Improving Task-Based Plan Coordination

Chetan Yadati Narasimha, Cees Witteveen, and Yingqian Zhang

Faculty EEMCS, Delft University of Technology, Mekelweg 4, 2628 CD, Delft, The Netherlands
{C.Yadati,C.Witteveen,Yingqian.Zhang}@tudelft.nl

Abstract. A multi-agent planning problem consists of a set of activities that need to be planned by several autonomous agents. Here, plan coordination methods play an important role, since independently generated plans by different agents can easily lead to an infeasible joint plan. We study a coordination-by-design approach which allows each agent to make its own plan completely independently of the others, while still guaranteeing the feasibility of the joint plan. The essence of this coordination approach is to determine a minimum number of additional constraints (a minimum coordination set) such that autonomously developed plans satisfying these constraints are always mergeable into an overall feasible plan. It has been shown that such coordination problems are very hard to solve. Therefore, approximation algorithms have been developed to compute a sufficient, but not necessarily minimum coordination set.

In this paper, we concentrate on a special class of multi-agent planning problems. These problems arise in several practical applications such as supply chain management and hospital patient treatment. The plan coordination instances in these applications turn out to have a special structure. Using so-called agent dependency graphs, we show that for this special class of problems a better approximation algorithm to compute a sufficient coordination set can be obtained.

1 Introduction

Multi-agent planning requires a collection of agents to solve a joint problem together. Typically, each of the agents is capable of solving only a (disjoint) part of the problem and would like to construct and apply its solution (or plan) *independently* of the others. Due to possible dependencies between the solutions provided, it is not guaranteed that partial solutions always can be merged into a feasible overall solution. Therefore, we need to provide a *coordination mechanism*.

Coordination in multi-agent planning can be viewed as an attempt to achieve a conflict-free joint plan given a set of self-interested multi-agent planners. In the multi-agent system community, quite some effort has been done to attack this problem [1]. The *plan merging* approach [2,3,4], for example, focuses on techniques that resolve conflicts *after* independently generated plans have been developed by different agents. Here, agents are required to exchange, negotiate, and revise their (partial) individual plans to arrive at a joint solution. Another approach treats coordination and planning as *intertwined* processes [5,6,7], where agents continuously exchange information during their planning in order to achieve a conflict-free solution. In these two coordination approaches, agents have to be cooperative in the sense that they should be willing to exchange their private information with others, and to revise their plans if necessary.

One particular approach has addressed the problem from the *coordination-by-design* perspective, that is, how to provide just sufficient coordination constraints on agents *before* the agents start to solve their own sub-problem, such that *(i)* a joint solution is always coordinated, yet *(ii)* each agent can construct its plan independently from others. One advantage of such a pre-planning coordination method is that it can be applied without relying on the assumption of cooperative agents.

General results obtained in this approach can be summarized as follows [8,9,10]: First of all, given a coordination instance, checking whether or not agents can plan independently without coordinating them is a coNP-complete problem. Secondly, finding a *minimum* number of constraints such that coordination is achieved is a computationally even more demanding problem (Σ_2^p-complete) [9]. Later, in [10], the authors presented a polynomial approximation algorithm, called the Depth-Partitioning Algorithm, which can be used to obtain a sufficient (but not necessarily minimum) set of coordination constraints. However, it only has been shown that this algorithm provides rather good results for logistic planning problems (as used in the AIPS-planning competition). Mors [8] identified two special cases where the coordination problem becomes computationally easier: the case where an agent has at most *one* task, and the case where all tasks with incoming or outgoing inter-agent arcs are *totally ordered* amongst themselves. However, these two cases are very restricted and hardly represent any real-world problems.

Built upon previous work on coordination-by-design approach, our main contributions in this paper are as follows:

Firstly, we identify two interesting multi-agent planning problems *supply chain management* and *hospital patient treatment* and we demonstrate that instances of these planning problems can be identified as coordination instances with a particular structure, called *intra-free* coordination instances.

Secondly, we show (in Section 3) that besides its practical interest, for a intra-free coordination instance it can be easily decided whether it is coordinated or not, using a so-called *agent dependency graph*. Consequently, the coordination checking problem for this class of instances becomes feasible. We also show that finding a minimum coordination set is NP-hard.

Thirdly, we propose a new polynomial approximation algorithm (in Section 4) for obtaining sufficient, but not necessarily minimum coordination sets for the intra-free instances. This approximation algorithm is developed based on a so-called agent dependency graph. We also demonstrate that the proposed algorithm is superior to the existing depth-partitioning algorithm that has been used to provide an approximation solution to the general coordination problem.

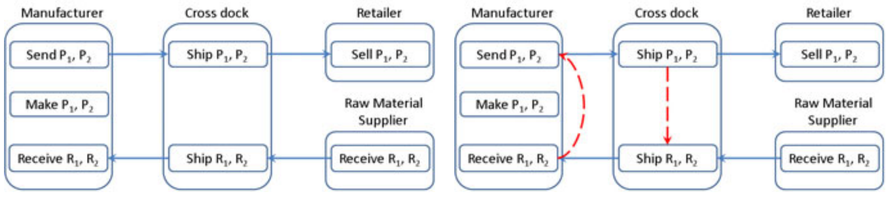
2 Two Multi-agent Planning Problems

To motivate the subject of our paper, in this section we discuss supply chain management problem and patient planning in hospitals as two particular multi-agent planning problems.

Plan coordination in supply chain management

In general, supply chain management is the management of material and information flows both in and between enterprises, such as vendors, manufacturers, and distribution centres in a network [11]. Figure 1(a). depicts a simple supply chain network, where four different enterprises are involved: a product manufacturer, a cross dock, a raw material supplier and a retailer. The cross dock is an enterprise which does not produce anything but is simply involved in distribution of products and raw materials. Each enterprise has to perform some specific tasks (sending, shipping, or selling goods). The flow of goods between the enterprises induces precedence constraints on the tasks to be performed. For example, goods cannot be sold before they are shipped to the retailer. These dependencies are indicated by a directed arrow between tasks. Each enterprise wants to make a plan for its set of tasks. Such a plan for the manufacturer, for example, might be to wait first for the receipt of R_1 and R_2 before sending P_1 and P_2. In general such a plan is a partially ordered set of precedence constraints, respecting the internal (intra-agent) dependency constraints. Note that here the supply chain coordination instance has a special property: the set of intra-agent constraints is empty.

Enterprises in this system might be unwilling to communicate with their partners about their own plans for many reasons like privacy, etc. Therefore, coordination in a supply chain network focusses on coordinating the activities and plans of the individual enterprises involved in the supply chain[1]. Now if the cross dock plans to send raw materials R_1, R_2 after sending products P_1, P_2 and the manufacturer decides to send P_1, P_2 after receiving R_1, R_2, then we have an infeasible joint plan as shown in Figure 1(b): it contains a dependency cycle. The coordination problem thus requires us to find a minimal set of additional constraints such that whatever specific plan (sequence of tasks) is chosen by an individual agent, their joint plan is always feasible.



(a) Example logistics scenario. (b) Plan deadlock in supply chains.

Fig. 1. Plan coordination in supply chain management

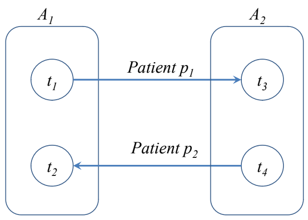
Plan coordination in Hospital patient treatment

Treatment procedures for patients in a hospital involve different sets/sequences of resource usage for each patient. This leads to significant amounts of time spent in setting up equipment/resources. Whenever set-up times are considerable, departments also

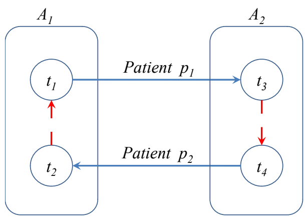
[1] Note here that we focus only on the topological task structure and ignore all other factors such as costs, time points and inventory levels.

need to carefully plan the order in which they treat their patients. Owing to their treatment sequences patients might need to get serviced by different departments in some prescribed order. For example, the anaesthetics department is required to anaesthetize the patient before any major surgery is performed. In such cases, there exists a precedence relationship between the activities of the departments involved. Since each of the specialists involved might belong to a different department, the effectiveness of patient management depends heavily on the 'health' of the coordination between departments.

Typically, because each department has to customize its equipment/resources for a patient, each department develops a plan to customize its equipment in a particular sequence (based on prior information about the patient). Because each department does its planning independently of others, there is a possibility that department plans conflict. This is particularly serious if patients have to follow a strict sequence of treatment procedures from different departments. Departments handle several patients from several treatment procedures and hence it gets extremely complicated for them to communicate with other departments while they are planning. As an example of this problem, consider two patients p_1 and p_2 who need to visit departments A_1 and A_2 as illustrated in Figure 2(a). The treatment sequence for patient p_1 might be stipulated as "visit department A_1 (task t_1) and then visit department A_2 (task t_3)". Similarly the treatment sequence for patient p_2 could be to "first visit department A_2 (task t_4) and then visit department A_1 (task t_2)". Now if department A_1 chooses to treat p_2 first and A_2 chooses to treat patient p_1 first, we clearly have a deadlock. If the treatment procedure in both departments requires a significant amount of set up time, then each department could lose a lot of time in undoing an earlier set up and setting up a new environment for a different patient.



(a) Treatment sequences for two patients p_1 and p_2.



(b) A_1 plans to treat patient p_2 first and A_2 plans to treat patient p_1 first.

Fig. 2. A patient planning example

We observe that in the applications presented above, several commonalities exist:

- both applications are characterised by the presence of autonomous agents;
- agents in both applications have tasks that are dependent upon tasks allocated to other agents;
- there exist no (intra-agent) dependency constraints between tasks allocated to a single agent. As we will see, this feature greatly reduces the computational complexity

of some aspects of coordination and enables us to come up with better approximation algorithms.

Before presenting our main results, in the next section, we will first introduce a formal model of the multi-agent coordination problem.

3 Multi-agent System Model

In modelling the hospital scenario as well as the supply chain management scenario as a multi-agent system, we assume that there is a set $A = \{A_1, \ldots, A_m\}$ agents (enterprises, departments) who need to decide on (partially) ordering a set $T = \{t_1, \ldots t_n\}$ of n tasks given to them. Each task t_i (a treatment, service or diagnosis) has to be accomplished by a unique agent A_j. There might exist dependency relations between tasks: if task t_i has to be performed before task t_j, this is indicated by a precedence relation $t_i \prec t_j$ between them. We assume that there exist no dependencies between tasks belonging to the same agent.

The precedence relation induces a partial order on the set T. We represent this partial order by an (acyclic) *task graph* $G_T = (T, \prec)$ where T is represented as a set of nodes and the precedence relationships between them as a set of directed arcs.

Since we assume that tasks (services, treatments) are uniquely coupled to agents, we can partition the set of concrete tasks T into subsets T_i, each of which represents the set of tasks that can be performed by a single agent A_i. Hence, the agents induce a partitioning $\{T_i\}_{i=1}^n$ of the set of tasks T. The (distributed) task graph can also be represented as the tuple $G_T = \langle \{T_i\}_{i=1}^n, \prec \rangle$.

Each agent A_i, is given a set of tasks T_i together with a set $\prec_i$ of partial order constraints induced by $\prec$ and has to come up with a *local plan* to complete its set of tasks T_i, while not violating the set of constraints $\prec_i$. We need not know what kind of planning software is used by the agents, neither how the details of their resulting plan do look like. The only information that is needed is the partial ordering that is induced on the set T_i as a result of this plan. Hence, (the consequences of) each such local plan of an agent A_i can be represented as a partial order $\langle T_i, \prec_i^* \rangle$, where $\prec_i^*$ is a *refinement* of $\prec_i$, i.e., the relation $\prec_i^*$ is a partial order relation that extends $\prec_i$.

Although each of these agents might have a perfect local plan $\langle T_i, \prec_i^* \rangle$, their combination might not be a valid plan since some *inter-agent* constraints might be violated.

Example 1. Referring to the situation in Figure 2(b), both agents A_1 and A_2 have perfectly valid local plans, where each plan satisfies the (empty set of) local constraints. But as demonstrated in the example, in combination these plans might suffer a deadlock.

Given the local plans of the agents, a *joint plan* for G_T is a structure $\langle \{T_i\}_{i=1}^n, \prec \cup \prec_1^* \cup \ldots \cup \prec_n^* \rangle$. Such a plan is called *infeasible* if the relation $\prec^* = \prec \cup \prec_1^* \cup \ldots \cup \prec_n^*$ is not acyclic. A cycle in a joint plan would indicate that the plan cannot be executed: completion of task t would need the completion of another task t' which in turn would require the completion of t.

To prevent such infeasible joint plans, in the coordination by design approach, a set of additional precedence constraints C, called the *coordination set*, is added to the planning instance. Such a set guarantees that, by respecting all the local constraints and the coordination constraints while preparing their local plans, agents can be sure that joint plans always will be feasible. These constraints in C are always intra-agent constraints and restrict agents from making plans that could lead to cycles in the joint plan.

Although quite some work already has been done on the analysis and construction of coordination sets, in this paper we will deal with the problem of providing suitable coordination sets in special cases like patient planning and supply-chain management. These cases are special in the sense that before coordination sets are provided, planning agents are completely free in planning the activities they have to complete. We call such planning instances *intra-free*:

Definition 1. *A coordination instance $G_T = \langle \{T_i\}_{i=1}^n, \prec \rangle$ is* intra-free *if, for every T_i, the subgraph $G_{T_i} = \langle T_i, \prec_i \rangle$ of G_T is the empty graph $\langle T_i, \emptyset \rangle$ on T_i, that is, there are no precedence relations between tasks assigned to the same agent.*

If an instance is intra-free, then every task t is either a completely isolated node in the task graph or is connected to a task t' assigned to another agent. We call a task t a *source task* if its in-degree is 0, i.e., $in(t) = 0$. A task t is a *sink task* if its out-degree is 0, i.e., $out(t) = 0$. If for all i and each task $t \in T_i$, it holds that t is either a sink or a source task, then such an instance is called *strictly intra-free*.[2]

Given an arbitrary intra-free instance, as we will show, there is a simple procedure to transform it into an equivalent strictly intra-free structure. Hence, we will assume that plan coordination instances are always strictly intra-free. The significance of intra-free coordination instances is that they alleviate the computational complexity of the plan coordination problem and allow for the investigation of new algorithms for finding coordination sets. First, we will show that coordination-checking, which in general is known to be coNP-complete [9], is solvable in polynomial time for intra-free coordination instances. Here, the coordination checking problem is to decide whether a given coordination instance is already coordinated, i.e., there is no need to provide a non-empty coordination set to ensure that a joint plan is always feasible whenever the individual agent plans are feasible.

3.1 Coordination Checking for Intra-free Coordination Instances

The technique used to prove that the coordination checking problem can be solved efficiently, is to use a so-called *agent dependency graph*. Essentially, this agent dependency graph is the contraction of the distributed task graph $(\{T_i\}_{i=1}^n, \prec)$ to a graph consisting only of the agents A_i and their dependencies:

Definition 2. *Let $G_T = \langle \{T_i\}_{i=1}^n, \prec \rangle$ be a distributed task graph and $A = \{A_i\}_{i=1}^n$ the associated set of agents. The agent dependency graph derived from G_T is a graph $G_A = (V_A, E_A)$, where $V_A = \{v_{A_i} : A_i \in A\}$ is the set of nodes corresponding to agents, and $E_A = \{(v_{A_i}, v_{A_j}) : \exists t \in T_i, \exists t' \in T_j, [t \prec t']\}$ the dependency relation between them.*

[2] There might be tasks t such that $in(t) = out(t) = 0$. Without loss of generality, however, such tasks can be completely neglected for coordination purposes.

For general coordination instances, we can use the acyclicity of the agent dependency graph to infer that the instance is already coordinated,

Proposition 1. *Let $G_T = \langle \{T_i\}_{i=1}^n, \prec \rangle$ be a coordination instance and G_A its associated agent dependency graph. Then the acyclicity of G_A implies that G_T is coordinated, i.e., G_T is a yes-instance of the coordination checking problem.*

Proof. If G_A is acyclic, the only cycles that could occur in any extension of G_T are cycles *within* a task set T_i of an agent A_i. This is excluded, since each individual extension $(T_i, \prec_i^*)$ of $(T_i, \prec_i)$ has to be a plan, i.e., an acyclic refinement of $\prec_i$. □

In general, the converse of this proposition is not true: even if G_A is cyclic, we might have a yes-instance of G_T. For a simple example, take Figure 2(a) and suppose that there is an intra-agent constraint in A_1, i.e., $\prec_{T_1} = \{t_1 \prec t_2\}$. Then the instance is coordinated, but the agent dependency graph, having two nodes A_1 and A_2 has a cycle. If, however, $G_T = \langle \{T_i\}_{i=1}^n, \prec \rangle$ is a strictly intra-free coordination instance, we can actually show that the converse holds, too:

Theorem 1. *Let $G_T = \langle \{T_i\}_{i=1}^n, \prec \rangle$ be a strictly intra-free coordination instance and G_A its agent dependency graph. Then G_A is acyclic iff G_T is a yes-instance of the coordination checking problem.*

Proof. Given Proposition 1, it suffices to show that G_T is not coordinated if G_A contains a cycle. So let G_A contain a simple cycle $(A_{i_1}, A_{i_2}, \ldots, A_{i_k}, A_{i_1})$. Since the coordination instance is strictly intra-agent free, there must exist a sequence of tasks $(t_{i_1,2}, t_{i_2,1}, t_{i_2,2}, \ldots, t_{i_k,1}, t_{i_k,2}, t_{i_1,1})$ with two tasks per agent, such that for $j = 1, \ldots, k$, we have $t_{i_j,1}, t_{i_j,2} \in T_{i_j}$, $t_{i_j,2} \prec t_{i_{j+1},1}$ and $t_{i_k,2} \prec t_{i_1,1}$. But then it immediately follows that each empty relation in $G_{T_j} = \langle V_{T_j}, \emptyset \rangle$ has a simple *acyclic extension* $\{t_{i_j,1} \prec_{i_j}^* t_{i_j,2}\}$ such that the graph $G_T^* = \langle V_T, \prec \cup \prec^* \rangle$, where $\prec^*$ is the union of these extensions $\prec_{i_j}^*$, contains a cycle. Hence, G_T is not coordinated. □

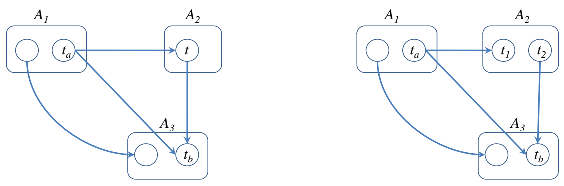
Note that these results hold for *strictly* intra-free coordination instances. It is, however, very easy to generalize them to intra-free coordination instances by polynomially reducing intra-free instances to strictly intra-free instances: we apply the following *task-splitting* procedure to tasks that violate the strict intra-free property.

Given an arbitrary intra-free coordination instance $G_T = \langle \{T_i\}_{i=1}^n, \prec \rangle$, for every i and every $t \in T_i$ such that $in(t), out(t) > 0$, do the following:

1. split t into two tasks t_1 and t_2
2. for all $t' \prec t$, add a precedence constraint $t' \prec t_1$, and for all $t \prec t''$, add a constraint $t_2 \prec t''$;
3. remove t and all precedence constraints it is mentioned in.

Clearly, the result is an equivalent, strictly intra-free coordination instance.

Example 2. As a simple example, Figure 3 illustrates how to convert an intra-free instance to a strictly intra-free instance. The task t of agent A_2 is split into two tasks t_1 and t_2. We can simply verify that when the (converted) strictly intra-free instance is coordinated, the (original) intra-free instance must be also coordinated. □



(a) An intra-free instance.

(b) Resulting *strictly* intra-free instance.

Fig. 3. An example on converting an intra-free instance to a strictly intra free instance

As a result, we have the following Proposition:

Proposition 2. *An intra-free coordination instance G_T is coordinated whenever its strictly intra-free variant $G_T{}'$ is coordinated. Therefore, the coordination checking problem for intra-free coordination instances can be decided in polynomial time.*

4 An Approximation Algorithm for Finding Coordination Sets

Given the fact that coordination checking for intra-free instances is easy, one might hope that finding a *minimum coordination set* is also tractable in the case of intra-free coordination instances. This turns out to be wrong. Finding such a minimum set of coordination constraints has been shown to be NP-hard:

Proposition 3. *The NP-complete* Directed Feedback Vertex Set *problem*[3] *is polynomially reducible to the decision version of the minimum coordination problem for intra-free coordination instances.*

Proof. (Sketch) Let $(G = (V,E),K)$ be an instance of the directed feedback vertex set (FVS) problem [12]. We obtain an instance of the intra-free coordination problem by first duplicating tasks: $T = V \cup \{v' : v \in V\}$, for every $v \in V$ creating an agent A_v having the tasks v and v' to complete, and adding constraints $v_i' \prec v_j$, whenever $\{v_i, v_j\} \in E$. It is not difficult to see that the resulting instance $\langle T, \prec \rangle$ is an intra-free coordination instance, since for tasks v' we have $in(v) = 0$ and for tasks v we have $out(v) = 0$.

We easily observe that G has a feedback vertex set of size K whenever K coordination arcs are needed to ensure that $\langle T, \prec \rangle$ is coordinated: For, let $W \subseteq V$ be a feedback vertex set of G of size K. Construct the coordination set $\Delta = \{v' \prec v) : v \in W\}$. Adding these constraints makes it impossible to use both v and v' in any cycle of an extension of G_T. Hence, it has exactly the same effect as removing the node A_w from the agent dependency graph. Since W is a feedback vertex set, this is precisely the set of agent nodes that render the agent dependency graph acyclic. □

[3] The Directed Feedback Vertex Set problem is, given a directed graph $G = (V,E)$ and a $K \in Z^+$, to decide whether there exists a subset of at most K nodes from V whose removal will render the remaining graph G acyclic.

Although Propostion 3 clearly indicates that finding the least number of constraints required to coordinate a given coordination instance is NP-hard, we might apply approximation algorithms to find an approximate solution to this problem. The Depth Partitioning (DP) algorithm by Steenhuisen et al. [10] is such an algorithm that finds a sufficient, but not necessarily minimum, number of coordination constraints required to coordinate a given instance.

This algorithm works as follows:

1. Take the partially ordered set $(T, \prec)$ of all tasks and determine the depth $depth(t)$ of each task $t \in T$ with respect to the precedence relation $\prec$. This depth is defined in the standard way as follows:

$$depth(t) = \begin{cases} 0 & \nexists t' \in T[t' \prec t] \\ 1 + max\{depth(t') : t' \prec t\} & \text{else} \end{cases}$$

2. For each task set T_i and each pair of tasks $t, t' \in T_i$, we add a coordination constraint $t \prec t'$ whenever $depth(t) < depth(t')$.

It is not difficult to show that the coordination set thus obtained is always sufficient to guarantee that the planning instance obtained is coordinated [10].

4.1 Finding a Better Algorithm for Intra-free Instances

The drawback, however, of the DP algorithm is that the algorithm adds a constraint whenever there are two tasks of different depths, without paying attention to whether such an addition is strictly necessary or not. We propose a more frugal way of applying depth partitioning while ensuring coordination: Instead of using the depth partitioning principle for every agent we will provide a filter to select between those agents where the depth partitioning needs to be applied and those where we do not need to apply it.

The basic idea is to use the agent dependency graph as such a filter. Whenever the agent dependency graph is acyclic, we do not need to add any coordination constraints. Whenever the agent dependency graph contains a cycle, we know, since the coordination instance is intra-free, that at least some coordination arcs have to be added. In the latter case, we can use an approximation of the minimum feedback vertex set to select an agent A_i occurring in the feedback set. We then apply the depth-partitioning algorithm to the task set T_i associated with this agent A_i. In general this will remove some of the cycles in the agent dependency graph.

In order to make the instance intra-free again, we split the task set T_i into the k depth levels of the tasks induced by applying the depth partitioning algorithm. These task sets $T_{i,1}, T_{i,2}, \ldots, T_{i,k}$ then together with the other task sets constitute an intra-free coordination instance again and can be represented by k agents $A_{i,1}, \ldots A_{i,k}$ in the resulting agent dependency graph. In order to ensure that these agents will not be chosen again in a feedback vertex set, we include them in a blackout list of vertices and test the acyclicity of the dependency graph again. Technically then, we need to use an approximation algorithm for the *(Blackout) Feedback Vertex Set* to indicate which agents create the cycle.

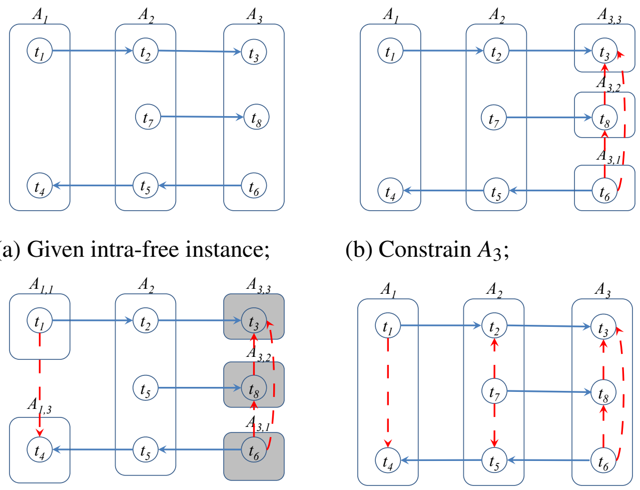
A Blackout Feedback Vertex Set problem is an extension of the general Feedback Vertex Set (FVS) problem. In the FVS problem, one is given a directed graph and is asked to find a subset of vertices that intersect every directed cycle in the graph such that cardinality of this subset is minimum. The blackout version also has an additional special subset of vertices B which cannot be part of the feedback vertex set. An approximation algorithm for this problem has been proposed by Even et al. [13].

Note that task sets T_i consisting of tasks all having the same depths will never be affected by the DP algorithm. In order to avoid that an agent A_i associated with such a task set will be included in a feedback vertex set, we include such agents in the blackout set in the beginning. This implies that the algorithm can only choose agents whose task sets consist of tasks that can be affected by the DP algorithm. The resulting algorithm can be stated as in Algorithm 1.

Algorithm 1. Advanced Depth Partitioning Algorithm (DP*)

```
Require: An intra free coordination instance $G_T = \langle \{T_i\}_{i=1}^n, \prec \rangle$; $A = \{A_1, A_2, \ldots, A_n\}$
Ensure: A set of coordination constraints $C$ added to $G_T$ to make it coordinated
 1: compute $depth(t)$ for every $t \in T$
 2: let $G_A = \langle V_A, E_A \rangle$ be the dependency graph associated with $G_T$
 3: $B = \{A_i$ : all tasks $t \in T_i$ have the same depth $\}$
 4: $C = \emptyset$
 5: while $G_A$ contains a cycle do
 6:    $F = BlackoutFVS(G_A, B)$
 7:    select $A_i \in F$
 8:    for every pair of tasks $t, t' \in T_i$ do
 9:       if $depth(t) < depth(t')$ then
10:          add $t \prec t'$ to $C$
11:       end if
12:       Split $T_i$ in $k$ subsets $T_{i,j}$ having the same depth
13:       $T = (T - T_i) \cup \{T_{i,1}, \ldots T_{i,k}\}$
14:       let $G_A$ be the new dependency graph associated with $G_T$
15:       add the nodes $A_{i,j}$ corresponding to the sets $T_{i,j}$ to $B$
16:    end for
17: end while
18: return $C$
```

Example 3. Consider the situation in Figure 4. It has three agents A_1, A_2, A_3 needing to perform the tasks $t_1, \ldots, t_8$. In the first step the blackout set B is empty and the feedback vertex set algorithm is free to choose any agent into its feedback vertex set. Suppose it picks agents A_1, A_3, i.e., $F = \{A_1, A_3\}$. In agent A_3 we first add coordination constraints $t_6 \prec t_3$, $t_6 \prec t_8$, $t_8 \prec t_3$, and then split the agent into three agents $A_{3,1}, A_{3,2}, A_{3,3}$. We then add these agents into B, i.e., $B = \{A_{3,1}, A_{3,2}, A_{3,3}\}$ and check if the agent dependency graph has a cycle. It turns out that the agent dependency graph still has a cycle and the blackout FVS algorithm returns $F = \{A_1\}$. So we now constrain A_1 by adding constraint $t_1 \prec t_4$. The agent dependency graph now becomes acyclic. Thus, the procedure returns four constraints $t_6 \prec t_3$, $t_6 \prec t_8$, $t_8 \prec t_3$, and $t_1 \prec t_4$. Notice here that DP algorithm would



(a) Given intra-free instance;

(b) Constrain A_3;

(c) Compute blackout set and constrain A_1.

(d) Coordination set derived from the DP algorithm.

Fig. 4. An example of how the advanced depth partitioning algorithm works

add constraints between all tasks that have different depth (among tasks of each agent), resulting in two more constraints $t_7 \prec t_2$ and $t_7 \prec t_5$ as shown in Figure 4 (d).

We now show the Advanced Depth Partitioning algorithm DP* is correct.

Proposition 4. *Let $G_T = (\{T_i\}_{i=1}^n, \prec)$ be a plan coordination instance and C the set of additional precedence constraints returned by the Advanced Depth Partitioning algorithm given in Algorithm 1. Then the instance $G'_T = (T, \prec \cup C)$ is plan coordinated.*

Proof. (Sketch) The correctness of this algorithm is almost immediate from the correctness of the DP-algorithm. In the DP*-algorithm, every time a different subset of agents is affected, the number of agents who can be constrained reduces because once an agent T_i has been constrained each of the $T_{i,j}$ agents that result by splitting it enter the Blackout set (all tasks within $T_{i,j}$ have the same depth). The DP*-algorithm will continue with adding constraints to agents until the associated agent dependency graph is acyclic. In the worst case all agents that could be affected by the depth-partitioning algorithm are affected. Note that the DP*-algorithm does not choose agents that will never be affected by the DP-algorithm to add constrains. Hence by the correctness of the DP-algorithm, the resulting instance must be coordinated. □

5 Discussion and Conclusions

This paper identified a new subclass of coordination problems called *intra-free* coordination instances which have a relevance to patient planning problems and supply chain planning problems. We have shown that the decision problem of whether or not such intra-free instances are coordinated can be solved efficiently, although the problem to find a minimum coordination set still remains NP-hard. Based on the efficient coordination checking method, we proposed a specialized approximation technique to design a 'good' coordination set efficiently.

As part of our future research, first of all we are investigating better heuristics for producing subset minimal coordination sets and develop the theoretical performance comparison analysis. We also seek to identify other and broader classes of plan coordination problems that can be efficiently solved. Hospitals typically need to handle a significant number of emergencies and patients without prior appointments. Online mechanisms are popular models for managing such scenarios. It would be interesting to extend the DP*-algorithm to handle online scenarios and compare its performance with respect to other online coordinating mechanisms.

References

1. Durfee, E.H.: Distributed problem solving and planning. In: Weiß, G. (ed.) Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, pp. 121–164. MIT Press, Cambridge (1999)
2. Foulser, D.E., Li, M., Yang, Q.: Theory and algorithms for plan merging. Artificial Intelligence 57(2-3), 143–181 (1992)
3. von Martial, F.: Coordinating Plans of Autonomous Agents. LNCS, vol. 610. Springer, Heidelberg (1992)
4. Tonino, H., Bos, A., de Weerdt, M., Witteveen, C.: Plan coordination by revision in collective agent based systems. Artificial Intelligence 142, 121–145 (2002)
5. Durfee, E.H., Lesser, V.R.: Partial global planning: A coordination framework for distributed hypothesis formation. IEEE Transactions on Systems, Man, and Cybernetics 21(5), 1167–1183 (1991)
6. Ephrati, E., Rosenschein, J.S.: Multi-agent planning as the process of merging distributed sub-plans. In: Proceedings of DAI 1993, pp. 115–129 (May 1993)
7. Lesser, V., Decker, K., Wagner, T., Carver, N., Garvey, A., Horling, B., Neiman, D., Podorozhny, R., NagendraPrasad, M., Raja, A., Vincent, R., Xuan, P., Zhang, X.: Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework. Journal of AAMAS 9(1), 87–143 (2004)
8. Mors, A.T.: Coordinating autonomous planning agents. Master's thesis, TU Delft, Delft, The Netherlands (April 2004)
9. Valk, J.: Coordination among Autonomous Planners. PhD thesis, Department of Software technology, EWI, TUDelft (2005)
10. Steenhuisen, J.R., Witteveen, C., Zhang, Y.: Plan-coordination mechanisms and the price of autonomy. In: Sadri, F., Satoh, K. (eds.) CLIMA VIII 2007. LNCS (LNAI), vol. 5056, pp. 1–21. Springer, Heidelberg (2008)
11. Kazemi, A., Zarandi, M.F., Husseini, S.M.: A multi-agent system to solve the productiondistribution planning problem for a supply chain: a genetic algorithm approach. International Journal of Advanced Manufacturing Technilogy 44, 180–193 (2009)
12. Festa, P., Pardalos, P., Resende, M.: Feedback set problems. In: Du, D.Z., Pardalos, P.M. (eds.) Handbook of Combinatorial Optimization, pp. 209–259. Kluwer Academic Publishers, Dordrecht (1999)
13. Even, G., Naor, J.S., Schieber, B., Sudan, M.: Approximating minimum feedback sets and multi-cuts in directed graphs. In: Proceedings of the 4th IIPCO Conference, London, UK, pp. 14–28. Springer, Heidelberg (1998)