Finding Optimal Plans for Multiple Teams of Robots through a Mediator: A Logic-Based Approach∗

Esra Erdem[1], Volkan Patoglu[1], Zeynep G. Saribatur[1], Peter Schüller[1] and Tansel Uras[2]
[1]*Faculty of Engineering and Natural Sciences, Sabancı University, İstanbul, Turkey*
[2]*Department of Computer Science, University of Southern California, Los Angeles, USA*

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

We study the problem of finding optimal plans for multiple teams of robots through a mediator, where each team is given a task to complete in its workspace on its own and where teams are allowed to transfer robots between each other, subject to the following constraints: 1) teams (and the mediator) do not know about each other's workspace or tasks (e.g., for privacy purposes); 2) every team can lend or borrow robots, but not both (e.g., transportation/calibration of robots between/for different workspaces is usually costly). We present a mathematical definition of this problem and analyze its computational complexity. We introduce a novel, logic-based method to solve this problem, utilizing action languages and answer set programming for representation, and the state-of-the-art ASP solvers for reasoning. We show the applicability and usefulness of our approach by experiments on various scenarios of responsive and energy-efficient cognitive factories.

KEYWORDS: answer set programming, decoupled planning, cognitive robotics

1 Introduction

As conventional manufacturing and assembly systems fall short of responding to constantly rising market demands for customized and variant-rich products in a cost effective manner within short delivery times, new approaches for automated fabrication of customized products become crucial for enhancing productivity, ensuring competitiveness and *economic sustainability* in the manufacturing sector. Along these lines, reconfigurable and flexible manufacturing systems have been deployed over the last decade. Cognitive factories (Beetz et al. 2007; Zaeh et al. 2009; Zaeh et al. 2012) are a further step in this direction aimed towards highly flexible and typically small to medium size manufacturing plants that can produce a very large variety of customized products even in low quantities. Rapidly responding to changing customer needs and customization requests, cognitive factories can demonstrate the flexibility of human workshops, while maintaining cost-effectiveness of mass production systems.

Cognitive factories endow manufacturing system with high-level reasoning capabilities in the style of cognitive robotics, such that these systems become capable of planning their

∗ This work is partially supported by TUBITAK Grant 111E116. Peter Schueller is supported by TUBITAK 2216 Fellowship. Tansel Uras' work is carried out while studying at Sabancı University.

own actions. 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.

One of the key challenges in cognitive factories is the coordination between multiple teams of robots to achieve overall shortest delivery time for a given manufacturing order. Minimizing the delivery lead time for a customized order not only leads to a more cost-effective process by reducing contribution of factory overhead per order, but also preserves energy resources and decreases negative environmental impacts by efficient use of facility infrastructure, such as HVAC (heating, ventilation, and air conditioning) and lighting.

With this motivation, we consider multiple teams of robots, where 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, 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); for that purpose, a team can either lend or borrow robots.

Note that a lender can lend robots to more than one team; and a borrower can borrow robots from more than one team. Note also that each team may have robots that cannot be transferred; the above problem considers transferrable robots only and assumes that capabilities of transferrable robots are indifferent.

We introduce a novel 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 other teams and when. 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 to which other team, and when), subject to the constraints C1 and C2 above. The optimal value for $\bar{l}$ can be found by a linear search between 1 and k.

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 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 (Turner 2002), and answering each query in the first phase is a planning problem with temporal constraints, and thus NP-complete (Turner 2002). 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 1 in Section 3). 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.

We propose to solve the planning problems with temporal constraints (and thus answering queries posed by the mediator), and the coordination problem using answer set programming (ASP) (Marek and Truszczyński 1999; Niemelä 1999; Lifschitz 2002; Lifschitz 2008; Brewka et al. 2011). To solve planning problems, first we represent action domains (i.e., workspaces) and planning problems (i.e., queries) in the input language of CCalc (McCain and Turner 1997), which allows representation of dynamic domains in a subset of the expressive action description language $\mathcal{C}+$ (Giunchiglia et al. 2004), and allows teams to solve planning problems with temporal constraints in a variation of the action query language $\mathcal{Q}$ (Gelfond and Lifschitz 1998). Then CCalc's input is transformed into ASP using the tool cplus2asp (Casolary and Lee 2011). After that, we can use state-of-the-art ASP solvers, like Clasp (Gebser et al. 2007), to compute plans. To solve the coordination problem using ASP solvers, we formulate the coordination problem in the representation language of ASP.

We show the usefulness and applicability of our approach by experiments on various scenarios of responsive energy-efficient cognitive factories.

2 Automating Reasoning for a Team of Robots

Our goal is to find an optimal global plan with at most k steps, where at most $\overline{m}$ robots can be transferred between teams. To find a coordination of teams for an optimal global plan, 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 $m \leq \overline{m}$:

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

Once such a coordination is found from the answers of these queries, the mediator informs each team how many robots it is expected to lend to (or borrow from) which other team and when. Then each team computes an optimal local plan to complete its own task, taking into account the relevant information of transfer of robots from/to it. Therefore, each team of robots performs two kinds of reasoning tasks: answering queries of form Q1, Q2 and Q3, and finding optimal plans with complex goals to complete its tasks. Although optimal plans can be found by some existing classical planners, queries

mentioned above cannot be answered by them directly. Also in many application scenarios for cognitive factories, actions of robots are concurrent (e.g., several robots working on different parts of an order at the same time) and there are ramifications and/or delayed effects of actions (e.g., after painting, a box gets dried after a while).

Due to the representation challenges mentioned above (about concurrent actions, ramifications, etc.), we represent workspaces (as action domains) and queries (as planning problems) in the input language of CCALC (McCain and Turner 1997), which allows representation of dynamic domains in a subset of the expressive action description language $\mathcal{C}+$ (Giunchiglia et al. 2004), and allows teams to answer various sorts of queries in an elaboration tolerant way (without having to modify the domain description) in a variation of the action query language $\mathcal{Q}$ (Gelfond and Lifschitz 1998). Also that the language $\mathcal{C}+$ has been used in various sophisticated real-world robotic applications (Erdem and Patoglu 2012), where discrete high-level reasoning is integrated tightly with continuous geometric reasoning and low-level controls (Erdem et al. 2011; Aker et al. 2012; Havur et al. 2013), is advantageous for multi-robot cognitive factory applications that have motivated our studies in this paper. We refer the reader to (Giunchiglia et al. 2004) for the syntax and semantics of $\mathcal{C}+$, and some examples.

In the action query language of CCALC, 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.

Suppose that F and G are fluent formulas denoting an initial state and goal conditions respectively. We can express the question "can you complete the task specified by the initial state F and the goal conditions G in k steps?", with a query of the form

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

Note that this query describes the problem of finding a plan of length k.

Suppose that the action formula $giveRobot(w)$ describes that the team lends the robot w. We can express the question "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 k'?", with a query of the form

$$\begin{aligned} &F \textbf{ holds at } 0 \ \wedge G \textbf{ holds at } k \ \wedge \exists T, W_1, \ldots, W_m : \\ &\quad T < k' \wedge W_1 < W_2 < \ldots < W_m \wedge \textstyle\bigwedge_{i=1}^{m} giveRobot(W_i) \textbf{ occurs at } T. \end{aligned}$$

Given an action description and a query in the input language of CCALC, we can use either state-of-the-art parallel SAT solvers (like MANYSAT (Hamadi et al. 2009)) or state-of-the-art ASP solvers (like CLASP (Gebser et al. 2007)) to find an answer the query. After some experimental evaluations comparing these two approaches (summarized in Appendix A), we have decided to use the ASP solver CLASP (with the grounder GRINGO (Gebser et al. 2011)) to find answers to queries and planning problems. CCALC's input can be transformed into an ASP program in the input language of CLASP, using the tool CPLUS2ASP (Casolary and Lee 2011). If CLASP finds an answer set (Gelfond and Lifschitz 1998) for the ASP program then the query is answered affirmatively; otherwise, the query is answered negatively.

Note that since each team answers queries that are relevant to its workspace, task, goals only, and independently of other teams, queries can be answered by the teams in parallel.

3 Coordination of Teams

From teams' answers to yes/no questions (of the forms Q1–Q3) 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 a set *Lenders* of lender teams and a set *Borrowers* of borrower teams (*Lenders*, *Borrowers* $\subset$ *Teams*): If a team answers "no" to question Q1, and "yes" to question Q3 for some l and m, then it is a borrower. If a team answers "yes" to question Q1 and answers "yes" to question Q2 for some l and m, then it is a lender. Otherwise, it is neither a lender nor a borrower.
- 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) robots, 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.

To precisely define this problem, let us introduce some notation. For every lender team $i \in$ *Lenders*, positive integer $m \leq \overline{m}$ and nonnegative integer $l \leq \bar{l}$, we denote by atoms of the form $lend(i, m, l)$ that the lender team i can lend m robots before time step l. Similarly, for every borrower team $i \in$ *Borrowers*, we denote by atoms of the form $borrow(i, m, l)$ that the borrower team i needs m robots before time step l.

To identify the earliest lend times and latest borrow times, we introduce a collection of partial functions for lenders and borrowers:

$$Lend\_earliest_m : Lenders \mapsto \{0, \ldots, \bar{l}\}$$
$$Borrow\_latest_m : Borrowers \mapsto \{0, \ldots, \bar{l}\}$$

where $Lend\_earliest_m$ returns the earliest step that a lender can lend m robots and $Borrow\_latest_m$ returns the latest step that a borrower needs to borrow m robots:

$$Lend\_earliest_m(i) = \arg\min_l \{lend(i, m, l) = 1\}$$
$$Borrow\_latest_m(j) = \arg\max_l \{borrow(j, m, l) = 1\}.$$

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. Such a delay time is defined by a function:

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

Now we can define when a set of lender teams can collaborate with a set of borrower teams:

Definition 1 *A* $\overline{ml}$-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*

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

(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:*

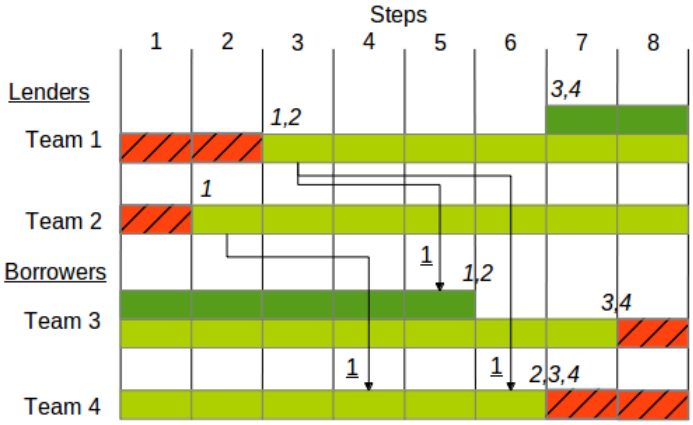

Fig. 1. A summary of teams' answers to queries.

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

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

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

(b) For every lender team $i \in Lenders$, there are some 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 $l_1, \ldots, l_s \leq \overline{l}$ and some positive integers $u_1, \ldots, u_s \leq \overline{m}$, and there is a positive integer $m \leq \overline{m}$ such that

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

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

Example 1 *Consider four teams of robots, where Teams 1 and 2 are lenders and Teams 3 and 4 are borrowers. Take $\overline{l} = 8$ and $\overline{m} = 4$. The lenders' answers to questions of the form Q2 ("Can you complete your task in $\overline{l}$ steps, if you lend m robots before step l?") and the borrowers' answers to questions of the form Q3 ("Can you complete your task in $\overline{l}$ steps, if you borrow m robots after step l?") are summarized in Figure 1. 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 $Delay(i,j) = |i-j|$. *We can show that a* $\overline{ml}$*-collaboration* f *exists:* $f(1,3) = (3,1)$, $f(1,4) = (3,1)$, $f(2,4) = (2,1)$. *Indeed,* f *satisfies the conditions stated in Def. 1. Condition (a): for Team 3,* $f(1,3) = (3,1)$, *and there exists* $m = 1 \leq 1$ *s.t.* $Borrow\_latest_1(3) = 5 \geq 3+2$. *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* $Borrow\_latest_2(4) = 6 \geq \max\{3+3, 2+2\}$. *Condition (b): for Team 1,* $f(1,3) = (3,1)$ *and* $f(1,4) = (3,1)$, *and there exists* $m = 2 \geq 1+1$ *s.t.* $Lend\_earliest_2(1) = 3 \leq 3$. *Similarly, for Team 2,* $f(2,4) = (2,1)$, *and there exists* $m = 1 \geq 1$ *such that* $Lend\_earliest_1(2) = 2 \leq 2$.

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

> FindCollaboration
> Input: For a set *Lenders* of lender teams, a set *Borrowers* of borrower teams, positive integers $\overline{l}$ and $\overline{m}$: a delay function *Delay* and a collection of functions $Lend\_earliest_m$ and $Borrow\_latest_m$ for every positive integer m ($m \leq \overline{m}$).
> Output: A $\overline{ml}$*-collaboration* between *Lenders* and *Borrowers* with at most $\overline{m}$ robot transfers and within at most $\overline{l}$ steps, relative to *Delay*.

As expected, this problem is intractable:

Proposition 1 *The decision version of* FindCollaboration *(i.e., existence of a* $\overline{ml}$*-collaboration) is NP-complete.*

Intuitively the membership proof is established by guessing and checking f in polynomial time. The hardness proof relies on a polynomial-time reduction from a 3SAT instance F with a atoms and b clauses, to a FindCollaboration problem instance with a lender teams and b borrower teams with $\overline{l} = 2a$ and $\overline{m}$ defined over the number of occurrences of literals in F, and with no delays. Basically, we associate each atom with two time steps (denoting true resp. false); for each clause we define a borrower that can complete its work in $2a$ steps if it can borrow enough robots for at least one time step corresponding to a literal in the clause. We create lenders that can give the required numbers of robots either early (atom is true) or late (atom is false). We configure the number of robots associated with each literal such that a borrower's requirements can only be satisfied by the correct literals. The detailed proof is contained in Appendix B.

4 Finding a Coordination of Teams in ASP

Deciding whether a program in ASP has an answer set is NP-complete (Dantsin et al. 2001); therefore, ASP is suitable for solving FindCollaboration problem. We formalize FindCollaboration in ASP as follows.

The input is represented by a set of facts, using atoms of the forms $delay(i,j,l)$, $lend\_earliest(i,m,l)$, and $borrow\_latest(j,m,l)$ where $i \in Lenders$, $j \in Borrowers$, $m \leq \overline{m}$, $l \leq \overline{l}$.

To formalize conditions (a) and (b) of Def. 1, we introduce atoms of the forms

condition_borrower(j) and *condition_lender*(i). Condition (a) is defined as follows:

$$\begin{aligned} \textit{condition\_borrower}(j) \leftarrow\ & \textit{sum}\langle\{u : f(i,j,l_2,u), i \in \textit{Lenders}, l_2 \leq \bar{l}\}\rangle \geq m, \\ & \textit{max}\langle\{l_1{+}t : f(i,j,l_1,u), \textit{delay}(i,j,t), i \in \textit{Lenders}, u \leq \overline{m}\}\rangle \leq l, \\ & \textit{borrow\_latest}(j,m,l). \end{aligned}$$

where $j \in \textit{Borrowers}$, $l \leq \bar{l}$,$m \leq \overline{m}$. The second line of the rule above computes the number m of robots lent to the borrower team j; the third line computes the latest time step l that team j borrows a robot; and the last line describes that team j needs m robots by step l. Similarly, we define condition (b).

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

$$\{f(i,j,l,u) : l \leq \bar{l}, u \leq \overline{m}\}1 \leftarrow \qquad (i \in \textit{Lenders}, j \in \textit{Borrowers})$$

and then "eliminating" the ones that do not satisfy conditions (a) and (b) of Def. 1:

$$\begin{aligned} &\leftarrow \textit{not}\ \textit{condition\_borrower}(j) \qquad (j \in \textit{Borrowers}), \\ &\leftarrow \textit{not}\ \textit{condition\_lender}(i) \qquad (i \in \textit{Lenders}). \end{aligned}$$

With the ASP formulation of FindCollaboration above, an ASP solver can find an $\overline{ml}$-collaboration.

5 Finding an Optimal Plan for Multiple Teams

Once a coordination of teams is found using an ASP solver for an optimal global plan with $\bar{l} \leq k$ steps, the mediator informs each team how many robots it is expected to lend/borrow to/from which team and when, along with the optimal plan length $\bar{l}$. Taking this information into account, each team computes an optimal local plan with at most $\bar{l}$ steps using an ASP solver, to complete its task, as described in Section 2. The union of these optimal local plans gives us an optimal global plan.

In a naive approach, every team answers $O\big(\overline{m}{\cdot}k^2\big)$ queries, within this overall algorithm. We can improve it by applying binary search between 1 and $\bar{l}$ to find the earliest lend times and the latest borrow times, and between 1 and k to find the optimal value for $\bar{l}$. With this improvement, every team answers $O\big(\overline{m}{\cdot}log(k)^2\big)$ queries.

On the other hand, the computation time to answer a query drastically increases as the plan length increases (due to inherent hardness of planning (Turner 2002; Erol et al. 1995)). In such cases, as suggested by Trejo et al. (Trejo et al. 2001), it is not a good idea to apply binary search to find the optimal value $\bar{l}$ for a global plan. Therefore, a better approach might be to use linear search to find the optimal value $\bar{l}$ for a global plan, and binary search to find optimal values for lending/borrowing times. This approach leads to more number of queries (i.e., every team answers $O\big(\overline{m}{\cdot}k{\cdot}log(k)\big)$ queries) but less amount of computation times as verified by experiments (Table A 2 in Appendix A).

Note that since each team computes an optimal local plan on its own, optimal local plans can be computed in parallel as well.

6 Experimental Evaluation

We investigated the scalability and usefulness of the proposed planning approach (e.g., in terms of quality of solutions) by means of some experiments.

Table 1. *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

We performed some experiments in a variation of the Painting Factory domain described in (Erdem et al. 2012). In this domain, a set of boxes must be manufactured within a given time. To manufacture a box it has to undergo various stages of painting, waxing, and stamping, obeying certain time constraints. There are two types of robots: *worker robots* operate on boxes, they can configure themselves for different stages of process, 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. We assembled teams of different sizes in this domain, so that exchanging worker robots between teams can reduce the time that is necessary to produce the requested amount of boxes.

For our experimental scenarios (see Table 1), we considered team workspaces of $5 \times 3 = 15$ (resp., $8 \times 3 = 24$) grid cells. We varied the number of teams, the number of robots in each team, and the number of boxes that must be manufactured by each team. In each team, for every two worker robots, there is one charger robot. As an example we discuss Scenario 5: three teams must manufacture 12 boxes. Each has a $8 \times 3 = 24$ workspace. The teams consist of 2, 4, resp., 6 worker robots. Accordingly, the teams have 1, 2, resp., 3 charger robots; this yields a total of 18 robots in Scenario 5. The sizes of the workspaces in these instances are reasonable considering real manufacturing processes, since every work cell in a real factory typically is of modest size with 3–12 operators (in our case 2–9 robots per workspace). The number of work cells in a factory ranges drastically from micro factories to large manufacturing plants; with the utilization of parallelization to answer queries, the number of workspaces can be increased further.

We performed experiments on a Linux server with 32 Intel® E5-2665 CPU cores with 2.4GHz and 64GB memory (note that our experiments never use more than 300MB). The overall algorithm described in Section 5 is implemented in PYTHON. The ASP solver CLASP version 2.1.3 (with GRINGO version 3.0.5) is used for answering queries (Section 2) and to solve the collaboration problem (Section 4).

Table 1 shows the results for six scenarios of varying size, averaged over three runs. 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 global plan with/without collaborations of teams. For instance, for Scenario 5,

a total of 171 queries are answered by the teams; average time to answer a query is 5.37 seconds; and finding a coordination of teams takes 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.

7 Related Work

The most related work to ours is on decoupling plans of multiple agents to coordinate their actions (M. M. de Weerdt 2009), and can be summarized in three parts:

Coordination before planning: These methods coordinate the agents before they even begin to plan. Some of them introduce *social laws* the agents must follow (Shoham and Tennenholtz 1995; ter Mors et al. 2004). These laws restrict the actions of agents and can be used to reduce planning and coordination time (e.g., if everyone drives on the right side of the road, no coordination with oncoming cars is required).

Coordination during planning: In these methods, agents find plans for themselves while sharing information about them, and adapt their plans accordingly to avoid conflicts. Partial Global Planning (PGP) framework (Durfee and Lesser 1987) and its extension, Generalized PGP (Decker and Lesser 1994), are examples of these sorts of methods. In these approaches, agents 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. The second agent then tries to improve the global plan. If it can, the improved plan is shown to the other agents who can accept/reject/modify it. Another example of this approach is the Plan Merging Paradigm (Alami et al. 1998), where each robot incrementally builds and executes its own plan taking into account the multi-robot context. There are also coordination methods where the agents exchange subgoals with auctions (van der Krogt et al. 2005).

Coordination after planning: These methods use plan merging. Given the individual plans of all agents, plan merging constructs a joint plan for all agents. Georgeff (Georgeff 1988) proposes a plan-synchronization process starting with individual plans. Stuart (Stuart

1985) uses propositional temporal logic to guarantee that only feasible states of the environment can be reached. Introducing restrictions on individual plans (as in coordination before planning) can be used to ensure efficient merging (Yang et al. 1992; Foulser et al. 1992). Another approach to merging plans is to use A^* search with a smart cost-based heuristic (Ephrati and Rosenschein 1993). There are also task allocation methods that assign the roles of agents for the execution of a given plan (Hunsberger and Grosz 2000).

Our method is different from these related work in the following ways: 1) In our approach, the teams do not communicate with each other, but with a mediator. 2) The communication is not done by passing information about plans or durations of actions: the teams answer the mediator's yes/no questions, ensuring that the teams do not have to share private information (e.g., workspace, tasks, goals) with the mediator; once a coordination is found, then the mediator informs each team about how many robots it should lend/borrow and when, so the teams do not know about which other teams lend/borrow robots. 3) Like the related work, each team computes its own an optimal local plan to complete its task taking into account some extra information; but this extra information is not about other teams (e.g., their plans, actions, tasks). 4) 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. 5) Such a coordination is found iteratively where each iteration involves individual teams' solving various planning problems with complex goals to answer mediator's questions; so determining a feasible coordination goes hand-in-hand with planning. 6) Our method assumes that a team cannot be both a lender or borrower, to ensure a small number of costly transfers of robots between teams; on the other hand, we do not assume that all teams are in the same workspace. Note that once a coordination of teams is found, then an optimal global plan is computed by combining the local plans (as in related work where coordination is done before/after planning).

Our work is more about team work to find a (optimal) global plan, like the related work discussed above, where teams are determined in advance (and in our case costly transfers of robots are not desired), rather than team formation (to decide how or when to join teams) (Gaston and desJardins 2008; Nair et al. 2002).

Our work is also different from the existing approaches on resource allocation in a multi-agent time-constrained domain (Sycara et al. 1991; Chevaleyre et al. 2006; Lin 2011) due to the second item above, because in our method no information is required about plans, ordering constraints on actions, or causal links.

It is important to note here that, the mediator in our approach is a neutral coordinator like in (Ehtamo et al. 1999), though it does not negotiate with the teams but simply gathers information to achieve a optimal global solution. The mediator 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.

Distributed planning for multiple agents with the help of a supervisor, has been studied using action languages (Dovier et al. 2013) and logic programming (Kowalski and Sadri 2013) as well. There are essential differences between these works and the proposed approach: differences in the expressivity of the languages (e.g., the action language in (Dovier et al. 2013) does not allow representation of ramifications, but on the other hand it includes formulas to represent communication messages); the role of the supervisor (in both of the related works, the role of the supervisor is to resolve conflicts; in our work, it

is to decide for efficient use of resources); communication of teams and supervisor (in the related approaches, agents can communicate and negotiate between each other, and the supervisor collects teams' to-be-executed actions to check for conflicts); workspaces (we assume that workspaces are not shared).

8 Discussion

We have introduced a novel method to find an optimal global plan for multiple teams of 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.. We have defined the problem of determining a coordination, and proved its intractability. Using the state-of-the-art ASP solvers, we have evaluated the usefulness of our approach in a cognitive factory setting, and observed a promising decrease in the total process time, which is important for larger cost savings and better customer satisfaction towards economic sustainability.

Lessons learnt The generic nature of this method allows other reasoners and solvers to be used for planning, query answering, or coordination finding. We have used CCALC; because 1) workspaces we consider in a cognitive factory involve concurrency and ramifications (which can be easily formalized in the input language of CCALC), and require external computations in continuous space of robots' configurations (e.g., to check collisions) as in (Erdem et al. 2011; Aker et al. 2012; Havur et al. 2013); and 2) planning problems and queries we consider involve complex goals and conditions. Also, CCALC can be used with a wide range of reasoners, like SAT solvers and ASP solvers.

We have used ASP for finding a coordination of teams, since it provides a concise provably correct description of the problem and the computation times with the state-of-the-art ASP solvers are quite good.

The fact that in our approach each team performs its own computations about completing its own task leads to a highly modular structure of computation and allows computation of local plans and answering queries in parallel.

In addition to the strengths of using these logic-based formalisms from the point of view of representation and efficient reasoning, it is also important that these formalisms are actually being used for challenging robotic applications; making it easier to apply our approach to robotic domains, like in cognitive factories (Erdem et al. 2012).

Future work Our approach can be extended in several ways. If the mediator is allowed to know about the basic tasks and the sorts of transferrable robots, then it can ask questions like "Can your team complete its task in k steps, while also lending a robot that can carry a heavy box, before step k'?". Teams can answer such queries because the background knowledge that associates tasks with robots can be embedded in action descriptions and queries, as in (Erdem and Patoglu 2012; Aker et al. 2012; Erdem et al. 2012). The ASP formulation for finding a coordination can be slightly modified by adding relevant constraints.

Our algorithm to find an optimal global plan can be embedded in an execution monitoring framework. When the plan fails during execution, our algorithm can be called to find an optimal global plan. For subsequent replans, we can reuse the information (e.g., roles of teams and bounds for each team) from the previously computed plan.

Table A 1. *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 SAT)
	#	#	#	#	#	#	sec	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

Appendix A Additional Experiments

A.1 Answering Queries: ASP vs. SAT

We can answer queries using CCalc with a SAT solver or with an ASP solver. In the former case, given an action description and a query, CCalc finds an answer to the query in the spirit of satisfiability planning (Kautz and Selman 1992): 1) it transforms the action description and the query into a set of formulas in propositional logic (Giunchiglia et al. 2004); 2) it calls a SAT solver (like manysat) to find a model of all these formulas; 3) if a model is found then it extracts the solution; otherwise, it answers the query negatively.

In the latter case, given an action description and a query in the language of CCalc, 1) we can use cplus2asp (Casolary and Lee 2011) to transform the action description and the query into an ASP program; 2) an ASP solver (like Clasp with the grounder Gringo) can be used to compute an answer set for the program; 3) if an answer set is found then we can extract the solution; otherwise, the query is answered negatively.

We performed experiments to compare these two approaches, with the same instances used in our experiments, as explained in Section 6. Table A 1 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 1.

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).

Table A 2. *Experimental results comparing two approaches to compute optimal values for plan length $\bar{l}$: linear search vs. binary search*

Search Method	Scenario	Teams	Workspace (grid cells)	Worker Robots	Total Robots	Order (boxes)	Questions (total)	Answering Questions (average time)
		#	#	#	#	#	#	sec
$\bar{l}$: linear	1	2	15	1,2	5	6	212	6.67
	2	3	15	1,2,3	9	9	437	6.13
	3	4	15	1,2,3,4	15	12	525	3.36
	4	2	24	2,4	8	8	127	7.91
	5	3	24	2,4,6	18	12	171	13.08
	6	4	24	2,4,6,8	30	16	293	151.33
$\bar{l}$: binary	1	2	15	1,2	5	6	100	8.78
	2	3	15	1,2,3	9	9	187	9.96
	3	4	15	1,2,3,4	15	12	310	7.78
	4	2	24	2,4	8	8	163	215.13
	5	3	24	2,4,6	18	12	201	224.32
	6	4	24	2,4,6,8	30	16	351	283.63

A.2 Finding Optimal Values: Linear vs. Binary 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.

As discussed in Section 5, 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 $O\big(\overline{m}{\cdot}log(k)^2\big)$ queries.

However, the computation time to answer a query drastically increases as the plan length increases (due to inherent hardness of planning (Turner 2002; Erol et al. 1995)). In such cases, as suggested by Trejo et al. (Trejo et al. 2001), 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 Trejo et al., 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 experimentally over the six scenarios used in our experiments (Section 6), using CCALC with MANYSAT. Table A 2 shows the results of these experiments. Results are averages over three runs.

The first section of the table shows the configuration which was already used in Table 1 and Table A 1: $\bar{l}$ is determined using linear search and within teams the earliest lend and latest borrow times are determined using binary search. The second section of Table A 2 shows the strategy using binary search in both cases.

We can observe that these experimental results confirm Trejo et al. (Trejo et al. 2001)'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.

Appendix B Proofs

Proof of Proposition 1
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 $|Lenders|$, the number of Borrowers $|Borrowers|$, the maximum number of steps $\bar{l}$, and the maximum number of robots $\overline{m}$, respectively.
- D has the form $D_{1,1} \wedge D_{1,2} \wedge ... \wedge D_{a,b}$ 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 $Delay(i, j)$.
- 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:

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

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 $\overline{m} = rNum(step(\neg x_n)) \cdot occ(\neg x_n)$; by eliminating the definition of *rNum* from the formula, we obtain the following equation

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

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:

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

Intuitively, each lender i can lend up to $rNum(step(\varphi(i)))\cdot occ(\varphi(i))$ robots with earliest time step $step(\varphi(i))$ or it can lend up to $rNum(step(\neg\varphi(i)))\cdot occ(\neg\varphi(i))$ 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:

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

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 2 *Figure B 1 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* $rNum(step(t)) = 2^{step(t)}$.

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 $\overline{l}$ 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 $j{+}n$,

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

where clause c_j contains s;

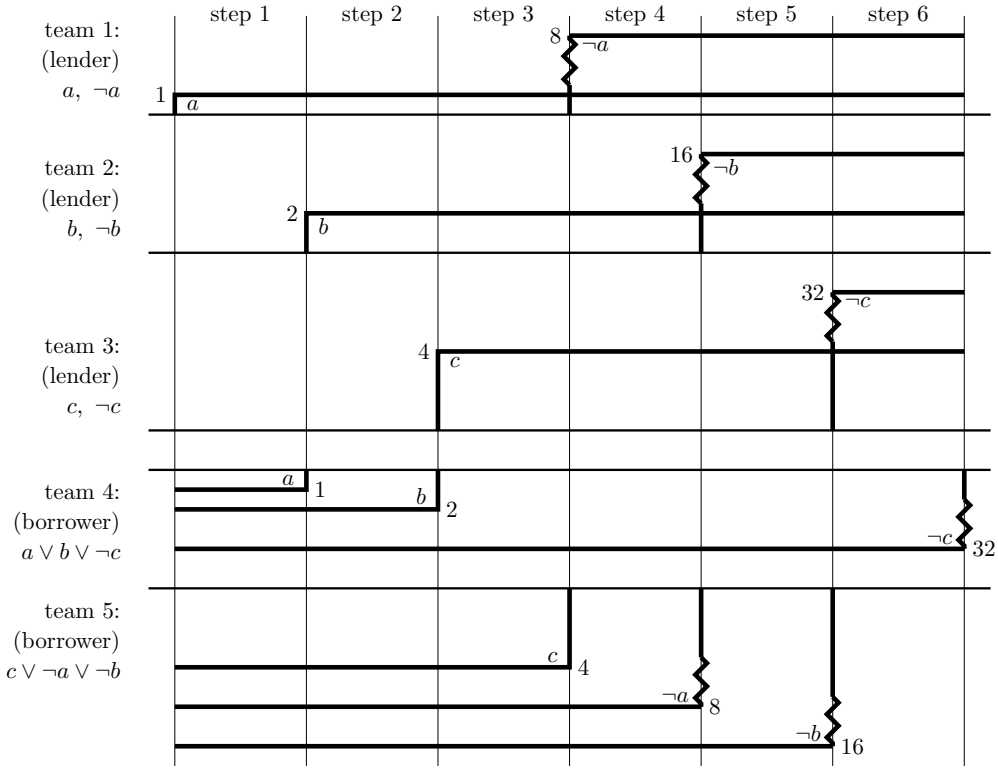

Fig. B 1. Example hardness reduction for 3-SAT formula $F_1 = (a \vee b \vee \neg c) \wedge (c \vee \neg a \vee \neg b)$

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

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

where clause c_j contains $\neg s$.

Example 3 (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 $f(i_k, j+n) = (step(t_{j,k}), rNum(step(t_{j,k})))$ where $step(t_{j,k}) \leq \bar{l}$ and $rNum(step(t_{j,k})) \leq \overline{m}$. Take $m = rNum(step(t_{j,k})) \leq \overline{m}$. Then the following hold:

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

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 $q \leq occ(\varphi(i))$ and $q = occ(\varphi(i))$ if and only if $\varphi(i)$ does not occur multiple times in any clause. The construction of f is as follows:

$$f(i, n{+}j_1) = (step(\varphi(i)), rNum(step(\varphi(i))))$$
$$\vdots$$
$$f(i, n{+}j_q) = (step(\varphi(i)), rNum(step(\varphi(i))))$$

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

$$Lend\_earliest_m(i) = step(\varphi(i)) \leq \min\{step(\varphi(i))\}$$
$$m \geq rNum(step(\varphi(i))) \cdot q.$$

— 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 $q = occ(\neg\varphi(i))$ and $q = occ(\neg\varphi(i))$ if and only if $\neg\varphi(i)$ does not occur multiple times in any clause. The construction of f is as follows:

$$f(i, n{+}j_1) = (step(\neg\varphi(i)), rNum(step(\neg\varphi(i))))$$
$$\vdots$$
$$f(i, n{+}j_q) = (step(\neg\varphi(i)), rNum(step(\neg\varphi(i))))$$

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

$$Lend\_earliest_m(i) = step(\neg\varphi(i)) \leq \min\{step(\neg\varphi(i))\}$$
$$m \geq rNum(step(\neg\varphi(i))) \cdot q.$$

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 → 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 $Delay(i, j) = 0$ 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 4 (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 1 *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 (ctd) *Given f_1 we saw that borrower 5 can complete with respect to $\neg b$. According to Lemma 1, borrower 5 receives at least 1 robot at $step(\neg b) = 5$ which is true as $f_1(2,5) = (5,16)$. Intuitively, Lemma 1 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 $1 \dots 4$. 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 2 which shows that these 2 robots cannot be given in that case.)*

Proof of Lemma 1

- 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 function f with:

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

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

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

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

$$rNum(step(t_1))\cdot occ(t_1) + \dots + rNum(step(t_{u-1}))\cdot occ(t_{u-1}) = \\ = \textstyle\sum_{i=1}^{u-1} rNum(step(t_i))\cdot occ(t_i) = rNum(step(t_u)) - 1$$

  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 $step(t_u)$.

- 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 $rNum(step(\neg t_u))$ robots with the latest time step $step(\neg t_u)$.
  Assume that borrower $j{+}n$ borrows m robots from lender $\varphi^{-1}(var(t_u))$. Then,

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

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

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

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

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

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 2 *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 2 shows that a collaboration corresponds to a *consistent* set of literals satisfying all clauses.

Example 6 (ctd) *Given f_1 borrower 5 can complete with respect to $\neg b$, Lemma 2 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 2

- Case 1: borrower $j{+}n$ can complete its task with respect to positive literal $t \in c_j$.
  By Lemma 1, 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 1, borrower $j'{+}n$ borrows $m \geq 1$ robots from lender $\varphi^{-1}(var(t))$ at step $step(\neg t)$, where

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

  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 2 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 corresonds to a (unique) satisfying truth assignment I for F. □

References

Aker, E., Patoglu, V., and Erdem, E. 2012. Answer set programming for reasoning with semantic knowledge in collaborative housekeeping robotics. In *Proc. of IFAC SYROCO*.

Alami, R., Ingrand, F., and Qutub, S. 1998. A scheme for coordinating multi-robots planning activities and plans execution. In *Proc. of ECAI*. 617–621.

Beetz, M., Buss, M., and Wollherr, D. 2007. CTS - What is the role of artificial intelligence? In *Proc. of KI*. 19–42.

Brewka, G., Eiter, T., and Truszczynski, M. 2011. Answer set programming at a glance. *Commun. ACM 54,* 12, 92–103.

Casolary, M. and Lee, J. 2011. Representing the language of the causal calculator in answer set programming. In *Proc. of ICLP (Technical Communications)*. 51–61.

Chevaleyre, Y., Dunne, P. E., Endriss, U., Lang, J., Lemaître, M., Maudet, N., Padget,

J. A., Phelps, S., Rodríguez-Aguilar, J. A., and Sousa, P. 2006. Issues in multiagent resource allocation. *Informatica 30,* 1, 3–31.

Dantsin, E., Eiter, T., Gottlob, G., and Voronkov, A. 2001. Complexity and expressive power of logic programming. *ACM Computing Surveys 33,* 3, 374–425.

Decker, K. and Lesser, V. 1994. Designing a family of coordination algorithms. In *Proc. of DAI*. 65–84.

Dovier, A., Formisano, A., and Pontelli, E. 2013. Autonomous agents coordination: Action languages meet clp() and linda. *TPLP 13,* 2, 149–173.

Durfee, E. H. and Lesser, V. R. 1987. Planning coordinated actions in dynamic domains. In *Proc. of the DARPA Knowledge-Based Planning Workshop*. 18.1–18.10.

Ehtamo, H., Hamalainen, R. P., Heiskanen, P., Teich, J., Verkama, M., and Zionts, S. 1999. Generating pareto solutions in a two-party setting: Constraint proposal methods. *Management Science 45,* 12, 1697–1709.

Ephrati, E. and Rosenschein, J. S. 1993. Multi-agent planning as the process of merging distributed sub-plans. In *Proc. of DAI*. 115–129.

Erdem, E., Aker, E., and Patoglu, V. 2012. Answer set programming for collaborative housekeeping robotics: Representation, reasoning, and execution. *Intelligent Service Robotics 5,* 4, 275–291.

Erdem, E., Haspalamutgil, K., Palaz, C., Patoglu, V., and Uras, T. 2011. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *Proc. of ICRA*.

Erdem, E., Haspalamutgil, K., Patoglu, V., and Uras, T. 2012. Causality-based planning and diagnostic reasoning for cognitive factories. In *Proc. of IEEE Int. Conf. Emerging Technologies and Factory Automation (ETFA)*.

Erdem, E. and Patoglu, V. 2012. Applications of action languages in cognitive robotics. In *Correct Reasoning*. 229–246.

Erol, K., Nau, D. S., and Subrahmanian, V. S. 1995. Complexity, decidability and undecidability results for domain-independent planning. *Artif. Intell. 76,* 1–2, 75–88.

Foulser, D., Li, M., and Yang, Q. 1992. Theory and algorithms for plan merging. *Artificial Intelligence Journal 57*, 143–182.

Gaston, M. E. and desJardins, M. 2008. The effect of network structure on dynamic team formation in multi-agent systems. *Computational Intelligence 24,* 2, 122–157.

Gebser, M., Kaminski, R., König, A., and Schaub, T. 2011. Advances in *gringo* series 3. In *Proc. of LPNMR*. 345–351.

Gebser, M., Kaufmann, B., Neumann, A., and Schaub, T. 2007. clasp: A conflict-driven answer set solver. In *Proc. of LPNMR*. 260–265.

Gelfond, M. and Lifschitz, V. 1998. Action languages. *Electronic Transactions on Artificial Intelligence 2*, 193–210.

Georgeff, M. P. 1988. Communication and interaction in multi-agent planning. In *Readings in Distributed AI*. 200–204.

Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., and Turner, H. 2004. Nonmonotonic causal theories. *AIJ 153*, 49–104.

Hamadi, Y., Jabbour, S., and Sais, L. 2009. Control-based clause sharing in parallel sat solving. In *Proc. of IJCAI*. 499–504.

Havur, G., Haspalamutgil, K., Palaz, C., Erdem, E., and Patoglu, V. 2013. A case study on the tower of hanoi challenge: Representation, reasoning and execution. In *Proc. of ICRA*.

Hunsberger, L. and Grosz, B. J. 2000. A combinatorial auction for collaborative planning. In *Proc. of ICMAS*. 151–158.

Kautz, H. and Selman, B. 1992. Planning as satisfiability. In *Proc. of ECAI*. 359–363.

Kowalski, R. A. and Sadri, F. 2013. Towards a logic-based unifying framework for computing. *CoRR abs/1301.6905*.

Lifschitz, V. 2002. Answer set programming and plan generation. *Artificial Intelligence 138*, 39–54.

Lifschitz, V. 2008. What is answer set programming? In *Proc. of AAAI*. MIT Press, 1594–1597.

Lin, S.-H. 2011. Coordinating time-constrained multi-agent resource sharing with fault detection. In *Proc. of IEEM*. 1000–1004.

M. M. de Weerdt, B. J. C. 2009. Introduction to planning in multiagent systems. *Multiagent and Grid Systems 5*, 345–355.

Marek, V. and Truszczyński, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*. Springer Verlag, 375–398.

McCain, N. and Turner, H. 1997. Causal theories of action and change. In *Proc. of AAAI/IAAI*. 460–465.

Nair, R., Tambe, M., and Marsella, S. 2002. Team formation for reformation in multiagent domains like robocuprescue. In *Proc. of RoboCup*. 150–161.

Niemelä, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence 25*, 241–273.

Shoham, Y. and Tennenholtz, M. 1995. On social laws for artificial agent societies:off-line design. *Artificial Intelligence 73*, 231–252.

Stuart, C. 1985. An implementation of a multi-agent plan synchronizer. In *Proc. of IJCAI*. 1031–1033.

Sycara, K. P., Roth, S. P., Sadeh, N. M., and Fox, M. S. 1991. Resource allocation in distributed factory scheduling. *IEEE Expert 6,* 1, 29–40.

ter Mors, A., Valk, J., and Witteveen, C. 2004. Coordinating autonomous planners. In *Proc. of IC-AI*. 795–801.

Trejo, R., Galloway, J., Sachar, C., Kreinovich, V., Baral, C., and Tuan, L.-C. 2001. From planning to searching for the shortest plan: An optimal transition. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 9,* 6, 827–837.

Turner, H. 2002. Polynomial-length planning spans the polynomial hierarchy. In *Proc. of JELIA*. 111–124.

van der Krogt, R., Roos, N., de Weerdt, M., and Witteveen, C. 2005. Multiagent planning through plan repair. In *Proc. of AAMAS*. 1337–1338.

Yang, Q., Nau, D. S., and Hendler, J. 1992. Merging separately generated plans with restricted interactions. *Computational Intelligence 8*, 648–676.

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

Zaeh, M., Ostgathe, M., Geiger, F., and Reinhart, G. 2012. Adaptive job control in the cognitive factory. In *Enabling Manufacturing Competitiveness and Economic Sustainability*, H. A. ElMaraghy, Ed. Springer Berlin Heidelberg, 10–17.