Plan-Coordination Mechanisms and the Price of Autonomy

J. Renze Steenhuisen, Cees Witteveen, and Yingqian Zhang

Delft University of Technology,
Faculty of Electrical Engineering, Mathematics and Computer Science,
Mekelweg 4, 2628CD Delft, The Netherlands,
{J.R.Steenhuisen,C.Witteveen,Yingqian.Zhang}@tudelft.nl

Abstract. Task-based planning problems for multi-agent systems require multiple agents to find a joint plan for a constrained set of tasks. Typically, each agent receives a subset of tasks to complete. Due to task interdependencies, such task allocations induce interdependencies between agents as well. These interdependencies will prevent the agents from making a plan for their subset of tasks independently from each other, since the combination of such autonomously constructed plans will most probably result in conflicting plans. Therefore, a plan-coordination mechanism is needed to guarantee a conflict-free globally feasible plan.
In this paper, we first present a brief overview of the main results achieved on plan coordination for autonomous planning agents, distinguishing between problems associated with deciding whether a coordination mechanism is necessary, designing an arbitrary coordination mechanism, and designing an optimal (minimal) coordination mechanism. After finding out that designing an optimal coordination mechanism is difficult, we concentrate on an algorithm that is able to find a (non-trivial) coordination mechanism that is not always minimal. We then discuss some subclasses of plan-coordination instances where this algorithm performs very badly, but also some class of instances where a nearly optimal coordination mechanism is returned.
Hereafter, we discuss the price of autonomy as a measure to determine the loss of (global) performance of a system due to the use of a coordination mechanism, and we offer a case study on multi-modal transportation where a coordination mechanism can be designed that offers minimal restrictions and guarantee nearly optimal performance. We will also place the use of these coordination mechanisms in a more general perspective, claiming that they can be used to reuse existing (single) agent software in a complex multi-agent environment.
Finally, we briefly discuss some recent extensions of our coordination framework dealing with temporal planning aspects.

Key words: Complex tasks, planning, coordination, autonomy, multi-agent systems.

1 Introduction

Task planning is the problem of finding a suitable plan for carrying out a *complex task*. By calling a task *complex*, we mean that *(i)* it consists of a number of elementary tasks that *(ii)* are interdependent, and *(iii)* (usually) cannot be completed by a single agent. For example, building a house constitutes a complex task since it consists of laying a foundation, then building the walls and the roof, then assembling the cases and painting the walls, assembling doors, etc. Typically, each of these tasks requires a different agent. Other examples that come to mind are building a large ship on a wharf, preparing a manned spacecraft for launch, and planning inter-modal transportation jobs.

Usually, we specify such a complex task $\mathcal{T}$ by stating the set T of elementary tasks to be completed, the set of capabilities required to execute each elementary task, and a set of constraints between the tasks that have to be respected in order to ensure a correct execution of the complex task.

The specification of the capabilities is important in deciding which agent will execute which (elementary) task. In this paper, however, we will not pay attention to this important (task allocation) problem[1] and simply assume that task allocation has been completed and that each agent has to find a way to achieve the set of tasks allocated to it. Therefore, we will simply omit the specification of the capabilities required.

A complex task then is a tuple $\mathcal{T} = (T, \prec)$ where T is a finite set of (elementary) tasks and $\prec$ is a partial order. Each elementary task $t \in T$, or simply *task*, is a unit of work that can be executed by a single agent. These elementary tasks are interrelated by a partially-ordered *precedence relation* $\prec$: A task t_1 is said to precede a task t_2, denoted by $t_1 \prec t_2$, if the execution of t_2 may not start until t_1 has been completed.[2] For example, building a wall of a house may not start before the work on the foundations has been completed.

Suppose that such a complex task $\mathcal{T} = (T, \prec)$ is given to a set of autonomous planning agents $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$. We assume that the tasks in T are assigned to the agents in $\mathcal{A}$ by some task assignment $f : T \to \mathcal{A}$, thereby inducing a *partitioning* $\{T_i\}_{i=1}^{n}$ of T, where $T_i = \{t \in T \mid f(t) = A_i\}$ denotes the set of tasks allocated to agent A_i.

Example 1. There are two agents involved in a complex construction task. Agent A_1 has to deliver bricks (t_1) to agent A_2 who will use them to build a wall (t_2). Agent A_2 also has to ensure that garbage is collected (t_3) and to deliver it to agent A_1 (t_4). Agent A_1 has to pickup the garbage (t_5), and then to transport it from the construction place to a dumping ground (t_6).

[1] How to find a suitable assignment for a set of agents is an interesting problem on its own [1, 2].

[2] Of course, this interpretation of the precedence relation might vary. In general, we might interpret $t \prec t'$ as task t should {start, be completed} before task t' is allowed to {start, be completed}.

There are some natural precedence relations between the tasks that must be obeyed: $t_1 \prec t_2$, $t_3 \prec t_4$, $t_4 \prec t_5$, and $t_5 \prec t_6$. For an illustration of this complex task, see Figure 1.

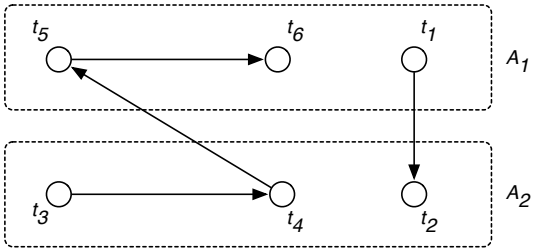

Fig. 1. A specification of a complex task. The tasks are allocated to two agents: A_1 and A_2. Task dependencies are denoted by arrows.

As a result of this task assignment, each agent A_i also inherits the precedence constraints that apply to T_i, i.e., the set $\prec_i = \prec \cap (T_i \times T_i)$. These sets $\prec_i$ together constitute the set $\prec_{intra} = \bigcup_{i=1}^{n} \prec_i$ of *intra-agent* constraints, while the remaining set of constraints $\prec_{inter} = \prec \setminus \prec_{intra}$ constitutes the set of *inter-agent* constraints. So each agent A_i is now responsible for achieving the (complex) subtask $(T_i, \prec_i)$, while the agents depend on each other via the inter-agent constraints $\prec_{inter}$.

Example 2. Continuing Example 1, as the result of task allocation, the set T is partitioned into two sets $T_1 = \{t_1, t_5, t_6\}$ and $T_2 = \{t_2, t_3, t_4\}$. The set of inter-agent constraints is $\prec_{inter} = \{t_1 \prec t_2, t_4 \prec t_5\}$, while the intra-agent constraints are $\prec_1 = \{t_5 \prec t_6\}$ and $\prec_2 = \{t_3 \prec t_4\}$.

Due to time and resource constraints, an agent A_i will be forced to make a *plan* for its set of tasks T_i. Such a plan should be understood as the specification of some partially-ordered set of actions (plan steps) that satisfies the task constraints between the tasks given to the agents. It is not important to know exactly which planning tools are used by the agent and which set of primitive actions is used to construct the plan: What counts is that the plan respects all the task constraints $\prec_i$. As a consequence, we assume that, whatever plan/schedule representation the agents (internally) employ, the result of an internal plan P_i, developed by agent A_i for its set of tasks T_i, can always be specified by a structure $P_i = (T_i, \prec_i^*)$ that *refines* the structure $(T_i, \prec_i)$. We, therefore, require $\prec_i \subseteq \prec_i^*$ to hold, which means that an agent's plan must respect the original precedence relation $\prec_i$, but his plan may of course induce additional constraints.

Remark 1. The plan representation $P_i = (T_i, \prec_i^*)$ may be viewed as an *abstraction* of a concrete original plan P_i^c. Suppose that the concrete plan can be modelled as a partially-ordered set S of plan steps, i.e., $P_i^c = (S, <)$. Then we say that $P_i = (T_i, \prec_i^*)$ is an abstraction of P_i^c if there exists some function $steps : T_i \rightarrow 2^S$

mapping tasks to plan steps, such that *(i)* for every task $t \in T_i$, the concrete sub plan $(steps(t), <)$ of P_i^c realises task t, and *(ii)* for every $t, t' \in T_i : t \prec_i^* t'$ iff $\forall s \in steps(t)\ \forall s' \in steps(t') : s < s'$. Note that such a concrete plan might contain plan steps s that have no relationship with any task t.

Example 3. Continuing our previous example, our agent A_1 who has to deliver bricks (t_1) and to pickup the garbage (t_5), and then to carry it away (t_6) might construct a plan where he first drives with his truck to the construction place, will pickup the garbage, drives to the dumping place, takes some coffee, then loads some bricks and drives back to the construction place. This plan induces the following order on the tasks to be completed: $t_5 \prec t_6 \prec t_1$. Hence, his plan can be represented as $P_1 = (T_1, \prec_1^*)$ where $\prec_1^* = \{t_5 \prec_1^* t_6, t_6 \prec_1^* t_1\}$ is a refinement of $\prec_1$ and, therefore, is a valid plan for $(T_1, \prec_1)$.

Often, when more than one agent is involved in the task-planning process, we have to take into account some degree of *autonomy* that each of the participating agents might require when planning its part of the job. Here, autonomy has to be understood in the sense that an agent A_i is not able to predict exactly how the plan P_j of another agent A_j will look like for the set of tasks T_j given to that agent, and vice versa. However, as the result of allocating subsets of tasks to different agents, these agents might become interdependent as interdependent tasks might be allocated to different agents. The result of, on the one hand, *interdependency* and, on the other hand, *unpredictability* of individual planning outcomes, might easily lead to *conflicting* plans, in the sense that the structure $P = (T, (\bigcup_{i=1}^{n} \prec_i^*) \cup \prec_{inter})$, resulting from joining the individual plans, is no longer partially ordered. If this is the case, we call the resulting joint plan P *infeasible*.

Example 4. Continuing the example, suppose that agent A_2 develops a plan completely independent from agent A_1. This agent might decide to execute t_2 before t_3 and creates a plan $P_2 = (T_2, \prec_2^*)$ where $\prec_2^* = (t_2 \prec_2^* t_3, t_3 \prec_2^* t_4)$. The result of this plan together with the plan P_1 of agent A_1 (see Example 3) is depicted in Figure 2. As can be seen, the two plans together create a cycle and, therefore, constitute an infeasible joint plan.

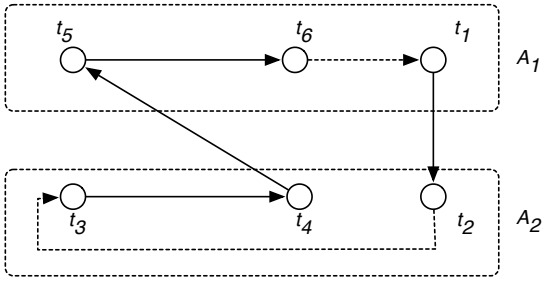

Fig. 2. Two plans of the agents, resulting in an infeasible joint plan.

Obviously, due to this combination of task dependencies and independent planning, some form of *coordination mechanism* [3] is needed to ensure that the results of the individual task-planning processes are jointly feasible.

In this paper, we will address the following *plan-coordination problem* for self-interested planning agents: How to provide adequate coordination mechanisms for autonomous planning systems that want to plan independently and are not willing or able to revise their plans. In particular,

1. we present a framework for studying this plan-coordination problem,
2. we investigate the computational complexity of designing such plan-coordination mechanisms, and
3. we discuss the *price of autonomy*, investigating the additional costs incurred by independent planning.

Organisation. This paper is organised as follows. First, we present a brief overview of research on (plan) coordination to make clear what the relation is between our approach to plan coordination and other approaches. Second, we present our framework for plan coordination for self-interested agents and we give an overview of the complexity results obtained for designing adequate plan-coordination mechanisms. To prepare the reader for a case study of the application of coordination in logistic problems, we discuss a polynomial algorithm that can be used to find a suitable coordination mechanism. We prove that for some particular classes of complex tasks it delivers a nearly optimal mechanism. After introducing the price of autonomy, we apply this algorithm to a logistic problem, showing that in some particular cases plan-coordination mechanisms can be applied with almost negligible overhead, while ensuring autonomous planning.

2 Plan Coordination for Autonomous Planning Agents

2.1 Background

In general, a plan-coordination mechanism should guarantee, by possibly *redesigning* the original planning task, some minimal overall performance even if the agents are completely selfish [3].

Redesigning the original planning task usually imposes additional restrictions on the tasks to be completed in order to guarantee a minimal performance. In general, such additional restrictions will possibly affect both the planning freedom of the participating agents and the quality (cost, efficiency) of the plans they are able to develop. Therefore, we propose to define the quality of a plan-coordination mechanism both in terms of the *tightness* of the restrictions imposed, and the overall *plan quality* it ensures. Such a definition, however, would neglect an important factor that influences the choice of an adequate coordination mechanism: the *collaboration level* between the agents. Depending on this collaboration level we have to determine which coordination mechanism is suitable for the agents and this also determines the quality of the (optimal) coordination mechanism. For example, in approaches like [4, 5] the authors propose

to manage the coordinated planning and execution of the tasks by letting the agents keep each other informed about any changes (e.g., completed, new, or re-scheduled tasks). Here, a plan-coordination mechanism can be designed to facilitate inter-agent information exchange in order to improve the quality of the joint plan.

In other approaches, like the plan-merging approach [6], plans sometimes need to be revised when merging them into an overall plan, despite that agents are allowed to construct their plans independently. In this case, a plan-coordination mechanism could facilitate the exchange of mutual plan information in order to improve the overall plan quality by merging techniques.

It is clear that these approaches require the agents to be more or less *collaborative*, each agent willing to *inform* other agents about details of its individual plan and/or willing to *revise* its plan when necessary. However, such approaches are hardly usable if the agents are selfish (unwilling to communicate or revise their plans), or are not able to do so (e.g., in disaster-rescue operations, when communication is often impossible or difficult to establish).

Approaches to coordinating *self-interested, non-cooperative* agents, however, are mostly concentrating on task execution processes that do not require extensive forms of planning. Typical examples here are the study of (combinatorial) auctions for task allocation and coalition formation processes in multi-agent systems [1, 7, 8] and its relations to combinatorial optimisation problems (c.f. [9]). In these approaches, the tasks to be completed consist of a set of atomic tasks and each of the agents receives one single task of a subset of tasks. Even if the task description is more elaborate like in the Traderbots architecture [10], it is assumed that the set of subtasks does not require an elaborate planning process to execute. Therefore, the problem of identifying *planning constraints* and the problem of allocating them do not occur here.

2.2 The plan coordination problem for self-interested agents

In contrast to the above mentioned approaches, some authors [11, 12] studied the (computational) properties of coordination mechanisms that can be used for *selfish planning* agents. In such approaches, where the collaboration level is low or even absent, it is assumed that the agents *(i)* require autonomous planning of their part of the task, and *(ii)* are not willing to revise their own plan, thereby ruling out any form of collaboration either during planning or after planning. To meet such requirements, a plan-coordination mechanism should ensure that whatever plans are proposed by the individual agents, their combination always constitutes a feasible plan for the total set of tasks. The basic setup of such an approach has been discussed in the Introduction. Summarizing, the main ingredients of this framework are as follows:

1. We have a complex task $\mathcal{T} = (T, \prec)$, specifying a partially-ordered set of elementary tasks $t \in T$, and a set of self-interested agents $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ that require planning autonomy.

2. The tasks $t \in T$ are allocated to the agents A_i inducing a *task partitioning* $(\{T_i\}_{i=1}^n, \{\prec_i\}_{i=1}^n, \prec_{inter})$ where $(T_i, \prec_i)$ is a partially-ordered complex task allocated to agent A_i. Here, $\prec_i$ is the restriction of $\prec$ to $(T_i \times T_i)$, and $\prec_{inter}$ is the set of precedence relations between tasks allocated to different agents.
3. Each agent A_i is allowed to construct a plan P_i for its set of tasks T_i completely independent from the other agents. We assume that each such a plan P_i is representable as a partial order $P_i = (T_i, \prec_i^*)$ where $\prec_i \subseteq \prec_i^*$ (i.e., each plan P_i respects the *local* constraints $\prec_i$).

Now, the coordination problem we are facing can be stated as follows.

How to ensure that, whatever plans $P_i = (T_i, \prec_i^*)$ are developed by the individual agents A_i, each respecting the local constraints $\prec_i$, their combination, together with the set of inter-agent constraints, constitutes a feasible plan, that is, how to ensure that for every i and for every partially-ordered extension $\prec_i^*$ of $\prec_i$, the relation $(\bigcup_{i=1}^n \prec_i^*) \cup \prec_{inter}$ again is a partial order?

As we have shown before [11, 12], the only way to solve this problem is to design a coordination mechanism that adds, to each set of individual precedence constraints $\prec_i$, a set Δ_i of precedence constraints. The resulting set $\Delta = \bigcup_{i=1}^n \Delta_i$ is called a *coordination set* and the redesigned complex task is said to be *plan coordinated*.

Example 5. Let us consider the construction task from the previous examples. As we have shown in Example 4, there exists some combination of plans that turns out to be infeasible, creating a cycle. Therefore, this particular complex task instance is not plan coordinated. If, however, we change the task specification for agent A_1 adding the coordination set $\Delta = \Delta_1 = \{t_1 \prec_1 t_5\}$ to the complex task, no possible combination of plans developed by agent A_1 and A_2 will create a cycle (see Figure 3).

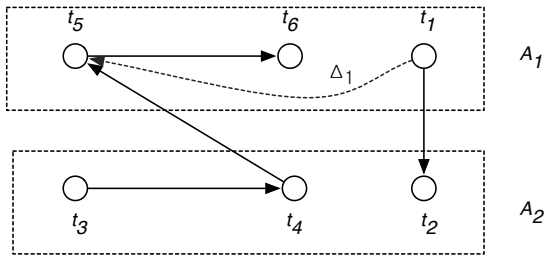

Fig. 3. Adding a precedence constraint prevents conflicting plans.

2.3 Complexity of designing plan coordination mechanisms: some results

The results of this approach to plan coordination for self-interested agents can be summarised as follows:

1. **Verifying whether a plan-coordination mechanism is necessary.** It is intractable (coNP-complete) to decide whether or not a redesign of the planning task is necessary to ensure feasibility of the joint plan (i.e., the problem to verify whether it is needed to design a plan-coordination mechanism is intractable)[11]. This holds already for instances with a few (at least 4) tasks per agent. If, however, the number of agents is fixed, the problem is polynomially decidable [12].
2. **Designing an arbitrary coordination mechanism.** It is always possible to find a (trivial) plan-coordination mechanism in polynomial time that guarantees the existence of a joint conflict-free plan. So the problem of finding an arbitrary plan-coordination problem is in P. The coordination mechanism simply specifies some additional constraints to the complex task to be solved in such a way that for every agent the set of tasks it has to complete is totally ordered [11].
3. **Designing a minimal coordination mechanism.** The construction of a minimal plan-coordination mechanism (minimally redesigning the original complex task specifications) is a highly complex task in itself. Even for instances where the agents are assigned 2 tasks, it is already an NP-complete problem, while in general, the problem of finding a minimal coordination mechanism is Σ_2^p-hard, even if the agents have a modest number of tasks (7 or more) to complete [12].

3 A Polynomial Algorithm to Achieve Plan Coordination

Since the problem of finding a minimal coordination set Δ is too complex to solve in reasonable time (unless P = NP), and the trivial solution to the coordination problem generates an inflexible solution, in this section, we investigate an algorithm to produce a coordination set that is more flexible than the trivial one, but not necessarily an optimal one. First, we show that the algorithm indeed may perform very badly on some instances of the coordination problem. Then we present a class of instances where the algorithm produces near-optimal results.

The algorithm, called the *Depth-Partitioning Algorithm*, is based on the simple construction described in Algorithm 1. This algorithm is capable of making any complex task plan coordinated.

Example 6. Let us consider the application of the algorithm to the complex task $\mathcal{T}$ discussed in Example 1 and presented in Figure 1. There are six tasks $t_1, \ldots, t_6$ involved. In Figure 4, the depths of the tasks are indicated. Since the tasks belonging to agent A_1 are t_1, t_5, t_6, their different depths induce the set $\Delta_1 = \{t_1 \prec_1 t_5,\ t_5 \prec_1 t_6\}$ of constraints and as a result, the (new) precedence tuple $t_1 \prec_1 t_5$ is added to $\prec_1$ (indicated by the dashed arrow).

Likewise, since t_2, t_3, t_4 belong to agent A_2 and the depth of t_3 is smaller than the depth of t_4 and t_2, $\Delta_2 = \{t_3 \prec_2 t_2,\ t_3 \prec_2 t_4\}$ and the (new) tuple $t_3 \prec_2 t_2$ is added to $\prec_2$, again indicated by the dashed arrow. As can easily be

Algorithm 1 Depth-Partitioning Algorithm.

1. Take the partially-ordered complex task $\mathcal{T} = (T, \prec)$ and consider the subsets $T^d = \{t \in T \mid \mathit{depth}(t) = d\}$ of tasks having the same depth in $\mathcal{T}$. Here, the depth $\mathit{depth}(t)$ of a task t is defined as follows: If t does not have predecessors in $\prec$ then $\mathit{depth}(t) = 0$, else $\mathit{depth}(t) = 1 + \max\{\mathit{depth}(t') \mid t' \prec t\}$. The depth $\mathit{depth}(T)$ of the set of tasks T is the maximum value of $\mathit{depth}(t)$ for a $t \in T$. Note that $\{T^d\}_{d=0}^{\mathit{depth}(T)}$ is a partitioning of T.
2. Consider the task partitioning $\{T_i\}_{i=1}^n$ of T induced by the task allocation to the agents $\{A_i\}_{i=1}^n$ and let T_i^d denote the set of tasks $t \in T_i$ such that $\mathit{depth}(t) = d$.
3. For every agent A_i, let $(T_i^{d_1}, T_i^{d_2}, \ldots, T_i^{d_k})$ be the sequence of all (non-empty) sets T_i^d sorted in increasing values of the depth value d_j.
4. For every agent A_i, let $\Delta_i = \{t \prec_i t' : (t, t') \in T_i^{d_j} \times T_i^{d_{j+1}},\ j = 1, 2, \ldots, k-1\}$ (i.e., all tasks of an agent A_i occurring at a lower depth are required to precede all tasks occurring at a higher depth).
5. Output $\Delta = \bigcup_{i=1}^n \Delta_i$.

seen, the instance is plan coordinated, since traversing the arrows, the depth of the tasks can never decrease and if a local plan constraint $t \prec_i t'$ is added, it can only be added if $\mathit{depth}(t) < \mathit{depth}(t')$. Hence, the task instance remains partially ordered whatever locally feasible plan constraints are added by the agents.

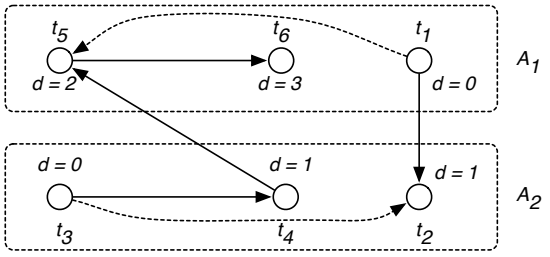

Fig. 4. Applying the Depth Partitioning Algorithm to the task discussed in Example 1. The dashed arrows indicate the precedence constraints added by the algorithm.

Proposition 1. *Let $\mathcal{T} = (T, \prec)$ be a complex task and Δ the set of additional precedence constraints induced by the depth-partitioning algorithm given above. Then the complex task $\mathcal{T}' = (T, \prec \cup \Delta)$ is plan coordinated.*

Proof. See Appendix A.

The quality of the coordination mechanism produced by an algorithm can be expressed as the number of constraints added by the algorithm versus the number of constraints added by a *minimal* coordination mechanism. We will call this ratio, a number greater than or equal to 1, the *performance ratio* of the algorithm. The closer this ratio is to 1, the better the algorithm is. It is not

difficult to see that our simple algorithm can produce arbitrary bad coordination sets, which is illustrated by the following example.

Example 7. Consider the following complex task with $n+1$ agents, each having 2 tasks. Each agent A_i, for $i = 1, 2, \ldots, n$, has one task that precedes the task a belonging to agent A_{n+1}, while agent A_{n+1} also has a second task b that has to precede all (second) tasks of the other agents. In Figure 5, such a complex task is depicted for $n = 6$.

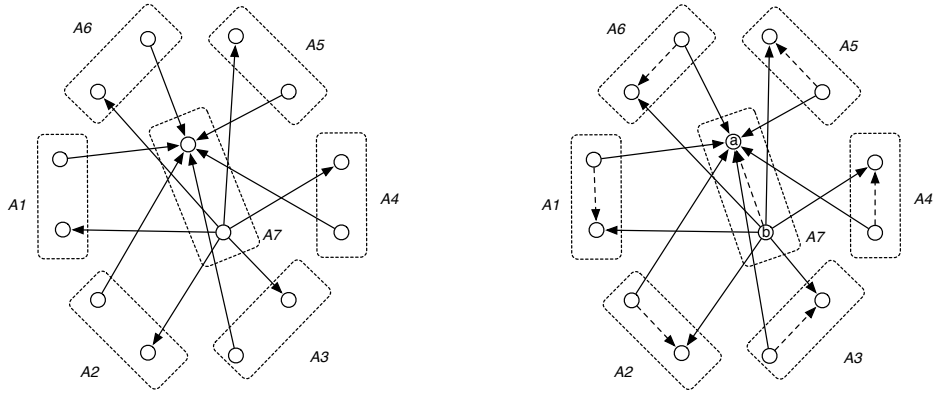

Fig. 5. A complex task (a) where the depth-partitioning algorithm performs badly, adding 7 constraints (indicated by the dashed arcs, see (b)) where only one (from task b to task a) is needed.

If we add one constraint between task b and task a for agent A_{n+1} then the instance is plan coordinated: Every inter-agent cycle is prevented by adding this constraint. On the other hand, every task t in this partially-ordered set of tasks has either depth 0 or depth 1. Moreover, every agent has exactly one task of depth 0 and one task of depth 1. Therefore, the algorithm will produce $n+1$ additional constraint arcs, for every agent one constraint. This shows that the performance ratio of this algorithm is $\frac{n+1}{1}$ and, hence, not bounded by a constant.

3.1 Task Chains

Although the algorithm produces coordination sets that are far from optimal in general, special cases exist where the algorithm produces (nearly) optimal results. One such a class is the class of complex tasks where the partial order defined on T generates a series of parallel chains each of depth d for some constant d. An example of such a set of chains and a partitioning of the tasks is given in Figure 6. Within this class we distinguish *left-right* and *right-left* chains and a partitioning such that an agent A_i, with $i = 0, 1, \ldots, d$, has tasks of depth i in left-right chains and tasks of depth $d - i$ of right-left chains (see Figure 6). Let us call a set of such chains of depth d a d-instance. If such a d-instance contains

k left-right and m right-left chains, we will call such a set a (d, k, m)-instance. Note that the total number of tasks in such a set is $|T| = (d + 1) \times (k + m)$.

It is not difficult to see that the minimum number of additional constraints to make a $(1, k, m)$ set coordinated is $k \times m$: In order to prevent cycles, we have to add an arc between the beginning of a left-right and the end of every right-left chain or vice-versa. Proceeding inductively, suppose that we add 1 to the length of every chain in a given (d, k, m)-instance with $d + 1$ agents and we create an additional agent to take care of the new tasks. Furthermore, suppose that the original (d, k, m)-instance was already coordinated. Then it is not difficult to see that again $k \times m$ coordination arcs have to be added in order to make this instance plan coordinated. This implies the following result.

Observation 1 *The minimum size of a coordination set to make a (d, k, m)-instance plan coordinated is $d \times k \times m$.*

We will show that the Depth-Partitioning Algorithm (Algorithm 1) produces minimal coordination sets for some (d, k, m)-sets. Note that the set of tasks of every agent (except the middle agent if d is even)[3] consists of exactly two subsets of tasks of *different* depth. If d is odd, the algorithm will, therefore, return a set of $(d + 1) \times k \times m$ additional constraints. If d is even, there is exactly one agent whose tasks all have the same depth. As a result, the algorithm will return a set of $d \times k \times m$ additional arcs. Hence, we have the following result.

Proposition 2. *The depth-partitioning algorithm returns an optimal coordination set for every (d, k, m)-set, $d \geq 1$, where d is an even number. If $d \geq 1$ is odd, the performance ratio of the algorithm is $\frac{d+1}{d} \leq 2$.*

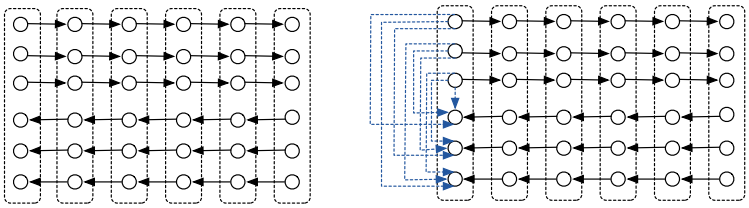

Fig. 6. A set of parallel chains (a $(5, 3, 3)$-set) as a complex task (a). In (b), the set Δ_1 of additional constraints for agent A_1 is shown.

In particular, these performance measures do not change if, instead of one agent A_i for every set of tasks at level i in left-right chains and depth $d - i$ in right-left chains, we have more than one agent for such a sets of tasks. One such example is the *logistic planning* problem. Before we discuss this case, we will introduce the price of autonomy to qualify coordination mechanisms in a more detailed way.

[3] Remember that d starts from 0. If d is even, the length of a chain and the number of agents is odd.

4 The Price of Autonomy and an Application to Logistic Planning

A coordination mechanism like the one we discussed in the previous section guarantees that a set of agents can plan independently. It realises this guarantee by imposing additional restrictions on the complex task to be completed. These additional restrictions can be seen as one aspect of the *price of autonomy*, since they restrict the planning freedom of the participating agents. One way to reduce the price of autonomy in this respect is to look for a *minimum* number of additional constraints. Taking into account the *loss of freedom*, however, is only one aspect of the price of autonomy.

Another, as important, behavioural aspect of autonomous planning is the *loss of performance* it might incur: Due to autonomous planning, the joint plan that is composed from the independently developed individual plans might have a significantly higher cost than an optimal plan for the complex task when assuming non-autonomous planners. So assuming an optimal (i.e., minimum) coordination mechanism that can be found efficiently, we could define the price of autonomy ρ w.r.t. the *performance* of the planning system as the ratio

$$\rho = \frac{\text{cost of joint plan composed of individually optimal plans}}{\text{cost of optimal joint plan}} \tag{1}$$

Here, the cost of a joint plan refers to the cost of the *concrete* joint plan of all agents together (see Remark 1). This implies that in order to establish the price of autonomy we have to investigate a concrete planning domain, where the set of plan steps used to compose plans is known and the cost of plans based on ordering these plan steps can be determined. Here we have chosen a simple *logistic planning* domain to investigate the price of autonomy.

4.1 Logistic Planning Problems

The logistic planning problem we have in mind consists of a triple (L, C, O) where L is a set of $m \times n$ locations $l_{i,j}$ and C is a set of n cities c_i. Each city c_i is a subset $c_i = \{l_{i,j} \mid j = 1, 2, \ldots, m\}$ of m locations in L. $O \subseteq (L \times L)$ is a set of orders $o = (l, l')$, indicating that a certain package has to be transported from (pickup) location l to (delivery) location l'. All locations belonging to a city c_i are interconnected via direct links. In each city c_i, we distinguish a special location $l_{i,1}$ as the *airport* of city c_i. All airports are assumed to be interconnected via direct flights. See Figure 7 for a simple illustration.

We assume a truck to be available in each city to carry packages from one location in the city to another. There is also a plane available carrying packages from one airport to another. Both trucks and planes can carry any amount of packages. We distinguish *intra-city* orders and *inter-city* orders. An intra-city order requires a *load action* at the pickup location, a *move action*, and an *unload action* at the destination location in the same city. So the cost of an intra-city order is minimally 3 actions. For an inter-city order, we distinguish a *pre-order*

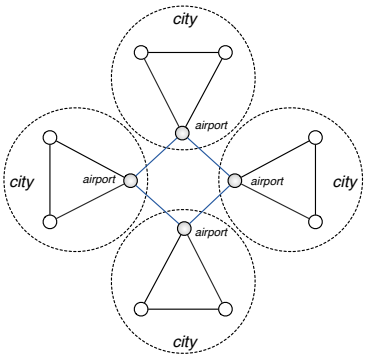

Fig. 7. A logistics example with 4 cities and 3 locations per city. No orders are specified.

phase, a *plane-phase*, and a *post-order* phase. In the pre-order phase, an intra-city order is carried out by transporting the package to the airport of the pickup city. In the plane-phase, the package is transported to the destination airport. In the post-order phase, the package is transported from the airport to its final destination. So an inter-city transportation might require at least 6 load/unload actions and at least 3 move actions. We use a simple uniform cost model where every load, unload and move action has unit cost. Hence, the cost of a plan simply equals the number of actions it contains.

Given an instance (L, C, O) of this logistic planning problem we are looking for a plan P that carries out all orders in O. Such a plan is a sequence of load/unload and move actions completing all the orders in O. The cost of P, denoted by $c(P)$, is the sum of the cost of all (move, load and unload) actions occurring in P and of course, we would like to obtain an optimal plan P^*, i.e., a plan with minimum cost.[4]

4.2 Coordinating the Logistic Planning Problem

Note that an instance (L, C, O) of the logistic planning problem can be easily translated into a complex task $(T, \prec)$: Every inter-city order o_j consists of a linearly-ordered sequence of three elementary tasks $t_{j1} \prec t_{j2} \prec t_{j3}$, where t_{j1} is a truck task of transporting the package from its pickup location to the airport, t_{j2} is the plane task of transporting the package to the airport of the destination city, and finally t_{j3} is a truck task consisting in transporting the package to its destination location. Note that the task assignment is trivial here, since the pre- and post-order phase of an order $o \in O$ are assigned to the truck agent in the city where the pick-up and delivery location belong to, respectively, and the plane-phase orders are assigned to the plane agent. Note that the set O of orders induces a 2-instance (an instance containing chains of length 3) of size $|O|$ in the complex task $(T, \prec)$.

It is easy to see that a feasible joint solution is not guaranteed if the truck and plane agents plan independently from each other. Applying the Depth-

[4] Note that we are not looking for a minimum time plan.

partitioning algorithm on this 2-set, it is easy to see that the set of orders O_i of every truck agent T_i is partitioned into two sets O_i^0 and O_i^2: The first set is the set of all pre-order transportations tasks (of depth $d = 0$ to the airport of the city, and O_i^2 is the set of all post-order transportation tasks of depth $d = 2$. The Depth-partitioning algorithm forces the addition[5] of the constraints $t \prec t'$ for every $t \in O_i^0$ and $t' \in O_i^2$. Since the complex task induced by the logistic planning problem is a $(2, k, m)$-instance, from the discussion in Section 3.1, it can be easily seen that this set of additional constraints is a minimum set.

The Depth-partitioning algorithm now guarantees that every plan developed by any of the truck agents and every plan developed by the plane agent can be joined together to constitute a feasible plan for the complex task. Moreover, for this particular logistic problem, it guarantees that the coordination mechanism imposes a minimum amount of additional constraints.

We would like to determine the price of autonomy ρ, that is the cost incurred by the coordination mechanism. This cost measure is determined by the sum of the costs of the individually optimal plans developed by the agents versus the cost of the globally optimal transportation plan. Apparently, the cost of the optimal plan seems to be fixed: Every transportation order requires 6 load and unload actions and 3 move actions, so the cost of n transportation orders would be $\geq 9n$ using a uniform cost model. This line of reasoning, however, does not recognise that we can often build cheaper plans by carefully combining pickup and delivery orders. For example, if a truck has to bring m packages from city locations to the airport, a worst-case plan would be first to drive to the first pickup location then to bring each package to the airport separately, incurring a cost of 1 pickup + 1 move + 1 delivery = 3 actions per order + m move actions to go to the pickup locations. This will result in a plan cost of $3m + m = 4m$. A better plan, however, would be first to drive to all pickup locations, picking up all packages ($2m$ actions) then to drive to the airport (1 move action) and finally to unload the truck (m actions). This requires a plan with cost at most $3m+1$. Likewise, the cost of an optimal plane plan depends on finding a smallest number of move actions required to satisfy all transportation orders.

4.3 Visiting Sequences

To determine the price of autonomy, we have to determine the sum $c(P^*_{loc})$ of the costs of the independently constructed (locally) optimal plans and the cost $c(P^*)$ of the optimal plan. From the discussion above, we conclude that to find these optimal plans, we have to find the minimal number of move actions in each of these plans. This number of move actions required can be determined by introducing the notion of a *visiting sequence*.

Let $L = \{l_1, l_2, \ldots, l_n\}$ be a set of locations in a given city (or a set of airports) and let $O \subseteq (L \times L)$ be a set of pickup-delivery orders over L. We assume that all locations are directly connected and the cost to travel from c_i to c_j is the same (one move action) for all $i \neq j$.

[5] The plane tasks, however, all belong to one partition.

A (move) plan for carrying out the set of orders O can be represented by a sequence S of locations over L such that all pickup-delivery orders can be carried out by visiting the locations in the order indicated by S. Such a *visiting sequence* S is a sequence over L with possible repetitions. An order $(l, l') \in O$ is fulfilled by a sequence S if there exists sub-sequences α, β, γ over L such that $S = \alpha l \beta l' \gamma$ (i.e., S contains an occurrence of l before an occurrence of l'). A visiting sequence S over L is a *solution* to the instance (L, C, O) if all orders in O are fulfilled by S.

The problem to find an arbitrary visiting sequence solution S for a set of orders O is an easy problem: Let S' be an arbitrary sequence where every city in L occurs exactly once and consider the concatenation $S = S' \circ S'$. Since for every $(l, l') \in O$ an occurrence of l in the first subsequence and an occurrence of l' in the second subsequence can be selected, clearly, S is a solution. However, to find an *optimal* solution S^* (i.e., a visiting sequence of minimum length) is intractable. This *minimal visiting sequence* problem is an NP-hard problem since the NP-hard *Feedback Vertex Set* (FVS) [13] is polynomially reducible to it.

Without loss of generality, we may assume that for every visiting sequence solution S for O it holds that no location l mentioned in O occurs more than twice in S. Otherwise, just keep the first and the last occurrence of l in S and we still have a solution. Therefore, for every visiting sequence solution S, it should hold that $|S| \leq 2 \times |S^*|$, since every location l in O should appear at least once in S^*.

4.4 Determining the Price of Autonomy

Note that the length of a visiting sequence S can be used to determine the cost of a plan P for a (truck or plane) planning problem: If S is the visiting sequence solution for the set of orders, the cost $c(P)$ of the corresponding transportation plan P equals $c(P) = 2 \cdot |O| + |S|$ (i.e., when taking into account one additional load and one unload action per action, each of unit cost).

We can now determine the price of autonomy for the logistic planning problem as follows: Let p be the number of orders, n the number of cities and $m+1 \geq 2$ the number of locations per city.[6] Without loss of generality, we may assume that each location in each city is mentioned at least once in the set of orders. Therefore, $p \geq 0.5(n \times m)$, because every order specifies two different locations (not equal to the airport).

Let us first consider the cost $c(P^*_{loc})$ of the combination of locally optimal plans. Each order requires 6 load and unload actions, so we need $6p$ load and unload actions in total. Suppose that in city c_i we have m_{i1} pickup locations and m_{i2} delivery locations. Clearly, we should have $m \leq m_{i1} + m_{i2} \leq 2m$. To transport packages to the airport we need at most $m_{i1} + 1$ move actions per city and to transport them from the airport we need m_{i2} move actions per city.

[6] We assume that all orders create a transportation chain of length 3, which implies that every order requires transportation to and from an airport.

This means at least $m+1 \leq m_{i1}+m_{i2}+1 \leq 2m+1$ move actions per city c_i. To transport the packages by plane, we need a minimal visiting sequence S_n^* for pickup and delivery at the n airports. Hence, the total cost of the optimal joint plan P_{loc}^* allowing independent planning equals

$$c(P_{loc}^*) = 6p + \sum_{i=1}^{n}(m_{i1}+m_{i2}+1) + |S_n^*|. \tag{2}$$

Considering the cost of a globally optimal plan P^*, we note that in every city transporting packages to the airport might be combined with the delivery of packages arrived by plane to their final destinations. Since every location in a city is mentioned at least once in some order (either as destination location or as source location) this number of (combined) move actions per city is $m+1$, so for all cities in total $(m+1) \times n$. The number of plane moves is $|S_n'^*|$, where the optimal plane plan has to satisfy additional constraints allowing for efficient pre- and post transportation. Thus, the total cost for the minimal plan $|P^*|$ is

$$c(P^*) = 6p + (m+1)n + |S_n'^*|, \tag{3}$$

where $|S_n'^*| \geq |\ S_n^*|$ since S_n^* is a globally optimal move plan for the plane. Hence,

$$\rho = \frac{c(P_{loc}^*)}{c(P^*)} = \frac{6p + \sum_{i=1}^{n}(m_{i1}+m_{i2}+1) + |S_n^*|}{6p + (m+1)n + |S_n'^*|}. \tag{4}$$

This ratio is maximal if $m_{i1} = m_{i2} = m$ for $i = 1, 2, \ldots, n$, implying that $p \geq m \times n$. Using the constraint $|S_n'^*| \geq |\ S_n^*| \geq n$, we derive

$$\rho = \frac{c(P_{loc}^*)}{c(P^*)} \leq \frac{6mn + 2mn + 2n}{6mn + mn + 2n} = \frac{8m+2}{7m+2} \leq \frac{8}{7} \approx 1.14\ . \tag{5}$$

Note that this price of autonomy is based on the assumption that the plane plan to be developed by the plane agent is optimal (i.e., it costs an optimal number of $|S_n^*|$ move actions). As we have remarked above, determining this optimal visiting sequence is an intractable problem. If we don't require this plane plan to be optimal, the total number of move actions can be at most twice the number of move actions in an optimal plan. Then, we can determine the effective price of autonomy ρ_{eff} as

$$\rho_{\mathit{eff}} = \frac{c(P_{\mathit{eff}})}{c(P^*)} \leq \frac{6p + 2mn + n + 2 \times |S_n^*|}{6p + mn + n + |S_n^*|} \leq \frac{8mn + 3n}{7mn + 2n} \leq \frac{11}{9} \approx 1.22\ . \tag{6}$$

Remark 2. To place this result into perspective, we should mention that, unless there is some major breakthrough in complexity theory,[7] we cannot hope for an ϵ-approximation algorithm that solves the minimum visiting sequence problem with $\epsilon < 2$. This implies that the best performance ratio of an approximation

[7] The breakthrough would be finding a constant-approximation algorithm for the Minimum Directed Feedback Vertex Set-problem.

algorithm for the logistic planning problem with $p = mn$ and $m = 1$ we can hope for is

$$\alpha = \frac{6mn + (m+1)n + 2 \times |S_n^*|}{6mn + (m+1)n + |S_n^*|} = \frac{7mn + 3n}{7mn + 2n} = \frac{7+3}{7+2} \geq \frac{10}{9} \approx 1.11 \quad (7)$$

If we compare this result with Equation 5 and $m = 1$ this directly implies that the best polynomial approximation algorithm would introduce almost the same overhead (worstcase) as locally optimal algorithms for solving the logistic problem with our coordination mechanism do.

5 Experimental Results

The results obtained in the previous sections are theoretical and worst-case results. We are guaranteed that the price of autonomy is never worse than about 1.22 implying that we do not increase the cost of our joint plan too much, compared to the cost of an optimal plan. Now, we will compare the performance of our coordination approach with planners that aim to solve the logistic problem centrally.

With this comparison we want to show two things: First, the average performance of our coordination approach is much better than what should be expected, if the worst-case performance ratio is taken as the norm. Second, the coordination approach can be used to enhance the planning power of existing planners significantly, thereby showing that it enables single-agent planning technology to be used in multi-agent problems.

In the Artificial Intelligence Planning and Scheduling (AIPS) competition of the year 2000, several general-purpose planning systems competed in a number of planning domains. The logistic planning problem as described in Section 4.1 was one of the domains featured. We have used the AIPS logistics dataset in our experiments because of its status as a benchmark problem set, and also because it allows us to compare our decomposition-by-coordination approach to a selection of centralised planning systems.

In Table 1, we compare plan cost (in terms of the number of moves in a plan) for four planners. In the second column, the costs of the optimal plans are given as calculated by encoding the complete instance as an ILP-problem and solving it exactly (of course not taking into account the time needed to find a solution). The third column represents the cost of the plans produced using the coordination approach. The fourth, fifth, and sixth columns represent a selection of the planning systems competing in the AIPS: The competition-winning TALplanner [14], and the above-average performers STAN [15] and HSP2 [16]. Each row in Table 1 represents an instance in the dataset, characterised by the number of packages that have to be transported for that instance. Of the roughly 200 instances in the dataset, we have made a random selection of 12.

It will come as no surprise that the results produced using the coordination approach, especially since the local plans were in fact solved exactly, deviate

Table 1. Results for 12 randomly chosen instances from the AIPS logistics dataset. For each instance the minimum number of moves is determined and for each planner the number of moves produced is given.

nr packages	min nr moves	Coordination	TALplanner	STAN	HSP2
20	107	113	111	110	145
25	143	150	152	149	206
30	175	182	183	177	250
35	177	181	186	182	264
40	228	239	239	232	337
45	269	284	285	276	–
50	286	299	306	293	–
55	319	327	338	326	–
60	369	391	398	376	–
65	371	387	397	382	–
70	405	426	437	416	–
75	438	458	471	448	–

little from the optimal plans (less than 5% on average). This is significantly better than the expected (25%) based on the worst-case performance ratio.

The plans produced by the coordination approach are comparable in quality with the plans produced by STAN (only 2% deviating from the optimum) and TALplanner (about 7%). To illustrate that for some solvers the problems in the logistics dataset are far from trivial, HSP2 does not manage to solve (within reasonable time and memory constraints) any instances where more than 40 packages have to be transported, and produces significantly worse plans (about 44% deviating from the optimum). The CPU-times needed to produce the plans by the coordination approach were a few seconds for each of the planning instances occurring in Table 1.

As we remarked before, the coordination approach cannot only be used to solve multi-agent planning problems using simpler single-agent planning tools. We can also apply it as a *pre-processing* step for a given planning system that has trouble solving such multi-agent planning problems. The idea is then to decompose the problem into smaller sub-problems that can be solved independently by the planning system. Thereafter, the solutions to the sub-problems can simply be combined into a solution to the whole problem instance.

Specifically, we propose to use the coordination approach to decompose a multi-agent logistics instance into a set of single-agent planning problems. Then, feed each of the single-agent planning sub-problems to the planning system, and combine the results (plans) according to the protocol into an overall plan, thereby solving the complete multi-agent instance. What we would like to see using this method is significant savings in computation time without significant loss in plan quality compared to the use of the planner solving the complete instance. For this experiment, we chose STAN (since it produced the best plans for the complete instance) and HSP2 (since it consumed a lot of CPU time).

To test these expectations, we randomly selected 51 problems from the logistics planning dataset and observed both the reduction in CPU-time and the reduction in plan cost, comparing a solution produced by using only the planner with a solution by using the planner in combination with the coordination approach. The results are given in Figure 8.

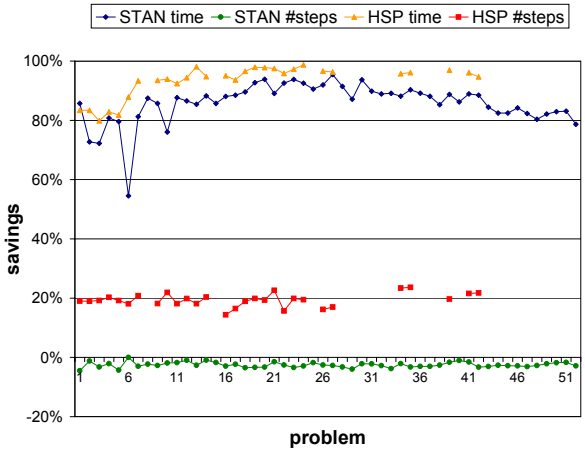

Fig. 8. Savings in CPU times and plan cost (#steps) when STAN and HSP2 make use of the coordination approach as a pre-processing step.

It can be observed that both STAN and HSP2 definitely benefit from preprocessing by the coordination approach: Both planning systems regularly achieve savings in computation time of over 80%. In addition, we can see that HSP2 produces plans that are on average 20% cheaper (i.e., requiring 20% less actions). Also note that the plan cost for STAN does not increase significantly when using the coordination approach. Finally, it can be observed that even after a decomposition into smaller sub-problems, for quite some instances HSP2 again was not able to produce a solution within reasonable time. This means that even for the local planning problems HSP2 still has considerable difficulty in solving them.

6 Discussion and Future Work

We discussed the plan-coordination problem for selfish agents. We showed that, although optimal plan-coordination mechanisms are hard to obtain in general, in some special cases a polynomial algorithm can be used to find minimal coordination sets. We also discussed the price of autonomy as a means to determine the performance loss one can expect when allowing a planning problem to be solved by autonomous agents.

In more recent work [17], we have extended this approach to tightly-coupled tasks with dependency and synchronisation constraints. There, we show that

plan-coordination mechanisms exist for such complex tasks, but that they require information exchange after planning to establish the exact time of scheduling synchronised tasks. This offers possibilities to extend the plan-coordination approach to the domain of temporal planning. We also showed that we can also provide coordination mechanisms for *durative tasks* with time constraints. Moreover, the techniques developed enable us to reduce the construction (and complexity) from previously-developed coordination mechanisms to the construction of coordination mechanisms for durative tasks with time constraints, implying that the latter are at least as hard to design as the former mechanisms.

A final extension we study are coordination mechanisms for temporal tasks where not only the feasibility of the total plan has to be guaranteed, but also completion time of the total task has to be minimised [18]. We show that, while the problem can be easily solved if the agents are able to execute an unbounded number of tasks concurrently, the problem to find suitable coordination algorithms in case the agents are autonomous and have bounded concurrency is difficult. In this latter context, we can determine the price of autonomy with respect to a coordination mechanism by taking the ratio of the completion time achieved by a coordination mechanism and the completion time achieved by a dictatorial optimal algorithm. Analogously, we can establish the price of coordination by taking into account the number of additional constraints enforced by the coordination mechanism. Taking both measures into account in determining the quality of coordination mechanisms enforces us to look at Pareto-efficient coordination mechanisms. In this way, we can choose the best alternative, given a tolerated performance loss and a desired level of autonomy.

References

1. Shehory, O., Kraus, S.: Methods for task allocation via agent coalition formation. Artificial Intelligence **101** (1998) 165–200
2. Zlot, R.M., Stentz, A.: Market-based multirobot coordination for complex tasks. International Journal of Robotics Research, Special Issue on the 4th Int. Conf. on Field and Service Robotics **25** (2006) 73–101
3. Christodoulou, G., Koutsoupias, E., Nanavati, A.: Coordination mechanisms. In Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D., eds.: Proc. of the 31st Int. Coll. on Automata, Languages and Programming (ICALP). Volume 3142 of Lecture Notes in Computer Science (LNCS)., Berlin, Germany, Springer (2004) 345–357
4. Smith, S.F., Gallagher, A., Zimmerman, T., Barbulescu, L., Rubinstein, Z.: Distributed management of flexible times schedules. In: Proc. of the 6th Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS). (2007) 472–479
5. Decker, K.S., Lesser, V.R.: Designing a family of coordination algorithms. In: Proc. of the 1st Int. Conf. on Multi-Agent Systems (ICMAS), San Francisco, CA, USA, AAAI Press, distributed by The MIT Press (1995)
6. Cox, J.S., Durfee, E.H.: Efficient mechanisms for multiagent plan merging. In: Proc. of the 3rd Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS). Volume 3., Washington, DC, USA, IEEE Computer Society (2004) 1342–1343

7. Sandhlom, T.W., Lesser, V.R.: Coalitions among computationally bounded agents. Artificial Intelligence **94** (1997) 99–137
8. Walsh, W.E., Wellman, M.P.: A market protocol for decentralized task allocation and scheduling with hierarchical dependencies. In: Proc. of the 3rd Int. Conf. on Multi-Agent Systems (ICMAS), IEEE Computer Society (1999) 325–332
9. Gerkey, B.P., Matarić, M.J.: A formal analysis and taxonomy of task allocation in multi-robot systems. Int. Journal of Robotics Research **23** (2004) 939–954
10. Dias, M.B., Stentz, A.: Traderbots: A market-based approach for resource, role, and task allocation in multirobot coordination. Technical Report CMU-RI-TR-03-19, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA (2003)
11. Buzing, P.C., ter Mors, A.W., Valk, J.M., Witteveen, C.: Coordinating self-interested planning agents. Autonomous Agents and Multi-Agent Systems **12** (2006) 199–218
12. Steenhuisen, J.R., Witteveen, C., ter Mors, A.W., Valk, J.M.: Framework and complexity results for coordinating non-cooperative planning agents. In Fischer, K., Timm, I.J., André, E., Zhong, N., eds.: Proc. of the 4th German Conf. on Multi-Agent System Technologies (MATES). Volume 4196 of Lecture Notes in Artificial Intelligence (LNAI)., Berlin, Germany, Springer (2006) 98–109
13. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, San Fransisco, CA, USA (1979)
14. Kvarnström, J., Doherty, P.: TALplanner: A temporal logic based forward chaining planner. Annals of Mathematics and Artificial Intelligence **30** (2000) 119–169
15. Long, D., Fox, M.: Efficient implementation of the plan graph in STAN. Journal of Artificial Intelligence Research **10** (1999) 87–115
16. Bonet, B., Geffner, H.: Heuristic search planner 2.0. AI Magazine **22** (2001) 77–80
17. Steenhuisen, J.R., Witteveen, C.: Plan coordination for durative tasks. In Salido, M.A., Garrido, A., Bartk, R., eds.: Proceedings of the Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS). (2007) 73–80
18. Chetan Yadati, Y.Z., Witteveen, C.: Performance of coordination mechanisms. In: to be published. (2008)

Appendix

A Proof of Proposition 1

Proof. Suppose, on the contrary, that the resulting complex task $\mathcal{T}'$ is not plan coordinated. Then there must exists some cycle $c = (t_{i_1}, t_{i_2}, \ldots, t_{i_m}, t_{i_1})$ where

1. c is not limited to tasks belonging to one agent, i.e., c contains at least two tasks t_{i_j} and t_{i_k} belonging to different agents.
2. c contains at least two tasks connected via a plan-refinement relation $\prec_i^*$. This means that c has a subsequence of tasks $(t_{i_j}, t_{i_{j+1}}, t_{i_{j+2}}, \ldots, t_{i_{j+k}}, t_{i_{j+k+1}})$, where $(t_{i_j}, t_{i_{j+1}}, t_{i_{j+2}}, \ldots, t_{i_{j+k}})$ is a $\prec^*$-sequence of tasks belonging to some agent A containing at least two different tasks, while the task $t_{i_{j+k+1}}$ belongs to another agent A'.

Since $t_{i_{j+k+1}}$ depends on $t_{i_{j+k}}$, it follows that

$$depth(t_{i_{j+k}}) < depth(t_{i_{j+k+1}}). \tag{8}$$

Due to the construction of the Δ_i-sets, for every subsequence $(t_{i_{j+h}}, t_{i_{j+h+1}})$ of an intra-agent path $(t_{i_j}, t_{i_{j+1}}, t_{i_{j+2}}, \ldots, t_{i_{j+k}})$ of agent A it must hold that

$$depth(t_{i_{j+h}}) \leq depth(t_{i_{j+h+1}}), \tag{9}$$

because $depth(t_{i_{j+h}}) > depth(t_{i_{j+h+1}})$ for two tasks belonging to the same agent would imply that $(t_{i_{j+h+1}}, t_{i_{j+h}}) \in \Delta_i$ and, therefore, $t_{i_{j+h}} \prec^* t_{i_{j+h+1}})$ cannot occur as part of A's plan, because it would create a cycle.

Equation 8 and 9 together imply that

1. traversing from one task t to another task t' via an inter-agent constraint in c strictly increases the depth: $depth(t) < depth(t')$, and
2. traversing from one task t to another task t' via an intra-agent constraint in c does not decrease the depth: $depth(t) \leq depth(t')$.

This implies that since there is at least one inter-agent constraint involved, the depth of the first task t_{i_1} occurring in c should be strictly less than the depth of the last task in c. But that implies $depth(t_{i_1}) < depth(t_{i_1})$: contradiction. Hence, such a cycle c cannot exist and the complex task $\mathcal{T}'$ is plan coordinated. □

B Responses to the reviewers

B.1 Reviewer 1

- We have corrected the mistake in the definition of the depth function: indeed the recursive part should read $depth(t) = 1 + max\{depth(t') : t' \prec t\}$.
- We also presented this definition in the main text as this reviewer suggested.
- We also corrected the reference to Equation 1.
- We corrected the depths of the tasks from 1 and 2 to 0 and 1 respectively in Figure 4 and the surrounding text.

Thanks for pointing out these mistakes!

B.2 Reviewer 2

- From a formal point of view we disagree with the points a) and b) raised by the reviewer.
  With respect to a): In the beginning of the paper we clearly mention that a complex task is a partially ordered set of (elementary) tasks. Now every structure that is (partially or totally) ordered is by definition an acyclic structure (since one of the specification axioms is irreflexitivity), so a total order containing a cycle is logically impossible.
  With respect to b): if $(T, <)$ is a partial order, it is immediate that there always exist a refinement of it that is acyclic: take a topological order of $<$. Also the counter example presented by the reviewer does not apply since it contains a cycle. This can never be an example of a complex task specification, see a).
  Nevertheless, we have added a few sentences to designing a coordination mechanism, stating that there always exists at least one solution to the plan coordination problem, hoping that this will take away the objections of the reviewer.
- The algorithm presented in in Sect. 3 has been provided with a title.
- We have corrected the depth definition and have placed it in the main text.

All minor remarks have been addressed. Thanks.

B.3 Reviewer 3

- We have adapted Section 2.1 according to the remarks
- We have added references to the results mentioned on page 8
- We have corrected the definition of depth(t)
- The proof of Proposition 1 has been moved to the Appendix.
- You are right in the remarks about the approximation ratio, we mixed up two different definitions. It has been corrected.
- On page 12, we have adapted the description of the algorithm to nearly optimal. Note that it pertains to the chains case.
- On page 17 : *do increase* has been changed in *do not increase*.

We have provided some additional references to recent work. Thank you for your comments.