupled, given the dependencies that exist when actions move a common block. As a result, STA and TCRA* did not benefit from the addition of robots.

The new conflict identification and resolution models and algorithms improved the overall plan merging performance. TCRA* and STA were able to scale to a

larger number of tasks and robots while generating high quality plans. TCRA*'s plan quality, represented by the resulting makespans, was not compromised and its computational cost was reduced, across all domains. This contribution allows multiple robot systems to synthesize plans and coordinate to resolve conflicts in a distributed manner, while guaranteeing the fastest plan execution duration. The plan merging and conflict resolution algorithms provide further ability to scale and solve larger, more complex task allocation and coordination problems.

5.6 Conclusion

A new family of plan coordination and conflict resolution algorithms, that scale to a large number of tasks and robots for complex problems, was introduced to merge independently generated plans and minimize the resulting makespan. Complexity analysis and experiments demonstrated that inadequate conflict models can severely impair the merging algorithms' ability to scale and solve problems with a growing number of tasks. The resulting plan quality and the computational cost was substantially improved by using the appropriate causal conflict models and merging algorithms.

TCRA* was introduced to merge independently generated plans and minimize the resulting makespan, while accounting for durative actions. TCRA*'s admissible heuristic prunes the search space and minimizes makespan optimally. A proof of optimality was provided and the algorithm was empirically evaluated across three multiple robot domains against two algorithms, the Serial Algorithm, and STA. STA was extracted from the PMA and used as a stand-alone algorithm that

does not minimize makespan. The conflict identification and resolution algorithms introduced allowed STA and TCRA* to scale and solve larger problems with more robots and tasks.

The Task Fusion heuristics improved coordination before planning, while the new plan merging and conflict resolution algorithms facilitated better coordination after planning and allowed the Coalition Formation then Planning framework to support uncertain action durations and minimize the resulting makespan. However, the simulated experiments fail to capture the complexity of real-world multiple robot systems and all of the associated unknowns.

Chapter 6: Multiple Robot System and Evaluation

Parallel plan execution on heterogeneous multiple robot systems requires the ability to scale and solve problems with an increasing number of robots and tasks. Restrictions to task allocation and plan coordination limited the scalability of existing methods, such as the Coalition Formation then Planning framework, presented in Chapter 2. The Coalition Formation then Planning framework limitations were mitigated by this dissertation's contributions to task allocation (Chapter 4) and plan merging (Chapter 5). Simulated experiments demonstrated that the Coalition Formation then Planning framework can scale when solving complex problems (Chapters 4 and 5). This Chapter demonstrates the capabilities of the Coalition Formation then Planning framework when deployed on physical robots solving real-world problems.

The multiple robot coordination system allows physically distributed mobile robots to deliberate independently, while coordinating at a high level of abstraction. The system facilitates real-world robot experiments, consisting of multiple trials, in order to evaluate the algorithms presented in Chapter 5. The system architecture is described, followed by experiments comparing the performance of the various merging algorithms.

6.1 Multiple Robot System Architecture

The multiple robot system's architecture is domain independent. The domain models and parameters are specified using the human-readable MAPL language, which is specified in Appendix A. The initial world state and the problem's goals are also specified using the MAPL language. The available robots' capabilities and initial states are gathered before planning in order to generate a planning problem, which is solved using the Coalition Formation then Planning framework. The robots receive the solution plan, follow the solution plan, and coordinate using their internal Robot Execution System.

6.1.1 The Robot Execution System

The Robot Execution System controls each robot's behavior at a high level during plan execution. Each robot executes an independent and distributed instance of the execution system, which is composed of multiple processes, called nodes. Nodes have individual responsibilities, and multiple nodes execute on each robot. The nodes communicate to one another, internal to a specific robot, using the Robot Operating System (ROS) [138]. ROS allows nodes to execute independently and permits integration with robust robot localization and navigation packages, such as the Adaptive Monte-Carlo Localization package [135]. Two nodes, the Communication Node and the Plan Execution Node, compose the Robot Execution System, as presented in Figure 6.1.

Robots use the Communication Node to request information from other robots,

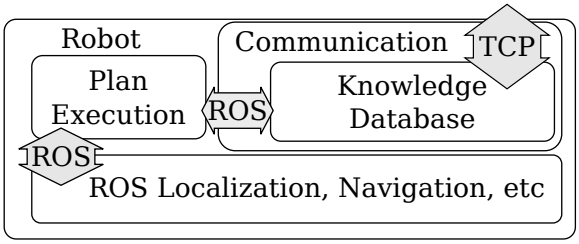


Figure 6.1: The Robot Execution System.

as ROS does not support multiple robots natively. The Communication Node uses the Transmission Control Protocol (TCP) [42] to exchange messages with other robots. A knowledge database includes all the information relevant for plan execution, including the global plan, and information communicated by other robots. A robot can query another robot, which in turn reads its internal knowledge database and sends a reply. Robots can also send specific information to other robots without being prompted. Robots are allowed a high level of autonomy during plan execution and the communication between robots is limited to disseminating plans and announcing action completions.

The Plan Execution Node commands a robot's actions. The Plan Execution Node interfaces with the Communication Node to obtain information about other robots, as shown in Figure 6.1, and standard ROS nodes for robot localization, navigation, etc. Plan execution begins as soon as a plan is received.

The Plan Execution Node implements the Plan Execution Algorithm, Algorithm 6.1, in order to coordinate plan execution. The pending action set, $A_{pending}$, represents the actions that remain to be executed, and is initialized with all the plan actions (line 1). The set of plan actions involving the robot, A_r, are iden-

tified (line 2). The set intersection between the pending actions and the robot actions, $A_{pending} \cap A_r$, represents the actions involving the robot that remain to be executed (i.e., the robot pending actions).

```
   Data: A multiagent plan π;
1  Initialize the pending action set A_pending with all the actions in plan π;
2  Identify the robots' actions, A_r, in plan π;
3  while A_pending ∩ A_r is not empty do
4  |  foreach action a ∈ A_pending ∩ A_r do
5  |  |  if action a does not have predecessors in A_pending then
6  |  |  |  Execute action a;
7  |  |  |  Remove action a from A_pending;
8  |  |  |  Announce that action a is completed;
9  |  Update the pending action set A_pending;
```

Algorithm 6.1: The Plan Execution Algorithm.

While there are robot pending actions (line 3), each pending action is evaluated (line 4). Actions that have no pending predecessors (line 5) are executed (line 6) and removed from the pending action set $A_{pending}$ (line 7). When the action is completed, it is announced (line 8) to all robots whose actions are successors of the completed action. The pending action set, $A_{pending}$, is updated by accounting for the action completion announcements issued by other robots (line 9). The predecessors of an action a are the actions a_0 that are ordered before action a, $a_0 \prec a$. The successors of an action a are the actions a_0 that are ordered after action a, $a \prec a_0$. The Plan Execution Algorithm allows parallel action execution as multiple robots execute their plan actions independently. The algorithm does not require a global coordination mechanism, as the robots execute their actions according to each action's individual predecessors.

6.1.2 The Safety Policy

Pioneer P3-DX robots, fitted with spinning laser range finders, were used for the experiments. Limited sensor accuracy can hinder the robots' ability to perceive other robots' positions while executing the plan. Collisions can occur when traversing narrow hallways, doorways, and navigating around obstacles; thus, a safety policy was developed in order to minimize collisions. Robots communicate their intent while moving between locations. The policy prevents multiple robots from moving across the same paths, improving safety.

The safety policy establishes a communication mechanism that prevents robots from moving simultaneously in collision-prone routes. A robot announces its origin and destination before moving and checks for other robots' announcements. Upon verifying that no other robot has claimed the same origin or destination, the robot proceeds to its destination and issues an arrival announcement. Upon detecting a conflict, the robot waits on other robots' arrival announcements before proceeding. The safety policy can increase the plan execution duration significantly, as more robots are forced to wait before moving between locations. The impacts of the policy on the overall safety and the plan execution duration are evaluated in the provided experiments (Chapter 6.2).

6.2 Experimental Methodology

The Coalition Formation then Planning framework limitations were mitigated by this dissertation's contributions to task allocation (Chapter 4) and plan merging

(Chapter 5). The experimental methodology was designed to evaluate the Coalition Formation then Planning framework, the plan merging algorithms (Chapter 5), and the effectiveness of the safety policy using a simulated real-world mission and multiple robots with specific capabilities. The experiments were performed in an indoor environment that offers a combination of large open spaces, narrow hallways, doorways, and an office space with three separate, but variable sized rooms.

A metric map of the environment was created using a Rao-Blackwellized particle filter simultaneous localization and mapping package [77]. The map was created in real-time by a single robot traversing the environment, while collecting range sensor and wheel odometry data. The map was partitioned into six separate areas using a Voronoi graph-based segmentation [30], as presented in Figure 6.2, resulting in a topological map of the environment that allows for high-level planning.

The reduced First Response Domain, described in Chapter 3.2.3, was used for the experiments. Each robot was assigned a rescuer, ambulance, or hazard collector role, which did not change, as shown in Figure 6.3. The robots are unable to physically load and unload the simulated victims or hazardous material objects, but they waited (5 seconds) and vocalized the completion of these and all tasks. The Festival speech synthesis system was used to vocalize the completion of each plan action [177].

The six robots, including two rescuers, two ambulances, and two hazard collectors, began each plan execution from the rescue base. Two victim rescue tasks and two hazard collection tasks were solved. The coalition formation algorithm

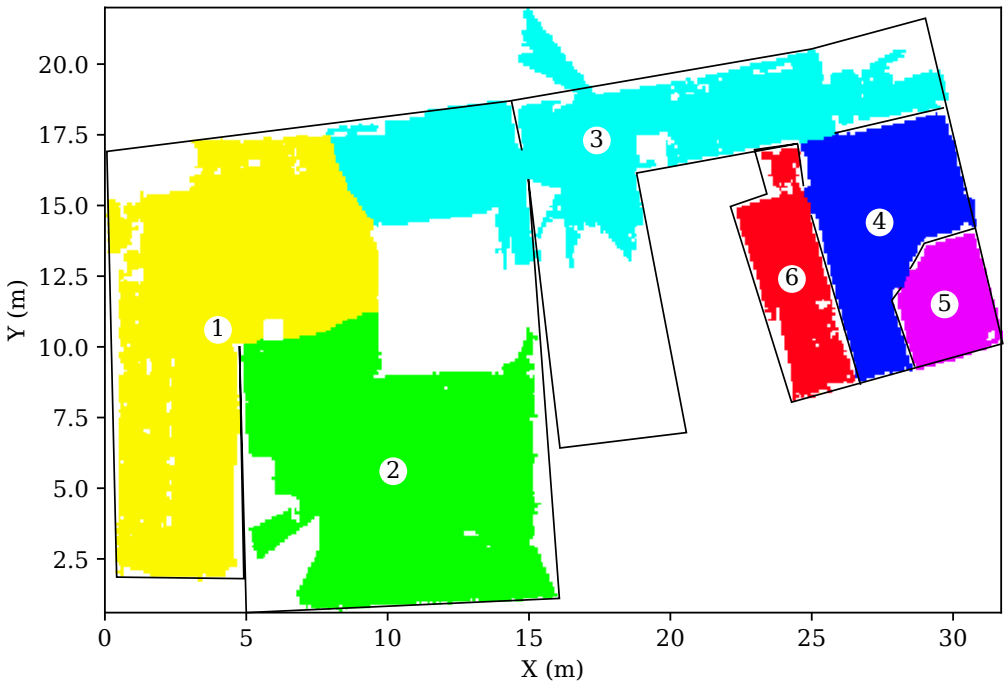


Figure 6.2: The metric map of the environment, segmented into six areas. Colors represent each area's navigable space. Black represents walls and white obstacles.

allocated each ambulance to each victim rescue task and each hazard collector to each hazard collection task. Rescuers were allocated two tasks each, as all tasks require a rescuer. The rescue base is located in Area 1, as presented in Figure 6.2. Victims are located in Areas 2 and 5. Hazards are located in Areas 3 and 6. An example plan, synthesized using the coalition and planning framework and the TCRA* merging algorithm, is shown in Figure 6.4. Ellipses represent actions and arrows represent the temporal orders. Actions are colored according to their tasks, shown in the legend.

Coupling between tasks arises from two primary sources. Locations are shared between tasks and multiple tasks are allocated to each rescuer, which generates

Figure 6.3: The rescue base. Each robot has a logo indicating their assigned roles.

dependencies and coupling. Shared locations between tasks and the multiple allocation of tasks to each rescuer generate dependencies and coupling. Robots must travel across common areas and the rescuer robots must clear each area before the arrival of the ambulance and hazard collection robots. Areas cleared while accomplishing one task are considered clear for the remainder of the trial. Thus, shared cleared areas generate temporal dependencies between task actions. Actions from multiple tasks become ordered after a clearing action. Note that both Victim Rescue Tasks have shared temporal orders, as shown in Figure 6.4. The "Rescuer 2: Clear Area" action (Victim Rescue Task 2), the second action from the top, is ordered before the action "Ambulance 1: Move to Area 2" (Victim Rescue Task 1). Ambulance 1 moves to Area 2 on its way to Area 5, as Area 2 is the closest

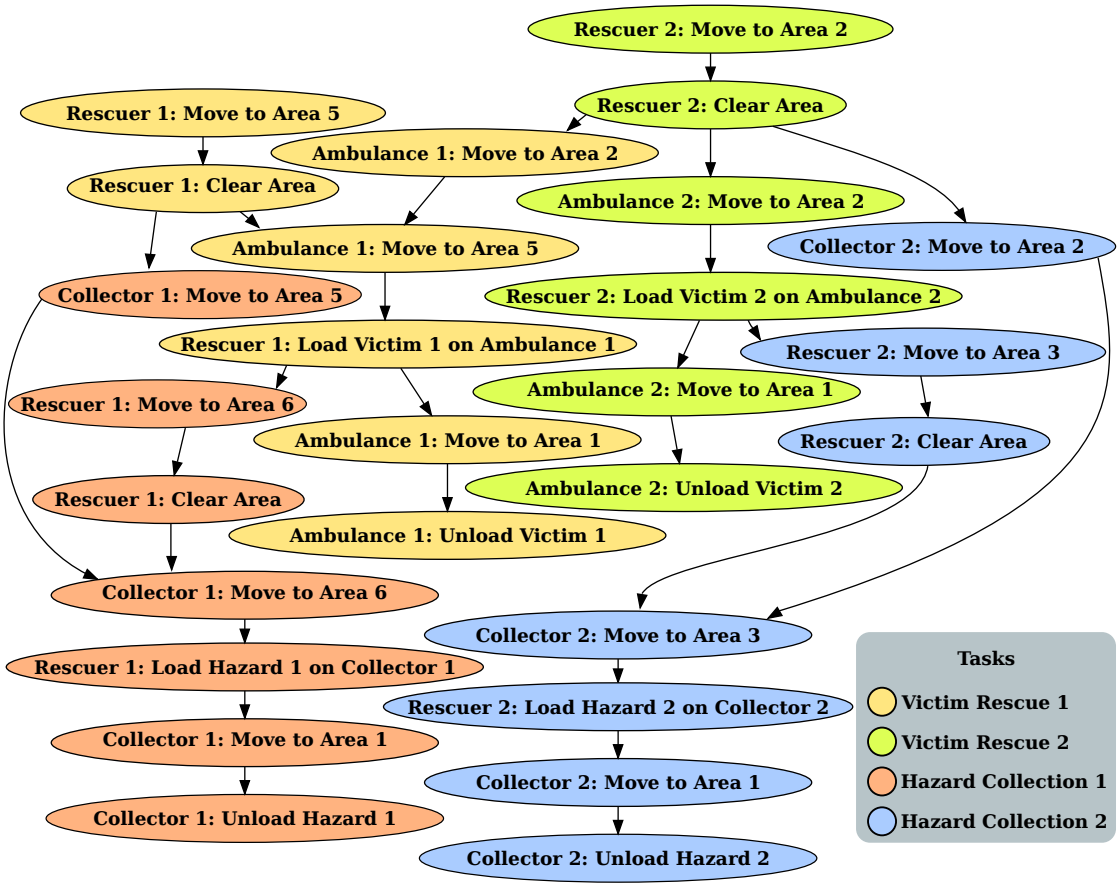


Figure 6.4: An example First Response plan synthesized by the Coalition Formation then Planning framework using the TCRA* plan merging algorithm.

cleared area to Area 5.

The multiple tasks allocated per rescuer also generate dependencies and coupling between tasks. Actions of an individual rescuer robot generate temporal orders between the rescuer's actions across its tasks. Rescuer 1 must finish loading Victim 1 on Ambulance 1, which is an action belonging to the Victim Rescue Task 1, before moving to Area 6, an action that belongs to the Hazard Collection Task 1, as presented in Figure 6.4. Action "Rescuer 1: Load Victim 1 on Ambulance

1" (Victim Rescue Task 1) is ordered before action "Rescuer 1: Move to Area 6" (Hazard Collection Task 1).

The coupling between tasks requires algorithms that systematically address conflicts while merging task plans, as presented in Chapter 5. Conflicts arise between plans' actions, as the effects of one action can make other actions infeasible. Plan merging algorithms are also responsible for minimizing the total plan execution duration, or makespan, while addressing plans' conflicts. The Serial, STA, and TCRA* ($\epsilon = 1$) merging algorithms, presented in Chapter 5, were evaluated in the experiments.

The robots were timed while executing their plans. The plan execution outcomes are: Success, all robots have completed their tasks and returned to the base successfully; and Failure, robots failed to finish within a one hour time limit or collided with one another and the experiment is terminated to prevent physical damage. All failures were due to collisions, as all trials finished with success or collision before the time limit expired. The rate of success and the makespan are the dependent variables.

The plan merging algorithm used to merge the various task plans, together with the safety policy, are the independent variables. The Serial, STA, and TCRA* merging algorithms were evaluated with the direct and transitive conflict models, with and without the transitive closure and the safety policy.

Tasks were allocated to robots according to the robots' capabilities and the tasks' requirements using a dynamic programming coalition formation algorithm [186]. Each task was individually solved using the Coalition Formation then Plan-

ning framework [62], and the Actions Concurrency and Time Uncertainty Planner (ActuPlan) [24]. ActuPlan was selected because it supports concurrent durative actions with uncertain action durations. Plans were generated once and executed multiple times. The simulated experiments were performed using the ActuPlan simulator [24], a high-level plan simulator.

Each plan merging algorithm instance synthesized a plan, producing a total of nine plans: (1) The TCRA* using the direct model without transitive closure; (2) The TCRA* using the direct model with transitive closure; (3) The TCRA* using the transitive model without transitive closure; (4) The TCRA* using the transitive model with transitive closure; (5) The STA using the direct model without transitive closure; (6) The STA using the direct model with transitive closure; (7) The STA using the transitive model without transitive closure; (8) The STA using the transitive model with transitive closure; and (9) The Serial Algorithm. Plans 1-6 were identical, as all TCRA* instances and the direct model STA resulted in the same plan. Plans 1-6 will be referred to as the TCRA* plan, even though the direct model STA also resulted in the same plan.

The TCRA* plan (Plans 1-6), the STA using the transitive model without transitive closure (Plan 7), the STA using the transitive model with transitive closure (Plan 8), and the Serial Algorithm Plan (Plan 9) were used in the simulated and the multiple robot experiments. The simulated experiment consisted of a thousand trials per plan, totaling four thousand trials. The multiple robot experiment consisted of twenty trials per plan, totaling eighty trials. Half of the multiple robot trials (ten trials per plan) used the safety policy.

The Kruskal–Wallis one-way analysis of variance and the pairwise Wilcoxon rank-sum tests were conducted in order to determine if there were statistically significant differences between the algorithm results [189].

6.3 Results

The TCRA* plan, which was identical for all TCRA* instances and for STA using the direct model, with and without closure, had the overall best makespan irrespective of the evaluation occurring in simulation or with the multiple robot system, and whether the multiple robot system used the safety policy or not, as shown in Tables 6.1, 6.2, 6.3, and Figure 6.5. STA using the transitive model without transitive closure resulted in the second best makespan and the STA using the transitive model with transitive closure resulted in the third best, for both the simulated and the multiple robot results, with and without the safety policy. The Serial plan was the worst overall.

The multiple robot system experiments demonstrated that the Coalition Formation then Planning framework and the plan merging algorithms behave similarly to the simulated environments. Physical aspects of the real-world system, such as the mobile robot dynamics, localization, path planning, and collision avoidance, are not modeled by the simulated experiments. Those aspects, during the multiple robot system experiments, resulted in an overall slower plan execution, with consistently longer makespans. However, these physical aspects did not impact the relative performance of the individual plan merging algorithms, as the same algorithms that performed best in simulation also performed best with the multiple

robot system.

Table 6.1: Simulated Makespan Descriptive Statistics.

Alg.	Model	Trans. Clos.	Mean	Std. Dev.	Median	Min	Max
TCRA*	**Direct**	**No**	10m25s	01m17s	10m25s	06m11s	14m38s
		Yes					
	Transitive	**No**					
		Yes					
STA	**Direct**	**No**					
		Yes					
	Transitive	**No**	12m27s	01m23s	12m27s	08m00s	17m16s
		Yes	14m09s	01m39s	14m04s	09m28s	19m39s
Serial	**N/A**	**N/A**	20m31s	01m57s	20m25s	14m48s	25m45s

Table 6.2: Multiple Robot Makespan Descriptive Statistics Without the Safety Policy.

Alg.	Model	Trans. Clos.	Mean	Std. Dev.	Median	Min	Max
TCRA*	**Direct**	**No**	12m46s	00m51s	12m49s	11m36s	14m15s
		Yes					
	Transitive	**No**					
		Yes					
STA	**Direct**	**No**					
		Yes					
	Transitive	**No**	15m56s	00m28s	16m01s	15m09s	16m41s
		Yes	18m26s	01m02s	18m10s	17m32s	20m42s
Serial	**N/A**	**N/A**	25m40s	01m22s	26m00s	23m35s	27m57s

The Kruskal–Wallis test found a statistically significant difference between the makespans for the simulated results ($p < 0.001$). The Wilcoxon rank-sum tests determined that the TCRA* plan had a significantly shorter makespan when compared to each of the other plans ($Z = 644185.0$, $p < 0.001$). The STA using

Table 6.3: Multiple Robot Makespan Descriptive Statistics Using the Safety Policy.

Alg.	Model	Trans. Clos.	Mean	Std. Dev.	Median	Min	Max
TCRA*	**Direct**	**No**	14m40s	01m32s	14m27s	12m30s	16m53s
		Yes					
	Transitive	**No**					
		Yes					
STA	**Direct**	**No**					
		Yes					
	Transitive	**No**	17m02s	01m30s	16m45s	15m25s	20m13s
		Yes	19m18s	01m45s	18m56s	17m50s	22m44s
Serial	**N/A**	**N/A**	25m49s	01m34s	25m55s	23m16s	28m03s

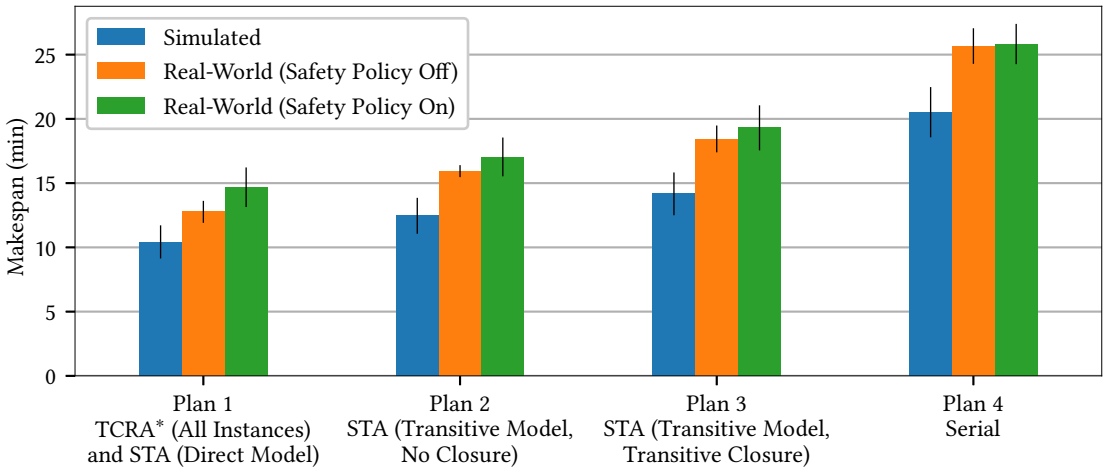


Figure 6.5: Simulated and Real-world Mean Makespan (min) Results.

the transitive model without transitive closure plan had a significantly shorter makespan when compared to the STA using the transitive model with transitive closure plan, as determined by the Wilcoxon rank-sum test ($Z = 1285111.0$, $p < 0.001$). The Wilcoxon rank-sum tests also determined that the STA using the transitive model with transitive closure plan had a significantly shorter makespan

when compared to the Serial Algorithm plan ($Z = 1493373.0$, $p < 0.001$).

A statistically significant difference was found by the Kruskal–Wallis test between the makespans for the multiple robot results without the safety policy ($p < 0.001$). The TCRA* plan had a significantly shorter makespan when compared to the STA using the transitive model without transitive closure plan, as determined by the Wilcoxon rank-sum test ($Z = 55.0$, $p < 0.001$). The Wilcoxon rank-sum tests determined that the STA using the transitive model without transitive closure plan had a significantly shorter makespan when compared to the STA using the transitive model with transitive closure plan ($Z = 155.0$, $p < 0.001$). The STA using the transitive model with transitive closure plan had a significantly shorter makespan when compared to the Serial Algorithm plan, as determined by the Wilcoxon rank-sum test ($Z = 155.0$, $p < 0.001$).

The makespans for the multiple robot results using the safety policy were also found to be significantly different by the Kruskal–Wallis test ($p < 0.001$). The Wilcoxon rank-sum tests determined that the TCRA* plan had a significantly shorter makespan when compared to all of the other plans ($Z = 69.0$, $p = 0.007$). The STA using the transitive model without transitive closure plan had a significantly shorter makespan when compared to the STA using the transitive model with transitive closure plan, as determined by the Wilcoxon rank-sum test ($Z = 67.0$, $p = 0.004$). The Wilcoxon rank-sum tests also determined that the STA using the transitive model with transitive closure plan had a significantly shorter makespan when compared to the Serial Algorithm plan ($Z = 155.0$, $p < 0.001$).

Not surprisingly, the safety policy resulted in higher success overall, as shown

in Table 6.4. Success was generally higher for plans with longer makespan. Longer makespans result from a more serialized action execution, which minimizes the chances of collisions between robots. Shorter makespans result from a highly parallelized plan action execution. Robots execute more actions concurrently; thus, more robots move simultaneously, more collisions occur, and more trials result in failure.

Table 6.4: Number of Multiple Robot Plan Execution Successful Trials.

	Safety Policy	
Plan	**Off**	**On**
TCRA* (All Instances) and STA (Direct Model)	5/10	9/10
STA (Transitive Model, No Closure)	6/10	8/10
STA (Transitive Model, Transitive Closure)	8/10	10/10
Serial	7/10	10/10

All the plans resulted in longer average makespans using the safety policy The differences were more pronounced for the plans that had better makespans, such as TCRA*'s. The safety policy forces robots to wait, serializing their actions. The worst plans, such as Serial's, are largely serialized already, and the safety policy's impact is lessened.

6.4 Discussion

TCRA* minimizes makespan directly and parallelizes action execution, resulting in the best makespans among the simulated and multiple robot results. STA searches for any conflict-free plan, does not minimize makespan, and can result in arbitrarily long makespans. STA's makespan results were influenced by the

conflict models and the transitive closure. The direct model STA resulted in the same makespan as TCRA*, whereas the transitive model was significantly worse. The Serial Algorithm serialized all tasks' actions naively and resulted in the worst simulated and multiple robot makespans. The multiple robot experimental results support the simulated results, and the same patterns were observed across both the multiple robot and the simulated results.

The experiments have proven the viability of the Coalition Formation then Planning framework to solve multiple robot tasks using heterogeneous multiple robot systems. The experimental system supported controlled repeated trials that compared robot coordination algorithms. The results show noticeable differences across the evaluated merging algorithms. The robots' success and the overall plan execution duration (i.e., makespan) significantly varied per merging algorithm. The most aggressive algorithms highly parallelized plans' actions, which increased the chances of collisions, as the robots had limited sensing abilities. The safety policy allowed the robots to reliably and repeatably execute their tasks, and allowed some plans to succeed on all experiments.

TCRA* offers the best plan execution duration, but exposes the robots to higher risks and results in more collisions. STA and the Serial Algorithm result in longer plan execution, but fewer collisions. The safety trade-off is largely due to the limited robot sensors. The inaccurate low cost laser range finders made the robot's perception of each other unreliable. The sensor's accuracy degraded with distance, in addition to the low update rate of 10 Hz cause difficulties detecting other robots. The localization algorithms also often diverge as the robot moves,

leading to collisions with other robots. More accurate laser range finders can mitigate collisions and failures.

The multiple robot results align with the simulated results, as the multiple robot makespans were consistently longer than their simulated counterparts. Imperfect robot sensors, mapping, localization, and navigation resulted in delayed action execution. Robots often need to perform localization recovery behaviors in order to localize themselves before moving. Robot traffic also incurs delays, as robots often stop and slow down to avoid collisions. None of these factors are modeled by the ActuPlan simulator; thus, the simulated experiments underestimate multiple robot makespans.

6.5 Conclusion

A multiple robot coordination system was introduced to evaluate the Coalition Formation then Planning framework. The system supported experimentation with a large number of trials, despite limitations in the robots' sensors, mapping, localization, and navigation accuracy. The simulated and multiple robot experiments evaluated different plan merging algorithms, described in Chapter 5. The multiple robot results were consistent with the simulated results. All instances of the TCRA* algorithm maximized simultaneous action execution and resulted in the shortest plan execution duration. STA and the Serial Algorithm resulted in longer plan execution duration, but fewer collisions.

Chapter 7: Conclusions

The fast-paced development of sensing, processing, and actuation devices at increasingly lower costs is resulting in robots with a growing variety of capabilities. Robots have proven potential to assist in response to major disasters, performing tasks, such as search and rescue and hazardous materials disposal, but currently highly trained operators make most decisions, while the robot decision making is limited to low level actions [4]. Exploiting the potential of autonomous robots will require scalable domain-independent task allocation and automated planning capable of modeling complex problems that incorporate heterogeneous robots. Real-world problems require concurrent actions (e.g., simultaneously triaging multiple victims), durative actions (e.g., travel times), uncertain action durations (e.g., unexpected travel delays), and a number of other features that increase the problem complexity. Dynamic and uncertain environments require domain-independent task allocation and planning approaches, in order to be applied to a variety of time-sensitive domains.

This dissertation contributes to task allocation and plan coordination. The domain-independent Coalition Formation then Planning framework improved scalability to a large number of robots, while supporting concurrent durative actions, by factoring the plan synthesis process [62]. Planning for tasks separately reduces the overall computational complexity, but limits cooperation between robots, lowering plan quality, and requiring more time to execute. This dissertation's contri-

butions mitigated the limitations of the Coalition Formation then Planning framework by improving task allocation, incorporating uncertain action durations, and improving the framework's ability to scale to a larger number of robots and tasks across domains.

7.1 Contributions

The first primary contribution is a significant improvement to task allocation while planning for multiple heterogeneous robot problems. A new family of coordination heuristics employed plan similarity concepts to fuse tasks and robot coalitions in order to improve the resulting plan quality and reduce the computational cost when allocating tasks. As a result, larger problems can be solved faster, requiring fewer computational resources for a larger number of tasks and robots. This contribution allows robots to devise plans autonomously and solve complex problems while minimizing the plan execution duration. The reductions in plan execution durations can make the difference between mission success or mission failure, especially in complex real-world time-sensitive domains. Real-world deployments now can solve plans faster and more quickly react to dynamic situations.

The second primary contribution is the $TCRA^*$ algorithm that allows robots to coordinate plans generated independently, while accounting for uncertain action durations. $TCRA^*$ is the first plan merging approach to account for action durations and minimize makespan optimally, the total time required to execute the plan. $TCRA^*$ is guaranteed to minimize makespan optimality and establishes a benchmark for minimizing makespan. Minimizing makespan is critical for real-

world domains, where different actions require different completion times. This contribution is relevant to first response, where indefinite processing time and memory are not available and life saving actions must be taken, as a reduction in makespan can offset the added computation time. The contribution is also important when plans can be generated in advance, but the plan execution is time critical. An example of such a domain is extraplanetary robot missions [21], where the robots' operation is limited to planetary daytime cycles due to solar power limitation and the plans can be generated overnight.

The third primary contribution introduced new conflict models and a family of conflict resolution algorithms that allow robots to solve conflicts more effectively while coordinating individual plans. The algorithms increased the number of tasks and robots supported, while synthesizing complex plans for multiple heterogeneous robots. This contribution demonstrated how previously overlooked assumptions about conflict resolution models can impair how the robots' ability to solve more complex problems with an increasing number of tasks and robots.

A secondary contribution extended the Coalition Formation then Planning framework in order to support uncertain action durations and introduced the MAPL language. The MAPL software suite compiles and decompiles domain descriptions, problems, and plans. The compiler generates semantic models, which are necessary to the Coalition Formation then Planning framework. MAPL is the first planning language to support coalition formation models with uncertain action durations. The human-readable language allows specifying the robot capabilities, the planning domain model, task goals, and task requirements. The full language

specification, developed as part of this dissertation, is presented in Appendix A.

Another secondary contribution introduced the first real-world multiple robot system to deploy the Coalition Formation then Planning framework. The system was capable of reliably executing complex plans, and experiments compared the new plan coordination and conflict resolution algorithms.

The proliferation of multiple robot systems will increase the demand for planning and decision-making systems capable of scaling to a growing number of tasks and robots. Heterogeneous systems and the diversifying portfolio of robot capabilities will require supporting increasing domain complexity and independence in order to adapt to dynamic and uncertain domains and environments. This dissertation contributed new domain-independent approaches that can be used for a range of real-world environments and domains; thus, moving the field closer to the ability to truly deploy robots for complex real-world missions, and solve larger, more complex task allocation and coordination problems.

7.2 Future Work

Real-world missions contain multiple sources of uncertainty, including uncertain action outcomes and uncertain action durations. The plan merging algorithms introduced in Chapter 5 support durative actions and uncertain action durations, allowing the Coalition Formation then Planning framework to operate under temporal uncertainty, but do not address uncertain action outcomes. Future research needs to extend the plan merging algorithms to support uncertain action outcomes, which require accommodating contingencies for all known action outcomes.

Solutions for uncertain action outcomes can be a decision tree or a finite-state controller, as presented in Chapter 2. Existing methods cannot scale to a large number of robots due to the exponential growth in the action outcomes. Each solution type imposes specific challenges on plan merging that will require new algorithms.

The Task Fusion algorithm considers only pairwise (binary) coalition fusion in order to avoid the complexity of evaluating all possible coalition combinations. Investigating algorithm alternatives, or extensions to support n-ary fusions constitutes future Task Fusion research. Additionally, algorithms are needed that can merge and generate relaxed plans for all $\binom{m}{2}$ pairs of coalition-task pairs, from the original set of m coalition-task pairs. New heuristics can compare the resulting plan quality and computational cost to the original coalition-task relaxed plans; however, several issues limit this approach. The first drawback is that a combinatorial number of relaxed plans will be generated, which does not scale linearly with the number of robots, and can jeopardize overall scalability. Potential approaches to overcome this issue include heuristics that can selectively prune unpromising pairs. The second limitation is that greedily minimizing each coalition-task pair's makespan and number of actions does not guarantee minimizing the resulting global plan's makespan or number of actions. Potential approaches to overcome this issue include merging the relaxed plans minimizing the global makespan directly. However, the cost of merging all relaxed plans can significantly increase the overall computational cost. Lastly, the processing time and the memory usage necessary to generate relaxed plans does not necessarily correlate to the compu-

tational cost required to generate full plans, limiting the relaxed plan heuristics accuracy. Entirely new heuristics can result in more effective Task Fusion.

The plan merging computational cost can be improved by leveraging domain knowledge, such as task ordering [159]. Coalition formation algorithms take task ordering constraints into account in order to allocate tasks [159]. Future research can leverage the coalition formation ordering restrictions in order to improve plan merging. Imposing task ordering constraints can reduce the plan merging search space and the overall computational complexity. A hybrid algorithm that combines TCRA* and the Serial Algorithm can selectively serialize tasks before resolving conflicts. Fewer conflicts will remain after the serialization, which will reduce the computational cost. Coalition formation algorithms also support online task discovery [148, 188]. Tasks are taken into account dynamically, as they arrive. Future work can extend plan merging algorithms to support discovery and merge new plans as they arrive. The effectiveness of the existing plan merging algorithms can be impacted by online task discovery, and new algorithms can be developed, such as iterative versions of TCRA*.

Bibliography

[1] Matthew Crosby, Michael Rovatsos, and Ronald P. A. Petrick. "Automated agent decomposition for classical planning." *International Conference on Automated Planning and Scheduling*. 2013, pp. 46–54.

[2] Sherief Abdallah and Victor Lesser. "Organization-based cooperative coalition formation." *IEEE/WIC/ACM International Conference on Intelligent Agent Technology*. 2004, pp. 162–168.

[3] Veena G. Adlakha and Vidyadhar Kulkarni. "A classified bibliography of research on stochastic PERT networks: 1966-1987." *INFOR: Information Systems and Operational Research* 27.3 (1989), pp. 272–296.

[4] Ron Alterovitz, Sven Koenig, and Maxim Likhachev. "Robot planning in the real world: Research challenges and opportunities." *AI Magazine* 37.2 (2016), pp. 76–84.

[5] Christopher Amato, Alan Carlin, and Shlomo Zilberstein. "Bounded Dynamic Programming for Decentralized POMDPs." *Workshop at the International Conference on Autonomous Agents and Multi-Agent Systems*. 2007.

[6] Christopher Amato, Daniel S. Bernstein, and Shlomo Zilberstein. "Optimizing Memory-bounded Controllers for Decentralized POMDPs." *Conference on Uncertainty in Artificial Intelligence*. 2007, pp. 1–8.

[7] Chistopher Amato, Jilles S. Dibangoye, and Shlomo Zilberstein. "Incremental Policy Generation for Finite-Horizon DEC-POMDPs." *International Conference on Automated Planning and Scheduling*. 2009, pp. 2–9.

[8] Christopher Amato and Shlomo Zilberstein. "Achieving Goals in Decentralized POMDPs." *International Conference on Autonomous Agents and Multiagent Systems*. 2009, pp. 593–600.

[9] Christopher Amato, Daniel S. Bernstein, and Shlomo Zilberstein. "Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs." *Autonomous Agents and Multi-Agent Systems* 21.3 (2010), pp. 293–320.

[10] Christopher Amato, Blai Bonet, and Shlomo Zilberstein. "Finite-State Controllers Based on Mealy Machines for Centralized and Decentralized POMDPs." *AAAI Conference on Artificial Intelligence*. 2010, pp. 1052–1058.

[11] Christopher Amato, Girish Chowdhary, Alborz Geramifard, N. Kemal Ure, and Mykel J. Kochenderfer. "Decentralized control of partially observable Markov decision processes." *IEEE Conference on Decision and Control*. 2013, pp. 2398–2405.

[12] Christopher Amato, George D. Konidaris, and Leslie P. Kaelbling. "Planning with macro-actions in decentralized POMDPs." *International Conference on Autonomous Agents and Multiagent Systems*. 2014, pp. 1273–1280.

[13] Christopher Amato, George Konidaris, Gabriel Cruz, Christopher A. Maynor, Jonathan P. How, and Leslie P. Kaelbling. "Planning for decentralized control of multiple robots under uncertainty." *IEEE International Conference on Robotics and Automation*. 2015, pp. 1241–1248.

[14] Christopher Amato. "Cooperative decision making." *Decision Making Under Uncertainty: Theory and Application*. Ed. by Mykel J. Kochenderfer. 2015, pp. 159–187.

[15] Christopher Amato, George Konidaris, Ariel Anders, Gabriel Cruz, Jonathan P. How, and Leslie P. Kaelbling. "Policy search for multi-robot coordination under uncertainty." *International Journal of Robotics Research* 35.14 (2016), pp. 1760–1778.

[16] Raghav Aras, Alain Dutech, and Francois Charpillet. "Mixed integer linear programming for exact finite-horizon planning in decentralized POMDPs." *International Conference on Automated Planning and Scheduling*. 2007, pp. 18–25.

[17] Manuel Arias Calleja, Francisco J. Diez Vegas, and Miguel A. Palacios Alonso. *ProbModelXML: A format for encoding probabilistic graphical models*. Tech. rep. 2012, pp. 15–67.

[18] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. "Complexity of Finding Embeddings in a k-Tree." *SIAM Journal on Algebraic Discrete Methods* 8.2 (1987), pp. 277–284.

[19] Michael Athans. "The role and use of the stochastic linear-quadratic-Gaussian problem in control system design." *IEEE Transactions on Automatic Control* 16.6 (1971), pp. 529–552.

[20] Christer Backstrom. "Computational Aspects of Reordering Plans." *Journal of Artificial Intelligence Research* 9 (1998), pp. 99–137.

[21] Max Bajracharya, Mark W. Maimone, and Daniel M. Helmick. "Autonomy for Mars Rovers: Past, Present, and Future." *IEEE Computer* 41.12 (2008), pp. 44–50.

[22] Eric Beaudry, Froduald Kabanza, and Francois Michaud. "Planning for concurrent action executions under action duration uncertainty using dynamically generated Bayesian networks." *International Conference on Automated Planning and Scheduling*. 2010, pp. 10–17.

[23] Eric Beaudry, Froduald Kabanza, and Francois Michaud. "Planning with Concurrency under Resources and Time Uncertainty." *European Conference on Artificial Intelligence*. 2010, pp. 217–222.

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

[25] Daniel S. Bernstein, Eric A. Hansen, and Shlomo Zilberstein. "Bounded Policy Iteration for Decentralized POMDPs." *International Joint Conference on Artificial Intelligence*. 2005, pp. 1287–1292.

[26] Daniel S. Bernstein, Christopher Amato, Eric A. Hansen, and Shlomo Zilberstein. "Policy iteration for decentralized control of Markov decision processes." *Journal of Artificial Intelligence Research* 34.1 (2009), pp. 89–132.

[27] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-dynamic programming*. 1st. Nashua, New Hampshire, 1996, pp. 560–564.

[28] Avrim L. Blum and Merrick L. Furst. "Fast planning through planning graph analysis." *Artificial Intelligence* 90.1-2 (1997), pp. 281–300.

[29] Blai Bonet and Hector Geffner. "Labeled RTDP: Improving the convergence of real-time dynamic programming." *International Conference on Automated Planning and Scheduling*. 2003, pp. 12–21.

[30] Richard Bormann, Florian Jordan, Wenzhe Li, Joshua Hampp, and Martin Hägele. "Room segmentation: Survey, implementation, and analysis." *International Conference on Robotics and Automation*. Ed. by Danica Kragic, Antonio Bicchi, and Alessandro De Luca. 2016, pp. 1019–1026.

[31] Daniel Borrajo. "Multi-agent planning by plan reuse." *International Conference on Autonomous Agents and Multi-Agent Systems*. 2013, pp. 1141–1142.

[32] Felix Bose, Jakub Piotrowski, and Bernd Scholz-Reiter. "Autonomously controlled storage management in vehicle logistics - applications of RFID and mobile computing systems." *International Journal of RF Technologies: Research and Applications* 1.1 (2009), pp. 57–76.

[33] Abdeslam Boularias and Brahim Chaib Draa. "Exact Dynamic Programming for Decentralized POMDPs with Lossless Policy Compression." *International Conference on Automated Planning and Scheduling*. 2008, pp. 20–27.

[34] Craig Boutilier and Ronen I. Brafman. "Partial-Order Planning with Concurrent Interacting Actions." *Journal of Artificial Intelligence Research* 14 (2001), pp. 105–136.

[35] Craig Boutilier, Raymond Reiter, and Bob Price. "Symbolic Dynamic Programming for First-Order MDPs." *International Joint Conference on Artificial Intelligence*. 2001, pp. 690–700.

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

[37] John Bresina, Richard Dearden, Nicolas Meuleau, Sailesh Ramakrishnan, David Smith, and Rich Washington. "Planning Under Continuous Time and Resource Uncertainty: A Challenge for AI." *Conference on Uncertainty in Artificial Intelligence*. 2002, pp. 77–84.

[38] Daniel Bryce, Sicun Gao, David J. Musliner, and Robert P. Goldman. "SMT-based nonlinear PDDL+ planning." *AAAI Conference on Artificial Intelligence*. 2015, pp. 3247–3253.

[39] Olivier Buffet and Douglas Aberdeen. "The factored policy-gradient planner." *Artificial Intelligence* 173.5-6 (2009), pp. 722–747.

[40] Pieter Buzing, Adriaan Mors, Jeroen Valk, and Cees Witteveen. "Coordinating Self-interested Planning Agents." *Autonomous Agents and Multi-Agent Systems* 12.2 (2006), pp. 199–218.

[41] Alan Carlin and Shlomo Zilberstein. "Value-Based Observation Compression for DEC-POMDPs." *International Conference on Autonomous Agents and Multiagent Systems*. 2008, pp. 501–508.

[42] Vinton G. Cerf and Robert E. Kahn. "A protocol for packet network intercommunication." *IEEE Transactions on Computing* 22 (1974), pp. 637–648.

[43] Alessandro Cimatti, Andrea Micheli, and Marco Roveri. "Strong Temporal Planning with Uncontrollable Durations: A State-Space Approach." *AAAI Conference on Artificial Intelligence*. 2015, pp. 3254–3260.

[44] Alessandro Cimatti, Minh B. Do, Andrea Micheli, Marco Roveri, and David E. Smith. "Strong temporal planning with uncontrollable durations." *Artificial Intelligence* 256 (2018), pp. 1–34.

[45] Madison Clark-Turner and Christopher Amato. "COG-DICE: An Algorithm for Solving Continuous-Observation Dec-POMDPs." *International Joint Conference on Artificial Intelligence*. 2017, pp. 4573–4579.

[46] Amanda J. Coles, Andrew I. Coles, Maria Fox, and Derek Long. "Forward-chaining partial-order planning." *International Conference on Automated Planning and Scheduling*. 2010, pp. 42–49.

[47] Amanda J. Coles, Andrew I. Coles, Maria Fox, and Derek Long. "POPF2: A forward-chaining partial order planner." *The International Planning Competition*. 2011, pp. 65–70.

[48] Amanda J. Coles, Andrew I. Coles, Angel Garcia-Olaya, Sergio Jimenez, Carlos Linares Lopez, Scott Sanner, and Sungwook Yoon. "A survey of the seventh International Planning Competition." *AI Magazine* 33.1 (2012), pp. 83–88.

[49] Amanda J. Coles, Andrew I. Coles, Maria Fox, and Derek Long. "COLIN: Planning with continuous linear numeric change." *Journal of Artificial Intelligence Research* 44 (2012), pp. 1–96.

[50] Alexandra Coman and Hector Munoz-Avila. "Generating Diverse Plans Using Quantitative and Qualitative Plan Distance Metrics." *AAAI Conference on Artificial Intelligence*. 2011, pp. 946–951.

[51] Jeffrey S. Cox and Edmund H. Durfee. "An efficient algorithm for multiagent plan coordination." *International Conference on Autonomous Agents and Multi-Agent Systems*. 2005, pp. 828–835.

[52] Jeffrey S. Cox and Edmund H. Durfee. "Efficient and distributable methods for solving the multiagent plan coordination problem." *Multiagent and Grid Systems* 5.4 (2009), pp. 373–408.

[53] Matthew Crosby, Ronald P. A. Petrick, Francesco Rovida, and Volker Krueger. "Integrating Mission and Task Planning in an Industrial Robotics Framework." *International Conference on Automated Planning and Scheduling*. 2017, pp. 471–479.

[54] William Cushing, Subbarao Kambhampati, Mausam, and Daniel S. Weld. "When is temporal planning really temporal?" *International Joint Conference on Artifical Intelligence*. 2007, pp. 1852–1859.

[55] Jilles S. Dibangoye, Abdel-Illah Mouaddib, and Brahim Chai Draa. "Point-Based Incremental Pruning Heuristic for Solving Finite-Horizon DEC-POMDPs." *International Conference on Autonomous Agents and Multi-agent Systems*. 2009, pp. 569–576.

[56] Jilles S. Dibangoye, Christopher Amato, Olivier Buffet, and Francois Charpillet. "Optimally solving Dec-POMDPs as continuous-state MDPs." *International Joint Conference on Artificial Intelligence*. 2013, pp. 90–96.

[57] Jilles S. Dibangoye, Christopher Amato, Olivier Buffet, and Francois Charpillet. *Optimally solving Dec-POMDPs as continuous-state MDPs: Theory and algorithms*. Tech. rep. 2014.

[58] Jilles S. Dibangoye, Olivier Buffet, and Francois Charpillet. "Error-Bounded Approximations for Infinite-Horizon Discounted Decentralized POMDPs." *Machine Learning and Knowledge Discovery in Databases*. Ed. by Toon Calders, Floriana Esposito, Eyke Hullermeier, and Rosa Meo. 2014, pp. 338–353.

[59] Jilles S. Dibangoye, Christopher Amato, Olivier Buffet, and Francois Charpillet. "Optimally solving Dec-POMDPs as continuous-state MDPs." *Journal of Artificial Intelligence Research* 55 (2016), pp. 443–497.

[60] Yannis Dimopoulos, Muhammad A. Hashmi, and Pavlos Moraitis. "mu-SATPLAN: Multi-agent planning as satisfiability." *Knowledge-Based Systems* 29 (2012), pp. 54–62.

[61] Anton Dukeman. "Hybrid mission planning with coalition formation." PhD dissertation. Vanderbilt University, 2017.

[62] Anton Dukeman and Julie A. Adams. "Hybrid mission planning with coalition formation." *Journal of Autonomous Agents and Multi-Agent Systems* 31.6 (2017), pp. 1424–1466.

[63] Edmund H. Durfee and Shlomo Zilberstein. "Multiagent planning, control, and execution." *Multiagent systems*. Ed. by Gerhard Weiss. 2013, pp. 485–545.

[64] Filip Dvorak, Arthur Bit-Monnot, Felix Ingrand, and Malik Ghallab. "A Flexible ANML Actor and Planner in Robotics." *Workshops at the International Conference on Automated Planning and Scheduling*. 2014.

[65] Baris Eker and H. Levent Akin. "Using evolution strategies to solve DEC-POMDP problems." *Soft Computing - A Fusion of Foundations, Methodologies and Applications* 14.1 (2010), pp. 35–47.

[66] Baris Eker, Ergin Ozkucur, Cetin Mericli, Tekin Mericli, and H. Levent Akin. "A finite horizon DEC-POMDP approach to multi-robot task learning." *International Conference on Application of Information and Communication Technologies*. 2011, pp. 1–5.

[67] Baris Eker. "Evolutionary Algorithms for Solving Dec-POMDP Problems." PhD dissertation. Bogazici University, 2012.

[68] Baris Eker and H. Levent Akin. "Solving decentralized POMDP problems using genetic algorithms." *Autonomous Agents and Multi-Agent Systems* 27.1 (2013), pp. 161–196.

[69] Rosemary Emery-Montemerlo, Geoff Gordon, Jeff Schneider, and Sebastian Thrun. "Approximate Solutions for Partially Observable Stochastic Games with Common Payoffs." *International Conference on Autonomous Agents and Multiagent Systems*. 2004, pp. 136–143.

[70] Eithan Ephrati and Jeffrey S. Rosenschein. "Divide and conquer in multi-agent planning." *AAAI Conference on Artificial Intelligence*. 1994, pp. 375–380.

[71] Michael A. Erdmann and Matthew T. Mason. "Exploration of sensorless manipulation." *IEEE Journal of Robotics and Automation* 4 (1988), pp. 369–379.

[72] Kutluhan Erol, Dana S. Nau, and V.S. Subrahmanian. "On the complexity of domain-independent planning." *National Conference on Artificial Intelligence*. 1992, pp. 381–386.

[73] Patrick Eyerich, Robert Mattmuller, and Gabriele Roger. "Using the context-enhanced additive heuristic for temporal and numeric planning." *Towards Service Robots for Everyday Environments*. Ed. by Erwin Prassler, Rainer Bischoff, Wolfram Burgard, Robert Haschke, Martin Hagele, Gisbert Lawitzky, Bernhard Nebel, Paul Ploger, Ulrich Reiser, and Marius Zollner. 2012, pp. 49–64.

[74] Matthew E. Gaston and Marie desJardins. "Agent-organized networks for dynamic team formation." *International Conference on Autonomous Agents and Multiagent Systems*. 2005, pp. 230–237.

[75] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Amsterdam, Netherlands, 2004.

[76] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning and acting*. New York, New York, 2016.

[77] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters." *IEEE Transactions on Robotics* 23.1 (2007), pp. 34–46.

[78] Naresh Gupta and Dana S. Nau. "On the complexity of blocks-world planning." *Artificial Intelligence* 56.2-3 (1992), pp. 223–254.

[79] Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. "Dynamic Programming for Partially Observable Stochastic Games." *National Conference on Artificial Intelligence*. 2004, pp. 709–715.

[80] Musad A. Haque and Magnus Egerstedt. "Coalition formation in multiagent systems based on bottlenose dolphin alliances." *American Control Conference*. 2009, pp. 3280–3285.

[81] Malte Helmert. "The fast downward planning system." *Journal of Artificial Intelligence Research* 26 (2006), pp. 191–246.

[82] Jesse Hoey, Robert St-Aubin, Alan Hu, and Craig Boutilier. "SPUDD: Stochastic planning using decision diagrams." *Conference on Uncertainty in Artificial Intelligence*. 1999, pp. 279–288.

[83] Jorg Hoffmann. "FF: The fast-forward planning system." *AI magazine* 22.3 (2001), pp. 57–62.

[84] Leslie P. Kaelbling, Michael L. Littman, and Anthony R. Cassandra. "Planning and acting in partially observable stochastic domains." *Artificial Intelligence* 101.1-2 (1998), pp. 99–134.

[85] Subbarao Kambhampati, Mark Cutkosky, Marty Tenenbaum, and Soo H. Lee. "Combining specialized reasoners and general purpose planners: a case study." *AAAI Conference on Artificial Intelligence*. 1991, pp. 199–205.

[86] Henry A. Kautz and Bart Selman. "Planning as satisfiability." *AAAI Conference on Artificial Intelligence*. 1992, pp. 359–363.

[87] Thomas Keller and Patrick Eyerich. "PROST: Probabilistic planning based on UCT." *International Conference on Automated Planning and Scheduling*. 2012, pp. 119–127.

[88] Levente Kocsis and Csaba Szepesvari. "Bandit based Monte-Carlo planning." *European Conference on Machine Learning*. 2006, pp. 282–203.

[89] Mary Koes, Illah Nourbakhsh, and Katia Sycara. "Heterogeneous multirobot coordination with spatial and temporal constraints." *National Conference on Artificial Intelligence*. 2005, pp. 1292–1297.

[90] Akshat Kumar and Shlomo Zilberstein. "Point-Based Backup for Decentralized POMDPs: Complexity and New Algorithms." *International Conference on Autonomous Agents and Multiagent Systems*. 2010, pp. 1315–1322.

[91] Akshat Kumar and Shlomo Zilberstein. "Anytime Planning for Decentralized POMDPs using Expectation Maximization." *Conference on Uncertainty in Artificial Intelligence*. 2010, pp. 294–301.

[92] Jan van Leeuwen, ed. *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*. Cambridge, Massachusetts, 1990.

[93] Joop M. I. M. Leo. "A general context-free parsing algorithm running in linear time on every LR(k) grammar without using lookahead." *Theoretical Computer Science* 82.1 (1991), pp. 165–176.

[94] Hector Levesque, Fiora Pirri, and Ray Reiter. "Foundations for the situation calculus." *Linkoping Electronic Articles in Computer and Information Science* 18 (1998), p. 18.

[95] Frank L. Lewis, Draguna L. Vrabie, and Vassilis L. Syrmos. *Optimal Control*. Hoboken, New Jersey, 2012.

[96] Leonid Libkin. *Elements of Finite Model Theory*. Berlin, Germany, 2004.

[97] Iain Little, Douglas Aberdeen, and Sylvie Thiebaux. "Prottle: a probabilistic temporal planner." *National Conference on Artificial Intelligence*. 2005, pp. 1181–1186.

[98] Miao Liu, Christopher Amato, Emily P. Anesta, J. Daniel Griffith, and Jonathan P. How. "Learning for Decentralized Control of Multiagent Systems in Large, Partially-observable Stochastic Environments." *AAAI Conference on Artificial Intelligence*. 2016, pp. 2523–2529.

[99] Derek Long and Maria Fox. "The 3rd International Planning Competition: Results and Analysis." *Journal of Artificial Intelligence Research* 20 (2003), pp. 1–59.

[100] Donald G. Malcolm, J. H. Roseboom, C. E. Clark, and Willard Fazar. "Application of a technique for research and development program evaluation." *Operations Research* 7.5 (1959), pp. 646–669.

[101] Gilberto Marcon dos Santos and Julie A. Adams. "Task Fusion Heuristics for Coalition Formation and Planning." *International Conference on Autonomous Agents and Multiagent Systems*. 2018, pp. 2198–2200.

[102] Gilberto Marcon dos Santos and Julie A. Adams. "Optimal Temporal Plan Merging." *International Conference on Autonomous Agents and Multiagent Systems*. 2020, pp. 851–859.

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

[104] Mausam and Andrey Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Williston, Vermont, 2012.

[105] Drew McDermott, Malik Ghallab, and Adele C. Howe. *PDDL-the planning domain definition language*. Tech. rep. 1998.

[106] Joao V. de Sousa Messias, Matthijs T. J. Spaan, and Pedro Lima. "GSMDPs for Multi-Robot Sequential Decision-Making." *AAAI Conference on Artificial Intelligence*. 2013, pp. 1408–1414.

[107] Joao V. de Sousa Messias. "Decision-Making under Uncertainty for Real Robot Teams." PhD dissertation. Instituto Superior Tecnico, 2014.

[108] Andrea Micheli, Minh B. Do, and David E. Smith. "Compiling Away Uncertainty in Strong Temporal Planning with Uncontrollable Durations." *International Joint Conference on Artificial Intelligence*. 2015, pp. 1631–1637.

[109] Adriaan ter Mors, Jeroen Valk, and Cees Witteveen. "Task coordination and decomposition in multi-actor planning systems." *Workshop on Software-Agents in Information Systems and Industrial Applications*. 2006, pp. 83–94.

[110] Kiriakos S. Mountakis, Tomas Klos, and Cees Witteveen. "Stochastic task networks: Trading performance for stability." *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. 2017, pp. 302–311.

[111] Lenka Mudrova, Bruno Lacerda, and Nick Hawes. "Partial order temporal plan merging for mobile robot tasks." *European Conference on Artificial Intelligence*. 2016, pp. 1537–1545.

[112] Christian J. Muise, Sheila A. McIlraith, and Christopher Beck. "Improved Non-Deterministic Planning by Exploiting State Relevance." *International Conference on Automated Planning and Scheduling*. 2012, pp. 172–180.

[113] Christian J. Muise, Vaishak Belle, and Sheila A. McIlraith. "Computing Contingent Plans via Fully Observable Non-deterministic Planning." *AAAI Conference on Artificial Intelligence*. 2014, pp. 2322–2329.

[114] Christian J. Muise, Paolo Felli, Tim Miller, Adrian R. Pearce, and Liz Sonenberg. "Leveraging FOND Planning Technology to Solve Multi-Agent Planning Problems." *Workshops at the International Conference on Automated Planning and Scheduling*. 2015, pp. 83–90.

[115] Ranjit Nair, Milind Tambe, Makoto Yokoo, David V. Pynadath, and Stacy Marsella. "Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings." *International Joint Conference on Artificial Intelligence*. 2003, pp. 705–711.

[116] Tuan A. Nguyen, Minh Do, Alfonso E. Gerevini, Ivan Serina, Biplav Srivastava, and Subbarao Kambhampati. "Generating diverse plans to handle unknown and partially known user preferences." *Artificial Intelligence* 190 (2012), pp. 1–31.

[117] Frans A. Oliehoek, Julian F. P. Kooij, and Nikos Vlassis. "The cross-entropy method for policy search in decentralized POMDPs." *Informatica* 32 (2008), pp. 341–357.

[118] Frans A. Oliehoek, Matthijs T. J. Spaan, and Nikos Vlassis. "Optimal and approximate Q-value functions for decentralized POMDPs." *Journal of Artificial Intelligence Research* 32 (2008), pp. 289–353.

[119] Frans A. Oliehoek, Matthijs T. J. Spaan, Shimon Whiteson, and Nikos Vlassis. "Exploiting locality of interaction in factored Dec-POMDPs." *International Conference on Autonomous Agents and Multiagent Systems*. Vol. 1. 2008, pp. 517–524.

[120] Frans A. Oliehoek, Shimon Whiteson, and Matthijs T. J. Spaan. "Lossless clustering of histories in decentralized POMDPs." *International Conference on Autonomous Agents and Multiagent Systems*. Vol. 1. 2009, pp. 577–584.

[121] Frans A. Oliehoek, Matthijs T. J. Spaan, Jilles S. Dibangoye, and Christopher Amato. "Heuristic Search for Identical Payoff Bayesian Games." *International Conference on Autonomous Agents and Multiagent Systems*. 2010, pp. 1115–1122.

[122] Frans A. Oliehoek. "Decentralized POMDPS." *Reinforcement Learning* 12 (2012), pp. 471–503.

[123] Frans A. Oliehoek, Matthijs T. J. Spaan, Christopher Amato, and Shimon Whiteson. "Incremental clustering and expansion for faster optimal planning in Dec-POMDPs." *Journal of Artificial Intelligence Research* 46 (2013), pp. 449–509.

[124] Frans A. Oliehoek and Christopher Amato. *A concise introduction to decentralized POMDPs.* Berlin, Germany, 2016.

[125] Frans A. Oliehoek, Matthijs T. J. Spaan, Bas Terwijn, Erwin Walraven, Joao V. de Sousa Messias, Philipp Robbel, Abdeslam Boularias, Xuanjie Liu, Julian Kooij, Tiago Veiga, Francisco Melo, Timon Kanters, and Philipp Beau. *The MADP Toolbox*. 2017.

[126] Shayegan Omidshafiei, Ali-akbar Agha mohammadi, Christopher Amato, and Jonathan P. How. "Decentralized control of Partially Observable Markov Decision Processes using belief space macro-actions." *IEEE International Conference on Robotics and Automation*. 2015, pp. 5962–5969.

[127] Shayegan Omidshafiei, Ali-akbar Agha mohammadi, Christopher Amato, Shih-Yuan Liu, Jonathan P. How, and John Vian. "Graph-based Cross Entropy method for solving multi-robot decentralized POMDPs." *IEEE International Conference on Robotics and Automation*. 2016, pp. 5395–5402.

[128] Shayegan Omidshafiei, Ali-Akbar Agha-Mohammadi, Christopher Amato, Shih-Yuan Liu, Jonathan P. How, and John Vian. "Decentralized control of multi-robot partially observable Markov decision processes using belief space macro-actions." *International Journal of Robotics Research* 36.2 (2017), pp. 231–258.

[129] Joni K. Pajarinen and Jaakko Peltonen. "Periodic Finite State Controllers for Efficient POMDP and DEC-POMDP Planning." *Advances in Neural Information Processing Systems* 24 (2011), pp. 2636–2644.

[130] Joni K. Pajarinen and Jaakko Peltonen. "Efficient Planning for Factored Infinite-horizon DEC-POMDPs." *International Joint Conference on Artificial Intelligence*. 2011, pp. 325–331.

[131] Joni K. Pajarinen and Jaakko Peltonen. "Expectation Maximization for Average Reward Decentralized POMDPs." *Machine Learning and Knowledge Discovery in Databases*. Ed. by Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Zelezny. Vol. 8188. Lecture Notes in Computer Science. 2013, pp. 129–144.

[132] Edwin P. D. Pednault. "ADL: Exploring the Middle Ground Between STRIPS and the Situation Calculus." *International Conference on Principles of Knowledge Representation and Reasoning*. 1989, pp. 324–332.

[133] Scott Penberthy and Daniel S. Weld. "Temporal planning with continuous change." *National Conference on Artificial Intelligence*. 1994, pp. 1010–1015.

[134] Tiago Pereira, Nerea Luis, Antonio Moreira, Daniel Borrajo, Manuela Veloso, and Susana Fernandez. "Heterogeneous multi-agent planning using actuation maps." *IEEE International Conference on Autonomous Robot Systems and Competitions*. 2018, pp. 219–224.

[135] Patrick Pfaff, Wolfram Burgard, and Dieter Fox. "Robust Monte-Carlo Localization Using Adaptive Likelihood Models." *European Robotics Symposium*. Ed. by Henrik I. Christensen. Vol. 22. Springer Tracts in Advanced Robotics. 2006, pp. 181–194.

[136] Martha E. Pollack, David Joslin, and Massimo Paolucci. "Flaw Selection Strategies For Partial-Order Planning." *Journal of Artificial Intelligence Research* 6 (1997), pp. 223–262.

[137] Pascal Poupart. "Exploiting structure to efficiently solve large scale POMDPs." PhD dissertation. University of Toronto, 2005.

[138] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. "ROS: An open-source robot operating system." *Workshops at the IEEE International Conference on Robotics and Automation*. 2009.

[139] Talal Rahwan and Nicholas R. Jennings. "An improved dynamic programming algorithm for coalition structure generation." *International Conference on Autonomous Agents and Multiagent Systems*. 2008, pp. 1417–1420.

[140] Zhigang Ren, Zuren Feng, and Xiaonian Wang. "An efficient ant colony optimization approach to agent coalition formation problem." *World Congress on Intelligent Control and Automation*. 2008, pp. 7879–7882.

[141] Khashayar Rohanimanesh and Sridhar Mahadevan. "Decision-theoretic planning with concurrent temporally extended actions." *Conference on Uncertainty in Artificial Intelligence*. 2001, pp. 472–479.

[142] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (3rd Edition)*. Upper Saddle River, New Jersey, 2009.

[143] Scott Sanner and Craig Boutilier. "Practical solution techniques for first-order MDPs." *Artificial Intelligence* 173.5-6 (2009), pp. 748–788.

[144] Scott Sanner. *Relational dynamic influence diagram language: Language description*. Tech. rep. 2010, p. 32.

[145] Scott Sanner and Kristian Kersting. "Symbolic Dynamic Programming for First-order POMDPs." *AAAI Conference on Artificial Intelligence*. 2010, pp. 1140–1146.

[146] Scott Sanner, Karina V. Delgado, and Leliane N. de Barros. "Symbolic dynamic programming for discrete and continuous state MDPs." *Conference on Uncertainty in Artificial Intelligence*. 2011, pp. 643–652.

[147] Sanem Sariel-Talay, Tucker R. Balch, and Nadia Erdogan. "A Generic Framework for Distributed Multirobot Cooperation." *Journal of Intelligent & Robotic Systems* 63.2 (2011), pp. 323–358.

[148] Sarvapali D. Ramchurn, Maria Polukarov, Alessandro Farinelli, Cuong Truong, and Nicholas R. Jennings. "Coalition formation with spatial and temporal constraints." *International Conference on Autonomous Agents and Multiagent Systems*. 2010, pp. 1181–1188.

[149] Sayan D. Sen and Julie A. Adams. "A decision network based framework for multiagent coalition formation." *International Conference on Autonomous Agents and Multi-Agent Systems*. 2013, pp. 55–62.

[150] Sayan D. Sen and Julie A. Adams. "Real-time optimal selection of multirobot coalition formation algorithms using conceptual clustering." *AAAI Conference on Artificial Intelligence*. 2015, pp. 29–35.

[151] Sayan D. Sen and Julie A. Adams. "An influence diagram based multi-criteria decision making framework for multirobot coalition formation." *Autonomous Agents and Multi-Agent Systems* 29.6 (2015), pp. 1061–1090.

[152] Sayan D. Sen. "An Intelligent and Unified Framework for Multiple Robot and Human Coalition Formation." PhD dissertation. Vanderbilt University, 2015.

[153] Travis C. Service and Julie A. Adams. *Theory and algorithms for coalition formation among heterogeneous agents*. Tech. rep. 2009.

[154] Travis C. Service and Julie A. Adams. "Coalition formation for task allocation: Theory and algorithms." *Autonomous Agents and Multi-Agent Systems* 22.2 (2011), pp. 225–248.

[155] Travis C. Service and Julie A. Adams. "Constant factor approximation algorithms for coalition structure generation." *Autonomous Agents and Multi-Agent Systems* 23.1 (2011), pp. 1–17.

[156] Travis C. Service and Julie A. Adams. "A simultaneous descending auction for task allocation." *IEEE International Conference on Systems, Man and Cybernetics*. 2014, pp. 379–384.

[157] Sven Seuken and Shlomo Zilberstein. "Formal models and algorithms for decentralized decision making under uncertainty." *Autonomous Agents and Multi-Agent Systems* 17.2 (2008), pp. 190–250.

[158] Mohsen Shahandasht, Binaya Pudasaini, and Sean Logan McCauley. *Autonomous Vehicles and Freight Transportation Analysis*. Tech. rep. 2019.

[159] Onn Shehory and Sarit Kraus. "Methods for task allocation via agent coalition formation." *Artificial Intelligence* 101.1-2 (1998), pp. 165–200.

[160] Pedro M. Shiroma and Mario F. M. Campos. "CoMutaR: A framework for multi-robot coordination and task allocation." *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2009, pp. 4817–4824.

[161] Steven S. Skiena. *The Algorithm Design Manual*. 2nd ed. London, United Kingdom, 2008.

[162] David E. Smith and Daniel S. Weld. "Temporal planning with mutual exclusion reasoning." *International Journal of Computer Applications*. 1999, pp. 326–337.

[163] David E. Smith and William Cushing. "The ANML language." *The International Conference on Automated Planning and Scheduling Workshop on Knowledge Engineering for Planning and Scheduling*. 2008.

[164] Adhiraj Somani, Nan Ye, David Hsu, and Wee S. Lee. "DESPOT: Online POMDP planning with regularization." *Advances in Neural Information Processing Systems* 26 (2013), pp. 1–9.

[165] Matthijs T. J. Spaan, Frans A. Oliehoek, and Christopher Amato. "Scaling up optimal heuristic search in Dec-POMDPs via incremental expansion." *International Joint Conference on Artificial Intelligence*. Vol. 22. 3. 2011, pp. 2027–2032.

[166] Sarath Sreedharan, Yu Zhang, and Subbarao Kambhampati. "A first multi-agent planner for required cooperation." *Competition of Distributed and Multi-Agent Planners*. 2015, pp. 17–20.

[167] Biplav Srivastava, Subbarao Kambhampati, Tuan A. Nguyen, Minh B. Do, Alfonso Gerevini, and Ivan Serina. "Domain Independent Approaches for Finding Diverse Plans." *International Joint Conference on Artificial Intelligence*. 2007, pp. 2016–2022.

[168] Marcel Steinmetz, Jorg Hoffmann, and Olivier Buffet. "Goal Probability Analysis in Probabilistic Planning: Exploring and Enhancing the State of the Art." *Journal of Artificial Intelligence Research* 57 (2016), pp. 229–271.

[169] P. B. Sujit, Joel M. George, and Randal W. Beard. "Multiple UAV coalition formation." *American Control Conference*. 2008, pp. 2010–2015.

[170] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. Cambridge, Massachusetts, 1998.

[171] Richard S. Sutton, Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning." *Artificial Intelligence* 112.1-2 (1999), pp. 181–211.

[172] Sven Seuken and Shlomo Zilberstein. "Memory-Bounded Dynamic Programming for DEC-POMDPs." *International Joint Conference on Artificial Intelligence*. 2007, pp. 2009–2015.

[173] Sven Seuken and Shlomo Zilberstein. "Improved Memory-Bounded Dynamic Programming for Decentralized POMDPs." *Conference on Uncertainty in Artificial Intelligence*. 2007, pp. 344–351.

[174] Daniel Szer and Francois Charpillet. "An Optimal Best-First Search Algorithm for Solving Infinite Horizon DEC-POMDPs." *Machine Learning: European Conference on Machine Learning*. Ed. by Joao Gama, Rui Camacho, Pavel Brazdil, Alipio Mario Jorge, and Luis Torgo. Vol. 3720. Lecture Notes in Computer Science. Berlin, Heidelberg, 2005. Chap. 38, pp. 389–399.

[175] Daniel Szer, Francois Charpillet, and Shlomo Zilberstein. "MAA*: A Heuristic Search Algorithm for Solving Decentralized POMDPs." *Conference on Uncertainty in Artificial Intelligence*. 2005, pp. 576–583.

[176] Daniel Szer and Francois Charpillet. "Point-Based Dynamic Programming for DEC-POMDPs." *National Conference on Artificial Intelligence*. 2006, pp. 1233–1238.

[177] Paul Taylor, Alan W. Black, and Richard Caley. "The architecture of the Festival speech synthesis system." *The International Speech Communication Association Speech Synthesis Workshop*. 1998, pp. 147–152.

[178] Alejandro Torreno, Eva Onaindia, and Oscar Sapena. "FMAP: Distributed cooperative multi-agent planning." *Applied Intelligence* 41.2 (2014), pp. 606–626.

[179] Alejandro Torreno, Eva Onaindia, Antonin Komenda, and Michal Stolba. "Cooperative multi-agent planning." *ACM Computing Surveys* 50.6 (2017), pp. 1–32. eprint: `1711.09057`.

[180] Predrag T. Tosic and Gul A. Agha. "Maximal Clique Based Distributed Coalition Formation for Task Allocation in Large-Scale Multi-agent Systems." *Massively Multi-Agent Systems I*. Ed. by Toru Ishida, Les Gasser, and Hideyuki Nakashima. Vol. 3446. Lecture Notes in Computer Science. 2005, pp. 104–120.

[181] Mauro Vallati, Luk Chrpa, Marek Grzes, T. L. McCluskey, Mark Roberts, and Scott Sanner. "The 2014 International Planning Competition: Progress and trends." *AI Magazine* 36.3 (2015), pp. 90–98.

[182] Pradeep Varakantham, Janusz Marecki, Yuichi Yabu, Milind Tambe, and Makoto Yokoo. "Letting Loose a SPIDER on a Network of POMDPs: Generating Quality Guaranteed Policies." *International Conference on Autonomous Agents and Multiagent Systems*. 2007.

[183] Thierry Vidal and Malik Ghallab. "Dealing with Uncertain Durations In Temporal Constraint Networks dedicated to Planning." *European Conference on Artificial Intelligence*. 1996, pp. 48–54.

[184] Vincent Vidal. "YAHSP3 and YAHSP3-MT in the International Planning Competition." *The International Planning Competition*. 2014, pp. 64–65.

[185] Lovekesh Vig and Julie A. Adams. "Issues in multi-robot coalition formation." *International Workshop on Multi-Robot Systems*. Ed. by Lynne E. Parker, Frank E. Schneider, and Alan C. Schultz. 2005, pp. 15–26.

[186] Lovekesh Vig and Julie A. Adams. "Multi-robot coalition formation." *IEEE Transactions on Robotics* 22.4 (2006), pp. 637–649.

[187] Lovekesh Vig and Julie A. Adams. "Market-based multi-robot coalition formation." *Distributed Autonomous Robotic Systems 7*. Ed. by Maria Gini and Richard Voyles. 2006, pp. 227–236.

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

[189] Ronald E. Walpole, Raymond H. Myers, Sharon L. Myers, and Keying Ye. *Probability & Statistics for Engineers and Scientists*. 8th ed. Upper Saddle River, New Jersey, 2007.

[190] Mathijs de Weerdt, Yingqian Zhang, and Tomas Klos. "Distributed task allocation in social networks." *International Conference on Autonomous Agents and Multiagent Systems*. 2007, pp. 488–495.

[191] Mathijs de Weerdt and Brad Clement. "Introduction to planning in multiagent systems." *Multiagent and Grid Systems* 5.4 (2009), pp. 345–355.

[192] David E. Wilkins and Karen L. Myers. "A multiagent planning architecture." *International Conference on Artificial Intelligence Planning Systems*. 1998, pp. 154–162.

[193] Patrick H. Winston. *Artificial Intelligence*. Boston, Massachusetts, 1995.

[194] Feng Wu, Shlomo Zilberstein, and Xiaoping Chen. "Point-Based Policy Generation for Decentralized POMDPs." *International Conference on Autonomous Agents and Multiagent Systems*. 2010, pp. 1307–1314.

[195] Feng Wu, Shlomo Zilberstein, and Nicholas R. Jennings. "Monte-Carlo Expectation Maximization for Decentralized POMDPs." *International Joint Conference on Artificial Intelligence*. 2013, pp. 397–403.

[196] Jinyou Xu and Wenli Li. "Solution of Overlapping Coalition Formation Based on Discrete Particle Swarm Optimization." *International Conference on Wireless Communications, Networking and Mobile Computing*. 2008, pp. 1–4.

[197] Sungwook Yoon, Alan Fern, Robert Givan, and Subbarao Kambhampati. "Probabilistic planning via determinization in hindsight." *AAAI Conference on Artificial Intelligence*. 2008, pp. 1010–1016.

[198] Sungwook Yoon, Wheeler Ruml, J. Benton, and Minh B. Do. "Improving determinization in hindsight for online probabilistic planning." *International Conference on Automated Planning and Scheduling*. 2010, pp. 209–216.

[199] Hakan L. S. Younes and Michael L. Littman. *PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects*. Tech. rep. 2004.

[200] Hakan L. S. Younes and Reid G. Simmons. "Policy generation for continuous-time stochastic domains with concurrency." *International Conference on Automated Planning and Scheduling*. 2004, pp. 325–333.

[201] Hakan L. S. Younes and Reid G. Simmons. "Solving generalized semi-Markov decision processes using continuous phase-type distributions." *International Conference on Artifical Intelligence*. 2004, pp. 742–747.

[202] Zahra Zamani, Scott Sanner, and Cheng Fang. "Symbolic Dynamic Programming for Continuous State and Action MDPs." *AAAI Conference on Artificial Intelligence*. 2012, pp. 1839–1845.

[203] Yu Zhang and Lynne E. Parker. "IQ-ASyMTRe: Synthesizing coalition formation and execution for tightly-coupled multirobot tasks." *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010, pp. 5595–5602.

[204] Yu Zhang, Sarath Sreedharan, and Subbarao Kambhampati. "Capability models and their applications in planning." *International Conference on Autonomous Agents and Multiagent Systems*. 2015, pp. 1151–1159.

[205] Eckart Zitzler and Lothar Thiele. "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach." *IEEE Transactions on Evolutionary Computation* 3.4 (1999), pp. 257–271.

APPENDICES

Appendix A: The MAPL Language

The MAPL language grammar is presented in Listing A.1.

Listing A.1: The MAPL Grammar.

```
// Token Regex
number          : /[0-9]+(\.[0-9]+)?/
id              : /[a-zA-Z][a-zA-Z0-9_]*/
when            : /@(start|end|overall)/
comparison      : /(<=|>=|<|>|==)/
num_effect      : /(increase|decrease|=)/
COMMENT         : '//' /(.)+\n/

// Starting node
?start          : main

main            : section+
section         : 'domain' '{' types worlddef statedef actiondefs '}'
                          -> domain
                | 'Problem' '{' agents objects agent_cap worldstate initialstate
                    goalstate task_cap '}'  -> problem
                | 'Plan' '{' plan_entries* '}'
                                          -> plan


// ============================================================================
// Domain
// ============================================================================


// Main sections
types           : 'types' curl_list
worlddef        : 'world' relationsdefs
statedef        : 'state' relationsdefs
relationsdefs   : '{' relationsdef* '}'
actiondefs      : actiondef*

// Relation types
relationsdef    : 'predicate'   id      paren_list ';'  -> predicatedef
                | 'numeric'     id      paren_list ';'  -> numericdef
                | 'function'    id id   paren_list ';'  -> functiondef

actiondef       : 'action'      id dual_paren_list '{' duration? cost? conditions
    effects '}'

// Lists
id_list         : ( id      ( ',' id    )* )?
dual_id_list    : ( id id   ( ',' id id )* )?
paren_list      : '(' id_list ')'
curl_list       : '{' id_list '}'
```

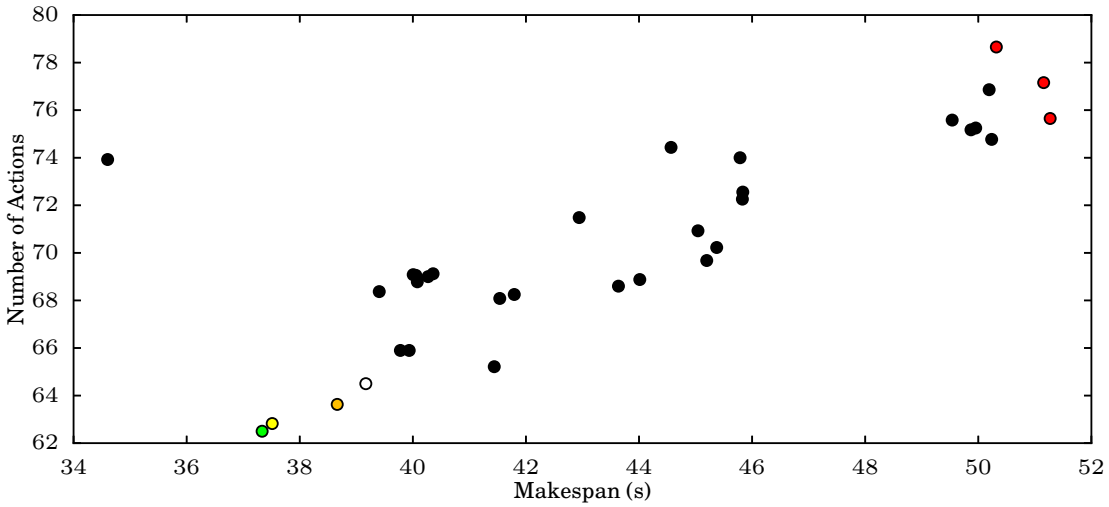
```
42 dual_paren_list : '(' dual_id_list ')'
43
44 // Action body
45 duration        : 'duration'    ':' distribution ';'
46 cost            : 'cost'        ':' distribution ';'
47 conditions      : 'conditions'  ':' condition*
48 effects         : 'effects'     ':' effect*
49
50
51 // Probability distributions
52 distribution    : 'constant'    exp     -> distribution_constant
53                 | 'uniform'     exp exp -> distribution_uniform
54                 | 'normal'      exp exp -> distribution_normal
55                 | 'exponential' exp     -> distribution_exponential
56
57 // Numerical expressions
58 exp             : id paren_list         -> exp_numeric
59                 | exp '*' exp           -> mul
60                 | exp '/' exp           -> div
61                 | exp '+' exp           -> add
62                 | exp '-' exp           -> sub
63                 | '(' exp ')'           -> list_strip
64                 | number                -> list_strip
65
66 // Temporal conditions
67 condition       : when ':' ( pred_cond | func_cond | num_cond | varsdiff ) ';'
68 effect          : when ':' ( pred_cond | func_cond | numeric_effects )';'
69
70 // Conditions
71 pred_cond       :     id paren_list -> pred_cond_pos
72                 | '!' id paren_list -> pred_cond_neg
73
74 func_cond       :     id paren_list '=' ( id | 'undefined' )    -> func_cond_pos
75                 | '!' id paren_list '=' ( id | 'undefined' )    -> func_cond_neg
76
77 num_cond        : id paren_list comparison exp
78
79 varsdiff        : id '!=' id
80
81 numeric_effects : id paren_list num_effect exp
82
83
84
85
86
87 // ============================================================
88 // Problem
89 // ============================================================
90
91
92 // Object list
93 objects         : 'Objects' '{' type_list* '}'
94 agents          : 'Agents' '{' type_list* '}'
95 type_list       : id id_list ';'
96
97 worldstate      : 'WorldState'      relations
98 initialstate    : 'InitialState'    relations
```

```
 99 goalstate       : 'Goal'            '{' task+ '}'
100 task            : id                relations ';'
101 relations       : '{' relation* '}'
102
103 relation        : id paren_list ';'                                  -> predicates
104                 | id paren_list '=' number ';'                       -> numerics
105                 | id paren_list ('='|'==') ('undefined' | id) ';'    -> functions
106
107 agent_cap       : 'AgentCapabilities' '{' group_cap* '}'
108 task_cap        : 'TaskCapabilities' '{' group_cap* '}'
109 group_cap       : '(' id_list ')' '{' capability_dict '}' ';'
110
111
112 capability_dict : capability_inst*
113 capability_inst : id ':' number ','
114
115
116
117 // ============================================================================
118 // Plan
119 // ============================================================================
120
121
122 plan_entries    : id ':' plan_action ';'    -> plan_actions
123                 | id '<' id ';'             -> ordering
124
125 plan_action     : id paren_list
```

Appendix B: Extended Task Fusion Results

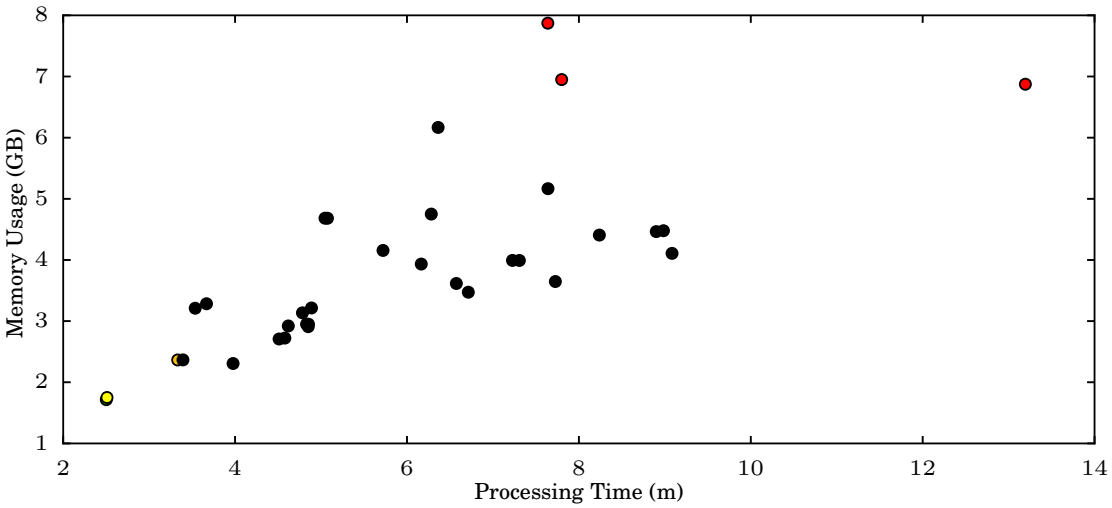
This Appendix provides the extended Pareto Strength Task Fusion results for the Blocks World and First Response Domains.

B.1 The Blocks World Domain with TFD



	Tool	f_{max}	Quality Pareto Strength
Best	**AO**	1.00	32
Second Best	**AO**	0.75	31
Third Best	**AOT**	1.00	30
Fourth Best	**AOT**	0.75	29
Worst	CS	0.50	0
Worst	**A**	0.25	0
Worst	**O**	0.25	0

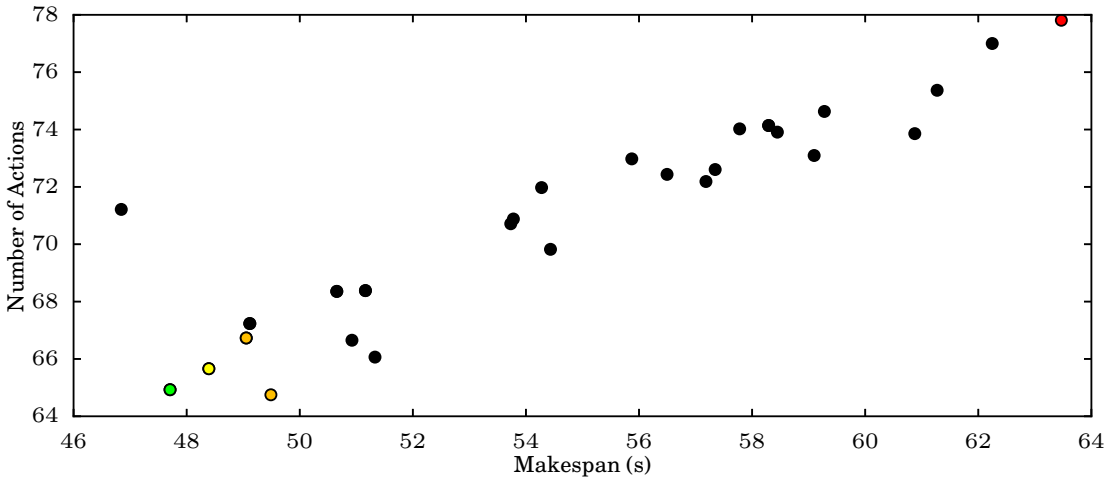
Figure B.1: Quality Pareto Strength for Blocks World with TFD.



	Tool	f_{max}	Cost Pareto Strength
Best	**AOT**	0.75	33
Second Best	**AOT**	1.00	32
Third Best	**OT**	0.75	30
Worst	CS	0.50	0
Worst	CFP	N/A	0
Worst	PA	N/A	0

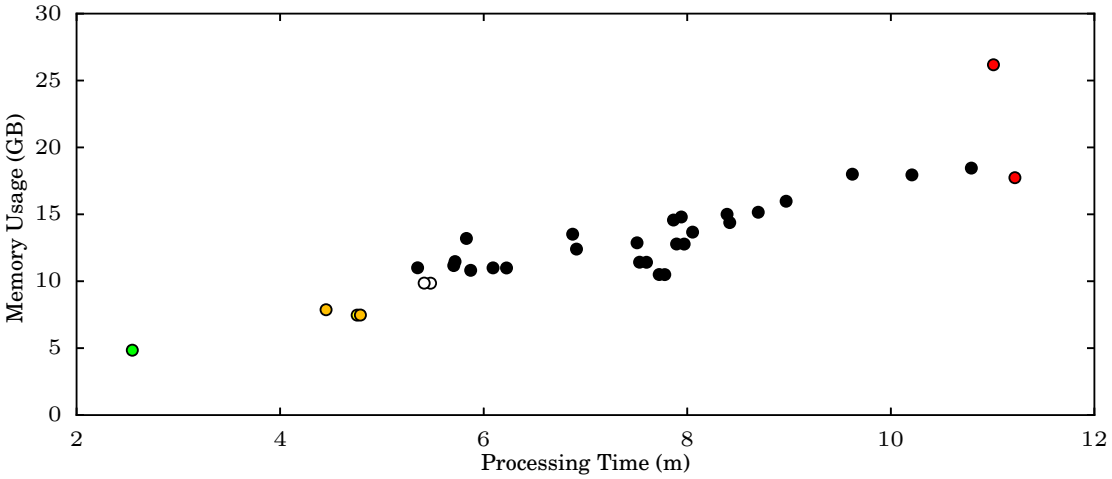
Figure B.2: Cost Pareto Strength for Blocks World with TFD.

B.2 The Blocks World Domain with COLIN



	Tool	f_{max}	Quality Pareto Strength
Best	**O**	0.75	30
Best	**O**	1.00	30
Second Best	**AO**	1.00	28
Second Best	**AO**	0.75	28
Third Best	**O**	0.50	24
Third Best	**OT**	1.00	24
Third Best	**OT**	0.75	24
Worst	CS	0.25	0

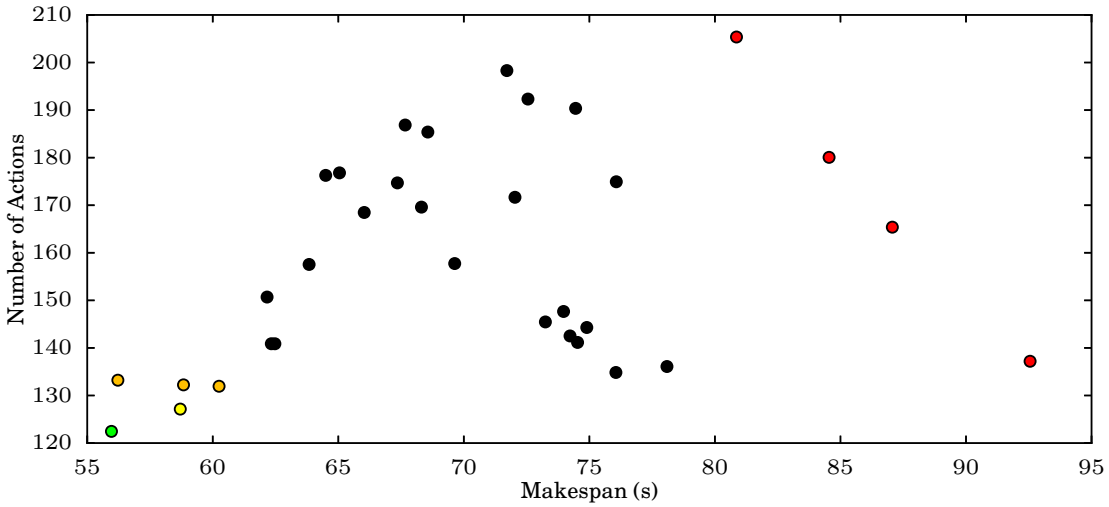
Figure B.3: Quality Pareto Strength for Blocks World with COLIN.



	Tool	f_{max}	Cost Pareto Strength
Best	CS	0.25	33
Second Best	CS	0.75	31
Third Best	CS	0.50	30
Third Best	CS	1.00	30
Fourth Best	**O**	0.75	27
Fourth Best	**O**	1.00	27
Worst	**A**	0.50	0
Worst	PA	N/A	0

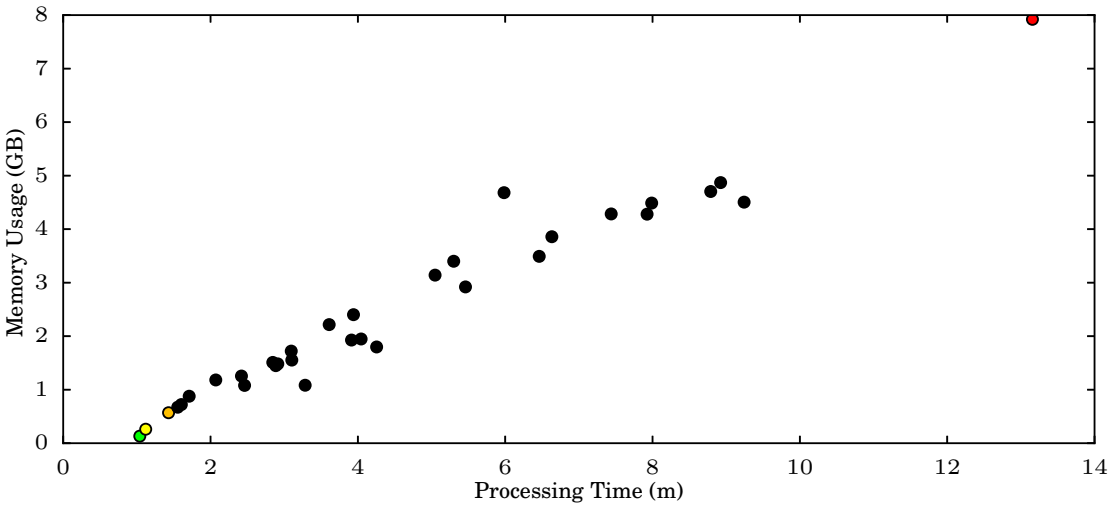
Figure B.4: Cost Pareto Strength for Blocks World with COLIN.

B.3 The First Response Domain



	Tool	f_{max}	Quality Pareto Strength
Best	**O**	0.75	32
Second Best	**OT**	0.75	30
Third Best	**AOT**	1.00	28
Third Best	**OT**	1.00	28
Third Best	**AO**	0.75	28
Worst	CS	1.00	0
Worst	CS	0.50	0
Worst	**AT**	0.75	0
Worst	CA	1.00	0

Figure B.5: Quality Pareto Strength for First Response.



	Tool	f_{max}	Cost Pareto Strength
Best	CFP	N/A	32
Second Best	CS	0.25	31
Third Best	**O**	0.75	30
Worst	**O**	1.00	0

Figure B.6: Cost Pareto Strength for First Response.

Appendix C: Extended Plan Merging Results

This appendix provides the complete set of success rate and makespan results for the plan merging and conflict resolution experiments, across the Logistics, Blocks World, and First Response Domains.

C.1 The Logistics Domain

All the Logistics Domain problems were successfully generated, for all tasks; however, the number of experimental samples for the aggregated makespan results diminished as the number of tasks increased, because of the diminishing plan merging success rates, as explained in Chapter 5.3. Therefore, the makespan results become noisier as the number of tasks grows, and are sometimes missing, due to the missing successful samples, for the highest number of tasks.

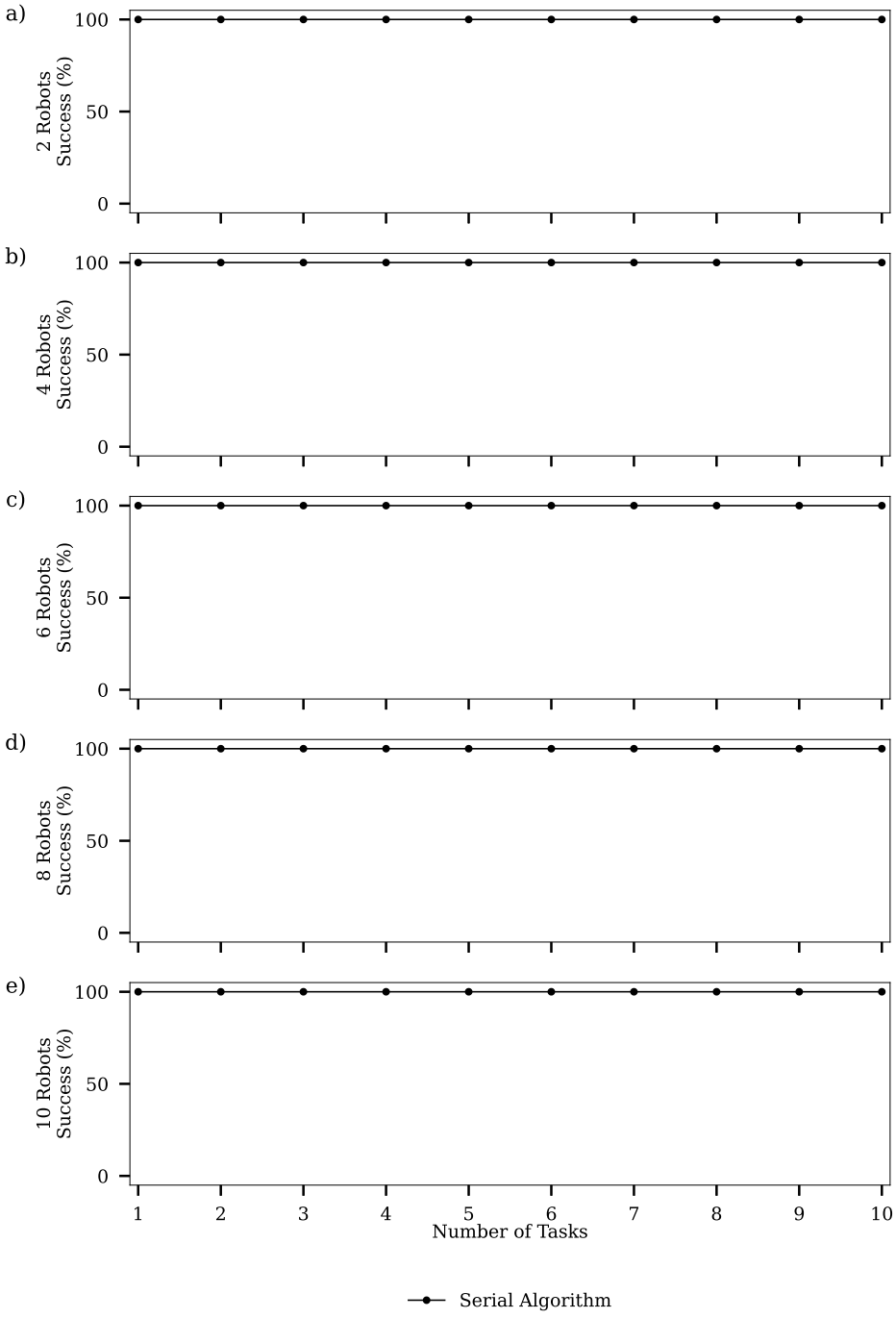


Figure C.1: Logistics Domain Serial success (%).

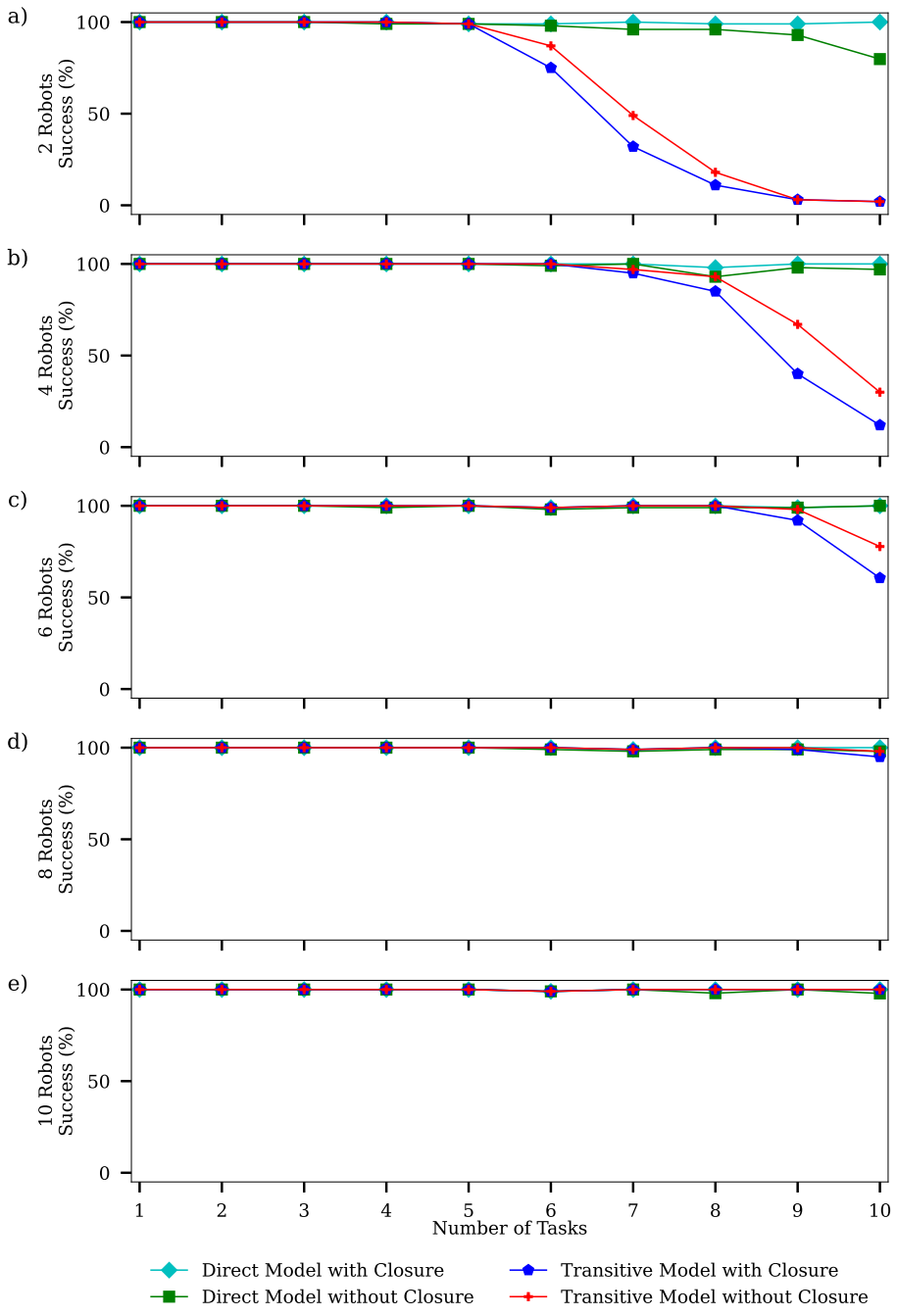


Figure C.2: Logistics Domain STA success (%).

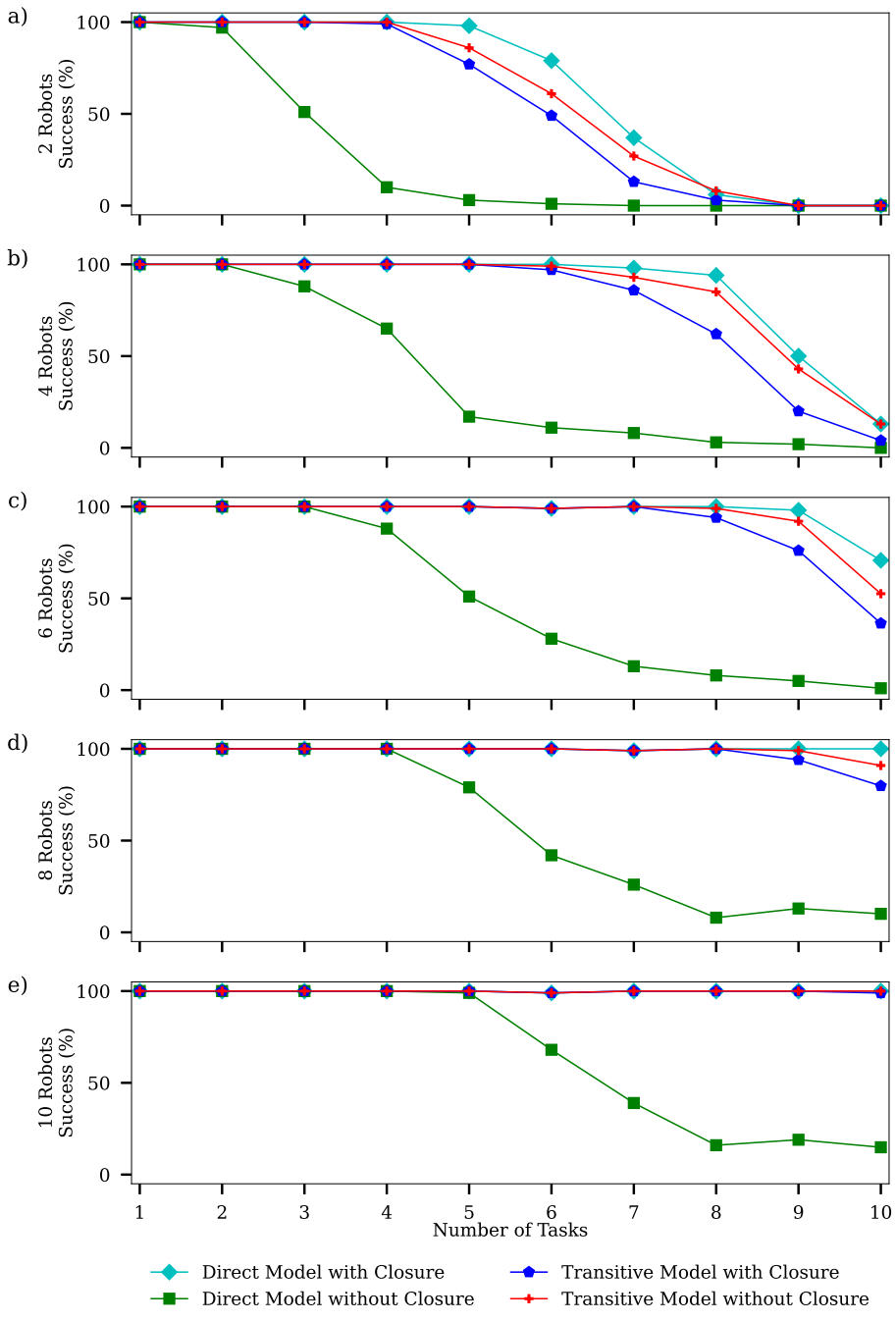


Figure C.3: Logistics Domain TCRA* ($\epsilon = 0$) success (%).

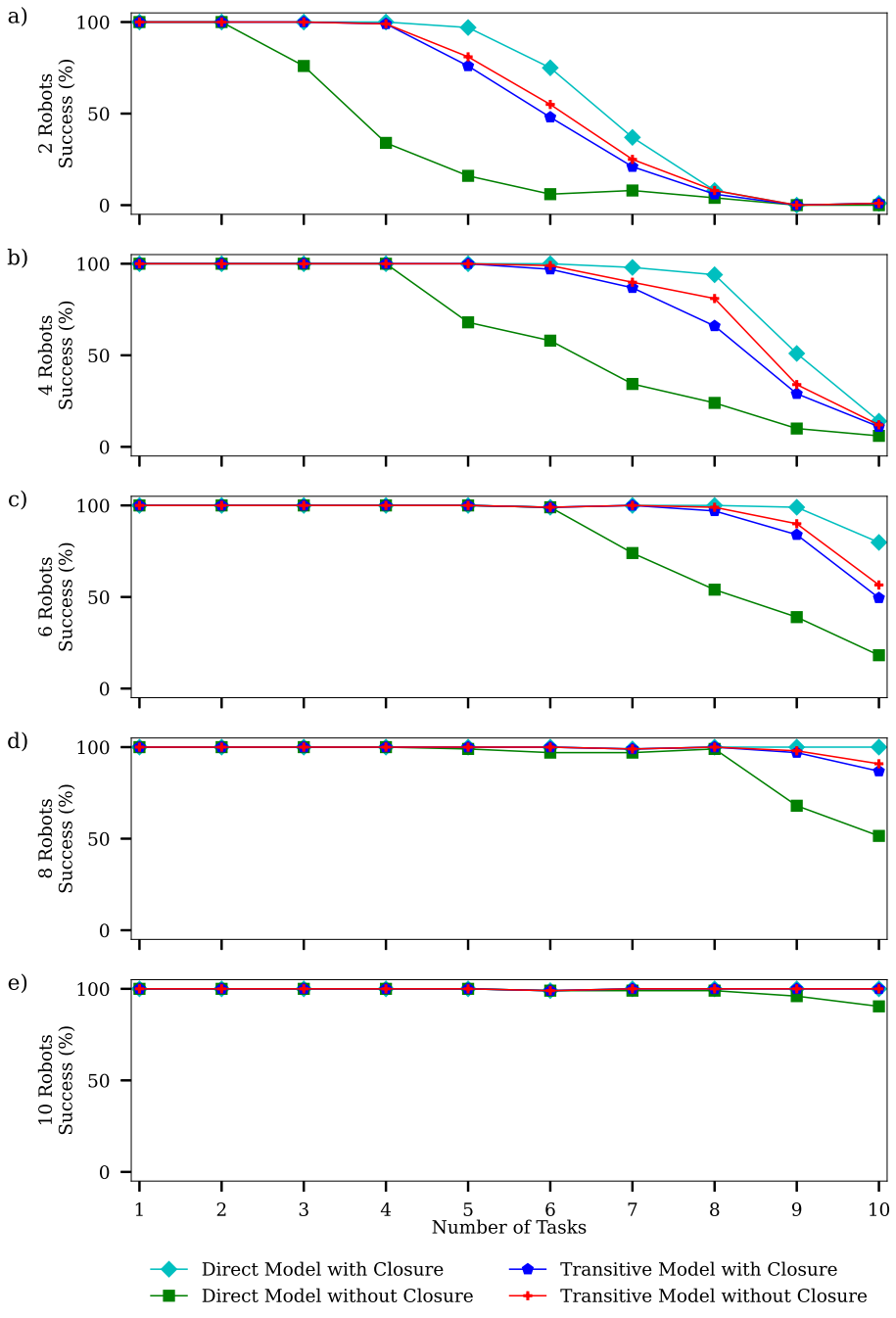


Figure C.4: Logistics Domain TCRA* ($\epsilon = 1$) success (%).

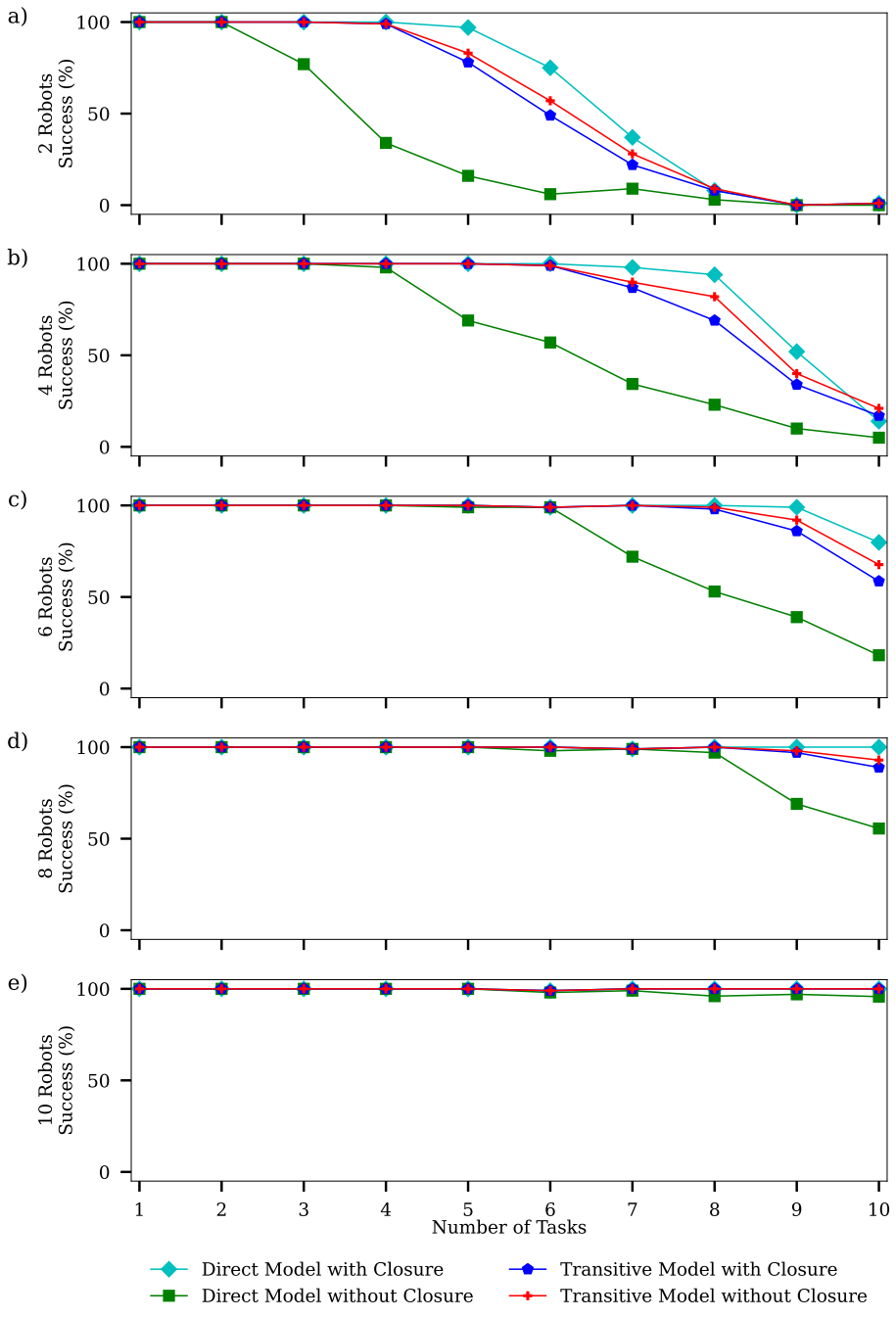


Figure C.5: Logistics Domain TCRA* ($\epsilon = 10$) success (%).

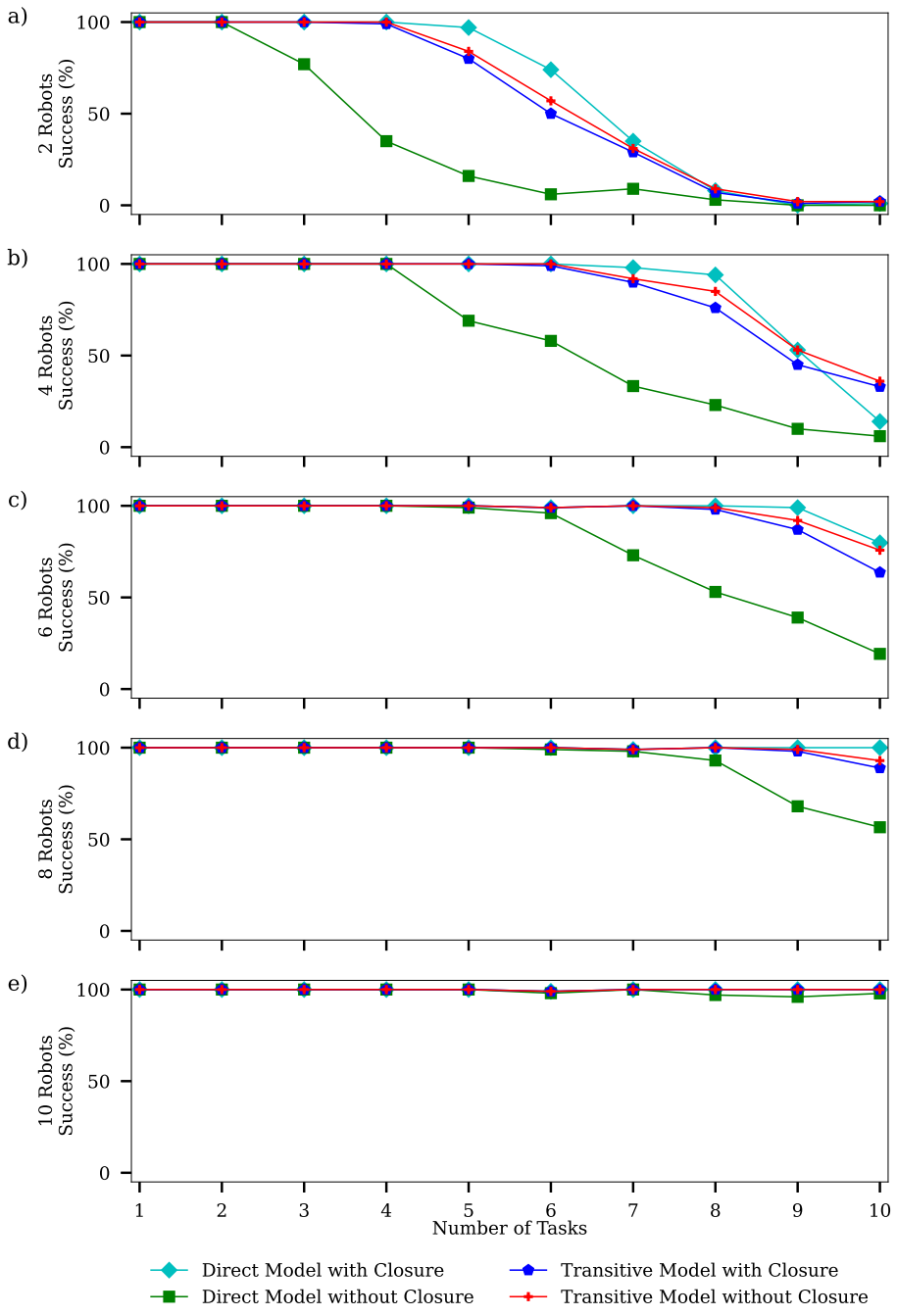


Figure C.6: Logistics Domain TCRA* ($\epsilon = 100$) success (%).

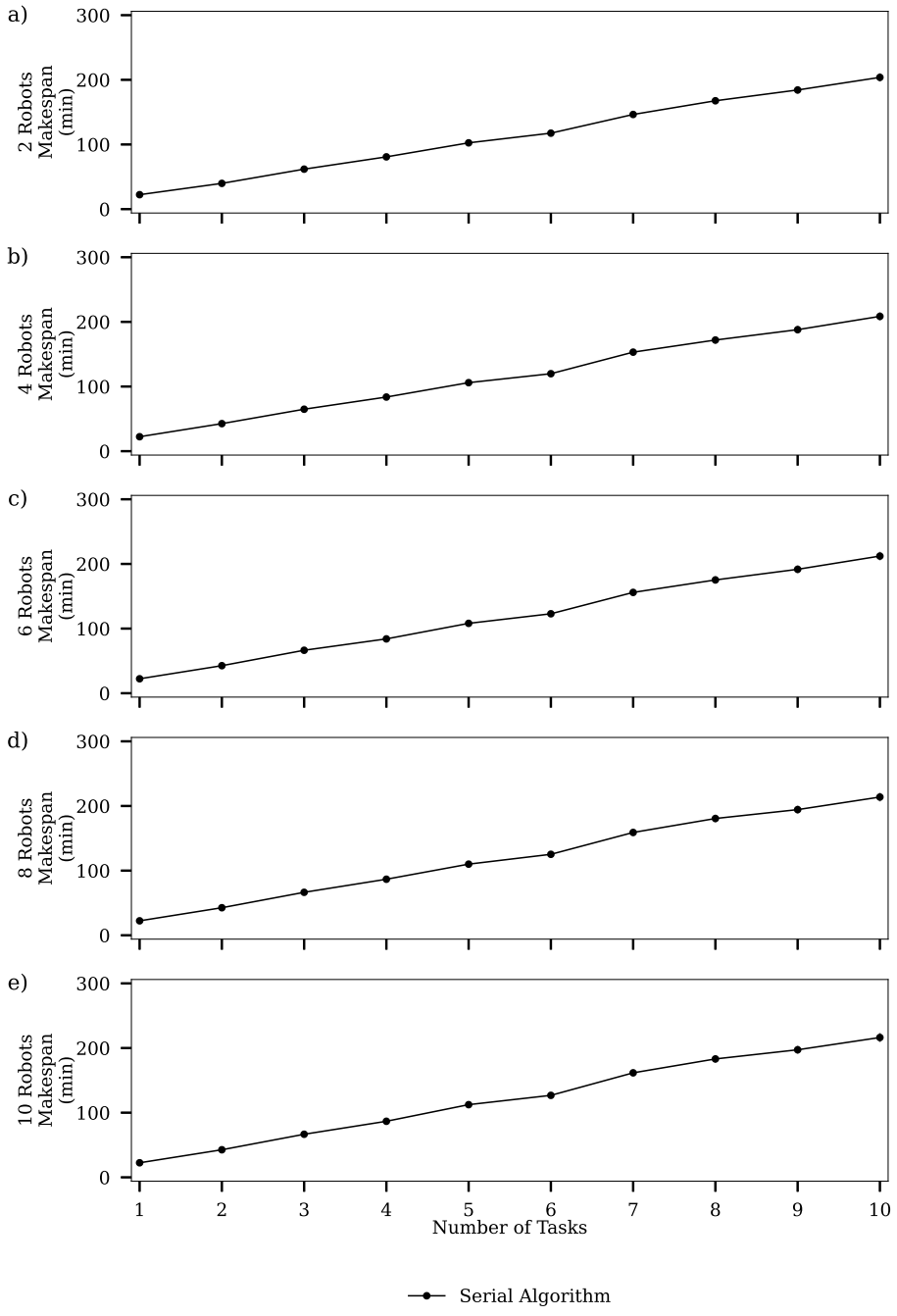


Figure C.7: Logistics Domain Serial makespan (min). The maximum makespan was 300 min for all numbers of robots.

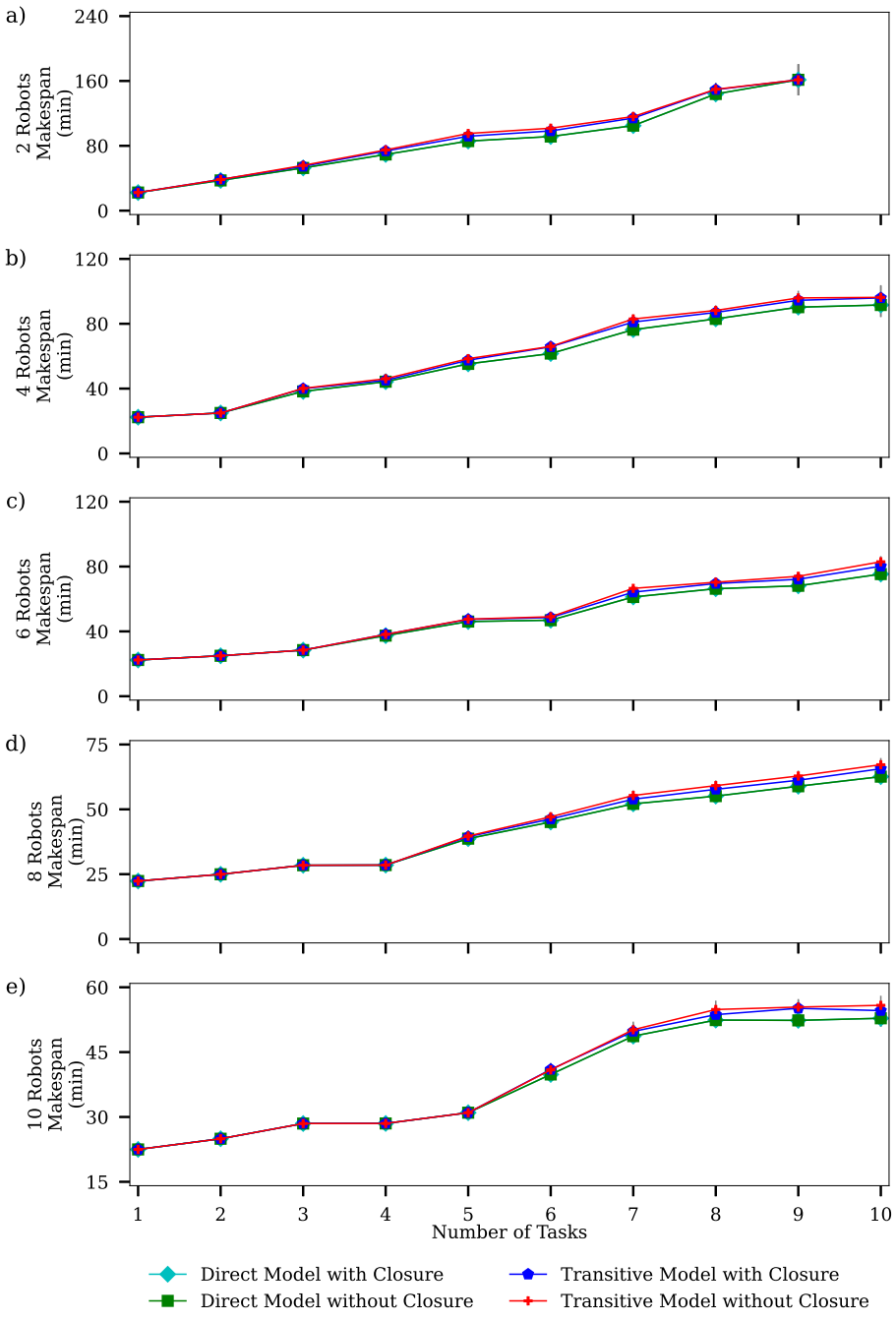


Figure C.8: Logistics Domain STA makespan (min). The maximum makespan was 240, 120, 120, 75, and 60 min for 2, 4, 6, 8, and 10 robots, respectivelly.

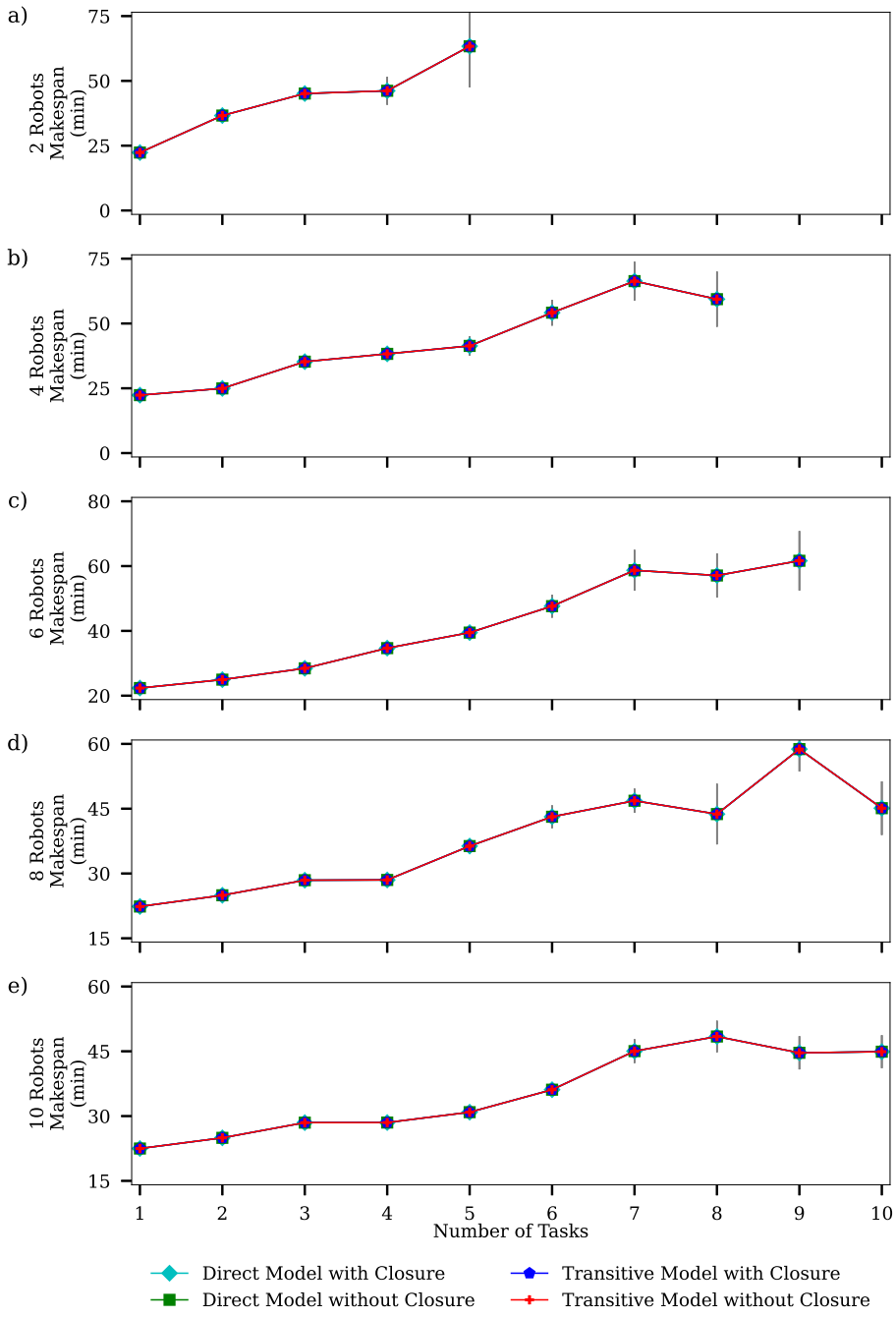


Figure C.9: Logistics Domain TCRA* ($\epsilon = 0$) makespan (min). The maximum makespan was 75, 75, 80, 60, and 60 min for 2, 4, 6, 8, and 10 robots, respectivelly.

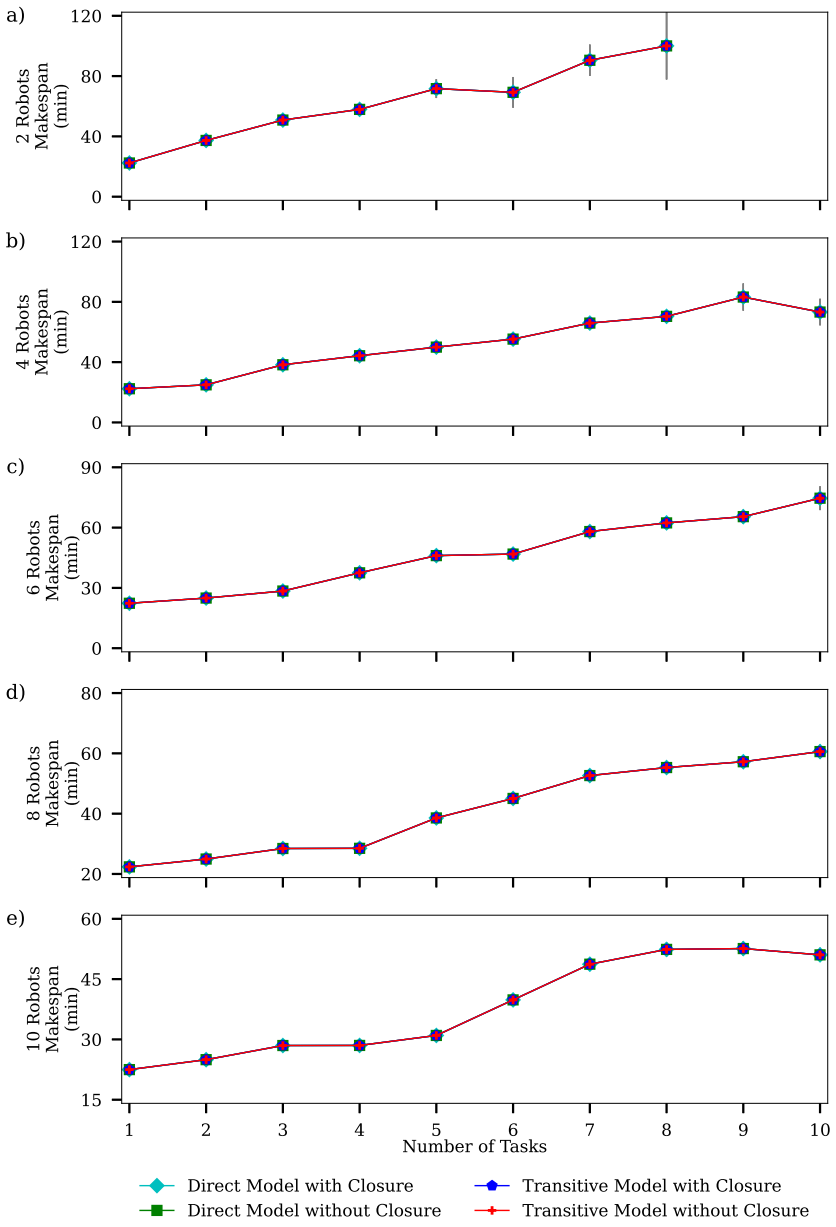


Figure C.10: Logistics Domain TCRA* ($\epsilon = 1$) makespan (min). The maximum makespan was 120, 120, 90, 80, and 60 min for 2, 4, 6, 8, and 10 robots, respectivelly.

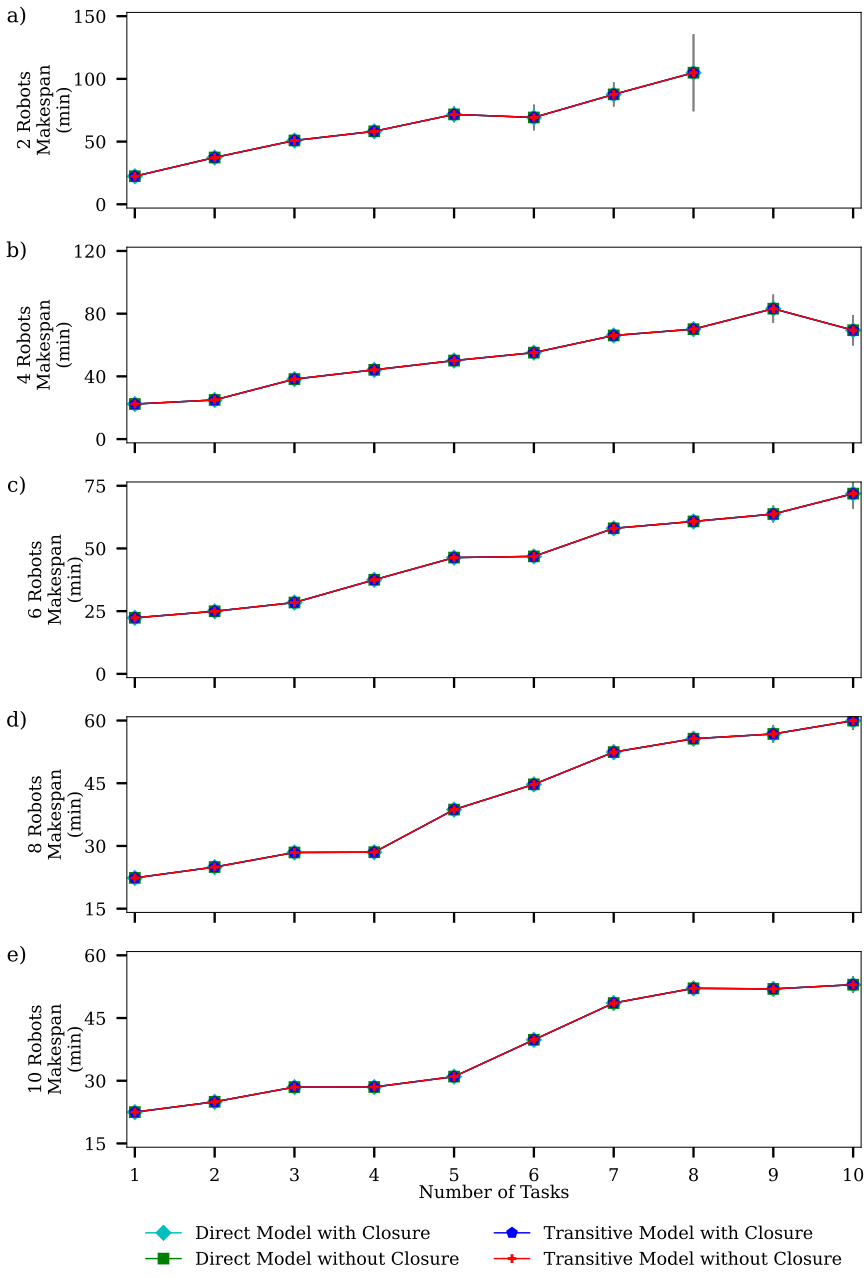


Figure C.11: Logistics Domain TCRA* ($\epsilon = 10$) makespan (min). The maximum makespan was 150, 120, 75, 60, and 60 min for 2, 4, 6, 8, and 10 robots, respectivelly.

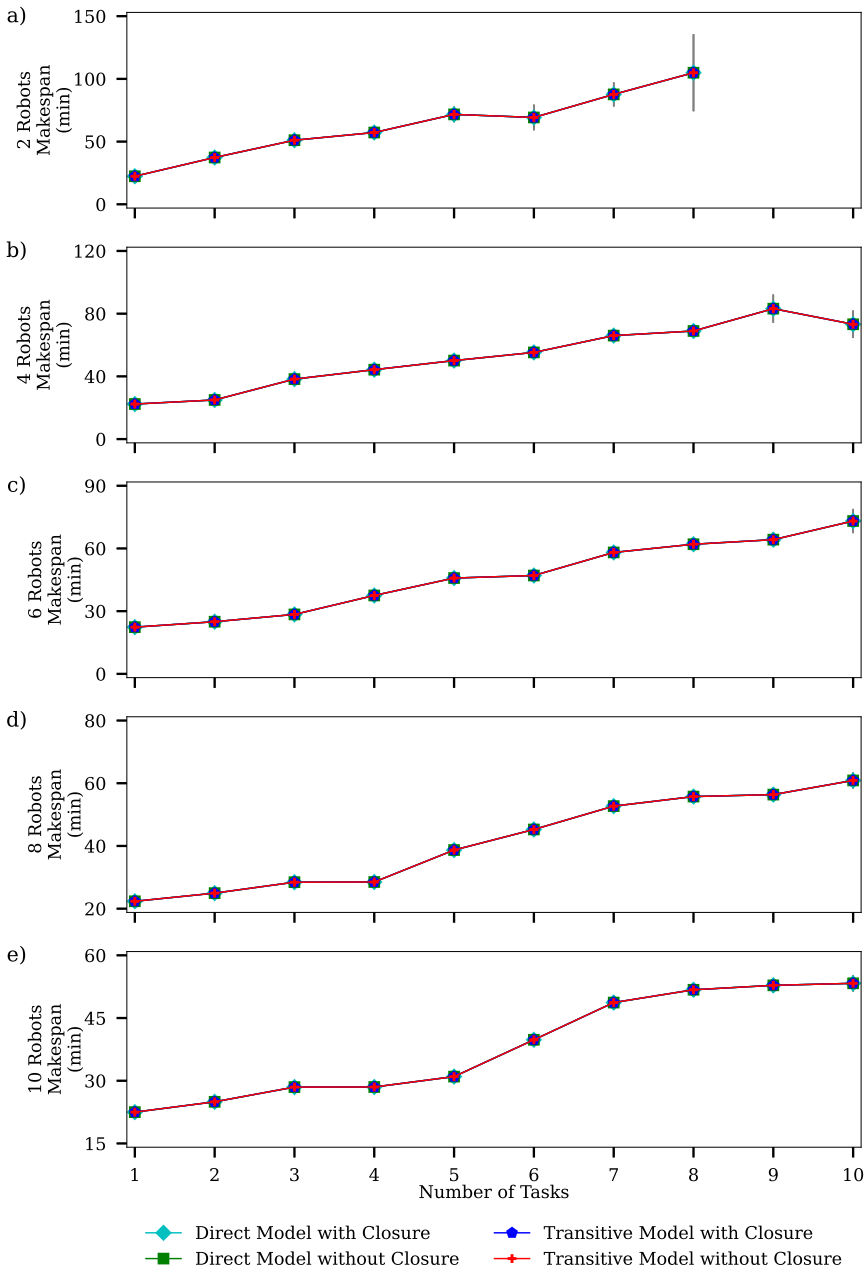


Figure C.12: Logistics Domain TCRA* ($\epsilon = 100$) makespan (min). The maximum makespan was 150, 120, 90, 80, and 60 min for 2, 4, 6, 8, and 10 robots, respectivelly.

C.2 The Blocksworld Domain

The number of experimental samples for the aggregated makespan results diminished as the number of tasks increased, because the ratio of Blocks World Domain problems successfully generated with 4-10 tasks was 99%, 82%, 67%, 43%, 15%, 8%, and 1%, respectively, as explained in Chapter 5.3. Therefore, the makespan results become noisier as the number of tasks grows, and are missing due to the missing successful samples, for the highest number of tasks.

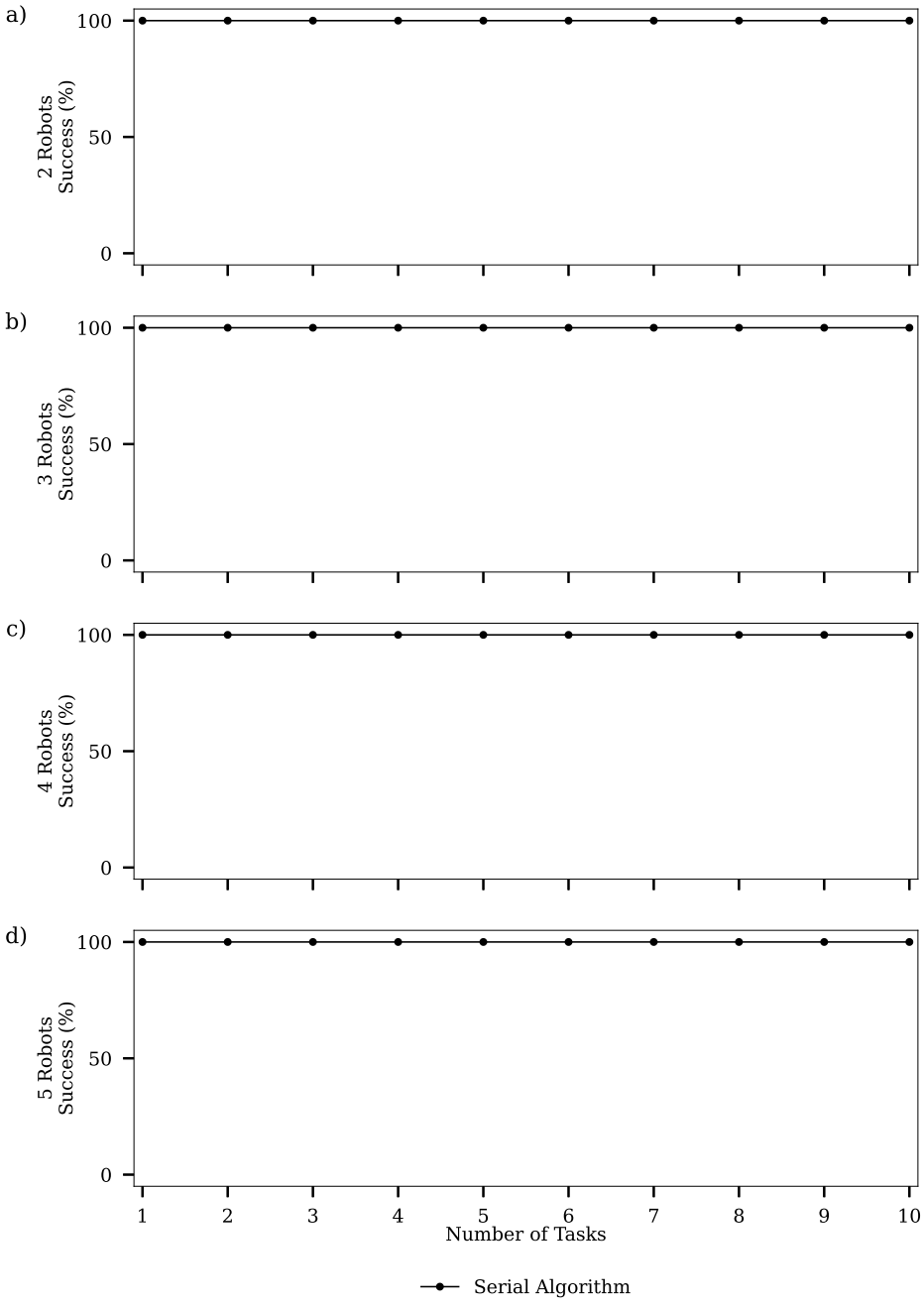


Figure C.13: Blocksworld Domain Serial success (%) for 1-5 robots.

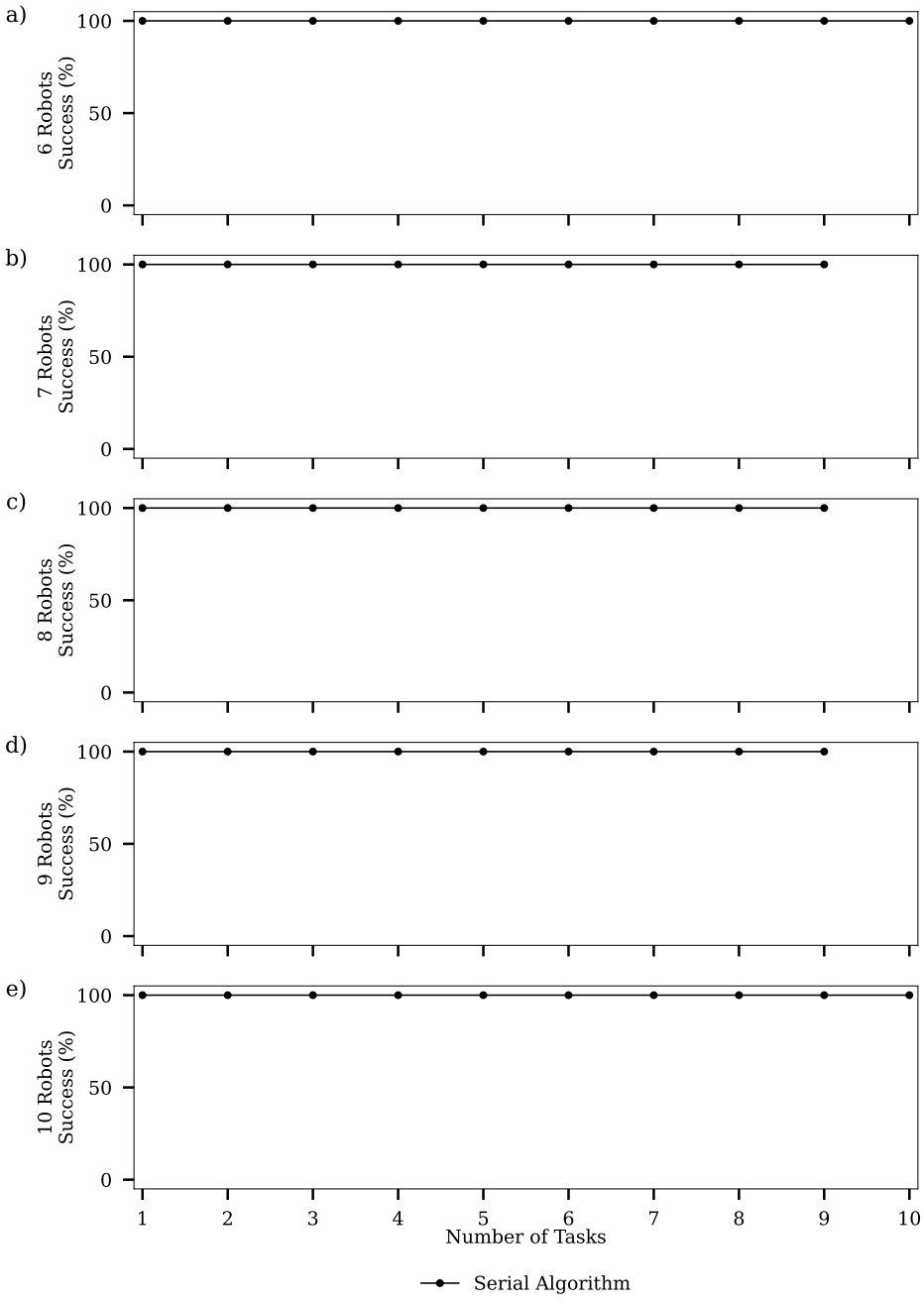


Figure C.14: Blocksworld Domain Serial success (%) for 6-10 robots.

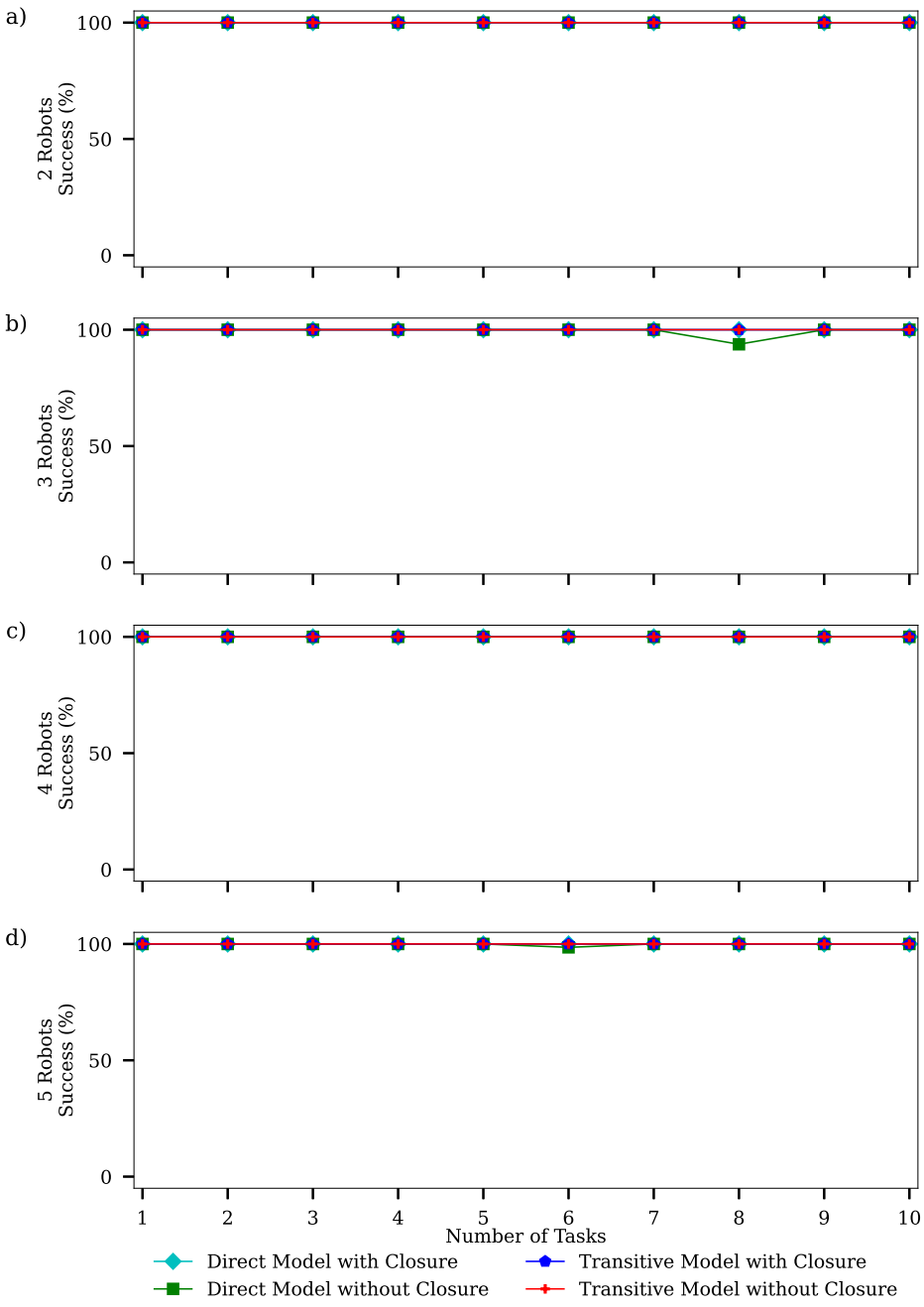


Figure C.15: Blocksworld Domain STA success (%) for 1-5 robots.

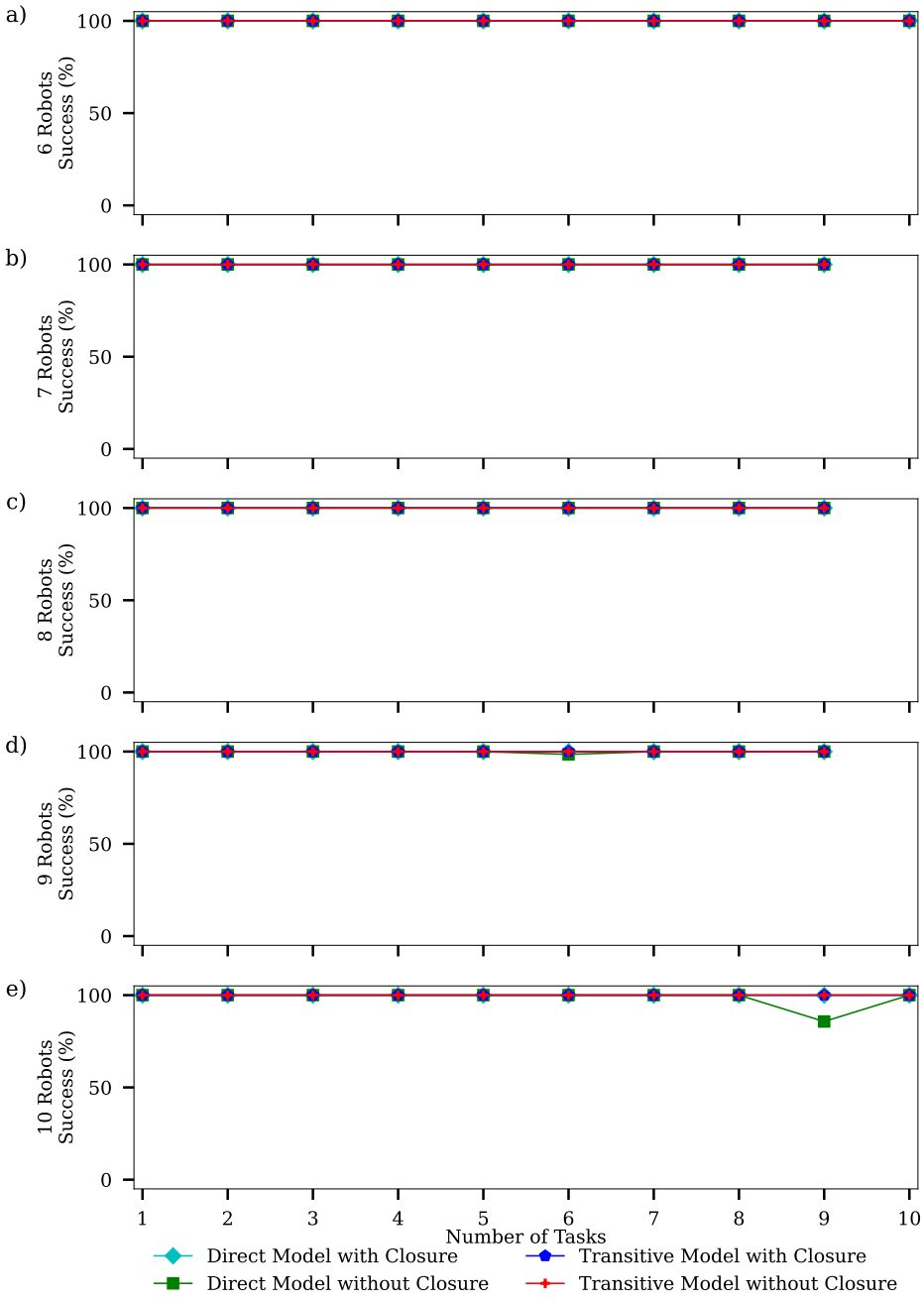


Figure C.16: Blocksworld Domain STA success (%) for 6-10 robots.

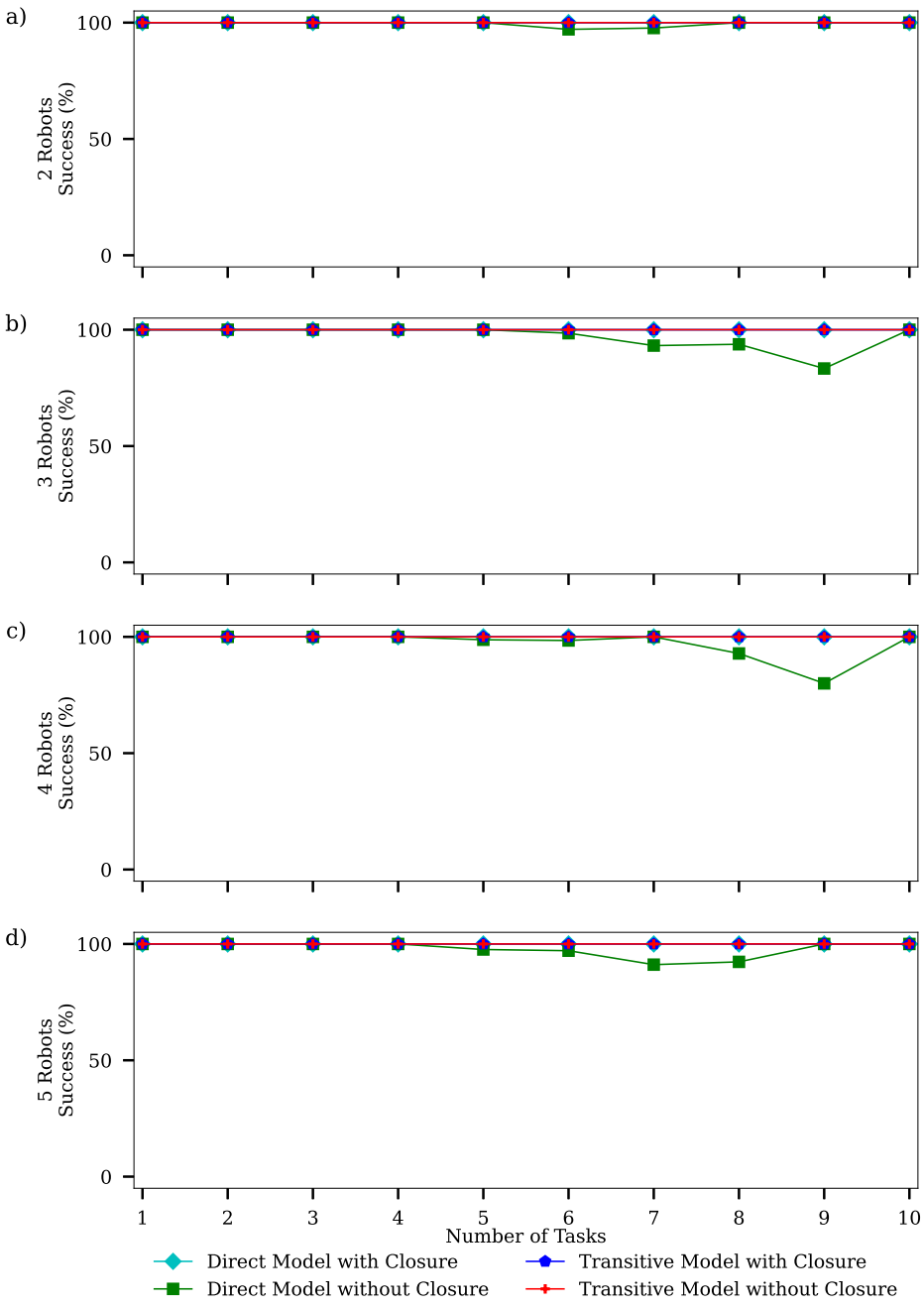


Figure C.17: Blocksworld Domain TCRA* ($\epsilon = 0$) success (%) for 1-5 robots.

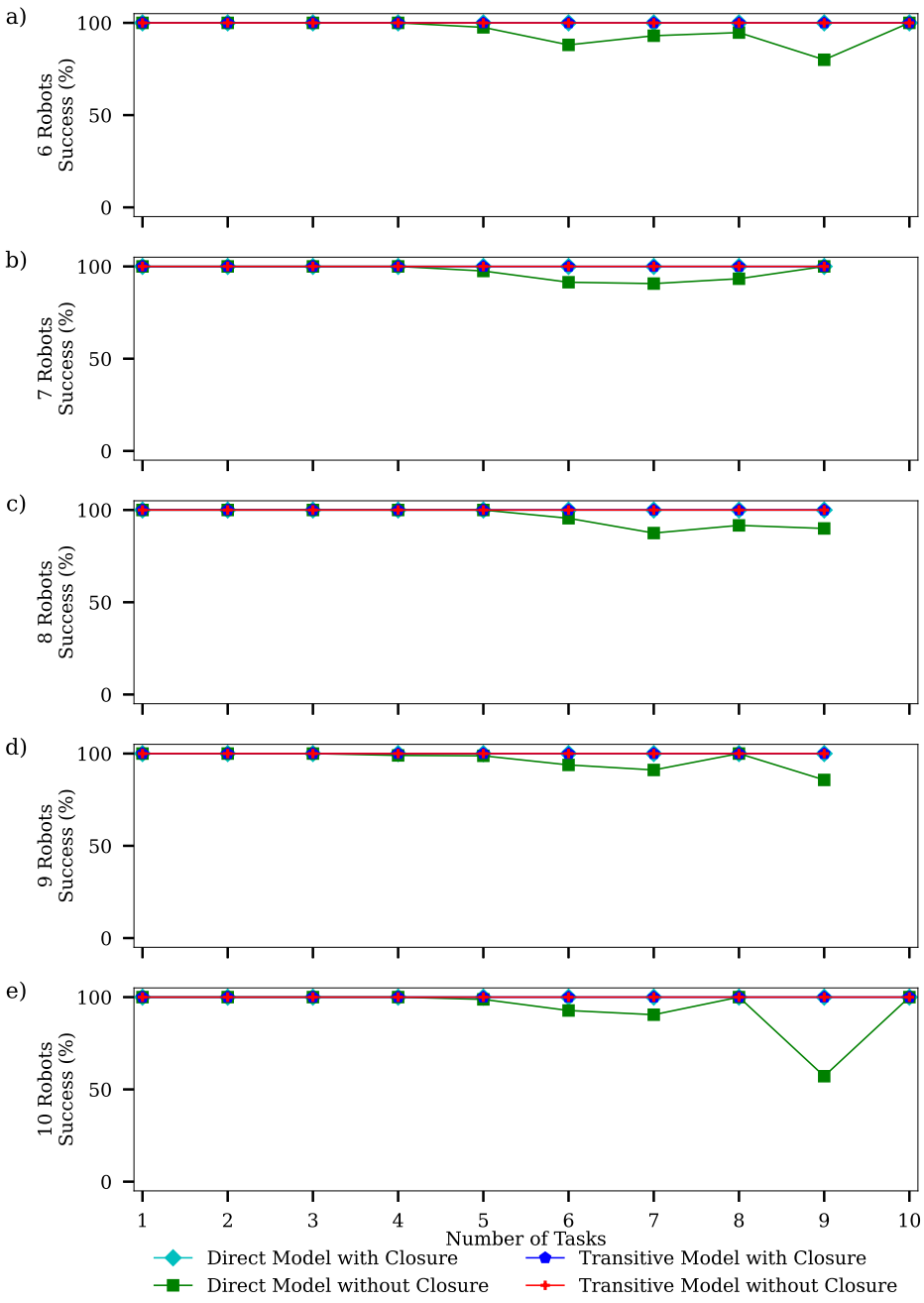


Figure C.18: Blocksworld Domain TCRA* ($\epsilon = 0$) success (%) for 6-10 robots.

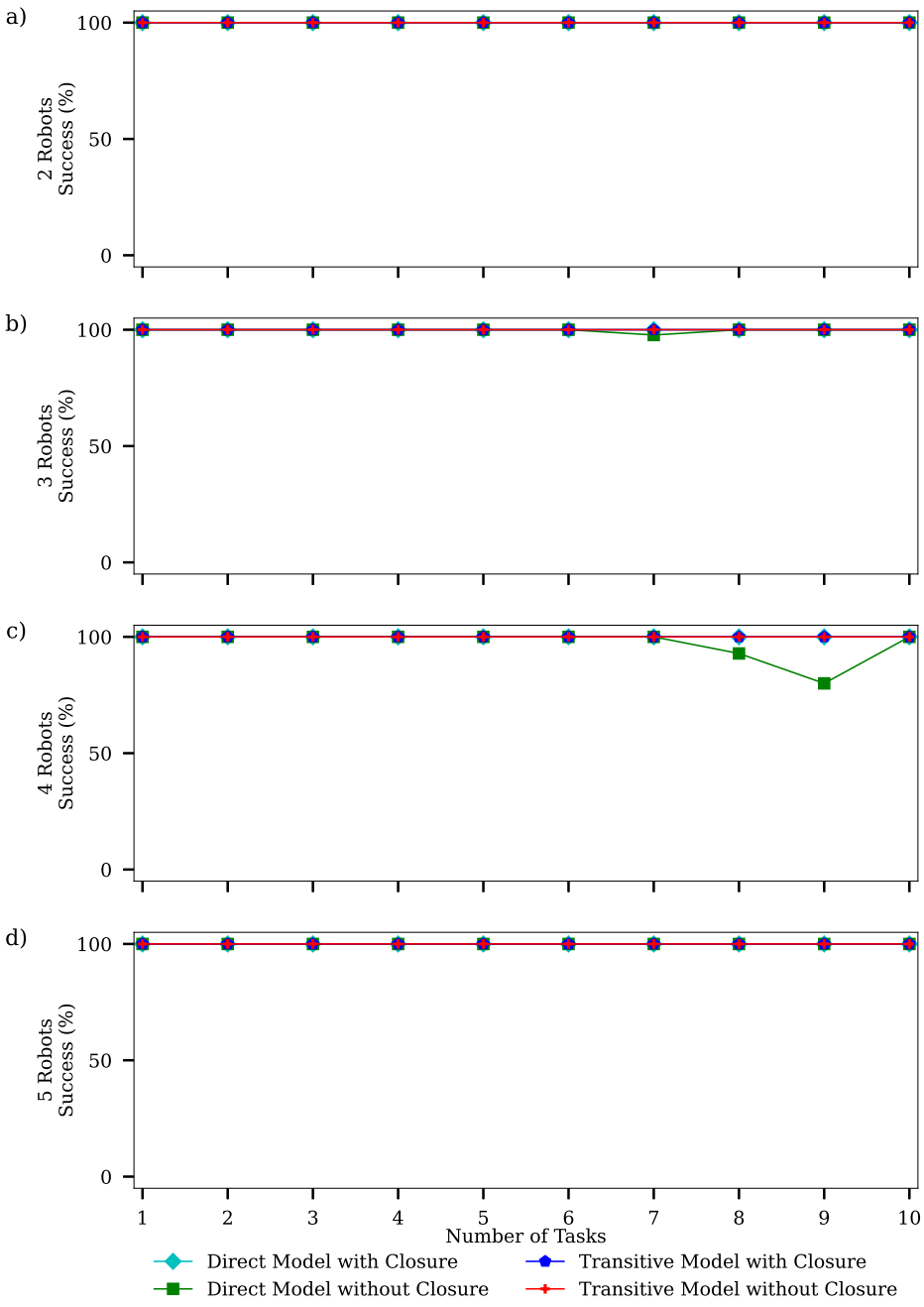


Figure C.19: Blocksworld Domain TCRA* ($\epsilon = 1$) success (%) for 1-5 robots.

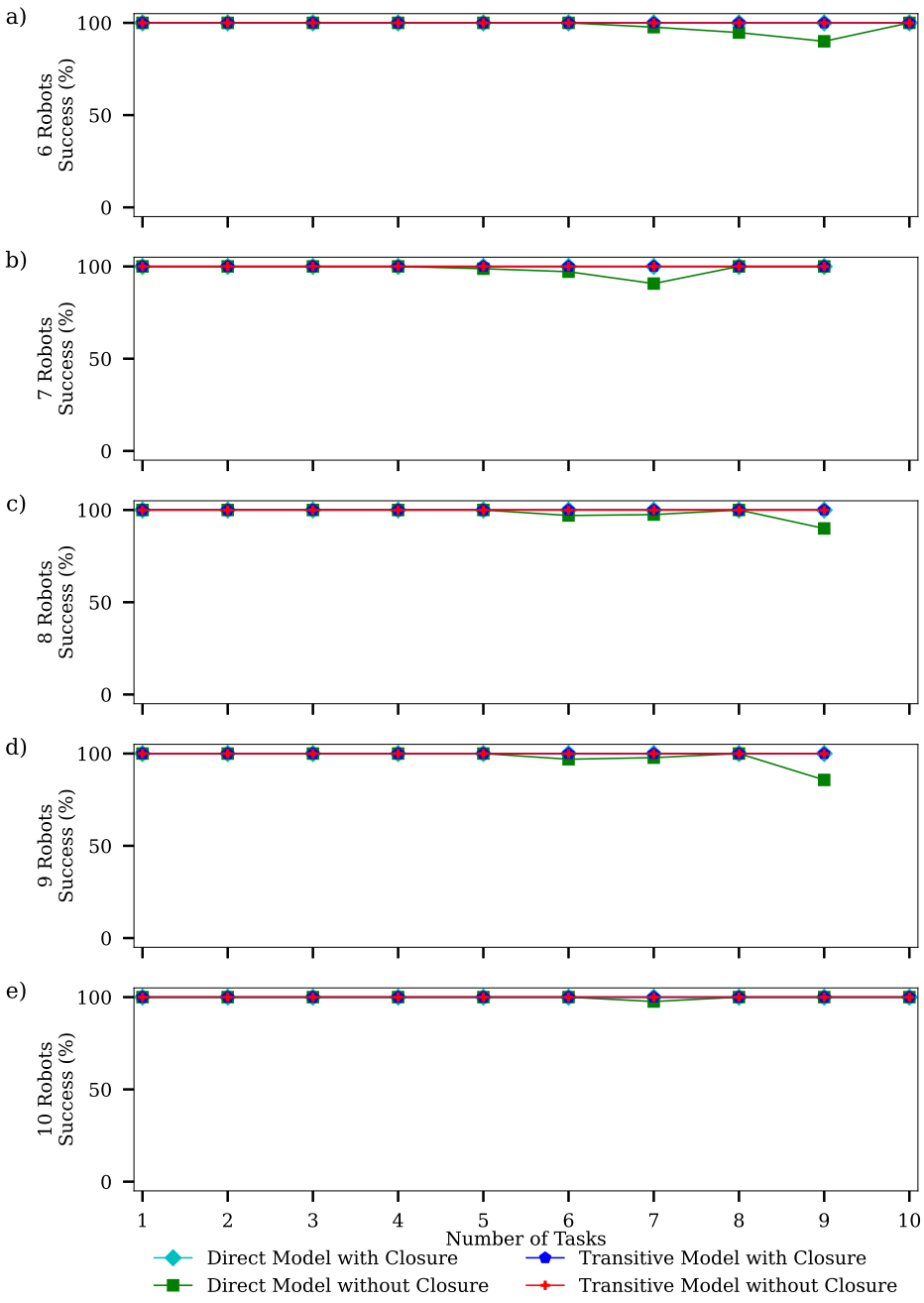


Figure C.20: Blocksworld Domain TCRA* ($\epsilon = 1$) success (%) for 6-10 robots.

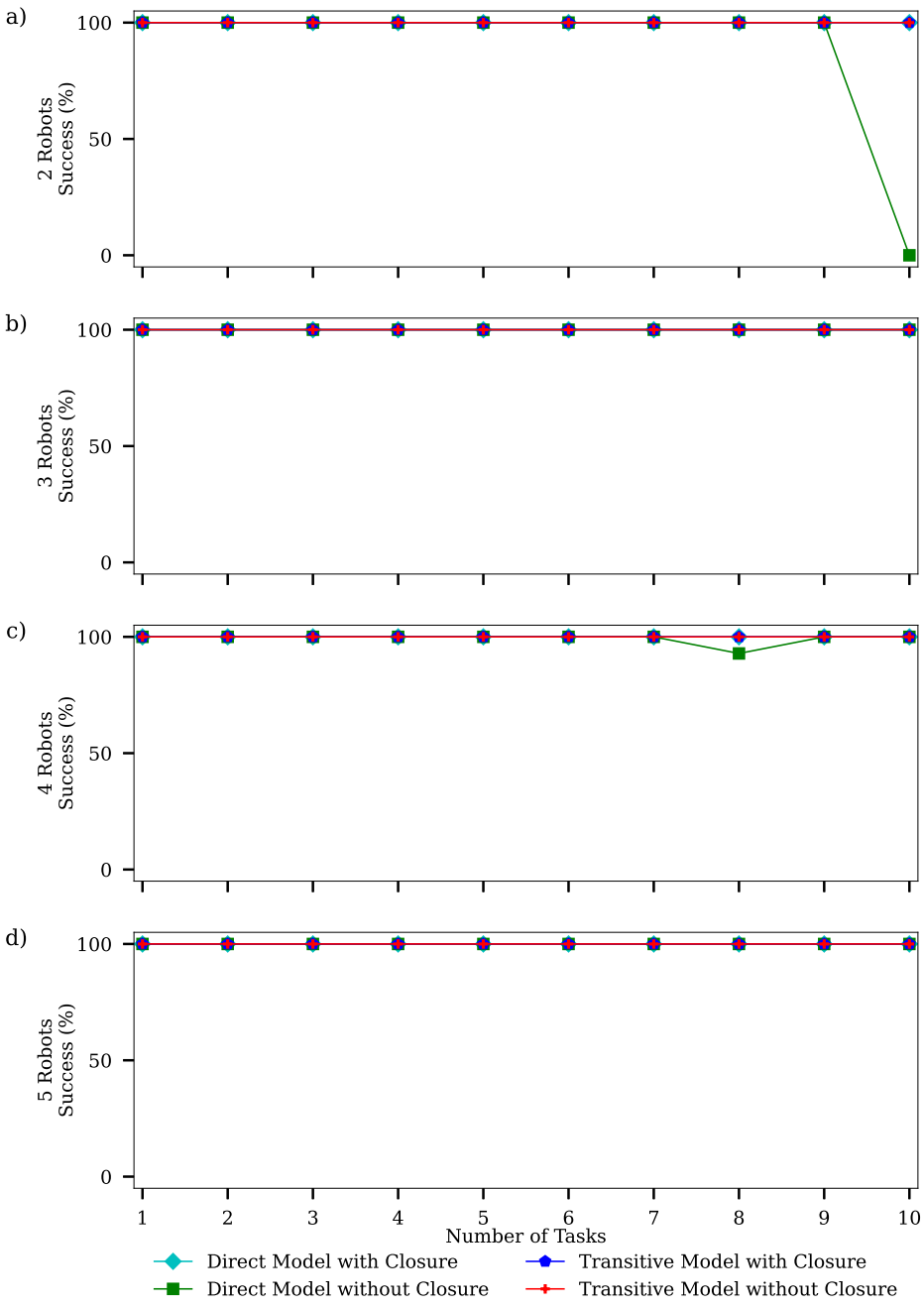


Figure C.21: Blocksworld Domain TCRA* ($\epsilon = 10$) success (%) for 1-5 robots.

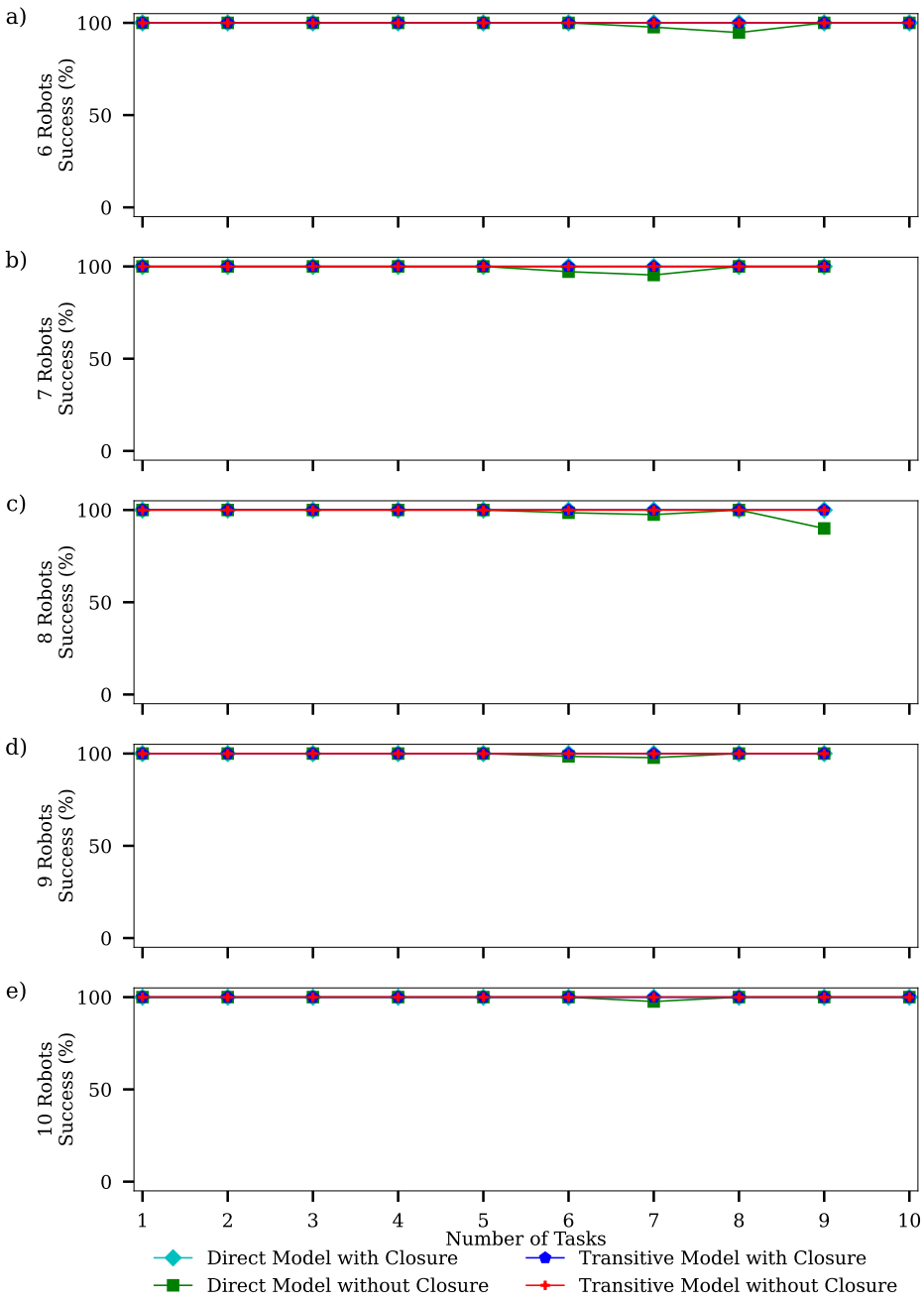


Figure C.22: Blocksworld Domain TCRA* ($\epsilon = 10$) success (%) for 6-10 robots.

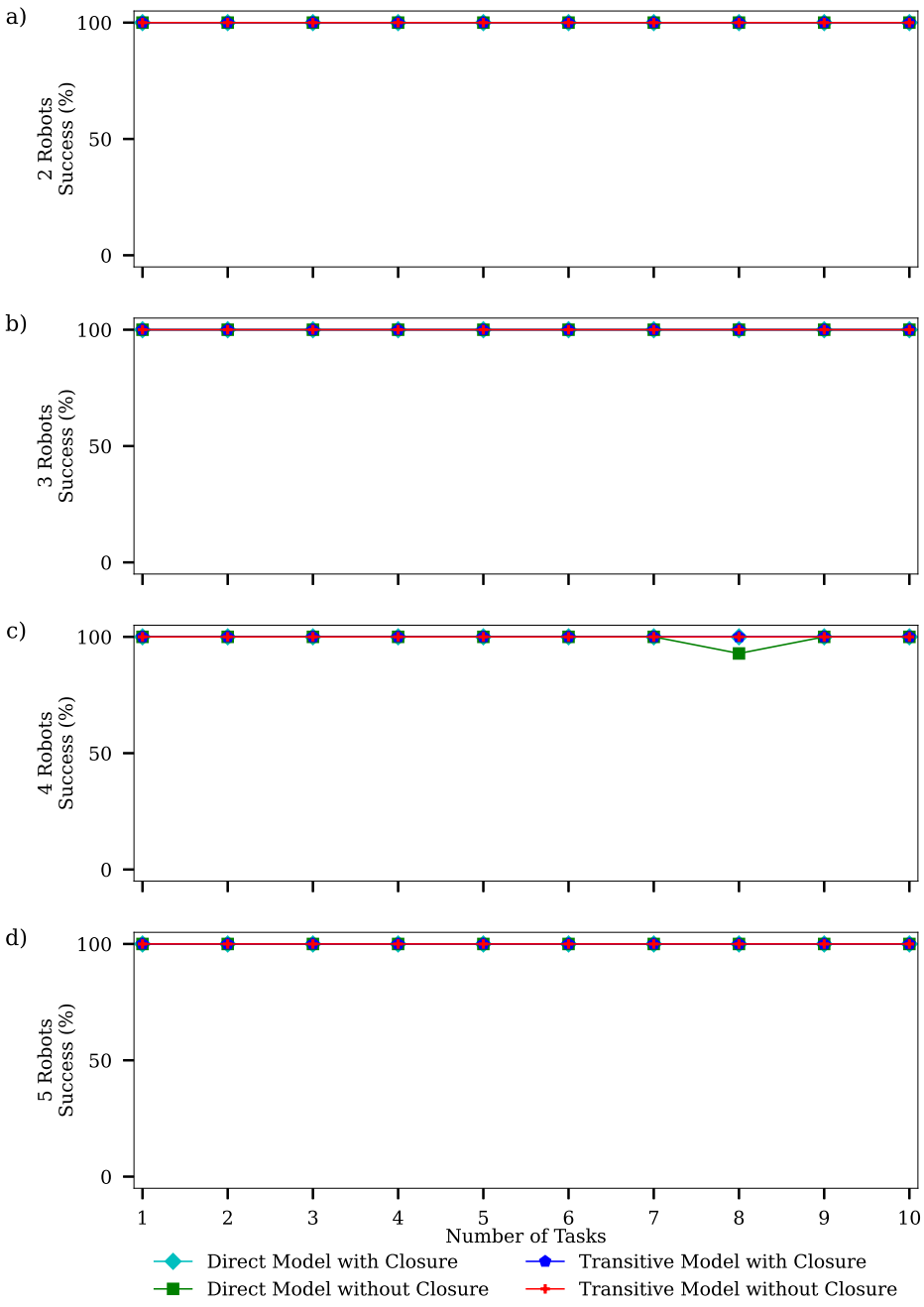


Figure C.23: Blocksworld Domain TCRA* ($\epsilon = 100$) success (%) for 1-5 robots.

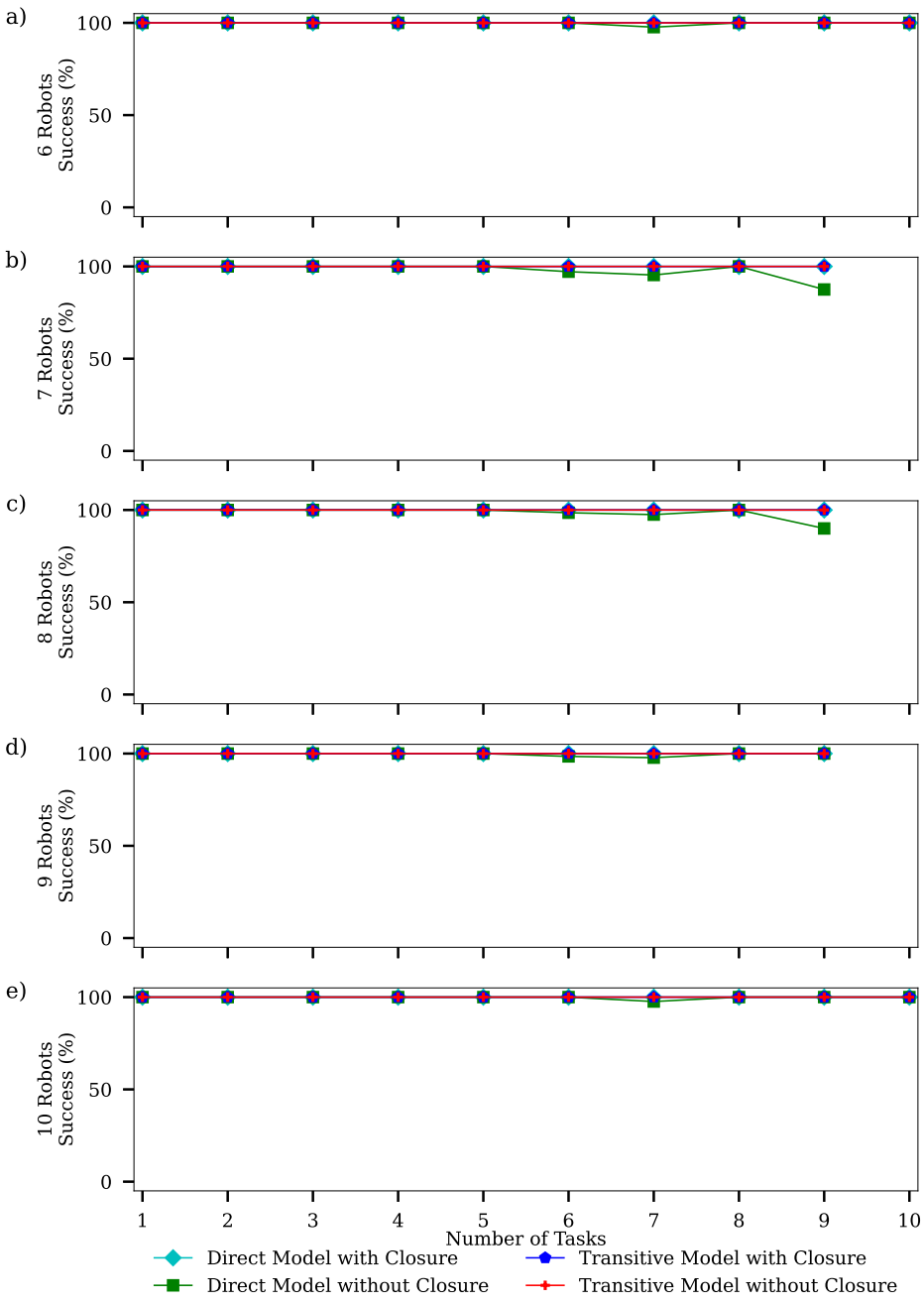


Figure C.24: Blocksworld Domain TCRA* ($\epsilon = 100$) success (%) for 6-10 robots.

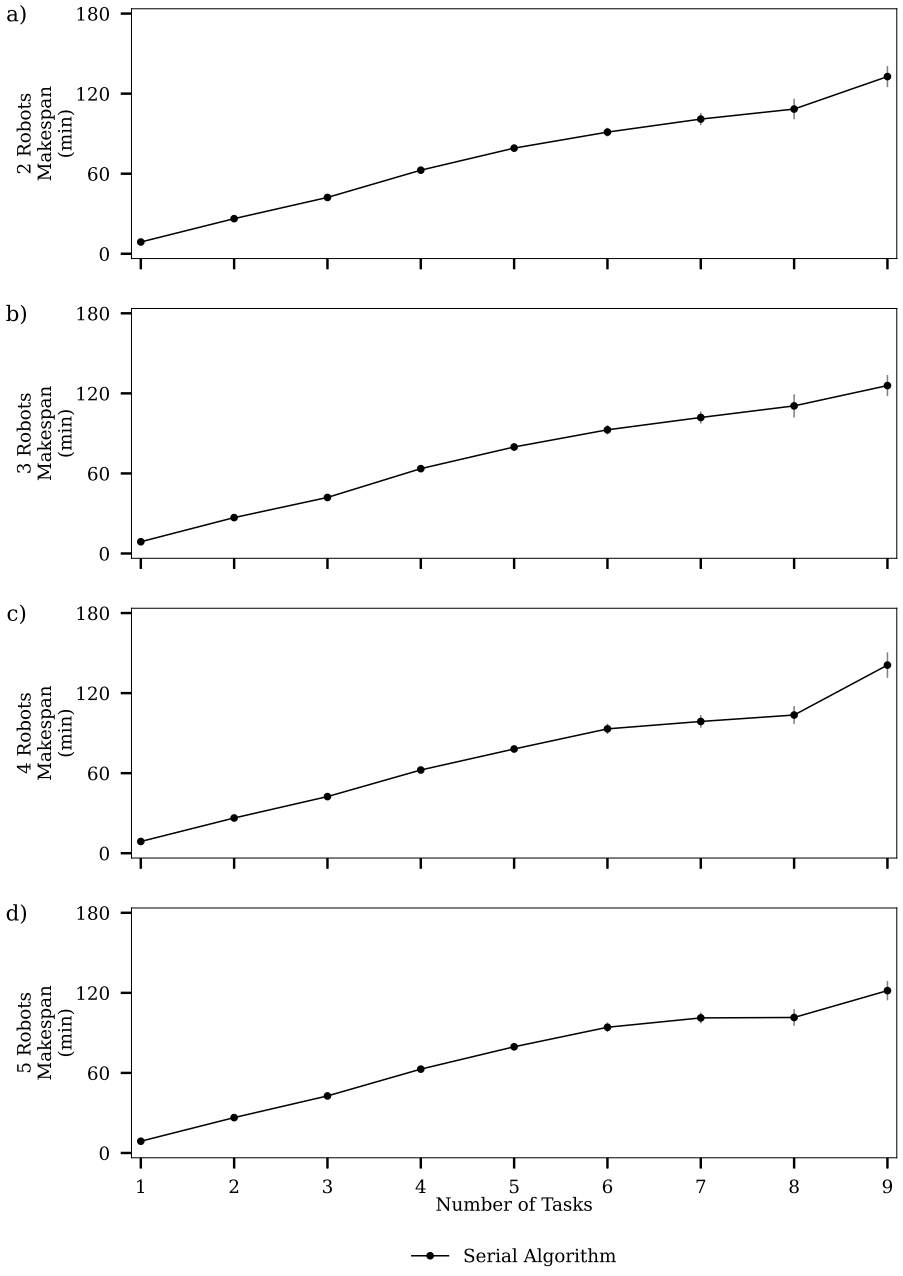


Figure C.25: Blocksworld Domain Serial makespan (min) for 1-5 robots. The maximum makespan was 150 min for all numbers of robots.

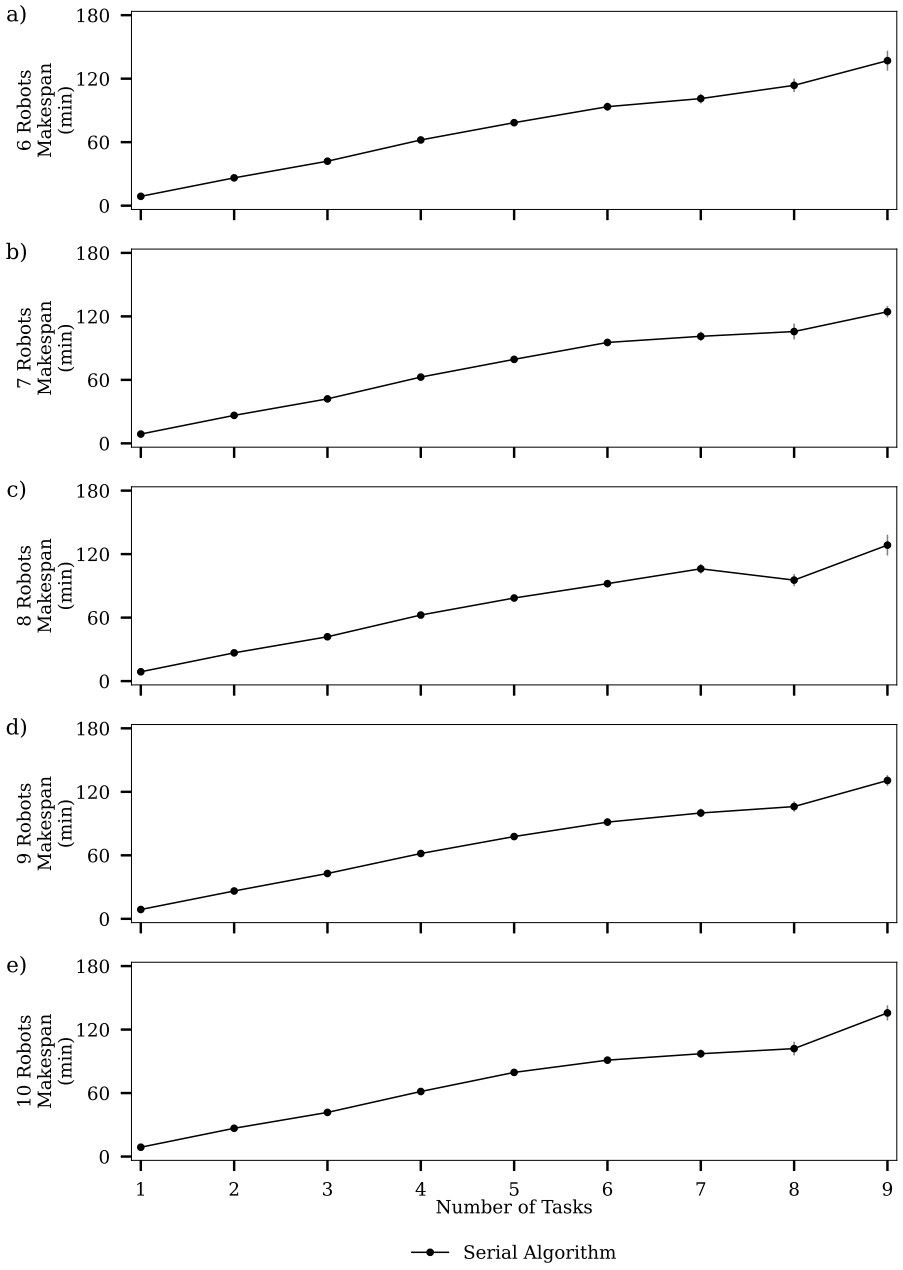


Figure C.26: Blocksworld Domain Serial makespan (min) for 6-10 robots. The maximum makespan was 150 min for all numbers of robots.

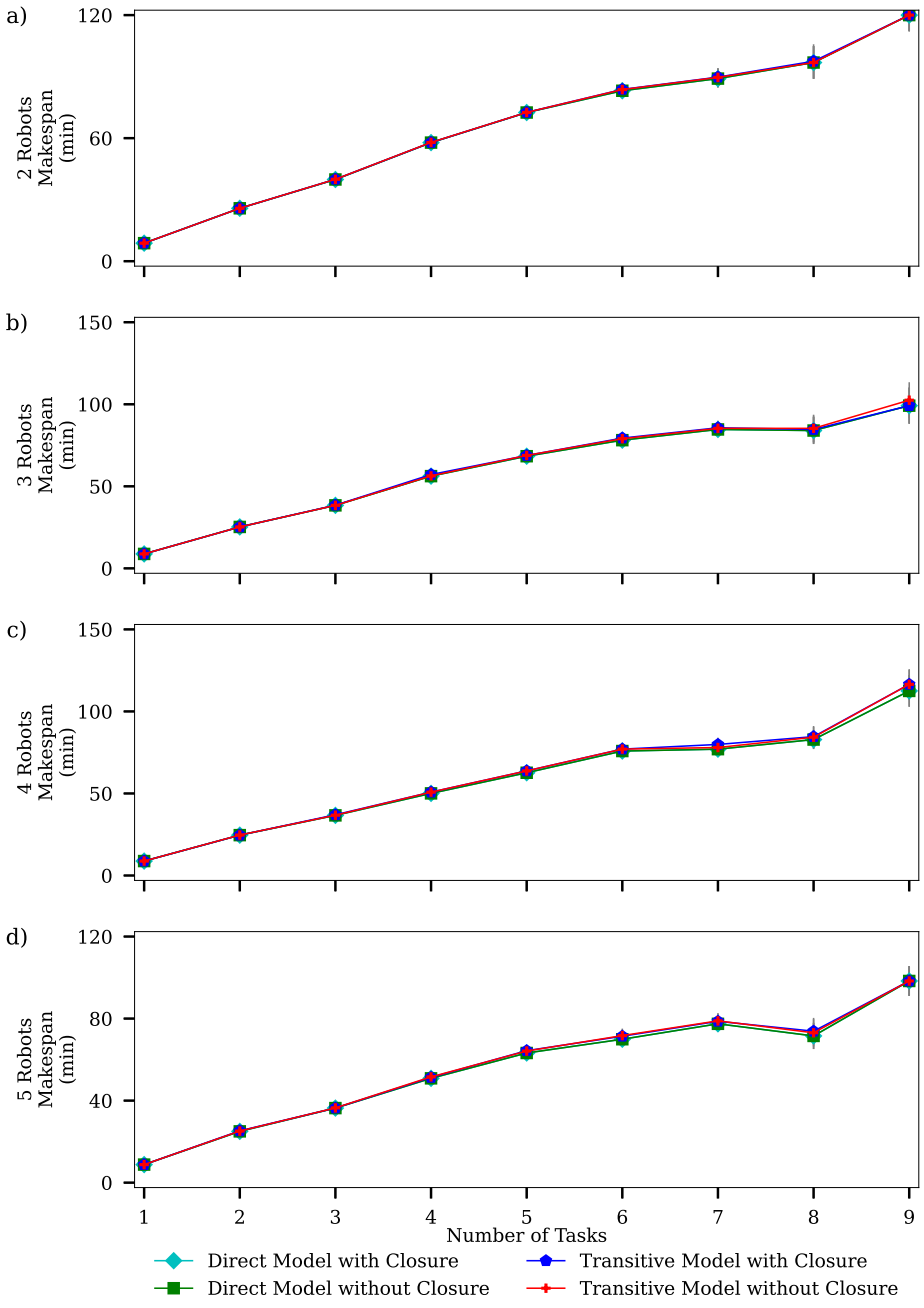


Figure C.27: Blocksworld Domain STA makespan (min) for 1-5 robots. The maximum makespan was 120, 120, 150, and 90 min for 2, 3, 4, and 5 robots, respectivelly.

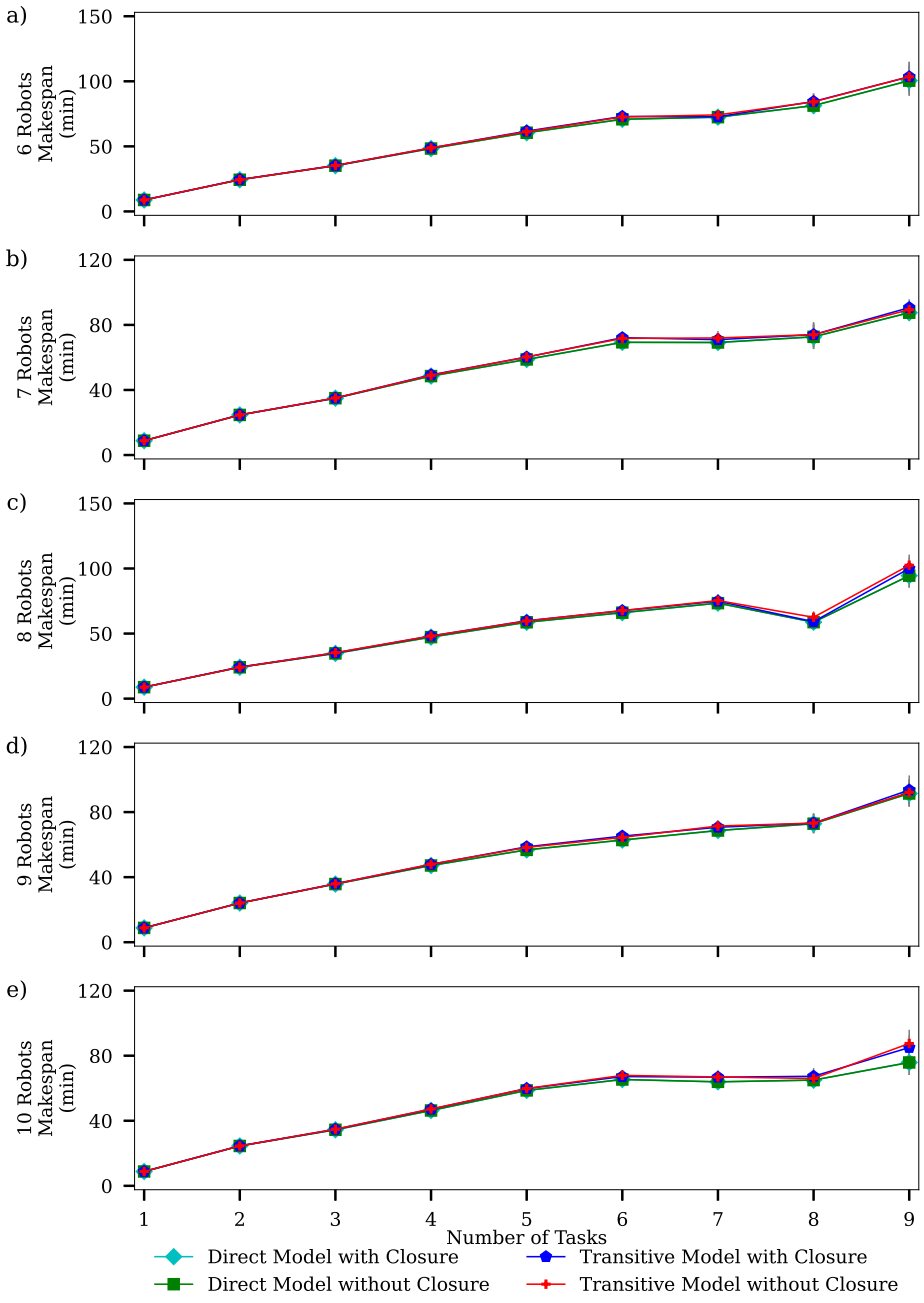


Figure C.28: Blocksworld Domain STA makespan (min) for 6-10 robots. The maximum makespan was 150, 90, 150, 90, and 90 min for 6, 7, 8, 9, and 10 robots, respectivelly.

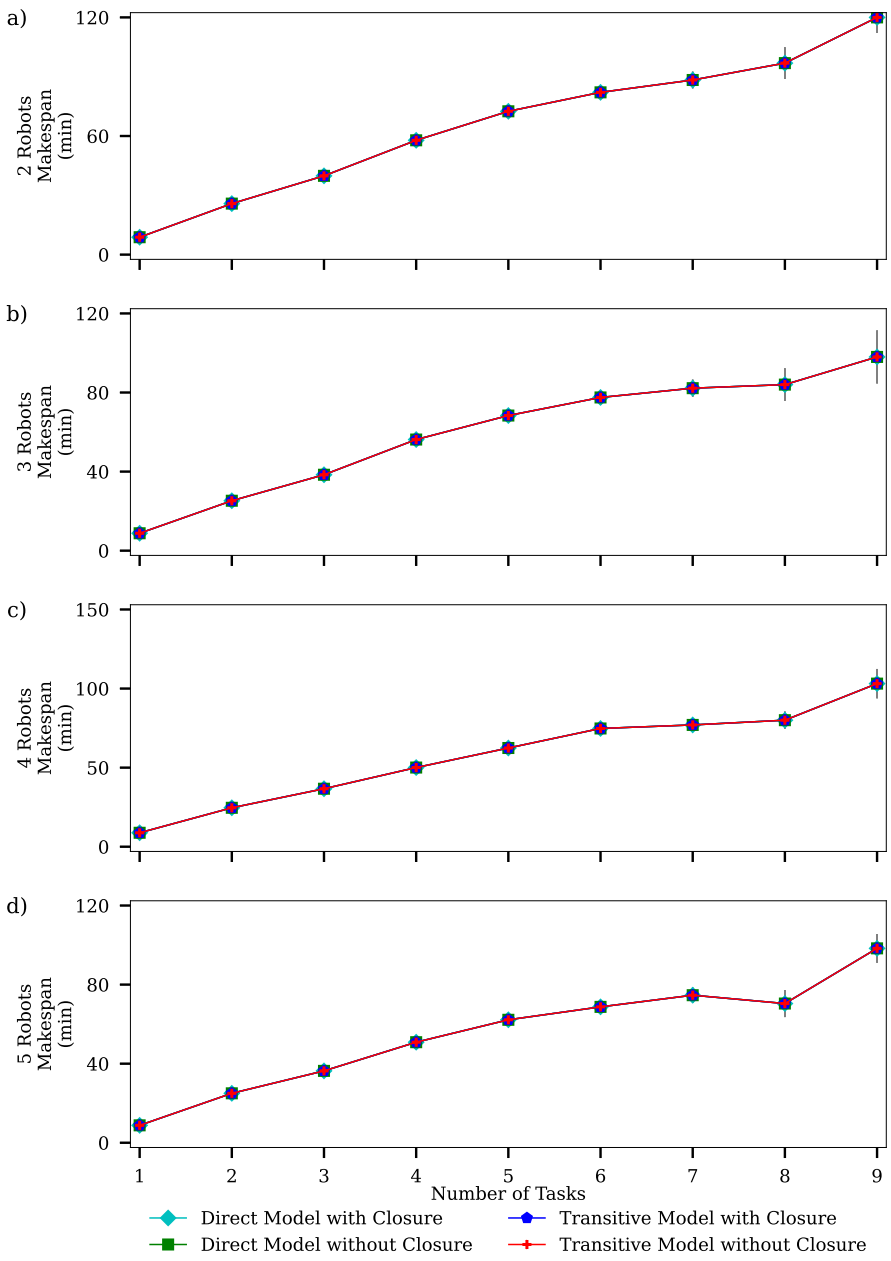


Figure C.29: Blocksworld Domain TCRA* ($\epsilon = 0$) makespan (min) for 1-5 robots. The maximum makespan was 120, 120, 150, and 120 min for 2, 3, 4, and 5 robots, respectivelly.

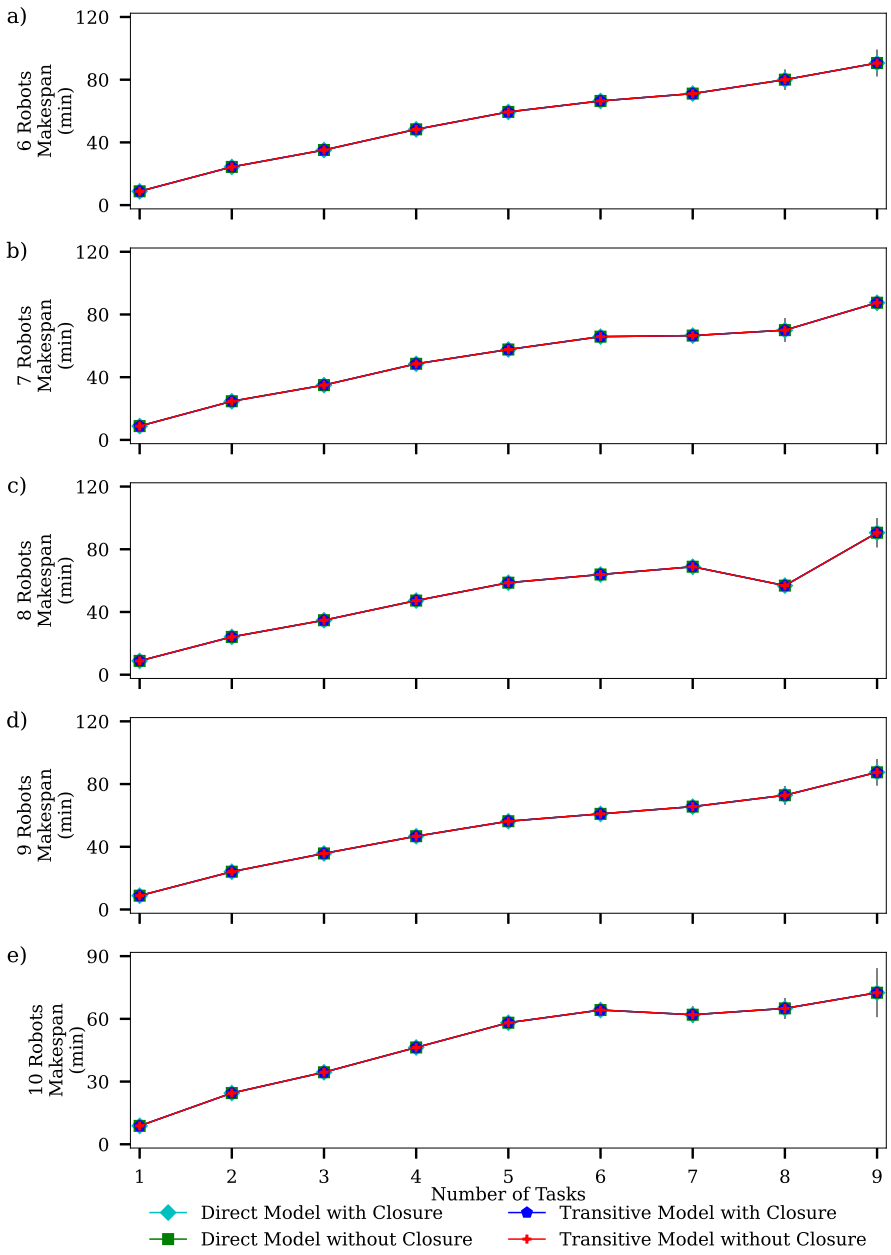


Figure C.30: Blocksworld Domain TCRA* ($\epsilon = 0$) makespan (min) for 6-10 robots. The maximum makespan was 120, 120, 120, 120, and 90 min for 6, 7, 8, 9, and 10 robots, respectivelly.

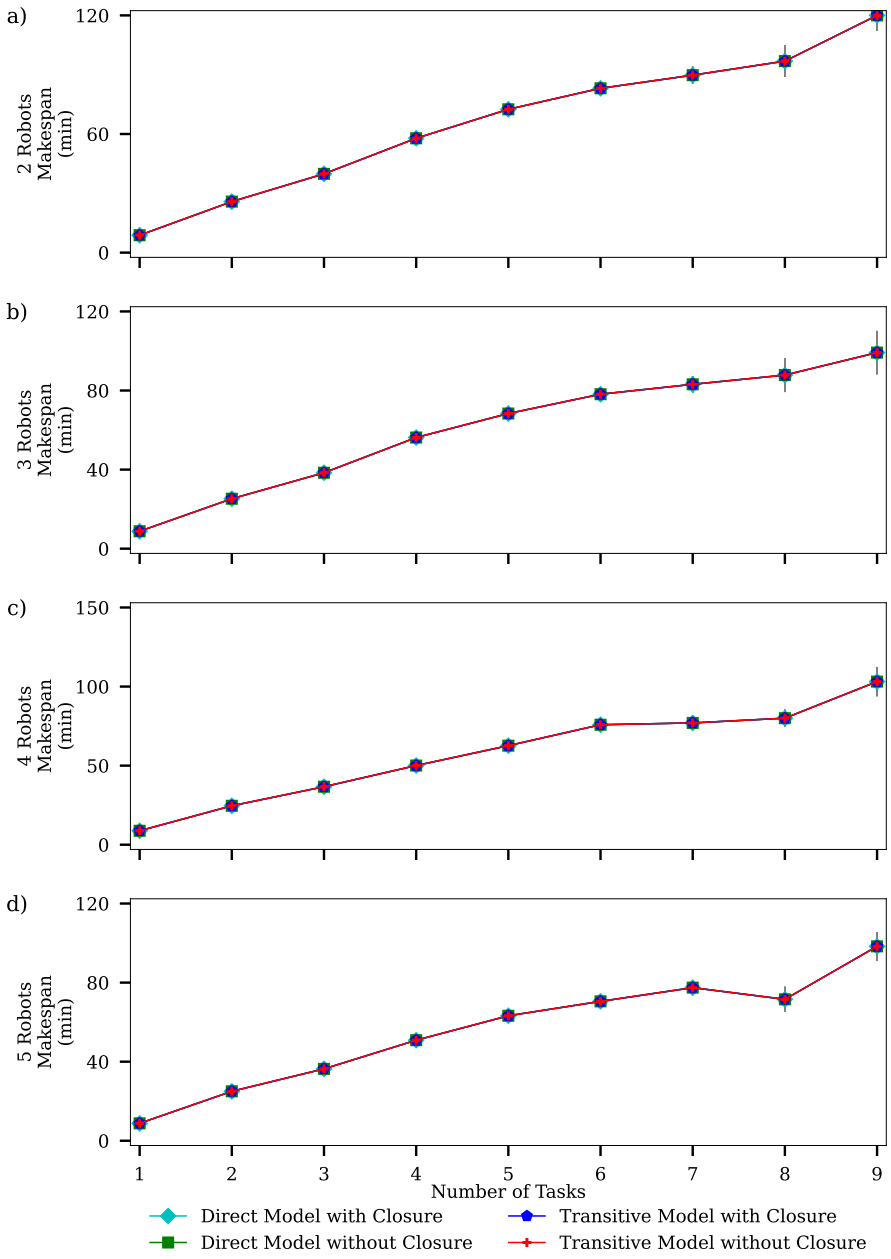


Figure C.31: Blocksworld Domain TCRA* ($\epsilon = 1$) makespan (min) for 1-5 robots. The maximum makespan was 120, 120, 150, and 120 min for 2, 3, 4, and 5 robots, respectivelly.

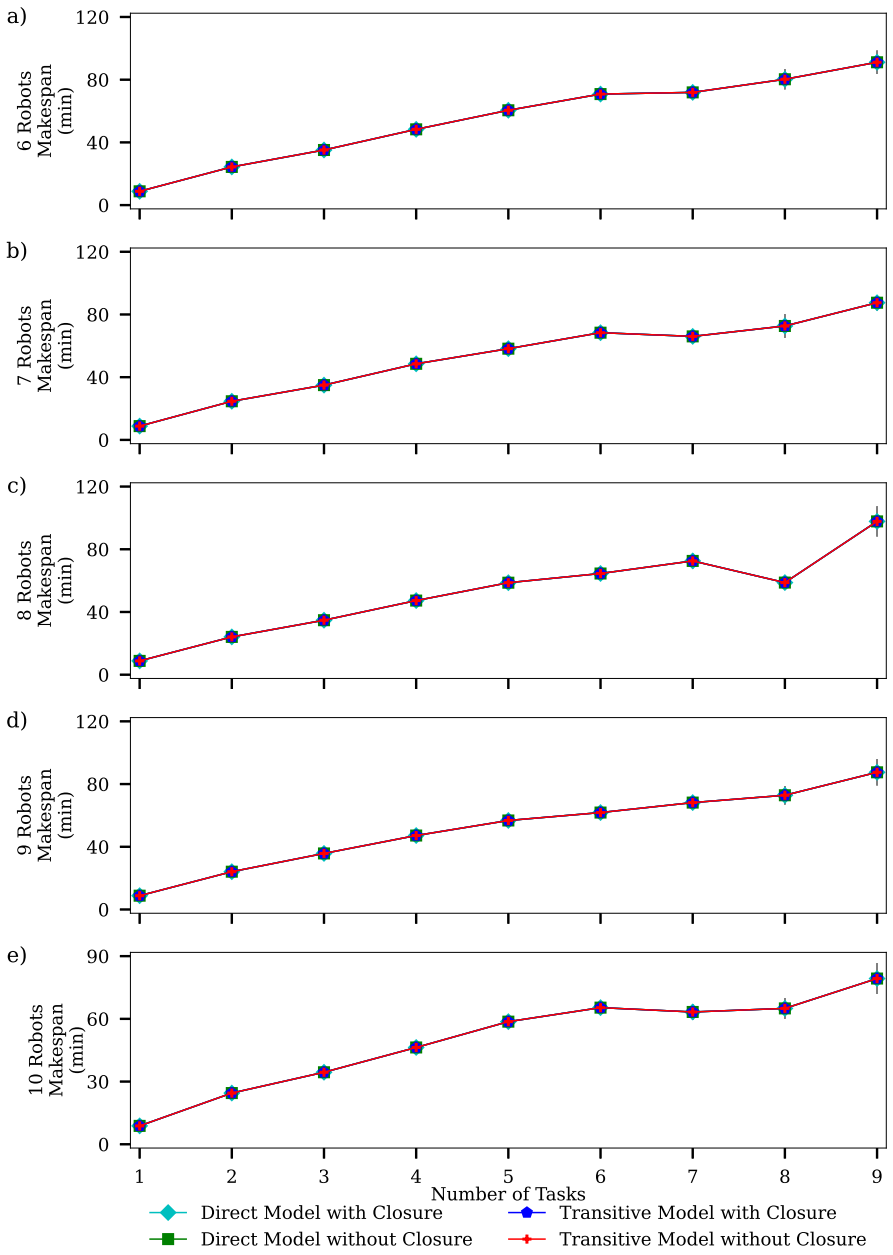


Figure C.32: Blocksworld Domain TCRA* ($\epsilon = 1$) makespan (min) for 6-10 robots. The maximum makespan was 120, 120, 120, 120, and 90 min for 6, 7, 8, 9, and 10 robots, respectivelly.

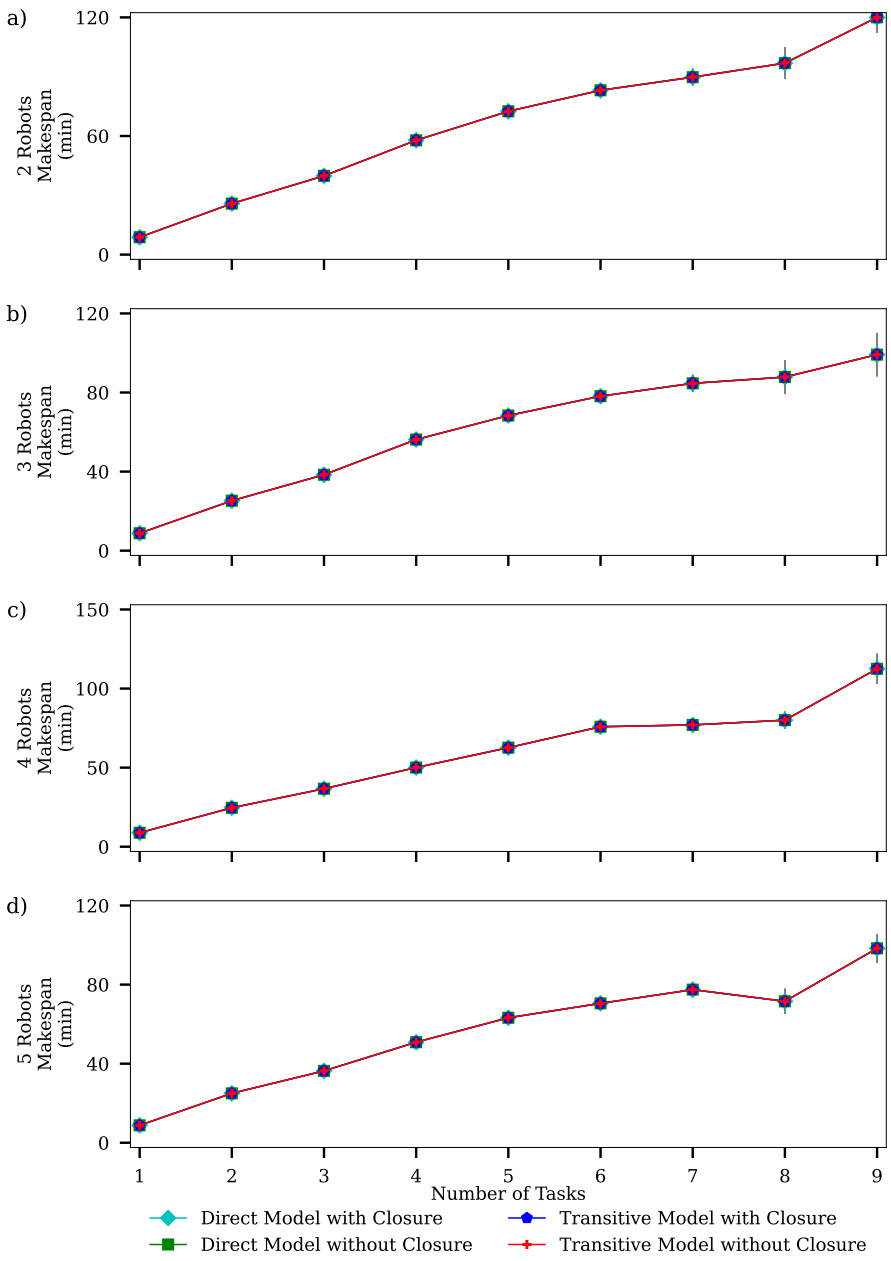


Figure C.33: Blocksworld Domain TCRA* ($\epsilon = 10$) makespan (min) for 1-5 robots. The maximum makespan was 120, 120, 150, and 120 min for 2, 3, 4, and 5 robots, respectivelly.

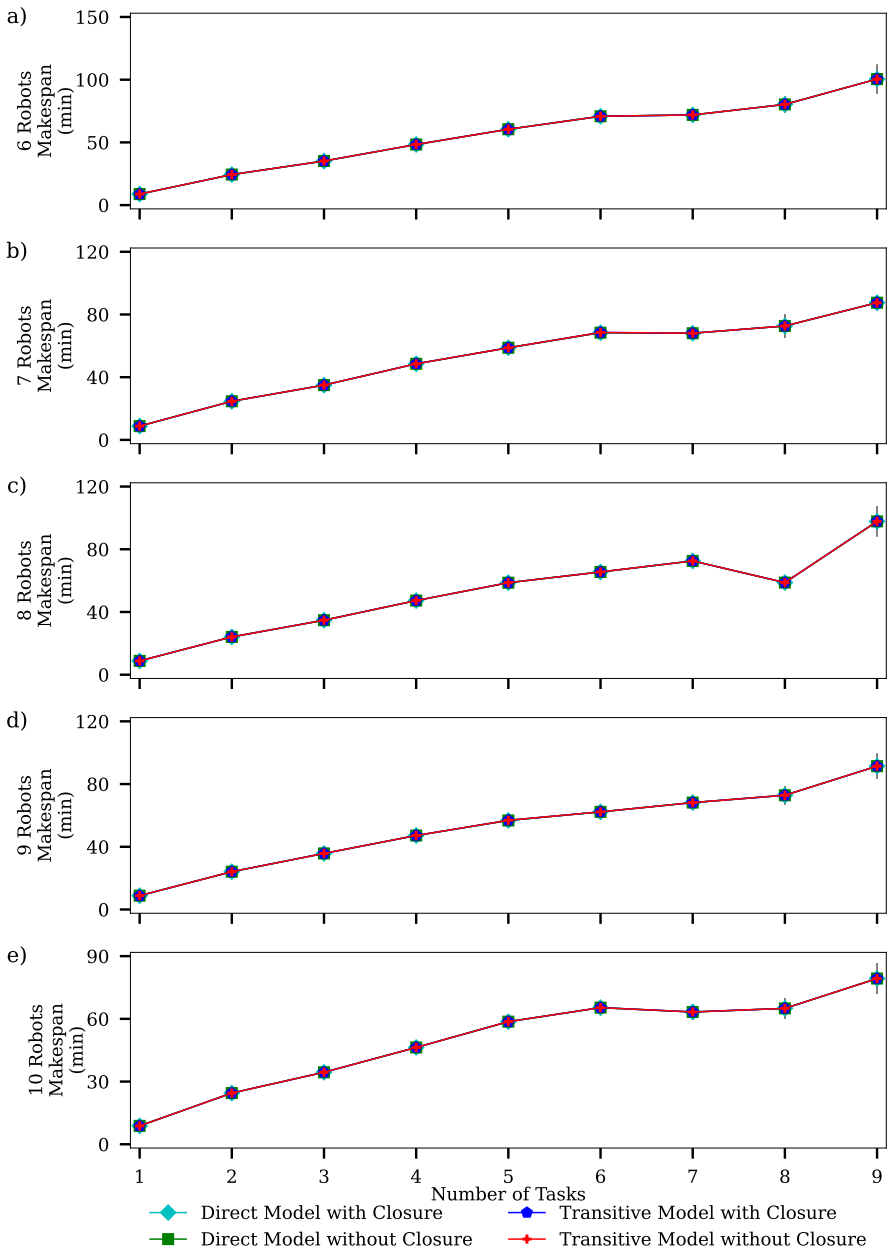


Figure C.34: Blocksworld Domain TCRA* ($\epsilon = 10$) makespan (min) for 6-10 robots. The maximum makespan was 150, 120, 120, 120, and 90 min for 6, 7, 8, 9, and 10 robots, respectivelly.

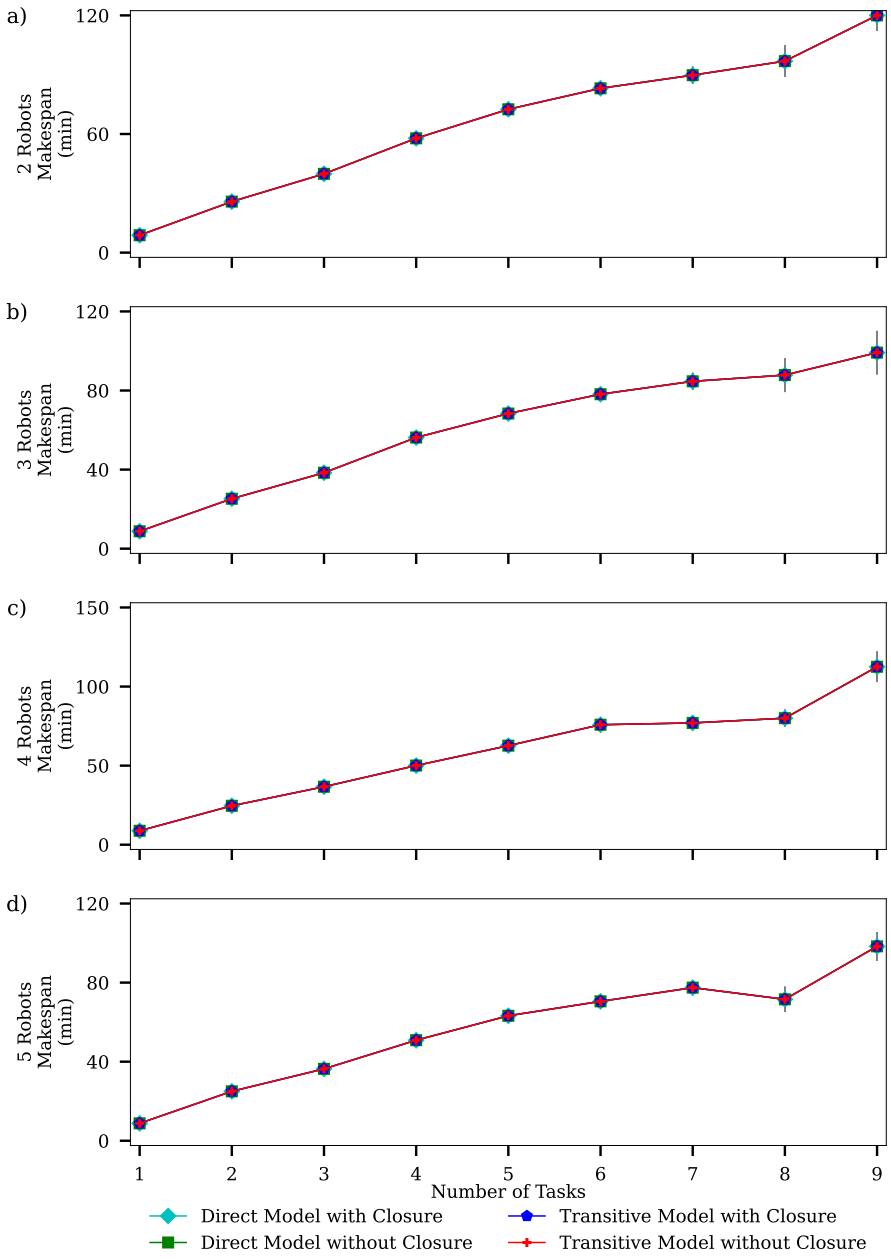


Figure C.35: Blocksworld Domain TCRA* ($\epsilon = 100$) makespan (min) for 1-5 robots. The maximum makespan was 120, 120, 150, and 120 min for 2, 3, 4, and 5 robots, respectivelly.

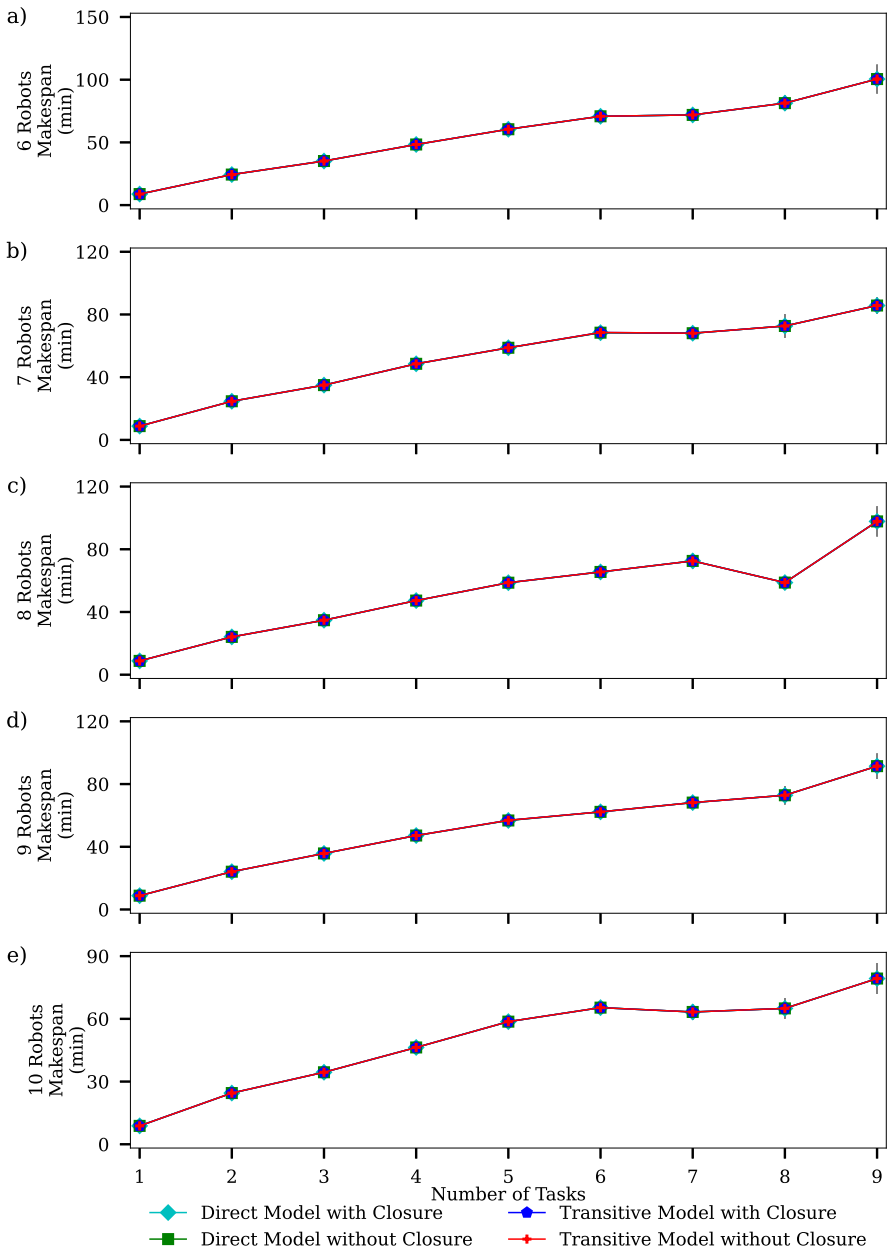


Figure C.36: Blocksworld Domain TCRA* ($\epsilon = 100$) makespan (min) for 6-10 robots. The maximum makespan was 150, 120, 120, 120, and 90 min for 6, 7, 8, 9, and 10 robots, respectivelly.

C.3 The First Response Domain

All the First Response Domain problems were successfully generated, for all tasks; however, the number of experimental samples for the aggregated makespan results diminished as the number of tasks increased, because of the diminishing plan merging success rates, as explained in Chapter 5.3. Therefore, the makespan results become noisier as the number of tasks grows, and are missing, due to the missing successful samples, for the highest number of tasks.

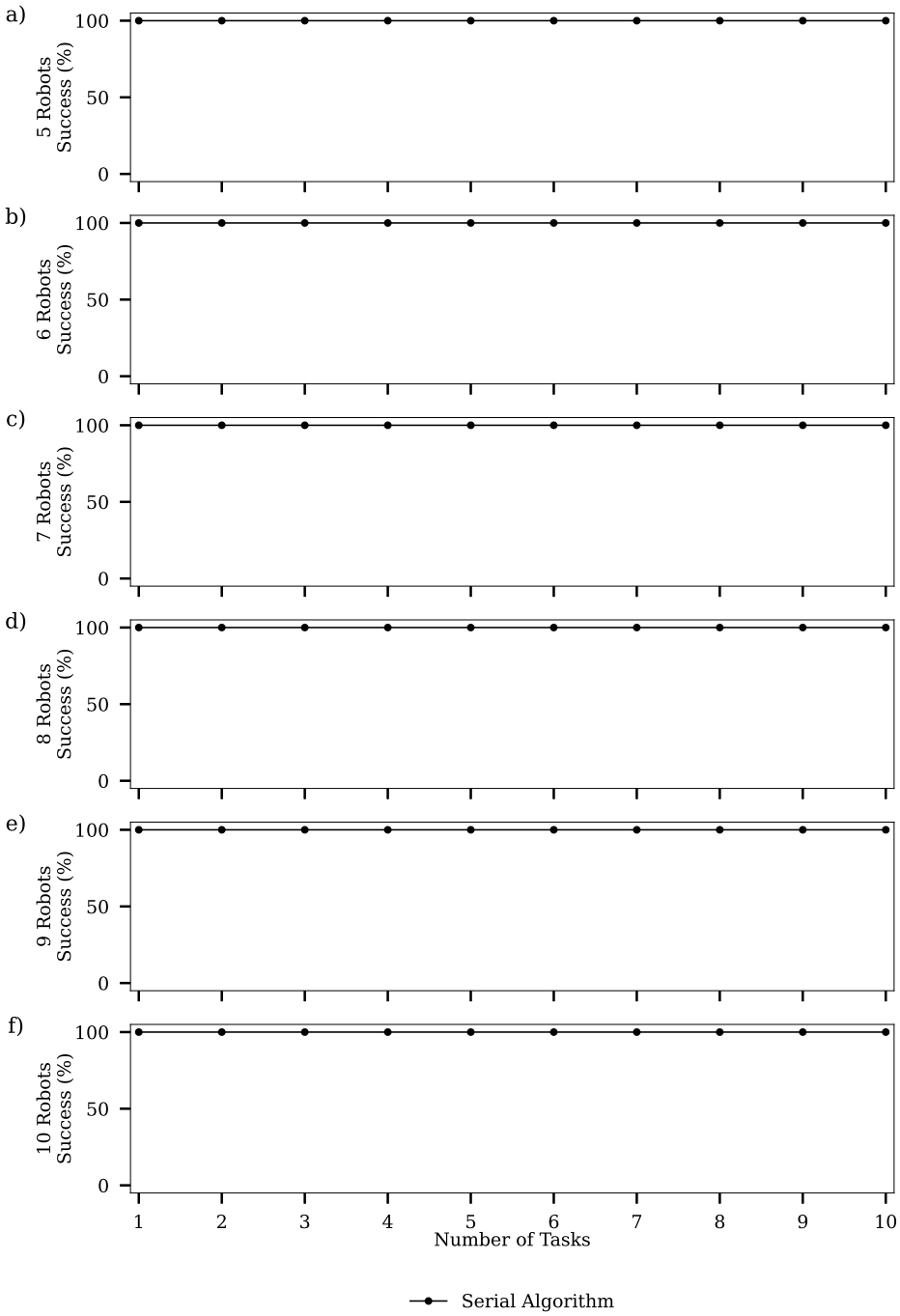


Figure C.37: First Response Domain Serial success (%).

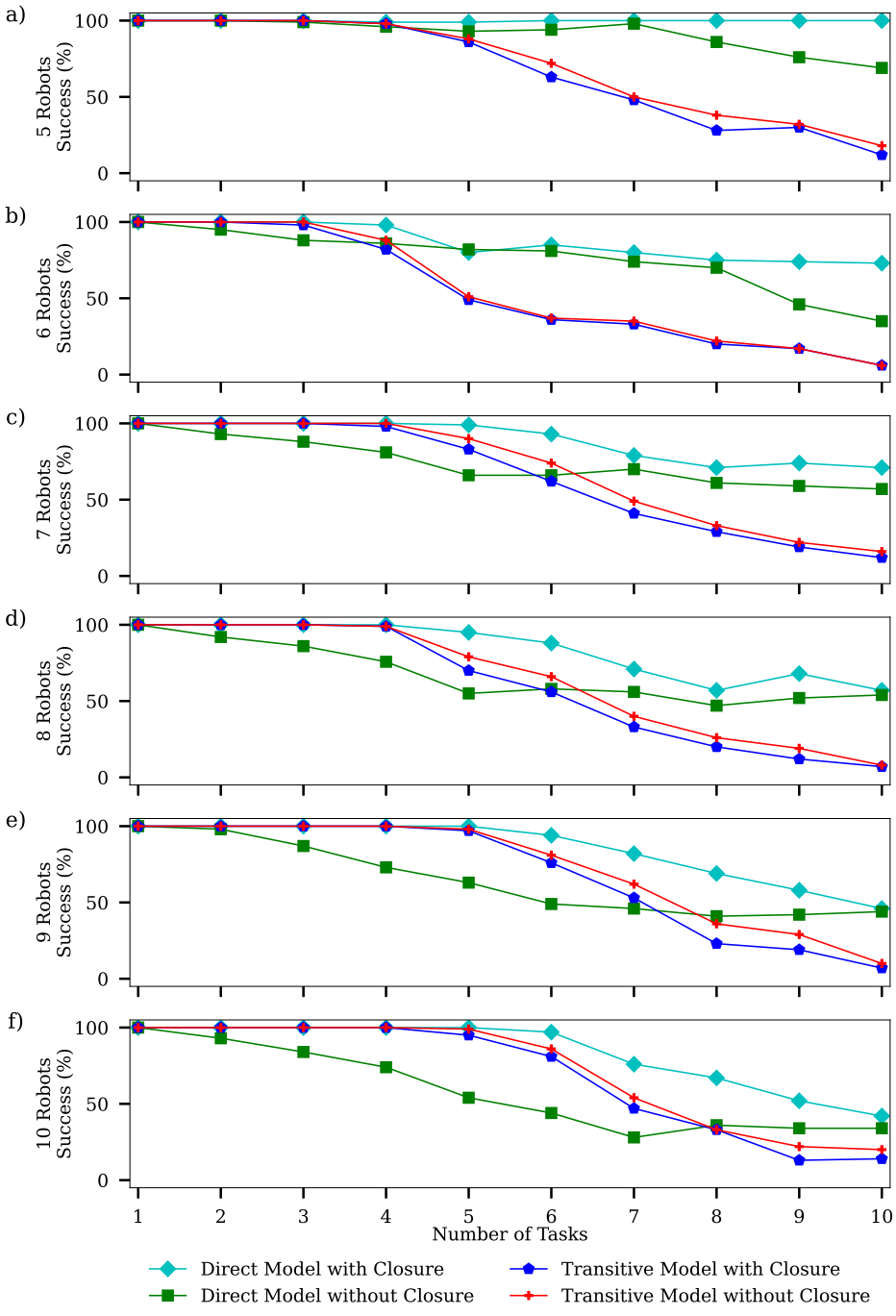


Figure C.38: First Response Domain STA success (%).

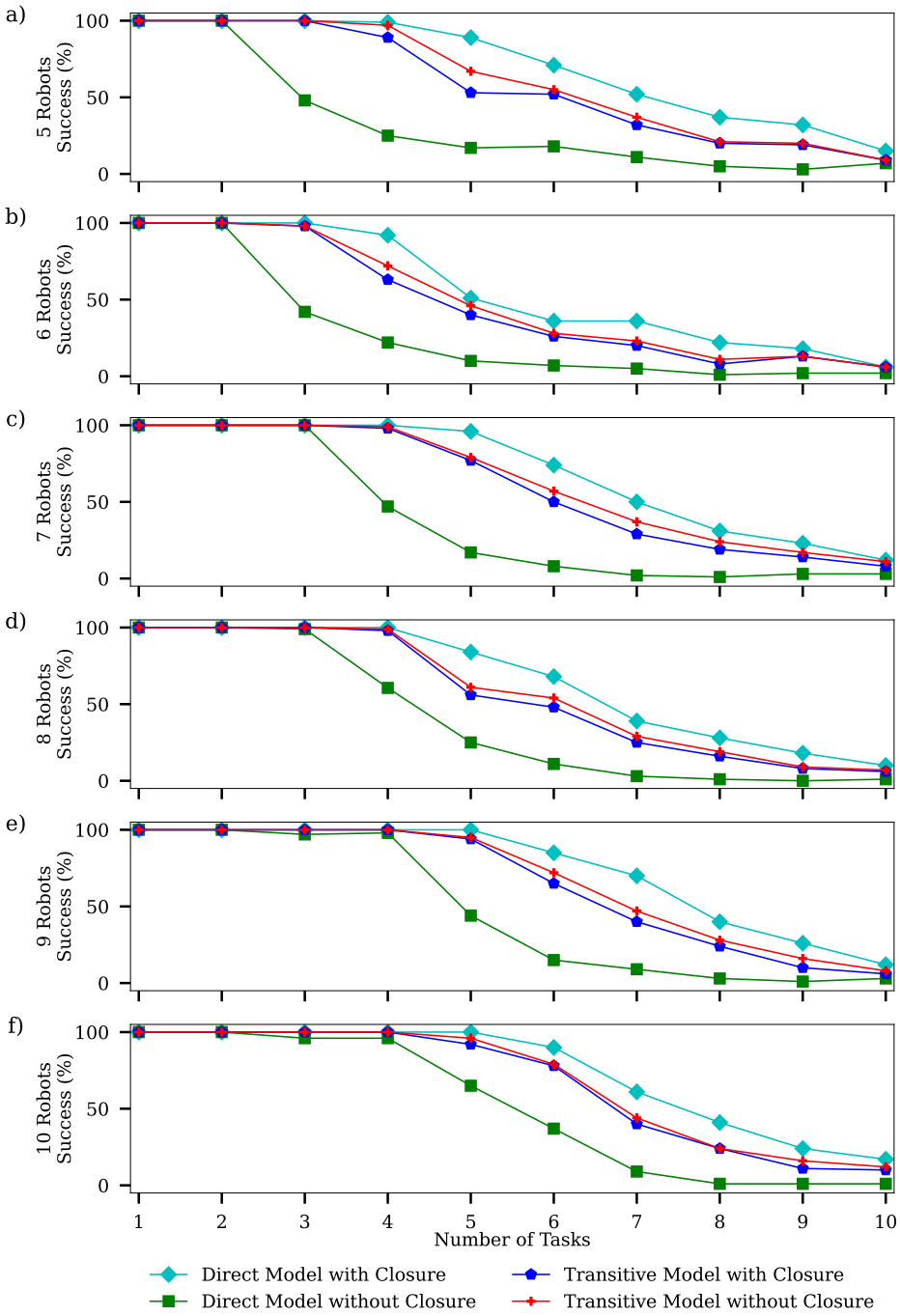


Figure C.39: First Response Domain TCRA* ($\epsilon = 0$) success (%).

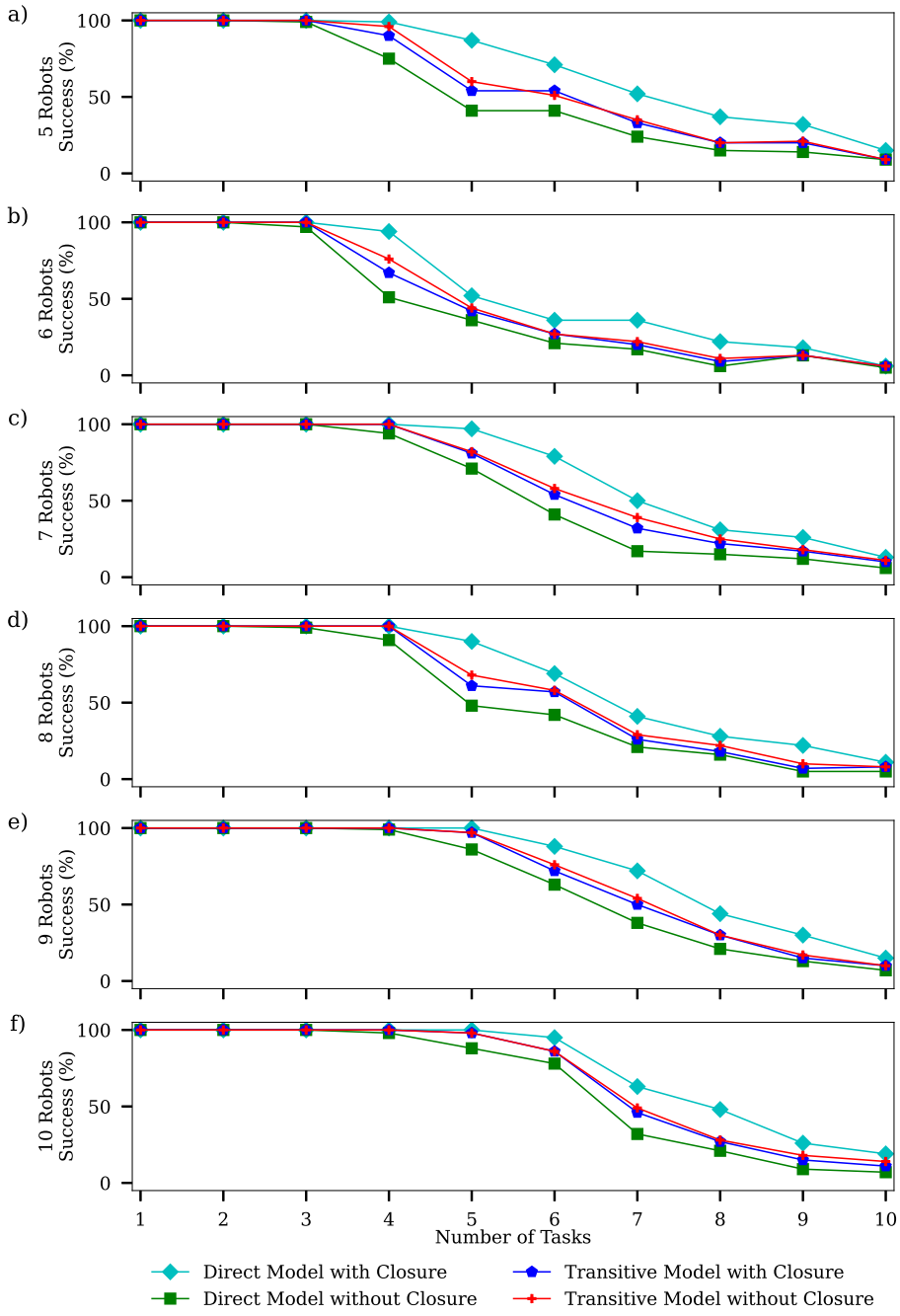


Figure C.40: First Response Domain TCRA* ($\epsilon = 1$) success (%).

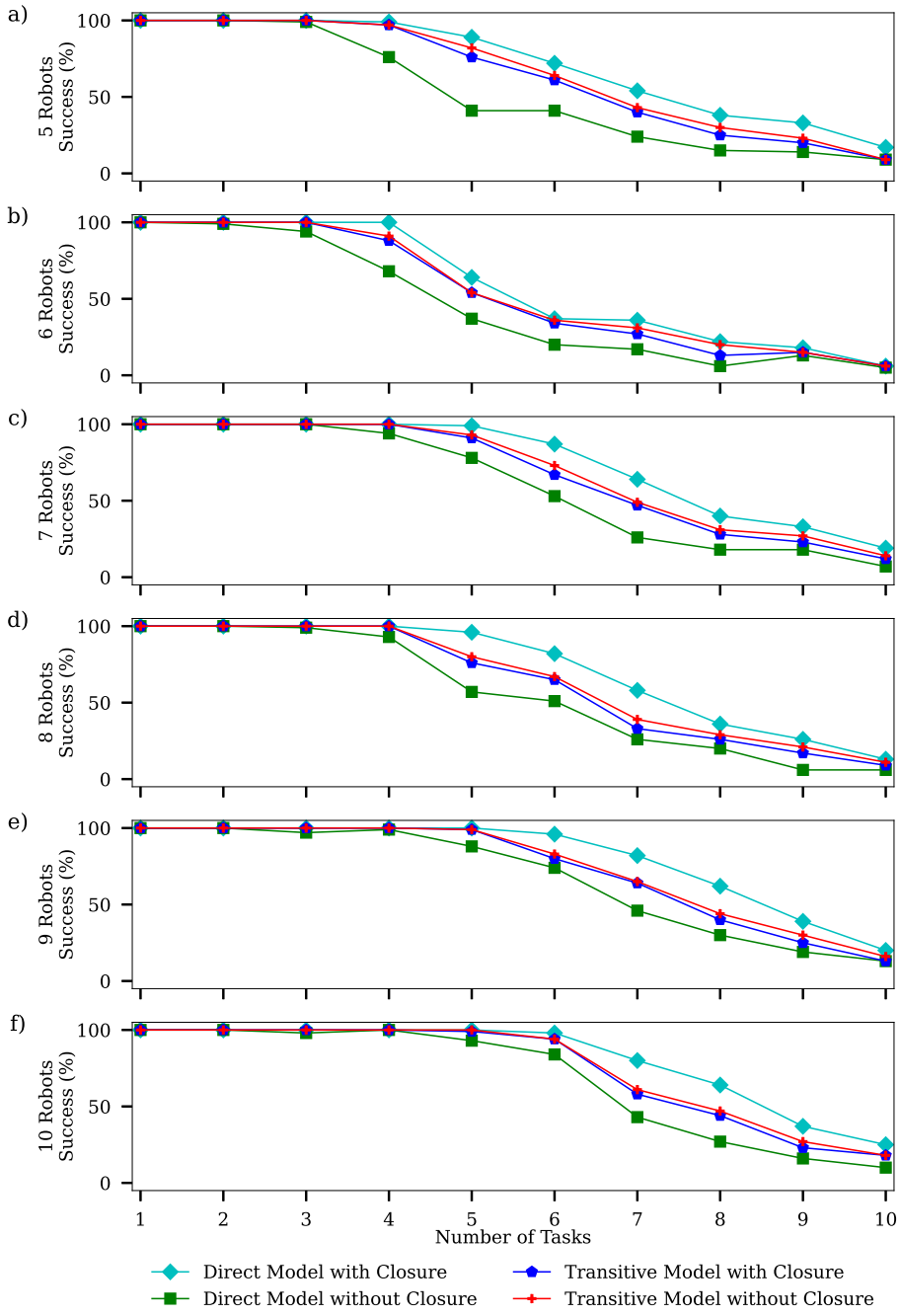


Figure C.41: First Response Domain TCRA* ($\epsilon = 10$) success (%).

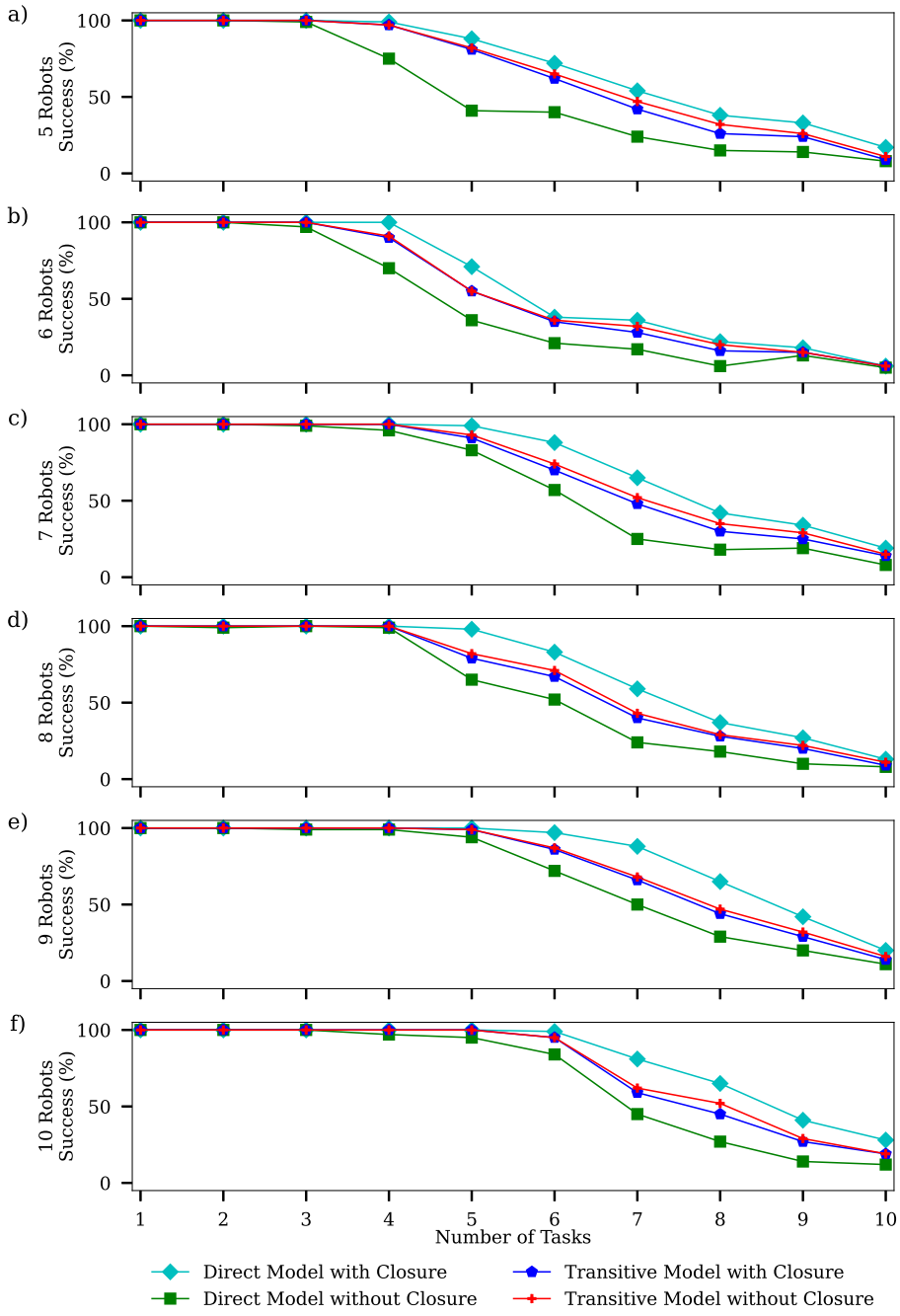


Figure C.42: First Response Domain TCRA* ($\epsilon = 100$) success (%).

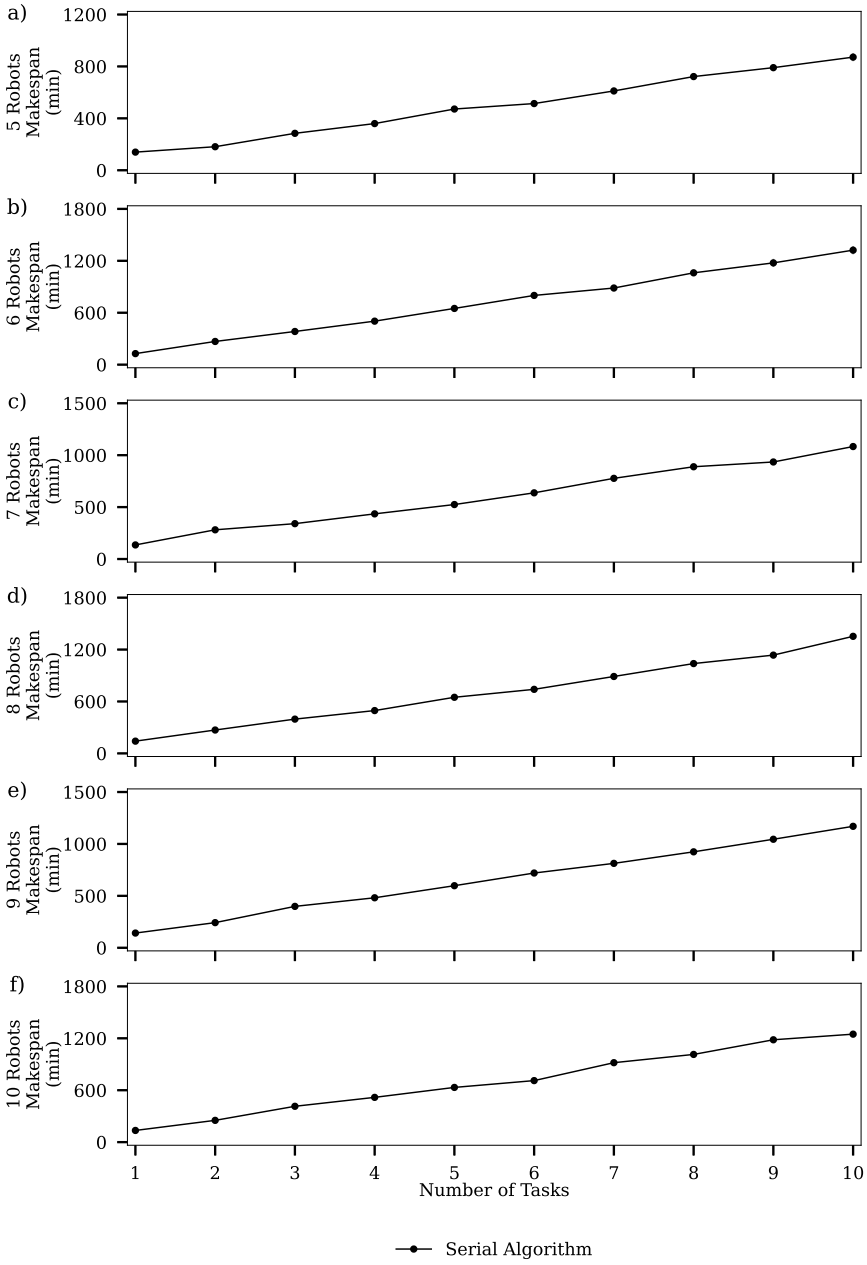


Figure C.43: First Response Domain Serial makespan (min). The maximum makespan was 1200, 1800, 1500, 1800, 1500, and 1800 min for 5, 6, 7, 8, 9, and 10 robots, respectivelly.

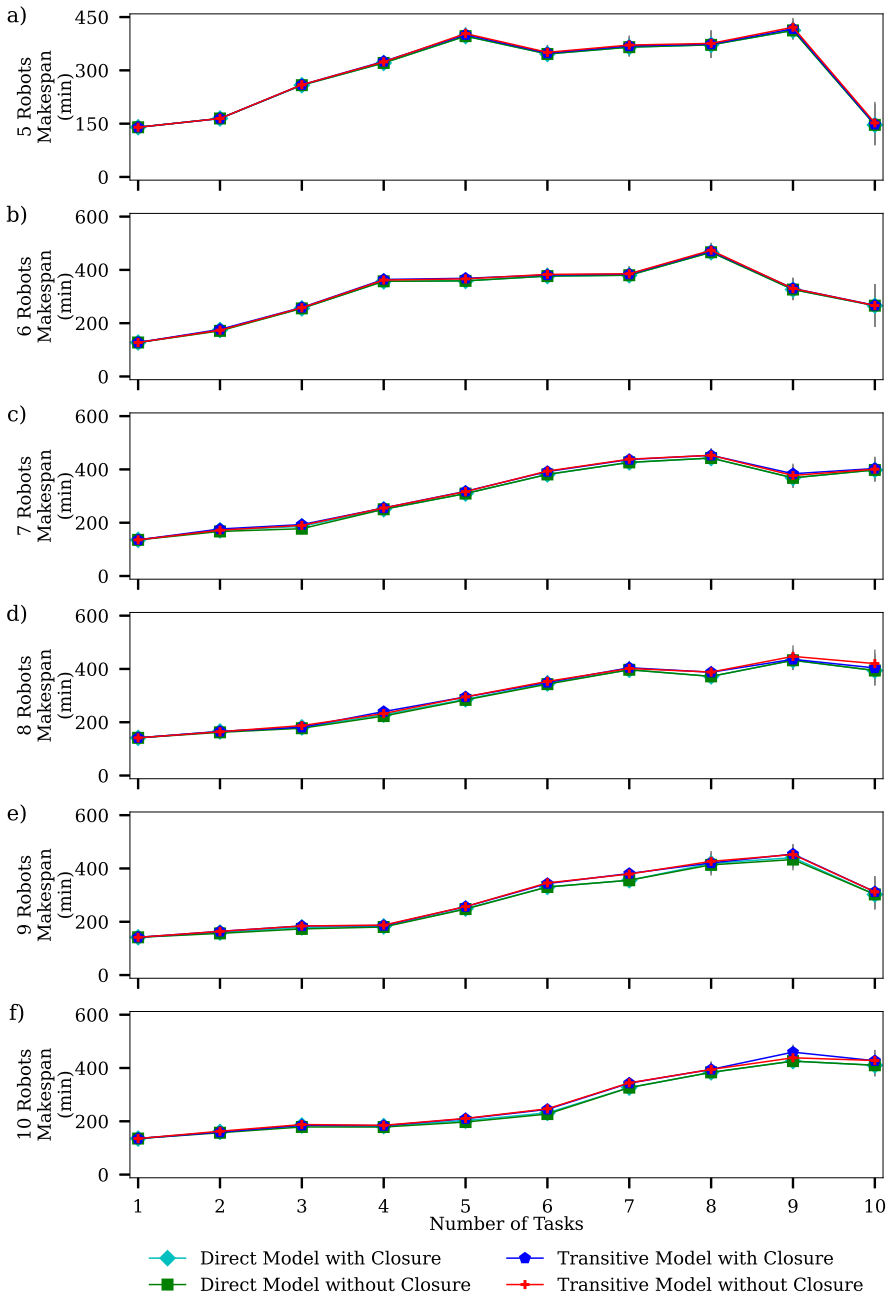


Figure C.44: First Response Domain STA makespan (min). The maximum makespan was 450, 600, 600, 600, 600, and 600 min for 5, 6, 7, 8, 9, and 10 robots, respectivelly.

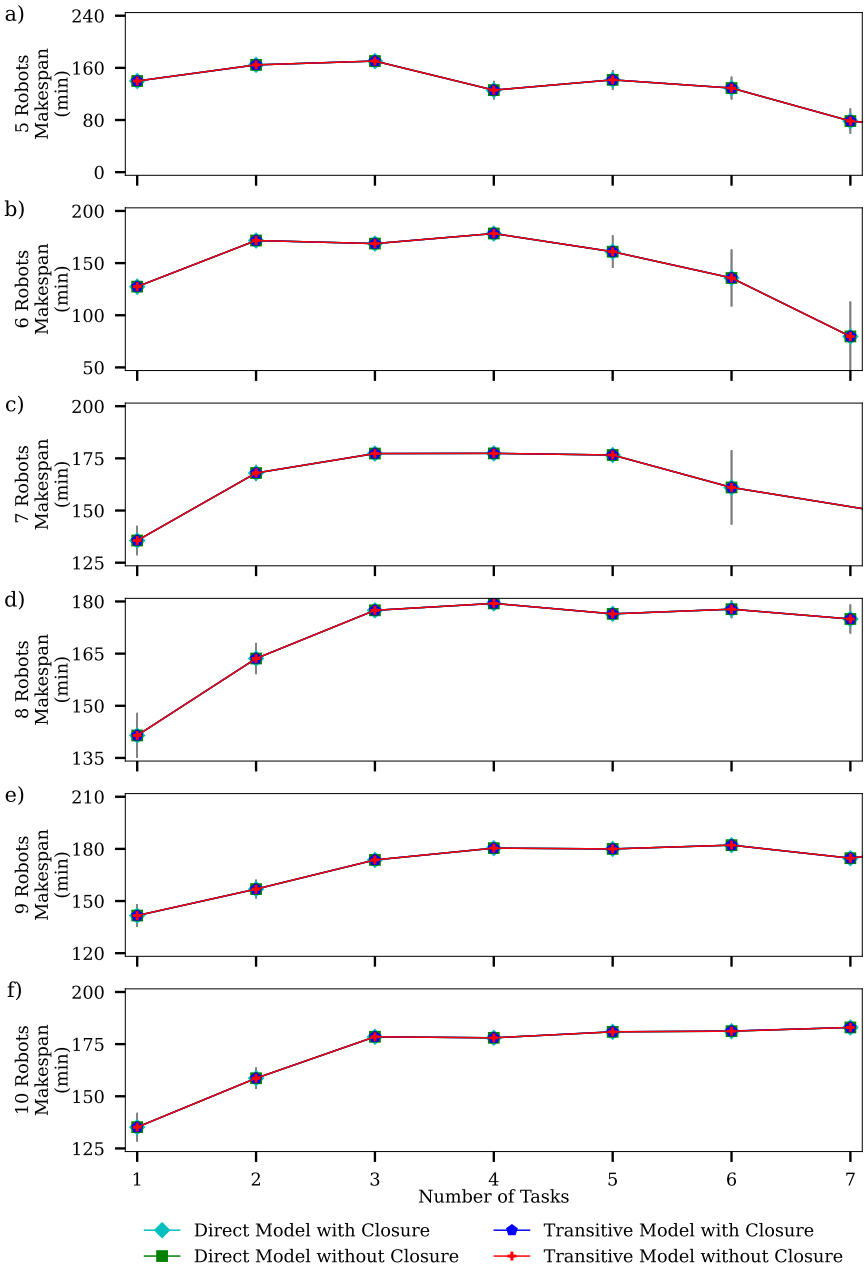


Figure C.45: First Response Domain TCRA* ($\epsilon = 0$) makespan (min). The maximum makespan was 240, 200, 200, 180, 210, and 200 min for 5, 6, 7, 8, 9, and 10 robots, respectivelly.

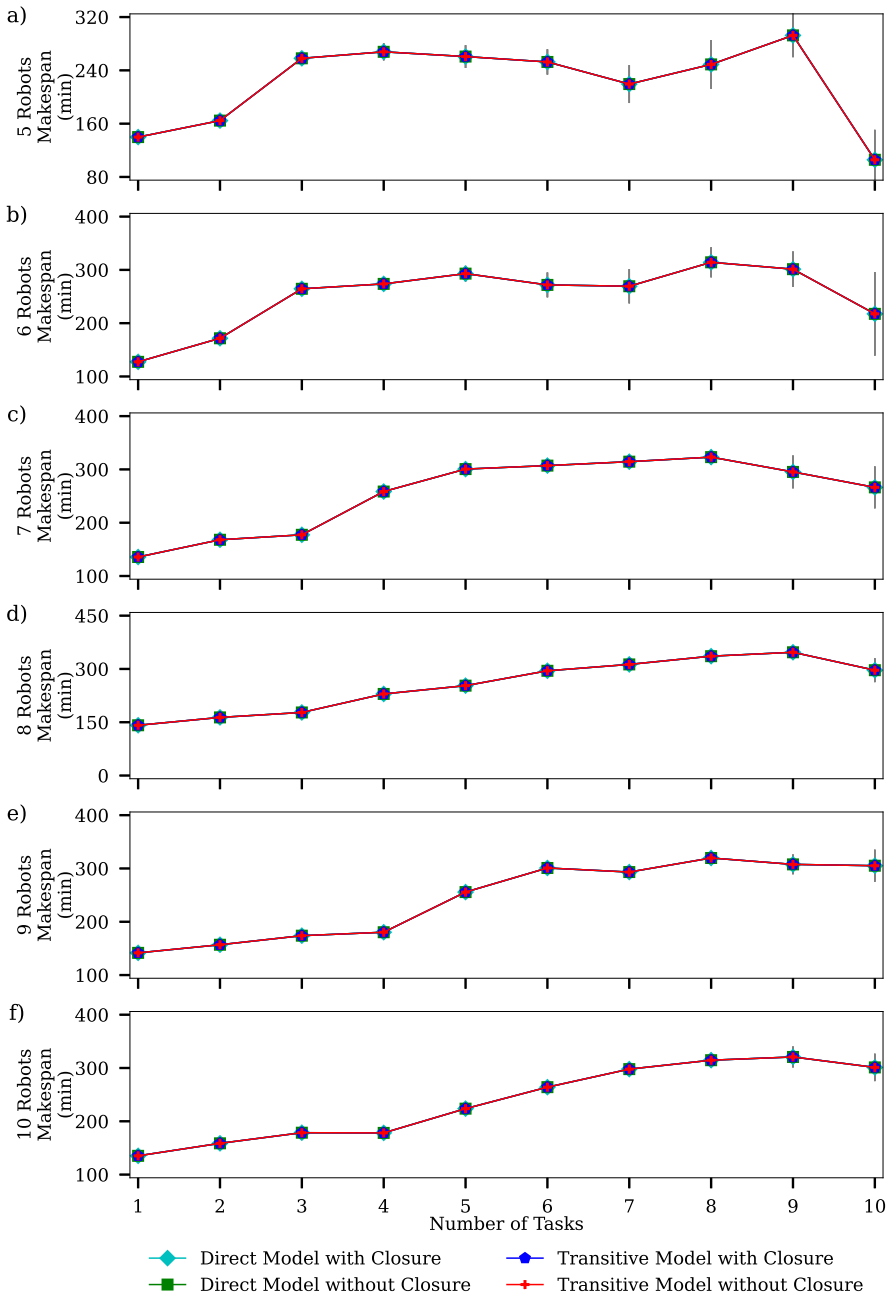


Figure C.46: First Response Domain TCRA* ($\epsilon = 1$) makespan (min). The maximum makespan was 320, 400, 400, 450, 400, and 400 min for 5, 6, 7, 8, 9, and 10 robots, respectivelly.

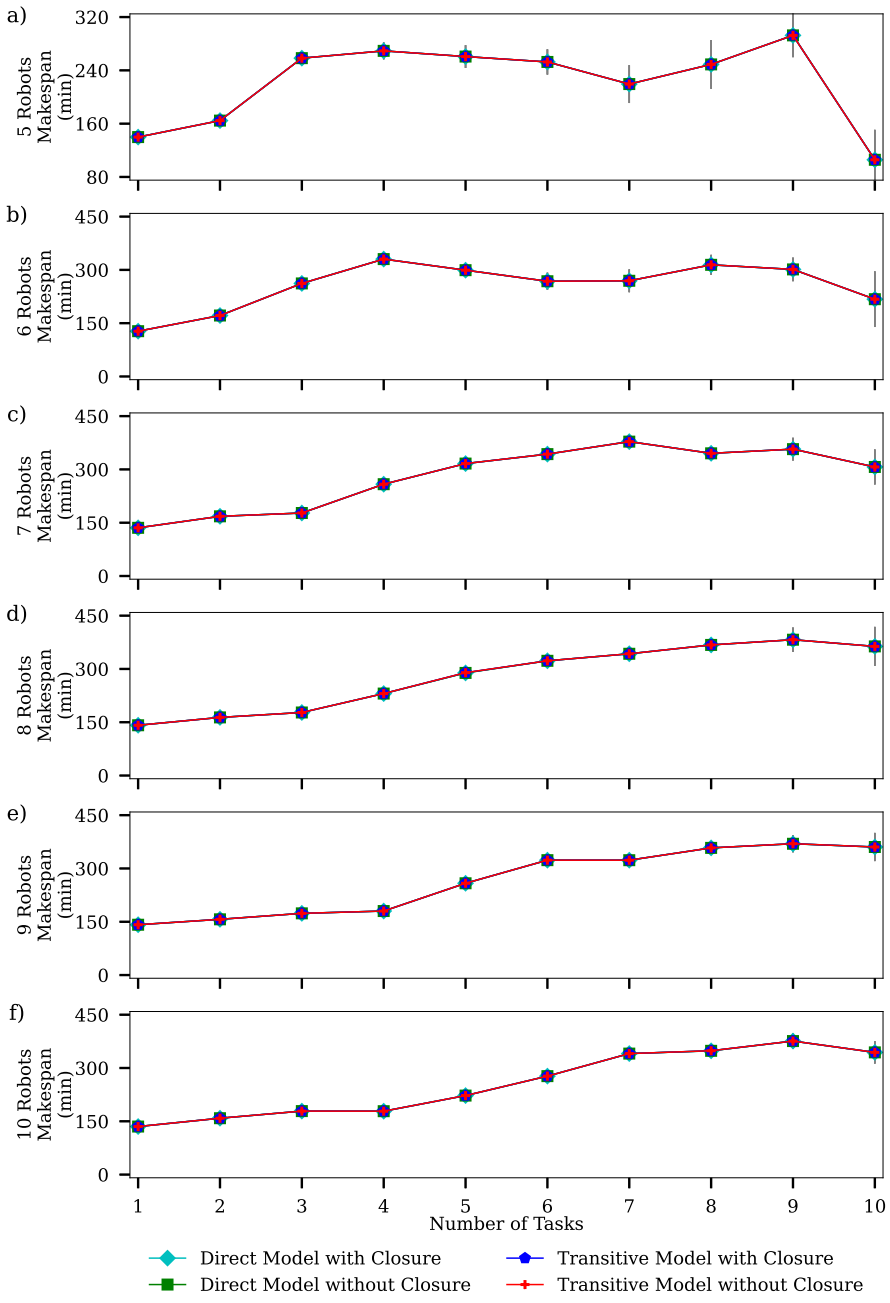


Figure C.47: First Response Domain TCRA* ($\epsilon = 10$) makespan (min). The maximum makespan was 320, 450, 450, 450, 450, and 450 min for 5, 6, 7, 8, 9, and 10 robots, respectivelly.

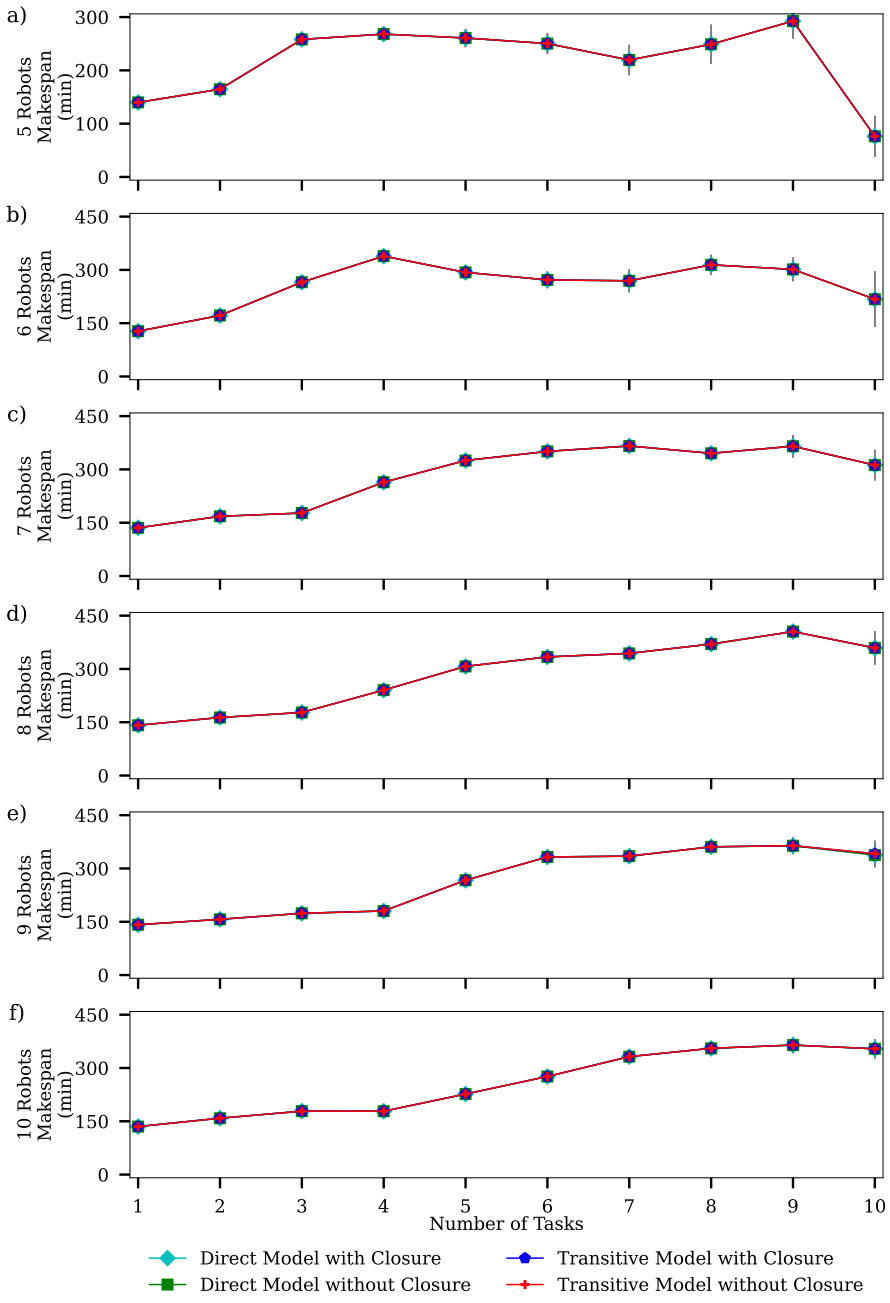


Figure C.48: First Response Domain TCRA* ($\epsilon = 100$) makespan (min). The maximum makespan was 300, 450, 450, 450, 450, and 450 min for 5, 6, 7, 8, 9, and 10 robots, respectivelly.