OPTIMAL GLOBAL PLANNING FOR COGNITIVE FACTORIES WITH MULTIPLE TEAMS OF HETEROGENEOUS ROBOTS

by

Zeynep Gözen Sarıbatur

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science

Sabancı University
August 2014

OPTIMAL GLOBAL PLANNING FOR COGNITIVE FACTORIES WITH MULTIPLE TEAMS OF HETEROGENEOUS ROBOTS

Approved by:

Assoc. Prof. Dr. Esra Erdem
(Thesis Supervisor) .....

Assoc. Prof. Dr. Volkan Patoğlu
(Thesis Supervisor) .....

Prof. Dr. Ali Rana Atılgan .....

Asst. Prof. Dr. Hüsnü Yenigün .....

Asst. Prof. Dr. Peter Schüller .....

Date of approval: 06.08.2014

OPTIMAL GLOBAL PLANNING FOR COGNITIVE FACTORIES WITH MULTIPLE TEAMS OF HETEROGENEOUS ROBOTS

Zeynep Gözen Sarıbatur

Computer Science and Engineering, Master of Science, 2014

Thesis Supervisors: Esra Erdem, Volkan Patoğlu

Keywords: cognitive robotics, decoupled planning, answer set programming

Abstract

We consider a cognitive factory domain with multiple teams of heterogeneous robots where the goal is for all teams to complete their tasks as soon as possible to achieve overall shortest delivery time for a given manufacturing order. Should the need arise, teams help each other by lending robots. This domain is challenging in the following ways: different capabilities of heterogeneous robots need to be considered in the model; discrete symbolic representation and reasoning need to be integrated with continuous external computations to find feasible plans (e.g., to avoid collisions); a coordination of the teams should be found for an optimal feasible global plan (with minimum makespan); in case of an encountered discrepancy/failure during plan execution, if the discrepancy/failure prevents the execution of the rest of the plan, then finding a diagnosis for the discrepancy/failure and recovering from the plan failure is required to achieve the goals.

We introduce a formal planning, execution and monitoring framework to address these challenges, by utilizing logic-based formalisms that allow us to embed external computations in continuous spaces, and the relevant state-of-the-art automated reasoners. To find a global plan with minimum makespan, we propose a semi-distributed approach that utilizes a mediator subject to the condition that the teams and the mediator do not know about each other's workspaces or tasks. According to this approach, 1) the mediator gathers sufficient information from the teams about when they can/need lend/borrow how many and what kind of robots, 2) based on this information, the mediator computes an optimal coordination of the teams and informs each team about this coordination, 3) each team computes its own optimal local plan to achieve its own tasks taking into account the information conveyed by the mediator as well as external computations to avoid collisions, 4) these optimal local plans are merged into an optimal global plan. For the

first and the third stages, we utilize methods and tools of hybrid reasoning. For the second stage, we formulate the problem of finding an optimal coordination of teams that can help each other, prove its intractability, and describe how to solve this problem using existing automated reasoners. For the last stage, we prove the optimality of the global plan. For execution and monitoring of an optimal global plan, we introduce a formal framework that provides methods to diagnose failures due to broken robots, and to handle changes in manufacturing orders and in workspaces. We illustrate the applicability of our approaches on various scenarios of cognitive factories with dynamic simulations and physical implementation.

BİLİŞSEL FABRİKALARDA BİRDEN FAZLA FARKLI YAPIDA ROBOT TAKIMI İÇİN ENİYİLEŞTİRİLMİŞ PLAN HESAPLANMASI

Zeynep Gözen Sarıbatur

Bilgisayar Bilimi ve Mühendisliği, Yüksek Lisans, 2014

Tez Danışmanları: Esra Erdem, Volkan Patoğlu

Anahtar Kelimeler: bilişsel robotik, ayrıştırılabilir plan hesaplaması, çözüm kümesi programlama

Özet

Birden fazla farklı yapıda robot takımlarından oluşan, verilmiş bir üretim siparişini en yakın teslim tarihine yetiştirmenin hedeflendiği bir bilişsel fabrika ortamını ele alıyoruz. İhtiyaç durumunda takımlar birbirine robot ödünç vererek yardım edebilirler. Söz konusu ortam şu zorlukları barındırmaktadır: Farklı yapıdaki robotların farklı kabiliyetlerinin modelde dikkate alınması gerekmektedir; uygulanabilir planların elde edilebilmesi için (örn., çarpışmalardan sakınmak amacıyla) kesikli simgesel gösterimin sürekli harici hesaplamalarla birleştirilmesi gerekmektedir; eniyileştirilmiş uygulanabilir geniş çaplı (en kısa üretim süreli) bir plan için takımların bir koordinasyonu bulunmalıdır; plan icrası sırasında bir uyuşmazlık ile karşılaşılması halinde, hedefe ulaşabilmek için, oluşan aksaklıklar eğer geriye kalan planın icrasını engelliyorsa, onları teşhis edebilmek ve uygun iyileşmeyi yapabilmek gerekmektedir.

Bu zorlukların üstesinden gelmek amacıyla, sürekli uzayda yapılan harici hesaplamaların gömülebildiği mantık tabanlı biçimselcilikler ve otomatik akıl yürütücülerin kullanıldığı bir biçimsel planlama, icra ve denetleme sistemini öne sürüyoruz. En kısa üretim süreli eniyileştirilmiş geniş çaplı planı bulmak için bir aracının kullanıldığı, takımların ve aracının birbirlerinin çalışma alanları ya da görevleri hakkında bilgi sahibi olmadıkları, yarı-dağıtık bir yöntem öneriyoruz. Bu yönteme göre, 1) aracı, takımların kaç adet hangi yapıdaki robotu ne zaman ödünç verebilecekleri/alabilecekleri bilgisini toplar, 2) bu bilgilere göre, aracı, eniyileştirilmiş bir koordinasyon hesaplar ve her takımı bu koordinasyon konusunda bilgilendirir, 3) aracı tarafından verilen bilgiyi ve çarpışmalardan sakınmak için yapılan harici hesaplamaları kullanarak her takım eniyileştirilmiş yerel planını

hesaplar, 4) eniyileştirilmiş yerel planlar eniyileştirilmiş ortak plan elde etmek için birleştirilir. Birinci ve üçüncü aşamalarda, hibrid akıl yürütme yöntemlerini ve araçlarını kullanıyoruz. İkinci aşamada, takımlar için eniyileştirilmiş koordinasyon bulma problemini tanımlıyoruz, zorluğunu kanıtlıyoruz, ve problemin mevcut otomatik akıl yürütücülerle nasıl çözülebileceğini gösteriyoruz. Son aşama için, geniş çaplı planın eniyileştirilmiş olduğunu kanıtlıyoruz. Eniyileştirilmiş geniş çaplı planın icrası ve denetlenmesi için, bozuk robotlar nedeniyle oluşan başarısızlıkları teşhis edebilmeyi sağlayan, ve üretim siparişinde ve çalışma alanlarında oluşabilecek değişikliklerle başa çıkabilen bir biçimsel sistem tanıtıyoruz. Yaklaşımlarımızın uygulanabilirliğini bilişsel fabrikalar üzerinde çeşitli senaryolarla yaptığımız simülasyon ve fiziksel uygulamalar aracılığıyla gösteriyoruz.

ACKNOWLEDGEMENTS

First of all, I want to thank my thesis supervisors Esra Erdem and Volkan Patoğlu for their invaluable guidance and support throughout this research. Thanks to their encouragement and determination, I had the opportunity to attend various events and present our work to numerous people, which have contributed to my skills and were pleasing moments.

I would also like to express my regards to the jury members, Prof. Ali Rana Atılgan, Asst. Prof. Hüsnü Yenigün, and Asst. Prof. Peter Schüller for their time and feedback, and letting my defense be an enjoyable moment.

I want to thank Peter Schüller separately as well, for his guidance and for giving me optimistic motivational speeches whenever I needed them during these years. I also thank Giray Havur for showing me a more humorous way to get through the difficult times, and for the laughters that made the sleepless nights bearable. I also want to thank my friends and colleagues, including Ezgi Karakaş, Zeynep Doğmuş, Ozan Tokatlı, Ahmetcan Erdoğan, Erdi Aker and Gökay Çoruhlu for their help and friendship throughout my time in Sabancı University. I was pleased to have your good company during these times. I also have to thank the cat who kept coming back to the office for a good sleep, for accompanying me in her own way.

I am also thankful to my friends whom I couldn't spend time with as much as I wanted in these last two years, but have been a part of my life during my journey to this point.

My special thanks are to Erdem Yaman for enriching my life and for always reminding me that there is more to life than work. He has been my constant source of love, support and serenity throughout these years, and was always there to lift my spirits high.

Last but not the least, I am deeply grateful to my mother Ayşe Gözen for being my inspiration and for her unconditional love, support and patience. She has cherished with me every great moment and never stopped believing in me.

This research was partially supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) under grants 111E116 and 113M422.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

Chapter 1

Introduction

Multiple teams of robots with heterogeneous capabilities are commonly employed to complete a task in a collaborative fashion in many application domains, ranging from search and rescue operations to exploration/surveillance missions, service robotics to cognitive factories. In this thesis, we focus on cognitive factories with multiple teams of heterogeneous robots.

Cognitive factory concept [71, 20] is a novel paradigm proposed to enhance productivity and ensure *economic sustainability and growth* in the manufacturing sector. Aimed towards highly flexible and typically small to medium size manufacturing plants, these factories are equipped with multiple teams of heterogeneous manufacturing tools, such as dexterous mobile manipulators. Since these factories are endowed with high-level reasoning capabilities, they can rapidly respond to changing customer needs and customization requests, and produce a large variety of customized products even in low quantities. Consequently, cognitive factories provide an ideal compromise between the flexibility of human workshops with cost-effectiveness of mass production systems.

1.1 Challenges

In the context of cognitive factories, we address three key challenges of such domains including teams of heterogeneous robots:

- **Hybrid reasoning for each team: Combining discrete task planning with continuous feasibility checks and perception.** In a cognitive factory, each team of robots needs to complete some tasks to achieve their manufacturing goals. These tasks require execution of various robotic actions, such as pick, place, move, carry, charge, etc. in some order as well as helping each other. For that, each team needs to find a collaborative task plan — a sequence of robotic actions. On the other hand, cognitive factories consist of static and dynamic obstacles in the environment. If

these obstacles are not considered, computed task plans may contain robots passing through each other, or touching an obstacle while going from one place to the other. Finding a feasible plan for the teams is necessary in order to avoid collisions that may prevent the execution of the desired plans. For that, an integration between continuous feasibility checks and discrete task planning should be established.

- **Finding an optimal global plan for multiple teams.** An optimal global plan for the teams in a cognitive factory minimizes the delivery lead time for a customized order. This leads to a more cost-effective process by reducing contribution of factory overhead per order, preserves energy resources and decreases negative environmental impacts by efficient use of facility infrastructure. For such an optimal global plan, the teams can help each other by lending/borrowing robots. This requires an optimal coordination of the robot transfers between teams.

- **Intelligent execution monitoring.** For the fault-awareness and reliability of a cognitive factory with multiple robots, it is essential that the robots have the capability of identifying discrepancies between the expected states and the observed states, check whether these discrepancies would lead to a plan failure, diagnose possible causes of relevant discrepancies, and find plans to reach their goals. For that, a general formal framework is needed for plan execution and monitoring that would utilize high-level reasoning.

1.2 Our Approach

In this thesis, we propose to use state-of-the-art automated reasoners (i) to endow each heterogeneous robot team with high-level reasoning capabilities in the style of cognitive robotics [48], such that each team becomes capable of planning their own actions; and (ii) to coordinate robot exchanges among the teams to ensure an optimal global plan. We propose to utilize, in particular, the knowledge representation and reasoning formalisms, the action description language $\mathcal{C}+$ [38] and Answer Set Programming (ASP) [49, 5, 34, 50], and relevant automated reasoners, such as the propositional satisfiability (SAT) solver MANYSAT [41] and ASP solver CLASP [33]. These formalisms are highly expressive which allows us to express concurrency and defaults. They support special constructs, called external atoms, that allow access to external computations. They also have efficient reasoners that are continuously being improved.

- For hybrid reasoning, we emphasize several core characteristics of cognitive factories, such as existence of heterogeneous robots with varying capabilities, ability of robots to execute concurrent actions, existence of complex (state/temporal) constraints/goal conditions, and provide a computational framework to find feasible

and optimal local plans for each team. The proposed method is based on earlier works on hybrid planning [19, 22, 29] that utilize the expressive logic-based formalisms and reasoners mentioned above, but in different domains, such as robotic manipulation and service robotics.

In particular, by combining discrete task planning with continuous feasibility checks and perception, we address existence of static/dynamic obstacles in the domain. For performing feasibility checks, we utilize pre-computation approach to embed information about static obstacles perceived by a Kinect RGB-D camera into the domain description, while we rely on guided replanning to account for possible robot-robot collisions [22]. Furthermore, we extend the domain to model heterogeneity of robots and conduct various optimizations on local plans. In particular, in addition to finding a local plan with minimum makespan, we further optimize the total cost of this plan by considering several other objectives relevant in cognitive factories: to minimize the number of robotic actions or to ensure that actions in a team are executed as early as possible or to minimize fuel consumption.

- For finding an optimal global plan (with minimum makespan) for multiple teams, we advocate the use of a semi-distributed approach to protect privacy of workspaces and to reduce message-passing and computational effort. Privacy is a concern in micro manufacturing plants that specialize on prototyping pre-release products, while reduction of communication among teams may be preferable when the means of communication is lossy or costly. Furthermore, a semi-distributed approach is advantageous in that it reduces the size of the global domain into manageable pieces, and provides a solution methodology that can utilize parallelization of computations.

  As in [20], our approach utilizes a mediator subject to the condition that the teams and the mediator do not know about each other's workspaces or tasks, and capitalizes on the fact that each team and the mediator is capable of hybrid reasoning. Our approach is not constrained with the tight restrictions of [20] where homogeneous robots are considered and at most one robot transfer is permitted in a global plan. Also our method utilizes a wider variety of automated reasoners in a more general setting of cognitive factories.

  According to our semi-distributed approach, 1) the mediator gathers sufficient information from the teams about when they can/need lend/borrow how many and what kind of robots, 2) based on this information, the mediator computes an optimal coordination of the teams and informs each team about this coordination, 3) each team computes its own optimal local plan to achieve its own tasks taking into account the information conveyed by the mediator as well as external computations to avoid collisions, 4) these optimal local plans are merged into an optimal global

plan. The first and the third steps utilize methods and tools of hybrid reasoning as decribed above. For the second step, we formulate the problem of finding an optimal coordination of teams that can help each other, prove its intractability, and use automated reasoners to solve it. For the last stage, we prove the optimality of the global plan.

- For intelligent execution and monitoring, we introduce an execution monitoring algorithm to address possible plan failures during the execution of the global plan, such as broken robots/parts preventing execution of some actions, new obstacles in the environment or new order arrival to the teams. Our algorithm allows diagnostic reasoning as in [20] when sensory information is not sufficient to identify the cause of a failure (i.e., which robot/part is broken), but in a more general cognitive factory setting with heterogenous robots and allowing different formalisms and reasoners. Once a diagnosis is found, our execution and monitoring algorithm finds a new plan, with the possibility of repairing the broken robots/parts if needed.

1.3 Contributions of the Thesis

Our contributions can be summarized as follows:

- We have introduced cognitive toy factory scenarios with heterogeneous robots, that necessitate teams to help each other and that requires integration of feasibility checks into task planning. These scenarios are important for better understanding the challenges of cognitive factories and useful as benchmarks for future studies.
- We have represented the cognitive toy factory domain in the formalisms of $\mathcal{C}+$ and ASP, by embedding feasibility checks like robot-obstacle collisions that utilize state-of-the-art technologies such as Open Dynamics Engine (ODE).
- We have presented methods for computing optimal hybrid plans with these formulations and state-of-the-art technologies, with respect to different optimization measures, such as minimum makespan and total cost of actions.
- We have introduced a novel semi-distributed method to compute an optimal global plan for multiple teams of heterogeneous robots.
- We defined the problem of determining a coordination between multiple teams of heterogeneous robots for an optimal global plan. We analyzed the computational complexity of the optimal coordination problem, and proved that it is NP-hard.
- We have introduced a method to solve the coordination problem using the state-of-the-art ASP solvers. We have performed experiments to show the scalability of our method.

- We have performed experiments to analyze the scalability of our method for computing optimal local/global plans, over cognitive toy factory settings with multiple teams of heterogeneous robots.

- We have introduced a general execution and monitoring algorithm that can handle surprises like new manufacturing orders, changes in workspaces, and broken robots/parts that cannot be identified by sensory information but requires deeper reasoning. The algorithm utilizes different methods to handle each one of these discrepancies.

- We have showed the applicability of our approach with various cognitive toy factory scenarios, both with dynamic simulation using OpenRave and with a physical implementation, using Kuka youBots and Lego NXTs.

1.4 Thesis Outline

In Chapter 2 we describe a cognitive toy factory domain with multiple teams of heterogeneous robots. We continue with some preliminaries on logic-based knowledge representation formalisms used in our studies. In Chapter 3, we describe the action description language $\mathcal{C}+$, and how the cognitive toy factory domain can be represented in this formalism. We introduce an action query language to facilitate high-level reasoning, such as planning, by means of queries over this domain descrription. We illustrate how the cognitive factory domain description and the queries can be used with the state-of-the-art automated reasoners. In Chapter 4, we describe Answer Set Programming (ASP) and show details on how to use ASP for high-level reasoning. In Chapter 5, we present our method to find an optimal global plan for multiple teams of heterogeneous robots and prove its correctness. We define the problem of determining a coordination, prove the intractability of the problem, formulate the problem using ASP and show scalability of this method via the experimental evaluation over cognitive factory scenarios. Examples are given for a better understanding of these methods. The applicability of our method for optimal global planning is illustrated with a dynamic simulation and a physical implementation; the scenarios are shown along with relevant snapshots. In Chapter 6, we analyze the scalability of our method. Chapter 7 presents our execution monitoring algorithm, and demostrates its applicability. Related work is discussed in Chapter 8. We conclude in Chapter 9 by summarizing our results. The URLs for the supplementary materials (videos of dynamic simulations and physical implementations) are provided in the thesis.

Chapter 2

Cognitive Factories with Multiple Teams of Heterogeneous Robots

We consider cognitive factories with multiple teams of heterogeneous robots. Cognitive factory concept [4, 71, 72, 20], Figure 2.1, is a novel paradigm proposed to enhance productivity and ensure economic sustainability and growth in the manufacturing sector. Aimed towards highly flexible and typically small to medium size manufacturing plants, these factories are equipped with multiple teams of heterogeneous manufacturing tools, such as dexterous mobile manipulators. They endow manufacturing system with high-level reasoning capabilities in the style of cognitive robotics, such that these systems become capable of planning their own actions. They can rapidly respond to changing customer needs and customization requests, and produce a large variety of customized products even in low quantities. By utilizing sophisticated planning and decision-making algorithms, cognitive factories can efficiently allocate their resources for daily/weekly/-monthly work load and ensure production of variant-rich products to guarantee pressing delivery deadlines. Consequently, cognitive factories provide an ideal compromise between the flexibility of human workshops with cost-effectiveness of mass production systems.

We consider multiple teams of robots with different capabilities. Each team is given a feasible task to complete in its workspace on its own, and where teams are allowed to transfer robots between each other. The goal is to find an optimal overall plan for all teams so that all tasks can be completed as soon as possible

As an example of a cognitive factory, we consider a cognitive toy factory with two teams of robots, where each team is located in a separate workspaces collectively working toward completion of an assigned task. There are two types of processes that are conducted in these workspaces: manufacturing process and painting process. In particular, Team 1 manufactures nutcracker toy soldiers through the sequential stages of cutting, carving and assembling, while Team 2 processes them by going through stages of painting

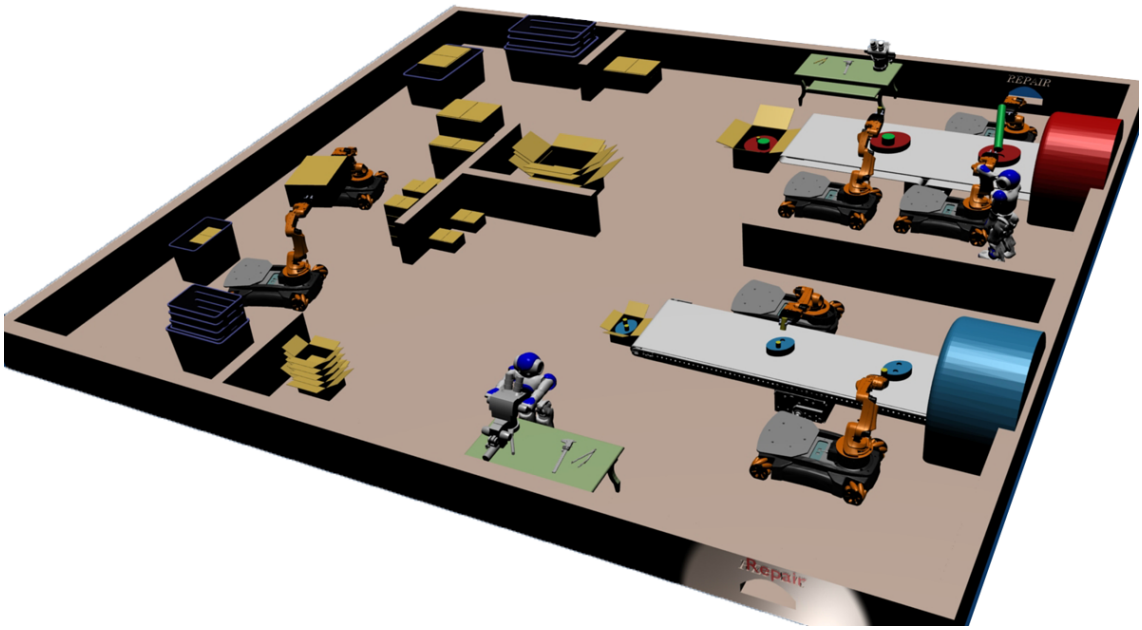


Figure 2.1: A cognitive factory

in black, painting in color, and stamping. Each workspace is depicted as a grid, as shown in Figure 2.2, contains an assembly line along the north wall to move the toys and a pit stop area where the worker robots can change their end-effectors. Each workspace also includes static obstacles.

The teams are heterogeneous, as each team is composed of three types robots with different capabilities. Worker robots operate on toys, they can configure themselves for different stages of processes, and they can be exchanged between teams; charger robots maintain the batteries of workers and monitor team's plan, and cannot be exchanged between teams. Worker robots are further categorized into two, based on their liquid resistance. In particular, wet (liquid resistant) robots can perform every stage of the processes involved in manufacturing and painting of the toys, while dry (non-liquid resistant) robots cannot be employed in painting and cutting stages, since there processes involve liquids. All robots are holonomic and can move from any grid to another one following straight paths.

In this cognitive factory, teams can help each other: at any step, a team can lend several of its worker robots through their pit stop such that after a transportation delay the worker robots show up in the pit stop of a borrowing team.

We have tested this cognitive toy factory scenario with both dynamic simulations using OPENRAVE [12], and with a physical implementation utilizing KuKa youBots and Lego NXT robots controlled over Robot Operating System (ROS). Snapshot of a state from the physical implementation by two teams utilizing Kuka youBots and Lego NXT robots can be seen in Figure 2.3, while a snapshot of a state from a dynamic simulation by two teams utilizing Kuka youBots is shown in Figure 2.4. In these snapshots we can see

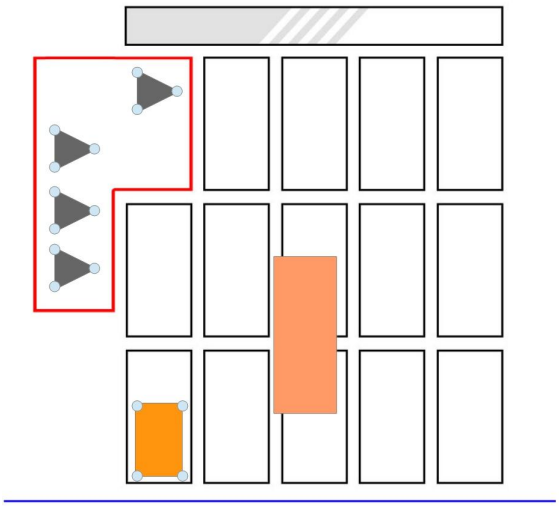


Figure 2.2: Workspace

that the worker robots are aligned near the assembly line to work on the boxes/toys, while the charger robots are approached to the worker robots to charge them when needed.

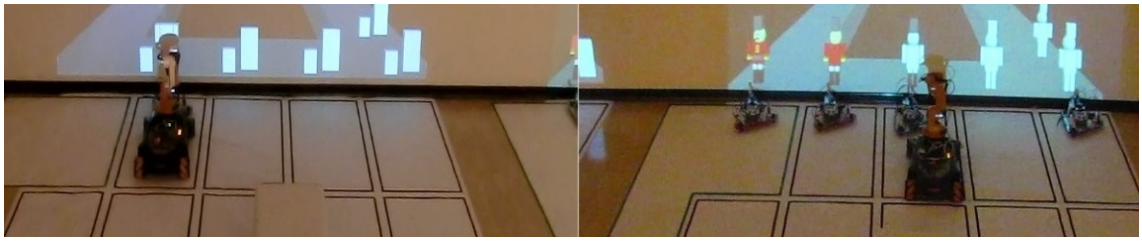


Figure 2.3: Snapshot of a state from the physical implementation

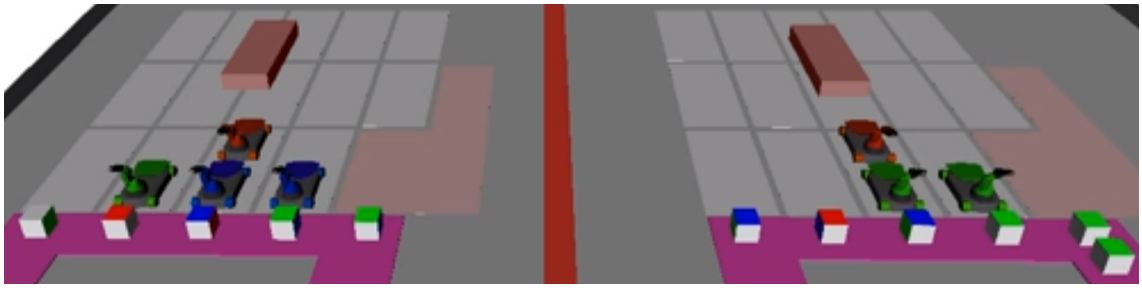


Figure 2.4: Snapshot of a state from a dynamic simulation

Chapter 3

Representation and Reasoning about a Dynamic Domain

In this thesis, we utilize nonmonotonic logic-based approaches to represent and reason about dynamic domains, such as action languages [35] and Answer Set Programming (ASP) [49, 5, 34, 50]. In this chapter, we will briefly describe preliminaries about action languages and their uses in conjunction with the relevant automated reasoners; ASP will be elaborated in the next chapter.

Action languages [35] are formal models of parts of natural language that are used for expressing dynamic systems. An action language consists of two parts: an action description language to describe the dynamic domain, and an action query language to facilitate reasoning tasks over this description. In this thesis, we use the action description language $\mathcal{C}+$ [38]; and a query language, which is a variation of $\mathcal{Q}$. These two kinds of action languages are explained in the following sections.

3.1 Action Description Language $\mathcal{C}+$

Let us briefly describe the action description language $\mathcal{C}+$, as in [38].

A *(multi-valued propositional) signature* is a set σ of symbols called *constants* partitioned into *fluent constants*, σ^{fl} and *action constants*, σ^{act}, along with a nonempty finite set *Dom(c)* of symbols, assigned to each constant *c*. An *atom* of a signature σ is an expression of the form $c = v$ where $c \in \sigma$ and $v \in Dom(c)$. A *formula* of σ is a propositional combination of atoms. A *causal rule* is an expression of the form $F \Leftarrow G$, where F and G are formulas of σ. A *causal theory* is a set of causal rules.

An *interpretation* of σ is a function that maps every element of σ to an element of its domain. An interpretation I *satisfies* an atom $c = v$ ($I \models c = v$) if $I(c) = v$. A *(world) state* is an interpretation of σ^{fl}. An *action* is an interpretation of σ^{act}.

A *model* of a set X of formulas is an interpretation that satisfies all formulas in X.

If X has a model it is said to be *consistent*. If every model of X satisfies a formula F then we say that X *entails* F and write $X \models F$.

A *Boolean* constant is one whose domain is the set $\{\mathbf{f}, \mathbf{t}\}$ of truth values.

Syntax In the action description language $\mathcal{C}+$ fluents are of two types: simple fluents and statically determined fluents. The values of simple fluents (resp. statically determined fluents) may directly (resp. indirectly) change through actions.

Actions and change are described in $\mathcal{C}+$ using three kinds of expressions:

(i) *static laws* of the form

$$\textbf{caused } F \textbf{ if } G \tag{3.1}$$

where F and G are *fluent formulas*, i.e.,formulas that consist of fluent constants only,

(ii) *action dynamic laws* of the form (3.1) in which F is an *action formula*, i.e., a formula that contains at least one action constant and no fluent constants, and G is a formula, and

(iii) *fluent dynamic laws* of the form

$$\textbf{caused } F \textbf{ if } G \textbf{ after } H \tag{3.2}$$

where F and G are fluent formulas and H is a formula.

In the causal laws, F is called the head. In (3.1) and (3.2) the part **if** G can be dropped if G is $True$.

A static law where F is $False$ is called a *state constraint*; a fluent dynamic law where F is $False$ is called a *transition constraint*.

A *causal law* is a static law, or an action dynamic law, or a fluent dynamic law. An *action description* is a set of causal laws.

In our descriptions of cognitive factories

- actions are Boolean constants and
- action dynamic laws are used for exogeneity of actions only.

Semantics Let D be an action description with a signature σ and $Dom(\sigma)$. The *transition diagram described* by D, denoted $T(D)$, is characterized by a set S of states and a set R of transitions, which are defined as follows:

(i) S is the set of all interpretations s of σ^{fl} such that, for every static law (3.1) in D, s satisfies F if s satisfies G, $s \models G \supset F$,

(ii) with a function $V : \sigma^{fl} \times S \to Dom(\sigma^{fl})$, $V(P, s) = s(P)$ is the value of P in s,

(iii) $R \subseteq S \times \{\mathbf{f}, \mathbf{t}\}^{\sigma^{act}} \times S$ is the set of all triples $\langle s, A, s' \rangle$ such that s' is the only interpretation of σ^{fl} which satisfies the heads of all

- static laws (3.1) in D for which s' satisfies G, and
- fluent dynamic laws (3.2) in D for which s' satisfies G and $s \cup A$ satisfies H.

The laws included in (iii) above are those that are *applicable* to a transition from s to s' caused by executing A.

The static causal laws make sure that s is a state, and handles the ramifications and the qualifications of A; whereas the dynamic causal laws handle the preconditions and the direct effects of A.

3.1.1 Abbreviations

In action descriptions, we can use an expression of the form

$$a \textbf{ causes } F \textbf{ if } G \tag{3.3}$$

to describe the direct effects of actions, which abbreviates the fluent dynamic law

$$\textbf{caused } F \textbf{ if } True \textbf{ after } G \wedge a. \tag{3.4}$$

We can express that F is a precondition of a by the expression

$$\textbf{nonexecutable } a \textbf{ if } \neg F \tag{3.5}$$

which stands for the fluent dynamic law

$$\textbf{caused } False \textbf{ if } True \textbf{ after } \neg F \wedge a. \tag{3.6}$$

Similarly, we can express that F holds by default by the expression

$$\textbf{default } F$$

that abbreviates the static law

$$\textbf{caused } F \textbf{ if } F.$$

We can use the expression of the form

$$\textbf{inertial } F$$

which stands for the dynamic law

caused F **if** F **after** F

to express the commonsense law of inertia: F does not change unless it is effected by an action.

In almost all action domains, we express that there is no cause for the occurrence of an action a, by the abbreviation

exogeneous a

which stands for the dynamic laws

caused a **if** a.

3.1.2 Representation of the Cognitive Factory Domain in $\mathcal{C}+$

We use $\mathcal{C}+$ to describe the cognitive toy factory domain as follows.

Fluents and actions We view the workspace as a grid. We consider simple fluent constants

- $xpos(r) = x$ and $ypos(r) = y$ ("robot r is at (x, y)")
- $battery(w) = bs$ ("battery of worker robot w has a capacity of bs units")
- $endEffector(w) = e$ ("end effector of worker robot w is for work stage e")
- $docked(c, w)$ ("charger robot c is docked to the worker robot w")
- $workDone(b) = ws$ ("box b is being processed at the stage ws")
- $wetpaint(b)$ ("box b has wetpaint")

that follow the inertia laws, and the actions

- $move(r, d)$ ("robot r moves in direction d")
- $workOn(w, b)$ ("worker robot w works on the box b")
- $charge(c)$ ("charger robot c does the charging action")

that are exogeneous.

To describe the actions we need to describe their direct effects and preconditions.

Direct effects of actions We describe the direct effects of the actions above by causal laws of the form (3.3). For instance, the following causal law expresses that a direct effect of $workOn(w, b)$ action is an increase in the work stage of box b by one:

$$workOn(w, b) \textbf{ causes } workDone(b) = ws \textbf{ if } workDone(b) = ws{-}1$$

Another direct effect of $workOn(w, b)$ action is expressed as

$$workOn(w, b) \textbf{ causes } battery(w) = bl{-}workCons \textbf{ if } battery(w) = bl$$

for $bl \geq workCons$, which means it causes the battery of worker robot w to decrease by some units specified by $workCons$.

Similarly, we describe that the charging by a charger robot c increases the battery of the worker robot w it is docked to:

$$charge(c) \textbf{ causes } battery(w) = maxBattery \textbf{ if } docked(c, w).$$

Preconditions of actions We describe the preconditions of actions by causal laws of the form (3.5). For instance, we describe that a worker robot w cannot work on the box b if its end effector is prepared for the work stage ws but the box b is not ready to be processed, i.e., box b is not at the work stage $ws{-}1$, by the law

$$\textbf{nonexecutable } workOn(w, b) \textbf{ if } endEffector(w) = ws1 \wedge workDone(b) \neq ws1{-}1.$$

To describe that a worker robot w cannot work on a box b if the robot has insufficient battery, by the law

$$\textbf{nonexecutable } workOn(w, b) \textbf{ if } battery(w) = bl$$

where $bl{<}workCons$.

Similarly, a charger robot c cannot do the charging action if it is not docked to any robot:

$$\textbf{nonexecutable } charge(c) \textbf{ if } \neg docked(c).$$

We can specify different capabilities of heterogeneous robots while describing the preconditions of actions that may be executed by only some types of robots. For example, to describe that a dry robot dr in a team that conducts the painting process in a cognitive toy factory cannot do the painting action, (which is the first work stage of the boxes), we have the following law:

$$\textbf{nonexecutable } workOn(dr, b) \textbf{ if } workDone(b) = 0.$$

Attributes of actions We introduce an *attribute* of $move$ action by an action constant

$$distance \text{ with domain } \{1, \ldots, maxUnit, None\}$$

to show how many units the robot is moving. The attribute takes the value $None$ if and only if the action is not executed.

We describe the direct effect of a robot r moving in direction $Right$, with the following law

$$move(r, Right) \textbf{ causes } xpos(r) = x+u \textbf{ if } distance(r, Right) = u \wedge xpos(r) = x$$

for $x+u \leq maxX$. We describe the effects of moving in other directions, $Left$, Up, $Down$, with similar laws.

We describe the direct effect on the battery of a moving worker robot by the following law:

$$move(w, d) \textbf{ causes } battery(w) = bl-u \textbf{ if } distance(r, d) = u \wedge battery(w) = bl$$

for $bl \geq u$.

We have other forms of expressions to describe the indirect effects of actions, state contraints, transition constraints and nonconcurrency constraints.

Ramifications We express that a box with wet paint dries at the next step by the following law

$$\textbf{caused } \neg wetpaint(b) \textbf{ after } wetpaint(b)$$

which is an indirect effect of $workOn$ action that causes box b to reach a workstage that contains paint.

Constraints We ensure that two worker robots do not occupy the same grid cell unless they are at the pitstop area $(pixX, pitY)$ or they are out of the workspace limits, by the state constraints

$$\begin{aligned} \textbf{caused } False \textbf{ if } & \neg(xpos(w1) = pitX \wedge ypos(w1) = pitY) \wedge \\ & xpos(w1){>}0 \wedge ypos(w1){>}0 \wedge \\ & xpos(w1) = xpos(w2) \wedge ypos(w1) = ypos(w2) \end{aligned}$$

for $w1 \neq w2$.

We ensure that a robot does not move in two reverse directions by the noncurrency constraints

nonexecutable $move(r, Right) \wedge move(r, Left)$
nonexecutable $move(r, Up) \wedge move(r, Down)$

External atoms We can define atoms whose truth values can be computed externally via some functions. External atoms take as input some parameters from the domain description (e.g., the locations of robots), to externally check some conditions and then return the boolean value of this external computation. These special constructs can be used to take care of the static obstacles in the environment. We can express that a worker w can not move in the directions $right, up$ with units $u1, u2$ if there is a collision with an obstacle by the following condition:

nonexecutable $move(w, right) \wedge move(w, up)$
if $distance(w, right) = u1 \wedge distance(w, up) = u2 \wedge$
$xpos(w) = x \wedge ypos(w) = y \wedge existsCollision(x, y, u1, u2).$

for $x + u1 \leq maxX$ and $y + u2 \leq maxY$. Here $existsCollision$ is an external function that checks whether the movement of a robot from grid cell (x, y) with units $u1, u2$ collides with any obstacles. Details can be found in Section 3.4

3.2 Action Query Language

Given an action domain description represented in $\mathcal{C}+$ as described above, we can perform reasoning tasks over it. Reasoning problems can be represented using queries in an "action query language" as described in [35].

We consider an action language, which is a variation of the action query language $\mathcal{Q}$ introduced in [35]. In this language an *atomic query* is one of the two forms, F **holds at** t or A **occurs at** t, where F is a fluent formula, A is an action formula, and t is a time step. A *query* is a propositional combination of atomic queries.

Let D be an action description and $T(D) = \langle S, V, R \rangle$ denote the transition diagram described by D, with a set S of states, a value function V mapping every fluent P to a value at each state s, and a set R of transitions. A *history* of D of length n is a sequence

$$s_0, A_0, s_1, \ldots, s_{n-1}, A_{n-1}, s_n \tag{3.7}$$

where each $\langle s_i, A_i, s_{i+1} \rangle$ $(0 \leq i < n)$ is in R. We say that a query Q of the form F **holds at** t (resp. A **occurs at** t) is *satisfied* by a history (3.7) if s_t satisfies F (resp. if A_t satisfies A). For nonatomic queries, *satisfaction* is defined by truth tables of propositional logic. We say that a query Q is *satisfied* by an action description D, if there is a history H of D that satisfies Q.

3.2.1 Planning

Suppose that F and G are fluent formulas denoting an initial state and goal conditions respectively. We can describe the problem of finding a plan of length k, with a query of the form

$$F \textbf{ holds at } 0 \wedge G \textbf{ holds at } \mathrm{k}.$$

We can also solve variations of these problems, where some intermediate states are specified or where the specified actions are not executed consecutively. This allows us to enforce, for instance, further temporal constraints in a planning problem.

For example, suppose that the action formula $giveRobot(w)$ describes that the team lends the robot w. We can express a question of the form "can you complete your task specified by the initial state F and the goal conditions G in k steps, while also lending m robots before step l?", with a query of the form

$$\begin{aligned} &F \textbf{ holds at } 0 \wedge G \textbf{ holds at } \mathrm{k} \wedge \exists T, W_1, \ldots, W_m : \\ &\quad T{<}l \wedge W_1{<}W_2{<}\ldots{<}W_m \wedge \bigwedge_{i=1}^{m} giveRobot(W_i) \textbf{ occurs at } \mathrm{T}. \end{aligned}$$

This query states that while reaching the goal state from the initial state, there should be a time step T before l, where m distinct robots are lent.

3.2.2 Prediction

Suppose that F is a fluent formula denoting an initial state. We can describe the problem of predicting the resulting state after an execution of a sequence $A_0, \ldots, A_{n-1}$ at a state F with a query of the form

$$F \textbf{ holds at } 0 \wedge \bigwedge_{i} A_i \textbf{ occurs at } \mathrm{i} \tag{3.8}$$

Prediction problems may be useful to predict the resulting states after an execution of a long (possibly partial) sequence of (possibly concurrent/nondeterministic) actions, before these actions are actually executed.

3.3 CCALC

The Causal Calculator (CCALC) [38] is a causality-based reasoning system that allows representation of dynamic domains in a subset of the expressive action description $\mathcal{C}+$ and can compute solutions to planning problems with temporal constraints described by a query in a variation of $\mathcal{Q}$. Given an action description and a query, CCALC finds an answer to the query in the spirit of satisfiability planning [44]: 1) it transforms the

```
:- objects
1 :: worker.

:- query
label::1;
maxstep::0..2;

0:
xpos(1)=3, ypos(1)=3,
battery(1)=7,
endEffector(1)=1,
linePos(1)=4,
workDone(1)=0,
-wetpaint(1);

maxstep:
workDone(1)=1.
```

Figure 3.1: A planning problem in a Cognitive Factory domain presented to CCALC

action description and the query into a set of formulas in propositional logic [38]; 2) it calls a SAT solver (like MANYSAT [41]) to find a model of this propositional theory; 3) if a model is found then it extracts the plan; otherwise, it answers the query negatively.

In the syntax of CCALC, conjunctions $\wedge$, disjunctions $\vee$, implications $\supset$, negations $\neg$, universal quantifiers $\forall$, and existential quantifiers $\exists$ are replaced with the symbols `&`, `++`, `->>`, `-`, `/\`, and `\/` respectively.

Planning problems are represented in the form of queries in CCALC. A planning problem in a Cognitive Factory domain can be seen in Figure 3.1. In the initial state, the worker robot `we1` is located at cell $(3, 3)$ with a battery of 7 and has its end effector prepared to work on the work stage 1. There is also a box, `1`, at position 4 with no work done on it. We want to find the shortest plan to have the working stage of box `1` at 1.

The output for the presented planning problem and the domain description is shown in Figure 3.2. In the plan, we see that if the assembly line shifts to make box `1` align with worker `we1` at step 0 and worker `we1` works on the box at step 1, box `1` will have the desired working stage at step 2.

CCALC allows us to embed geometric reasoning using external atoms, which are not part of the signature of the domain description (i.e., they are not declared as fluents or actions), and are implemented externally in some programming language of the user' choice as functions, (e.g., a collision checker utilizing a probabilistic motion planning algorithm). They are evaluated in SWI Prolog while grounding the causal laws.

```
0:  workDone(1)=0 endEffector(1)=1 battery(1)=7 linePos(1)=4
    xpos(1)=3 ypos(1)=3 -wetpaint(1)
ACTIONS: lineShift
1:  workDone(1)=0 endEffector(1)=1 battery(1)=7 linePos(1)=3
    xpos(1)=3 ypos(1)=3 -wetpaint(1)
ACTIONS: workOn(we1, 1)
2:  wetpaint(1) workDone(1)=1 endEffector(1)=1 battery(1)=5
    linePos(1)=3 xpos(1)=3 ypos(1)=3
```

Figure 3.2: A Cognitive Factory plan obtained from CCALC

3.4 Embedding Continuous Feasibility Checks into Causal Planning

Our method on combining discrete task planning with continuous feasibility checks is based on earlier works [19, 22] on hybrid planning.

With the pre-computation method (PRE): We externally compute all possible cases of robot-obstacle collisions in advance and then embed this information into the action domain description via external predicates. For instance, the external predicate `existsCollision(robot_type,X,Y,U1,U2)` is implemented in C++ utilizing Open Dynamics Engine (ODE) [1]. It considers all the static obstacles in the environment and checks whether the robot at `(X,Y)` moving in units `U1,U2` collides with any of the obstacles. It returns 1 is there is a collision, and 0 if there is no collision. We can express that a worker `W` of type `1` can not move in the directions `right,up` with units `U1,U2` if there is a collision by the following condition:

```
nonexecutable move(W,right) & move(W,up) if  distance(W,right)=U1 &
     distance(W,up)=U2 & xpos(W)=X & ypos(W)=Y  where X+U1=<maxX &
     Y+U2=<maxY & existsCollision(1,X,Y,U1,U2).
```

With the guided replanning method (GREPL): After computing a plan, we can check for collisions of robots, using the external computations provided by ODE. If the plan is found infeasible, then we can identify which actions c are being executed at which state s when a collision occurs. Based on this information, we can ask for a new plan which does not involve execution of c at s, by adding a constraint to the planning problem description. For instance, if it is found by collision checks that two worker robots `W1` and `W2` can not cross each other diagonally between locations `(X1,Y1)` and `(X2,Y2)`, then we can add the following constraint to the planning problem

```
caused false if xpos(W1)=X2 & ypos(W1)=Y2 & xpos(W2)=X1 & ypos(W2)=Y1
```

[1] http://pyode.sourceforge.net

```
after xpos(W1)=X1 & ypos(W1)=Y1 & xpos(W2)=X2 & ypos(W2)=Y2
where W1\=W2.
```

to guide the solver to find a new plan where the workers `W1` and `W2` do not exchange their locations `(X1,Y1)` to `(X2,Y2)`, respectively, at any time step.

Chapter 4

Answer Set Programming

Answer Set Programming (ASP) [49, 5, 34, 50] is a declarative programming paradigm oriented towards solving combinatorial search problems as well as knowledge-intensive problems. The idea is to represent a problem as a "program" whose models (called "answer sets") correspond to the solutions, and to find the answer sets for that program using an answer set solver. In the representation of a problem use rules to "generate" many answer sets corresponding to "possible solutions", and constraints to "eliminate" the answer sets that do not correspond to solutions.

4.1 Programs and Answer Sets

Let us briefly describe the details of Answer Set Programming (ASP) as in [34, 49].

Syntax An *ASP program* Π over signature σ is a finite set of rules of the form

$$A_0 \leftarrow A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, A_n \tag{4.1}$$

where $n \geq m \geq 0$ and each $A_i \in \sigma$ is an atom.

A_0 is an atom or $\bot$, and it is called the *head*, and $A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, A_n$ is the *body* of the rule. A rule is called a *fact* if $m{=}n{=}0$ and a *constraint* if A_0 is $\bot$. We generally omit the $\leftarrow$ sign for facts, and the $\bot$ sign for constraints.

Semantics An interpretation I of an ASP program Π is a set of atoms. A rule is *satisfied* by an interpretation I if the head of the rule is true whenever the body of the same rule is true with respect to I. An interpretation I satisfies a program Π if it satisfies all the rules in the program, and it is called a *model* of Π.

The *reduct* Π^I of Π with respect to I is the set of rules

$$A_0 \leftarrow A_1, \ldots, A_m \tag{4.2}$$

for all rules in Π of the form (4.1) with $\{A_{m+1}, \ldots, A_n\} \cap I = \emptyset$.

An *answer set* of a program Π that does not contain negation as failure is defined to be a model I that is the "minimal" among the other models of Π (relative to set-theoretic inclusion). For example, consider the following program without negation as failure

$$a \leftarrow b.$$

Note that this simple program has three models $\{\}, \{a\}$ and $\{a, b\}$. Since both $\{a, b\}$ and $\{a\}$ have strict subsets which are also a model of the program, they cannot be answer sets. On the other hand, $\{\}$ is a minimal model, and thus, the answer set of the program.

Now consider a program Π that may contain negation. A model I is an *answer set* for Π, if it is an answer set of the reduct Π^I. Consider, for instance, the program

$$a \leftarrow \textit{not } b$$
$$b \leftarrow \textit{not } a$$

and let a partial interpretation I be $\{b\}$. The reduct of this program relative to I is

$$b \leftarrow$$

and it is satisfied by I. Since the only strict subset of I is $\{\}$, and it does not satisfy the reduct, I is a minimal model of the reduct. Thus, I is an answer set of the program. Similarly, another answer set of the program is $\{a\}$.

More general forms of ASP programs are also defined such as forms that may contain classical negation $\neg$ and disjunction [34] and nested expressions [51] in the heads of rules as well.

In ASP, we use special constructs to express choice and cardinality constraints. The constructs of the form

$$\{L_1, \ldots, L_k\}^c \tag{4.3}$$

are called *choice expressions*, and constructs of the form

$$m \leq \{L_1, \ldots, L_k\} \leq n \tag{4.4}$$

are called *cardinality expressions* which allows to bound, from below (by m) and from above (by n), the number of literals included in the answer set. Programs using these construct can be viewed as abbreviations for programs defined above [27]. For instance the following program

$$\{a\}^c$$

stands for the program

$$a \leftarrow \textit{not not } a.$$

The constraint

$$\leftarrow 2\{a, b, c\}$$

stands for

$$\leftarrow a, b$$
$$\leftarrow a, c$$
$$\leftarrow b, c.$$

Expression (4.3) describes subsets of $\{L_1, \ldots, L_k\}$. When used in heads of rules, (4.3) generates many answer sets, and these type of rules are called *choice rules*. For instance, the program

$$\{a, b, c\}^c \tag{4.5}$$

has answer sets that are arbitrary subsets of $\{a, b, c\}$. Expression (4.4) describes subsets of $\{L_1, \ldots, L_k\}$ whose cardinalities are between m and n. When used in constraints, it eliminates some answer sets, and these are called *cardinality constraints*. For instance, adding the constraint

$$\leftarrow 2\{a, b, c\}$$

to program (4.5) eliminates the answer sets for (4.5) whose cardinalities are at least 2. Adding

$$\leftarrow \textit{not}\ (1\{a, b, c\}1)$$

eliminates the answer sets for (4.5) whose cardinalities are not exactly 1.

Another special construct is used to express optimization statements [64]. To indicate preferences, we use *minimize* and *maximize* statements of the form

$$\textit{minimize}\ \{L_1 = w_{L_1}, \ldots, L_m = w_{L_m}, \textit{not}\ L_{m+1} = w_{L_{m+1}}, \ldots, \textit{not}\ L_n = w_{L_n}\}$$

to associate weights with specific literals. This statement expresses that we want to find an answer set with the smallest weight. If there are several minimize statements, then the answer sets are ordered lexicographically according to the weights of the statements. The first statement is considered to be the most significant.

Let us give examples for the use of these special constructs in a cognitive factory domain. Cardinality constraints can be used to express that a team needs at least two robots at time step t and optimization statements can be used to optimize a plan by minimizing the total cost of actions. For instance, the following expression

$$\#\textit{minimize}\ [\ \textit{cost}(r, c, t) : \textit{robot}(r) = c\ ] \tag{4.6}$$

is used to minimize the sum of all costs c of robotic actions performed in a plan, where costs of actions performed by robot r at time step t are defined by atoms of the form $\textit{cost}(r, c, t)$.

4.2 External Atoms in ASP

In this thesis, we consider nondisjunctive HEX programs [18], which has an extension of answer set semantics to higher order atoms with external atoms. An external atom is an expression of the form $\&g[y_1, \ldots, y_k](x_1, \ldots, x_l)$ where $y_1, \ldots, y_k$ and $x_1, \ldots, x_l$ are two lists of terms (called input and output lists, respectively), and $\&g$ is an external predicate name. Intuitively, an external atom provides a way for deciding the truth value of an output tuple depending on the extension of a set of input predicates. External atoms allow us to embed results of external computations into ASP programs. They are usually implemented in a programming language of the user's choice, like C++.

Consider, for instance, a workspace in a cognitive factory, with some obstacles. With the method PRE for integration: An object detection algorithm can be used to identify all locations l occupied by these obstacles, and the results of this external computation can be embedded into the formulation of a state constraint expressing that a robot r cannot be at a location l occupied by an obstacle at any time step t:

$$\leftarrow at(r, l, t), \&\, obstacleAt[l]()$$

where $\&obstacleAt$ is an external predicate whose value is determined by an object detection algorithm [15] using Point Cloud Library over data obtained by a Kinect RGB-D camera. We can also express a transition constraint to avoid collisions of robots with obstacles while the robots move from one location l_1 to another l_2:

$$\leftarrow at(r, l_1, t), goto(r, l_2, t), \&\, collision[r, l_1, l_2]()$$

where the external predicate $collision$ checks for such collisions using Open Dynamics Engine (ODE).

4.3 CLASP

After representing a computational problem as a program whose answer sets correspond to solutions of the problem, we can use an answer set solver to compute the solutions of the problem.

We need to make some syntactic modifications on the program to present it to an answer set solver, like CLASP [33]. In the syntax of CLASP, the head of a rule can be of the forms (4.3) or (4.4) but with the superscript ^{c} and the sign $\leq$ dropped. The body can contain cardinality expressions with the sign $\leq$ dropped. Each arrow symbol $\leftarrow$ is replaced with the symbol `:-`, and each rule is followed by a period.

CLASP finds an answer set for a program in two stages: first a "grounder", like

```
query_label(query) :- true.
false :- not h(eql(xpos(1),3),0),query_label(1).
false :- not h(eql(ypos(1),3),0),query_label(1).
false :- not -h(eql(wetpaint(1),true),0),query_label(1).
false :- not h(eql(linePos(1),4),0),query_label(1).
false :- not h(eql(workDone(1),0),0),query_label(1).
false :- not h(eql(battery(1),7),0),query_label(1).
false :- not h(eql(endEffector(1),1),0),query_label(1).
false :- not h(eql(workDone(1),1),maxstep),query_label(1).
true.
:- false.
```

Figure 4.1: A planning problem in a Cognitive Factory domain presented to CLASP

```
Answer: 1
h(eql(workOn(1,1),true),1) h(eql(move(1,right),true),0)
h(eql(xpos(1),3),0) h(eql(ypos(1),3),0) h(eql(linePos(1),4),0)
h(eql(workDone(1),0),0) h(eql(endEffector(1),1),0) h(eql(battery(1),7),0)
h(eql(battery(1),6),1) h(eql(endEffector(1),1),1) h(eql(workDone(1),0),1)
h(eql(linePos(1),4),1) h(eql(ypos(1),3),1) h(eql(xpos(1),4),1)
h(eql(xpos(1),4),2) h(eql(ypos(1),3),2) h(eql(linePos(1),4),2)
h(eql(workDone(1),1),2) h(eql(endEffector(1),1),2) h(eql(battery(1),4),2)
h(eql(wetpaint(1),true),2)
SATISFIABLE
```

Figure 4.2: A Cognitive Factory plan obtained from CLASP

GRINGO is used to get rid of the schematic variables, and then a DPLL-like branch and bound algorithm is used to find an answer set for the ground program. The plan can be extracted from the answer set of the ground program, if found.

A planning problem can be solved using ASP, by specifying the initial state and the goal state in the query. An example can be seen in Figure 4.1, which is the transformation of the planning problem in Figure 3.1 into ASP. The output of the presented planning problem and the domain description is shown in Figure 4.2.

4.4 Transformations from $\mathcal{C}+$ to ASP

There are various transformations between causal logic, $\mathcal{C}+$, and ASP, such as [38, 56, 25, 52, 26]. We use an automated tool CPLUS2ASP [7], which uses the modified translation [26] of McCain's translation, to transform a domain description in $\mathcal{C}+$ to an ASP program.

CPLUS2ASP is a standard library which describes the constructs of the input lan-

guage of CCALC in terms of ASP. It can handle multi-valued constants and turn the language of CCALC into formulas under the stable model semantics. Afterwards, it uses the software system F2LP ("formulas to logic programs") to turn these into the input language of ASP solvers.

For example, a law expressing the direct effect of an action in the language of CCALC such as

```
workOn(W,B) causes workDone(B)=WS1 if workDone(B)=WS1-1.
```

is translated into the language of CLASP as follows

```
h(eql(workDone(B),WS1),V_astep+1) :-
    h(eql(workOn(W,B),true),V_astep),h(eql(workDone(B),WS1-1),V_astep).
```

Here `V_astep` denotes the time step ranges in $\{0..maxstep - 1\}$.

Terms of the form $eql(c, v)$ are atoms to express that fluent c has the value v, and atoms of the form $h(f, t)$ express that f is true at time step t.

A law expressing the precondition of an action such as

```
nonexecutable workOn(W,B)
        if battery(W) = BL where BL < workCons.
```

is translated into a constraint

```
false :- h(eql(workOn(W,B),true),V_astep),
      h(eql(battery(W),BL),V_astep),BL<workCons.
```

The translation of the planning problem shown in Figure 3.1 into ASP, can be seen in Figure 4.1.

Chapter 5

Finding an Optimal Global Plan for Multiple Teams of Heterogeneous Robots

In this chapter we describe our logic-based method to find optimal global plans (with minimum makespan) for multiple teams of heterogeneous robots through a mediator.

We consider multiple teams of n types of robots, where each team is given a feasible task to complete in its workspace on its own using hybrid reasoning as described above, and where teams are allowed to transfer robots between each other. The goal is to find an optimal feasible global plan for all teams so that all tasks can be completed as soon as possible within at most k steps, where at most $\overline{m}_x$ robots of type x can be transferred between any two teams, and subject to the following constraints:

C1 Teams do not know about each other's workspace or tasks (e.g., for the purpose of privacy in micro manufacturing plants that specialize on prototyping pre-release products).

C2 Lending/borrowing robots between workspaces back and forth is not desired (e.g., transportation of robots is usually costly, also, since tasks may be different in workspaces, robots need some tuning). Also, for similar reasons, robots can be transferred between two teams in a single batch.

The approach we introduce in this chapter was published in [21] for the special case where only homogeneous robots were considered ($n = 1$), and in [59] for the general case.

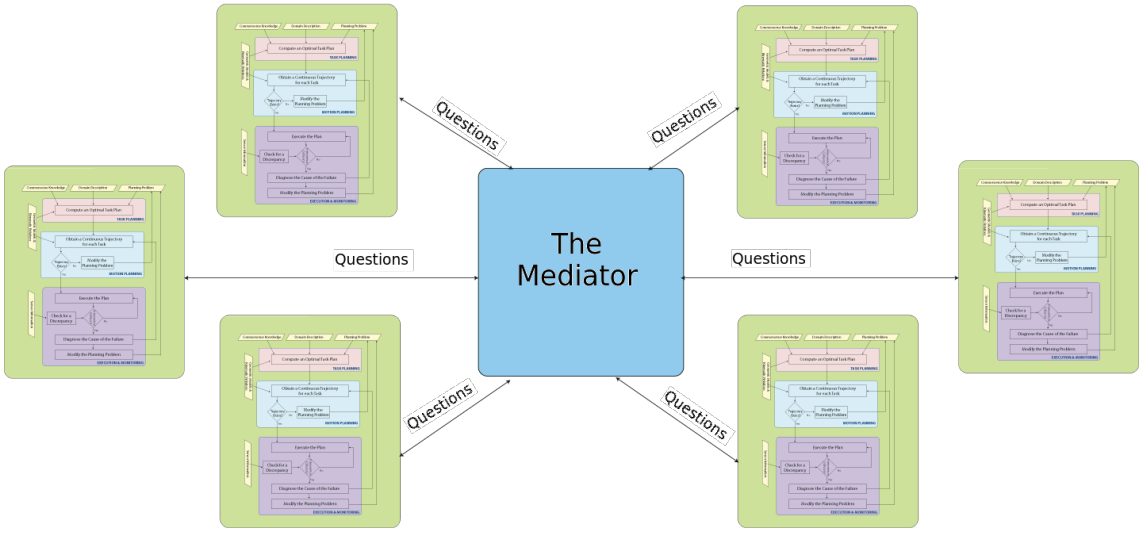


Figure 5.1: A semi-distributed approach

5.1 Overall Approach

We introduce a novel semi-distributed method to find an optimal global plan for all teams, with at most k steps, subject to constraints [C1] and [C2], and the presence of a mediator who does not belong to any team and who does not know anything about teams' workspaces, tasks and goals. Our method consists of two phases: finding a coordination of the teams and then an optimal global plan.

In the first phase, for a nonnegative integer $\bar{l} \leq k$ denoting the length of a global plan: 1) The mediator asks yes/no questions to every team (in any order), to identify whether a team can complete its task in $\bar{l}$ steps, while lending/borrowing how many robots to/from of which sort. 2) Once answers to these questions are collected, the mediator tries to find a coordination of the teams (i.e., which team should lend how many robots of which sort to which other team, and when), subject to the constraints [C1] and [C2]. The optimal value for $\bar{l}$ can be found by a linear search.

In the second phase, after some coordination of teams is found for an optimal value of $\bar{l}$, the mediator informs each team how many robots it is expected to lend to (or borrow from) which other team and when. Taking this information into account, each team computes an optimal local plan (whose length is less than or equal to $\bar{l}$) to complete its task. An optimal global plan for all teams is then obtained as the union of all optimal local plans.

Note that the mediator cannot find a global plan on its own since it does not know about teams' workspace, tasks, plans, actions, goals, etc.. In fact, a centralized approach to compute a global plan is in most cases not scalable due to large domain description that formalizes all workspaces and teams. Also note that teams do not communicate with each other. Otherwise, the number of queries (and the number of rounds of exchanging

messages) would increase substantially, leading to a more time-consuming process to find an optimal global plan.

Both phases involve solving computational problems that are intractable, since finding plans of length $\bar{l}$ possibly with temporal constraints is NP-complete [69], and answering each query in the first phase is a planning problem with temporal constraints, and thus NP-complete [69]. We prove that finding a coordination of the teams for a global plan with at most $\bar{l}$ steps is also NP-complete (see Proposition 5.4).

Note that, in the first phase, each team answers queries that are relevant to its workspace, task, goals only, and independently of other teams; therefore, queries can be answered in parallel. In the second phase, each team computes an optimal local plan on its own; therefore, optimal local plans can be computed in parallel as well.

5.2 Querying Teams

The mediator asks yes/no questions of the following three forms to every team (in any order), for every $\bar{l} \leq k$, $l \leq \bar{l}$ and $m \leq \overline{m}_x$, $x \leq n$:

Q1 Can you complete your task in $\bar{l}$ steps?

Q2 Can you complete your task in $\bar{l}$ steps, if you lend m robots of type x before step l?

Q3 Can you complete your task in $\bar{l}$ steps, if you borrow m robots of type x after step l?

Answering each question is a planning problem with temporal constraints, and thus NP-complete [69]. These questions can be further generalized, considering different combinations of types of robots that are lent or borrowed. We do not consider such generalizations, for computational efficiency purposes. Therefore we have a third constraint

C3 Teams can borrow/lend robots of the same sort.

Examples for queries of the forms Q1-Q3 are shown in Figure 5.2. The query in Figure 5.2a asks for a plan of length 25. The query in Figure 5.2b asks for a plan of length 25 with the constraint of lending 1 dry robot before step 3. The query in Figure 5.2c asks for a plan of length 25 with the constraint of borrowing 2 dry robots after step 7.

Figure 5.2: Examples of queries of forms Q1-Q3 presented to CCALC

```
:- query
label::1;
maxstep::25;

0:
 xpos(c1)=1, ypos(c1)=1, xpos(1)=1, ypos(1)=3,
[/\C /\W | -docked(C,W)], [/\R | -docked(R)],
[/\B | -wetpaint(B)],
linePos(B)=B+lineLength,
[/\B | workDone(B) = 0], [/\W | battery(W)=maxBattery],
[/\W | endEffector(W)=1],
[/\C | -bench(C)];

maxstep:
[/\C /\W | -docked(C,W)], [/\R | -docked(R)],
linePos(maxBox) = 0, [/\ B | workDone(B) = 3].
```

(a) Query of form Q1

```
:- macros
maxT -> 50.
:- variables
T :: 0..maxT-1;
WE1:: dryrobot.

:- query
label::1;
maxstep::25;

0:
xpos(c1)=1, ypos(c1)=1, xpos(1)=1, ypos(1)=3, xpos(2)=1, ypos(2)=3,
[/\C /\W | -docked(C,W)], [/\R | -docked(R)],
[/\B | -wetpaint(B)],
linePos(B)=B+lineLength,
[/\B | workDone(B) = 0], [/\W | battery(W)=maxBattery],
[/\W | endEffector(W)=1],
[/\C | -bench(C)];

maxstep:
[\/T | T<3,[\/WE1 | (T: giveRobot(WE1))]],
[/\C /\W | -docked(C,W)], [/\R | -docked(R)],
linePos(maxBox) = 0, [/\ B | workDone(B) = 3].
```

(b) Query of form Q2

```
:- macros
maxT -> 50.
:- variables
T :: 0..maxT-1.
:- objects
we1,we2:: dryrobot.

:- query
label::1;
maxstep::25;

0:
xpos(c1)=1, ypos(c1)=1, xpos(1)=1, ypos(1)=3,
[/\C /\W | -docked(C,W)], [/\R | -docked(R)],
[/\B | -wetpaint(B)],
linePos(B)=B+lineLength,
[/\B | workDone(B) = 0], [/\W | battery(W)=maxBattery],
[/\W | endEffector(W)=1],
[/\C | -bench(C)];

7: (bench(we1), xpos(we1)=minX-1,ypos(we1)=minY-1,
bench(we2),xpos(we2)=minX-1,ypos(we2)=minY-1);

maxstep:
[\/T | T>=7, T<maxstep, (T: takeRobot(we1)),(T: takeRobot(we2))],
[/\C /\W | -docked(C,W)], [/\R | -docked(R)],
linePos(maxBox) = 0, [/\ B | workDone(B) = 3].
```

(c) Query of form Q3

5.3 Inferring Knowledge about Robot Transfers

From teams' answers to the yes/no questions posed by the mediator, the following can be inferred:

- If there is a team that answers "no" to every question, then there is no overall plan of length $\bar{l}$ where every team completes its own tasks.
- Otherwise, we can identify sets *Lenders*$_x \subset$ *Lenders* of lender teams that can lend robots of type x and sets *Borrowers*$_x \subset$ *Borrowers* of borrower teams that needs to borrow robots of type x, where $x \leq n$ (*Lenders*, *Borrowers* $\subset$ *Teams*): If a team answers no to question Q1 and "yes" to question Q3 for some l, m and x, then it is a borrower for robot type x. If a team answers "yes" to question Q1 and "yes" to question Q2 for some l, m and x, then it is a lender for robot type x.
- For every lender (resp., borrower) team, from its answers to queries Q2 (resp., Q3), we can identify the earliest (resp., latest) time it can lend (resp., borrow) m robots of type x, $x \leq n$, in order to complete its tasks in $\bar{l}$ steps.

For every $\bar{l} \leq k$, these inferences can be used to decide whether lenders and borrowers can collaborate with each other, so that every team completes its task in $\bar{l}$ steps as follows.

For instance, we can identify the earliest lend times and latest borrow times by a collection of partial functions:

$$\textit{Lend\_earliest}_{m,x} : \textit{Lenders}_x \mapsto \{0, \ldots, \bar{l}\}$$
$$\textit{Borrow\_latest}_{m,x} : \textit{Borrowers}_x \mapsto \{0, \ldots, \bar{l}\}$$

5.4 Coordination of Teams

Usually transferring robots from one team to another team takes some time, not only due to transportation but also due to calibration of the robots for a different workspace. Let us define such a delay time by a function:

$$\textit{Delay} : \textit{Lenders} \times \textit{Borrowers} \times \{1, \ldots, n\} \mapsto \{0, \ldots, \bar{l}\}.$$

Considering the earliest lend times, latest borrow times, and the delay of transfers, let us define the conditions under which a set of lender teams can collaborate with a set of borrower teams.

Definition 5.1. An $n\overline{m}\bar{l}$*-collaboration* between *Lenders* and *Borrowers* with at most $\overline{m} = \max\{\overline{m}_x\}$ robot transfers, with n types of robots, and within at most $\bar{l}$ steps, relative to *Delay*, is a partial function

$$f : \textit{Lenders} \times \textit{Borrowers} \times \{1, \ldots, n\} \mapsto \{0, \ldots, \bar{l}\} \times \{0, \ldots, \overline{m}\}$$

(where $f(i, j, x) = (l, u)$ denotes that team i lends u robots of type x to team j at time step l) such that the following hold:

(a) For every borrower team $j \in \textit{Borrowers}_x$, there are some lender teams $i_1, \ldots, i_s \in \textit{Lenders}_x$, $x \leq n$, where the following two conditions hold:

- $f(i_1, j, x) = (l_1, u_1), \ldots, f(i_s, j, x) = (l_s, u_s)$ for some time steps $l_1, \ldots, l_s \leq \bar{l}$, some positive integers $u_1, \ldots, u_s \leq \overline{m}_x$, and some type x,
- $\textit{Delay}(i_1, j, x) = t_1, \ldots, \textit{Delay}(i_s, j, x) = t_s$ for some time steps $t_1, \ldots, t_s \leq \bar{l}$;

and there is a positive integer $m \leq \overline{m}_x$ such that

$$\max\{l_1 + t_1, \ldots, l_s + t_s\} \leq \textit{Borrow\_latest}_{m,x}(j)$$
$$m \leq \textstyle\sum_{k=1}^{s} u_k.$$

(b) For every lender team $i \in$ *Lenders*$_x$, for all borrower teams $j_1, \ldots, j_s \in$ *Borrowers*$_x$, $x \leq n$, such that $f(i, j_1, x) = (l_1, u_1), \ldots, f(i, j_s, x) = (l_s, u_s)$ for some time steps $l_1, \ldots, l_s \leq \bar{l}$, some positive integers $u_1, \ldots, u_s \leq \overline{m}_x$, and some type x, and there is a positive integer $m \leq \overline{m}_x$ such that

$$\textit{Lend\_earliest}_{m,x}(i) \leq \min\{l_1, \ldots, l_s\}$$
$$m \geq \textstyle\sum_{k=1}^{s} u_k.$$

Condition (a), which ensures that a borrower team does not borrow fewer robots than it needs, and Condition (b), which ensures that a lender team does not lend more robots than it can, together entail the existence of a lender team that can lend robots when a borrower team needs them.

Now we are ready to precisely describe the computational problem of finding a coordination of multiple teams of heterogeneous robots, to complete all the tasks as soon as possible in at most $\bar{l}$ steps where at most $\overline{m}$ robots can be relocated:

FINDCOLLABORATION_n
INPUT: For a set *Lenders* of lender teams, a set *Borrowers* of borrower teams, positive integers n, $\bar{l}$ and $\overline{m}_x$, $x \leq n$: a delay function *Delay* and a collection of functions *Lend_earliest*$_{m,x}$ and *Borrow_latest*$_{m,x}$ for every positive integer $m \leq \overline{m}_x$, $x \leq n$.
OUTPUT: A $n\overline{ml}$*-collaboration* between *Lenders* and *Borrowers* with at most $\overline{m} = \max\{\overline{m}_x\}$ robot transfers, with at most n types of robots, and within at most $\bar{l}$ steps, relative to *Delay*.

As expected, this problem is intractable even when the robots are homogeneous (assuming that P $\neq$ NP). The proof is shown in the next section.

Example 5.2. Consider four teams of robots, where Teams 1 and 2 are lenders and Teams 3 and 4 are borrowers. Take $\bar{l}=8, m_1=4$ and $n = 1$. The lenders' answers to questions of the form Q2 ("Can you complete your task in $\bar{l}$ steps, if you lend m robots of type x before step l?") and the borrowers' answers to questions of the form Q3 ("Can you complete your task in $\bar{l}$ steps, if you borrow m robots of type x after step l?") are summarized in Figure 5.3. The affirmative (resp., negative) answers to questions for time step l are denoted by green/solid (resp., red/hatched); the number m of robots that can be lent or needs to be borrowed are denoted above the rows. According to these answers, Team 1 can lend 2 robots after step 3 or 4 robots after step 7, Team 2 can lend 1 robot after step 2, Team 3 needs to borrow 1 robot before step 5 or 3 robots before step 7, and Team 4 needs to borrow 2 robots before step 6. Suppose the delay time is $Delay(i, j) = \mid i - j \mid$.

We can show that an $n\overline{ml}$*-collaboration* f exists: $f(1,3) = (3,1), f(1,4) = (3,1), f(2,4) = (2,1)$. Indeed, f satisfies the conditions in Definition 5.1.

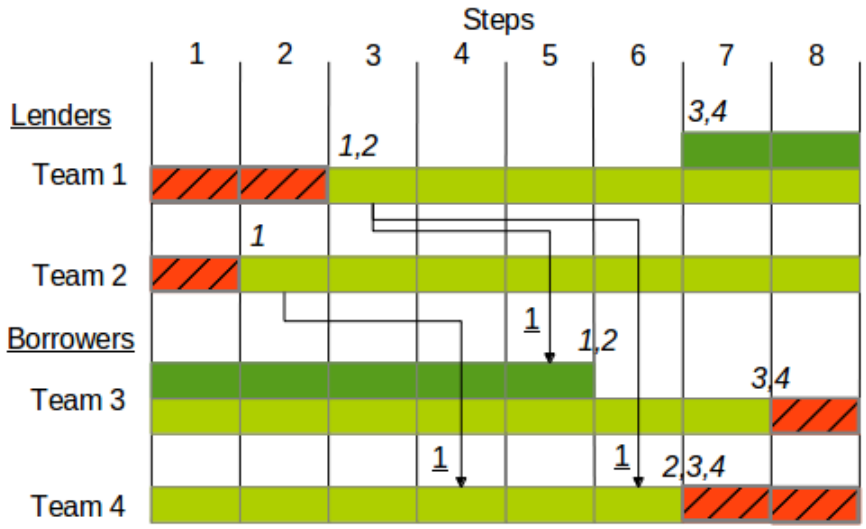


Figure 5.3: A summary of teams' answers to queries

(a) for Team 3, $f(1,3) = (3,1)$, and there exists $m = 1 \leq 1$ such that $Borrow\_latest_{1,1}(3)$=5>3+2. So Team 3 can finish its task in 8 steps if it borrows 1 robot before step 5. Similarly, for Team 4, $f(1,4) = (3,1)$ and $f(2,4) = (2,1)$, and there exists $m = 2 \leq 1$+1 such that $Borrow\_latest_{2,1}(4) = 6 > \max 3+3, 2+2$.

(b) for Team 1, $f(1,3) = (3,1)$ and $f(1,4) = (3,1)$, and there exists $m = 2 > 1+1$ such that $Lend\_earliest_{2,1}(1) = 3 \leq 3$. Similarly, for Team 2, $f(2,4) = (2,1)$, and there exists $m = 1 > 1$ such that $Lend\_earliest_{1,1}(2) = 2 \leq 2$.

5.5 Finding a Team Coordination is Hard

The problem FINDCOLLABORATION_n is intractable, as expected.

Proposition 5.3. *The decision version of* FINDCOLLABORATION_n *(i.e., existence of a* $n\overline{ml}$*-collaboration) is NP-complete.*

To prove this proposition, we will need the following lemma (Proposition 1 in [21]):

Lemma 5.4. *The decision version of* FINDCOLLABORATION *(*FINDCOLLABORATION_n *for n=1) is NP-complete.*

Proof of Proposition 5.3. We prove the membership as in the proof of Lemma 5.4. We prove the hardness by a polynomial-time reduction from FINDCOLLABORATION, which is an NP-complete problem: consider one type of robots in FINDCOLLABORATION_n. □

For Lemma 5.4 we consider an $n\overline{ml}$-collaboration for one type of robots ($n = 1$). Therefore the definition of an $n\overline{ml}$-collaboration is reduced to the following form:

Definition 5.5. An $\overline{ml}$*-collaboration* between *Lenders* and *Borrowers* with at most $\overline{m}$ robot transfers and within at most $\bar{l}$ steps, relative to *Delay*, is a partial function

$$f : \textit{Lenders} \times \textit{Borrowers} \to \{0, \ldots, \bar{l}\} \times \{0, \ldots, \overline{m}\}$$

(where $f(i,j) = (l,u)$ denotes that team i lends u robots to team j at time step l) such that the following hold:

(a) For every borrower team $j \in$ *Borrowers*, there are some lender teams $i_1, \ldots, i_s \in$ *Lenders* where

- $f(i_1, j) = (l_1, u_1), \ldots, f(i_s, j) = (l_s, u_s)$ for some time steps $l_1, \ldots, l_s \leq \bar{l}$ and some positive integers $u_1, \ldots, u_s \leq \overline{m}$, and
- $\textit{Delay}(i_1, j) = t_1, \ldots, \textit{Delay}(i_s, j) = t_s$ for some time steps $t_1, \ldots, t_s \leq \bar{l}$;

and there is a positive integer $m \leq \overline{m}$ such that

$$\begin{aligned} \max\{l_1 + t_1, \ldots, l_s + t_s\} &\leq \textit{Borrow\_latest}_m(j) \\ m &\leq \textstyle\sum_{k=1}^{s} u_k. \end{aligned}$$

(b) For every lender team $i \in$ *Lenders*, for all borrower teams $j_1, \ldots, j_s \in$ *Borrowers* such that $f(i, j_1) = (l_1, u_1), \ldots, f(i, j_s) = (l_s, u_s)$ for some time steps $l_1, \ldots, l_s \leq \bar{l}$ and some positive integers $u_1, \ldots, u_s \leq \overline{m}$, and there is a positive integer $m \leq \overline{m}$ such that

$$\begin{aligned} \textit{Lend\_earliest}_m(i) &\leq \min\{l_1, \ldots, l_s\} \\ m &\geq \textstyle\sum_{k=1}^{s} u_k. \end{aligned}$$

Proof of Lemma 5.4. Let us denote by FINDCOLLABORATION$_D$ the decision version of FINDCOLLABORATION, i.e., decide for an existence of a $\overline{ml}$-collaboration. We prove that FINDCOLLABORATION$_D$ is NP-complete in two parts: membership and hardness.

Membership: Let $\Sigma = \{0, 1, \wedge, \vee, \cdot, \circ, \bullet, \star\}$ be the alphabet, and let Σ^* denote the set of all strings over Σ^*. We define a language L to be the set of all strings in Σ of the form $B_1 \wedge B_2 \wedge B_3 \wedge B_4 \wedge D \wedge X \wedge Y$ where

- B_1, B_2, B_3, B_4 are binary representations of the number of Lenders $|\textit{Lenders}|$, the number of Borrowers $|\textit{Borrowers}|$, the maximum number of steps $\bar{l}$, and the maximum number of robots $\overline{m}$, respectively.
- D has the form $D_{1,1} \wedge D_{1,2} \wedge \ldots \wedge D_{a,b}$ with each $D_{i,j}$ of the form $(I_d \wedge J_d) \cdot D'$ where I_d and J_d are the binary representation of lender index i and borrower index j, and D' is the binary representation of the value $\textit{Delay}(i, j)$.

- X has the form $X_{m_1,i_1} \wedge X_{m_1,i_2} \wedge \cdots$ with each $X_{m,i}$ of the form $(M_x \wedge I_x) \circ S_x$ where M_x and I_x are the binary representation of number m and lender index i, and S_x is the binary representation of the value ***Lend_earliest***$_m(i)$.

- Y has the form $Y_{m_1,j_1} \wedge Y_{m_1,j_2} \wedge \cdots$ with each $Y_{m,j}$ of the form $(M_y \wedge J_y) \bullet S_y$ where M_y and J_y are the binary representation of number m and borrower index j, and S_y is the binary representation of the value $Borrow\_latest_m(j)$,

such that, given an input $x \in \Sigma^*$ then $x \in L$ iff FINDCOLLABORATION$_D$ with input corresponding to x returns yes.

Note that FINDCOLLABORATION$_D$ is a decision problem since FINDCOLLABORATION$_D$ returns yes if and only if $x \in L$.

We will show that FINDCOLLABORATION$_D$ is in NP by showing that (A) the above representation $x \in \Sigma^*$ is polynomial in the size of an input to FINDCOLLABORATION$_D$, (B) we can describe a guess y of polynomial size corresponding to a potential collaboration function f, and (C) checking whether y satisfies all conditions of Def. 1 with respect to input x can be done by a polynomial time algorithm.

(A) The input $x \in \Sigma^*$ consists of four numbers and at most $1+2{\cdot}\overline{m}$ functions (***Delta*** plus a maximum of $\overline{m}$ $Lend\_earliest$ and $Borrow\_latest$ functions), where ***Delta*** has size $\mathcal{O}(|Lenders| \cdot |Borrowers| \cdot \log(\bar{l}))$ and the other functions are below that. Therefore, $|x|$ has polynomial size in $\bar{l}{\cdot}\overline{m}$.

(B) A guess y, corresponding to a collaboration function f, will be of the form $F_1 \wedge \cdots \wedge F_w$ with $w = \mathcal{O}(|Lenders| \cdot |Borrowers| \cdot \bar{l} \cdot \log(\overline{m})) = \mathcal{O}(|x|^4)$ where each F_i is of the form $(I_u \wedge J_u) \star (L_u \wedge M_u)$ with I_u, J_u, L_u, M_u the binary representations of lender i_u, borrower j_u, time step l_u and number of robots m_u, for $f(i_u, j_u) = (l_u, m_u)$. Each F_i has linear size in $|x|$ therefore $|y| = \mathcal{O}(|x|^5)$ and therefore polynomial.

(C) We can check whether y conforms to all conditions in polynomial time: For condition (a) and (b), we check at most $\overline{m}$ values of $Borrow\_latest$ and $Lend\_earliest$ functions for each $Borrower$ and $Lender$, respectively. Therefore, the checking algorithm takes polynomial time in $|x|$.

Hence, FINDCOLLABORATION$_D$ is in NP.

Hardness: Take any 3-SAT instance F over signature σ of n variables $x_1, \ldots, x_n$ and p clauses $c_1, ..., c_p$ of the form $c_j = (t_{j,1}, t_{j,2}, t_{j,3})$, where $t_{j,1}, t_{j,2}, t_{j,3}$ are literals. We can reduce F to an instance of FINDCOLLABORATION$_D$ as follows.

First, we define the sets $Lenders$ and $Borrowers$. The set of *L*enders has n lenders for each variable in F. We define a function $\varphi : Lenders \to \sigma$ such that $\varphi(i) = x_i$, to

denote the relation between the lenders and the variables, $Lenders = \{1, ..., n\}$. The set of *B*orrowers is defined for each clause in F, $Borrowers = \{n{+}1, ..., n{+}p\}$. So there is a 1-1 mapping between lenders and variables, and between borrowers and clauses.

We define $\bar{l} = 2 * |\sigma| = 2n$, and a mapping *step*$(x_i)$ from literal x_i, (resp., $\neg x_i$) to time steps such that *step*$(x_i) = i$, (resp., *step*$(\neg x_i) = n + i$).

Given a literal t in F, we denote by $occ(t)$ the number of clauses of F which contain t. Without loss of generality, we assume that no literal is contained twice in any clause. Using $occ(t)$ we define a function *rNum*$: \{1, \ldots, \bar{l}\} \to \{1, \ldots, \overline{m}\}$ where $\overline{m}$ is a positive integer explained below; *rNum* associates each time step (i.e., each literal) with a number of robots.

Intuitively, for each clause $c \in F$ containing literal t, *rNum*(*step*(t)) robots must be transferred to satisfy c. To achieve this, we define *rNum* such that for each time step u, the number of robots that must be transferred is larger than the total number of robots that can be transferred before u, i.e., larger than the number of robots in all previous steps multiplied by their respective occurrence counts of associated literals:

$$\begin{aligned} &\textit{rNum}(1) = 1 \\ &\textit{rNum}(u) = 1 + \textstyle\sum_{i=1}^{u-1} \textit{rNum}(i) \cdot \textit{occ}(\textit{step}^{-1}(i)) \quad \text{for} \quad 1 < u \leq \bar{l} \end{aligned}$$

The constant $\overline{m}$ is the maximum number of robots that can be given at the latest time step, i.e., the largest value of *rNum* for a given 3-SAT instance. This value is $\overline{m} = \textit{rNum}(\textit{step}(\neg x_n)) \cdot \textit{occ}(\neg x_n)$; by eliminating the definition of *rNum* from the formula, we obtain the following equation

$$\overline{m} = \big(\textit{occ}(x_1){+}1\big) \cdot \ldots \cdot \big(\textit{occ}(x_n){+}1\big) \cdot \big(\textit{occ}(\neg x_1){+}1\big) \cdot \ldots \cdot \big(\textit{occ}(\neg x_{n-1}){+}1\big) \cdot \textit{occ}(\neg x_n).$$

Since a literal may occur in at most p clauses, $\overline{m} = \mathcal{O}(p^{2n})$ which is exponential in the input size. This is not a problem as the value $\overline{m}$ can be computable in polynomial time and represented in linear space, moreover our reduction never requires to explicitly represent $\overline{m}$ functions: $Lend\_earliest$ is defined on three intervals per lender and $Borrow\_latest$ is defined on four intervals per borrower, hence a polynomial time reduction.

We define $Delay(i, j) = 0$, for every $i \in Lenders, j \in Borrowers$.

For each lender i, we define $Lend\_earliest_m(i)$ as follows:

$$Lend\_earliest_m(i) = \begin{cases} \textit{step}(\varphi(i)) & \text{if } 1 \leq m \leq \textit{rNum}(\textit{step}(\varphi(i))) \cdot \textit{occ}(\varphi(i)) \\ \textit{step}(\neg\varphi(i)) & \text{if } \textit{rNum}(\textit{step}(\varphi(i))) \cdot \textit{occ}(\varphi(i)) < m \text{ and} \\ & \quad m \leq \textit{rNum}(\textit{step}(\neg\varphi(i))) \cdot \textit{occ}(\neg\varphi(i)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Intuitively, each lender i can lend up to *rNum*(*step*$(\varphi(i)))\cdot$*occ*$(\varphi(i))$ robots with earli-

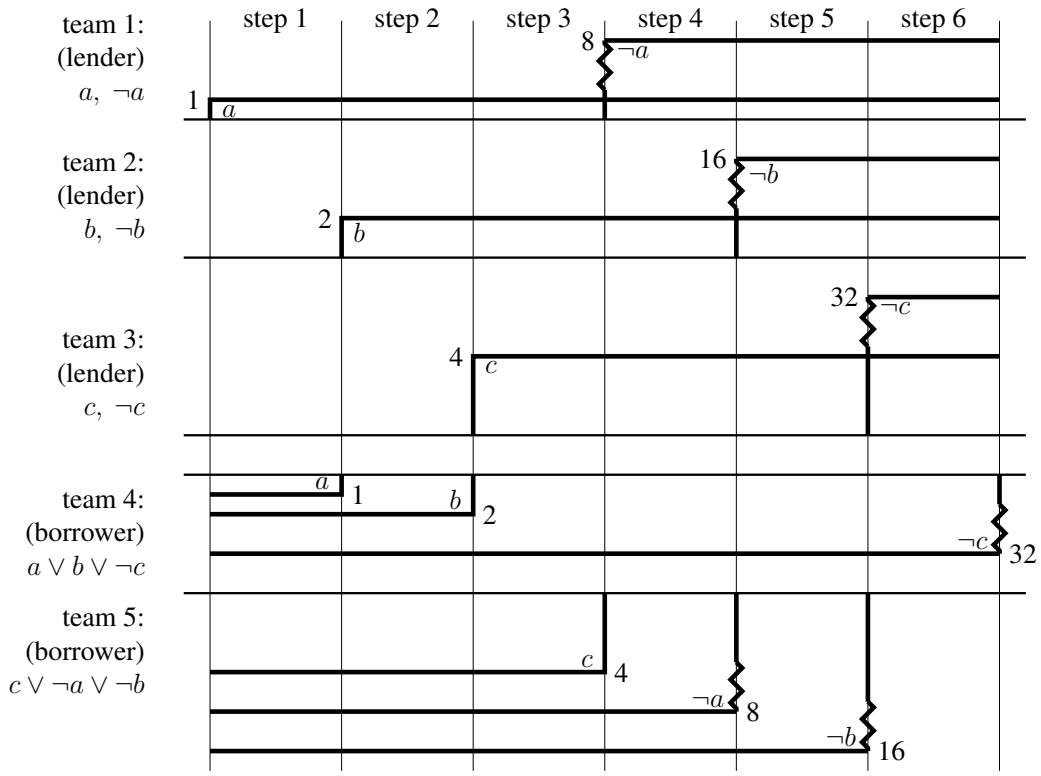


Figure 5.4: Example hardness reduction for 3-SAT formula $F_1 = (a \vee b \vee \neg c) \wedge (c \vee \neg a \vee \neg b)$

est time step *step*$(\varphi(i))$ or it can lend up to *rNum*(*step*$(\neg\varphi(i))$·*occ*$(\neg\varphi(i))$ robots with earliest time step *step*$(\neg\varphi(i))$.

For each clause $c_j = (t_{j,1}, t_{j,2}, t_{j,3})$ in F, without loss of generality, we assume that *step*$(t_{j,1}) \leq$ *step*$(t_{j,2}) \leq$ *step*$(t_{j,3})$, and, for each borrower $j{+}n$, we define $Borrow\_latest_m(j{+}n)$ as follows:

$$Borrow\_latest_m(j{+}n) = \begin{cases} \text{undefined} & \text{if} \quad 1 \leq m < \textit{rNum}(\textit{step}(t_{j,1})) \\ \textit{step}(t_{j,1}) & \text{if} \quad \textit{rNum}(\textit{step}(t_{j,1})) \leq m < \textit{rNum}(\textit{step}(t_{j,2})) \\ \textit{step}(t_{j,2}) & \text{if} \quad \textit{rNum}(\textit{step}(t_{j,2})) \leq m < \textit{rNum}(\textit{step}(t_{j,3})) \\ \textit{step}(t_{j,3}) & \text{if} \quad \textit{rNum}(\textit{step}(t_{j,3})) \leq m \leq \overline{m} \end{cases}$$

Intuitively, each borrower $j{+}n$ corresponding to clause c_j needs to borrow at least the number of robots associated with at least one literals in c_j, at the latest time step that is associated with that particular literal.

Example 5.6. Figure 5.4 shows an example reduction from 3-SAT formula $F_1 = (a \vee b \vee \neg c) \wedge (c \vee \neg a \vee \neg b)$. Bold lines indicate $Lend\_earliest$ and $Borrow\_latest$ functions, e.g., lender 2, corresponding to variable b has $Lend\_earliest_{16}(2) = 5$. Numbers given next to bold lines indicate values of *rNum* for the respective step, e.g., *rNum*$(5) = 16$. Note that *occ*$(t) = 1$ for every literal t, hence *rNum*(*step*$(t)) = 2^{\textit{step}(t)}$.

Note that the reduction from 3-SAT to FINDCOLLABORATION$_D$ can be done in time

polynomial in the size of the input formula. Let us prove that this is a correct reduction: F is satisfiable iff there is an $\overline{ml}$-collaboration between $Lenders$ and $Borrowers$ with at most $\overline{m}$ robot transfers and at most in $\bar{l}$ steps defined above.

Hardness: SAT $\rightarrow$ collaboration Let I be an interpretation mapping σ to truth values such that this assignment satisfies F. Here and in the following we denote an interpretation I by the set of atoms in σ whose values are mapped to true.

We define the collaboration function

$$f : Lenders \times Borrowers \rightarrow \{0, ..., \bar{l}\} \times \{0, ..., \overline{m}\}$$

as follows:

- for every variable $s \in I$ and for every borrower $j+n$,

  $$f(\varphi^{-1}(s), j+n) = (\textit{step}(s), \textit{rNum}(\textit{step}(s)))$$

  where clause c_j contains s;

- for every variable $s \notin I$ and for every borrower $j + n$,

  $$f(\varphi^{-1}(s), j+n) = (\textit{step}(\neg s), \textit{rNum}(\textit{step}(\neg s)))$$

  where clause c_j contains $\neg s$.

Example 5.7 (ctd)**.** Interpretation $I_1 = \{a, c\}$ satisfies F_1 and induces the following collaboration f_1: $f_1(1, 4) = (1, 1)$, $f_1(2, 5) = (5, 16)$, $f_1(3, 5) = (3, 4)$.

We can now show that f, as obtained above from I, indeed satisfies all conditions of Def. 1, i.e., it is an $\overline{ml}$-collaboration.

- Def. 1(a): I satisfies each clause in F. For every borrower j corresponding to clause $c_j = (t_{j,1}, t_{j,2}, t_{j,3})$, let $t_{j,k}$ be a literal in c_j satisfied by I. Take any borrower $j + n$. By our construction of f, there is a lender $i_k = \varphi^{-1}(var(t_{j,k}))$ such that $f(i_k, j + n) = (\textit{step}(t_{j,k}), \textit{rNum}(\textit{step}(t_{j,k})))$ where $\textit{step}(t_{j,k}) \leq \bar{l}$ and $\textit{rNum}(\textit{step}(t_{j,k})) \leq \overline{m}$. Take $m = \textit{rNum}(\textit{step}(t_{j,k})) \leq \overline{m}$. Then the following hold:

  $$\max\{\textit{step}(t_{j,k})\} \leq \textit{step}(t_{j,k}) = \textit{Borrow\_latest}_m(j+n)$$
  $$m \leq \textit{rNum}(\textit{step}(t_{j,k})).$$

  Hence condition (a) holds.

- Def. 1(b): Take any lender i (corresponding to variable $\varphi(i)$). Consider two cases:

– Case 1: $\varphi(i) \in I$. Lender i has cooperations with borrowers $n{+}j_1, .., n{+}j_q$, corresponding to clauses $c_{j_1}, ..., c_{j_q}$ which contain $\varphi(i)$. Note that $q \leq \mathit{occ}(\varphi(i))$ and $q = \mathit{occ}(\varphi(i))$ if and only if $\varphi(i)$ does not occur multiple times in any clause. The construction of f is as follows:

$$\begin{aligned} f(i, n{+}j_1) &= (\mathit{step}(\varphi(i)), \mathit{rNum}(\mathit{step}(\varphi(i)))) \\ &\vdots \\ f(i, n{+}j_q) &= (\mathit{step}(\varphi(i)), \mathit{rNum}(\mathit{step}(\varphi(i)))) \end{aligned}$$

where $\mathit{step}(\varphi(i)) = l_1 = \cdots = l_s \leq \bar{l}$, and $\mathit{rNum}(\mathit{step}(\varphi(i))) = u_1 = \cdots = u_s \leq \overline{m}$. Take $m = \mathit{rNum}(\mathit{step}(\varphi(i))) \cdot \mathit{occ}(\varphi(i)) \leq \overline{m}$. Then the following hold:

$$\begin{aligned} \mathit{Lend\_earliest}_m(i) = \mathit{step}(\varphi(i)) &\leq \min\{\mathit{step}(\varphi(i))\} \\ m &\geq \mathit{rNum}(\mathit{step}(\varphi(i))) \cdot q. \end{aligned}$$

– Case 2: $\varphi(i) \notin I$. Lender i has cooperations with borrowers $n{+}j_1, .., n{+}j_q$, corresponding to clauses $c_{j_1}, ..., c_{j_q}$ which contain $\neg\varphi(i)$. Note that $q = \mathit{occ}(\neg\varphi(i))$ and $q = \mathit{occ}(\neg\varphi(i))$ if and only if $\neg\varphi(i)$ does not occur multiple times in any clause. The construction of f is as follows:

$$\begin{aligned} f(i, n{+}j_1) &= (\mathit{step}(\neg\varphi(i)), \mathit{rNum}(\mathit{step}(\neg\varphi(i)))) \\ &\vdots \\ f(i, n{+}j_q) &= (\mathit{step}(\neg\varphi(i)), \mathit{rNum}(\mathit{step}(\neg\varphi(i)))) \end{aligned}$$

where $\mathit{step}(\neg\varphi(i)) = l_1 = \cdots = l_s \leq \bar{l}$ and $\mathit{rNum}(\mathit{step}(\neg\varphi(i))) = u_1 = \cdots = u_s \leq \overline{m}$. Take $m = \mathit{rNum}(\mathit{step}(\neg\varphi(i))) \cdot \mathit{occ}(\neg\varphi(i)) \leq \overline{m}$. Then the following hold:

$$\begin{aligned} \mathit{Lend\_earliest}_m(i) = \mathit{step}(\neg\varphi(i)) &\leq \min\{\mathit{step}(\neg\varphi(i))\} \\ m &\geq \mathit{rNum}(\mathit{step}(\neg\varphi(i))) \cdot q. \end{aligned}$$

Hence condition (b) holds.

Therefore, a function f obtained as shown above from a satisfying assignment I of F is a collaboration according to Def. 1.

Hardness: collaboration $\rightarrow$ SAT

Let f be a collaboration function defined via the reduction explained above. We need to show that there is an interpretation that satisfies F. Without loss of generality, we assume that $\mathit{Delay}(i, j) = 0$ for all i, j, and do not mention delay in the following.

Since f is a collaboration function, it satisfies the conditions in Def. 1.

Given f and a borrower $j{+}n$ corresponding to clause c_j, we say that *borrower $j{+}n$ can complete its task with respect to a literal $t \in c_j$* iff the borrower borrows at least *rNum*(*step*(t)) robots up to time step *step*(t) (inclusive) and there is no $t' \in c_j$ with *step*(t)<*step*(t') such that the borrower borrows *rNum*(*step*(t')) or more robots at a step after *step*(t).

Intuitively, a borrower can complete its task with respect to literal $t \in c_j$ iff its task can be completed by obtaining robots until *step*(t) and this is not true for another literal after that step. Given a collaboration function f, per definition of collaboration each borrower can complete its task with respect to *exactly one* literal.

Example 5.8 (ctd)**.** In collaboration f_1, borrower 4 can complete its task with respect to a and with respect to no other literal, borrower 5 can complete its task with respect to $\neg b$, and not with respect to c because *step*(c)<*step*$(\neg b)$.

Towards proving the result, we introduce two lemmas.

Lemma 5.9. *If borrower $j{+}n$, corresponding to clause c_j, can complete its task with respect to literal $t_u \in c_j$, then borrower $j{+}n$ borrows at least one robot from lender $\varphi^{-1}(var(t_u))$ at step step(t_u) and that robot cannot be lent before step step(t_u).*

Intuitively, this lemma states that a borrower which can complete its task with respect to a certain literal t_u always borrows at least one robot from the lender corresponding to $var(t_u)$, and that robot is exchanged at the step corresponding to t_u, i.e., at the step corresponding to the correct (with respect to polarity of the variable) literal.

Example 5.10 (ctd)**.** Given f_1 we saw that borrower 5 can complete with respect to $\neg b$. According to Lemma 5.9, borrower 5 receives at least 1 robot at *step*$(\neg b) = 5$ which is true as $f_1(2, 5) = (5, 16)$. Intuitively, Lemma 5.9 holds because borrower 5 requires 16 robots to complete its task with respect to $\neg b$ whereas lenders can only lend 15 robots in total during steps $1 \ldots 4$. Therefore, borrower 5 must receive at least 1 of the 16 robots that can be given at step 5, and this robot is in addition to 2 robots that could potentially be given at step 2. (See also Lemma 5.11 which shows that these 2 robots cannot be given in that case.)

Proof of Lemma 5.9.

- Case 1: t_u is a positive literal.

  Borrower $j{+}n$ can complete its task with respect to $t_u \in c_j$, so it borrows at least *rNum*(*step*(t_u)) robots with the latest time step *step*(t_u).

  Borrower $j{+}n$ can cooperate with lenders $\varphi^{-1}(var(t_1)), ..., \varphi^{-1}(var(t_u))$.

  Towards a contradiction, assume that borrower $j{+}n$ does not borrow any robots from lender $\varphi^{-1}(var(t_u))$ at step *step*(t_u). Then there exists a collaboration func-

tion f with:

$$f(\varphi^{-1}(var(t_\alpha)), j{+}n) = (l_\alpha, m_\alpha)$$
$$\vdots$$
$$f(\varphi^{-1}(var(t_\beta)), j{+}n) = (l_\beta, m_\beta)$$

where $\textit{step}(t_1) \leq l_\alpha \leq l_\beta < \textit{step}(t_u)$ and $\textit{rNum}(\textit{step}(t_u)) \leq (m_\alpha + \cdots + m_\beta)$ and $\max\{l_\alpha, \ldots, l_\beta\} \leq \textit{step}(t_u)$. Note that this function excludes borrowing at step $\textit{step}(t_u)$ hence it excludes

$$f(\varphi^{-1}(var(t_u)), j{+}n) = (\textit{step}(t_u), m) \text{ for every } m.$$

Therefore, the maximum number of robots that borrower $j{+}n$ can borrow with f is

$$\begin{gathered}\textit{rNum}(\textit{step}(t_1))\cdot\textit{occ}(t_1) + \cdots + \textit{rNum}(\textit{step}(t_{u-1}))\cdot\textit{occ}(t_{u-1}) = \\ = \textstyle\sum_{i=1}^{u-1} \textit{rNum}(\textit{step}(t_i))\cdot\textit{occ}(t_i) = \textit{rNum}(\textit{step}(t_u)) - 1\end{gathered}$$

which is exactly one robot less than borrower $j{+}n$ needs to be able to complete with respect to t_u.

We have reached a contradiction: there cannot be a collaboration function that satisfies condition (a) of Def. 1 without lender $\varphi^{-}(var(t_u))$ lending at least one robot to borrower j at step $\textit{step}(t_u)$.

- Case 2: t_u is a negative literal.

  If borrower $j{+}n$, corresponding to clause c_j, can complete its task with respect to literal $\neg t_u \in c_j$, then it borrows at least $\textit{rNum}(\textit{step}(\neg t_u))$ robots with the latest time step $\textit{step}(\neg t_u)$.

  Assume that borrower $j{+}n$ borrows m robots from lender $\varphi^{-1}(var(t_u))$. Then,

  $$Lend\_earliest(\varphi^{-1}(var(t_u)))_m = \textit{step}(t_u).$$

  By our construction of $Lend\_earliest$, we have $m \leq \textit{rNum}(\textit{step}(t_u))\cdot\textit{occ}(t_u)$. With this assumption, there exists a collaboration function f with:

  $$f(\varphi^{-1}(var(t_\alpha)), j{+}n) = (l_\alpha, m_\alpha)$$
  $$\vdots$$
  $$f(\varphi^{-1}(var(t_\beta)), j{+}n) = (l_\beta, m_\beta)$$
  $$f(\varphi^{-1}(var(t_u)), j{+}n) = (l_\gamma, m)$$

  where $\textit{step}(t_1) \leq l_\alpha \leq l_\beta \leq l_\gamma \leq \textit{step}(\neg t)$ and $\textit{rNum}(\textit{step}(\neg t)) \leq (m_\alpha + \cdots +$

$m_\beta{+}m$) and $\max\{l_\alpha, \ldots, l_\beta, l_\gamma\} \leq$ *step*$(\neg t)$.

The maximum number of robots borrower $j{+}n$ can borrow can be computed as follows:

$$\begin{aligned} &\textit{rNum}(\textit{step}(t_1))\cdot\textit{occ}(t_1)+\ldots+\textit{rNum}(\textit{step}(t_u))\cdot\textit{occ}(t_u)+\ldots \\ &\qquad\qquad \ldots+\textit{rNum}(\textit{step}(\neg t_{u-1}))\cdot\textit{occ}(\neg t_{u-1}) = \\ &= \textstyle\sum_{i=1}^{u-1} \textit{rNum}(\textit{step}(t_i))\cdot\textit{occ}(t_i) = \textit{rNum}(\textit{step}(\neg t_u)) - 1 \end{aligned}$$

We have reached a contradiction: there cannot be a collaboration function that satisfies condition (a) of Def. 1 without lender $\varphi^-(var(t_u))$ lending at least one robot to borrower j at the time step where the lender has the earliest possibility to lend this robot, i.e., at step *step*(t_u).

□

Lemma 5.11. *If borrower $j{+}n$, corresponding to clause c_j, can complete its task with respect to positive literal $t \in c_j$ (resp., with respect to negative literal $\neg t \in c_j$) then no borrower can complete its task with respect to negative literal $\neg t$ (resp., with respect to positive literal t).*

Intuitively, this lemma states that if a borrower can complete its task with respect to a certain literal t, no other borrower can complete its task with respect to the negation of literal t. In terms of truth values and clause satisfiability, Lemma 5.11 shows that a collaboration corresponds to a *consistent* set of literals satisfying all clauses.

Example 5.12 (ctd)**.** Given f_1 borrower 5 can complete with respect to $\neg b$, Lemma 5.11 states that no borrower can complete with respect to b. Intuitively, this holds because lender 2 provides at least one of 16 possible robots at step 5. Therefore, it cannot give robots already at step 2.

Proof of Lemma 5.11.

- Case 1: borrower $j{+}n$ can complete its task with respect to positive literal $t \in c_j$.

  By Lemma 5.9, borrower $j{+}n$ borrows at least one robot from lender $\varphi^{-1}(var(t))$ at step *step*(t).

  Since lender $\varphi^{-1}(var(t))$ lends robots with earliest step *step*(t), by $Lend\_earliest$ function, the lender can lend at most *rNum*$($*step*$(t))\cdot$*occ*(t) robots.

  Let borrower $j'{+}n$ be a borrower corresponding to a clause $c_{j'}$ which contains $\neg t$. To complete its task by borrowing robots from lender $\varphi^{-1}(var(t))$, it needs to borrow at least *rNum*$($*step*$(\neg t))$ robots. However, *rNum*$($*step*$(\neg t))>$*rNum*$($*step*$(t))\cdot$*occ*(t).

  Therefore, lender $\varphi^{-1}(var(t))$ cannot be the lender that satisfies borrower $j'{+}n$. In other words, clause $c_{j'}$ cannot be satisfied by literal $\neg t$.

- Case 2: borrower $j{+}n$ can complete with respect to negative literal $t \in c_j$.

  By Lemma 5.9, borrower $j'{+}n$ borrows $m \geq 1$ robots from lender $\varphi^{-1}(var(t))$ at step *step*$(\neg t)$, where

  $$Lend\_earliest(\varphi^{-1}(var(t))) = \textit{step}(\neg t).$$

  Since lender $\varphi^{-1}(var(t))$ lends a number of robots with earliest time step as above, this lender cannot lend any robots before *step*$(\neg t)$. Therefore, lender $\varphi^{-1}(var(t))$ cannot be the lender that satisfies a borrower $j{+}n$ with respect to literal $t \in c_j$. In other words, clause c_j cannot be satisfied by literal t.

  □

We can now prove that, if f is a $\overline{ml}$-collaboration then F is satisfiable.

As f is a collaboration function it satisfies the conditions in Def. 1. By condition (a) every borrower $j{+}n$ can complete its task with respect to some literal $t \in c_j$. Given f, let J be the set of all literals t s.t. some borrower can complete with respect to t. Due to Lemma 5.11 no two borrowers complete their respective tasks with respect to complementary literals x and $\neg x$ for some variable x. Hence J does not contain both positive and negative literals for any variable; it is a consistent set of literals.

Each borrower $j{+}n$ corresponding to clause c_j can complete its task with respect to some literal $t \in J$ and $t \in c$, therefore all clauses of F are satisfied by J. Therefore, given a collaboration function f, the consistent set of literals J corresponds to a (unique) satisfying truth assignment I for F. □

5.6 Finding a Coordination of Teams in ASP

Since the computational problem is intractable, ASP is suitable for solving it: Deciding whether a program in ASP has an answer set is NP-complete [9]. We model FIND_COLLABORATION$_n$ as follows. The input is represented by a set of facts, using atoms of the form *delay*(i, j, l), *lend_earliest*(i, m, l, x), and *borrow _latest*(j, m, l, x) where $1 \leq x \leq n, i \in$ *Lenders*$_x$, $j \in$ *Borrowers*$_x$, $m \leq \overline{m}, l \leq \overline{l}$.

Condition (a) of Definition 5.1 is defined for each borrower j as follows:

condition_borrower$(j, x) \leftarrow$
  borrow_latest(j, m, l, x),
  sum$\langle\{u : f(i, j, l_1, u, x), i \in$ *Lenders*$_x, l_1 \leq \overline{l}, u \leq \overline{m}\}\rangle \geq m$,
  max$\langle\{l_1{+}t : f(i, j, l_1, u, x), delay(i, j, t), i \in$ *Lenders*$_x, l_1 \leq \overline{l}, u \leq \overline{m}\}\rangle \leq l$

where $1 \leq x \leq n, j \in$ *Borrowers*$_x, l \leq \overline{l}, m \leq \overline{m}$. The second line of the rule above describes that team j needs m robots of type x by step l. The third and fourth lines express

that the number of robots lent to the borrower team j is at least m; the last two lines express the latest time step l that team j borrows a robot of type x. Similarly, we define condition (b).

$$
\begin{aligned}
&\textit{condition\_lender}(i,x) \leftarrow \\
&\quad \textit{lend\_earliest}(i,m,l,x), \\
&\quad \textit{sum}\langle\{u : f(i,j,l_1,u,x), j \in \textit{Borrowers}_x, l_1 \leq \bar{l}, u \leq \overline{m}\}\rangle \leq m, \\
&\quad \textit{max}\langle\{l_1 : f(i,j,l_1,u,x), \textit{delay}(i,j,t), j \in \textit{Borrowers}_x, l_1 \leq \bar{l}, u \leq \overline{m}\}\rangle \geq l
\end{aligned}
$$

We define an $\overline{nml}$-collaboration f, by atoms of the form $f(i,j,l,u,x)$, describing $f(i,j,x) = (l,u)$, by first "generating" partial functions f:

$$
\begin{aligned}
&\{f(i,j,l,u,x) : l \leq \bar{l}, u \leq \overline{m}\}1 \leftarrow \\
&\qquad (1 \leq x \leq n, i \in \textit{Lenders}_x, j \in \textit{Borrowers}_x)
\end{aligned}
$$

and then ensuring that the borrowers can borrow exactly one type of robot and that lenders can lend at most one type of robots:

$$
\begin{aligned}
&\leftarrow \textit{not}\ 1\{fB(j,x) : 1 \leq x \leq n\}1 \qquad (j \in \textit{Borrowers}) \\
&\leftarrow 2\{fL(i,x) : 1 \leq x \leq n\} \qquad (i \in \textit{Lenders}))
\end{aligned}
$$

where $fB(j,x)$ and $fL(i,x)$ are projections of f onto j,x and i,x, respectively. Finally we "eliminate" the partial functions that do not satisfy conditions (a) and (b) of Definition 5.1:

$$
\begin{aligned}
&\leftarrow \textit{not condition\_borrower}(j,x), fB(j,x) \qquad (j \in \textit{Borrowers}_x, 1 \leq x \leq n) \\
&\leftarrow \textit{not condition\_lender}(i,x) \qquad (i \in \textit{Lenders}_x, 1 \leq x \leq n)
\end{aligned}
$$

With the ASP formulation above, an ASP solver can find an $\overline{nml}$-collaboration. The formulation represented to CLASP is shown in Figure 5.5.

5.7 Decoupling Plans for an Optimal Global Plan

Once a coordination of the teams is found for an optimal value of $\bar{l}$, the necessary information of which team lends to (or borrow from) which other team and when is passed to each team. Then each team computes an optimal local plan whose length is less than or equal to $\bar{l}$. The optimal global plan is found by decoupling the optimal local plans.

In this section, we discuss the correctness of our approach.

Lemma 5.13. *Under C1, C2 and C3, for every $\bar{l}, \overline{m}$ and n, there is an $\overline{nml}$-collaboration of lenders and borrowers if and only if there is a collaboration of lenders and borrowers.*

```
% team I is a lender for robots of type X
lenderx(I,X) :- lend(I,M,L,X).
% team J is a borrower for robots of type X
borrowerx(J,X) :- borrow(J,M,L,X).

% if a team is a lender/borrower for a type of robot
% then it is a lender/borrower.
lender(I) :- lenderx(I,X).
borrower(J) :- borrowerx(J,X).

% earliest time that lender I can lend M robots of type X is step L
lend_earliest(I,M,L,X) :- L #min[lend(I,M,L1,X)=L1] L,num(M),step(L),
            lenderx(I,X),type(X).
% latest time that borrower J needs to borrow M robots of type X is step L
borrow_latest(J,M,L,X) :- L=#max[borrow(J,M,L1,X)=L1],num(M),step(L),
            borrowerx(J,X),type(X).

% lender I lends U robots of type X to borrower J at step L
{f(I,J,L,U,X) : num(U): step(L)}1:- lenderx(I,X),borrowerx(J,X),type(X).

%projection of f onto J and X.
fB(J,X):-f(I,J,L,U,X).
%projection of f onto I and X.
fL(I,X):-f(I,J,L,U,X).

% a borrower team can only borrow one type of robots
:- not 1{fB(J,X) : type(X)}1, borrower(J).
% lender team can lend at most one type of robots
:- 2{fL(I,X) : type(X)}, lender(I).

% a borrower team does not borrow fewer robots than it needs
condition_borrower(J,X) :-
     M [f(I,J,L1,U,X)=U:lenderx(I,X):step(L1):num(U)],
     #max[f(I,J,L1,U,X)=L1+T:lenderx(I,X):num(U):step(L1):delay(I,J,T)]L,
     borrow_latest(J,M,L,X),borrowerx(J,X),step(L),num(M),type(X).
:- not condition_borrower(J,X),borrowerx(J,X),fB(J,X).

% a lender team does not lend more robots than it can
condition_lender(I,X) :-
     [f(I,J,L1,U,X)=U:borrowerx(J,X):step(L1):num(U)] M ,
     L #min[f(I,J,L1,U,X)=L1:borrowerx(J,X):num(U):step(L1)],
     lend_earliest(I,M,L,X),lenderx(I,X),step(L),num(M),type(X).
:- not condition_lender(I,X),lenderx(I,X).
```

Figure 5.5: ASP formulation of the coordination problem

Proof. ($\Rightarrow$) Assume that there is an $n\overline{ml}$-collaboration of lenders and borrowers under C1, C2 and C3. The $n\overline{ml}$-collaboration tells which lender should lend how many number of robots of which type to which borrower at which step. Since the workspaces of the teams are separate and the teams execute their local plans according to the $n\overline{ml}$-collaboration there is no collision of robots during the execution of the plans. Therefore the $n\overline{ml}$-collaboration is a collaboration of lenders and borrowers.

($\Leftarrow$) Assume that there is a collaboration of lenders and borrowers for some $\bar{l}, \overline{m}$ and n under C1, C2 and C3. Since this collaboration satisfies C2 and C3, it means that lending/borrowing is done in a single batch and the teams lend/borrow robots of the same type.

Let team i be a lender that lends $m_1, ..., m_r$ robots of type x to teams $j_1, ..., j_r$ at steps $l_1, ..., l_r$, respectively. Team i can finish its task in at most $\bar{l}$ steps while it lends $m_1 + ... + m_r = m$ robots. So if *Lend_earliest*$_{m,x}(i) = l$, then $l \leq \min\{l_1, ..., l_r\}$ will hold. Similarly, for a borrower team j that borrows $m_1, ..., m_r$ robots of type x from teams $i_1, ..., i_r$ at steps $l_1, ..., l_r$, we can conclude that if *Borrow_latest*$_{m,x}(j) = l$ for $m = m_1 + ... + m_r$, then $l \geq \max\{l_1, ..., l_r\}$. Also if there is delay between teams we can determine it by taking the time difference of the step l' that team i lends robots to team j and the step l'' that team j borrows robots from team i.

Therefore if there is a collaboration then it satisfies the conditions for an $n\overline{ml}$-collaboration. So there is an $n\overline{ml}$-collaboration under C1, C2 and C3. □

By incrementing $\bar{l}$ one by one until k, we can find the optimal value for the plan length. After finding an optimal plan length for the teams, each team computes its local plan regarding the $n\overline{ml}$-collaboration.

Lemma 5.14. *The union of local plans is a global plan.*

Proof. Each team computes its local plan with the information of at which step how many robots they are suppose to lend/borrow, stated by the $n\overline{ml}$-collaboration. Since the constraints in the domain description rules out the actions which would result in collisions, during the execution of the local plans there won't be any conflicts.

By taking the union of the local plans, each team will concurrently perform their actions at each step. At the lending steps, the robots that are lent will move to the bench, which is a seperate area from the workspaces. At the borrowing steps, the robots will move from the bench to the borrower's workspace. The delay between two workspaces are also considered, so at the borrowing step for the team, the robots will be available at the bench to enter the workspace.

Since the workspaces of the teams are seperate, during the execution of the plans there won't be any conflicts between teams. Therefore the union of the local plans can be determined as the global plan of the teams. □

5.8 Algorithm for Finding an Optimal Global Plan

In this section, we describe our algorithm (Algorithm 5.1) to find an optimal global plan of r teams, given the action domain description $\mathcal{D}$, maximum plan length k, number of types of robots n, maximum number of robots m_x that can be transferred for type $x, x \leq n$ and the planning problems $\mathcal{P}_1, \ldots, \mathcal{P}_r$ for each team with initial states and goals.

To find an optimal global plan, first the roles of each team (lender or borrower) are identified by asking queries of form Q1 5.2. After that, by gathering yes/no answers to queries of the forms Q2 and Q3 from every lender/borrower team, the earliest lend times and the latest borrow times are inferred (Algorithms 5.3 and 5.4). Then based on these earliest and latest times of robot transfers, and considering delays of robot transfers, an optimal coordination among the teams is computed as described in Section 5.6. Once such a coordination is found for an optimal global plan of length, necessary information about this coordination is conveyed to each lender or borrower as constraints. After that, each team computes its own optimal local hybrid plan (as described in Chapters 3 and 4) whose length is at most $\bar{l}$, and which satisfies the coordination constraints conveyed to by the mediator. Finally, these optimal local hybrid plans are decoupled for an optimal global plan as described in Section 5.7. According to Algorithm 5.1 linear search is used to find the optimal value of $\bar{l}$.

The function DETERMINEROLES (described in Algorithm 5.2) determines the role of each team for length $\bar{l}$ given the action domain description $\mathcal{D}$ and the planning problems of each team. This function asks each team i for a plan of length $\bar{l}$ with domain $\mathcal{D}$ and planning problem $\mathcal{P}_i$. If the team can find a plan, then there is a possibility that the team may be able to lend some of its robots and still reach its goal state in $\bar{l}$ steps, so it is identified as a lender. Otherwise, it is identified as a borrower.

The function GATHERANSWERS_LEND (described in Algorithm 5.3) asks lender teams queries of the form Q2 for every $x \leq n, m \leq m_x$ and $l \leq \bar{l}$, by updating the domain description $\mathcal{D}$ with $\mathcal{D}_{lend,x}$ which is a description of the lending action and additional constraints on only allowing the teams to lend robots of type x only. The constraints are given to avoid undesired lending actions of other type of robots. After collecting the answers, the earliest lend time for each lender team (i.e., the minimum time step l that the team can lend robots, while completing its tasks within $\bar{l}$ steps) is found.

Similarly, for borrowers, GATHERANSWERS_BORROW (described in Algorithm 5.4) asks queries of the form Q3 for every $x \leq n, m \leq m_x$. This function updates the domain description $\mathcal{D}$ now with $\mathcal{D}_{borrow,x}$ which is a description of the borrowing action and additional constraints on only allowing the teams to borrow robots of type x only. The constraints are to avoid undesired borrowing actions of other type of robots. After the answers are collected, the latest borrow time for each borrower team (i.e., the maximum time step l at which the team borrows robots, to complete its tasks within $\bar{l}$ steps) is found.

Algorithm 5.1 FIND_OPTIMAL_GLOBAL_PLAN

Input: An action domain description $\mathcal{D}$, positive integers k, n and $m_x, x \leq n$, r planning problems $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_r$ (one for each team) with initial states $s_1, s_2, ..., s_r$ and goal states $g_1, g_2, ..., g_r$, and a transportation delay t_d

Output: A tuple consisting of, for each team i, a plan $P[i]$ of length at most $\bar{l} \leq k$, team role $role[i]$, lending/borrowing constraints for each team $C[i]$

1: $planfound := false$;
2: $\bar{l} := 0$;
3: **while** $\neg planfound \wedge \bar{l} < k$ **do**
4: $\quad \bar{l} = \bar{l} + 1$;
5: $\quad role \leftarrow$ DETERMINEROLES$(\mathcal{D}, \mathcal{P}_1, ..., \mathcal{P}_r, \bar{l})$;
6: $\quad$ **if** all teams are Borrowers **then**
//There isn't a lender team to help the borrower teams
7: $\quad\quad$ continue;
8: $\quad$ **end if**
9: $\quad$ **if** all teams are Lenders **then**
//Every team can complete on its own
10: $\quad\quad planfound = true$;
11: $\quad\quad$ continue;
12: $\quad$ **end if**
13: $\quad$ **for all** teams i **do**
14: $\quad\quad$ **if** $role[i] = Lender$ **then**
15: $\quad\quad\quad lend_{m,x}[i] \leftarrow$ GATHERANSWERS_LEND$(\mathcal{D}, \mathcal{P}_i, \bar{l}, n, m_1, ..., m_n)$;
16: $\quad\quad$ **else**
17: $\quad\quad\quad borrow_{m,x}[i] \leftarrow$ GATHERANSWERS_BORROW$(\mathcal{D}, \mathcal{P}_i, \bar{l}, n, m_1, ..., m_n)$;
18: $\quad\quad$ **end if**
19: $\quad$ **end for**
20: $\quad$ **if** FIND_COORDINATION$(role, \bar{l}, n, m_1, ..., m_n, lend_{m,x}, borrow_{m,x}, t_d)$ **then**
21: $\quad\quad C \leftarrow$ determine constraints from the coordination;
22: $\quad\quad planfound = true$;
23: $\quad$ **end if**
24: **end while**
25: **for all** teams i **do**
26: $\quad P[i] \leftarrow$ FIND_LOCAL_PLAN$(\mathcal{D}_i, \bar{l}, \mathcal{P}_i, C_i)$;
27: **end for**

Algorithm 5.2 DETERMINEROLES

Input: An action domain description $\mathcal{D}$, r planning problems $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_r$ (one for each team), positive integer $\bar{l}$

Output: team roles, $role$

1: **for all** teams i **do**
2: Ask for a plan of length $\bar{l}$ with domain $\mathcal{D}$ and planning problem $\mathcal{P}_i$
3: **if** answer=success **then**
4: $role[i] = Lender$;
5: **else**
6: $role[i] = Borrower$;
7: **end if**
8: **end for**

Algorithm 5.3 GATHERANSWERS_LEND

Input: An action domain description $\mathcal{D}$, planning problem $\mathcal{P}$, positive integers $\bar{l}, n$ and $m_x, x \leq n$

Output: $lend_{m,x}$ for each $m \leq m_x, x \leq n$

$L_{1,1}, ..., L_{m_1,1}, ..., L_{1,n}..., L_{m_n,n} \leftarrow$ empty sets for lend times of each number of robot type;

1: **for all** robot types x $\leq$ n **do**
2: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_{lend,x}$;
3: **for all** m $\leq m_x$ **do**
4: **for all** l $\leq \bar{l}$ **do**
5: $C \leftarrow$ the constraint of lending m robots of type x before step l;
6: Ask for a plan of length $\bar{l}$ with domain $\mathcal{D}$ and planning problem $\mathcal{P} \cup C$;
7: **if** answer=success **then**
8: $L_{m,x} = L_{m,x} \cup \{l\}$;
9: **end if**
10: **end for**
11: $lend_{m,x} = min\{L_{m,x}\}$;
12: **end for**
13: **end for**

5.9 Demonstrations

We have shown the applicability of our approach with both dynamic simulations using OPENRAVE [12] (with different optimizations of local feasible plans), and with a physical implementation utilizing KuKa youBots and Lego NXT robots controlled over Robot Operating System (ROS). Videos of these implementations are available at `http://cogrobo.sabanciuniv.edu/?p=748`.

A plan sequence from the physical implementation can be seen in Figure 5.6. In this setting, team 1 is responsible for the manufacturing process, whereas team 2 does the painting process on the toys. As the plan execution begins, the worker robots align themselves in front of the assembly line to be able to work on the products that shift on

Algorithm 5.4 GATHERANSWERS_BORROW

Input: An action domain description $\mathcal{D}$, planning problem $\mathcal{P}$, positive integers $\bar{l}, n$ and $m_x, x \leq n$

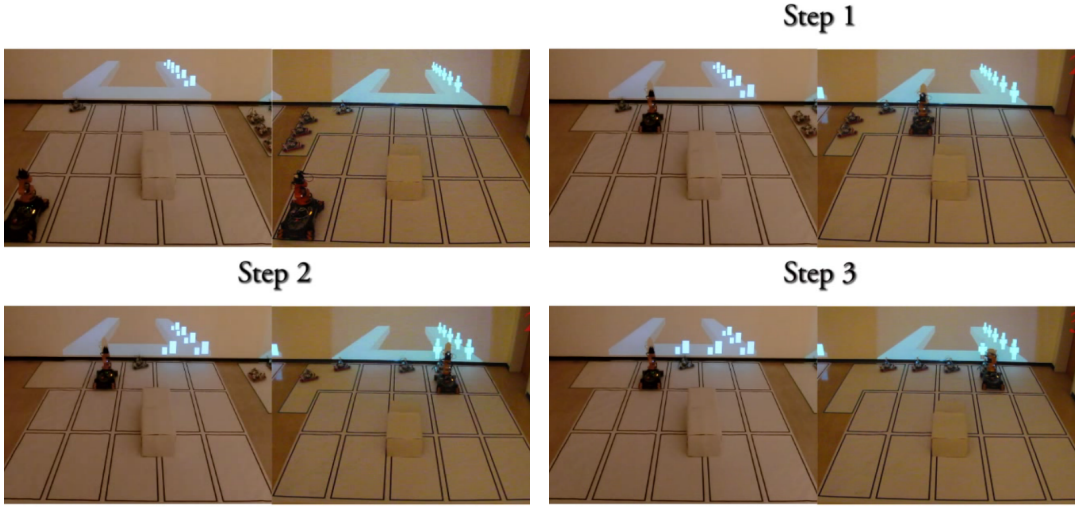
Output: $borrow_{m,x}$ for each $m \leq m_x, x \leq n$

$B_{1,1}, ..., B_{m_1,1}, ..., B_{1,n}..., B_{m_n,n} \leftarrow$ empty sets for borrow times of each number of robot type;

1: **for all** robot types x $\leq$ n **do**
2: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_{borrow,x}$;
3: **for all** m $\leq m_x$ **do**
4: **for all** l $\leq \bar{l}$ **do**
5: $C \leftarrow$ the constraint of borrowing m robots of type x after step l;
6: Ask for a plan of length $\bar{l}$ with domain $\mathcal{D}$ and planning problem $\mathcal{P} \cup C$;
7: **if** answer=success **then**
8: $B_{m,x} = B_{m,x} \cup \{l\}$;
9: **end if**
10: **end for**
11: $borrow_{m,x} = max\{B_{m,x}\}$;
12: **end for**
13: **end for**

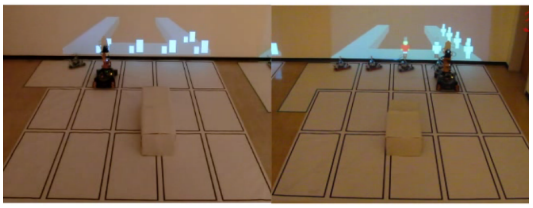
the line to the places in front of the worker robots. For example, at step 3 in team 2, the first workstage of the first product, painting in black, is completed. The chargers also move towards the worker robots to charge them when needed. For example, in step 3, the charger in team 2 dockes to a worker robot, charges it in step 4 and undocks from the worker robot at step 4. At step 12, team 2 lends two dry robots, and the robots move to the bench. The delay time is considered to be 3 steps and while the lent robots move in the bench, the teams continue to execute their plans. At step 15, team 1 borrows the two dry robots and the teams continue to execute their plans until all the products are completed.

Figure 5.6: Snapshots of a plan sequence from the physical implementation

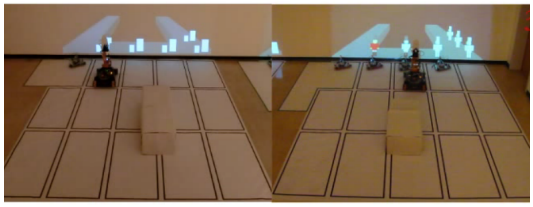


Step 4

Step 5

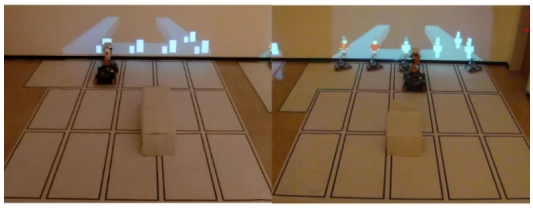


Step 6

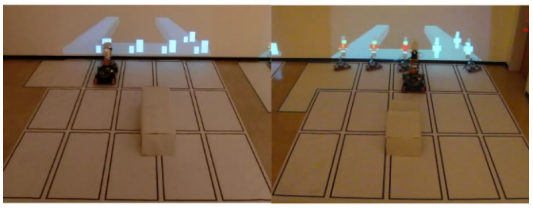


Step 7

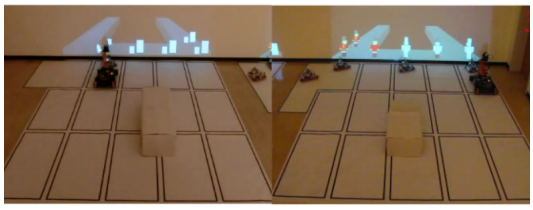
Step 8



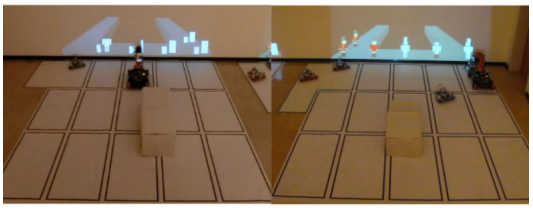
Step 9



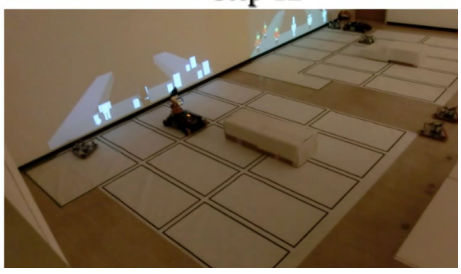
Step 10



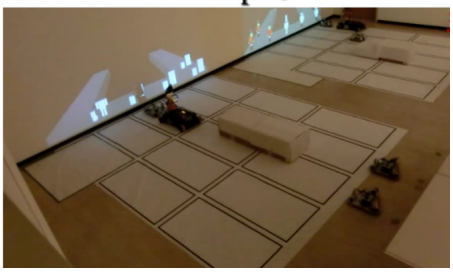
Step 11



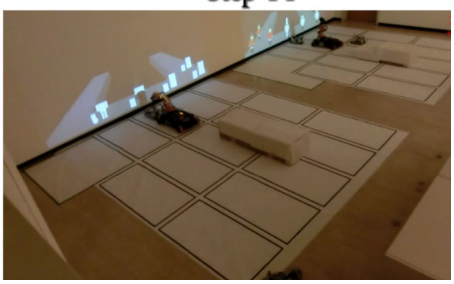
Step 12



Step 13

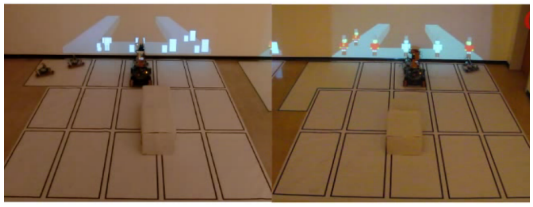


Step 14

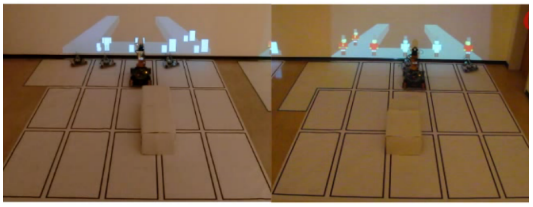


Step 15

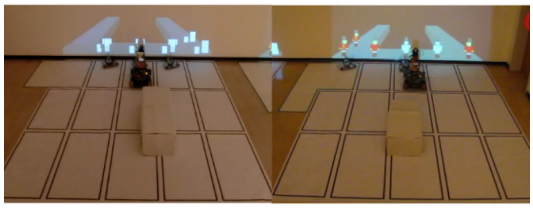
Step 16



Step 17



Step 18



Step 19

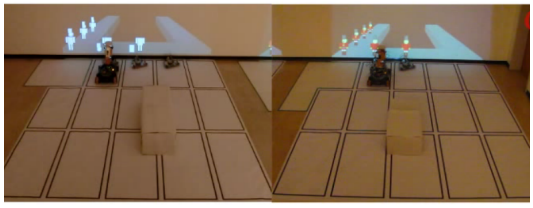
Step 20

Step 21

Step 22

Step 23

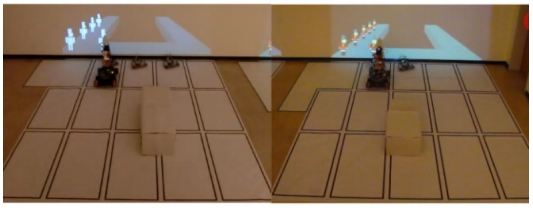
Step 24



Step 25

Step 26

Step 27



Step 28

Step 29

Chapter 6

Experimental Evaluation

We investigated the scalability and usefulness of the proposed optimal global planning method by means of some experiments over the cognitive toy factory domain (Chapter 2).

6.1 Setting

We have performed our experiments on a Linux server with 16 Intel E5-2665 CPU cores (32 threads) with 2.4GHz and 64GB memory (note that our experiments never use more than 300MB).

Algorithms 5.1,5.2,5.3 and 5.4 are implemented in Python. The ASP solver Clasp version 2.1.3 (with Gringo version 3.0.5), with `configuration=handy` as the command line option, is used for answering queries. To solve the collaboration problem, we also use CLASP. To compare the computation times for answering queries, we also performed experiments using CCALC with the multi-threaded SAT-solver MANYSAT (limited to four threads). For the transformation of the domain representation in $\mathcal{C}+$ to ASP we used the tool CPLUS2ASP v1 [7].

First of all, it is important to observe that, since each team is equipped with its own computation unit and the queries are designed in a way that teams do not need to communicate with each other, queries for teams can be trivially parallelized as depicted in Figure 6.1. Two sorts of parallelization are utilized: i) mediator asks sets of queries to teams in parallel and ii) each team takes advantage of multiple cores in its computation unit to process multiple queries simultaneously. Thanks to this parallelization, the total time required to find a coordination for an optimal global plan through a mediator, consists of the time it takes for the mediator to compute a coordination plus the time it takes for the slowest team to answer all queries asked by the mediator: $T_{total} = T_{coord} + T_{query}$. Furthermore, each team can additionally take advantage of multiple core/threads that may exists in its computation unit, to process multiple queries simultaneously. Therefore, we

can analyze the scalability of the proposed approach by investigating T_{coord} and T_{query}, respectively.

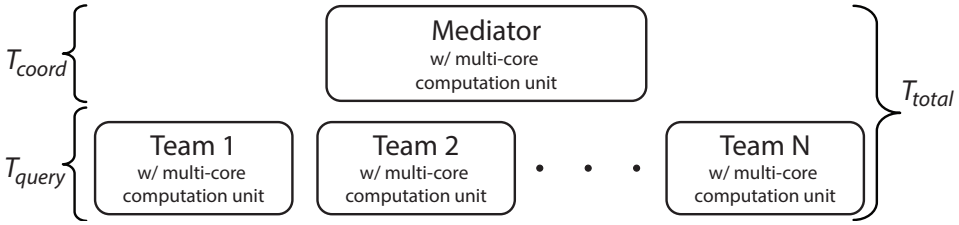


Figure 6.1: Parallelization of queries

6.2 T_{query}: Querying Teams

To study the scalability of querying teams, we performed experiments both for homogeneous teams and for heterogeneous teams. In the homogeneuous case, we considered two sets of scenarios where workspaces are 7×3 grid cells, as shown in Table 6.1.The team sizes in these scenarios are kept reasonable (2–9 robots per workspace) considering real manufacturing processes, since every work cell in a real factory typically is of modest size with 3–12 operators. In the first set each team has a different size. Scenarios 1–3 (resp. Scenarios 4 and 5) are built incrementally. For example, in Scenario 2 there are three teams with 1, 2, 4 worker robots, respectively, and the total number of robots in this scenario, including the charger robots, is 11, whereas in Scenario 3 an additional team with 6 worker robots and 3 charger robots is added. In the second set of scenarios, some teams may have the same size. Here also Scenarios 1_b–3_b (resp. Scenarios 4_b –6_b) are built incrementally, but considering teams of same sizes. For instance, in Scenario 1_b there are three teams with 1, 1, 2 worker robots, whereas in Scenario 2_b there are four teams with 1, 1, 2, 2 worker robots, the team numbers are increased by adding new teams of the same size.

We also considered the heterogeneous cases (n = 2) of these scenarios, where the number of a worker robot of one type is computed by dividing the total number of workers by n. For example, the heterogeneous case of Scenario 4 is two teams with 1 wet, 1 dry worker robots and 2 wet, 2 dry worker robots, respectively.

In these experiments, in order to find the optimal global plan length for the teams in each scenario, we considered queries of the form Q1-Q3 as described in Chapter 5. Total number of queries in each scenario corresponds to the total number of queries answered by all the teams until a coordination for an optimal length is found. The computation time T_{query} in each scenario is computed by the time it takes for the slowest team to answer all the queries.

We would like to analyze the results to better understand how the computation time (CPU seconds) for answering queries and the optimal global plan length are affected by a

Table 6.1: Scenarios

Scenario	Teams	Worker robots	Charger robots	Total robots
1	2	1,2	1,1	5
2	3	1,2,4	1,1,2	11
3	4	1,2,4,6	1,1,2,3	20
4	2	2,4	1,2	9
5	3	2,4,6	1,2,3	18
1_b	3	1,1,2	1,1,1	7
2_b	4	1,1,2,2	1,1,1,1	10
3_b	5	1,1,2,2,2	1,1,1,1,1	13
4_b	3	2,2,4	1,1,2	12
5_b	4	2,2,4,4	1,1,2,2	18
6_b	5	2,2,4,4,4	1,1,2,2,2	24

change in the team size, the number of teams, the number of orders, the maximum number of robot exchange between teams and when heterogeneous robots with varying capabilities are considered. Although the experimental evaluation of different sorts of hybrid reasoning is extensively studied in a companion paper [60], we have also conducted some experiments to observe the effect of hybrid reasoning.

6.2.1 Changing the team sizes and the workspace sizes

To analyze the effects of changing the team size (i.e., the total number of robots in each team) and the workspace size (i.e., number of grid cells) on the computational efficiency (T_{query}) and the quality of plans (i.e., makespans), first we considered scenarios with one team of homogeneous robots. Therefore, we generated two sets of instances for Scenario 2 from Table 6.1 to compare teams with different sizes, that vary over two different sizes of workspaces (with 5×3=15 and 7×3=21 grid cells), with the maximum makespan of k = 50.

Table 6.2 shows the results of query answering. We can observe from this table that the total time for a team of 2 robots to answer all 93 queries (essentially planning problems) is 9.17 seconds, under multi-threading. In the same workspace size, it takes 47.43 seconds for a team of 6 robot to answer all the 95 queries. For a team of 2 robots in a workspace of 21 size, it takes 20.56 seconds to answer all 86 queries. So the computation time increases as the number of robots per team and the size of the workspace increase.

Next, we also considered heterogeneous robots and multiple teams. In particular, we experimented over scenarios 1_b, 2_b, 4_b and 5_b of Table 6.1, where the team sizes double. Table 6.3 shows the results of query answering in these scenarios that are comparable with each other in terms of team sizes. We can observe from this table that as the team size increases the optimal plan length decreases, since there are more robots that can be used for the tasks.

Table 6.2: Team size and workspace size vs. computation time and plan quality: for one team of multiple homogeneous robots in Scenario 2, the max number of robot exchange is 2 and the order number is 4

Number of robots	Total grid cells	Optimal plan length	Total queries	Time to answer all queries (secs)
2			93	9.17
3	15	24	94	45.01
6			95	47.43
2			86	20.56
3	21	25	87	105.95
6			85	102.45

Table 6.3: Team size and robot transfers vs. computation time and plan quality: multiple teams, where workspaces are of the same size 7×3, the order number is 4, and the maximum makespan is 50

Scenario	Worker robots	Charger robots	Max. robot exchange	homogeneous		heterogeneous		Optimal plan
				Total queries	T_{query}	Total queries	T_{query}	
	#	#	#	#	sec	#	sec	length
1_b	1,1,2	1,1,1	1	287	249.91	502	139.81	39
			2	353	390.21	710	137.04	35\ 39
4_b	2,2,4	1,1,2	1	139	41.24	245	34.59	25
			2	207	79.06	389	57.77	23
2_b	1,1,2,2	1,1,1,1	2	264	61.37	394	81.09	30
			3	312	252.11	490	85.63	30
5_b	2,2,4,4	1,1,2,2	2	44	4.15	76	5.67	17
			3	58	6.81	92	6.25	17

Consider the scenarios 1_b and 4_b: each one has 3 teams, but scenario 4_b doubles the number of worker robots in the teams. Consider that case where the maximum number of robot transfers is 1. Under these conditions, the plan length decreases from 39 in Scenario 1_b to 25 in Scenario 4_b, since larger teams are able to finish their tasks quickly. This results in less number of queries and the computation time becomes smaller: it takes 249.91 CPU seconds to solve Scenario 1_b, whereas it takes 41.24 CPU seconds to solve Scenario 4_b. The results are also similar for scenarios 2_b and 5_b.

Now consider the case where the maximum number of robots transfers is 2: so the teams can help each other even more. Since the number of queries increase due to the increase of maximum robot transfers, the computation time increases in all of the scenarios. Also the increase in the number of robot transfer leads the teams to help each other more and in scenarios 1_b and 4_b we can see it affecting the plan quality, since the

Table 6.4: Team number vs. computation time and plan quality: multiple teams, where workspaces are of the same size 7×3, the maximum number of robot transfers is 2, and the maximum makespan is 50

Scenario	Teams	Order	homogeneous		heterogeneous		Optimal plan
			Total queries	T_{query}	Total queries	T_{query}	
	#	#	#	sec	#	sec	length
4	3,6	4	22	4.51	38	6.98	17
2	2,3,6	4	258	105.95	484	86.99	25
5	3,6,9	4	49	13.48	89	19.70	17
1_b	2,2,3	3	298	33.40	478	23.81	30\31
2_b	2,2,3,3	3	166	12.80	246	7.95	24
3_b	2,2,3,3,3	3	205	12.95	291	8.09	24
4_b	3,3,6	5	212	245.69	450	156.33	25\ 26
5_b	3,3,6,6	5	88	16.48	154	18.14	20
6_b	3,3,6,6,6	5	110	18.34	193	20.37	20

plan lengths decrease from 39 to 35 in Scenario 1_b, and from 25 to 23 in Scenario 4_b.

6.2.2 Changing the number of teams

To analyze the effects of changing the number of teams on the computational efficiency and the quality of plans, we considered some of the scenarios of Table 6.1 that have different number of teams but with same parameters such as the maximum number of robot exchange or the number of orders, to compare the effect of the change in team number. Table 6.4 shows the results of query answering from examples of two sets of scenarios.

Consider the scenarios 2, 4 and 5: Scenario 4 has 2 teams and Scenario 2 has 3 teams with the additional team having a smaller size than the other teams. Scenario 5 also has 3 teams, but the additional team has a larger size than the other teams. We can observe from Table 6.4 how the computation time and the plan quality are affected when the number of teams is increased by adding a team with a smaller size. The computation time from Scenario 4 to Scenario 2 increases from 4.51 to 105.95, since a longer global plan length is needed to be found for the smaller team to complete its task. We can see that the optimal global plan length of Scenario 2 is found to be 25, whereas for Scenario 4 it is 17. When the number of teams is increased by adding a team with a larger size as in Scenario 5, the computation time increases slightly with respect to Scenario 4, since the large size of the new team takes more computation time.

In the second set of instances, the team numbers are increased by adding teams of

the same size. Consider the scenarios 1_b and 2_b: we can observe a decrease in the plan length from 30 to 24, because the additional team is not of a smaller size and it is used to help the other teams to reach a shorter plan. This results in the decrease of total number of queries and hence the decrease in the computation times: it takes 33.40 CPU seconds to solve Scenario 1_b, whereas it takes 12.80 CPU seconds to solve Scenario 2_b.

When an additional team is added to Scenario 2_b to have Scenario 3_b, we can observe that the plan quality remains the same, which shows that a coordination for a shorter length cannot be found in this setting. Also the computation time stays the same, since the additional team has the same size, so it does not take a longer time for the team to answer all the queries.

We can observe the similar results in scenarios 4_b, 5_b and 6_b. There is a decrease in the computation time from 245.69 CPU seconds of Scenario 4_b to 16.48 CPU seconds of Scenario 5_b. In these instances the team sizes are larger than of scenarios 1_b, 2_b and 3_b, and the additional teams also has a large size. So since the new team added to Scenario 4_b is large, it is able to help the other teams while finishing its task. This results in the decrease of the plan length from 25 to 20, and in the number of total queries. When an additional team is added to Scenario 5_b, this does not affect the plan quality, which shows that this is the optimal length that can be reached in this setting.

Additionally, it can be observed from Table 6.4 that the plan length increases in certain cases when heterogeneity is considered, since the existence of robots with restricted capabilities may cause delays for the accomplishment of some tasks (check for instance Scenario 1_b, when the teams can help each other by borrowing at most 1 robot or at most 2 robots).

6.2.3 Changing the maximum number of robot transfers

For comparing the effect of the maximum number of robot exchanges between teams, we considered the scenarios in which there are slightly larger teams, since we want the possibility of exchanging more robots between teams. Table 6.5 shows the results of query answering. As expected, the results show that as more number of robots are allowed to be exchanged between teams, the teams become more collaborative which results in smaller optimal global plan lengths.

Consider Scenario 1: when the maximum number of robot exchanges between teams are limited to 1, the optimal plan is found to be of length 30 and the computation takes 795.41 CPU seconds, whereas when the maximum number is set to 2, the plan length decreases to 20 and the computation time decreases to 30.60, which is a large difference.

In Scenario 5, the plan length continues to decrease when a maximum of 3 robot transfers are allowed. The computation time does not show a deep decrease as in Scenario

Table 6.5: Robot transfers vs. computation time and plan quality: multiple teams, where workspaces are of the same size 7×3 and the maximum makespan is 50

Scenario	Worker robots	Charger robots	Order	Max. robot exchange	homogeneous		heterogeneous		Optimal plan
					Total queries	T_{query}	Total queries	T_{query}	
	#	#	#	#	#	sec	#	sec	length
3	1,2,4,6	1,1,2,3	5	1	134	795.41	204	245.02	30
				2	80	30.60	137	34.70	20
5	2,4,6	1,2,3	6	1	74	179.40	115	136.37	25
				2	93	639.95	159	263.35	23
				3	94	261.11	161	174.52	21
1_b	1,1,2	1,1,1	4	1	287	249.91	502	139.81	39
				2	353	390.21	710	137.04	35\39
4_b	2,2,4	1,1,2	5	1	139	41.24	245	34.59	25
				2	207	79.06	389	57.77	23

1, because even though the plan length decreases, the computatation time is more affected by the increase in the total number of queries. Also in scenarios 1_b and 4_b, increasing the maximum number of robot transfers shortens the plan lengths, while the computation times increase slightly.

6.2.4 Changing the number of boxes

We considered the scenarios in Table 6.1 to analyze the effects of changing the number of orders/boxes on the computational efficienct and the quality of plans. Since the resuls are similar in two sets of scenarios, Table 6.6 shows the results of query answering for scenarios 1, 2 and 3.

We can observe from Table 6.6 that, in Scenario 1, the total time for two teams with small sizes to answer all the queries increase considerably as the order number increases: the computation takes 8.49 CPU seconds when the order is 3 boxes, whereas it takes 5232.9 CPU seconds when the order is 5 boxes. This is due to the fact that the work that each teams needs to complete increases, and it takes more time to find a plan utilizing small amount of robots. Also it can be observed that the optimal plan length increases in considerable amount, even if only one more box is added to the order of the teams: the optimal plan length is 24 when the order is 3 boxes, whereas the plan length is 30 when the order is 4.

When we consider Scenario 2, the increase in the computation time is not as severe as in Scenario 1: the computation takes 36.37 CPU seconds when the order is 3 boxes, whereas it takes 1891.75 CPU seconds when the order is 5 boxes. As there are more teams

Table 6.6: Order numbers vs. computation time and plan quality: multiple teams, where workspaces are of the same size 7×3 and the maximum makespan is 50

Scenario #	Worker robots #	Charger robots #	Max. robot exchange #	Order #	homogeneous Total queries #	homogeneous T_{query} sec	heterogeneous Total queries #	heterogeneous T_{query} sec	Optimal plan length
1	1,2	1,1	1	3	55	8.49	91	7.57	24
				4	90	166.15	149	91.40	30
				5	114	5232.90	186	1140.90	36
2	1,2,4	1,1,2	2	3	191	36.37	358	40.99	21
				4	258	106.34	484	85.28	25
				5	338	1891.75	633	533.65	30
3	1,2,4,6	1,1,2,3	3	3	50	13.3	90	10.83	15
				4	83	18.96	148	29.54	17
				5	143	61.78	250	84.05	20

that can help the smaller teams in Scenario 2, the optimal plan length is found to be 21, and as the number of boxes increase the optimal plan length increases.

In Scenario 3, there are teams of large sizes and this results in shorter plan lengths than Scenario 1 and 2. The computation times are also smaller, since the total number of queries are low. We can observe that the increase of the plan length in Scenario 3 is more narrow than in Scenario 1. While the plan length of Scenario 1 increases by 8 steps when 2 more boxes are added to the order, in Scenario 3 the plan length increases by 5 steps. This shows that the teams with larger sizes are able to react more efficiently to new orders.

6.2.5 Using Hybrid Reasoning

Table 6.7 shows the results when the teams in the scenarios shown in Table 6.6 utilizes hybrid reasoning while answering the queries.

We can observe from the table that the computation time increases in all cases, since finding a feasible plan with the constraints is time consuming. Also we can observe the increase in the optimal plan length for some scenarios. For example, for Scenario 1 with 4 orders has an optimal plan length of 30 without hybrid reasoning, whereas the plan length becomes 31 when hybrid reasoning is included. This shows that the optimal global plan found without hybrid reasoning was unfeasible, but with the help of hybrid reasoning a feasible optimal global plan is found.

Table 6.7: Order numbers vs. computation time and plan quality: multiple teams (using hybrid reasoning), where workspaces are of the same size 7×3 and the maximum makespan is 50

Scenario	Worker robots	Charger robots	Max. robot exchange	Order	homogeneous		heterogeneous		Optimal plan
					Total queries	T_{query}	Total queries	T_{query}	
	#	#	#	#	#	sec	#	sec	length
1	1,2	1,1	1	3	48	26.56	75	15.33	24
				4	93	990.05	152	312.63	31
2	1,2,4	1,1,2	2	3	219	69.88	415	73.72	22
				4	287	317.51	537	157.81	26
3	1,2,4,6	1,1,2,3	3	3	49	22.38	90	45.48	15
				4	70	39.92	125	107.76	17

6.3 T_{coord}: Coordination of Teams

To study the scalability of our method for finding a coordination of teams, we have generated two sets of instances, one with homogeneous worker robots ($n = 1$) and one with heterogeneous worker robots ($n = 2, 4$), that vary over the number of teams (ranging between 2–16) and the maximum number of robots that can be transferred ($\overline{m} = 2, 4$). Table 6.8 shows the results of coordination; each reported CPU time is the average for at least $\overline{m} \times n$ instances.

We can observe from this figure that the computation time increases slightly as the factory involves more heterogeneous robots and more number of teams. These results can be observed more clearly, as the number of transferred robots also increases. When the factory involves 16 teams with heterogeneous robots, and the number of robots lent/borrowed between any two teams is at most 4, the computation time is still less than 15 seconds. Intuitively, involving more heterogeneous robots (resp. requiring more robot transfers between teams) makes the coordination problem harder since different capabilities of the robots (resp. more combinations of robot transfers) have to be considered while searching for a coordination.

6.4 Overall Scenarios

Table 6.9 shows the results for six scenarios of varying size for teams of robots with only 1 type of workers, averaged over three runs, without hybrid reasoning. For each scenario, we report the total number of questions answered by the teams, the average CPU time to answer a query and to find a coordination of teams, the length of an optimal

Table 6.8: CPU time in seconds for T_{coord}

number of teams	$\bar{m}$ = 2		$\bar{m}$ = 4	
	n = 1	$n > 1$	n = 1	$n > 1$
2	$< 10^{-6}$	$< 10^{-6}$	$< 10^{-6}$	0.005
4	$< 10^{-6}$	0.014	0.023	0.245
8	0.005	0.061	2.92	3.61
16	0.080	0.432	6.63	14.9

Table 6.9: Experimental results for six scenarios

Scenario	Teams	Workspace (grid cells)	Worker Robots	Total Robots	Order (boxes)	Questions (total)	Answering Questions (average time)	Finding Collaboration (average time)	Optimal Global Plan (with collaboration)	Optimal Global Plan (without collaboration)
	#	#	#	#	#	#	sec	sec	length	length
1	2	15	1,2	5	6	212	3.96	< 0.1	30	34
2	3	15	1,2,3	9	9	437	3.92	< 0.1	25	34
3	4	15	1,2,3,4	15	12	525	1.82	< 0.1	21	34
4	2	24	2,4	8	8	127	4.76	< 0.1	20	29
5	3	24	2,4,6	18	12	171	5.37	< 0.1	18	29
6	4	24	2,4,6,8	30	16	293	79.96	< 0.1	18	29

global plan with/without collaborations of teams. For instance, for Scenario 5, a total of 171 queries are answered by the teams; average less than a second. An optimal global plan with such a coordination has 18 steps; whereas an optimal global plan without any collaborations has 29 steps.

We can observe from the table that finding a coordination function by the mediator takes a negligible amount of time. The majority of the total computation time is spent for the teams to answer questions. As the problem size increases, the size of the ASP program gets larger, making it hard for CLASP to find an answer. Since teams' query answering can be parallelized, scalability of the approach to factories with many workspaces seems plausible.

We can also observe the tradeoff between the optimal global plan length and the total computation time, with and without team collaborations. For instance, for Scenario 2, if we allow collaborations of teams, then we can find an optimal global plan of length 25, in about 33 minutes; otherwise, we can find an optimal global plan of length 34 in about 5 minutes. This computational cost is negligible compared to 27% decrease in process length resulting in large cost savings for the manufacturing industry; time gains achieved by such a decrease in process length will help economic sustainability under low quantity orders, and result also in better customer satisfaction.

Table 6.10: Experimental results comparing ASP vs. SAT

Scenario	*Teams*	*Workspace (grid cells)*	*Worker Robots*	*Total Robots*	*Order (boxes)*	*Questions (total)*	*Answering Questions (average time ASP)*	*Answering Questions (average time CCALC)*
	#	#	#	#	#	#	sec	
1	2	15	1,2	5	6	212	3.96	6.67
2	3	15	1,2,3	9	9	437	3.92	6.13
3	4	15	1,2,3,4	15	12	525	1.82	3.36
4	2	24	2,4	8	8	127	4.76	7.91
5	3	24	2,4,6	18	12	171	5.37	13.08
6	4	24	2,4,6,8	30	16	293	79.96	151.33

6.5 Answering Queries: CCALC vs. ASP

We performed experiments to compare the use of CCALC and ASP for answering queries, with the same instances used in our experiments, as explained in Section 6.4. Table 6.10 summarizes the results of these experiments, comparing the computation times using the ASP solver CLASP with the grounder GRINGO, with the computation times using CCALC with the multi-threaded SAT-solver MANYSAT (limited to four threads). The computation times are average CPU times in seconds, obtained over three repeated runs of all scenarios. The time reported for ASP includes the time spent for grounding by GRINGO; the time reported for SAT includes the time spent for obtaining the propositional theory by CCALC. Note that except for the computation times used to answer queries, all other numbers are the same as in Table 6.10. We observe from these results that the ASP solver CLASP performs better than CCALC with the SAT solver MANYSAT in all cases. This is also true for real time (not shown in tables), not only for CPU time (shown in tables).

6.6 Finding Optimal Values: Binary Search vs. Linear Search

To find the optimal value for a global plan length $\bar{l}$, and to find the earliest/latest lend/borrow times l of individual teams, we can use binary search or linear search.

One possibility is to apply binary search between 1 and $\bar{l}$ to find the earliest lend times and the latest borrow times l, and between 1 and k to find the optimal value for the

global plan length $\bar{l}$. With this approach, every team answers $O(n \cdot \overline{m} \cdot log(k)^2)$ queries.

However, the computation time to answer a query drastically increases as the plan length increases (due to inherent hardness of planning [69, 23]). In such cases, as suggested by Trejo et al. [68], it is not a good idea to apply binary search to find the optimal value for a global plan length $\bar{l}$. Meanwhile, given a plan length, queries to find the earliest lend times and the latest borrow times take about the same time; in such cases, as also suggested by [68], it is a good idea to apply binary search to find these optimal values. Therefore, a better approach might be to use linear search to find the optimal value for a global plan length $\bar{l}$, and binary search to find optimal values for lending/borrowing times. We compared these two approaches to compute optimal values for plan length $\bar{l}$ experimentally over the six scenarios used in our experiments, using CCALC with MANYSAT. Table 6.11 shows the results of these experiments. Results are averages over three runs.

We can observe that these experimental results confirm Trejo et al. [68]'s results summarized above. For small scenarios, the overall time to find a solution (not shown in the table) is smaller with two binary searches while for large scenarios using linear search for $\bar{l}$ gives a better overall performance. For example, for Scenario 1, a total of 1012 seconds is required to find the optimal value for $\bar{l}$ with binary search, while using linear search requires 1670 seconds. On the other hand, finding the optimal value for $\bar{l}$ in Scenario 4 requires 36737 seconds using binary search while linear search requires only 2114 seconds. Nevertheless the overall time to find the optimal solution increases in all scenarios either due to an increased effort for answering questions or due to a significantly increased amount of queries.

Table 6.11: Experimental results comparing linear search vs. binary search

Search Method	*Scenario*	*Teams*	*Workspace (grid cells)*	*Worker Robots*	*Total Robots*	*Order (boxes)*	*Questions (total)*	*Answering Questions (average time)*	*Finding Collaboration (average time)*	*Optimal Global Plan (with collaboration)*	*Optimal Global Plan (without collaboration)*
		#	#	#	#	#	#	sec	sec	length	length
$\bar{l}$: linear	1	2	15	1,2	5	6	212	6.67	< 0.1	30	34
	2	3	15	1,2,3	9	9	437	6.13	< 0.1	25	34
	3	4	15	1,2,3,4	15	12	525	3.36	< 0.1	21	34
	4	2	24	2,4	8	8	127	7.91	< 0.1	20	29
	5	3	24	2,4,6	18	12	171	13.08	< 0.1	18	29
	6	4	24	2,4,6,8	30	16	293	151.33	< 0.1	18	29
$\bar{l}$: binary	1	2	15	1,2	5	6	100	8.78	< 0.1	30	34
	2	3	15	1,2,3	9	9	187	9.96	< 0.1	25	34
	3	4	15	1,2,3,4	15	12	310	7.78	< 0.1	21	34
	4	2	24	2,4	8	8	163	215.13	< 0.1	20	29
	5	3	24	2,4,6	18	12	201	224.32	< 0.1	18	29
	6	4	24	2,4,6,8	30	16	351	283.63	< 0.1	18	29

Chapter 7

Execution Monitoring

Once an optimal global plan is computed for multiple teams of heterogeneous robots to reach their manufacturing goals as soon as possible (via Algorithm 5.1 described in Chapter 5), it is executed by the teams. During the execution of this plan, there may be some changes in the environment (such as a change in the number of the obstacles), or changes in the manufacturing orders (such as an increase in the number of orders). In addition to these changes that can be detected by sensory information or simply by direct communication between the teams and the mediator, there may be some changes that are hard to detect (such as a broken part of a robot). We present a general planning and execution monitoring algorithm, that computes an optimal global plan as described in the earlier sections, monitors this plan's execution, detects discrepancies between the observed states and the expected states, and if the discrepancies lead to plan failures then it applies methods to recover from these failures.

7.1 Algorithm for Execution and Monitoring of Plans

Execution of the global plan with optimal value $\bar{l} \leq k$ consists of each team executing its own local plan in its workspace. Therefore our execution monitoring algorithm checks the execution of the local plans separately. The flowchart in Figure 7.1 shows the general flow of monitoring the execution of each local plan. The algorithm (Algorithm 7.1) observes the current state via the sensors at each step of the plan and checks for a discrepancy or changes. If a discrepancy/change is detected it modifies the planning problem and checks whether this discrepancy/changes causes the rest of the plan to fail, by checking whether executing the rest of the plan reached the goals. If there is plan failure, then replanning is done to reach the deadline in at most k steps. If the team cannot find a plan until the deadline on its own, the planning problem is modified to borrow some robots from other teams after they finish executing their own local plans at step $\bar{l}$. If a plan cannot be found even if the team borrows some robots, then a global replanning is needed.

Figure 7.2 shows the detection of a discrepancy/change during the execution of the local plan. In this thesis, we assume that detected discrepancies are due to broken parts of robots in the workspace. If a discrepancy is detected, the cause of the discrepancy is diagnosed and the diagnosis is added to the planning problem as constraints. If a change in the order is detected, the goals of the planning problem is modified with respect to the new order. If a change in the environment is detected, then the information about the obstacles is added to the planning problem as constraints.

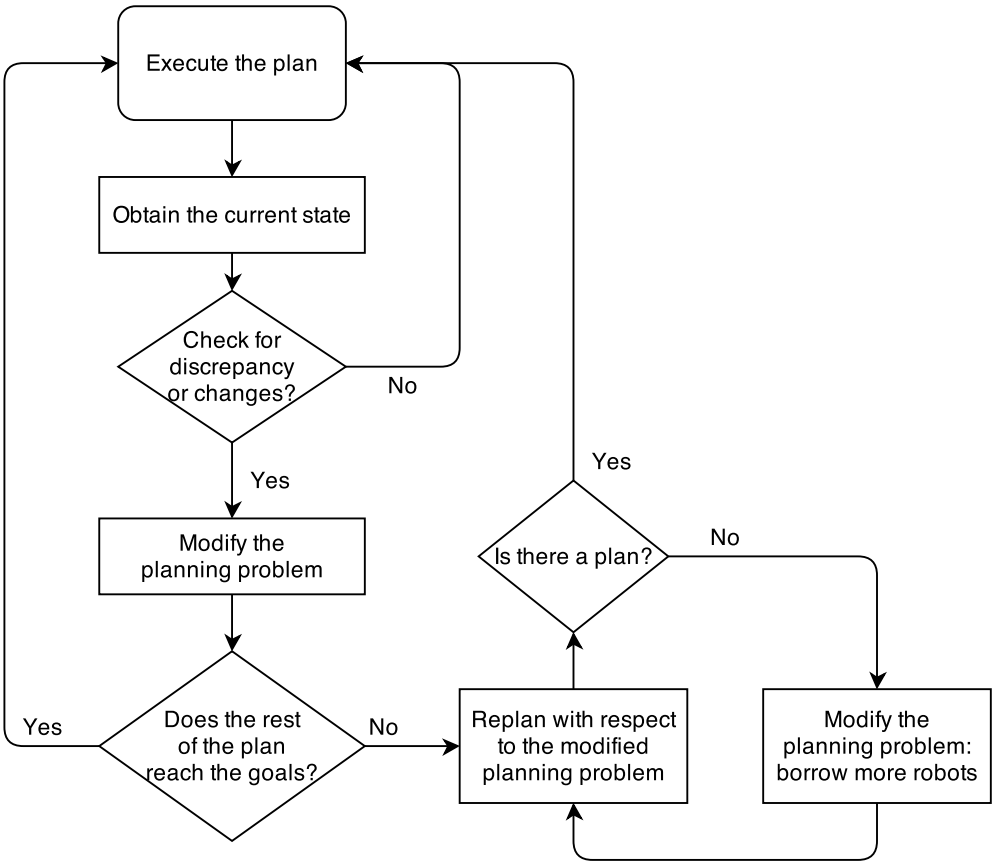


Figure 7.1: Flowchart of a local execution and monitoring algorithm

We now describe the execution and monitoring algorithm in further detail. As the teams execute the computed optimal global plan (i.e., each team executes its own local plan), our execution monitoring algorithm (Algorithm 7.1) checks at each step of the plan

(Line 14) whether there is a discrepancy between the observed state of the world and the expected state (with respect to the monitored fluents),

(Line 20) whether there is some change in the environment such as unknown obstacles in the environment,

(Line 25) whether there is some change in the manufacturing orders.

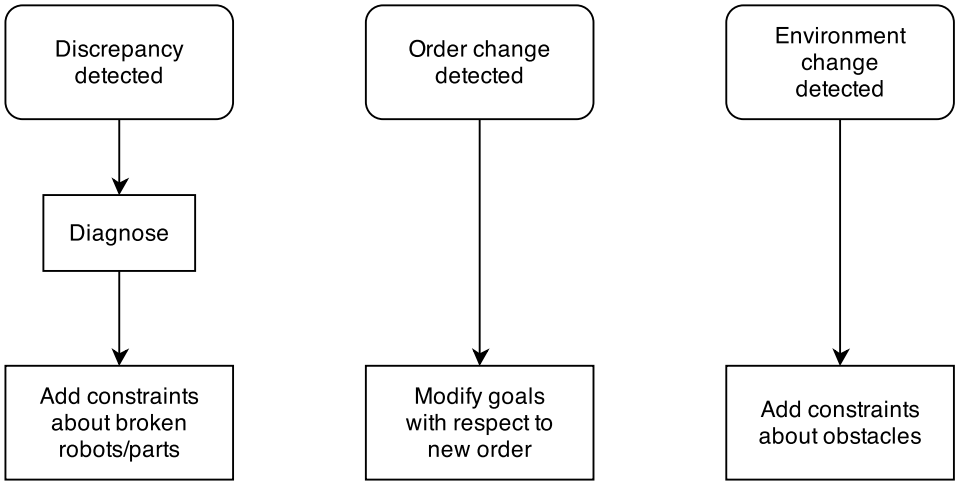


Figure 7.2: Flowchart of detecting a discrepancy/change during the execution

Once a discrepancy is detected (Line 14), our algorithm tries to find a diagnosis of the discrepancy (Line 18) utilizing a diagnosis algorithm (Algorithm 7.3) similar to that of [20]: Algorithm 7.3 tries to find the cause of the failure (i.e., which robots are broken) (Line 3 of Algorithm 7.3), and verifies that the robots are indeed broken (Line 4 of Algorithm 7.3).

If changes are detected in the environment then Algorithm 7.1 obtains full information about the current state (Line 27). If the manufacturing orders change then Algorithm 7.1 updates the goals of the planning problem.

The diagnose algorithm shown in Algorithm 7.3 assumes that there is a discrepancy at the given step l. Until a correct diagnosis is found by MINIMAL_DIAGNOSIS (Line 3) it calls the consulting agent by VERIFYDIAGNOSIS (Line 4) to check whether the diagnosis returned by MINIMAL_DIAGNOSIS is correct or not. If the diagnosis is not correct, it updates the state by expressing that the diagnosis does not hold (Line 5) and the loop again calls for MINIMAL_DIAGNOSIS. Otherwise it exists the loop, and returns the diagnosis. To find a minimal diagnosis, MINIMAL_DIAGNOSIS (Algorithm 7.2) checks for every subset r of i robots, for $i \leq m$, where m denotes the total number of robots, whether or not a plan can be found (Line 5) considering the robots in r as broken. If a plan can be found, then the set of robots are diagnosed to be broken and the algorithm stops the execution.

After a discrepancy/change is detected, Algorithm 7.1 stops the execution of the plan, and checks

(i) whether this discrepancy or change will prevent the execution of the rest of the plan at the current state (Lines 33), and

(ii) if it does not prevent the execution of the rest of the plan at the current state, whether

Algorithm 7.1 PLAN&EXECUTE&MONITOR

Input: An action domain description $\mathcal{D}$, a set **B** of robots/parts that may get broken, positive integers k, n and $m_x, x \leq n$, r planning problems $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_r$ (one for each team) with initial states $s_1, s_2, ..., s_r$ and goal states $g_1, g_2, ..., g_r$, and a transportation delay t_d

Output: Achieve the goals of all teams in minimum time steps

//Let X be a tuple consisting of, for each team i, a plan $P_X[i]$ of length at most $\bar{l} \leq k$, team role $role_X[i]$, lending/borrowing constraints for each team $C[i]$.

1: $step := 0$;
2: $k_X := k$;
3: $X \leftarrow$ FIND_OPTIMAL_GLOBAL_PLAN($\mathcal{D}, k, n, m_1, ..., m_n, \mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_r, t_d$);
4: **while** $k_X > 0$ **do**
5: $globalreplan := false$;
6: **while** $\neg globalreplan \wedge k_X > 0$ **do**
7: $step = step + 1$;
8: $k_X = k_X - 1$;
9: **for all** teams i **do**
10: $\mathcal{D}_i \leftarrow \mathcal{D} \cup \mathcal{D}_{borrow}$ if $role_X[i] = Borrower$;
$\mathcal{D} \cup \mathcal{D}_{lend}$ if $role_X[i] = Lender$;
11: $A_i, c_i, e_i, o_i \leftarrow$ extract from $P_X[i]$ the actions to be executed at $step$, the current state, the expected next state and the observed state after A_i;
12: $localreplan := false$;
13: $discrepancy := false$;
14: **if** $o_i \neq e_i$ **then**
15: $localreplan = true$;
16: $discrepancy = true$;
//Obtain $\mathcal{D}_b$ from $\mathcal{D}$ using **B**.
17: $\mathcal{D}_i \leftarrow \mathcal{D}_b \cup \mathcal{D}_{borrow}$ if $role_X[i] = Borrower$;
$\mathcal{D}_b \cup \mathcal{D}_{lend}$ if $role_X[i] = Lender$;
//obtain the full information of the current state (the observed state with the diagnose)
18: $s_i \leftarrow o_i \cup$ DIAGNOSE($\mathcal{D}_i$, **B**, $P_X[i], s_i, o_i, step$);
19: **end if**
20: **if** new order of boxes **then**
21: $localreplan = true$;
22: $\mathcal{P}_i \leftarrow$ modify the planning problem $\mathcal{P}_i$ with the changed order of boxes;
23: $s_i \leftarrow$ obtain the current state;
24: **end if**
25: **if** the environment is updated **then**
26: $localreplan = true$;
27: $s_i \leftarrow o_i$;
28: **end if**

29: **if** $localreplan$ **then**
30: **if** $discrepancy$ **then**
//Obtain $\mathcal{D}_x$ from $\mathcal{D}$ using **B**.
31: $\mathcal{D}_i \leftarrow \mathcal{D}_x \cup \mathcal{D}_{borrow}$ if $role_X[i] = Borrower$;
$\mathcal{D}_x \cup \mathcal{D}_{lend}$ if $role_X[i] = Lender$;
32: **end if**
// the planning problem of team i, $\mathcal{P}_i$, now has an updated initial state s_i
33: **if** not CHECK_PLAN($\mathcal{D}_i, P_X[i], \mathcal{P}_i, C_i$) **then**
34: **if** $discrepancy$ **then**
35: $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \mathcal{D}_{repair}$;
36: **end if**
37: $P_X[i] \leftarrow$ FIND_LOCAL_PLAN($\mathcal{D}_i, k_X, \mathcal{P}_i, C_i$);
38: **if** $P_X[i] = none$ **then**
39: $\mathcal{P}_i \leftarrow$ modify the planning problem $\mathcal{P}_i$ by allowing borrowing robots after step $\bar{l}$;
40: $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \mathcal{D}_{borrow}$;
41: $P_X[i] \leftarrow$ FIND_LOCAL_PLAN($\mathcal{D}_i, k_X, \mathcal{P}_i, C_i$);
42: **end if**
43: **if** $P_X[i] = none$ **then**
44: $globalreplan = true$;
45: **end if**
46: **end if**
47: **end if**
48: **end for**
49: **end while**
50: **if** globalreplan **then**
51: $X \leftarrow$ FIND_OPTIMAL_GLOBAL_PLAN($\mathcal{D}, k, n, m_1, ..., m_n, \mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_r, t_d$);
52: **end if**
53: **end while**

the rest of the plan will lead to a goal state (Lines 37).

Algorithm 7.1 checks (i) and (ii), i.e., whether the rest of the plan reaches the goals from the current state. If the rest of the plan can still be executed from the current state, CHECK_PLAN returns true, the execution of the plan continues. If the rest of the plan cannot be executed to reach the goals, then a new plan is needed to recover from the possible plan failure. If the reason for a new plan is an encountered discrepancy, the domain description $\mathcal{D}_i$ is updated with $\mathcal{D}_{repair}$ which contains the repair actions to allow repairs of broken robots if needed. Then a new local plan with at most k_X length satisfying the constraints in C_i is asked. If such a local plan cannot be found for Team i then the planning problem P_i is modified to allow borrowing robots after step $\bar{l}$. A new plan is asked with modified planning problem and the updated domain description.

If a local plan to recover from a plan failure at a team cannot be found, then a global replanning needs to be done.

Algorithm 7.2 MINIMALDIAGNOSIS

Input: D: domain description where robots may be broken, **B**, P: executed plan, s: initial state, o: observed state, nonnegative integers m, l

Output: C_B: a set of broken robots/robot parts

1: $i := 1$;
2: **while** $i \leq m$ and $C_B \neq 0$ **do**
3: **for all** set r of i robots **do**
4: $o' \leftarrow$ add to o the robots in r as broken, and the remaining robots as not broken;
5: **if** FIND_PLAN(D, P, s, o', l) **then**
6: $C_B \leftarrow$ update by adding the diagnosed parts in the plan;
7: **break**;
8: **end if**
9: **end for**
10: i++;
11: **end while**

Algorithm 7.3 DIAGNOSE

Input: D: domain description where robots may be broken, **B**, P: executed plan, s: initial state, o: observed state, nonnegative integers m, l

Output: C: the diagnosis

//Assumption: a discrepancy is already detected

1: $correctdiagnosis := false$;
2: **while** $\neg correctdiagnosis$ **do**
3: $C \leftarrow$ MINIMALDIAGNOSIS(D, **B**, P, s, o, m, l);
4: **if** not VERIFYDIAGNOSIS() **then**
5: $o \leftarrow$update o stating that the diagnosis does not hold;
6: **else**
7: $correctdiagnosis = true$;
8: **end if**
9: **end while**

To summarize our approach when there is a change during execution: (i) First the algorithm calls CHECK_PLAN to check whether the current plan can still be executed. (ii) If not, then the algorithm calls FIND_LOCAL_PLAN to find a plan from the current state to the goal state of length at most k. (iii) If a plan cannot be found, then the planning problem is modified and FIND_LOCAL_PLAN is again called to find a plan from the current state to the goal state of length at most k by allowing the team borrowing robots after step $\bar{l}$. (iv) If a plan is found in either of the calls of FIND_LOCAL_PLAN, then the team continues to execute this new local plan. Otherwise a global replanning is needed. A new optimal global plan of length less than or equal to k is tried to be found by calling FIND_OPTIMAL_GLOBAL_PLAN. (v) If a plan of length at most k cannot be found then k is increased one by one and FIND_OPTIMAL_GLOBAL_PLAN is called until a plan is found.

7.1.1 Example

Consider two teams with heterogeneous robots in a cogtive toy factory. Team 1 has one wet robot and one charger, team 2 has two wet robots, two dry robots, and one charger. Both teams have 6 products to work on. A coordination of these teams if found for an optimal value of $\bar{l} = 29$. According to this coordination, team 2 lends two dry robots to team 1 at step 12 and team 1 borrows these robots at step 15. Table 7.1 shows the execution of the global plan calculated according to the coordination and $\bar{l}$.

The teams begin by shifting the boxes on the line and moving the worker robots near the assembly line. At step 3 charger `c1` is expected to do a charging action on worker `1`, but it fails to do so. This failure causes a difference in the observed state and the expected state of step 3. A diagnosis is found for the detected discrepancy and a new local replan is computed. Team 2 begins executing its new plan, whereas team 1 continues executing its old plan. At step 12, team 2 lends the dry robots as expected. Then a new order of one more box arrives to team 1. A new local plan is computed, and team 1 begins executing its new plan. At step 15, team 1 borrows the robots as expected. The teams continue executing their plans.

Table 7.1: Execution of the Plans

(The symbol '-' denotes "doing nothing".)

Time	Team1	Team2
0	`lineShift`	`lineShift move(c1,up) move(c1,right) move(1,right)`
1	`lineShift`	`swapEndEffector(2,2) swapEndEffector(4,3) lineShift move(c1,right) move(1,right)`
2	`llineShift move(1,right)`	`dock(c1,1) workOn(1,1) move(2,right)`
3	`workOn(1,1) move(c1,up) move(c1,right)`	`charge(c1) lineShift move(4,right)`

To be continued on next page

	Continued from previous page	Charging action failed.
	-	Diagnose failure. Replan.
4	move(1,left)	repairInlet(1)
5	swapEndEffector(1,2)	workOn(2,1) workOn(1,2)
6	move(1,right)	charge(c1) ineShift
7	dock(c1,1)	undock(c1) workOn(2,2) workOn(1,3) workOn(4,1)
8	charge(c1)	lineShift move(c1,left)
9	lineShift	dock(c1,2) workOn(1,4) workOn(4,2)
10	undock(c1) workOn(1,1)	charge(c1) move(4,down) move(1,right)
11	move(c1,right) move(1,left)	undock(c1) workOn(2,3) workOn(1,5) move(4,up) move(4,left)
12	swapEndEffector(1,1) New order.	giveRobot(4) giveRobot(3) lineShift move(c1,right)
	Replan.	-
13	move(1,right)	dock(c1,1)
14	takeRobot(we2) takeRobot(we1) dock(c1,1)	charge(c1) workOn(2,4)
15	swapEndEffector(we1,2) charge(c1)	undock(c1) workOn(1,6) move(2,down) move(2,right)
16	undock(c1) workOn(1,2)	move(c1,left) move(1,left)
17	move(c1,right) move(1,right)	dock(c1,1)
18	swapEndEffector(we2,3) dock(c1,1) workOn(1,3) move(we1,right)	charge(c1) move(2,up)
19	charge(c1) workOn(we1,2) move(we2,right)	undock(c1)
20	undock(c1) workOn(we2,1)	move(1,left) move(c1,right)
21	lineShift move(c1,left)	swapEndEffector(1,3) dock(c1,2)
22	dock(c1,we1) workOn(1,4) workOn(we2,2) workOn(we1,3)	charge(c1) move(1,right)
23	charge(c1) lineShift	undock(c1) workOn(2,5) workOn(1,3)
24	undock(c1) workOn(1,5) workOn(we2,3) workOn(we1,4)	lineShift move(c1,left)
25	lineShift move(c1,left)	dock(c1,1) workOn(2,6) workOn(1,4)
26	dock(c1,we2) workOn(1,6)	charge(c1) lineShift
27	charge(c1)	undock(c1) workOn(1,5)
28	workOn(we2,4) workOn(we1,5)	lineShift
29	lineShift	workOn(1,6)
30	workOn(we2,5) workOn(we1,6)	lineShift
31	lineShift	lineShift
32	workOn(we2,6)	-
33	undock(c1) lineShift	-
34	lineShift	-

7.2 Discussion

In our algorithm for execution and monitoring, while searching for a new plan for Team i, we take into account its constraints C_i from the coordination that was found for all teams for length $\bar{l}$. Therefore, if a new plan is found, it still satisfies the desired constraints. Also, we allow the team to borrow robots after $\bar{l}$ and since every other team is suppose

to finish at time step $\bar{l}$, they can easily lend the necessary robots. Last but not least, a local replanning of one team does not affect the other teams' plans, since each team has its own workspace. Therefore, this algorithm creates no conflicts when the overall plan is modified.

7.3 Demonstrations

We have shown the applicability of our approach with a simulation of a cognitive toy factory. The implementation is done in C++ and Python. Plan execution is simulated using OpenRave. A video clip illustraying the simulation of the example above can be found in the following address: `http://youtu.be/CU9gWG-dRSs`

Chapter 8

Related Work

In this thesis, we address a variety of challenges from coordination for decoupled planning to execution monitoring. Hence, we examine the related work in literature under several subtopics.

8.1 Decoupled Planning

There are works on decoupling plans of multiple agents to coordinate their actions [10] in that local plans are computed by agents and then combined in order to compute a global plan. In these related works, there are studies on specifying social laws before local planning to have a conflict-free coordination. These social laws are followed by every agent, and they restrict the agents' behavior. The traffic rule of everyone driving on the right side of the road traffic rules can be given as example to social laws. Shoham and Tennenholtz [63] studies how social laws can be created in the design phase of a multi-agent system. Another pre-planning coordination method is studied by Mors et al. [67] to find a minimal set of additional constraints on subplans to ensure a coordination of the individual plans.

Another approach that is used by the related works is putting restrictions on local plans to be able synchronize them or merge them into a global plan. Georgeff [36] is the first to propose a plan-synchronization method from individual plans. They resolve the interaction between individual actions, then determine the possible ”unsafe” situations, and insert synchronization primitives to the subplans to avoid the unsafe situations. Stuart [65] uses a propositional temporal logic to specify constraints to a theorem prover to generate sequences of communication actions that guarantee that no event will fail. Yang et al. [70] describe a method to merge individually generated plans with a set of limitations on the allowable interactions between goals. Foulser et al. [30] give formulations on plan merging and define the mergeability of operations.

There are also methods that integrate individual planning with coordination of the

plans, by exchanging information between teams about their partial plans or goals. The Partial Global Planning framework [16] and its extension, Generalized PGP [11], for agents to share their plans using a specialized plan representation. Coordination is achieved as follows: if an agent informs a second agent of its own plan, the second agent merges this information into its own partial global plan and then tries to improve the global plan. The improved plan is shown to the other agents who can accept/reject/modify it. Alami et al. [1] present a scheme for multi-agent cooperation where each robot coordinates its own plan with other robots' plans to ensure proper execution.

Our method is different from these works in that no restrictions are put on the order of actions for local planning of each team, and that teams do not exchange information about their plans or goals with each other. Each team communicates with the mediator, by only answering the mediator's yes/no questions; so the teams do not have to share private information with each other. Also, we do not assume that all teams are in the same workspace, or all robots are of the same sort. Moreover, our goal is not to find any coordination of teams that would allow decoupling of their local plans, but to find a coordination of teams for an optimal global plan (with minimum makespan); therefore, we also consider exchange of robots between teams. Furthermore, after the mediator informs the teams about when they are expected to lend/borrow robots, each team computes optimal local plans (with minimum makespans, and possibly with some other optimizations of total action costs).

A neutral coordinator is used by Ehtam et al. [17] to negotiate with the decision makers. Each decision maker solves its own problem, communicates with the mediator about the preferred points. Negotiation continues until a pareto-optimal solution is reached. There are also existing work on task assignment and scheduling. Luo et. al [54] present an auction-based approach where robots bid to get assigned to a task, and reach an almost-optimal solution. Gombolay et. al [39] introduce a multi-agent task sequencer that computes schedules that satisfies temporal deadlines and spatial restrictions and produces near-optimal solutions.

In our method, the mediator does not negotiate with the teams but simply gathers information from them to find an optimal global solution. It does not know anything about the teams's goals, tasks or workspaces, and the teams do not know what the mediator is trying to optimize. It does not try to assign/schedule tasks but only informs the teams about when they are expected to lend/borrow robots.

Robots can be considered as shared resources in our approach since they are exchanged between teams. An approach by Lin [53] considers a domain where the agents negotiate for common resources and informs the central coordination of their next events and proposals of time intervals. The agents schedules their events until the coordinator approves the schedule while conducting the fault detection task. In the work by Sycara et al. [66] robots communicate with each other their demand of shared resources and

schedule their tasks according to resource availability.

Our work is different from the existing approaches on resource allocation in a multi-agent time-constrained domain [53, 66, 8], because the mediator does not require any information about local plans, ordering constraints on actions, or causal links, to decide for resource allocation.

There are studies also on team formation. For example, Gaston and desJardins [32] consider the existence of a centralized task knowledge that provides information on which skills have been filled, and the agents then attempt to form teams to accomplish tasks. Local information is available to the individual agents for deciding which teams to initiate and which teams to join. Although robot exchanges modifies the teams, our work is different from the works on team formation because our method does not aim for deciding how or when to join teams.

8.2 Hybrid Reasoning

Task planners are very efficient and are able to solve complex problems. However integrating the geometric information of the environment or robots in these planners is a challange. Therefore the work of combining task planning and motion planning in efficient ways plays an important role in robotics. The related work on hybrid reasoning can be separated into two groups: integration at search level and integration at representation level.

Some works that integrate geometric reasoning and task planning while searching for a plan are summarized. In the work by Alili et. al. [2], the overall planning system starts from a goal, and builds a plan which is based on planned actions for the robot and estimation of feasibility of actions for the human. While planning, the symbolic planner queries to the geometric planner about the feasibility of each action. Kaelbling and Lozano-Pérez [43] construct a plan at an abstract level, by constraining the plan steps so that they are serializable and then recursively plan and execute actions to achieve the first step in the abstract plan without constructing the rest of the plan in detail. They handle the integration of continuous geometric planning with task planning by using approximate geometric computations that construct appropriate choices for the parameters of an operator. Cambon et. al. [6], introduce a task planner that is based on a hybrid planning process in which a fast heuristic forward planner for "high-level" plans is used in close interaction with a manipulation planning subsystem for instantiating and validating actions in the three-dimensional environment. Plaku and Hager [58], use symbolic action planning to guide the tree-based exploration by identifying and selecting discrete actions and regions of the continuous space that sampling-based motion planning can further explore to significantly advance the search for a solution trajectory.

The second group of works integrate geometric reasoning and task planning at the

representation level. Guitton and Farges [40] propose a method allowing to take into account geometric constraints at the symbolic description of an action and to satisfy them. These constraints are formulated through the addition of a new set of preconditions, the geometric preconditions, which are sent to the motion planner through planning requests. The approach by Gashler et. al. [31] uses 3D geometric volumes as the underlying representation for symbolic planning and motion planning, and combines this idea with a general-purpose AI planner supporting deterministic planning with incomplete information and sensing. techniques. Kresse and Beetz [45] specify motions using constraints, which tightly links symbolic reasoning to control theoretic execution. They combine symbolic and geometric reasoning by using plan-based action control to implement sophisticated symbolic actions. Eyerich et. al. [24, 14] consider semantic attachments whose values are determined by external mechanisms. The subproblems like path planning are dealt with in the semantic attachments and the information actually needed to solve the problem is generated at the time the semantic attachment is invoked by the high-level planner. They implement their approach on an autonomous robot capable of performing service tasks in a typical kitchen environment.

Our approach is in the second group of work. It is similar with the methods in [19, 42], however the implementation is on a new domain.

8.3 Execution Monitoring

For execution monitoring, there are works that consider replanning when a discrepancy is detected during the execution. De Giacomo et al. [37] provide a situation calculus-based approach in which after executing each action the robot compares its mental world model with reality, if a discrepancy is detected it recovers from this unanticipated event. Parsons et al. [57] integrate the execution of low-level navigation primitives to high-level reasoning processes. Depending on the intentions of the robot a plan is computed, and if there is inadequacy during the execution of the plan, it switches to a new plan considering current degrees of satisfaction and of adequacy.

Doherty et al. [13] present a temporal-logic based task planning end execution monitoring framework. The plan is monitored by a knowledge processing middleware framework which informs the command executor if a violation is detected, and then a recovery procedure is handled. Lemai and Ingrand [46] integrate deliberative planning, plan repair and execution control in a dynamic environment with tight temporal constraints.

In the work by Fichtner et al. [28] discrepancies are detected during the execution of the plan, explanations for action failures are provided in terms of which action has failed, then a recovery plan is computed.

In our method, if a discrepancy is detected during the execution of the plan, we diagnose the failure in terms of broken robots/parts that caused the plans not to be executed as

expected. We also consider other types of changes in the cognitive factory workspace such as new manufacturing orders or new obstacles. After a discrepancy/change is detected, we first check whether this discrepancy/change causes plan failure. Then we consider replanning to recover from the current state to reach the goal. We also check whether a plan can be found if the team gets help from the other teams, which prevents a global replanning. If a recovering plan cannot be found even if the team gets help from other teams, then we consider global replanning.

8.4 Cognitive Factories

There are recent works relevant to cognitive factories [61, 3] that complements our framework. Shea et. al. [62] use generative CNC machining planning with shape grammars and automated fixture design for automated fabrication of customized part geometries. Maier et. al. [55] introduce a model-based approach that computes success probabilities of plans utilizing online observations for product manufacturing. Lenz et al. [47] present a system that anticipates human behavior, based on knowledge databases and decision processes, ensuring an effective collaboration between the human and robot.

In our method, we use knowledge representation and automated reasoning formalisms and tools to endow each multiple teams of robots with high-level reasoning capabilities, similar with [20, 21]. However, our method utilizes a wider variety of automated reasoners in a more general setting of cognitive factories, not constrained with tight restrictions on robot types or number of robot transfers.

Chapter 9

Conclusion

We have introduced a novel semi-distributed method to find an optimal global plan (with minimum makespan) for multiple teams of heterogeneous robots, by means of determining a coordination of teams based on their answers to yes/no questions that do not convey private information about their workspace, tasks, robots, plans, actions, goals, etc. in cognitive factories. According to this method, teams can help each other by lending/borrowing robots (which motivates the use of a central mediator for coordination) but they are not allowed to know about each other's workspaces or tasks (which motivates distributed computation of a global plan). This planning method consists of two core stages: finding a coordination of teams in such a way to guarantee optimality of a global plan, and decoupling hybrid local plans into a feasible global plan. The correctness of this method is proven in this thesis. Both stages involve NP-hard problems; the intractability of the coordination problem is proven in this thesis. Fortunately, the design of the optimal global planning method allows for extensive use of parallelization, and thus improves scalability.

We have also introduced an execution monitoring framework that checks for various sorts of discrepancies and provides methods for plan failure recoveries if needed. This framework considers execution failures due to (i) a discrepancy during the execution of the plan, (ii) changes in the manufacturing orders of the teams or in the environment like unknown obstacles in workspaces of the teams. To recover from failures due to discrepancies, we have introduced an algorithm that uses high-level diagnostic reasoning methods. To recover from failures due to changes in the manufacturing orders or environment, we have introduced methods for replanning of local/global plans taking into account complex goals and constraint.

We have introduced cognitive toy factory scenarios and represented the cognitive toy factory domain in the formalisms of $\mathcal{C}+$ and ASP. We have performed experiments to analyze the scalability of local hybrid planning, finding a coordination and global optimal planning. When more heterogeneous robots were involved in the collaboration problem,

the computation time for finding a coordination slightly increased, but still remained reasonable. For optimal global planning, the benefits of extensive use of parallelization were observed in the experiments, and gave promising results.

Bibliography

[1] Rachid Alami, Felix Ingrand, and Samer Qutub. A scheme for coordinating multi-robots planning activities and plans execution. In *Proc. of ECAI*, 1998.

[2] Samir Alili, Amit Kumar Pandey, Emrah Akin Sisbot, and Rachid Alami. Interleaving symbolic and geometric reasoning for a robotic assistant. In *ICAPS Workshop on Combining Action and Motion Planning (CAMP)*, 2010.

[3] Alexander Bannat, Thibault Bautze, Michael Beetz, Juergen Blume, Klaus Diepold, Christoph Ertelt, and Florian Geiger et al. Artificial cognition in production systems. *IEEE Transactions*, 8(1):148–174, 2011.

[4] Michael Beetz, Martin Buss, and Dirk Wollherr. CTS - What is the role of artificial intelligence? In *Proc. of KI*, pages 19–42, 2007.

[5] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

[6] Stephane Cambon, Rachid Alami, and Fabien Gravot. A hybrid approach to intricate motion, manipulation and task planning. *The International Journal of Robotics Research*, 28(1):104–126, 2009.

[7] Michael Casolary and Joohyung Lee. Representing the language of the causal calculator in answer set programming. In *Proc. of ICLP (Technical Communications)*, pages 51–61, 2011.

[8] Yann Chevaleyre, Paul E. Dunne, Ulle Endriss, Jérôme Lang, Michel Lemaître, Nicolas Maudet, Julian A. Padget, Steve Phelps, Juan A. Rodríguez-Aguilar, and Paulo Sousa. Issues in multiagent resource allocation. *Informatica*, 30(1):3–31, 2006.

[9] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

[10] Mathijs de Weerdt and Brad Clement. Introduction to planning in multiagent systems. *Multiagent and Grid Systems*, 5:345–355, 2009.

[11] Keith S. Decker and Victor R. Lesser. Designing a family of coordination algorithms. In *Proc. of DAI*, pages 65–84, 1994.

[12] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.

[13] Patrick Doherty, Jonas Kvarnström, and Fredrik Heintz. A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. *Autonomous Agents and Multi-Agent Systems*, 19(3):332–377, 2009.

[14] Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trùg, Michael Brenner, and Bernhard Nebel. Semantic attachments for domain-independent planning systems. In *Proc. of ICAPS*, pages 114–121, 2009.

[15] Damien J. Duff, Esra Erdem, and Volkan Patoglu. Integration of 3D object recognition and planning for robotic manipulation: A preliminary report. In *Proc. ICLP 2013 Workshop on Knowledge Representation and Reasoning in Robotics*, 2013.

[16] Edmund H. Durfee and Victor R. Lesser. Planning coordinated actions in dynamic domains. In *Proc. of the DARPA Knowledge-Based Planning Workshop*, pages 18.1–18.10, 1987.

[17] Harri Ehtamo, Raimo P. Hamalainen, Pirja Heiskanen, Jeffrey Teich, Markku Verkama, and Stanley Zionts. Generating pareto solutions in a two-party setting: Constraint proposal methods. *Management Science*, 45(12):1697–1709, 1999.

[18] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer-Set Programming. In *Proc. of IJCAI*, pages 90–96, 2005.

[19] Esra Erdem, Kadir Haspalamutgil, Can Palaz, Volkan Patoglu, and Tansel Uras. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *Proc. of ICRA*, 2011.

[20] Esra Erdem, Kadir Haspalamutgil, Volkan Patoglu, and Tansel Uras. Causality-based planning and diagnostic reasoning for cognitive factories. In *Proc. of IEEE Int. Conf. Emerging Technologies and Factory Automation (ETFA)*, 2012.

[21] Esra Erdem, Volkan Patoglu, Zeynep G. Saribatur, Peter Schuller, and Tansel Uras. Finding optimal plans for multiple teams of robots through a mediator: A logic-based approach. *Theory and Practice of Logic Programming*, 13(4–5):831–846, 2013.

[22] Esra Erdem, Volkan Patoglu, and Peter Schüller. A systematic analysis of levels of integration between high-level task planning and low-level feasibility checks. In *Proc. of RCRA*, 2014.

[23] Kutluhan Erol, Dana S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artif. Intell.*, 76(1–2):75–88, 1995.

[24] Patrick Eyerich, Thomas Keller, and Bernhard Nebel. Combining action and motion planning via semantic attachments. In *Workshop on Combining Action and Motion Planning at ICAPS*, 2010.

[25] Paolo Ferraris. A logic program characterization of causal theories. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 366–371, 2007.

[26] Paolo Ferraris, Joohyung Lee, Yuliya Lierler, Vladimir Lifschitz, and Fangkai Yang. Representing first-order causal theories by logic programs. *Theory and Practice of Logic Programming*, 12(3):383–412, 2012.

[27] Paolo Ferraris and Vladimir Lifschitz. Weight constraints as nested expressions. *Theory and Practice of Logic Programming*, 5(1–2):45–74, 2005.

[28] Matthias Fichtner, Axel Großmann, and Michael Thielscher. Intelligent execution monitoring in dynamic environments. *Fundamenta Informaticae*, 57(2–4):371–392, 2003.

[29] Jeffrey Finger. *Exploiting Constraints in Design Synthesis*. PhD thesis, Stanford University, 1986.

[30] David E. Foulser, Ming Li, and Qiang Yang. Theory and algorithms for plan merging. *Artif. Intell.*, 57:143–182, 1992.

[31] Andre Gaschler, Ronald P.A. Petrick, Manuel Giuliani, Markus Rickert, and Alois Knoll. Kvp: A knowledge of volumes approach to robot task planning. In *Proc. of IROS*, 2013.

[32] Matthew E. Gaston and Marie desJardins. The effect of network structure on dynamic team formation in multi-agent systems. *Computational Intelligence*, 24(2):122–157, 2008.

[33] Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. clasp: A conflict-driven answer set solver. In *Proc. of LPNMR*, 2007.

[34] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[35] Michael Gelfond and Vladimir Lifschitz. Action languages. *Electronic Transactions on Artificial Intelligence*, 2:193–210, 1998.

[36] M. P. Georgeff. Communication and interaction in multi-agent planning. In *Proc. of DAI*, pages 200–204. 1988.

[37] Giuseppe De Giacomo, Raymond Reiter, and Mikhail Soutchanski. Execution monitoring of high-level robot programs. In *Proc. of KR*, pages 453–465, 1998.

[38] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic causal theories. *Artif. Intell.*, 153:49–104, 2004.

[39] Matthew C. Gombolay, Ronald J. Wilcox, and Julie A. Shah. Fast scheduling of multi-robot teams with temporospatial constraints. In *Proc. of RSS*, 2013.

[40] Julien Guitton and Jean-Loup Farges. Taking into account geometric constraints for task-oriented motion planning. In *ICAPS Workshop on Bridging the Gap Between Task and Motion Planning*, pages 26–33, 2009.

[41] Youssef Hamadi, Saïd Jabbour, and Lakhdar Sais. Control-based clause sharing in parallel sat solving. In *Proc. of IJCAI*, pages 499–504, 2009.

[42] Giray Havur, Guchan Ozbilgin, Esra Erdem, and Volkan Patoglu. Geometric rearrangement of multiple movable objects on cluttered surfaces: A hybrid reasoning approach. In *Proc. of ICRA*, 2014.

[43] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *Proc. of ICRA*, pages 1470–1477, 2011.

[44] Henry Kautz and Bart Selman. Planning as satisfiability. In *Proc. of ECAI*, pages 359–363, 1992.

[45] Ingo Kresse and Michael Beetz. Movement-aware action control – integrating symbolic and control-theoretic action execution. In *Proc. of ICRA*, 2012.

[46] Solange Lemai and Félix Ingrand. Interleaving temporal planning and execution in robotics domains. In *Proc. of AAAI*, pages 617–622, 2004.

[47] Claus Lenz, Suraj Nair, Markus Rickert, Alois Knoll, W. Rosel, Jürgen Gast, Alexander Bannat, and Frank Wallhoff. Joint-action for humans and industrial robots for assembly tasks. In *Proc. of ROMAN*, pages 130–135, 2008.

[48] Hector Levesque and Gerhard Lakemeyer. Cognitive robotics. In *Handbook of Knowledge Representation*. Elsevier, 2007.

[49] Vladimir Lifschitz. Answer set programming and plan generation. *Artif. Intell.*, 138:39–54, 2002.

[50] Vladimir Lifschitz. What is answer set programming? In *Proc. of AAAI*, pages 1594–1597. MIT Press, 2008.

[51] Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25:369–389, 1999.

[52] Vladimir Lifschitz and Hudson Turner. Representing transition systems by logic programs. In *Proc. of LPNMR*, pages 92–106, 1999.

[53] Shieu-Hong Lin. Coordinating time-constrained multi-agent resource sharing with fault detection. In *Proc. of IEEM*, pages 1000–1004, 2011.

[54] Lingzhi Luo, Nilanjan Chakraborty, and Katia Sycara. Distributed algorithm design for multi-robot task assignment with deadlines for tasks. In *Proc. of ICRA*, 2013.

[55] Paul Maier, Martin Sachenbacher, Thomas Rühr, and Lukas Kuhn. Automated plan assessment in cognitive manufacturing. *Advanced Engineering Informatics*, 24(3):308 – 319, 2010.

[56] Norman Clayton Mccain. *Causality in Commonsense Reasoning about Actions*. PhD thesis, 1997.

[57] Simon Parsons, Ola Pettersson, Alessandro Saffiotti, and Michael Wooldridge. Robots with the best of intentions. In *Artificial Intelligence Today*, pages 329–338. Springer Berlin Heidelberg, 1999.

[58] Erion Plaku and Gregory D. Hager. Sampling-based motion and symbolic action planning with geometric and differential constraints. In *Proc. of ICRA*, pages 5002–5008, 2010.

[59] Zeynep G. Saribatur, Esra Erdem, and Volkan Patoglu. Cognitive factories with multiple teams of heterogeneous robots: Hybrid reasoning for optimal feasible global plans. In *Proc. of IROS*, 2014.

[60] Peter Schüller, Volkan Patoglu, and Esra Erdem. A systematic analysis of levels of integration between low-level reasoning and task planning. In *Proc. ICRA 2013 Workshop on Combining Task and Motion Planning*, 2013.

[61] Kristina Shea. Special issue in cognitive robotics. *Advanced Engineering Informatics*, 24, 2010.

[62] Kristina Shea, Christoph Ertelt, Thomas Gmeiner, and Farhad Ameri. Design-to-fabrication automation for the cognitive machine shop. *Advanced Engineering Informatics*, 24(3):251–268, 2010.

[63] Yoav Shoham and Moshe Tennenholtz. On social laws for artificial agent societies:off-line design. *Artif. Intell.*, 73:231–252, 1995.

[64] Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *Artif. Intell.*, 138(1):181–234, 2002.

[65] Christopher J. Stuart. An implementation of a multi-agent plan synchronizer. In *Proc. of IJCAI*, pages 1031–1033, 1985.

[66] Katia P. Sycara, Steven P. Roth, Norman M. Sadeh, and Mark S. Fox. Resource allocation in distributed factory scheduling. *IEEE Expert*, 6(1):29–40, 1991.

[67] Adriaan ter Mors, Jeroen Valk, and Cees Witteveen. Coordinating autonomous planners. In *Proc. of IC-AI*, pages 795–801, 2004.

[68] Raul Trejo, Joel Galloway, Charanjiv Sachar, Vladik Kreinovich, Chitta Baral, and Le-Chi Tuan. From planning to searching for the shortest plan: An optimal transition. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 9(6):827–837, 2001.

[69] Hudson Turner. Polynomial-length planning spans the polynomial hierarchy. In *Proc. of JELIA*, pages 111–124, 2002.

[70] Qiang Yang, Dana S. Nau, and James Hendler. Merging separately generated plans with restricted interactions. *Computational Intelligence*, 8:648–676, 1992.

[71] M.F. Zaeh, M. Beetz, K. Shea, G. Reinhart, K. Bender, C. Lau, M. Ostgathe, W. Vogl, M. Wiesbeck, M. Engelhard, C. Ertelt, T. Rhr, M. Friedrich, and S. Herle. The cognitive factory. In *Changeable and Reconf. Manufacturing Systems*, pages 355–371. 2009.

[72] Michael F. Zaeh, M. Ostgathe, Florian Geiger, and Gunther Reinhart. Adaptive job control in the cognitive factory. In Hoda A. ElMaraghy, editor, *Enabling Manufacturing Competitiveness and Economic Sustainability*, pages 10–17. Springer Berlin Heidelberg, 2012.