Delft University of Technology
Parallel and Distributed Systems Report
Series

Complexity of Coordinating Autonomous Planning Agents

Adriaan ter Mors, Jeroen Valk and Cees Witteveen
{a.w.termors, j.m.valk, c.witteveen}@ewi.tudelft.nl

report number PDS-2004-002

PDS

ISSN 1387-2109

Published and produced by:
Parallel and Distributed Systems Section
Faculty of Information Technology and Systems Department of Technical Mathematics and Informatics
Delft University of Technology
Zuidplantsoen 4
2628 BZ Delft
The Netherlands

Information about Parallel and Distributed Systems Report Series:
reports@pds.twi.tudelft.nl

Information about Parallel and Distributed Systems Section:
http://pds.twi.tudelft.nl/

Abstract

We consider the problem of coordinating autonomous agents that have to perform a joint task. This joint task consists of a set of *elementary tasks*, partially ordered by a set of precedence constraints. Each agent is assigned (using some given task allocation protocol) a subset of the available tasks. We assume that agents wish to be fully independent during the planning process, yet they are dependent on each other because of the precedences between tasks allocated to different agents.

In this paper we concentrate on the following coordination problem: how to offer complete autonomy for the participating agents in planning their part of the tasks, while at the same time guaranteeing that the individually constructed feasible plans (whatever they may be) are fully respected in constructing the joint plan. As we will see, this problem comes down to finding a minimal set of additional constraints (a coordination set) such that after adding these constraints a combined plan for executing all tasks can be achieved by simply joining whatever plans have been developed by the individual agents. We show that this problem can be decomposed into two subproblems: *verifying* that a given set of additional constraints is a coordination set, i.e., allows agents to plan independently (the coordination verification problem), and finding out whether a coordination set is of minimal size (the minimal coordination problem).

We will show that the coordination verification problem alone is co-NP-complete. In the general case, if the only dependencies are precedence constraints between tasks, the coordination problem is Σ_2^p-complete; if we also allow simultaneity constraints between tasks (i.e., constraints specifying that tasks may not be performed simultaneously), then the coordination problem is Π_3^p-complete.

Even rather simple cases of the coordination problem turn out to be intractable and we show that it is very unlikely that constant-ratio approximation algorithms for this problem exist even if each agent has only a trivial planning task to perform.

Chapter 1

Introduction

In this paper we study the computational complexity of coordinating a set of autonomous planning agents. The multi-agent planning setting we choose is deceptively simple: We assume that agents have work together on some joint task T that consists of a number of elementary tasks t_j. Furthermore, these elementary tasks are allocated to agents using some unspecified task assignment protocol such as e.g. discussed by Shehory and Kraus [5]. As in their framework, elementary tasks assigned to (possibly different) agents may be interrelated by a set of precedence constraints $t_i \prec t_j$, requiring that t_i has to be completed before t_j can start. Unlike in their framework, however, *(i)* execution of the subset T_i of tasks assigned to agent A_i requires careful planning of agent A_i and *(ii)* we assume that agents are *autonomous planning* agents, that is, these agents do not want to be interfered *during* planning by other agents, neither do they want to revise their plans *after* planning if their plans turn out to be incompatible with the plans of other agents.

The precedence constraints between the tasks induce dependencies between the agents: if a task t, allocated to agent A_j, is preceded by a task t' from a different agent A_i, then A_j is dependent on A_i. Clearly, to manage these dependencies between agents, some form of coordination is required. We are therefore faced with the following *coordination problem*: how to guarantee that, irrespective of the plans developed by these autonomous planning agents, these plans can be combined into a feasible joint-agent plan without the need to revise them?

Of course, plan coordination in multi-agent systems is not an entirely new topic. In principle, following the literature we could try to solve this problem in several ways: *(i)* Restrict the planning autonomy of the agents by commu-

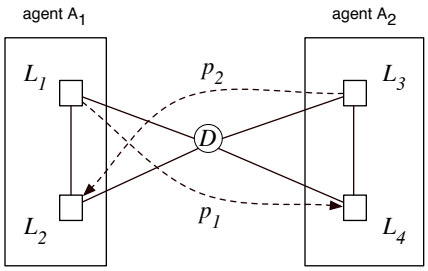

Figure 1.1: Parcels must be delivered between locations; a package can be transferred from one truck to another at the depot.

nicating and negotiating about the inter-agent constraints during planning, (cf. [2, 1]); *(ii)* Perform independent planning and integrate the possibly incompatible plans (by applying replanning, if necessary). This approach suggests approaching the coordination problem by studying and exploiting (positive and negative) relationships between plans like [9]; Finally, we could opt for *(iii)* finding a minimal set of additional problem constraints such that adding them ensures complete planning autonomy by the agents and guarantees that the result of simply combining the individual plans always results in a feasible joint plan.

Obviously, the first two approaches mentioned above are less appropriate in applications where the requirement of planning autonomy is strict: competitive relations between the agents, for example, may simply prohibit agents to reveal details of their plans to other agents. Therefore, in this paper we focus on the third approach, where in the pre-planning phase, we try to find a set of minimal constraints that guarantees that a simple joint plan can be found that respects the individually developed plans.

Note that this approach, by separating coordination concerns from planning, also offers possibilities for upgrading current stand-alone planning tools: by separating coordination from autonomous single-agent planning processes, single-agent planning systems can be re-used for multi-agent planning problems.

To present a simple example to illustrate these ideas, we use a simple logistic chain as a guiding example.

Example 1. *We have a multi-agent planning problem where parcels have to be transported between locations: a package* p_1 *must be transported from*

location L_1 to L_4 and a package p_2 that must be transported from L_3 to L_2 (see Figure 1.1). We have two transportation agents: agent A_1 handles locations L_1, L_2 and the depot D, while agent A_2 serves locations L_3, L_4 and D. Both agents have to start and finish in the depot D. Transportation of package p_1 requires agent A_1 to pick up the package at L_1, but as location L_4 is out of A_1's region, he will only transport p_1 as far as the depot D. From there, agent A_2 will drive the package to L_4. Similarly, agent A_2 will bring p_2 to the depot, but A_1 must perform the final trip to L_2.

Now agent A_1 has to make a plan for carrying out the tasks $t_1 = (L_1, D)$ and $t_2 = (D, L_2)$, while A_2 has to make a plan for $t_3 = (L_3, D)$ and $t_4 = (D, L_4)$. These tasks are interrelated: t_1 has to be completed before t_4 can start and likewise t_3 has to precede t_2. Now each agent has to solve a route-planning problem[1], but is dependent upon the plan of the other agent. If we allow both agents to plan independently, the two plans can easily become incompatible: For example, if agent A_1 would aim for the plan (visiting sequence) $D - L_2 - L_1 - D$ to achieve his tasks t_2 and t_1, and A_2 would aim for the visiting sequence $D - L_4 - L_3 - D$, these plans cannot be combined in a multi-agent plan achieving T: Clearly, to start its plan, agent A_1 has to wait in D until t_3 has been accomplished by agent A_2, but agent A_2 cannot complete this task before t_1 has been completed by A_1. Hence, trying to execute the agent plans would result in a deadlock.

In the above example, it is not hard to see how the agents can be coordinated: either agent A_1 must perform its pre-depot task t_1 before its post-depot task t_2, or agent A_2 must perform its pre-depot task before starting on its post-depot task. If, however, we scale up the problem to include an arbitrary number of tasks, agents, and precedence constraints, then we are faced with a very hard problem. In fact, in Section 3, we will prove that the coordination(-solution) verification problem, which is to verify whether a given set of additional restrictions allows agents to plan independently, is co-NP-complete. This proof consists of reducing the NP-complete Path With Forbidden Pairs problem to the complement of the coordination verification problem.

In Section 4 we prove that the coordination problem itself, which is to find a minimum set of constraints to allow independent planning, is Σ_2^p-complete. In this case, the proof consists of reducing a quantified version of the path

[1]It is not difficult to see that even if all distances between locations are the same, finding a shortest route plan is NP-hard.

with forbidden pairs problem, which is Σ_2^p-complete, to the coordination problem. This reduction can be extended to show that, if we also allow simultaneity constraints, then the coordination problem is Π_3^p-complete.

Not all coordination instances are equally hard, however, as there exist subcases that are 'only' NP-complete. In Section 6, we will show that if a coordination instance lacks certain structural characteristics, then it is in NP.

In Section 7 we investigate the approximability of the coordination problem. Even for the case where each agent is restricted to having at most two tasks to perform, the coordination problem is both NP-hard and hard to approximate, in the sense that (most likely) no constant-ratio approximations exist. We can prove this by reducing the Feedback Vertex Set (FVS) problem to this restricted version of the coordination problem, as FVS is NP-hard, and no constant-ratio approximations have been found, despite considerable research effort.

Chapter 2

Problem Statement

A typical instance of the coordination problem consists of a partition $\mathbf{T} = \{T_1, \ldots, T_n\}$ of a set of elementary tasks $T = T_1 \cup \cdots \cup T_n$ (representing a distribution of T over n agents $A_1, \ldots, A_n$), and a set E of *precedence constraints* specifying a partial order on T.[1] Usually, we will specify such an instance as a tuple $(\mathbf{T}, G = (T, E))$ where $\mathbf{T}$ is the partitioning scheme and $G = (T, E)$ is a directed acyclic graph (dag). The subgraph $G_i = (T_i, E_i)$ is the subgraph of G generated by T_i and representing the set of tasks and the set of *intra-agent* constraints for agent A_i. We define E_i as:

$$E_i = E^+ \cap (T_i \times T_i)$$

where E^+ is the transitive closure of E. The set of *inter-agent* constraints E_{inter} consists of all precedence constraints $(t, t') \in E$, where $t \in T_i$, $t' \in T_j$ and $i \neq j$.

An elementary task can be executed by a single agent. Executing its set T_i of elementary tasks requires the agent A_i to make a plan. In Example 1, such a plan consists of the specification of a visiting-sequence route e.g. $D - L_2 - L_1 - D$ for executing his tasks. Such a *concrete plan*, however, will contain much more information about e.g. resources used by the plan than is needed for coordination. Since we assume that agents are only dependent upon each other through the precedence constraints between tasks, we are only interested in the order in which an agent A_i plans to perform its set T_i of tasks. Therefore, we consider only an *abstract plan* for agent A_i, derived from its concrete plan, that specifies the order in which the tasks T_i are planned.

[1] More precisely, E is the transitive reduction of the precedence relation on T.

It is obvious that such an abstract plan should *(i)* satisfy the intra-agent constraints E_i imposed on T_i and *(ii)* should specify a partial order on T_i. Therefore, each such an abstract plan can be specified as an acyclic graph $G_i^P = (T_i, E_i \cup \hat{E}_i)$, where $E_i \cup \hat{E}_i$ extends the partial order specified by E_i.

Example 2. *Continuing Example 1, let the concrete plans developed by the agents A_1 and A_2 be given as the visiting sequences $D - L_2 - L_1 - D$ and $D - L_4 - L_3 - D$ respectively. The abstract plans corresponding to these concrete plans are $G_1^P = (\{t_1, t_2\}, \{(t_2, t_1)\})$ and $G_2^P = (\{t_3, t_4\}, \{(t_4, t_3)\})$, respectively. Note that since $G_1 = (\{t_1, t_2\}, \emptyset)$ and $G_2 = (\{t_3, t_4\}, \emptyset)$, these plans satisfy the intra-agent constraints.*

It is easy to see that by developing (abstract) plans autonomously, the feasibility of the combination of these plans cannot be guaranteed: as can be seen from the example, although the plans may be individually feasible, their combination results in an infeasible plan since the original set E of precedence constraints together with the added constraints $\hat{E} = \hat{E}_1 \cup \hat{E}_2$ is no longer acyclic.

Therefore, the coordination problem now can be stated as follows: how to guarantee that, whatever constraints $\hat{E}_i$ that are added by individual agents A_i in making their individually feasible plans, the resulting combined precedence relation $E \cup \hat{E}$ is still acyclic, where $\hat{E} = \hat{E}_1 \cup \cdots \cup \hat{E}_n$.

As it turns out, a simpler problem is to check whether such a coordination problem will occur in a coordination instance $(\mathbf{T}, G)$. This problem we call the *coordination verification* problem:

Definition 2.1 (Coordination Verification Problem). *The coordination verification problem (CVP) is: given a coordination instance $(\mathbf{T}, G)$ with $G = (T, E)$ a dag and $\mathbf{T} = \{T_1, \ldots, T_n\}$ a partition of T, does it hold that, for all sets of refinements $\hat{E} \subseteq T \times T$ satisfying:*

1. *$\hat{E} = \hat{E}_1 \cup \cdots \cup \hat{E}_n$ where $\hat{E}_i \subseteq T_i \times T_i$ for $i = 1, \ldots, n$, and*
2. *for each i, the graph $(T_i, E_i \cup \hat{E}_i)$ is acyclic,*

the graph $(T, E \cup \hat{E})$ is acyclic?

Note that in a yes-instance of CVP, it holds that whatever locally acyclic plans (partial orders) the agents A_i come up with, the result can be combined into a feasible joint plan.

If we are confronted with a no-instance $(\mathbf{T}, (T, E))$ of the CVP problem, it is easy to see that the only way to ensure the planning autonomy of the participating agents is to come up with a set Δ of *additional constraints* on the set of tasks T such that $(\mathbf{T}, (T, E \cup \Delta))$ is a yes-instance of the CVP-problem. The coordination problem (CP), then, is the problem to find such a set Δ of minimum cardinality. The decision variant of this problem can be specified as follows:

Definition 2.2 (Coordination Problem). *The coordination problem (CP) is: given a coordination instance $(\mathbf{T}, G)$ find a set of precedence constraints $\Delta = \Delta_1 \cup \cdots \cup \Delta_n$ with $\Delta_i \subseteq T_i \times T_i$ such that:*

1. *$E \cup \Delta$ is acyclic,*
2. *$(T, E \cup \Delta)$ is a yes-instance of CVP, and*
3. *$|\Delta|$ is minimal.*

2.1 Planning Arcs

Before we start on the analysis of the complexity of the coordination problem, we will first introduce the set of *planning arcs* that represent the freedom agents have during planning. In particular, we need to define the set of all possible (allowed) precedence constraints that can be added during planning, that are also relevant for coordination; we will denote this set of planning arcs by $\hat{E}_{tot}$.

First, note that planning arcs must be contained within agents, i.e., $\hat{E}_{tot} \subseteq \bigcup_{i=1}^{n} T_i \times T_i$. Second, we require agents to add only planning arcs that do not directly lead to a local cycle. The addition of an arc e leads to a local cycle in case e^{-1} is an existing precedence constraint. Hence, $\hat{E}_{tot} \cap [E^{-1}]^+ = \emptyset$. Also, of course, if an arc e is already in E, then we do not consider it a planning arc either.

Finally, an arc is only relevant for coordination in case it can contribute to an inter-agent cycle.[2] Note that any inter-agent cycle must 'enter' an agent at some task and 'exit' an agent through another task. The former task is in

[2]An inter-agent cycle C is a cycle in $E \cup \hat{E}_{tot}$ that intersects the set of inter-agent precedence constraints, i.e., $C \cap E_{inter} \neq \emptyset$.

$T^{in} = \mathrm{dom}(E_{inter})$, the latter is in $T^{out} = \mathrm{ran}(E_{inter})$. Thus, we define $\hat{E}_{tot}$ as follows:

$$\hat{E}_{tot} = \bigcup_{i=1}^{n} T_i^{in} \times T_i^{out} \setminus \left(E^+ \cup [E^{-1}]^+\right)$$

The following proposition justifies restricting our attention to arcs in $\hat{E}_{tot}$.

Proposition 2.3. *If there exists a set $\hat{E} = \{\hat{E}_1, \ldots, \hat{E}_n\}$ such that*

1. *$\hat{E} = \hat{E}_1 \cup \cdots \cup \hat{E}_n$ where $\hat{E}_i \subseteq T_i \times T_i$ for $i = 1, \ldots, n$,*
2. *for each i, the graph $(T_i, E_i \cup \hat{E}_i)$ is acyclic, and*
3. *the graph $(T, E \cup \hat{E})$ contains a cycle C,*

then there exists a set $\hat{E}' \subseteq \hat{E}_{tot}$ satisfying the same conditions.

Proof. Let $C = e_1 - \cdots - e_m$; note that in C, (sequences of) arcs in E_{inter} alternate with (sequences of) arcs in $E_i \cup \hat{E}_i$. We will now show how to form the set $\hat{E}'$ by replacing all arcs in $\hat{E}_i$ with arcs in $\hat{E}_{tot}$ — if necessary.

Let $e_j - \cdots - e_k$ be a sequence in $E_i \cup \hat{E}_i$ such that e_{j-1} and e_{k+1} are in E_{inter}. We distinguish two cases:

case 1: $(\mathrm{dom}(e_j), \mathrm{ran}(e_k)) \in E_i$. Clearly, $(\mathrm{dom}(e_j), \mathrm{ran}(e_k)) \in E^+$; we can choose $\hat{E}'_i = \emptyset$, as C is enabled by arcs in E.

case 2: $(\mathrm{dom}(e_j), \mathrm{ran}(e_k)) \notin E_i$; we can choose $\hat{E}'_i = \{\hat{e}_i = (\mathrm{dom}(e_j), \mathrm{ran}(e_k))\}$. Note that $\hat{e}_i \in \hat{E}_{tot}$:

1. $(\mathrm{dom}(e_j), \mathrm{ran}(e_k)) \in T^{in} \times T^{out}$;
2. $(\mathrm{dom}(e_j), \mathrm{ran}(e_k)) \notin E^+$;
3. $(\mathrm{dom}(e_j), \mathrm{ran}(e_k)) \notin [E^{-1}]^+$, since otherwise $E_i \cup \hat{E}_i$ would have been cyclic.

□

The set $E \cup \hat{E}_{tot}$ represents the precedence relation E, augmented with the set of arcs $\hat{E}_{tot}$ that represent the planning freedom of the agents. We will call the associated directed graph $G = (T, E \cup \hat{E}_{tot})$ the *coordination graph*.

Chapter 3

Complexity of Coordination Verification

The coordination-verification problem is to verify whether there *cannot* exist a set of planning arcs that create an inter-agent cycle, while remaining locally acyclic. It is easy to see that CVP is in co-NP.

Proposition 3.1. *CVP is in co-NP.*

Proof. A no-certificate consists of a set of planning arcs that creates an inter-agent cycle. More specifically, a no-certificate is a set $\hat{E}$ satisfying properties *(i)* and *(ii)* of Definition 2.1 that creates a cycle in the graph $(T, E \cup \hat{E})$. Clearly, both properties and the cyclicity of $(T, E \cup \hat{E})$ can be verified in polynomial time. □

To prove that this problem is co-NP-complete, we will present a reduction from the path with forbidden pairs problem.

Definition 3.2. *The Path With Forbidden Pairs problem (PWFP) is: given a tuple* (G_0, C, s, t) *where* $G_0 = (V, E_0)$ *a directed acyclic graph,* $C = \{c_1, c_2, \ldots, c_n\}$ *a set of pairs of arcs in* E_0*, and two distinct nodes* s *and* t *in* V*, does there exist a path from* s *to* t *using at most one arc from every* $c_j \in C$*.*

Specifically, we will reduce PWFP to the complement of the coordination-verification problem. The complement of CVP, which we will call the Coordination-Failure Detection problem (CFD) asks whether there *does* exist a set $\hat{E}$ satisfying both conditions of Definition 2.1, yet creating an inter-agent cycle in the graph $(T, E \cup \hat{E})$.

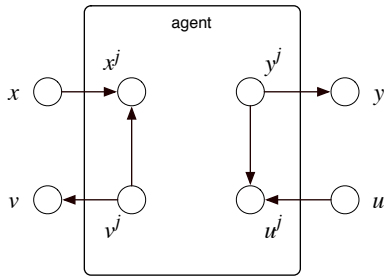

Figure 3.1: The subgraph constructed for a forbidden pair $\{(x, y), (u, v)\} \in C$.

To reduce PWFP to CFD, we need to transform the graph $G_0 = (V, E_0)$ (the PWFP instance) to a tuple $(\mathbf{T}, G = (T, E))$. Intuitively, this transformation consists of two parts: an interesting part and a not-so-interesting part. The not-so-interesting part entails *(i)* creating a single-task agent for every vertex in V, and *(ii)* replicating those arcs that are not involved in any forbidden pair into the CFD-graph.

The interesting part regards the transformation of forbidden pairs. We encode each forbidden pair as a so-called *path-blocking gadget*, depicted in Figure 3.1. For a certain forbidden pair $c_j = \{(x, y), (u, v)\}$ in the PWFP problem, we may choose either, but not both of these arcs for creating a path from s to t. Similarly, in creating an inter-agent cycle in CFD, we add either planning arc (x^j, y^j), or (u^j, v^j), but not both. This mutual exlusion is enforced by the arcs in (v^j, x^j) and (y^j, u^j): addition of e.g. planning arc (x^j, y^j) creates a path $v^j - x^j - y^j - u^j$ in E_i. Subsequently, (u^j, v^j) may no longer be added, as it would create a local cycle.

To complete the transformation, we need to connect t to s. In this way, we can equate the existence of an $s - t$ path in PWFP to the existence of an inter-agent cycle in CFD. However, we do not directly connect t to s, to avoid instances with a trivial $s - t$ path (using no arcs from forbidden pairs) to result in a cyclic coordination instance. Instead, we place an agent between s and t, that can connect t to s by adding a single planning arc. An example of a transformation from PWFP to CFD is given in Figure 3.2.

Formally, the reduction of PWFP to CFD is defined as follows:

1. For $i = 1, \ldots n$: $T_i = \{v_i\}$; for $j = 1, \ldots, k$ (with $k = |C|$): $T_{n+j} = \{x^j, y^j, u^j, v^j \mid \{(x, y), (u, v)\} \in C\}$ and $T_{n+k+1} = \{s_0, t_0\}$, where both

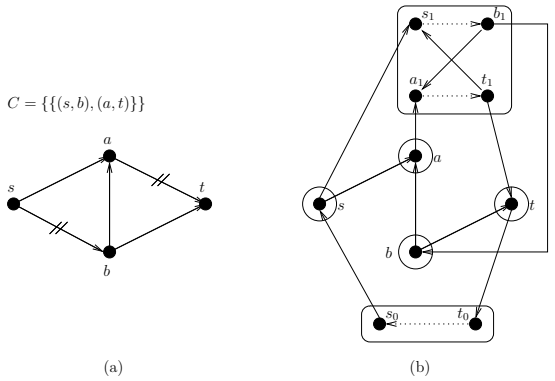

Figure 3.2: (a): A PWFP instance with a single forbidden pair and (b): the corresponding coordination instance, in which we have depicted the set $\hat{E}_{tot}$ of planning arcs by dotted arrows.

s_0 and t_0 do not occur in V. Obviously, $T = \bigcup_{i=1}^{n+k+1} T_i$.

2. E is the smallest set of arcs satisfying the following conditions:
   (a) For every arc $e = \{u, v\} \in E_0$ not occurring in a pair of arcs in C, e occurs in E.
   (b) For every constraint-pair of arcs $c_j = \{(x, y), (u, v)\} \in C$, E contains the following arcs
   $$(x, x^j), (y^j, y), (u, u^j), (v^j, v), (y^j, u^j), (v^j, x^j)$$
   (See Figure 3.1 for an illustration).
   (c) Finally, E contains the arcs (t, t_0) and (s_0, s).

Proposition 3.3. *PWFP reduces to CFD.*

To prove the correctness of the above reduction, we need to show three things:

1. The transformation results in a correct instance of the CFD problem; this means that $G = (T, E)$ must be a *dag*.
2. A yes-instance of the PWFP problem maps to a yes-instance of the CFD problem.

3. A yes-instance of the CFD problem maps to a yes-instance of the PWFP problem.

Proof. The graph $G = (\{T_i\}_{i=1}^{i=n+k+1}, E)$ is acyclic: the subgraph of G restricted to nodes mapped from vertices in V is clearly acyclic. The additional nodes $\{x^j, u^j\}_{j=1}^{k}$ and the node t_0 are clearly endpoint of paths; the additional nodes $\{y^j, v^j\}_{j=1}^{k}$ and the node s_0 are clearly starting points of paths. Hence, the additional nodes cannot contribute to a cycle.

A yes-certificate for a PWFP instance is given by a set of arcs $E'_0 = \{e_1, \ldots, e_m\}$ such that each e_j corresponds to one arc from the constraint-pair $c_j \in C$. Clearly, E'_0 contains at most one arc from every pair in C. We map the set E'_0 to a yes-certificate $\hat{E}$ for CFD in the following manner: if (x, y) from c_j in E'_0, then $(x^j, y^j) \in \hat{E}$. We complete the set $\hat{E}$ by adding arc (t_0, s_0). Clearly, $\hat{E}$ creates an inter-agent cycle in G: $\hat{E}$ creates a path from s to t, and a path from t to s, so we have a cycle.

For CFD, a yes-certificate $\hat{E}$ creates an inter-agent cycle in $(T, E \cup \hat{E})$. It is easily verified that any inter-agent cycle must include the arc (t_0, s_0) from agent A_{n+k+1}. Thus, $\hat{E}' = \hat{E} \setminus \{(t_0, s_0)\}$ creates a path from s to t in the CFD-graph. The set $\hat{E}'$ can only map to a yes-certificate for PWFP. First, note that only path-blocking-gadget agents are capable of adding planning arcs (apart from agent A_{n+k+1}, of course). Second, each such an agent can add at most one planning arc: either (x^j, y^j), or (u^j, v^j) (these are the only planning arcs for one agent). Adding both would result in a cyclic local plan, which is not allowed. Thus, $\hat{E}'$ maps to a yes-certificate for the PWFP instance, since an $s-t$ path can be created using at most one arc from every forbidden path. □

Corollary 3.4. *The Coordination Verification Problem is co-NP-complete*

Proof. Because of Proposition 3.1, CVP is in co-NP; because of the fact that its complement CFD is NP-hard, according to Proposition 3.3, CVP is co-NP-complete. □

Chapter 4

Complexity of the Coordination Problem

A solution Δ of a CP instance $(\mathbf{T}, G)$ is a cardinal-minimal set Δ of additional arcs that is sufficient to guarantee feasibility of the joint plan given arbitrary, individually feasible plans. Given a coordination instance I and an integer $K \geq 0$, the coordination problem asks for the existence of a solution Δ of size at most K.[1] In this section, we will show that for arbitrary values of $K > 0$, we are faced with a Σ_2^p-complete problem. To prove this, we need to introduce the following quantified version of the PWFP problem.

Definition 4.1 ($\exists\forall\neg$PWFP). *Given a* PWFP *instance* $(G_0 = (V, E_0), C, s, t)$*, and a partitioning* $\{C_1, C_2\}$ *of* C*, the* $\exists\forall\neg$PWFP *problem is to find an* exclusive choice *from* C_1*, i.e., a set* X_1 *that contains exactly one arc from every pair of forbidden pairs in* C_1*, such that for every exclusive choice* X_2 *from* C_2*, there does* not *exist a path from* s *to* t *in the set of arcs* $E_0' = (E_0 \setminus C) \cup X_1 \cup X_2$.[2]

We use the complement of the PWFP problem, in order to be able to relate a coordination set — ensuring that no inter-agent cycles can exist — to a solution for $\exists\forall\neg$PWFP that ensures that no $s - t$ path can be formed.

Showing that $\exists\forall\neg$PWFP is Σ_2^p-complete is straightforward using a reduction from the quantified satisfiability problem QSAT_2 that slightly adapts a

[1] Note that in case $K = 0$, this problem equals the CVP problem.

[2] Here, $E_0 \setminus C$ is a shorthand for the set of arcs from E_0 that do not occur in any forbidden pair.

standard reduction from 3-SAT to PWFP (cf. [6]).

Proposition 4.2. *The coordination problem is in* Σ_2^p.

Proof. To see that the coordination problem is in Σ_2^p, take a coordination instance $(\mathbf{T}, G)$ and a $K > 0$. Nondeterministically, guess a set of arcs $\Delta = \Delta_1 \cup \cdots \cup \Delta_n$ and verify whether: *(i)* $|\Delta| \leq K$, *(ii)* each $(T_i, E_i \cup \Delta_i)$ is acyclic, and *(iii)* $(\mathbf{T}, (T, E \cup \Delta))$ is a yes-instance of the CVP problem. The first two verifications can be done in polynomial time, while the last verification requires the consultation of an NP-oracle. Hence, the problem is in Σ_2^p. □

To prove that coordination is also Σ_2^p-hard, we will reduce $\exists\forall\neg$PWFP to the coordination problem. Intuitively, the reduction consists of three parts, two of which are interesting and one that is not. The non-interesting part is the same as the trivial part for the previous reduction, i.e., it consists of replicating vertices and non-interesting arcs. The first interesting part is also the same as the interesting part of the previous reduction: for every forbidden pair in C_2, we create the path-blocking gadget of Figure 3.1. The second interesting part is the transformation of forbidden pairs in C_1. For this, we introduce the so-called *forced-choice gadget*, which extends the path-blocking gadget.

The idea behind the reduction is to link the finding of a coordination set to the finding of an exclusive choice for C_1. That is, if we have found a coordination set — which means that whatever planning arcs the agents might add, no inter-agent cycle can be created — then we should automatically have an exclusive choice for C_1 such that no path from s to t can be found — whatever exclusive choices we come up with for C_2.

The forced-choice gadget (Figure 4.1) forces a coordination set to be constructed by adding only constraints within the forced-choice-gadget agents ('C_1-agents'). This is accomplished by introducing two potential paths from s to t through each C_1-agent: if the agent adds either planning arc (a^j, b^j) or planning arc (c^j, d^j), a path from s to t is created, and, since we connect t to s, an inter-agent cycle is created. The forced-choice gadget allows both these potential paths to be broken by adding a single constraint: if either (x^j, y^j) or (u^j, v^j) is added, then neither (a^j, b^j) nor (c^j, d^j) can be added without creating a local cycle.

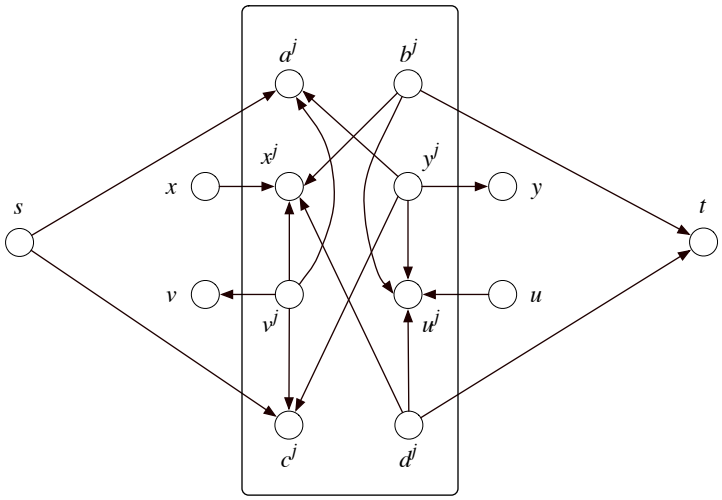

Figure 4.1: The subgraph containing a forced-choice gadget, constructed for a forbidden pair $c_j = \{(x^j, y^j), (u^j, v^j)\} \in C_1$.

Finally, we have to slightly alter the agent connecting t to s for this reduction. Since all inter-agent cycles pass through the agent connecting t to s, a coordination set can be found simply by 'blocking' this agent. Therefore, we ensure that this agent has $K + 1$ planning arcs to choose from in connecting t to s; breaking them all would require $K + 1$ constraints, while we are only allowed to find a coordination set of size K.

Specifically, the reduction is specified as follows:

1. For every $v_i \in V$, $T_i = \{v_i\}$. For every pair $(\{x, y\}, \{u, v\})$ occurring in C, T_{n+j} contains the additional nodes x^j, y^j, u^j, v^j. Moreover, for every pair $(\{x, y\}, \{u, v\})$ occurring in C_1, T_{n+j} contains four additional nodes (tasks): a^j, b^j, c^j and d^j. Finally, T_{n+m+1} $(m = |C|)$ contains the node s_0, and the $K + 1$ $(K = |C_1|)$ nodes $t_0, \ldots, t_k$.

2. The set of arcs E contains the following elements:

   (a) For every arc $e = \{u, v\} \in E_0$ not occurring in a pair of arcs in C, e occurs in E;

   (b) For every constraint-pair of arcs $c_j = \{(x, y), (u, v)\} \in C$, E con-

tains the following arcs

$$(x, x^j), (y^j, y), (u, u^j), (v^j, v), (y^j, u^j), (v^j, x^j)$$

(c) For every pair of arcs $\{(x, y), (u, v)\} \in C_1$, E contains the following additional arcs:

$$(v^j, a^j), (v^j, c^j), (y^j, a^j), (y^j, c^j), (b^j, x^j), (b^j, u^j)$$
$$(d^j, x^j), (d^j, u^j), (s, a^j), (s, c^j), (b^j, t), (d^j, t)$$

(d) Finally, E contains the arcs $\{(t, t_0), \ldots, (t, t_k)\}$ and (s_0, s).

Proposition 4.3. $\exists\forall\neg$PWFP *reduces to CP.*

Again, we need to prove:

1. The transformation results in a correct instance of the CP problem; this means that $G = (T, E)$ must be a *dag*.

2. A yes-instance of the $\exists\forall\neg$PWFP problem maps to a yes-instance of the CP problem.

3. A yes-instance of the CP problem maps to a yes-instance of the $\exists\forall\neg$PWFP problem.

Proof. To understand that the transformation results in an acyclic graph, note that all additional nodes in the CP instance either have only incoming arcs, or only outgoing arcs. Hence, these new nodes cannot introduce a cycle.

A yes-certificate for $\exists\forall\neg$PWFP consists of an exclusive choice X_1 for C_1. We can directly map X_1 to a yes-certificate Δ for CP: $(x, y) \in X_1 \rightarrow (x^j, y^j) \in \Delta$. To verify that Δ coordinates the CP instance, first note that any possible inter-agent cycle must include a path from s to t, because of the way the transformation works. Second, note that after adding Δ to the CP-instance, the following agents still have the capability to add planning arcs: agent A_{n+m+1} ($m = |C|$) can add the arcs $\{(t_0, s_0), \ldots, (t_k, s_0)\}$ (each of which effectively creates a path from t to s), and the agents corresponding to a C_2 constraint pair can add exactly one planning arc, from a choice of two: agent A_{n+j} can either add (x^j, y^j), or (u^j, v^j).

Clearly, the set of possible combinations of planning arcs for the C_2-agents corresponds to the set of all possible exclusive choices X_2 for C_2. Consequently, since in the $\exists\forall\neg$PWFP instance there does not exist an exlusive choice X_2 that creates a path from s to t (given X_1), there does not exist a set of planning arcs in the CP instance connecting s to t. Hence, Δ is a coordination set.

We must show that *any* coordination set Δ directly maps to an exclusive choice X_1 for C_1. There are three types of agents that are capable of adding constraints to Δ:

1. The agent A_{n+m+1}: every potential inter-agent cycle must pass through this agent, since it connects t to s. Hence, we can coordinate the instance by 'blocking' this agent. However, that would take $K+1$ constraints, since agent A_{n+m+1} can add $K+1$ planning arcs that would connect s to t.

2. The C_1-agents: the forced-choice gadget enables two direct paths from s to t, either by adding planning arc (a^j, b^j) or by adding (c^j, d^j). Adding either (x^j, y^j) or (u^j, v^j) blocks both these direct paths.

3. The C_2-agents; clearly though, the C_2-agents cannot break the direct paths from s to t through the C_1-agents.

Hence, a coordination set must consist of arcs from C_1-agents, exactly one per agent, since we have K C_1-agents, and we are allowed to use K constraints.

It is easy to see that a coordination set indeed maps to an exclusive choice X_1 solving the PWFP instance: note that in the coordinated CP instance, agent A_{n+m+1} is unconstrained, allowing it to add planning arcs connecting t to s. If other agents (i.e, C_2-agents, since these are the only other agents capable of adding planning arcs) would be able to add planning arcs connecting s to t, then it would be possible to create an inter-agent cycle. This is not possible, however, since Δ is a coordination set. Hence, no path can be created from s to t, and consequently, X_1 is a yes-certificate for the $\exists\forall\neg$PWFP instance. □

Chapter 5

Coordination with Simultaneity Constraints

So far in our analysis of the pre-planning coordination problem, we have considered only *precedence* constraints between tasks. Precedence constraints naturally lend themselves to modeling situations where execution of one task is a pre-requisite for execution of another task. Another common type of dependency is that of resource contention between tasks, e.g. if two tasks require the same tool, space, or other resource to be performed. In our task-oriented framework, we can model resource conflicts using *simultaneity constraints*.[1]

The complexity of the coordination problem increases from Σ_2^p-complete to Π_3^p-complete if we allow *simultaneity constraints* in addition to precedence constraints. There exists a simultaneity constraint between two tasks t and t', denoted by $t \bowtie t'$, if either $t \prec t'$ or $t' \prec t$ must hold. We can now specify a coordination instance as a tuple $(T, \prec, \bowtie)$, where T and $\prec$ have their usual meaning, and $\bowtie \subseteq (T \times T)$ is a set of simultaneity constraints. We say that a set $\prec^{\bowtie}$ *minimally satisfies* $\bowtie$ if (i) for each constraint $t \bowtie t'$ in $\bowtie$, exactly one of the two precedences $(t \prec t')$ and $(t' \prec t)$ is added to $\prec^{\bowtie}$, and (ii) no other precedence constraints are added to (the transitive reduction of) $\prec^{\bowtie}$.

[1] More specifically, we can model contention over a *non-consumable* resource between two tasks; simultaneity constraints do not allow modeling of contention over consumable resources such as money.

We now formulate the following coordination problem:

Definition 5.1 (Coordination Problem (∀-version)). *Given a coordination instance* $(\mathbf{T}, \prec, \bowtie)$ *and an integer* $K \in Z^+$*, the* $\forall$*CP problem is: Does it hold for all instances* $(\mathbf{T}, \prec', \emptyset)$ *where* $\prec' = \prec \cup \prec^{\bowtie}$ *is a partial order and* $\prec^{\bowtie}$ *minimally satisfies* $\bowtie$*, that there exist a set* $\Delta = \bigcup_{i=1}^{n} \Delta_i$ *of precedence constraints with* $\Delta_i \subseteq T_i \times T_i$ *such that:*

1. $|\Delta| \leq K$*,*
2. *each* $(T_i, \prec'_i \cup \Delta_i)$ *is acyclic, and*
3. *for all sets* $\hat{\prec'} = \bigcup_{i=1}^{n} \hat{\prec'}_i$*: if* $(T_i, \prec'_i \cup \Delta_i \cup \hat{\prec'}_i)$ *is acyclic and satisfies* $(T_i, \prec_i, \bowtie_i)$ *then* $(T, \prec' \cup \Delta \cup \hat{\prec'})$ *is acyclic.*

This definition of the coordination problem asks for a minimal set of additional precedence constraints that ensures that agents can plan independently of each other, *irrespective* of how the set of simultaneity constraints is satisfied (i.e., for every $(t, t') \in \bowtie$, either $t \prec t'$, or $t' \prec t$ must be chosen).

The set of simultaneity constraints quite naturally translates to a set of forbidden pairs, and so we can prove that $\forall$CP is Π_3^p-complete simply by extending the reduction from $\exists\forall\neg$PWFP to CP: first, we must define the $\forall\exists\forall\neg$PWFP problem:

Definition 5.2 ($\forall\exists\forall\neg$PWFP). *Given a* PWFP *instance* $(G_0 = (V, E_0), C, s, t)$*, and a partitioning* $\{C_1, C_2, C_3\}$ *of* C*, the* $\forall\exists\forall\neg$PWFP *problem is: for every exclusive choice* X_1*, does there exist an exclusive choice* X_2 *for* C_2*, such that for every exclusive choice* X_3 *from* C_3*, there does* not *exist a path from* s *to* t *in the set of arcs* $E'_0 = (E_0 \setminus C) \cup X_1 \cup X_2 \cup X_3$*.*

Again, by using the reduction in [6], it is not hard to show that the $\forall\exists\forall\neg$PWFP problem is Π_3^p-complete.

For the reduction from $\forall\exists\forall\neg$PWFP to $\forall$CP, we associate the first set of forbidden pairs C_1 with the set of simultaneity constraints. That is, we construct the following gadget to model forbidden pairs in C_1 (Figure 5.1):

The forbidden pair $c_j = \{(x, y), (u, v)\}$ is modeled in a gadget similar to the path-blocking gadget of Figure 3.1, the difference being that now there exists a simultaneity constraint $x^j \bowtie y^j$ between x^j and y^j, and a simultaneity constraint $u^j \bowtie v^j$ between u^j and v^j.

Due to the precedence constraints $v^j \prec x^j$ and $y^j \prec u^j$, if both $x^j \prec y^j$ and $u^j \prec v^j$ were chosen to refine the simultaneity constraints, then a local

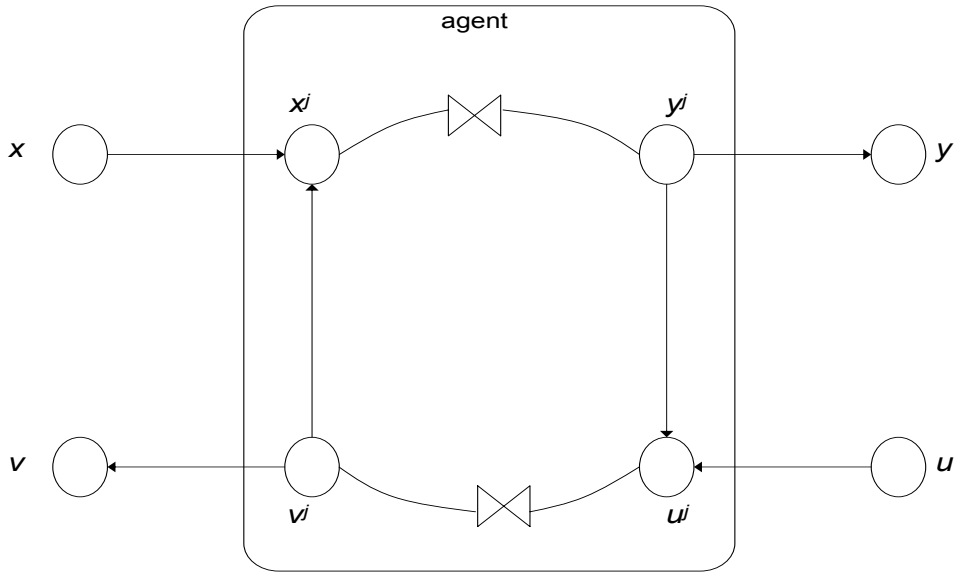

Figure 5.1: For this gadget, $a_i \bowtie b_i$ may be refined to $a_i \prec b_i$; $c_i \bowtie d_i$ may be refined to $d_i \prec c_i$, but not both.

cycle would result. Thus, either $x^j \prec y^j$, or $u^j \prec v^j$ may be chosen, but not both. Hence, the gadget of Figure 5.1 successfully models a forbidden pair.

Chapter 6

Subclasses of Coordination in NP

We have not found any non-trivial coordination instances in P, but we can identify structural characteristics of a coordination instance without which the coordination problem is in NP, and coordination verification in P.

Note that, intuitively, the difficulty with verifying a coordination set Δ is that we have to verify, for a possibly exponential number of sets of planning arcs $\hat{E}$, whether $E \cup \Delta \cup \hat{E}$ is acyclic if, for all $i = 1, \ldots, n$, $E_i \cup \Delta_i \cup \hat{E}_i$ is acyclic.

In this section, we identify a class of coordination instances for which, in order to solve the coordination verification problem, we merely have to check whether $E \cup \Delta \cup \hat{E}_{tot}$ contains no inter-agent cycles. Clearly, this check can be performed in polynomial time, so CVP is in P for these instances and, consequently, coordination itself is in NP.

Definition 6.1 (Local Planning Cycle). *A Local Planning Cycle, abbreviated as locplan-cycle, for agent A_i is an intra-agent cycle $C \subseteq (\hat{E}_i \cup E_i)$, such that $C \cap \hat{E}_i \neq \emptyset$ and $C \cap E_i \neq \emptyset$.*

Proposition 6.2. *Let $I = (\mathbf{T}, G = (T, E))$ be a coordination instance such that its coordination graph contains no locplan-cycles, then the following two assertions are equivalent:*

1. *The instance I is coordinated.*
2. *The coordination graph contains no inter-agent cycles.*

Proof.

(1: instance coordinated → no inter-agent cycles)
Suppose on the contrary that the instance is coordinated, yet the coordination graph still contains an inter-agent cycle C.

The fact that I is coordinated means that there does *not* exist a set $\hat{E} = \bigcup_{i=1}^{n} \hat{E}_i,\ \hat{E} \subseteq \hat{E}_{tot}$, such that (i) for all $i = 1, \ldots, n : \prec_i \cup \hat{E}_i$ is acyclic, while (ii) $(E \cup \hat{E})$ contains a cycle. Hence, the presence of the inter-agent cycle C implies that if we choose $\hat{E} = \hat{E}_{tot} \cap C$, then there must be at least one agent for which $\prec_i \cup \hat{E}_i$ contains a cycle C'.

It follows almost immediately that C' is a locplan-cycle:

- C' cannot consist exclusively of arcs in $\prec_i$, because $\prec$ is acyclic;
- C' cannot consist exclusively of arcs from $\hat{E}_i$, otherwise C would not be an elementary cycle: if all arcs in C' are also in C, then C is not elementary.

The presence of the locplan-cycle C' is a contradiction.

(2: no inter-agent cycle → instance coordinated)
If the coordination graph contains no inter-agent cycle, this clearly it is impossible for agents to add planning arcs $\hat{E} \subseteq \hat{E}_{tot}$ that would lead to a cycle in the joint plan involving more than one agent. Consequently, the instance is coordinated.

□

We cannot directly claim that if an instance contains no locplan-cycle, then it is in NP. We must also require that it is not possible that after adding some arcs Δ to $\prec_i$ (e.g., for coordination purposes), $\prec_i \cup \Delta \cup \hat{E}_i$ contains a locplan-cycle ($\hat{E}_{tot}$ defined with regard to the precedence relation $\prec \cup \Delta$). In other words, the coordination instance must not be *refinable* to a coordination instance that contains a locplan-cycle.

Proposition 6.3. *Let $I = (\mathbf{T}, (T, E))$ be a coordination instance that cannot be refined to contain a locplan-cycle, i.e., there exists no set of additional constraints $\Delta = \bigcup_{i=1}^{n} \Delta_i$, such that:*

1. $\Delta_i \subseteq (T_i \times T_i)$,

2. $\prec_i \cup \Delta_i$ *is acyclic, and*

3. *the coordination graph contains a locplan-cycle.*

Then I is in NP.

Proof. Let Δ be a solution for I, i.e., $I' = (\mathbf{T}, (T, E \cup \Delta)\ )$ is a coordinated instance. Verifying that Δ is solution for I can be done by verifying that the coordination graph contains no inter-agent cycle, according to Proposition 6.2. To prove that I is in NP, we need to show that this check can be performed in polynomial time.

Consider the subset feedback arc set problem, which is defined as follows:

> Given a tuple $I_{SFAS} = (V, E_0, X)$, with (V, E_0) a directed graph and $X \subseteq E_0$, find a minimum subset $F \subseteq E_0$, such that F contains at least one arc from every directed cycle in (V, E_0) that also intersects X.

The set of inter-agent cycles in the coordination graph is the set of cycles intersecting *INTER*, so we can choose $X = INTER$.

As SFAS is in NP[1], we can verify in polynomial time that $F = \emptyset$ is a subset feedback arc set for the coordination graph of $I' = (T, E \cup \Delta)$. This implies that we can verify in polynomial time that the coordination graph of I' contains no inter-agent cycles. □

At the moment, we do not yet know if we can verify, in polynomial time, whether an arbitrary coordination instance can contain locplan-cycles, i.e., whether Proposition 6.3 applies. However, even if such a check would require exponential time, Proposition 6.3 can still be of value if we can prove that a certain class of instances cannot contain locplan-cycles. For instance, in the following two corollaries, we identify classes of coordination instances that are in NP, on account of the fact that these instances cannot contain a locplan-cycle.

Corollary 6.4. *The set of coordination instances for which each agent has either at most one task in T^{in}, or at most one task in T^{out}, is in NP.*

[1]The decision variant of SFAS is in NP, SFAS itself is in NPO.

Proof. We will prove that we cannot form a locplan-cycle. Let C be an inter-agent cycle, let $\hat{E}_i = \hat{E}_{tot} \cap T_i \times T_i$, and let $\hat{E}_i(C)$ be the intersection of C with $\hat{E}_i$.

For this type of coordination instance, every arc in $\hat{E}_i$ has precisely one task (node) in common: if, for instance, an agent A_i has exactly one task t in T^{in}, then every arc in $\hat{E}_i$ has starting vertex t. Hence, C cannot be an elementary inter-agent cycle in case $\hat{E}_i(C)$ contains more than one arc.

In case $\hat{E}_i(C)$ is a single arc e, then it cannot create a locplan cycle: suppose on the contrary that there is a locplan cycle C' with e the only arc from $\hat{E}_i$. This implies $(\text{ran}(e), \text{dom}(e)) \in E^+$, and thus $e \in [E^{-1}]^+$. But then e cannot be in $\hat{E}_{tot}$. □

Corollary 6.5. *The set of coordination instances in which for all agents A_i, the sets $T^{in} \cap T_i$ and $T^{out} \cap T_i$ are totally ordered, is in NP.*

Proof. Let C be an inter-agent cycle, and suppose on the contrary that it is possible that C induces a locplan-cycle C' in some agent A_i. Let $\hat{E}_i = \hat{E}_{tot} \cap T_i \times T_i$, and let $\hat{E}_i(C)$ be the intersection of C (and C') with $\hat{E}_i$.

From the proof Corollary 6.4, we know that C' must contain at least two planning arcs. If we denote $C' = \{e_1, \ldots, e_m\}$, then let e_j be the first arc in $\hat{E}_i(C)$ and let e_k be the last arc in $\hat{E}_i(C)$ (first and last defined in terms of the indices i of arcs e_i in C').

We have $(\text{ran}(e_k), \text{dom}(e_j)) \in E^+$, since all arcs from $e_k \ldots e_m$ to $e_1 \ldots e_j$ are in E. However, we also have $(\text{dom}(e_j), \text{dom}(e_k)) \in E^+$, because all arcs in T^{in} are totally ordered. Together, this implies $(\text{ran}(e_k), \text{dom}(e_k)) \in E^+$. This, however, means that e_k can not be in $\hat{E}_i$. □

Chapter 7

Approximability of the Coordination Problem

In this section, we will show that the coordination problem is APX-hard. We do this by reducing the APX-hard problem Feedback Vertex Set (FVS) to the coordination problem. Furthermore, this reduction shows that it is likely that the coordination problem is outside APX, that is, it is unlikely that there exist constant-ratio approximations for the coordination problem. If there did exist constant-ratio approximations for CP, then the following reduction would immediately yield a constant-ratio approximation for FVS; however, despite much research effort, the best known approximations for FVS are $\mathcal{O}(\log |V| \log \log |V|)$ [4, 3].

Although slightly disappointing, it is of course not surpsrising that coordination is APX-hard, as it is outside NPO. However, the reduction will show that even for severly restricted coordination instances, the APX-hardness result holds. In fact, we reduce the FVS problem to coordination instances where agents have at most two tasks. Due to Proposition 6.2, these instances are clearly in NP, yet they are still APX-hard.

The feedback vertex set problem is defined as follows:

Definition 7.1. *The Feedback Vertex Set problem is: given a directed $G_0 = (V, E_0)$ and integer K, find a subset of vertices $F \subseteq V$ such that F contains at least one vertex for every directed cycle in G, and $|F| \leq K$.*

To reduce FVS to CP, we split each vertex up into two vertices — one vertex incident on all incoming arcs, the other incident on all outgoing arcs

— and create an agent for those two tasks. The reduction is illustrated in Figure 7.1.

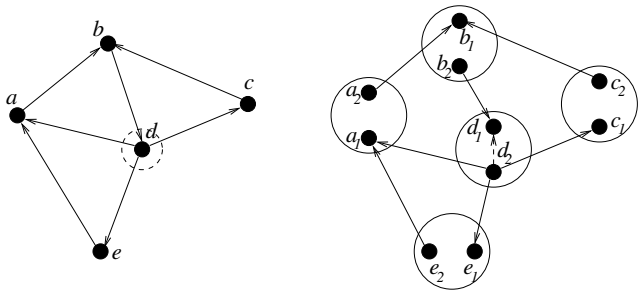

Figure 7.1: (a) an FVS instance and (b) its corresponding CP instance, coordinated by adding constraint (d_2, d_1).

More formally, given a FVS instance $G_0 = (V, E_0)$, we obtain a CP instance $(\mathbf{T}, G = (T, E))$ by the following transformation:

1. For every $v_j \in V$, we construct the agent A_j, having set of tasks $T_j = \{t_j^i, t_j^o\}$.
2. For every $(v_j, v_k) \in E_0$, E contains the arc (t_j^o, t_k^i).

The correspondence between a feedback arc set and a coordination set is given by:

$$v_j \in F \leftrightarrow (t_j^o, t_j^i) \in \Delta$$

Proposition 7.2. *The Feedback Vertex Set problem reduces to the coordination problem.*

The correctness of Proposition 7.2 can be understood by noting that a cycle in the FVS-instance maps to a 'potential cycle' in the coordination instance: that is, if all agents A_j along the cycle were to add the planning arc (t_j^i, t_j^o), then a cyclic joint plan would result. The transformed CP instance is solved if and only if, for every such potential cycle, at least one agent A_j intersected by the cycle adds the constraint (t_j^o, t_j^i). Consequently, a coordination set of cardinality K maps to a feedback vertex set of cardinality K — and vice versa.

Corollary 7.3. *The coordination problem is APX-hard.*

Chapter 8

Conclusions and Related Work

In this technical report we have analyzed the complexity of coordinating autonomous agents that have to work together on a joint task. The coordination problem, which is to find a minimal set of constraints to allow agents to plan independently, turns out to be computanionally hard — it is Σ_2^p-complete. We can identify coordination instances that are 'only' NP-complete, but NP-complete is still regarded as intractable. Finally, the coordination problem is hard to approximate, as it is unlikely that there exist constant-ratio approximations for the coordination problem.

These complexity results do not form a promising basis for constructing efficient coordination algorithms. However, rather than giving in dejectedly, we instead shift our attention to developing distributed coordination protocols that allow agents to remain autonomous not only during planning, but also in the coordination phase. Early versions of these coordination protocols are presented in [8]. In [7], we advance these coordination protocols, and discuss the kind of strategies that are open to agents. Also, we analyze the efficiency of these coordination protocols not only in terms of planning autonomy (i.e., in terms of the number of additional precedence constraints) as we have done in this paper, but also in terms of *plan cost*.

Bibliography

[1] Keith S. Decker and Victor R. Lesser. Designing a family of coordination algorithms. In *Proceedings of the Thirteenth International Workshop on Distributed Artificial Intelligence (DAI-94)*, pages 65–84, 1994.

[2] Eithan Ephrati and Jeffrey S. Rosenschein. Multi-agent planning as the process of merging distributed sub-plans. In *Proceedings of the Twelfth International Workshop on Distributed Artificial Intelligence (DAI-93)*, pages 115–129, May 1993.

[3] Guy Even, Joseph Naor, Baruch Schieber, and Madhu Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2), 1998.

[4] P.D. Seymour. Packing directed circuits fractionally. *Combinatorics*, 15, 1995.

[5] Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 1998.

[6] Stefan Szeider. Finding paths in graphs avoiding forbidden transitions. *Discrete Applied Mathematics*, 126:261–273, 2003.

[7] Jeroen Valk. *Coordination in Multi-Agent Systems*. PhD thesis, Delft University of Technology, 2004.

[8] Jeroen Valk and Cees Witteveen. Multi-agent coordination in planning. In *Seventh Pacific Rim International Conference on Artificial Intelligence*, pages 335–344, Tokyo, Japan, august 2002. Springer.

[9] Frank von Martial. *Coordinating Plans of Autonomous Agents*, volume 610 of *Lecture Notes on Artificial Intelligence*. Springer Verlag, Berlin, 1992.