COORDINATING AUTONOMOUS PLANNING AND SCHEDULING

Chetan Yadati Narasimha

॥ श्री नरहरि सद्गुरु प्रसन्नः ॥

Coordinating Autonomous Planning and Scheduling

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties in het openbaar
te verdedigen op maandag 28 januari 2013 om 15.00 uur
door

Chetan YADATI NARASIMHA

Master of Science in Industrial Engineering and Management, Oklahoma
State University, USA
geboren te Bangalore, India

SIKS Dissertation Series No. 2013-04. The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Netherlands Organisation for Scientific Research This work is part of the research programme 612.066.411, which is financed by the Netherlands Organisation for Scientific Research (NWO).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter we hope to demonstrate the problems of plan and schedule coordination through a detailed example. The example has a set of agents who desire to (i) *plan or schedule autonomously and* (ii) *attempt to achieve a joint goal. It reveals that without a coordination mechanism, if we allow agents to plan (schedule) autonomously, then it can lead to failure in achieving their joint goal. The chapter then moves on to provide an overview of the remaining chapters and also list the research contributions of this thesis.*

1.1 Motivating example

The need for coordinating plans and schedules arises in several domains such as supply chains, distributed manufacturing and hospital patient treatment. Unfortunately, despite the practical abundance of coordination problems, efficient mechanisms to solve coordination problems are unavailable as these problems have been shown to be computationally intractable. The motivating example below is an illustration of the problem of plan and schedule coordination in the transportation domain.

Crude oil is a naturally occurring mixture of hydrocarbons. It is the source of several products such as petrol, diesel and kerosene. The huge number of consumable products, specially the energy related ones, that can be derived from crude oil gives it a very central position in the economy of the world [Bacon 2011].

It is generally believed that crude oil was formed from biomass (from animal and plant remains) getting compressed and heated by geological heat within layers of earth over thousands of years. Geological formations that capture crude oil occur at various places below the earth surface. Many times, these formations occur at the bottom of the sea as well. Generally, because of their geology, crude oil drilling installations (drilling rigs) do not contain facilities for refining crude oil. Therefore, crude oil is transported to locations where such refineries exist.

Drilling rigs require large amounts of diesel to operate. However, because they cannot refine the obtained crude oil, diesel has to be procured to operate the rigs. Let us illustrate this journey of crude oil from below the earth to the refinery and back into the machines that drill the earth.

Suppose *Location A* contains the drilling rig and *Location C*, *Location D* and *Location E* contain the refineries. Diesel required to run the drills is transported from these refineries to the rig at *Location A*. Crude oil is transported via *Location B* to reach the refinery in *Location C* and *Location D*, whereas it can directly go from *Location A* to *Location E*. The same route (in the reverse direction) is followed for the transportation of diesel from the refineries to the drilling rig. The refinery in *Location D* is close to a rail track and the refinery in *Location E* is close to an airport. Therefore, different modes of transport have to be followed to reach each of the locations. Further, the routes to be followed impose constraints on the order of transportation activities. For instance, crude oil arriving through Ship 1 cannot be transported from *Location B* until the truck from *Location A* has transported it to Ship 1 and Ship 1 has in turn transported it to *Location B*. Similarly, diesel from *Location E* has to be first transported by a truck to the plane which then transports it to *Location A* and so on. These constraints that establish a partial order on the set of (transportation) activities are referred to as *precedence constraints* and are represented using the symbol $\prec$. Thus, $t_1 \prec t_2$ implies that task t_1 *precedes* task t_2.

Each of the trucks, trains, ships and planes used for transportation is owned and operated by companies that are different from the drilling or the refinery company. This scenario is represented in Figure 1.1. The arrows represent the movement of crude oil (diesel) containers. The tasks allocated to each company are shown in the rectangular boxes above the transportation vehicle used by the company.

Each transportation company could be involved not just in transporting crude

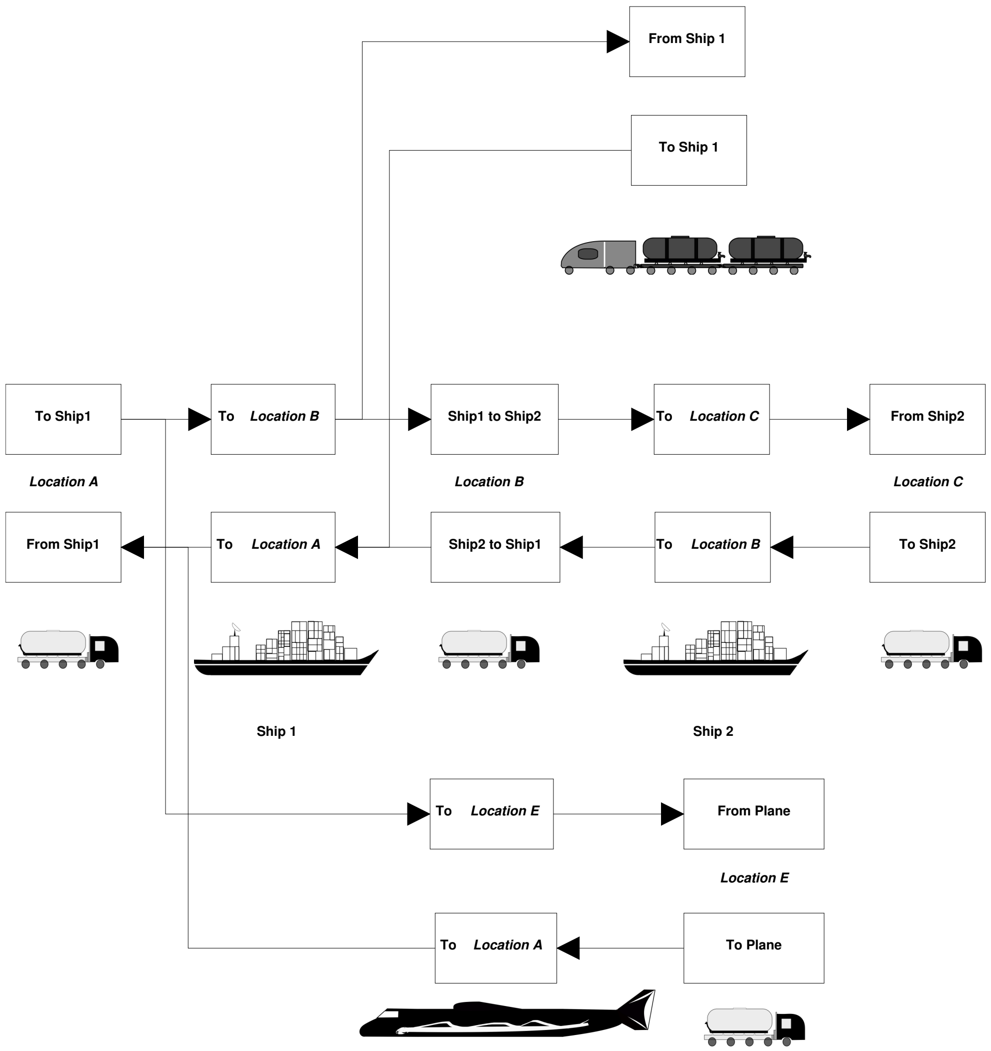


Figure 1.1: A multi-modal transportation scenario.

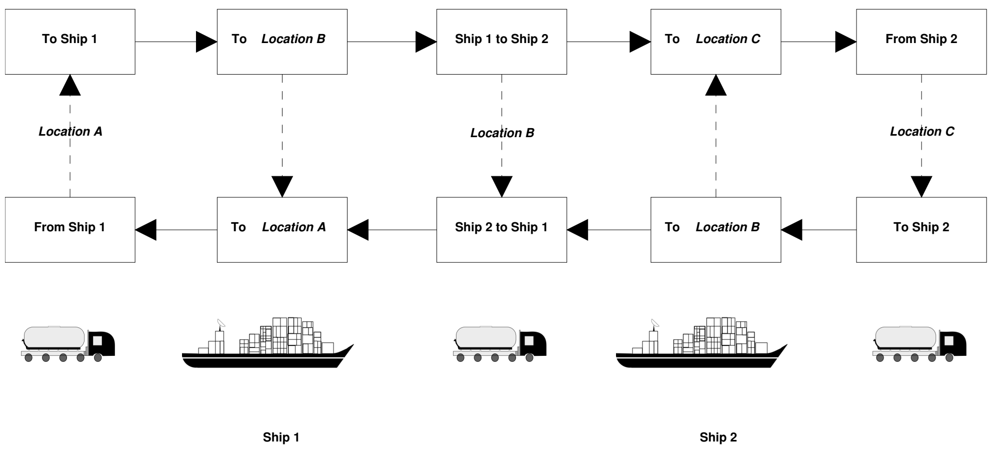


Figure 1.2: Each transportation company makes an acyclic local plan but in combination it leads to a cycle in the global plan.

oil/diesel, but also other products (or probably involved in transportation activities related to several drilling rigs). Transportation companies cannot exchange information regarding their activities between themselves owing to information privacy issues. Therefore, designing a local plan that best suits a company's interests has to be done *autonomously.* A local plan here refers to the intended order in which a company wants to carry out its tasks. In our case,

> *a local plan for each of the transportation companies is a partial order on the set of activities it has to accomplish.*

We know that the transportation companies make local plans for their activities. Therefore, we consider the global plan for all the tasks to be a combination of all local plans. The question, however, is whether such a global plan is a *feasible plan.*

To ensure that drilling happens smoothly, the transportation companies have to ensure that their local plans are *coordinated.* To see why feasibility could be a problem, suppose now that the companies involved in transportation between *Location A* and *Location C* make their plans as shown in Figure 1.2. The local plan (a partial ordering of the tasks) for each of the transportation companies is indicated by a dotted arrow between the tasks of the company. Here a dotted arrow from task t to a task t' means that, task t is planned to be executed before task t'. Consider now a global plan formed by the combination of local plans shown in Figure 1.2. A directed cycle is formed between the partial orders (plans) of the truck agent at *Location A* and *Ship 1*. This directed cycle means that Ship 1 intends to carry the crude oil from the refinery before it transports diesel to the refinery. On the other hand the truck wants to transport diesel first and then transport crude oil. Clearly this situation implies that the global plan is infeasible. From this example we now have a simple test to determine if a given global plan is coordinated — if no directed cycle exists in the global plan then the plan is coordinated. Note here that each transportation company's plan is locally coordinated (because there are no local cycles). However, in combination they result in an uncoordinated global plan.

One solution to the problem would be to centrally develop a plan to perform the whole transportation. However, this would not be practical because centrally creating a plan for this transportation scenario could conflict with the other commitments of the company. Another solution would be for each company to *inform* a central coordinator about their plans. The central coordinator could then detect all possible conflicts, resolve them and inform the companies regarding the changes they have to make to ensure coordination. This solution too is unacceptable because this might require companies to share private information or make cascading changes that affect their other commitments. *Therefore, we need a mechanism that coordinates plans without requiring companies to part with any information regarding their plans.*

Suppose now that the precedence constraints *To Ship1* $\prec$ *From Ship 1; To* Location B $\prec$ *To* Location A*; Ship 1 to Ship 2* $\prec$ *Ship 2 to Ship 1; To* Location C $\prec$ *To* Location B and *To Plane* $\prec$ *From Plane*, are added to the problem as shown in Figure 1.3. Agents are then allowed to make their local plans based on this updated problem. Notice now that whatever plan the truck agent in *Location C*, the train agent in *Location D* or the Plane agent make, the global plan will be cycle free. This means that it is possible to allow some degree of autonomy while still ensuring a coordinated global plan.

This solution requires only 5 constraints whereas a centralised solution would imply 8 constraints (one on each transportation company). This reduction in the

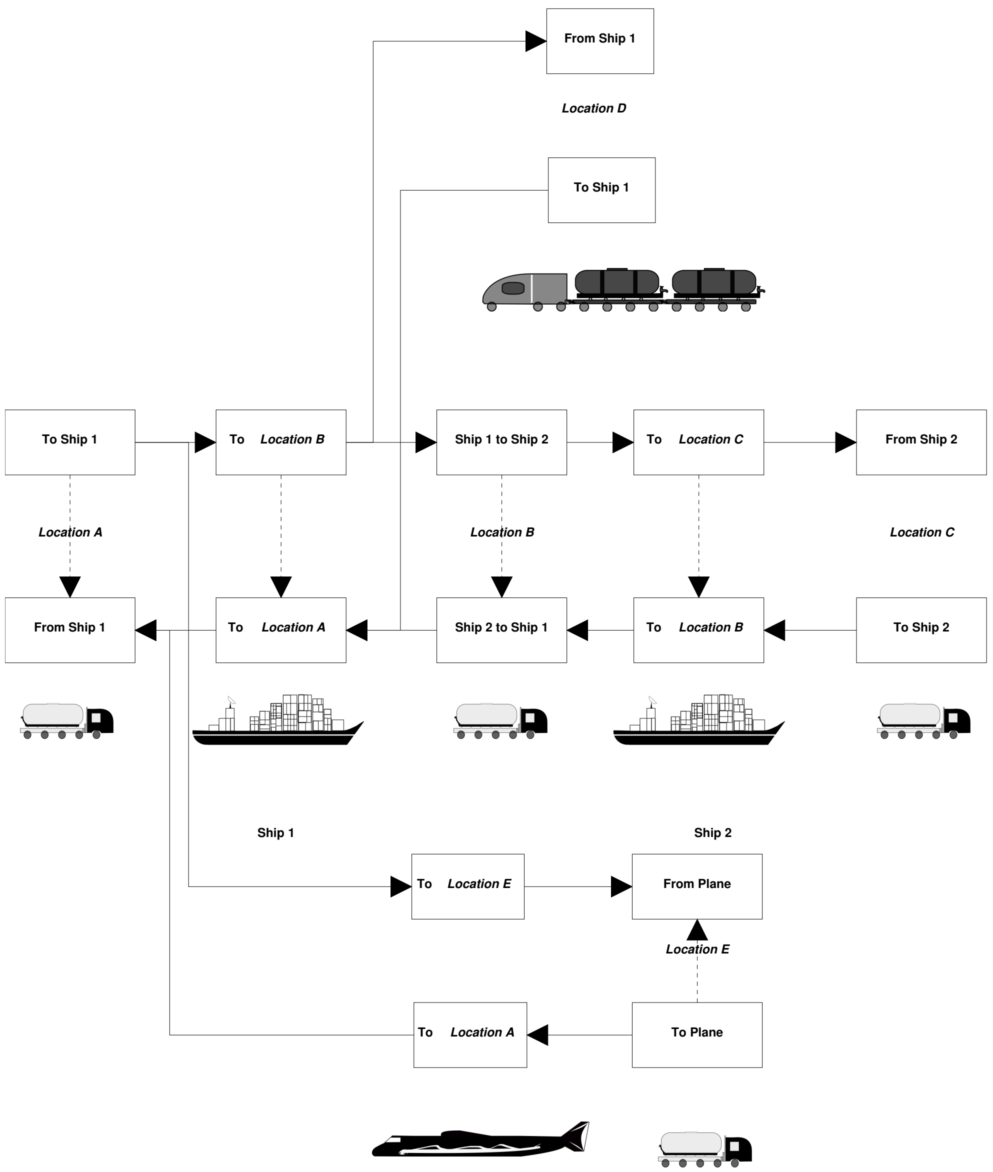


Figure 1.3: A minimal set of coordination constraints.

number of constraints implies that three agents, truck agent at *Location C*, the train agent at *Location D* and the plane agent can choose to either transport crude oil first or transport diesel first without affecting the feasibility of the global solution. We see this increase in the amount of choices to agents (while retaining the feasibility) as an increase in the amount of *autonomy* offered to the agents.

In other words, if we can ensure that the number of constraints added to make the global plan feasible are minimum, then it is the maximum possible autonomy one can guarantee. Unfortunately, earlier research by Valk [Valk 2005] has shown that in general, this problem of finding the minimum number of constraints that ensure coordination is Σ_2^p-complete. The complexity is derived from the problem of verifying if a set of constraints is sufficient to ensure coordination and the problem of finding a sufficient constraint set of minimum cardinality, both of which are NP-hard in general. Therefore in this thesis, our first goal is to discover subclasses of the general case where this problem can be solved more efficiently.

Let us now look at the same oil transportation scenario assuming that we have more detailed information about it. In this more detailed or extended scenario, we recognise that *(i)* the time taken for each of the transportation companies for their activities is different and *(ii)* the capacity of each of these modes of transport is different as well. Therefore we collect information regarding the travelling times of each of the modes of transport and the number of vehicles used by each company for this transportation job.

Ensuring that the drilling rig can operate unhindered, now requires a different approach than earlier. We would have to also take the duration, the exact time instant that each task starts and the number of vehicles used by each company into consideration. In other words, we would have to *coordinate the schedules* of the transportation companies. To illustrate that coordinating plans alone is insufficient in this extended scenario, consider again the route between *Location A* and *Location C*.

The durations mentioned below the vehicles indicate the time required for the company to carry out its transportation tasks. To simplify our discussion, let us assume that each company has *infinite resources* (number of vehicles) for performing its tasks. Suppose, the whole transportation job starts at 0:00 hours on a Monday. That is, the crude oil is available for transportation at *Location A* at 0:00 hours on Day 1 and similarly, the diesel required for drilling is also available at the refinery

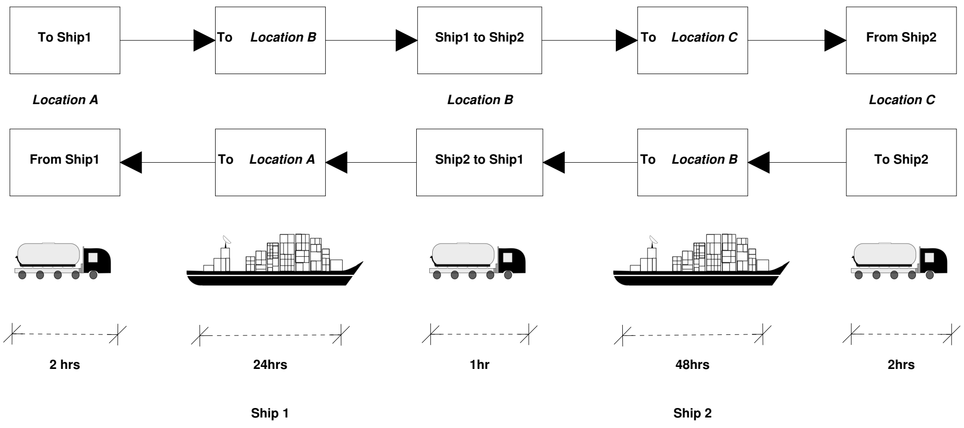


Figure 1.4: Schedule coordination problem.

Task	**Starting Time**
To Ship 1	Day 1, 12:00 hours
To *Location B* (Ship 1)	Day 2, 08:00 hours
Ship 1 to Ship 2	Day 3, 08:00 hours
To *Location C*	Day 3, 16:00 hours
From Ship 1	Day 5, 12:00 hours
To *Location A*	Day 5, 12:00 hours
Ship 2 to Ship 1	Day 5, 12:00 hours
To *Location B* (Ship 2)	Day 6, 00:00 hours
From Ship 2	Day 6, 00:00 hours
To Ship 2	Day 6, 08:00 hours

Table 1.1: Schedules made by different agents for transportation activities between *Location A* and *Location C*.

at *Location c* at 0:00 hours on Day 1. Now, suppose the various companies make schedules as shown in Table 1.1. Notice that from the starting times of each of the tasks we can easily derive the local plans (partial orders) of the transportation companies. In fact, these local plans are also compatible with each other as can be

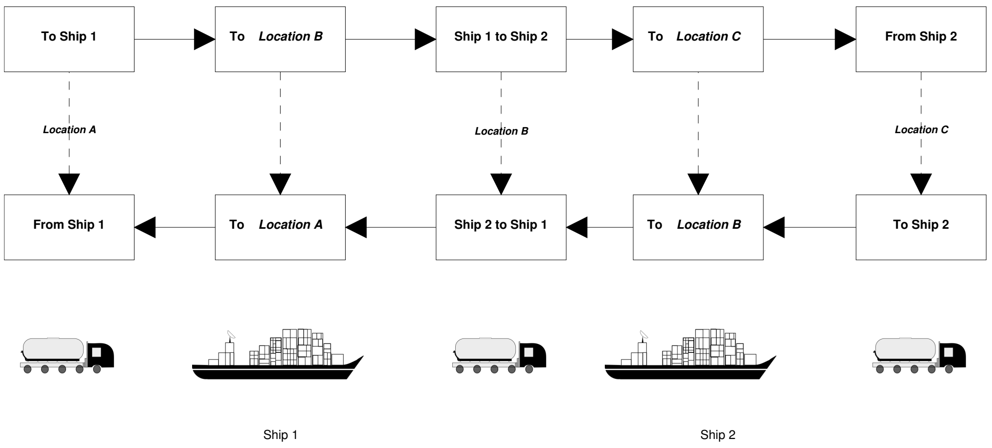


Figure 1.5: A feasible plan, but an infeasible schedule.

seen in Figure 1.5. However, if transportation companies were to start their activities as planned above then the whole operation would be infeasible. The truck company at *Location A* cannot transport diesel to the drilling rig because Ship 1 would not have brought it to *Location A* at the time when the truck leaves the shipping port. The same holds for the truck company in *Location B* because Ship 2 will not have arrived before it starts moving from the shipyard.

Suppose, we add the constraints shown in Table 1.2 to the problem. Then any local schedule that respects these constraints will now be coordinated. Thus, both the schedules in Table 1.3 are coordinated.

Notice that the local schedule of Ship agent (Ship 1) need not change along with the schedule of the truck agent in *Location A*. It could continue to schedule its task at Day 1, 03:00 hours, even when the truck agent starts its task at Day 1, 00:30 hours. Note that if the Ship agent (Ship 1) now changes its schedule to Day 1, 03:30 hours, it does not affect the feasibility of the overall schedule. Therefore, we can clearly see that agents have the autonomy to choose from multiple local schedules without affecting the feasibility of the global schedule. However, also note that agents are not completely free, for instance, Ship agent (Ship 1) cannot change its schedule to Day 1, 04:30 hours. Such a change would create an infeasibility. We therefore have

Task	**Earliest starting time**	**Latest starting time**
To Ship 1	Day 1, 00:00 hours	Day 1, 01:00 hours
To *Location B* (Ship 1)	Day 1, 03:00 hours	Day 1, 04:00 hours
Ship 1 to Ship 2	Day 2, 04:00 hours	Day 2, 05:00 hours
To *Location C*	Day 2, 06:00 hours	Day 2, 07:00 hours
From Ship 2	Day 4, 07:00 hours	Day 4, 08:00 hours
To Ship 2	Day 1, 00:00 hours	Day 1, 01:00 hours
To *Location B* (Ship 2)	Day 3, 01:00 hours	Day 3, 02:00 hours
Ship 2 to Ship 1	Day 3, 03:00 hours	Day 3, 04:00 hours
To *Location A*	Day 4, 04:00 hours	Day 4, 05:00 hours
From Ship 1	Day 4, 07:00 hours	Day 4, 08:00 hours

Table 1.2: Coordination constraints are added to the scheduling problem to ensure feasibility.

a trade-off between the autonomy allowed to agents and the overall makespan.

We have seen from the scenario in Figure 1.4 that coordinating plans is not sufficient to ensure that the drilling operation can work unhindered. What we require thus, is a mechanism that coordinates schedules. Further, such a mechanism must also allow for agent autonomy for the same reasons as those for planning. Studying this problem of coordinating schedules is the second goal of this thesis.

1.2 Structure of this thesis

In this thesis we deal with both the coordination of plans and schedules. In the first part of the thesis we deal with the so called *plan coordination problem* (in Chapter 3). The general problem has been proven to be intractable [Valk 2005]. However, there exist several practical problems such as supply chain management and hospital patient treatment that exhibit special properties which can be exploited. Therefore, we focus on defining restricted subclasses of this problem that are easier to solve and then develop algorithms to solve these problems efficiently.

In the second part, we develop algorithms that ensure *schedule coordination.* The mechanisms we develop aim at ensuring that *(i)* the global schedule (derived

Task	**Schedule 1**	**Schedule 2**
	Starting time	**Starting time**
To Ship 1	Day 1, 0:00 hours	Day 1, 0:30 hours
To *Location B* (Ship 1)	Day 1, 3:00 hours	Day 1, 3:00 hours
Ship 1 to Ship 2	Day 2, 4:00 hours	Day 2, 4:30 hours
To *Location C*	Day 2, 6:00 hours	Day 2, 6:30 hours
From Ship 2	Day 4, 7:00 hours	Day 4, 7:00 hours
To Ship 2	Day 1, 0:00 hours	Day 1, 0:30 hours
To *Location B* (Ship 2)	Day 3, 1:00 hours	Day 3, 1:30 hours
Ship 2 to Ship 1	Day 3, 3:00 hours	Day 3, 3:00 hours
To *Location A*	Day 4, 4:00 hours	Day 4, 4:30 hours
From Ship 1	Day 4, 7:00 hours	Day 4, 7:30 hours

Table 1.3: Two feasible global schedules.

as a combination of local schedules) is always feasible *(ii)* agents can design their schedules independently and *(iii)* also that the makespan of the global schedule is as close to optimality as possible.

Schedule coordination includes notions of both time (durations, starting times) and capacity (number of vehicles used). Thus we deal with the schedule coordination problem in two chapters *(i)* Chapter 4, when there is unbounded capacity and *(ii)* Chapter 5, when the capacity of agents is bounded. In Chapter 4 we develop an efficient technique — ISA— that designs coordination constraints such that local plans that satisfy these constraints shall always lead to a makespan minimal solution.

In Chapter 5, we study the case where agents have fixed bounds on their capacity. Here we first show that the general problem is intractable and then devise algorithms for various restricted classes of the problem. The algorithms we devise in this chapter have varying degrees of success in ensuring makespan minimality, but they always ensure that agents can develop schedules independently and that their local schedules can always be combined to derive a feasible global schedule.

In Chapter 6, we perform an empirical evaluation of $\mathrm{M}_{\mathrm{ISAn}}$, an algorithm developed in Chapter 5 to show the practicality of its use.

The research contributions of this thesis are listed in the following section.

1.3 Research contributions

This thesis contributes to research in multi-agent coordination in the following ways:

Plan coordination

- We study a class of plan coordination instances called the *intra-free instances*. We represent graphs of intra-free instances using a compact representation called *agent dependency graphs*. Using agent dependency graphs, we show that the coordination verification problem for intra-free instances is solvable in polynomial time, thus showing that limiting problem instances to intra-free instances reduces the complexity of finding minimum coordination sets from being Σ_2^p-complete to being NP-complete.

- We present a heuristic method, DP^* algorithm, for coordinating intra-free instances. We also show that DP^* algorithm performs better than the current state of the art *depth partitioning method* for solving intra-free plan coordination instances.

- We show that a restricted subclass of intra-free instances called the *special linear intra-free instances* can be coordinated in polynomial time. We also present a procedure to determine the coordination set in polynomial time.

Schedule coordination

- We present an efficient algorithm — ISA— to solve the case of coordinating schedules when agents have unbounded concurrency. ISA is derived from the temporal decoupling algorithm of Hunsberger and is shown to be asymptotically faster than Hunsberger's algorithm in solving schedule coordination problems.

- We prove that the schedule coordination problem is NP-complete when agents have bounded capacity. In addition, we prove that even when agents are sequential and tasks are of unit duration, the complexity of the schedule coordination problem stays NP-complete.

- For the subclass of schedule coordination problems with sequential agents and unit duration tasks, we propose a matching based algorithm $\mathrm{M}_{\mathrm{ISAsq}}$ that is adapted from ISA. The approximation ratio of this algorithm is found to be not very good on the problem with a general task structure. Thus, we develop further heuristics that exploit the task graph structure and improve the approximation ratio in more restricted subclasses (grids and unequal chains) of problem instances.

- Finally, we propose $\mathrm{M}_{\mathrm{ISAn}}$ based on $\mathrm{M}_{\mathrm{ISAsq}}$ to handle the general case of the schedule coordination problem. We apply $\mathrm{M}_{\mathrm{ISAn}}$ to the problem of coordinating ground handling operations at an airport. By applying $\mathrm{M}_{\mathrm{ISAn}}$ to this real-world problem, we gain additional insights on the performance of the proposed algorithm and in addition, the theoretical results that were thus far derived are validated in an empirical setting.

Following is the list (in chronological order) of publications that resulted out of the research carried out for this thesis.

- C. Yadati, C. Witteveen, Y. Zhang, M. Wu and H.La Poutré. *Autonomous Scheduling.* In Proceedings of the Foundations of Computer Science, pages 73 – 79, 2008.

- C. Yadati, C. Witteveen, Y. Zhang, M. Wu and H. La Poutré. *Autonomous Scheduling with Unbounded and Bounded Agents.* In Proceedings of the 6th German conference on Multiagent System Technologies, MATES '08, pages 195–206, 2008.

- A. terMors, C. Yadati, C. Witteveen and Y. Zhang. *Coordination by design and the price of autonomy.* Autonomous Agents and Multi-Agent Systems, volume 20, pages 308–341, 2010.

- C. Yadati, C. Witteveen and Y. Zhang. *COORDINATING AGENTS: An analysis of coordination in supply-chain management like tasks.* In The 2nd International Conference on Agents and Artificial Intelligence, volume 2, pages 218–223, 2010.

- C. Yadati, C. Witteveen and Y. Zhang. *Improving Task-Based Plan Coordination.* In Collaborative Agents - Research and Development, volume 6066 of *Lecture Notes in Computer Science*, pages 175–186. 2011.

Chapter 2

Plan and schedule coordination mechanisms: An overview

In this chapter, we perform a survey of existing methods for plan and schedule coordination. The survey basically categorises coordination methods into two broad categories — methods that use communication and methods do not use communication. Since our interest is to allow for agent autonomy, we favor the methods that do not use communication to achieve coordination. This survey also provides the context to introduce our framework and define plan and schedule coordination problems formally.

Whenever multiple autonomous decision makers come together to achieve a goal beyond their individual capabilities, coordination is required. Coordination is a necessity in a wide variety of problems — traffic control, disaster relief, hospital management, factory management, military operations and so on (cf. [Steeb *et al.* 1981,Kovács and Spens 2007,Argote 1982,Cicirello and Smith 2004,Azarewicz *et al.* 1989]). While it is easy to define the lack of coordination, the exact definition of coordination seems to be application dependent. It is defined as the management of dependencies between activities by some [Malone and Crowston 1994]. Others define it as the integration and harmonious adjustment of individual work efforts towards the accomplishment of a larger goal [Singh 1992]. The specifics of each definition is probably inspired by the class of problems dealt by the author, but the motivation for coordination is in most situations the same.

Coordination is necessary because (see [Jennings 1996]) of the following reasons:

1. The decisions and actions of a single agent can affect the decisions and activities of other agents in the system. For example, the decision of a car driver to switch lanes can affect the speed and the lane taken by car drivers behind him.

2. Achieving the global goal requires that individual goals do not conflict with each other. For example, vehicles must travel on the left (right) side of the road, otherwise they may interfere with oncoming traffic.

The first point indicates the interdependency between agents and the second indicates the difficulty with uncoordinated plans of agents. Clearly, if there are no traffic rules to coordinate the behaviour of drivers, then the community of decision makers (drivers) could quickly degenerate into chaos [Jennings 1996].

Agents take actions/decisions throughout the life time of a multi-agent system and hence coordination also needs to occur throughout the life cycle of a system. In this thesis however, we are interested only in coordinating *plans and schedules* of agents in a multi-agent system. In other words, we are interested in coordination problems that occur *before* a system is in operation.

Concerning plan coordination, Valk (see [Valk 2005]) has shown that coordinating autonomous plans is intractable in general. Therefore, our quest in this thesis is to answer the following question concerning plan coordination:

- Do there exist subclasses of plan coordination problem instances where an efficient process exists to find coordinated global plans?

Concerning schedule coordination, as far as we could find, there exists no result which establishes the complexity of the problem. Thus, in this thesis, we try to answer the following question regarding schedule coordination:

- Can we design a mechanism which guarantees that autonomously developed schedules of agents, are always coordinated? Further, can we ensure that such a coordinated global schedule also satisfies additional *quality* criteria such as makespan minimality?

Note that while requirements such as efficiency, quality and information privacy have well established meanings, the requirement for autonomy is somewhat hazy. For our purposes, we consider autonomy as the possibility for an individual agent to determine its own plan, given a set of tasks to perform. That is, to decide on its own which plan to construct independently from the other agents. Of course there are limiting scenarios where the task constraints only allow for one solution, and in general, the more alternative plans that can be constructed, the larger the autonomy of an individual agent is. An elaborate discussion of autonomy and methods to maximize it, however, is beyond the scope of this thesis.

Coordinated multi-agent systems A coordinated multi-agent system can be broadly described using Figure 2.1. In this figure, we have a central system (problem owner) which determines the allocation of tasks in the coordination problem to a set of agents. These agents in turn develop a plan(schedule) for their local set of tasks, which are then merged into the global plan(schedule) by the problem owner. The coordination layer (the band in grey) ensures that this merging of local solutions does not result in contradictions within the global solution.

In coordinated multi-agent systems, agents make independent decisions and take actions to satisfy their own individual agent goals. However, the coordination layer ensures that these agent goals

1. neither conflict with each other nor

2. conflict with the system goal.

The coordination layer, therefore, needs to interact with both the problem owner (an entity other than any of the agents involved making plans or schedules), who defines the system goal, as well as the agents in order to achieve coordination. Considering the traffic example presented earlier, the multi-agent system contains drivers who each need to go from a source to a destination. The system goal is to ensure that traffic flow through the roads is smooth and accident free. The coordination layer consists of traffic rules which ensure that agent goals and system goals are achieved.

The process of coordination is essentially a process of managing inter-agent dependencies as pointed out by [Malone and Crowston 1994]. Differences between coordinating mechanisms arise in how they manage these inter-agent dependencies. All

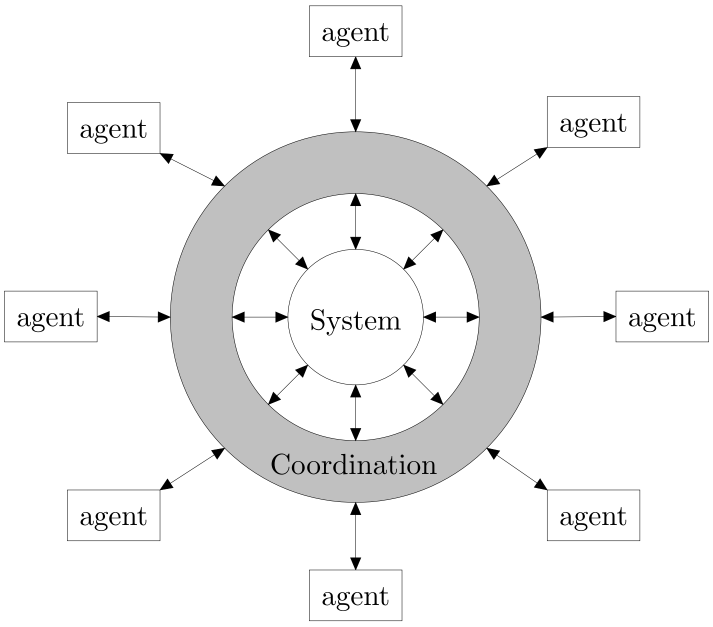


Figure 2.1: Coordination in multi-agent systems.

coordination approaches can be broadly classified based on who initiates coordination. In most systems, (cf. [Smith and Peot 1996, Younes and Simmons 2003, Durfee and Lesser 1987, Decker and Lesser 1992, Schubert and A.Gerevini 1995, Kutanoglu and Wu 1999, Solberg and Lin 1992, Dewan and Joshi 2002, Kouiss *et al.* 1997, Liu and Sycara 1995]) coordination is initiated by the agents themselves. That is, agents make their individual plans (schedules) and later communicate with a coordination layer. The coordination layer determines whether there are conflicts between agent plans (schedules) and suggests changes to be made to individual plans (schedules) (cf. [Cox and Durfee 2003, Durfee and Lesser 1987, Ephrati and Rosenschein 1993]). To determine whether conflicts exist, the coordination layer can adopt one or both of the following two approaches:

Method 1a: Analyse, determine and resolve conflicts centrally. In this method, a central entity determines if the plan of any agent is creating conflicts and then suggests changes to the individual plans of agents. It is easy to see that in this system agents need to be cooperative to accept all the suggested changes (cf. [von Martial 1992, Ephrati and Rosenschein 1994, Weerdt *et al.* 2003, Alami *et al.* 1998, Foulser *et al.* 1992]).

Method 1b: Analyse, determine and resolve conflicts through agent interaction. If this method is adopted, agents first have to themselves determine if there are conflicts between their plans. They have to then determine, ways to change their solutions so that conflicts are avoided. This is a common approach adopted when agents are non-cooperative (cf. [der Krogt and Weerdt 2005b, Wangermann and Stengel 1998, Gerkey and Mataric 2002, Gerkey and Mataric 2003a]).

Notice that in both methods, the coordination layer acts as a communication or an interaction layer that ensures conflicts do not arise. Hence, we also refer to these methods as interaction or communication based methods. This situation is depicted in Figure 2.2.

Few approaches, however, the attempt to coordinate is initiated by the problem owner itself (see [Korf 1987, Valk 2005]). Suppose agents are unable to interact amongst themselves and are also not cooperative (examples include scenarios where communication can compromise national security or information privacy needs). In such situations, the system has to assume the initiative in ensuring that agents can act independently of each other. This means that all inter-agent dependencies have to be removed or made redundant before hand. One approach to ensuring that inter-agent dependencies are redundant, is by adding additional (tighter) intra-agent constraints such that together they imply the inter-agent constraints. The problem of coordination in this situation, can be viewed as the following decomposition problem:

How to find a set of additional constraints, such that the original multi-agent planning (scheduling) problem can be solved, by solving a set of single agent planning (scheduling) problems independently?

This approach is illustrated in Figure 2.3. In this scenario, the coordination layer acts as a decomposition layer which makes all inter-agent dependencies redundant. Hence, we alternatively refer to these approaches as decomposition based approaches.

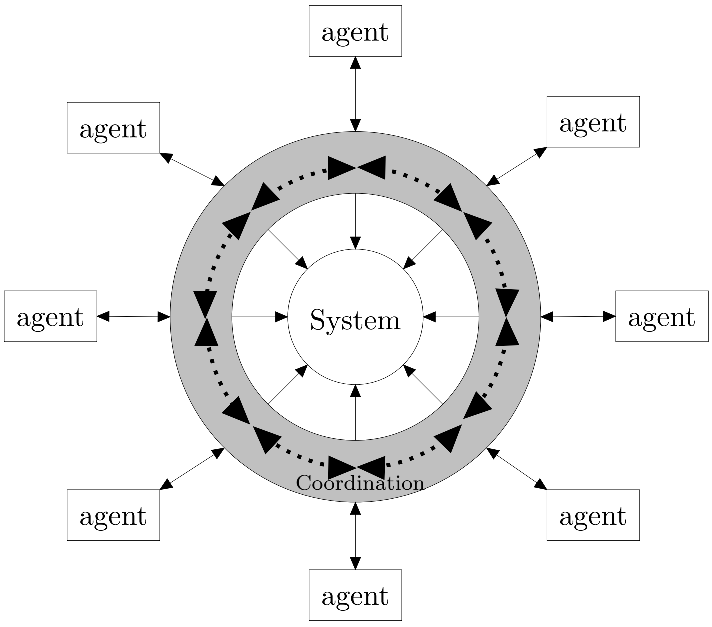


Figure 2.2: Coordination initiated by agents. Notice that the coordination layer acts as a communication layer between agents.

In this thesis we adopt this second approach. We shall see in further sections that there are different methods of accomplishing this decomposition.

Both approaches to coordination have their own advantages as well as disadvantages. While interaction allows agents to handle dynamic systems more effectively, decomposition based approaches allow agents to maintain autonomy and can be used if communication is not possible between agents.

Recall that our quest in this thesis was to ensure coordination while guaranteeing autonomy to the maximum possible extent. Therefore, we are interested in developing efficient (polynomial time) procedures that ensure coordination when interaction is not required. The motivation to study such systems comes from the fact that in several real world situations such as multi-modal logistics and military reconnaissance, coordination has to be often achieved when agents cannot share their local

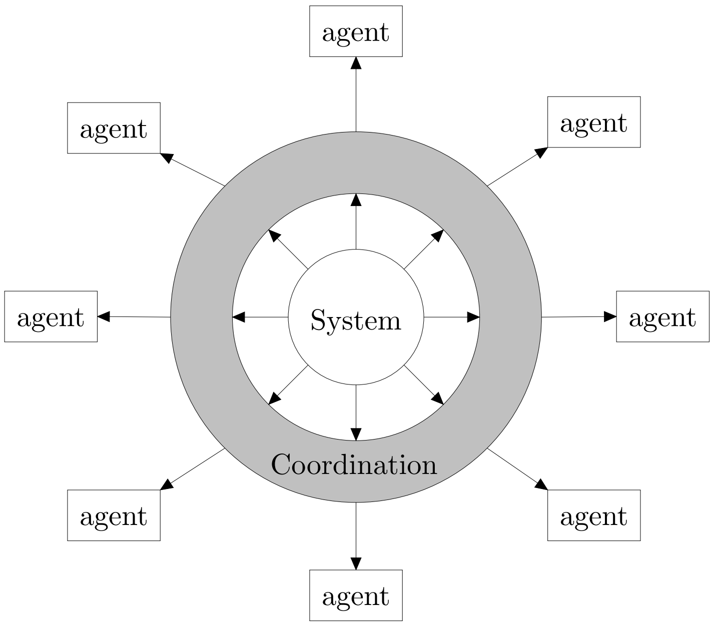


Figure 2.3: Coordination initiated by the problem owner. Coordination layer ensures decomposition.

plans (schedules). In fact, we think that in large multi-agent systems where agents are selfish, it is more practical to expect that agents are unwilling or unable to share their plans with other agents of the system. Therefore in this thesis we adopt the stand that coordinating a system, is similar to decomposing it such that solutions to all parts can be eventually combined to derive a solution to the global problem.

The rest of the chapter is organised in two parts. The first part discusses *plan coordination mechanisms*, and the second part discusses *schedule coordination mechanisms*. When discussing plan coordination, we start with mechanisms where agents interact cooperatively, then discuss mechanisms where they interact competitively and finally we discuss mechanisms where they do not interact at all. Because our interest is to design an efficient mechanism for coordination for non-interacting agents, we also specify the framework and the formal problem description as a continuation of the discussion on decomposition based approaches.

In the second part, we discuss schedule coordination methods. Here again, we start by discussing schedule coordination approaches that use communication. However, we next directly jump into discussing our framework for schedule coordination methods. There are two causes for this *jump*. Firstly, the framework we use for schedule coordination is an enriched version of the framework used for plan coordination. Therefore, we believe that we can directly discuss this enriched version without much ado. Secondly, we discuss the *Temporal decoupling problem*, which is based on the *Simple Temporal Network* framework, in our discussion of decoupling techniques. The simple temporal network framework, as we shall show later on in Chapter 4, can also describe a class of schedule coordination problem instances. Therefore, we believe that it would be better to first discuss our framework so that the differences between the two frameworks are evident.

We end this chapter with a summary of all the methods used for both plan and schedule coordination methods.

2.1 Plan coordination

Essentially the goal of the problem owner in a plan coordination problem is to come up with a feasible plan for the entire set of tasks. Therefore, one can view it as a planning problem. The only difference between traditional planning problems and plan coordination problems is that, the problem owner in coordination problems does not bother about the actual planning, but is concerned only with ensuring that plans found by agents for their set of tasks can be coordinated to achieve a feasible global plan. That is, for coordination, we are only interested in the partial order in which tasks occur in the plans of agents. Therefore, we abstract from the concrete plans of agents, only taking into account the precedence ordering of tasks. As already pointed out, local plans of agents when combined together may conflict with each other. Thus, the role of a plan coordination mechanism is to oversee that agent plans,

1. do not conflict with each other and

2. they can be combined to achieve a feasible global plan.

Further, it is desirable for agents that the coordination mechanism offers greater amount of autonomy. Therefore, agents would desire that the coordination mechanism imposes the fewest possible constraints on their planning.

Given this background let us now proceed to surveying the various methods available in literature to achieve plan coordination.

The relevance of plan coordination in multi-agent systems has been well studied in literature (cf. [Alami *et al.* 1998, Caridi and Sianesi 2000, Desjardins *et al.* 1999, Cox and Durfee 2003, Cox and Durfee 2005, Cox *et al.* 2005]). As pointed out earlier, plan coordination mechanisms can achieve their objective either by allowing interaction between agents or by decomposing the problem. Further, when allowed, interactions could be between cooperative agents or competitive agents. Techniques such as *plan merging* (cf. [von Martial 1992, Ephrati and Rosenschein 1994, Weerdt *et al.* 2003, Weerdt 2003, Alami *et al.* 1998, Foulser *et al.* 1992]) and *plan repair* [der Krogt and Weerdt 2005a, Arangú *et al.* 2008, Alami *et al.* 2002], which we will discuss later in this chapter, require agents to interact cooperatively between each other, whereas techniques which employ market mechanisms (cf. [der Krogt and Weerdt 2005b, Wangermann and Stengel 1998, Gerkey and Mataric 2002, Gerkey and Mataric 2003a, Gerkey and Mataric 2003b]) require agents to interact competitively (Section 2.1.1). Techniques that involve decomposition do not require agents to interact at all (Section 2.1.2).

In discussing each of the plan coordination mechanisms we will use the following simple instance of a plan coordination problem as a running example.

Example 2.1. Consider the situation in Figure 2.4(a). Two agents, A_1 and A_2 have to prepare individual plans for their sets of tasks $\{t_1, t_2\}$ and $\{t_3, t_4\}$ respectively. The catch, however, is that task t_3 can only be started after task t_1 and task t_2 can only be started after task t_4.

Suppose A_1 plans to perform t_2 first and task t_1 later. Similarly A_2 plans to perform task t_3 first and then task t_4. In combination their plans create an infeasibility. Task t_1 cannot be started until task t_2 is finished and task t_2 cannot be started until task t_4 is finished, which in turn cannot be started until task t_3 is finished. However, task t_3 also cannot be started since task t_1 is not complete. Thus, this combination of plans leads to a deadlock situation making the global plan infeasible. This deadlock is indicated by the existence of the directed cycle $(t_1, t_3, t_4, t_2, t_1)$ in

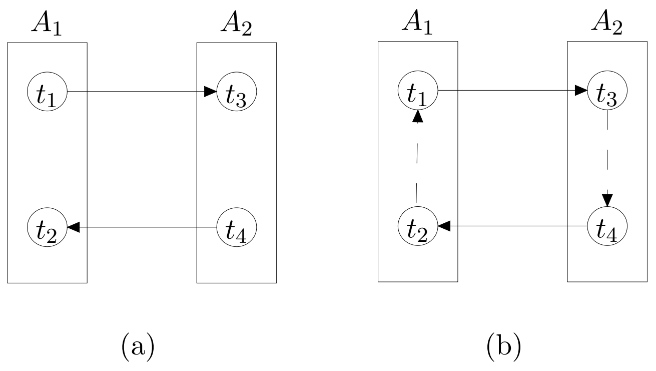


Figure 2.4: Running example. The solid arrow from t_1 to t_3 and t_4 to t_2 indicate the precedence constraints $t_1 \prec t_3$ and $t_4 \prec t_2$ respectively. The dotted arrows between tasks of individual agents represent the local plans of the agents.

Figure 2.4(b). We want plan coordination mechanisms to ensure that such cycles, responsible for deadlocks, do not occur in a combined plan. ■

Referring to the example above, suppose agents A_1 and A_2 were allowed to interact, either directly or through a central authority, then clearly the deadlock could be detected and resolved. Deadlock detection could be accomplished by simply verifying if the global plan has a directed cycle. Resolution of deadlocks when agents are cooperative, would require one or both agents to change their plans so as to avoid the deadlock. When they are non-cooperative, incentives have to be designed so that, agents are willing to accept changes to their local plans. In the following text, we shall deal with both these scenarios in greater detail.

2.1.1 Achieving coordination through interaction

As mentioned earlier the coordination layer functions as a communication layer when interactions are the preferred way to achieve coordination. Typically, in this approach agents initiate coordination. That is, agents first prepare their local plans and then try to coordinate those plans (cf. [Alami *et al.* 1998, Ephrati and Rosenschein 1993, Weerdt *et al.* 2003, Weerdt 2003, Weiss 2000]). Sometimes, agents prepare partial plans and communicate with other agents so that positive interactions can be utilised and negative interactions can be avoided (cf. [der Krogt and Weerdt 2005b, der Krogt and Weerdt 2005a, Fox *et al.* 2006]). In either case, agents can be cooperative and accept the changes suggested or can be selfish and require some incentive to do so.

Plan coordination through cooperative interaction Several authors have propounded cooperation as an effective coordination tool (cf. [Alami *et al.* 1998, Ephrati and Rosenschein 1993, Ephrati and Rosenschein 1994, Desjardins *et al.* 1999, Jennings 1993, Malone and Crowston 1994]). Coordination through cooperation can be achieved either during the process of merging local plans (partial plans) or after.

Plan merging techniques are used when plans are coordinated during the process of combining local plans into a global plan. Plan merging is achieved by merging *operators*. An operator can be conceived as a primitive task or activity that has pre-conditions and effects. Plan merging is typically carried out to ensure that redundant operations can be avoided [Foulser *et al.* 1992]. Examples of plan merging approaches are abundantly found in literature (cf. [von Martial 1992, Ephrati and Rosenschein 1994, Weerdt *et al.* 2003, Weerdt 2003, Alami *et al.* 1998, Foulser *et al.* 1992]).

Plan merging has also been used to ensure coordination of different aspects of a plan (cf. [Weerdt *et al.* 2003, Weerdt 2003, Qutub *et al.* 1997]). Some researchers such as [Weerdt *et al.* 2003] use plan merging to coordinate the efficient usage of resources. Others such as [Alami *et al.* 1994] use plan merging to ensure that in a multi-robot situation there is better reactivity. The idea of plan merging has also been extended in [Qutub *et al.* 1997] to ensure that deadlocks are resolved within a plan merging paradigm. This last objective (deadlock avoidance) is of greater interest since it is closer to our requirements.

The general idea of deadlock avoidance through plan merging is to first ask agents to construct a (partial) plan for a set of tasks. If a deadlock situation is detected, that is, a sequence of pre-conditions and effects is found to be cyclic, then one of the agents participating in the deadlock assumes the role of a *coordinator*. It then proposes a solution for the deadlock according to which agents change their plans. In this scheme, since authors assume that agents are cooperative there is no question of them rejecting the solution as long as it does not cause any further deadlocks. Thus, in some sense the coordination constraint or the solution is imposed on the remaining agents.

Example 2.2. Referring to our running example — we would first let both agents prepare any local plans they desire. Suppose they make the plans as in Figure 2.4(b). One of the agents, suppose A_1, assumes the role of the coordinator and detects the deadlock involving the four tasks. It can now propose a solution or a coordination constraint where the plans of agent A_2 are reversed. That is, task t_4 is performed before task t_3. Since this modification does not cause any further deadlocks A_2 changes its plan accordingly and a coordinated global plan is formed. ■

Suppose that a local plan cannot be merged (because it leads to an uncoordinated situation). Furthermore, suppose that no agent is able to come up with a solution to resolve the deadlock. In such a case, plan merging fails and agents might be required to completely replan. Some researchers (cf. [Fox *et al.* 2006]) have argued that, it is often more beneficial to *repair* an existing plan to resolve conflicts than replanning.

This means that the plan that existed before the deadlock was detected, could be revised so that the deadlock situation is avoided. This idea of revising an existing plan (through addition or deletion of constraints) is termed *plan repair.* Plan repair can be employed either during planning, or after all the local plans have been constructed. In plan repair, local (partial) plans are communicated to a central *repairer*, which then determines the set of revisions required to make the integrated joint plan feasible. These changes are communicated back to the agents who incorporate the changes into their local plans. This process is repeated until a complete and valid global plan is computed (cf. [der Krogt and Weerdt 2005a, Arangú *et al.* 2008, Alami *et al.* 2002]).

Example 2.3. With respect to our running example, suppose again that agents make the plans as in Figure 2.4(b). The agents then communicate their plans to a central authority. The central authority could find out that combining task t_1 and t_3 does not create a cycle. However, as soon as they include tasks t_4 and t_2 a deadlock is created. Therefore, it revises the original plan. In the revised plann, it could enforce that task t_1 precedes t_2 to avoid conflicts. ■

Notice however that both plan repair and merging techniques, appeal for cooperation to resolve conflicts. This need for cooperation, violates the planning autonomy of agents. Agents are now *forced* to accept changes to their local plans. Further, information privacy is not guaranteed: agents have to share information regarding their plans. Hence if agents are strictly autonomous and conscious of their information privacy needs, then both techniques may not be suitable in the current setup.

Plan coordination through competitive interaction We mentioned earlier that, both plan repair and plan merging techniques require agents to be cooperative. However, even when agents are non-cooperative, these techniques can be made to work. Only now, each agent has to be suitably compensated for its cooperation. From the solutions described for our running example earlier, it is evident that to ensure coordination, it is enough that one agent changes its plans. Now if both agents are adamant and not willing to change their plans, then the result is an uncoordinated situation. However, if agents agree to compensate other agents for changing their plans, then it is possible that a solution might emerge. Several researchers have explored this idea and developed market protocols that facilitate plan coordination (cf. [der Krogt and Weerdt 2005b, Wangermann and Stengel 1998, Gerkey and Mataric 2002, Gerkey and Mataric 2003a, Gerkey and Mataric 2003b]).

Some researchers (cf. [der Krogt and Weerdt 2005b, Gerkey and Mataric 2002, Gerkey and Mataric 2003a, Gerkey and Mataric 2003b]) use auctions to incentivise agents to cooperate. Auction based mechanisms, typically involve the auctioning of tasks themselves. In (cf. [Gerkey and Mataric 2002, Gerkey and Mataric 2003a, Gerkey and Mataric 2003b]), tasks are auctioned off in an effort to make efficient usage of resources and as a result constraints that ensure coordination are imposed on the local plans of agents. In the plan coordination mechanism of [der Krogt and Weerdt 2005b], agents first concurrently plan for a single goal, after which they take part in an auction (if there is any) to exchange goals (tasks) and subgoals (tasks). Then, they apply a plan repair technique to add another goal to their plan. They then take part in an auction again. They continue to alternatively perform these steps of adapting a plan using plan repair and taking part in an auction, until a complete and valid global plan is computed. Another approach to avoid deadlocks using auctions can be imagined. This approach is illustrated in the example below.

Example 2.4. In our running example suppose that each agent gets some amount x to perform its activities. Further suppose that the problem owner requests bids from the agents quoting their price for taking an additional constraint. Agent A_1 might quote a price of y_1 and agent A_2 might quote a price of y_2 for taking an additional constraint. Based on the price they quote, the problem owner might decide to either impose $t_1 \prec t_2$ on agent A_1 or $t_4 \prec t_3$ on agent A_2. Thus, if agent A_1 wins the auction, then it gets a profit of $x + y_1$ and agent A_2 gets a profit of x. ■

Auctions are unidirectional, where sellers have no control over the compensation that might be offered for their goods (or constraints in this case). Negotiations offer a richer model of interactions. Here both parties involved can have a say in the final outcome.

Example 2.5. In our example, suppose again as earlier that the whole operation results in a profit of x. Further suppose that each agent gets paid an amount of $\frac{x}{2}$. Suppose agent A_2 announces that if agent A_1 can take an additional constraint, then it would pay an additional amount of $\frac{x}{8}$ from its share of profits. Note that if both agents agree to take additional constraints, then their share of profits remains at $\frac{x}{2}$. Whereas if both of them disagree to take additional constraints and their plans contradict, then neither of them will get any profit. However, if agent A_1 is agreeable to such a compensation then by paying an additional amount of $\frac{x}{8}$ and adding the constraint that t_1 precedes task t_2, agent A_2 could ensure coordination. This would imply that A_1 would now need to modify its plan to accommodate the constraints imposed by A_2. On the other hand, if agent A_1 is not agreeable to the compensation, it might reject the offer and also inform the same to agent A_2. Agent A_2 could then continue looking for an offer that is acceptable for agent A_1. ■

A technique called the *principled negotiation* technique was proposed in [Wangermann and Stengel 1998] for coordinating agents. In this technique, agents repeatedly search for options (alternative global plans) that provide mutual gain. When an agent finds a plan that suits its own interests best, it proposes the new global plan. The other agents evaluate it and either accept it or reject it. If all agents accept to it, the global plan is then implemented. However, if an agent rejects it, it sends a message to the proposing agent regarding reasons for rejection. The proposing agent, then uses this information to improve its search. If the negotiation succeeds then coordination in such a set up is ensured, because if the proposed global plan has conflicts, then other agents simply reject it. Note that principled negotiation is not a technique that involves competition in the strict sense. However, it is also not too different from traditional negotiation since the only difference here seems to be in the contents of an agent's proposal. While in traditional negotiation agents simply propose a share of profits and a plan for their set of tasks, in this method the proposing agent is obliged to develop a global plan as well as a share of profits. Thus, while the profits sharing part could be competitive, the part where an agent proposes a global plan seems to be a cooperative activity.

Example 2.6. If principled negotiation was used on our running example, then one of the agents say A_1 would propose a global plan as well as a share of profits. It could propose a global plan where,

- agent A_2 is forced to take the constraint $t_4 \prec t_3$ since, agent A_1 plans to perform t_1 after task t_2 and,
- A_1 desires a profit of $\frac{x}{2}$.

A_2 could review this proposal and either accept or reject it. If it accepts the proposal, the global plan proposed is confirmed. ■

Auctions and negotiations are effective methods because all agents can be sure that the outcome of the process is the best they could hope for. Further, information privacy concerns are well addressed in market mechanisms. However, one of the possibilities is that the auction as well as the negotiation mechanism may end in a disagreement. In such a case, one cannot guarantee that coordination is ensured.

Our interest in this thesis is to *ensure* that autonomous plans/schedules of agents can be *always* coordinated. This implies that we are forced to choose methods which come with a guarantee that there will be a coordinated solution. Thus, auction based or negotiation based methods are not suitable for our purpose.

So far we have seen interaction based methods to ensure that a global plan is coordinated. In these methods, we saw that if agents are cooperative, then coordination could be ensured, but planning autonomy of agents would be compromised. On the other hand if we allowed agents to be non-cooperative, then we could ensure planning autonomy, but we cannot ensure that a feasible global solution is always achieved. However, we are interested in methods that guarantee both the *(i)* ability to autonomously plan and *(ii)* that these local plans can be always combined into a coordinated global plan. In the next section, we shall study methods where interaction between agents is not required, in the hope that they might assist us in achieving both objectives mentioned above.

2.1.2 Plan coordination without interaction

In many practical situations it is against an agent's interest to share any information regarding its plan, possibly because it might handover competitive advantage to the agent's rivals and competitors (cf. [der Krogt and Weerdt 2005b]). But, the need for coordination is intact. This section looks at mechanisms that ensure coordination, but do not require agents to interact.

So far in the previous approaches, inter-agent dependencies were managed through interaction between agents. Suppose that the internal constraints of an agent are more restrictive than the ones imposed by inter-agent dependencies, then any valid local solution also satisfies inter-agent dependencies. That is, inter-agent dependencies would then become redundant.

Therefore, the answer to avoiding conflicts, is to design a set of *coordination constraints* that allow us to *decompose* the global problem into subproblems, such that every solution to each of the subproblems can be merged into a global solution. More specifically, we need a decomposition of the *global* problem such that all valid *local* solutions can be always combined into a valid global solution.

Decompositions can be brought about in at least two ways :

- **Problem level:** Here the constraints are such that every instance of the problem can be decomposed due to the same set of constraints.
- **Instance level:** Here coordination constraints are designed to decompose each instance of the problem separately. However, the mechanism adopted to design these constraints is the same.

Suppose now that a large class of problem instances become uncoordinated for the same reason. For instance, in a traffic situation, a large number of vehicular accidents happen between vehicles going in opposite directions. In such a case, a convention or norm for instance, to drive on the left (right) side of the road, can be stipulated to avoid uncoordinated situations. Such laws, which can be used to coordinate a large number of problem instances are known as *social laws.* Social laws, offer planning autonomy to agents, but add restrictions that disallow a subset of these plans and hence ensure coordination. Let us next study social laws is greater detail.

Social Laws *Social laws* are problem level coordination constraints used to ensure decomposition. In other words, social laws (cf. [Moses and Tennenholtz 1992, Shoham and Tennenholtz 1995]) assure us that for each problem instance, all agent plans that do not violate these laws will always be mergeable into a global plan. A popular example for social laws is the set of traffic rules. For a given country, traffic rules are designed to ensure that in *any traffic situation* there will be no conflicts as long as drivers adhere to these rules.

Example 2.7. For our running example, if we had a social law saying that agents must not be idle, then A_1 would be forced to perform t_1 first and similarly agent A_2 would be required to perform t_4 first.

Clearly this would solve the example problem instance. Not only this instance, but it would solve all problem instances where all tasks have either predecessors or successors but never both. It is easy to see that in any directed cycle of the global plan, all tasks succeed and precede each other and hence result in an infeasibility. The rule ensures that predecessor-free tasks never have to succeed successor-free tasks. Therefore deadlocks (cycles) can never be formed. ■

Social laws can be unduly restrictive. Consider the following example as an illustration.

Example 2.8. Suppose we have the situation shown in Figure 2.5. Here, three agents have been allocated with 6 tasks, the precedence relationships between tasks are as shown. In this case, all tasks are either predecessor free or successor free. Applying the same rule as earlier implies that tasks t_1 and t_3 would be preferred over tasks t_2, t_4 respectively. Indeed, this does result in an acyclic global plan. However, observe that in this instance, no coordination constraints are actually necessary. In

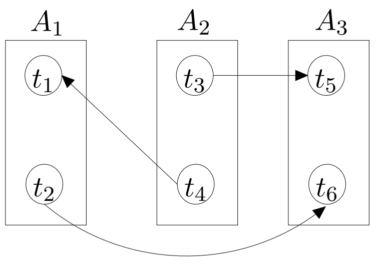


Figure 2.5: Social laws can be overly restrictive in certain cases. It is impossible to create a directed cycle in the above scenario.

other words, the instance is already coordinated. In such cases, rules as above are unnecessarily restrictive to the agents. ■

As seen in the example of Figure 2.5 social laws can be overly restrictive. Furthermore, even when the application area is fixed, plans could become uncoordinated in a large variety of ways. Thus, it is not realistic to expect a single set of social laws to be able to achieve coordination without being overly restrictive. Therefore, we turn to the next strategy — instance level decomposition — to achieve our objective.

Coordination protocols A social law can be seen as a solution for each instance of a coordination problem. As pointed out earlier, designing social laws can be unrealistic in several cases. Therefore in such situations, one can resort to developing *social law like* solutions that are tailored for each instance of a coordination problem. Such laws are termed *coordination protocols* and such coordination techniques are known as *instance level decomposition techniques.* Several authors (cf. [Jennings 1996, Gerson 1976, Valk 2005]) have designed coordination protocols for planning. We are specifically inspired by Valk's coordination protocol (cf. [Valk 2005]) for two reasons:

1. His protocol guarantees that a solution will always be found and

2. It also guarantees that the solution so found will always have a minimum number of coordination constraints.

The second point is particularly impressive, since we can now guarantee that agents will be minimally restricted in constructing their plans.

Unfortunately though, Valk (cf. [Valk 2005]) showed that this problem of finding a minimum set of coordination (decomposition) constraints, is a Σ_2^p- complete problem. The problem derives its complexity from two sources:

- it is co-NP-complete to verify if an arbitrary set of constraints is *sufficient* to decompose the plan coordination problem and
- it remains NP-hard to find a *minimum* set of constraints even if we could find an arbitrary set of constraints that decompose the plan coordination problem in polynomial time.

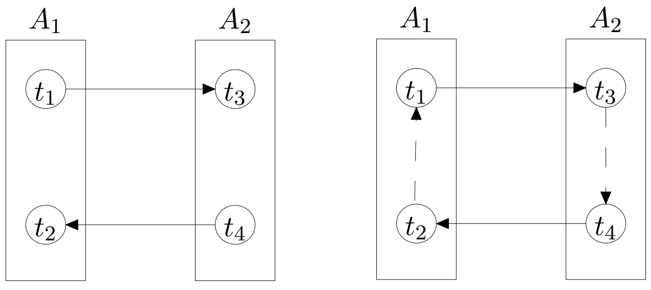
There is still hope. Valk himself points out that all plan coordination problem instances are not equally hard. In some cases, we can efficiently verify if a given solution indeed decomposes the original problem. Our attempt in the first part of the thesis is to identify such a class of efficiently verifiable problems. The motivation is that it might be possible to develop good approximation algorithms to solve the coordination problem for this class of instances. Further, it could also allow us to identify subclasses of this class of problem instances where solving the coordination problem can also be done efficiently.

Several researchers such as [Buzing *et al.* 2006, terMors and Witteveen 2005, terMors 2004, Steenhuisen *et al.* 2006] have adapted Valk's framework to describe plan coordination problems. Owing to its simplicity and also the close relevance of results obtained from Valk's framework to our work, we too adopt his framework for describing plan and schedule coordination problems in this thesis.

Let us now describe Valk's framework in detail and also state the coordination problem formally using his framework.

2.1.3 Framework and problem definition

In Valk's framework, problem instances consist of a set of tasks $T = \{t_1, \ldots, t_m\}$, allocated to a set $A = \{A_1, \ldots, A_n\}$ of n autonomous agents, according to a task allocation function $\phi : T \rightarrow A$. The set of tasks in T, are partially ordered by set of $\prec$ of *precedence* relations, where $t_i \prec t_j$ indicates that task t_i must be completed



(a) (b)

Figure 2.6: Task graph of our running example.

before task t_j can start. Thus, a *plan coordination instance* can be represented as a tuple $\Pi = \langle T, A, \prec, \phi \rangle$. We denote the set of tasks allocated to agent A_i by $T_i = \phi^{-1}(A_i)$.

Example 2.9. Our running example, can be described using Valk's framework as $\Pi = \langle T = \{t_1, t_2, t_3, t_4\}, A = \{A_1, A_2\}, \prec = \{t_1 \prec t_3; t_4 \prec t_2\}, \phi(t_1) = \phi(t_2) = A_1; \phi(t_3) = \phi(t_4) = A_2 \rangle$. ■

Note that ϕ^{-1} induces a partitioning $\{T_i = \phi^{-1}(A_i)\}_{i=1}^{n}$ of T. Likewise, the precedence relation $\prec_i$ is the precedence relation $\prec$ restricted to T_i. Frequently, we denote a plan coordination instance also as $\Pi = \langle \{T_i\}_{i=1}^{n}, \prec \rangle$. The only difference between the two representations is that, in the latter representation, partitions of the task set are explicitly mentioned in the latter, but the set of agents and the allocation function are implicit.

Plan coordination instances can be conveniently represented as directed graphs. Given a plan coordination problem $\Pi = \langle T, A, \prec, \phi \rangle$, we can draw a directed graph $G_\Pi = (T, E_\prec)$ corresponding to it, where the set of nodes of G_Π is the set T of tasks. An edge is added to $(t, t') \in E_\prec$ whenever $t \prec t'$. We refer to such a graph G_Π as the *task graph* of the given problem Π.

Recall that we are dealing with situations where agents wish to be autonomous and hence care only about the set of activities they are responsible for. In other words, agents need to be only presented with a subproblem of the global planning problem. This subproblem instance provided to an agent is termed as the *local planning instance*.

Definition 2.1. Local planning instance for agent A_i: Given a plan coordination problem $\Pi = \langle T, A, \prec, \phi \rangle$ and an agent $A_i \in A$, a local planning instance Π_i belonging to agent A_i is a tuple $\Pi_i = \langle T_i, \prec_i \rangle$, where $T_i \subseteq T$ is the set of tasks allocated to A_i and $\prec_i \subseteq \prec$ is the set of precedence constraints restricted to T_i. ■

An agent A_i can solve its local planning problem Π_i by making a plan for executing the set of tasks T_i, taking into account the partial order $\prec_i$. Every agent can use its own favourite planning tool to accomplish such a plan. What is important for coordination purposes, however, are not all the details of the plan, i.e., all the actions planned by an agent to complete its tasks. Instead, we only need to know in which partial order the tasks in T_i will be completed according to the *concrete plan* agent A_i will construct. That is, we only need to know the partial order imposed on T_i as a consequence of this concrete plan. Since $\prec_i$ is always a subset of this partial order, we consider the *local plan* of agent A_i for its local planning problem Π_i to be a *refinement* of $\prec_i$.

Definition 2.2. Local plan for agent A_i: Given a local planning instance $\Pi_i = \langle T_i, \prec_i \rangle$, a local plan ψ_i of A_i is a tuple $\psi_i = \langle T_i, \prec_i^* \rangle$ where $\prec_i^*$ is a partial ordering of T_i that refines $\prec_i$, i.e., $\prec_i \subseteq \prec_i^* \subset T_i \times T_i$. ■

Example 2.10. Consider agent A_1 from our running example. Its local planning instance can be described as $\Pi_1 = \langle \{t_1, t_2\}, \emptyset \rangle$. A possible local plan for A_1 could be $\psi_1 = \langle T_1, \{t_1 \prec_1 t_2\} \rangle$. ■

Each of the agents can only plan for its own set of tasks, but we need to *combine* these plans to derive a plan for the entire set of tasks. We call such a combination as a *global plan*. Formally, we define a *global plan* as follows:

Definition 2.3. Global plan: Given a plan coordination instance $\Pi = \langle \{T_i\}_{i=1}^n, \prec \rangle$ a global plan Ψ is a tuple $\Psi = \langle T, \prec^* \rangle$ where $\prec^*$ is a partial ordering of T that refines $\prec$. That is,

$$\prec \;\subseteq\; \prec^* \;\subset\; T \times T$$

■

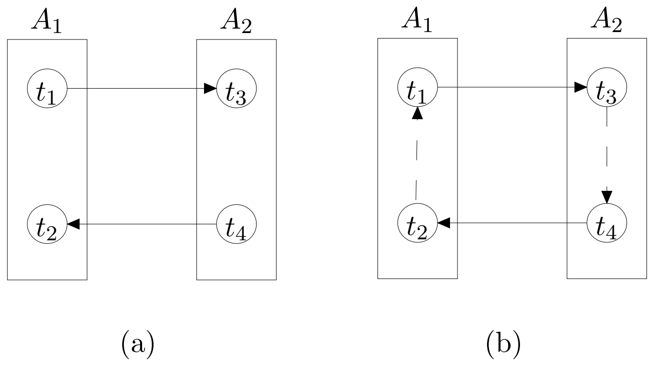


Figure 2.7: The solid arrow from t_1 to t_3 and t_4 to t_2 indicate the precedence constraints $t_1 \prec t_3$ and $t_4 \prec t_2$ respectively. The dotted arrows between tasks of individual agents represent the local plans of the agents.

Plans can be combined in different ways. As noted earlier, if agent autonomy was not a concern, then we could use plan merging and repair techniques to combine local plans. However, in our setting, we are *not allowed to modify local plans.* That is, each local plan should occur as an integral part of the total global plan. Therefore, a global plan in our case is always the result of a union of local plans and should satisfy all given constraints. In other words, the precedence order $\prec^*$ in a global plan $\Psi = (T, \prec^*)$ should always satisfy

$$\prec_1^* \cup \ldots \cup \prec_n^* \cup \prec \subseteq \prec^* \tag{2.1}$$

Unfortunately, local plans cannot always be combined to obtain a global plan. This is because, $\prec_1^* \cup \ldots \cup \prec_n^* \cup \prec$ might be a cyclic relation inducing $\prec^*$ to be cyclic. As an illustration, consider the following example.

Example 2.11. Consider the situation in Figure 2.7. The plan coordination instance corresponding to the figure can be described as $\Pi = \langle \{\{t_1, t_2\}, \{t_3, t_4\}\}\{t_1 \prec t_3; t_4 \prec t_2\}\rangle$. The local planning instance of agent A_1 denoted $\Pi_1 = \langle \{t_1, t_2\}, \emptyset \rangle$ and that of agent A_2 denoted $\Pi_2 = \langle \{t_3, t_4\}, \{\}\rangle$. The local plans constructed are depicted in Figure 2.7 and specified by $\Psi_1 = \langle \{t_1, t_2\}, \{t_2 \prec_1^* t_1\}\rangle$ and

$\Psi_2 = \langle\{t_3, t_4\}, \{t_3 \prec_2^* t_4\}\rangle$. Combining the local plans and the inter-agent constraints we derive $\{t_2 \prec^* t_1, t_3 \prec^* t_4, t_1 \prec^* t_3, t_4 \prec^* t_2\}$. Clearly, this results in a cyclic dependency, involving all tasks. Therefore, a global plan cannot be found from these local plans. Notice though that the local plans are acyclic. ■

Fortunately, it is possible to avoid such cycles. It has been shown by Valk [Valk 2005] and others [terMors 2004] that in general, by adding a *suitable set* of constraints to a given coordination instance *it is always possible* to guarantee the existence of a global plan. As an illustration consider the following example.

Example 2.12. Suppose now that we have a plan coordination instance $\Pi = \langle T, A, \prec, \phi\rangle$ that is similar to the instance in Figure 2.7, but contains an additional constraint $t_1 \prec t_2$. Now, both the plans of A_2 i.e., $t_3 \prec_2^* t_4$ and $t_4 \prec_2^* t_3$ result in a partial order when combined with $t_1 \prec t_2$. That is, both $\{t_1 \prec t_2, t_3 \prec_2 t_4\}$ and $\{t_1 \prec t_2, t_2 \prec_2 t_3\}$ are partial orders on T. Notice that agent A_1 is prohibited from creating a local plan, $t_2 \prec_1^* t_1$. Clearly therefore, the constraint $t_1 \prec t_2$, avoids any cyclic relationship that can be formed in the union of local plans. ■

These additional constraints are called *coordination constraints* and the set of coordination constraints is known as a *coordination set.*

Definition 2.4. Coordination set: Given a plan coordination instance $\Pi = \langle T, A, \prec, \phi\rangle$, a coordination set $\Delta = \bigcup_{i=1}^n \Delta_i$ is a set of intra-agent (local) precedence constraints, such that every set of ${\psi_i}_{i=1}^n$ of feasible local plans always results in a feasible global plan for $\Pi_\Delta = \langle T, A, \prec \cup \Delta, \phi\rangle$. ■

Example 2.13. Referring back to Figure 2.7, we could have several possible sets of additional constraints. For instance, $\Delta' = \{t_1 \prec_1 t_2\}; \Delta'' = \{t_4 \prec_2 t_3\}; \Delta''' = \{t_1 \prec_1 t_2, t_4 \prec_2 t_3\}$. All the three sets of constraints are coordination sets. In Δ', the coordination constraint ensures that none of agent A_2's plans — $\langle T_2, t_3 \prec_2 t_4\rangle$ or $\langle T_2, t_4 \prec_2 t_3\rangle$ — can create a cycle. By similar reasoning in Δ'' the coordination constraint disallows a cycle in the global plan despite agent A_1 being allowed to choose any plan. Δ''' clearly results in an acyclic global plan but it rules out any choice for either agent in picking a suitable plan. ■

Clearly, it is better for the agents to find a coordination set which is as small as possible. Small coordination sets allow agents to construct a greater number of valid local plans. In other words, smaller coordination sets allow greater flexibility to the agents in their planning. However, finding the smallest coordination set is not always easy. In fact, Valk in [Valk 2005] shows that this problem of finding a minimum sized coordination set is intractable. This intractable problem (and solutions to it), termed the *plan coordination problem*, forms the core of this thesis.

Plan coordination problem The essence of the plan coordination problem is to find a minimal coordination set. However, this is a two step process. Firstly, we have to ensure that a given additional set of constraints is indeed a coordination set. Secondly, we have to ensure that there exists no other coordination set with lesser cardinality. Each of these steps has been shown to be intractable (cf. [Valk 2005]). The first problem to determine whether a given set of constraints is sufficient to ensure coordination is termed as the *Coordination verification problem (CVP)* (cf. [Valk 2005]). Formally, the *Coordination verification problem(CVP)* and can be defined as below.

Definition 2.5. Coordination verification problem (adapted from [Valk 2005]): Given a coordination instance $\Pi = \langle \{T_i\}_{i=1}^n, \prec \rangle$ and a set of constraints Δ, such that $\Delta \subseteq \bigcup_{i=1}^n \{T_i \times T_i\}$, is the set Δ a coordination set for Π? ■

One of our requirements is to ensure that agents have maximum possible autonomy. Therefore, it seems reasonable to concentrate on methods that result in a coordination set of minimum cardinality. This problem of determining a minimum set of coordination constraints is what we call as the *coordination problem (CP)* [Valk 2005]. Formally,

Definition 2.6. Coordination problem (adapted from [Valk 2005]): Given, a coordination instance $\Pi = \langle \{T_i\}_{i=1}^n, \prec \rangle$ and an arbitrary integer K does there exist a coordination set $\Delta \subseteq \bigcup_{i=1}^n \{T_i \times T_i\}$ such that $|\Delta| \leq K$? ■

Note that to determine a minimum set of coordination constraints, we also have to answer whether such a set of constraints is sufficient (otherwise we could always simply return an empty set). Valk (cf. [Valk 2005]) showed that the CVP is co-NP-complete. Based on the complexity of solving the CVP, he also showed that the decision version of the coordination problem was Σ_2^p - complete.

It is clear from the complexity of the coordination problem that any algorithm that hopes to find a minimal coordination set would in general be intractable. One could hope that there probably existed some approximation technique for solving them efficiently. Unfortunately however, Valk (cf. [Valk 2005]) shows that this problem is unlikely to be in APX. This implies that in general we cannot hope to find an approximation algorithm which guarantees that the cardinality of the coordination set derived is within a constant factor of the minimum cardinality coordination set. Therefore in our next subsection, we restrict our scope to subclasses of the plan coordination problem that can be solved, or at least approximated, efficiently.

2.1.4 Problem statement

As pointed out earlier, we hope to find a class of coordination instances where coordination problem can be solved easily. Extensive study of the complexity of the plan coordination problem using Valk's framework has been carried out already by various researchers (cf. [Valk 2005, terMors and Witteveen 2005, Steenhuisen *et al.* 2006, Steenhuisen *et al.* 2008, Buzing *et al.* 2006]). Different aspects that affect the complexity of this problem have been studied by them:

Number of tasks Suppose we restrict the number of tasks, then [Steenhuisen *et al.* 2006] proved that the CVP becomes polynomially solvable only if the number of tasks is less than 3 for each agent.

Number of agents Suppose instead of the tasks, we are able to fix the number of agents. In this case, [Steenhuisen *et al.* 2006] proved that the verification problem is polynomially solvable. As an easy consequence, the coordination problem is in NP for this case. However, this would imply that we would have to have different mechanisms for each fixed number of agents. This approach would be inconvenient since, in several situations such as multi-modal transport, the number of agents might change frequently and each time such a change happens we would require to run a different mechanism to solve the verification problem.

Task graph structure Researchers such as [terMors 2004, Valk 2005] have shown that, some classes of coordination instances are efficiently solvable. In particular, [terMors 2004] showed that if each agent has either at most one task with incoming arcs, or at most one task with outgoing arcs,[1] then the coordination problem for instances in this class is in NP. He further showed that the coordination problem was efficiently solvable if each agent had either a totally ordered set of incoming or a totally ordered set of outgoing arcs. More generally, Valk, in [Valk 2005], showed that if agents were free of the so called *locplan cycles* then coordination verification could be done efficiently.

Studying subclasses where either the number of tasks or the number of agents is restricted, does not yield any new, easier to solve, class of coordination instances. There already exists research (cf. [Steenhuisen *et al.* 2006]) which identifies the classes where CVP can be efficiently solved.

Restricting problem instances based on their task graph structure is a more hopeful ground for further research. Buoyed by the success of [terMors 2004], we hope that more general classes of coordination instances where CVP can be solved efficiently can be found through the study of task graph structures. In summary, our objective of finding easier to solve coordination problem instances can be stated as follows:

- Does there exist an efficiently identifiable class of plan coordination problems instances where CVP can be efficiently performed?
- If so, can we design an approximation algorithm for CP, which exploits the polynomial solvability of CVP?
- Finally, does a class of coordination problem instances exist within this class of efficiently verifiable problem instances, such that even CP can be solved polynomially for that class?

The first question is answered affirmatively in Section 3.1.1. We then design an algorithm — DP^* algorithm which exploits the polynomial solvability of CVP in Section 3.2. Finally, in Section 3.3 we show that there does exist a class of coordination problem instances which can be efficiently solved. Since we can find a positive answer to the last question, we have achieved what we initially set out for

[1]Given an agent A_i, incoming arcs result from precedence constraints $\{t_i \prec t_j | t_j \in T_i\}$ and $t_i \notin T_i$. Out going arcs result from precedence constraints $\{t_i \prec t_j | t_i \in T_i\}$ and $t_j \notin T_i$.

— to devise a way of efficiently guaranteeing plan coordination, although in a very restricted sense.

2.2 Schedule coordination

So far, we have only discussed coordination mechanisms for plans. However, recall that we set out to look for mechanisms that might involve temporal information as well. Therefore, we would also require to study existing literature to achieve schedule coordination. Temporal information in scheduling problems is presented in many forms:

- Durations of tasks (see [Roundy *et al.* 1991, Prabhu and Duffie 1994, Liu and Sycara 1995, Burke and Prosser 1991, Maturana and Norrie 1996, Agarwal *et al.* 1995, Babayan and He 2004, Wang *et al.* 2008]).
- Release times of tasks (see [Wang *et al.* 1997, Baptiste 1999, Baptiste *et al.* 2004]).
- Deadlines (see [Liu and Sycara 1995, Burke and Prosser 1991, Maturana and Norrie 1996]).

Additionally, scheduling problems also involve explicit resource constraints. Presence of temporal information expectedly changes the texture of the (plan) coordination problem. For instance:

1. Schedules typically require to specify the time instant at which a task will start getting processed.
2. A coordinated global plan might allow more than one task to be simultaneously processed. However, such a plan might be infeasible due to resource constraints.

The presence of temporal information also affects the system goal. In plan coordination, the system goal was to simply ensure that the global plan was conflict free or in other words *feasible.* In terms of global plan quality, that is the best one can offer, since no other information is available. On the other hand in schedules, the presence of temporal information allows us seek a more sophisticated system goal. Several system goals such as *minimizing tardiness* (see [Roundy *et al.* 1991, Prabhu and Duffie 1994]), *meeting deadlines* (see [Liu and Sycara 1995, Burke and Prosser 1991, Maturana and Norrie 1996]) and *minimising makespan* (see [Agarwal *et al.* 1995, Babayan and He 2004, Wang *et al.* 2008]) have been pursued as system goals for scheduling in literature.

In this thesis, we choose to evaluate global schedules based on their *makespan*. That is, we seek to ensure that a feasible global schedule is always attained and has a makespan which is as small as possible. Further, we hope to ensure that agent autonomy is never compromised in achieving such a global schedule. That is, we would like to ensure that *(i)* agents can independently construct local schedules and *(ii)* whatever individual schedules agents select, they can be merged into a feasible global schedule. We could achieve this in two ways:

1. Centrally develop a global schedule. Split the global schedule into individual schedules according to the tasks allocated to them. In this method, agents are forced to select the centrally developed solution.

2. Split the problem into individual subproblems in such a way that whatever the solution chosen by the agents, a global solution can be formed by merging individual solutions.

The first approach clearly violates agent autonomy by allowing no choice to the agents in the construction of a schedule for their tasks. The second approach however, allows agents to choose any suitable schedule for their tasks as long as it satisfies the constraints of the sub problem they are faced with. Our interest is to assure agents of autonomy and hence, we choose to adopt the second approach. Thus, in summary, we are seeking to design schedule coordination mechanisms which:

1. Guarantee that a global schedule is feasible.

2. Ensure that agents can independently design schedules for their set of tasks, while

3. ensuring that makespan of such a global schedule is minimum.

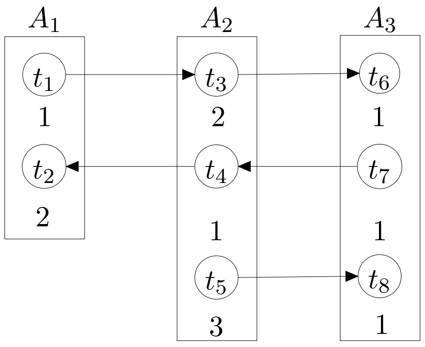


Figure 2.8: Running example to illustrate schedule coordination methods.

Before we develop methods to achieve our goals, we would like to present a brief survey of different schedule coordination techniques and also present a framework to describe schedule coordination problems. To assist us in understanding different scheduling techniques, we make use of the simple example shown in Figure 2.8.

Example 2.14. In this example, there are 8 tasks $t_1, \ldots, t_8$ allocated to agents A_1, A_2 and A_3. The precedence constraints between tasks are shown as directed arcs and durations of tasks are shown below each task in the figure. The objective of each agent in this example is to autonomously construct a schedule specifying the starting time for each of its tasks. The system objective or the coordination problem is to ensure that the schedules constructed by agents do not violate precedence constraints when merged whil ensuring a minimal makespan for the global schedule. ∎

We present our study of relevant literature in the following sequence. We start with a brief survey of interaction based methods and then discuss the schedule coordination framework. We end this section with a discussion of decomposition based methods for schedule coordination.

2.2.1 Schedule coordination through interaction

When interaction between agents is allowed, it usually also means that complete information privacy cannot be maintained. At least, the information regarding *relevant* parts of their schedules has to be exchanged. However, by allowing for interaction, we can draw from a large volume of research comprising of several interesting research areas. One such area is that of distributed scheduling.

The process of coordinating schedules can also be visualised as a process of developing a feasible global schedule in a distributed way. Viewed this way, research in distributed scheduling becomes relevant to our problem. Distributed scheduling can be described as the process where, smaller parts of a scheduling problem are solved by local decision makers and then through the use of some interaction mechanism a coordinated global solution is achieved [Toptal and Sabuncuoglu 2009].

Thus techniques used for distributed scheduling might benefit us in solving the schedule coordination problem, when interaction is allowed. Many interaction protocols have been used to ensure that an efficient coordinated schedule is obtained. These interaction protocols can be broadly grouped under 5 categories — cooperation (cf. [Kouiss *et al.* 1997, Decker and Lesser 1992]), iterative refinement (cf. [Liu and Sycara 1995]), bidding (cf. [Kutanoglu and Wu 1999]), iterative bidding (cf. [Solberg and Lin 1992]) and negotiation (cf. [Dewan and Joshi 2002]). In this section we review mechanisms that use these interaction protocols.

Schedule coordination through cooperation In cooperative distributed scheduling mechanisms, agents communicate with each other and determine a schedule that is feasible. It is important to note that while agents in cooperative systems do communicate, the intention is to achieve the global goal rather than serve individual goals. Abundant number of distributed cooperative scheduling systems have been presented in literature (see [Ramamritham *et al.* 1989, Durfee and Lesser 1991, Decker and Lesser 1992, Prabhu and Duffie 1994, Gou *et al.* 1998, Malewicz *et al.* 2006]). In general, when cooperation is used as the basis of developing a distributed schedule, agents (local decision makers) are expected to be able to make changes to their existing schedule. In that sense, the only difference between centralised systems and cooperative mechanisms is the fact that in a centralised mechanism, all decisions are made by a single entity whereas, in a

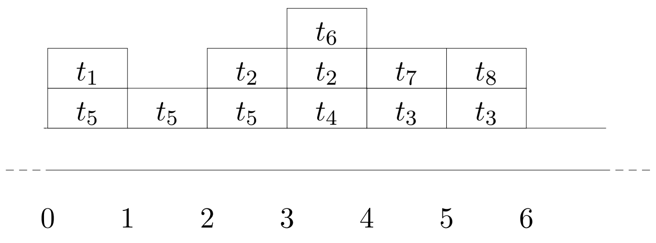


Figure 2.9: Initial schedules for tasks.

cooperative mechanism only the coordination decisions are taken collectively.

Example 2.15. Let us turn to our example to illustrate the general idea behind using cooperation for developing a distributed schedule. Suppose the agents have the schedule shown in Table 2.1 (the same schedule is pictorially depicted in Figure 2.9).

A_1		A_2		A_3	
Task	**Timepoint**	**Task**	**Timepoint**	**Task**	**Timepoint**
t_1	0	t_3	4	t_6	3
t_2	2	t_4	3	t_7	4
		t_5	0	t_8	5

Table 2.1: Initial schedules for tasks.

Clearly, this schedule is not feasible because task t_7 starts at time point 4 whereas task t_4 which succeeds task t_7 starts at time point 3. Similarly t_2 starts earlier than t_4 and t_6 starts earlier than t_3. Now because the agents are cooperative and hence willing to make changes to their schedules, they interact between themselves and could decide upon the schedule shown in Figure 2.10.

This schedule is clearly feasible as no precedence constraints are violated. This schedule has a makespan of 6 time units which is also the minimum makespan possible for this problem instance. ■

In this scheme of coordination, there are four critical steps :

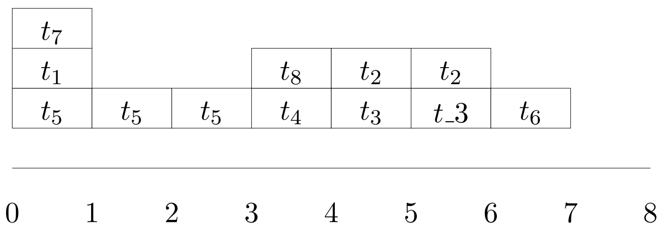


Figure 2.10: Coordinated schedule through cooperation.

1. Information exchange.
2. Detection of precedence constraint violation.
3. Determining the solution and
4. finally making the suggested changes to local schedules.

Clearly, the first and the last steps violate the idea of autonomy. If agents are autonomous, then it is unrealistic to expect them to share information. Suppose we were to design a workaround for this issue by assuming a setting where agents are only communicating with a problem owner who has all the information. Even in such a case, imposing a solution (or a part of the solution) on the agents would violate autonomy. Thus, while coordination can be guaranteed in this scheme, autonomy cannot be guaranteed.

Schedule coordination through iterative refinement Suppose agents submit only partial solutions to their local subproblems. Then many rounds of refinement might be required to arrive at a complete global schedule. Although this scheme does not eliminate the privacy and autonomy concerns, this method is still important since agents now can reveal only parts of their plans rather than the entire plan. In some sense, this reduces the amount of information sharing they require to do. Some researchers such as [Liu and Sycara 1995], have employed this idea of iteratively re-fining schedules to solve distributed job shop scheduling problems. In their strategy, one of the agents is chosen to be an *anchor* and this agent is allowed to optimize its schedule to satisfy its local efficiency criteria. The additional constraints, that result out of such optimization, are then communicated to the remaining agents, who determine if they can compute local schedules if the additional constraints are present. If they can, then the process is stopped and a global schedule is derived. Otherwise, they communicate the constraints that violate their local schedules to the anchor agent, which then modifies its current solution accordingly. The whole process is repeated until a global solution is found.

Example 2.16. Considering our running example again, the agents could start with the initial schedule being the one in Figure 2.9 and finally reach the schedule in Figure 2.10 as follows.

In the first iteration suppose agent A_3 is chosen as an anchor. Agent A_3 now optimises its local schedule according to its own criteria and specifies that task t_7 would be performed at time point 0, task t_8 would be performed at time point 1 and finally task t_6 would be performed at time point 2. This schedule imposes the following additional constraints — task t_3 should have finished by time point 2, task t_5 should be complete by time point 1 and task t_4 cannot start before time point 1. Clearly the first two constraints cannot be satisfied by agent A_2. Therefore A_2 now informs the same to agent A_3. As a result of this interaction, A_3 now modifies its schedule such that t_7 starts at time point 0, t_8 starts at time point 3 and t_6 starts at time point 6. This process, as is easy to see, results in the schedule in Figure 2.10.
■

This technique is reminiscent of the plan-repair techniques we discussed earlier. It expectedly suffers from the same drawbacks. As explained in the discussion on plan-repair techniques, this technique requires agents to constantly keep exchanging information so that a coordinated global schedule is arrived at. Further, if the anchor agent is unwilling to make changes into its schedule then it might result in a situation where no schedule is possible. Similarly, if the non-anchor agents refuse the additional constraints then too a global schedule might be impossible.

Now suppose that the anchor agent were to propose a compensation scheme to the other agents, then the regular agents might be more enthused to accept the suggested additional constraints. A variant of this idea where instead of an anchor agent a centralised job owner proposes and compensates for additional constraints has been extensively investigated. This idea forms the basis of the next two interaction protocols that we present.

Schedule coordination through auctions Auctions, have caught the imagination of several researchers as potential mechanisms to achieve solutions for a multitude of problems in a distributed way. Distributed scheduling has also benefited through the use of auctions. In auction based systems, jobs are announced through a manager agent and bids are requested for their processing. Agents prepare bids by considering their own capabilities and the best bid is chosen by the manager agent according to some pre-determined criteria. The Contract-Net protocol (see [Smith 1980]) is a seminal work that employs bidding. Several other systems which employ bidding (cf. [Parunak 1987,Shaw and Whinston 1988,Lima *et al.* 2006]) use the contract-net protocol as a basis for distributed scheduling. Some bidding mechanisms stop at a single iteration, whereas others allow agents to revise their bids after each iteration. For instance, in the system of [Dewan and Joshi 2002], each auction continues for several rounds until the winning bid meets certain criteria of the auctioneer. Such *iterative bidding* mechanisms are also found in (see [Dewan and Joshi 2002,Lau *et al.* 2005,Lau *et al.* 2006,Liu and Sycara 1995]). Notice that in such systems, the allocation of tasks is not fixed, it could vary based on the winner of the auction. Further, if nobody wins an auction, then such a task will not be scheduled. This implies that one cannot even guarantee that a feasible schedule will result out of the process. The benefit though is that auctions guarantee complete autonomy to the agents. Our quest however requires a guarantee that a global schedule will be achieved.

Example 2.17. Referring to our example from Figure 2.8, let us suppose that tasks t_1, t_3, t_6 are part of job J_1 tasks t_7, t_4, t_2 belong to job J_2 and tasks t_5, t_8 belong job J_3. As is the case with most auction based systems in literature there is no pre-determined allocation of tasks. Thus, we ignore the allocation in our running example. Further, suppose that all jobs have the same owner. The job owner announces its requirement through a manager agent and requests for bids. Each bid from the agents consist of two pieces of information — *(i)* the time point at which the job is performed and the *(ii)* cost of performing the job.

Suppose each bid is a 4-tuple $\langle a \in \mathscr{A}, t \in T, \sigma() : T \to R, p() : T \to R \rangle$ specifying the agent id, a task id, its proposed starting time and the cost for performing the task. Now suppose that the job owner receives bids from 4 different agents as shown in Table 2.2:

A_1's bids	$\langle A_1, t_1, 0, 10\rangle$	$\langle A_1, t_2, 4, 4\rangle$	$\langle A_1, t_6, 6, 5\rangle$
A_2's bids	$\langle A_2, t_3, 3, 15\rangle$	$\langle A_2, t_7, 0, 14\rangle$	$\langle A_2, t_4, 6, 4\rangle$
A_3's bids	$\langle A_3, t_5, 0, 5\rangle$	$\langle A_3, t_8, 5, 5\rangle$	
A_4's bids	$\langle A_4, t_1, 0, 50\rangle$	$\langle A_4, t_2, 1, 30\rangle$	$\langle A_4, t_4, 3, 20\rangle$

Table 2.2: Bids received by agents for performing the tasks in Figure 2.8.

Clearly, the only tasks whose allocation can be decided by the job owner are t_1, t_2, t_4, because, there is a single bid for the remaining tasks. The only valid schedule possible in this case would require that tasks t_2, t_4 are allocated to agent A_4. ■

Note that several kinds of bids can be used in such auctions. The bid structure that was shown in the example could have additional constraints that specify that all or none of an agent's bids must be selected. In such cases finding an allocation of tasks, such that a feasible schedule is possible, itself becomes a challenge.

Schedule coordination through negotiation Negotiation is another popular mechanism for distributed scheduling (cf. [Solberg and Lin 1992,Kim and B. C. Paulson 2003, Kaplansky and Meisels 2007, Wu *et al.* 2009b]). In general, negotiations are employed in distributed scheduling for two possible reasons — to construct a feasible global schedule and to improve the efficiency of a global schedule.

Example 2.18. Consider again our running example from Figure 2.8. If negotiations were used, we just have a set of tasks (with precedence constraints among them) and a set of agents who can perform those tasks. If the negotiation is about allocation of tasks then, the same solution as that of auctions could also be achieved through negotiations. Only now, we could have two different settings — one where the agents negotiate with job owner regarding the starting time of tasks and the cost of performing the tasks or a second setting where there is no central job owner and agents negotiate among themselves to decide who performs which task.

Suppose, they have the schedule of Figure 2.9 initially. They could arrive at the schedule of Figure 2.10 through negotiation about modifications to their schedules as well as the share of profits they receive as follows. Suppose that agent A_1 derives a profit of p_1, A_2 derives a profit of p_2 and agent A_3 derives a profit of p_3 by scheduling as in Figure 2.9. Agent A_2 starts the negotiation with A_3, with the proposal consisting of the list of changes it requires agent A_3 to make to its schedule — (start task t_7 at time 0; start task t_8 at time 3). It also proposes the additional profit p_x that A_2 is willing to pay agent A_3. Agent A_3 could reject the offer and in turn propose a new offer, where it could give the list of changes to the schedule of A_2 that it desires — (start t_4 at time point 3) as well as the share of profits p_y it is willing to give A_2. If agent A_2 agrees then the final schedule would be the same as that of Figure 2.10. ■

Negotiations and auctions offer the possibility that agents could retain their autonomy while arriving at a coordinated solution. They also offer the possibility of easily extending to settings where shared resources are present. Researchers [Wu *et al.* 2009b, Wu *et al.* 2009a], have tried to build negotiation models for solving schedule coordination problems in the recent past. As noted earlier the possibility of failure to reach an agreement greatly diminishes their applicability for solving schedule coordination problems.

Summarising the discussion so far, interaction between agents allows for many different approaches to achieve coordination. As we have seen so far, when interaction based methods allow for autonomy of decision making, as in the case of auctions and negotiations, they do not guarantee that a solution can be found. On the other hand when interaction based methods can guarantee that a solution is found, they compromise on autonomy of decision making as in the case of cooperation and iterative refinement. Neither compromise, the compromise on not achieving a solution or the compromise of autonomy, is acceptable to us. Therefore, we next study decomposition based techniques for schedule coordination.

As a slight departure from our approach towards plan coordination techniques, we first describe the framework we use to describe schedule coordination problems in the next subsection and then describe relevant literature. This change allows us to point out the differences between our framework and the *Simple Temporal Network* (*STN*) framework — a competing framework that could also be used to describe a class of schedule coordination problems. Further, the framework we employ to describe schedule coordination problems is a straightforward enrichment of Valk's framework we used to describe plan coordination problems and hence, we hope that the reader is already familiar with it and can easily grasp the changes necessitated by scheduling.

2.2.2 Framework and problem definition

Schedule coordination problem can be thought of as a constraint system where the starting point of each task is a temporal variable. The durations of tasks, precedence relationships between tasks and agent capacities act as constraints in developing a schedule. As with plans, if this constraint system can be *decomposed* into subproblems such that every solution for each of the subproblems can be merged, then we can be sure that schedules are coordinated. Notice that a *decomposition* that ensures schedule coordination does not require to actually generate a schedule. All it requires is to somehow design a set of constraints, such that every local schedule that honours these constraints is bound to lead to a makespan minimal global schedule. Thus, in reality we require to address a decomposition problem that can be stated as follows:

> *Given an instance of the schedule coordination problem, find a makespan minimising decomposition.*

Temporal information in scheduling problems we consider, pertains to the duration required to process each task.[2] This duration can be expressed as a function $l : T \to \mathbb{Z}$. As mentioned earlier, agents in a schedule coordination instance can have limits on the number of tasks they can perform simultaneously. These constraints usually occur because of the number of resources agents can use to accomplish their tasks. To capture such capacity constraints we define a function $c : A \to \mathbb{Z}^+ \cup \{\infty\}$ assigning to each agent A_i its concurrency bound $c(i)$. This concurrency bound is the upper bound on the number of tasks agent A_i is capable of performing simultaneously.

All tasks in the system need not be available for processing right in the beginning. They may become available at some arbitrary instant in time. This time at which a task is available for processing is called its *release time* and is encoded as $r : T \to \mathbb{Z}$. Finally, tasks also have *deadlines* $d : T \to \mathbb{Z}$ within which they need

[2]In our thesis we choose to only deal with integer durations.

to be processed. For the sake of simplicity we always assume that release times and deadlines are initialised as $r() = 0$ and $d() = \infty$. In summary, we say that $\Pi = \langle T, A, \prec, \phi, l(), c(), r(), d() \rangle$ is a *scheduling instance.*

As mentioned earlier, our idea is to use decomposition as a means to solve the schedule coordination problem. We aim to derive a set of decomposition constraints which will enable agents to *(i)* allow agents to develop local schedules autonomously, and *(ii)* ensure that the autonomously developed schedules can be always combined and finally, *(iii)* ensure that the global makespan is as small as possible.

Similar to planning, each agent is now given a local scheduling instance $\Pi_i = \langle T_i, \prec_i, l(i), c(i), r(), d() \rangle$ for which the agent prepares a local schedule σ_i.

Definition 2.7. Local schedule for agent A_i: Given a local scheduling instance $\Pi_i = \langle T_i, \prec_i, l(i), c(i), r(), d() \rangle$, a local schedule for Π_i is a function $\sigma_i : T_i \rightarrow \mathbb{Z}$ such that:

- For every $t \in Z^+$, $|\{t \in T_i \mid t \in [\sigma(t), \sigma(t) + l(t)]\}| \leq c(i)$, that is, the concurrency bounds for agent A_i should be respected.
- For every pair $t, t' \in T_i$, if $t \prec t'$, then $\sigma_i(t) + l(t) \leq \sigma_i(t')$

■

As earlier, each agent schedules its own tasks whereas we require a global schedule for all the tasks. Such a global schedule can be defined as follows.

Definition 2.8. Global schedule: Given a scheduling instance $\Pi = \langle T, A, \prec, \phi, l(), c(), r(), d() \rangle$, a global schedule for it is a function $\sigma : T \rightarrow \mathbb{Z}^+$ determining the starting time $\sigma(t)$ for each task $t \in T$ such that:

- For each task $t \in T, \sigma(t) = \sigma_i(t)$ if $t \in T_i$ and
- for every pair $t, t' \in T$, if $t \prec t'$, then $\sigma(t) + l(t) \leq \sigma(t')$.

■

As stated earlier, with schedule coordination, we seek more sophisticated system goals than simply ensuring a feasible global schedule. Thus, we also would like to achieve makespan minimality. Therefore, we prefer a global schedule σ such that $max_{t \in T}\{\sigma(t) + l(t)\} \leq max_{t \in T}\{\sigma'(t) + l(t)\}$, where $\sigma'(t)$ is any other feasible global schedule.

In summary, this schedule coordination problem termed as the *Coordinated autonomous scheduling (*CAS*)* problem can be stated as follows:

Definition 2.9. CAS problem: Given a scheduling instance $\Pi = \langle T, A, \prec, \phi, l(), c(), r(), d()\rangle$, the CAS problem is to guarantee that a feasible global schedule σ can be obtained by imposing upon each agent a set of additional constraints C_i, such that if C_i is specified for the scheduling instance $\Pi_i = \langle T_i, \prec_i, l(i), c(i), r(), d()\rangle$ of agent A_i, then all *locally feasible* schedules σ_i satisfying their *local* constraints C_i can be merged into a globally feasible schedule σ for the original total scheduling instance. ■

Once we can ensure that the global schedule is feasible, we further want to ensure that the makespan of each global schedule is minimum.

Consider the situation in our running example. We can represent it as the CAS problem shown in the illustration.

Example 2.19. The coordination instance in Figure 2.8 has eight tasks $t_1, \ldots, t_8$ and three agents A_1, A_2, A_3. $\phi(t_1), \phi(t_2) = A_1$ and $\phi(t_3), \phi(t_4), \phi(t_5) = A_2$ and the remaining tasks, t_6, t_7 and t_8 are allocated to A_3. The set of precedence constraints are — $t_1 \prec t_3, t_3 \prec t_6, t_7 \prec t_4, t_4 \prec t_2$ and $t_5 \prec t_8$. The durations of tasks are as follows. $l(t_1) = 1$, $l(t_2) = 2$, $l(t_3) = 2$, $l(t_4) = 1$, $l(t_5) = 3$ and the remaining tasks t_6, t_7, t_8 have a duration of 1. Each agent has a concurrency bound of 1. That is, $c(i) = 1\ \forall i \in \{1, 2, 3\}$. The release times are all equal to 0 and the deadline for each task is equal to ∞. Thus,

$$\begin{aligned}
\Pi =& \langle\{t_1, \ldots, t_8\}, \{t_1 \prec t_3, t_3 \prec t_6, t_7 \prec t_4, t_4 \prec t_2, t_5 \prec t_8\}, \\
& l(t_1) = l(t_4) = l(t_6) = l(t_7) = l(t_8) = 1,\ l(t_2) = l(t_3) = 2, l(t_5) = 3 \\
& c(1) = 1, c(2) = 1, c(3) = 1 \\
& r(t) = 0\ \forall t \in T \\
& d(t) = \infty\ \forall t \in T\rangle
\end{aligned}$$

From the description of the CAS problem instance it is easy to see that the local scheduling problems faced by the agents are the following:

$$\Pi_1 = \langle \{t_1, t_2\},\ \emptyset,\ l(t_1) = 1, l(t_2) = 2,\ c(1) = 1,\ r() = 0,\ d() = \infty \rangle$$
$$\Pi_2 = \langle \{t_3, t_4, t_5\},\ \emptyset,\ l(t_3) = 2,\ (t_4) = 1,\ l(t_5) = 3,\ c(1) = 1,\ r() = 0,\ d() = \infty \rangle$$
$$\Pi_3 = \langle \{t_6, t_7, t_8\},\ \emptyset,\ l(t_6) = 1,\ l(t_7) = 1,\ l(t_8) = 1,\ c(1) = 1,\ r() = 0,\ d() = \infty \rangle$$

The CAS problem now is to design a set of constraints C such that any schedule for Π_1, Π_2, Π_3 can be combined into a feasible solution for Π. That is, we have to design a set of constraints C such that:

$$\sigma(t) = \sigma_i(t), \forall\ t \in T_i$$

Further, we desire to ensure that $\max_{t \in T}\{\sigma(t) + l(t)\}$ is minimum. ■

Suppose, we just focus on decomposing the CAS problem and ignore the concurrency constraints. In that case, there exists another well studied problem called the *Temporal decoupling problem (TDP)*, which has an objective similar to the CAS problem. Furthermore, solution to a TDP is a decomposition of a given linear temporal constraint system. Most importantly, it has been shown that the TDP can be solved in polynomial time (see [Hunsberger 2002b]).

TDP is based on the *Simple temporal Network (STN)* framework which is a popular framework to describe temporal constraint systems. We hope that by studying *STN* and TDP, we can derive a method to solve CAS problems in polynomial time. Therefore, we next briefly describe the essentials of the *STN* framework and then proceed to describe Hunsberger's solution for TDP.

2.2.3 Schedule coordination through decomposition

Scheduling can also be seen as a planning mechanism using the notion of time. We have seen from our problem definition that the structure of the scheduling problem is quite simple. The precedence relations in the problem can be easily expressed as simple linear constraints involving differences between time points. Researchers proposed in [Dechter *et al.* 1991] that constraints among time points could be grouped together and expressed as a network of temporal constraints called the *Simple temporal network (STN)*. *STN*s can be used to represent

- temporal constraints among tasks of an agent,
- durations of tasks and
- precedence relationships among tasks.

The standard operations on *STN*s— propagation of constraints, determination of solutions and addition of extra constraints without losing solvability — all take polynomial time. In this section, we review the basic concepts in *STN* theory and also briefly describe the *Temporal decoupling problem.*

Definition 2.10. Simple temporal network (STN)(adopted from [Dechter *et al.* 1991]): A *STN* is a pair $\mathcal{S} = (\mathcal{T}, \mathcal{C})$, where $\mathcal{T}$ is a set $\{z, t_1, \ldots, t_N\}$ of time point variables and $\mathcal{C}$ is a finite set of binary constraints on those variables. ■

Each constraint in an *STN* has the form $t_i - t_j \leq \delta$, for some real number δ. The constraints in the set $\mathcal{C}$ are also called *explicit constraints.* The variable z represents an arbitrary fixed reference point on the time line. Note that in an *STN*, all constraints are difference constraints upper bounded by a constant. Hence, we have to express $t_j - t_i \geq l(t_i)$ as $t_i - t_j \leq -l(t_i)$.

Solutions to *STN*s associate a real value with each time point variable. That is,

Definition 2.11. Solution to an STN (adopted from [Dechter *et al.* 1991]): Given an *STN* $\mathcal{S} = (\mathcal{T}, \mathcal{C})$, a solution to it is a complete set of variable assignments

$$z = 0;\ t_i = w_1;\ t_2 = w_2; \ldots;\ t_N = w_N, \quad \text{where } w_i \in \mathbb{R},$$

that satisfies the constraints in $\mathcal{C}$. ■

*STN*s are usually represented using directed graphs termed as *Distance graphs.* Suppose we have the following constraints $t_j - t_i \leq 20$ and $t_i - t_j \leq -10$, the graph representation would be as in Figure 2.11.

Example 2.20. Suppose we had a deadline of 8 time units on the tasks in Example 2.19. It is easy to see that, this is the same as imposing a deadline of 8 time units to complete tasks t_6, t_2 and t_8. This deadline of 8 time units can be expressed as a constraint on the temporal distance between the time variables representing the starting times of tasks. That is, one could specify the following:

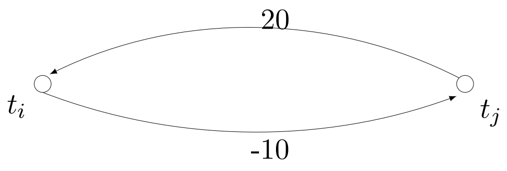


Figure 2.11: A simple distance graph.

$$\begin{array}{llll} t_3 - t_1 \geq 1; & t_3 - t_1 \leq 3; & t_6 - t_3 \geq 2; & t_6 - t_3 \leq 3 \\ t_4 - t_7 \geq 1; & t_4 - t_7 \leq 2; & t_2 - t_4 \geq 1; & t_2 - t_4 \leq 3 \\ t_8 - t_5 \geq 3; & t_8 - t_5 \leq 4 & & \\ t_6 - z \leq 7; & t_2 - z \leq 7; & t_8 - z \leq 7 & \end{array}$$

Once such a linear system of temporal constraints is developed, we only need to *fine tune* these constraints such that the temporal interval within which a task can be performed does not interfere with either the interval of its predecessor task(s) or its successor task(s). If we are able to do this, then we can just give the resulting intervals to each agent and ask them to schedule independently. Since the intervals do not overlap, any schedule they choose can be merged. Or in other words the system is decomposed.

The distance graph representation of the *STN* described above is shown in Figure 2.12. In Figure 2.12, the maximum possible temporal distance between two time point variables is indicated using a positive number and the minimum possible distance is indicated using a negative number. Thus, task t_4 cannot start within 1 time unit of t_7 nor can it start later than 2 time units of t_7. Notice that tasks t_6, t_2, t_8 cannot start later than 7 time units from the point the first task starts getting processed. This constraint ensures that all tasks are completed before 8 time units. ■

For the example in 2.20, a solution could be to start t_1, t_7, t_5 at time instant 0, t_3, t_4, at time instant 1, t_2 at time instant 2 and finally t_6, t_8 at time instant 3.

*STN*s which do not allow any solutions are called *inconsistent* and conversely *STN*s which allow at least a single solution are called *consistent* (see [Dechter *et al.* 1991]).

Based on the distance graph, a matrix which captures the shortest distance between every pair of time points in the distance graph can be computed. Such a

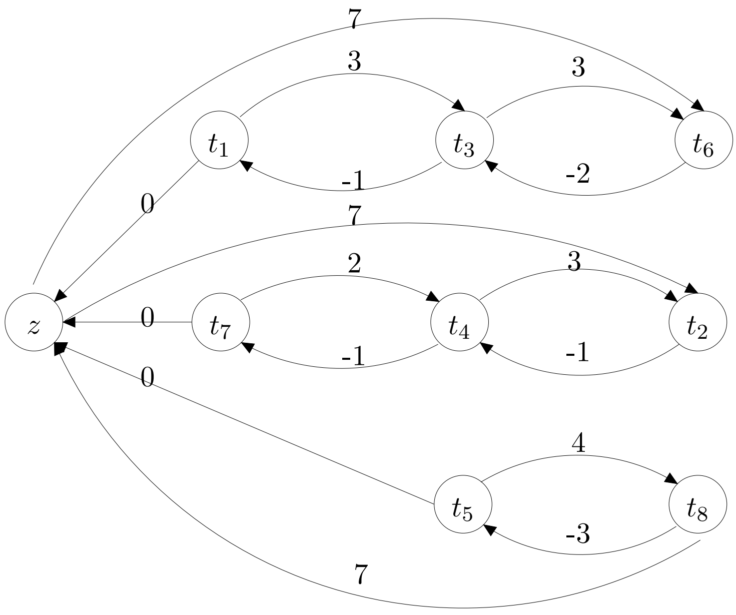


Figure 2.12: Converting the temporal constraints of Example 2.20 into an STN

matrix is termed a *Distance matrix* and is denoted by D. The entries in a distance matrix $D[t_i, t_j]$ denote the length of the shortest path from time point t_i to time point t_j. In other words, they represent the strongest implicit constraint between the two time point variables. Typically, distance matrices are computed using Floyd -Warshalls *all pairs shortest path* algorithm [Floyd 1962]. The time complexity of this computation is well known to be $O(\mathcal{T}^3)$.

Temporal decoupling problem (TDP) Notice from Example 2.20 that, if each agent were to independently develop schedules, agent A_1 might want to start processing task t_1 at time instant 4, whereas agent A_2 might want to start t_3 at time instant 2. Clearly, when combined these two schedules create an infeasibility. Therefore, we require some method to ensure that such infeasibility does not arise even when agents schedule independently. This problem is known as the *Temporal decoupling* *problem (TDP)* (see [Hunsberger 2002b]).

Informally, the TDP arises when the set of temporal variables in a given STN are partitioned into several STNs and it is required to ensure that any solution to the subSTNs can be merged into a solution of the whole STN. Fortunately, Hunsberger in [Hunsberger 2002b], has developed an efficient procedure to solve the TDP. Clearly, TDP cannot arise if there is no partitioning of time point variables. However, the z variable must exist in each partition so that a solution to the TDP can be found. The z variable functions as a common clock for all the partitions. Such partitions which include a common clock are called *z-partitions.* Formally,

Definition 2.12. z-partition (adopted from [Hunsberger 2002a]): $\mathcal{T}_1, \ldots, \mathcal{T}_n$ are said to z-partition $\mathcal{T}$ if

- $\bigcap_{i=1}^{n} \mathcal{T}_i = \{z\}$, i.e., the z time point variable is the only common reference point for all partitions and hence the only element common to all of the partitions.
- $\bigcup_{i=1}^{n} \mathcal{T}_i = \mathcal{T}$, i.e., every time point variable in $\mathcal{T}$ belongs to some partition $\mathcal{T}_i$.

■

We can now formally define the *Temporal decoupling problem (TDP)* as follows.

Definition 2.13. Temporal decoupling problem (TDP)(adopted from [Hunsberger 2002a]): A temporal decoupling of an STN $\mathcal{S} = (\mathcal{T}, \mathcal{C})$ is a set of STN $\{\mathcal{S}_1 = (\mathcal{T}_1, \mathcal{C}_1), \ldots, \mathcal{S}_n(\mathcal{T}_n, \mathcal{C}_n)\}$, such that:

- each of the STN in the set $\{\mathcal{S}_1 = (\mathcal{T}_1, \mathcal{C}_1), \ldots, \mathcal{S}_n(\mathcal{T}_n, \mathcal{C}_n)\}$ are consistent,
- $\mathcal{T}_1, \ldots, \mathcal{T}_n$, z-partition $\mathcal{T}$,
- merging any solutions for $\mathcal{S}_1, \ldots, \mathcal{S}_n$ yields a solution for $\mathcal{S}$.

■

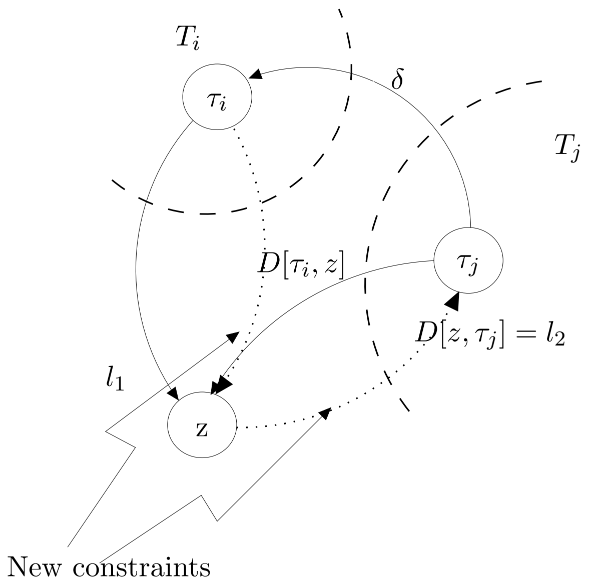


Figure 2.13: Temporal decoupling process.

Hunsberger's algorithm is based on the following idea. Suppose there exists a constraint $t_j - t_i \leq \delta$. Also suppose that $D[z, t_i] = l_1$ and $D[z, t_j] = l_2$ and that $l_1 + \delta > l_2$. By adding a constraint $z - t_i \leq (l_1 - l_2 - \delta)$, the consistency of the STN is not harmed. However, adding such a constraint would make the constraint $t_j - t_i \leq \delta$ redundant. In other words, we are *tightening* the constraints between the z and time point variables t_i, t_j by removing the slack $(l_1 - l_2 - \delta)$. If t_j and t_i belong to different partitions, then the inter-partition constraint between t_i and t_j is now redundant. Similarly, we could render each inter-partition constraint redundant by adding constraints as above. Or in other words, we could remove each of the inter-partition constraints and as a result decouple the STN. Figure 2.13 represents the process.

Example 2.21. Referring back to the Example in 2.20, one could apply Hunsberger's method and derive the set of *tightened* constraints shown in Table 2.3:

Clearly, now it does not matter whether agent A_2 schedules task t_5 at time point

$0 \leq t_1 - z;$	$t_1 - z \leq 1$
$4 \leq t_2 - z;$	$t_2 - z \leq 7$
$1 \leq t_3 - z;$	$t_3 - z \leq 2$
$1 \leq t_4 - z;$	$t_4 - z \leq 3$
$0 \leq t_5 - z;$	$t_5 - z \leq 2$
$4 \leq t_6 - z;$	$t_6 - z \leq 7$
$0 \leq t_7 - z;$	$t_7 - z \leq 1$
$5 \leq t_8 - z;$	$t_8 - z \leq 7$

Table 2.3: A possible set of constraints added by Hunsberger's method.

Task	Interval
t_1	[0,1]
t_2	[4,7]
t_3	[1,2]
t_4	[1,3]
t_5	[0,2]
t_6	[4,7]
t_7	[0,1]
t_8	[5,7]

Table 2.4: A solution to the original CAS problem instance.

0 or time point 1 or time point 2. In all three cases, agent A_3's schedule for task t_8 will not violate the precedence constraint between the two tasks. It is easy to verify that it is the same for all tasks. This means that we have achieved a decoupling.

Notice from the example that the solution derived from the TDP can be translated back into a solution for the CAS problem. That is, we could specify the time intervals for each task within which they need to be scheduled.

Now it is easy to see that, as long as agents schedule tasks within the specified intervals, the makespan is never going to exceed 8. ■

We have seen so far that CAS problems can possibly be solved by *(i)* first representing them as STNs and *(ii)* then solving the TDP that represents the CAS problem and finally, *(iii)* translating the solution from TDP to a solution of the CAS problem. Therefore, we could in principle use Hunsberger's algorithm to solve the CAS problem instances where agents have unbounded concurrency. While this is good news, there are several shortcomings of such a solution.

- A major shortcoming is the inability to deal with concurrency constraints.
- While any given deadline can be met, we do not know if the resulting global makespan is the least possible.
- Finally, it is likely that we can avoid the Floyd Warshall's algorithm to determine the distance matrix. This can be an expensive computation if the set of tasks/time point variables is large.[3]

Thus, our first challenge would be to

> design an efficient way of decoupling any given, concurrency constraint free, CAS problem instance.

We hope to adapt Hunsberger's solution to the TDP to achieve this. Once such a method is found we would then like to

> design an efficient way of decoupling CAS problem instances with concurrency constraints.

We hope that we will be able to adapt the mechanism derived to solve the case when there are no concurrency bounds to also handle the case when there are concurrency bounds. In either case, we would like to ensure that the makespan of the global solution is as small as possible.

The additional necessity to ensure makespan minimality can be tricky. In fact, as we shall prove in Chapter 5, for a general task graph, it is NP-complete to obtain a makespan minimal decomposition if capacity constraints are involved. Therefore, when capacity constraints are involved, our primary focus shall be to obtain a decomposition so that agents can independently construct their schedules. Once we are able to ensure a decomposition, we can study subclasses of problems where we can give better guarantees on the global makespan.

[3]We will discuss this in greater detail in Chapter 4.

2.3 Summary

In this chapter, we studied various methods that can be used to coordinate plans and schedules of autonomous agents. Broadly, they were based on two important criteria — interaction and decomposition. When agents could interact, then methods such as plan repair, plan merging and iterative refinement could be employed if they are cooperative. However, when agents are competitive instead of cooperative, then market based methods such as auction and negotiation could be used to achieve coordination. When agents cannot interact at all, we resort to decomposition based techniques to achieve coordination.

Information privacy is a major problem with interacting systems, the other issue is the necessity for interaction itself. In several systems for example— adhoc communication networks— it is almost always assumed that communication links between agents are failure prone. Therefore, to achieve true autonomy and ensure coordination we must look into decomposition based methods for coordination. We described Valk's plan coordination framework that is suitable for using decomposition for coordination. We extended it to also describe schedule coordination problems. Finally, using the framework we were able to formally also define the plan and schedule coordination problems that we seek to solve in this thesis.

The summary of plan coordination methods we reviewed and the reasons why they were not chosen are summarised in Table 2.5.

Based on the insights we gained though the review of the plan coordination methods, we attempt to answer the following open problems

1. Does there exist an easily identifiable class of plan coordination problems instances where CVP can be efficiently performed?
2. Can we design an approximation algorithm for CP, which exploits the fact that CVP is polynomially solvable?
3. Does there exist a class of coordination problem instances such that even CP can be solved polynomially for that class?

Similar to the summary table for plan coordination methods, Table 2.6 summarises the various distributed scheduling methods as well as reasons why they were deemed unsuitable for our work.

Method	**Information exchanged**	**Disadvantages**
Plan merging	Exchange of partial plans and deadlock information.	Sharing of partial/full plans and the requirement that agents are able to make changes to their local plans imply a compromise on autonomy.
Plan repair	Exchange of partial or full plans and required changes.	Same as above
Auction	Exchange of bids.	There may be no bids to perform some task(s) or the bids for some tasks may be prohibitively costly leading to a failed auction.
Principled negotiation	Proposals for schedule changes and compensation.	None of the proposals may be agreeable.
Social laws	—	Impractical because coordination instances have little in common.

Table 2.5: Various plan coordination methods and reasons why they are unsuitable.

Apart from distributed scheduling methods we also researched the temporal decomposition approach of Hunsberger. As noted earlier, this method could not be used for our purposes because of its inability to handle capacity constraints. However, the STN framework can be used to describe schedule coordination problem that we deal with in a very basic case — case when there are no concurrency constraints. Thus, our challenge can now be stated as follows.

Can a decoupling technique be developed such that it

1. ensures that every local schedule for the decomposed problem can

Scheduling method	**Disadvantages**
Cooperation	Information regarding local scheduling is shared and agents are required to make changes to their local schedules. Thus compromising autonomy.
Iterative refinement	Same as above.
Bidding, iterative bidding	Absence of bidders for a task or infeasibility of a bid might lead to a failed auction.
Negotiations	Agents may not arrive at a set of proposals that result in a feasible schedule.
STN based techniques	Inability to handle concurrency constraints and absence of a guarantee on minimality of makespan.

Table 2.6: Various distributed scheduling methods and reasons why they are unsuitable.

be merged into a feasible global solution

2. ensures concurrency constraints are not violated and
3. also minimises the global makespan?

So far we have seen several methods that assist us in coordinating plans and schedules. We have also seen that if we do not allow for interaction, then the best way forward is to attempt decomposition. Based on this insight, in the next chapter we deal with the problems related to plan coordination. In Chapter 4 and Chapter 5 we apply the same learning to solve problems related to schedule coordination.

In Chapter 3, we first examine if there exist subclasses of plan coordination problems which can be solved more efficiently than the general case. Later we design approximation algorithms that allow us to coordinate such subclasses of plan coordination problems. In Chapter 4 we show that if resource constraints are absent, then we can encode problems in our framework as STNs. Thus, we are sure that there exists at least one method to solve schedule coordination problems efficiently.

In fact, we go on to show that we can avoid the process of encoding a given CAS problem into an STN and yet solve it using a method derived from Hunsberger's method. We also show that our method is more efficient in solving the CAS problem instances when compared with Hunsberger's method.

In Chapter 5, we study CAS problems with resource constraints and establish their complexity. We later develop approximation algorithms for each of the variants and show approximation bounds for the respective algorithms. Finally, in Chapter 6 we compare the performance of an algorithm, $\mathrm{M}_{\mathrm{ISAn}}$, developed in Chapter 5 against an optimal solution to measure the loss in makespan efficiency when task graphs have very general structures. We perform this comparison based on an empirical analysis of $\mathrm{M}_{\mathrm{ISAn}}$ in coordinating the scheduling activities for ground handling service providers at an airport.

Chapter 3

Tractable plan coordination

In Chapter 2 we presented the state of the art research in plan coordination methods. In general, the problem of coordinating plans of agents using decomposition is known to be Σ_2^p-complete. Therefore, we also stated that we seek to identify subclasses of plan coordination problem instances where at least coordination verification (CVP) can be efficiently performed. Once such a subclass is found, we would like to develop approximation methods that exploit the fact that CVP can be efficiently solved. Further, we also seek to detect if there exist subclasses of efficiently verifiable plan coordination instances, where even the plan coordination problem can be solved efficiently. This chapter is focussed on (i) *finding a subclass of plan coordination instances where coordination verification can be performed efficiently,* (ii) *develop a plan coordination method that exploits such a class of coordination instances and* (iii) *discover, if any, subclasses that exist within this subclass of coordination instances, where plan coordination can be performed efficiently.*

We discussed in Chapter 2 that our best hopes lie in detecting special properties of task graphs that enable efficient verification. To motivate the existence of such structures and hence such subclasses of problem instances, let us first look at an example of plan coordination that arises in a simple supply chain.

Example 3.1 (Supply chain management)**.** Figure 3.1 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. A cross dock is an enterprise which

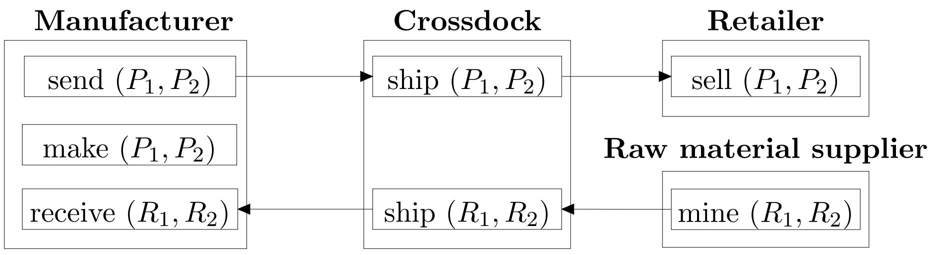


Figure 3.1: Example supply chain scenario. The arrows represent the flow of material in the supply chain.

does not produce anything but is simply involved in distribution of products and raw materials. Each enterprise has to perform some specific tasks such as *send*(P_1, P_2), *ship*(R_1, R_2), or *sell*(P_1, P_2). Flow of goods between enterprises induces precedence constraints on tasks to be performed. For example, goods cannot be sold before they are shipped to the retailer and unless the raw material is mined for, they cannot be shipped to the manufacturer and so on. 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. Such a plan is a partially ordered set of precedence constraints respecting the internal (intra-agent) dependency constraints.

Suppose now that the cross dock plans to send raw materials R_1 and R_2, after sending products P_1 and P_2 and the manufacturer decides to send P_1 and P_2, after receiving R_1 and R_2. Both plans are valid local plans since they do not violate any internal constraints. However, when these plans are combined together we have an infeasible joint plan as shown in Figure 3.2: it contains a dependency cycle. If enterprises could communicate with each other regarding their plans, then quite easily such a dependency cycle can be avoided. However, enterprises in this system are unwilling to communicate with their partners about their own plans for reasons such as information privacy. Therefore, we require a method which does not require agents to interact but still ensures that such dependency cycles are not formed in the global plan.

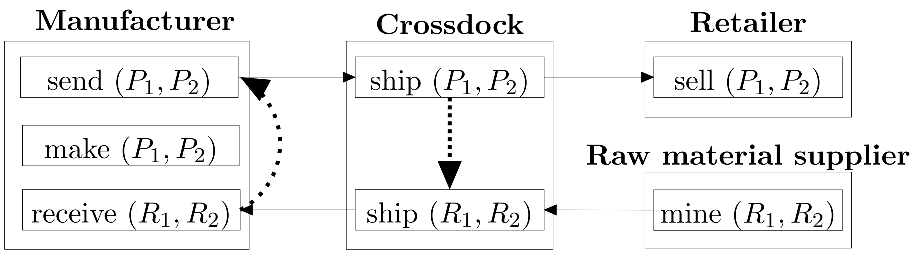


Figure 3.2: Plan deadlock between activities *send*(P_1, P_2), *ship*(P_1, P_2), *ship*(R_1, R_2) and *receive*(R_1, R_2) in the supply chain of Figure 3.1. The deadlock is shown as dotted arcs between the activities.

One simple solution for coordinating plans, is to add a large set of coordination constraints before agents make their local plans. These constraints are such that each agent is left with only a single possible local plan. That is, suppose we add the following precedence constraints — *receive*$(R_1, R_2) \prec$ *make*(P_1, P_2); *make*$(P_1, P_2) \prec$ *send*(P_1, P_2); *send*$(R_1, R_2) \prec$ *send*(P_1, P_2) to the original instance and then let agent formulate their plans. The constraints already imply a unique plan for both the manufacturer and the crossdock. In fact, this is equivalent to a centralised solution where agents have no choice in picking an appropriate plan and hence not suitable. However, acyclicity of the global plan can be guaranteed in a simpler way by just adding a single precedence constraint: *ship*$(R_1, R_2) \prec$ *ship*(P_1, P_2). This allows the manufacturer to plan its tasks in any order. Therefore, this solution is more preferable to the earlier one where none of the agents had any choice. In other words, coordination mechanisms that find a minimum set of coordination constraints are more preferable to ones which add larger sets of coordination constraints. ■

Recall from the previous chapter that plan coordination instances are described by a tuple $\Pi = \langle T, A, \prec, \phi \rangle$. The supply chain situation that was discussed in Example 3.1 can be formally represented as a plan coordination instance as shown below.

Example 3.2. The manufacturer, the crossdock, the retailer and the raw material supplier can be represented as agents A_1, A_2, A_3, A_4 respectively and the actions of agents correspond to the tasks in $\Pi = \langle T, A, \prec, \phi \rangle$. Because of the flow of raw materials and products, the following precedence constraints are imposed on this set of tasks: $mine(R_1, R_2) \prec ship(R_1, R_2)$; $ship(R_1, R_2) \prec receive(R_1, R_2)$; $send(P_1, P_2) \prec ship(P_1, P_2)$; $ship(P_1, P_2) \prec sell(P_1, P_2)$. The allocation of tasks also corresponds to the allocation of actions of the agents.

As explained in the example, the coordination problem faced by the agents can be solved by adding the coordination set $\Delta = \{ship(R_1, R_2) \prec ship(P_1, P_2)\}$. Incidentally, this set Δ is also the minimum coordination set. ■

It is easy to check if the coordination set for the simple supply chain scenario in Example 3.1, is indeed sufficient for coordination. However, the same cannot be said in general. Valk [Valk 2005] shows that verifying whether a coordination set is sufficient is an intractable problem. That said, not all coordination problems are equally hard. As noted in the previous chapter, Valk himself indicates that there could be classes of problems where coordination verification can be efficiently performed.

In Example 3.1, note that tasks allocated to the same agent do not have any precedence constraints between each other. This means that for every i the precedence relation $\prec_i$ is empty. This property, called the *intra-freeness* property, ensures that coordination verification can be done efficiently. We present a more elaborate proof that confirms that the CVP for plan coordination instances with this intra-freeness property can be efficiently solved later in the chapter. Coordination instances with this property are termed *intra-free instances* and are the focal point of this chapter.

In the next section, we will show that the coordination verification problem for intra-free instances is indeed polynomially solvable. Thus reducing the coordination problem to an NP-complete problem — a step lower in the complexity hierarchy than the general coordination problem. As we will see, these results will allow us to develop a new (and better) approximation algorithm — the Advanced depth partitioning algorithm (DP^*) — to solve intra-free coordination instances in Section 3.2. We show that the DP^* algorithm improves upon the existing state of the art depth partitioning approach as it never constructs a coordination set bigger than the one constructed by depth partitioning. Finally, in Section 3.3.2 we present a special class of intra-free instances called the *Special linear intra-free instances (*SLIF*)* where even the coordination problem can be efficiently solved.

3.1 Intra-freeness and tractable coordination verification

Let us start our study by formally defining the property of intra-freeness.

Definition 3.1. Intra-free instance: A coordination instance $\Pi = \langle \{T_i\}_{i=1}^n, \prec \rangle$ is *intra-free* if, for every T_i, the corresponding subgraph $G_{\Pi_i} = (T_i, \prec_i)$ of the task graph G_Π is the empty graph $(T_i, \emptyset)$. That is, there are no precedence relations between tasks assigned to the same agent. Intra-free instances are described by the tuple $\Pi = \langle \{T_i\}_{i=1}^n, \prec, \emptyset \rangle$. The empty set here indicates that each set $\prec_i$ in $\Pi = \langle \{T_i\}_{i=1}^n, \prec \rangle$ is empty. ■

If an instance is intra-free, then every task t is either an isolated node in G_Π or is adjacent to a task t' assigned to another agent.

For our analysis, we define a special subclass of intra-free coordination instances. We call it the class of *strictly intra-free instances*, which will be of special importance in proving properties of intra-free coordination instances:

Definition 3.2. Strictly intra-free instance: An intra-free coordination instance $\Pi = \langle \{T_i\}_{i=1}^n, \prec \rangle$ is called *strictly intra-free* if, for every i, the set T_i of nodes in the graph G_{Π_i} can be partitioned in two disjoint subsets $In(T_i)$ and $Out(T_i)$ such that

1. $In(T_i)$ consists of nodes t with out-degree equal to 0 ($out(t) = 0$) and
2. $Out(T_i)$ consists of nodes t with in-degree equal to 0 ($in(t) = 0$).

The nodes in $In(T_i)$ are called *sinks* and the nodes in $Out(T_i)$ are called *sources*. Strictly intra-free instances are described as the tuple $\Pi = \langle \{T_i\}_{i=1}^n, \prec, strict \rangle$. To differentiate between intra-free instances and strictly intra-free instances, we use the word *strict* within the tuple $\Pi = \langle \{T_i\}_{i=1}^n, \prec, strict \rangle$. ■

Example 3.3. Consider the task graph in Figure 3.3. The coordination instance can be described as the tuple $\Pi = \langle \{\{t_1, t_2\}, \{t_3, t_4, t_5\}\}, \{t_1 \prec t_3, t_1 \prec t_4, t_5 \prec t_2\}, strict \rangle$. Tasks t_1, t_5 are sources and tasks t_2, t_3, t_4 are sinks. ■

Notice that in strictly intra-free coordination instances, we *do not have* tasks t with both $out(t) > 0$ as well as $in(t) > 0$. That is, we *exclude* intra-free instances such that there are tasks t', t'' in other partition blocks T_j, T_k such that $t' \prec t$ and $t \prec t''$.

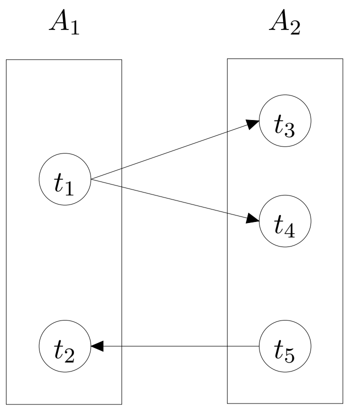


Figure 3.3: Strictly intra free instance.

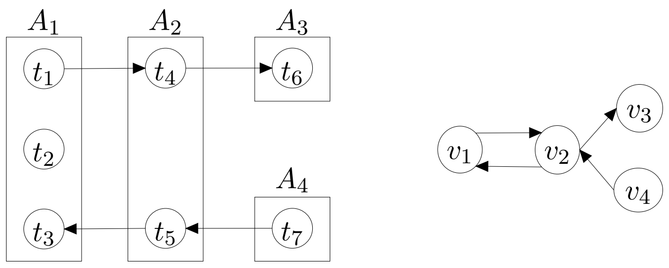
3.1.1 Agent dependency graphs

The rather surprising result we prove now is that for intra-free coordination instances, the coordination verification problem can be reduced to checking the acyclicity of a (smaller) graph called the *agent dependency graph*. Because checking for acyclicity of a graph is in P [Christofides 1975], the coordination verification problem for intra-free coordination instances is in P as well.

Definition 3.3. Agent dependency graph: Given a task graph G_Π of an intra-free instance $\Pi = \langle T, A, \prec, \phi, \emptyset \rangle$, the agent dependency graph derived from G_Π is a graph $G_A = (V, E)$, where $V = \{v_i : A_i \in A\}$ is the set of nodes corresponding to agents and $E = \{(v_i, v_j) : (t, t') \in \prec \text{ and } t \in T_i, t' \in T_j\}$ is the dependency relation between them. ■

Example 3.4. Consider the Supply Chain example (Example 3.1), where seven tasks are allocated to four agents. Its task graph is shown in Figure 3.4(a). Figure 3.4(b) shows its corresponding agent dependency graph. ■

We start by first proving some results for strictly intra-free coordination instances. Then we will show that intra-free instances in general can be easily transformed to strictly intra-free coordination instances. We show that the properties holding for strictly intra-free instances also hold for this larger class of instances.



(a) Task graph (b) Agent dependency graph

Figure 3.4: The task graph of an intra-free instance and its corresponding agent dependency graph.

Recall that given a task graph $G_\Pi = (T, \prec)$ and a partitioning $\{T_i\}_{i=1}^n$ of the set of nodes T, the coordination verification problem for intra-free instances comes down to, deciding whether for all possible *acyclic* extensions ${G_{\Pi_i}}^* = (T_i, \prec)$ of the subgraphs $G_{\Pi_i} = (T_i, \emptyset)$, the resulting extension ${G_\Pi}^* = (T, \prec \cup \prec_1^* \cup \ldots \cup \prec_n^*)$ of G_Π is still acyclic. While the problem to decide whether a given graph is acyclic is in P, the CVP is intractable [Valk 2005] for general coordination instances.

3.1.2 Coordination verification through agent dependency graphs

There is an obvious connection between the coordination verification problem for $\Pi = \langle \{T_i\}_{i=1}^n, \prec, \emptyset \rangle$ and the *acyclicity* of G_A. The only cycles that can occur in any CVP instance $\Pi = \langle \{T_i\}_{i=1}^n, \prec, \emptyset \rangle$ are inter-agent cycles and therefore, if the agent dependency graph does not contain a cycle then the intra-free instance also does not contain one:

Proposition 3.1. *Let* $\Pi = \langle \{T_i\}_{i=1}^n, \prec, \emptyset \rangle$ *be an intra-free coordination instance and* G_A *its associated agent dependency graph. Then* G_A *is acyclic implies that* Π *is coordinated, i.e.,* Π *is a yes-instance of the coordination verification problem.*

Proof. If G_A is acyclic, the only cycles that could occur in any extension of G_Π are

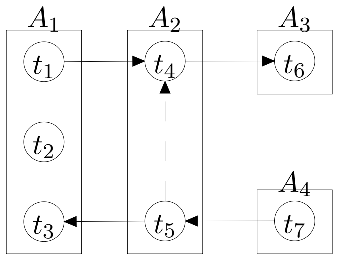


Figure 3.5: Coordinated instance of a task graph.

cycles *within* a task set T_i of an agent A_i. These are excluded, since each individual extension $\langle T_i, \prec_i^* \rangle$ of $\langle T_i, \prec_i \rangle$ has to be an acyclic refinement of $\prec_i$. □

In general, Proposition 3.1 is true for every coordination instance. However, the converse is not true: even if G_A is cyclic, we might have a yes-instance of $\Pi = \langle \{T_i\}_{i=1}^n, \prec, \emptyset \rangle$. That is, after adding the coordination arcs, the agent dependency graph can still contain a cycle, although the instance now is coordinated. For instance, the agent dependency graph for the Example in 3.1 remains the same even after addition of the coordination arc. Therefore, it is not possible to detect from just the agent dependency graph whether an instance is coordinated.

Example 3.5. Consider the scenario in Example 3.1. This coordination instance is represented by the task graph shown in Figure 3.4(a). The agent dependency graph, as shown in 3.4(b), contains a cycle: v_1 is dependent upon v_2 vice-versa. However, if an additional constraint $t_5 \prec t_4$ is given to agent A_2 as shown in Figure 3.5 then, whatever plan A_1 chooses, it does not affect the feasibility of the total plan. That is, the instance is coordinated. However, the agent dependency graph remains the same. ■

Interestingly, if $\Pi = \langle \{T_i\}_{i=1}^n, \prec, \emptyset \rangle$ is a strictly intra-free coordination instance we can actually show that the converse holds, too:

Proposition 3.2. *Let* $\Pi = \langle \{T_i\}_{i=1}^n, \prec, strict \rangle$ *be a strictly intra-free coordination instance and* G_A *its agent dependency graph. Then the following holds:* G_A *is acyclic iff* Π *is a yes-instance of the coordination verification problem.*

Proof. The if direction has been shown in Proposition 3.1. Here, we have to show that $\Pi = \langle \{T_i\}_{i=1}^n, \prec, \mathit{strict}\rangle$ is not coordinated if G_A contains a cycle. So, let G_A contain a simple cycle $(v_1, v_2, \ldots, v_k, v_1)$. Since the coordination instance is *strictly* intra-free, there must exist a sequence of tasks

$$(t_{i_1,2}, t_{i_2,1}, t_{i_2,2}, \ldots, t_{i_k,1}, t_{i_k,2}, t_{i_1,1})$$

such that for $j = 1, \ldots, k$, we have

$$\begin{aligned}
&t_{i_j,1}, t_{i_j,2} \in T_{i_j}\\
&t_{i_j,2} \prec t_{i_{j+1},1} \text{ and}\\
&t_{i_k,2} \prec t_{i_1,1}
\end{aligned}$$

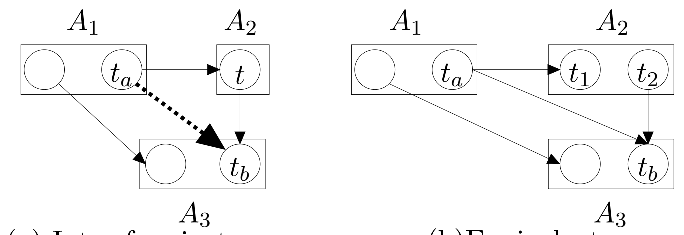
But then, it immediately follows that the empty precedence relation $\emptyset$ in $G_{\Pi j} = (T_j, \emptyset)$ has a simple acyclic extension $\{t_{i_j,1} \prec^* t_{i_j,2}\}$ such that the graph $G^*_T = (T_j, \prec \cup \prec^*)$ where $\prec^*$ is the union of extensions $\prec^*_{i_j}$ contains a cycle. Hence, $\Pi = \langle \{T_i\}_{i=1}^n, \prec, \mathit{strict}\rangle$ 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. We can transform intra-free instances to strictly intra-free instances by applying the following *task-splitting* procedure on tasks that violate the strict intra-freeness property.

To perform task splitting, given an arbitrary intra-free coordination instance $\Pi = \langle \{T_i\}_{i=1}^n, \prec, \emptyset\rangle$, consider the transitive closure $G_\Pi{}^+ = (T, \prec^+)$ of its task graph where, $\prec^+$ is the transitive closure on the precedence relationships in G_Π. 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 each precedence constraint $t' \prec t$ add a precedence constraint $t' \prec t_1$ and add $t_2 \prec t'$, whenever $t \prec t'$;
3. Remove t and all precedence constraints it is mentioned in.

We call the strictly intra-free instance Π' obtained by the above procedure as the *strictly intra-free variant* of Π. To show that the task splitting procedure results in an intra-free instance with only sources and sinks, observe the following:



(a) Intra free instance (b)Equivalent Strictly intra-free instance

Figure 3.6: Converting to strictly intra-free instance.

- The only tasks unaffected by the procedure are those tasks which are already sinks or sources,
- If a task t is split into t_1, t_2 then there exists no path between t_1 and t_2.

It is easy to see that at the end of this procedure there are no tasks with both in degree and out degree greater than 0 since, every task that does not satisfy this property is split into a sink and a source.

Example 3.6. Figure 3.6 illustrates the task-splitting procedure. Note that the transitive closure of the precedence relationship between tasks t_a, t, t_b is represented by the dashed arrow (t_a, t_b). The task t of agent A_2 has been split into two tasks t_1 and t_2. We can now simply verify that when the strictly intra-free variant is coordinated, the intra-free instance must be also coordinated. ∎

We need to show now that the task splitting procedure preserves all the coordination properties. Therefore, we obtain the following proposition.

Proposition 3.3. *An intra-free coordination instance* Π *is coordinated whenever its strictly intra-free variant* Π' *is coordinated.*

Proof. We only prove the if-part. The only-if part goes analogously. Suppose the contrary that the strictly intra-free variant $\Pi' = \langle \{T_i'\}_{i=1}^{n}, \prec', strict \rangle$ is coordinated while the intra-free instance $\Pi = \langle \{T_i\}_{i=1}^n, \prec, \emptyset \rangle$ is not. Then for every i, $i = 1, 2, \ldots, n$, there exists some partially ordered extension $\prec_i^*$ of $\langle T_i, \prec_i \rangle$ such that $\prec^* = \prec \cup \prec_1^* \cup \ldots \cup \prec_n^*$ is cyclic. Given $\prec^*$, create the following extension $\prec'^*$ of $\prec'$:

For every $(t, t') \in \prec^*$ such that $t, t' \in T_i$

- if both t, t' have been split, add (t_1, t'_2) to $\prec'^*$,
- else if t has been split add (t_1, t') to $\prec'^*$,
- else if t' has been split add (t, t'_2) to $\prec'^*$,
- else add (t, t') to $\prec'^*$.

It is not difficult to see that for every i, $i = 1, 2, \ldots, n$, $\langle T'_i, \prec_i'^* \rangle$ is a partial order, but $\langle \{T'_i\}_{i=1}^n \prec'^* \rangle$ is not. Hence Π' cannot be coordinated. □

Let us illustrate the proof with a simple example.

Example 3.7. Consider the situation in Figure 3.7. The original intra-free instance is shown in Figure 3.7(a). Tasks t_3, t_4 are the only tasks in the instance that can be split. The strictly intra-free equivalent of the instance in Figure 3.7(a) is shown in Figure 3.7(c). If there exists a cyclic extension of the intra-free instance as shown in Figure 3.7(b), then we can create a similar cyclic extension in the strictly intra-free version. Consider the cycle $t_3 \prec t_5 \prec t_6 \prec t_4 \prec t_3$. To create a cycle in the strictly intra-free version, let us first consider $t_4 \prec t_3$. In the strictly intra-free version, t_3 is split into t_3^1 and t_3^2 and t_4 is split into t_4^1, t_4^2. Now as noted in the proof, we add a constraint $t_4^1 \prec t_3^2$. Similarly, for the constraint $t_5 \prec t_6$, since, neither task is split, we also add $t_5 \prec t_6$ to the strictly intra-free version. Clearly this creates a cycle $t_4^1 \prec t_3^2 \prec t_5 \prec t_6 \prec t_4^1$. The existence of such a cyclic extension for the strictly intra-free instance contradicts the assumption that it was coordinated. ■

Based on these results we can now claim that the coordination verification problem for intra-free instances reduces to checking the acyclicity of the corresponding agent dependency graph of the strictly intra-free variant. Thus, the following theorem holds:

Theorem 3.4. *Coordination verification for intra-free instances can be performed in polynomial time.*

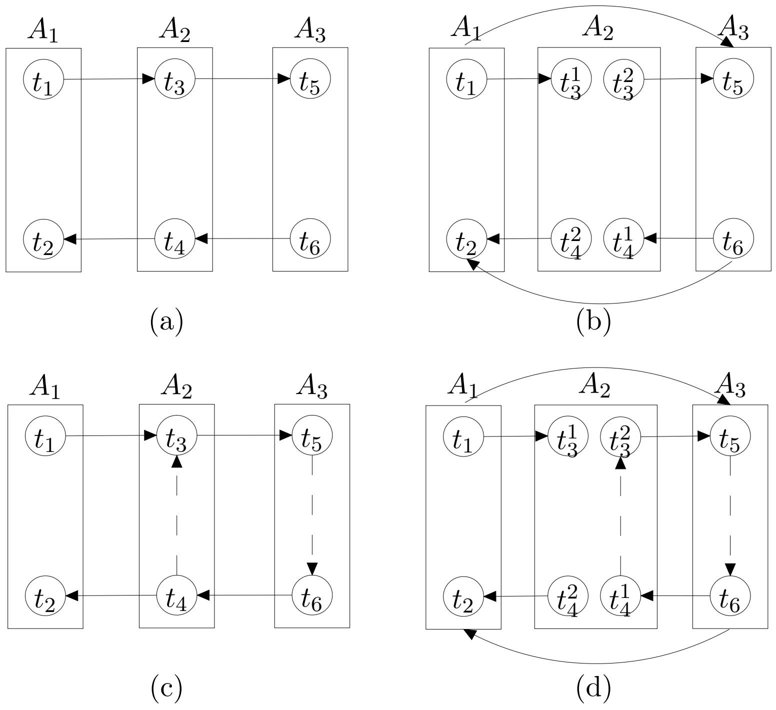


Figure 3.7: (a) Intra-free instance. (b) Equivalent strictly intra-free instance.(c) Cyclic extension of intra-free instance. (d) Cyclic extension of strictly intra-free instance.

Proof. From Propositions 3.1 and 3.2 we know that if the agent dependency graph is acyclic then the intra-free instance is coordinated. Further, because of the task splitting procedure and Observation 3.1.2 we know that it is enough to verify if the strictly intra-free variant is coordinated or not. Testing the acyclicity of a graph can be done in polynomial time [Cormen *et al.* 1990]. Thus, we claim that coordination verification can be done in polynomial time for intra-free instances. □

3.2 Approximation algorithms for intra-free plan coordination

In the previous sections it was established that coordination verification for intra-free instances could be achieved efficiently. One might hope therefore that, finding a *minimum coordination set* is also tractable in the case of intra-free coordination instances. This turns out to be wrong. We show below that the problem of finding such a minimum set of coordination constraints is NP-complete:

Proposition 3.5. *The decision version of coordination problem for intra-free coordination instances is NP-complete.*

Proof. Theorem 3.4 shows that if we are given a coordination set for an intra-free instance it is possible to efficiently verify whether the coordination set ensures that global plans are always acyclic.

We now prove the NP-hardness of the intra-free coordination problem by reducing the Directed feedback vertex set problem to the plan coordination problem.

Definition 3.4. Directed feedback vertex set problem: 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. ■

Let $I = (G = (V, E), K)$ be an instance of the directed feedback vertex set (FVS) problem [Festa *et al.* 1999]. We obtain an instance $\Pi = \langle \{T_i\}_{i=1}^n, \prec, \emptyset \rangle$ of the intra-free coordination problem by

1. duplicating tasks: $T = V \cup \ \{v' : v' \in V\}$ and
2. for every $v \in V$ creating an agent A_v having the tasks v and v' to complete and
3. 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 strictly intra-free, since for tasks v' we have $in(v') = 0$ and for tasks v we have $out(v) = 0$.

Observe that G has a feedback vertex set of size K, whenever K coordination arcs are needed to ensure that $\langle T, \prec, strict\rangle$ is coordinated. Suppose $W \subseteq V$ is a feedback vertex set 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 cyclic 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. Further, since we can verify both the sufficiency of the coordination set as well as the size of the coordination set in polynomial time, we conclude that the decision version of the coordination problem is also NP-complete. □

Let us understand the proof of Proposition 3.5 through a simple illustration.

Example 3.8. Consider the situation in Figure 3.8. The instance of the FVS problem $I = (G = (V, E), K)$ has a vertex set $V = \{v_1, \dots, v_6\}$, edge set $E = \{(v_1, v_2)(v_2, v_3)(v_6, v_5)(v_5, v_4)\}$ and let $K = 4$. We construct an instance of the intra-free plan coordination problem as follows:

1. We create a set of tasks $T = \{t_1, \dots, t_6, t'_1, \dots, t'_6\}$,
2. The set of precedence constraints $\prec = \{(t'_1 \prec t_2), (t'_2 \prec t_3), (t'_6 \prec t_5), (t'_5 \prec t_4)\}$,
3. We create 6 agents $A = \{A_1, \dots, A_6\}$ and
4. Finally, due to our allocation function, $T_1 = \{t_1, t'_1\}, T_2 = \{t_2, t'_2\}, T_3 = \{t_3, t'_3\}, T_1 = \{t_4, t'_4\}, T_5 = \{t_5, t'_5\}, T_6 = \{t_6, t'_6\}$.

Now suppose, the feedback vertex set contains the vertices v_1, v_2, v_3, v_4, then we can construct a solution to the intra-free coordination instance by adding the following arcs: $(t'_1 \prec t_1), (t'_2 \prec t_2), (t'_3 \prec t_3), (t'_4 \prec t_4)$. Quite clearly, the only agents who can still make local plans are A_5 and A_6. A_6 cannot be part of any inter-agent cycle as it has only out going inter-agent arcs. A_5 alone cannot create an inter-agent cycle, and hence the instance is coordinated. ■

Although Proposition 3.5 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.

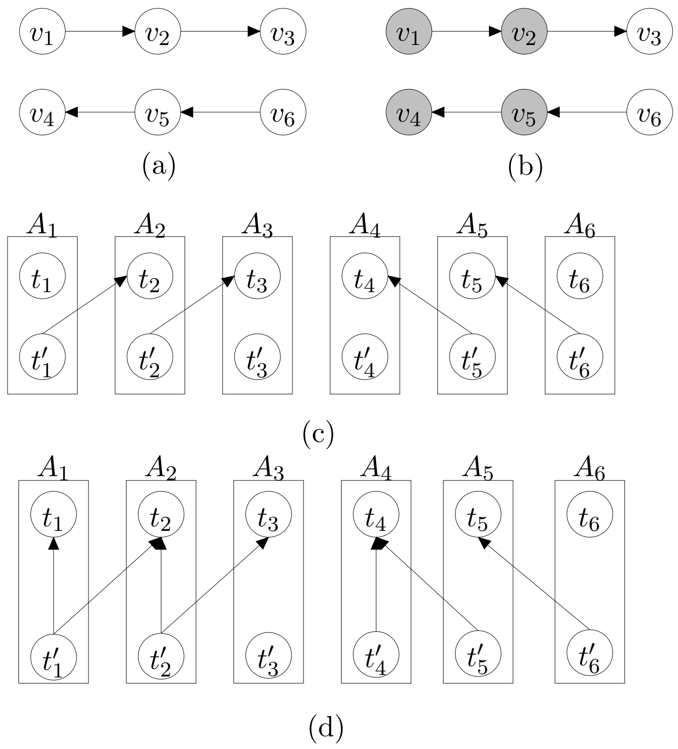


Figure 3.8: Transforming a feedback vertex set into a coordination set.

The Depth partitioning (DP) algorithm [Steenhuisen *et al.* 2008] is such an algorithm that finds a sufficient, but not necessarily minimum, number of coordination constraints required to coordinate a given instance.

This algorithm can be stated 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 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 $t, t' \in T_i$, 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 [Steenhuisen *et al.* 2008]:

Proposition 3.6. *The DP algorithm results in a set of constraints sufficient to ensure coordination of the given plan coordination instance* Π.

Proof. (adopted from [Steenhuisen *et al.* 2008]) Suppose the contrary, $\langle \{T_i\}_{i=1}^n, \prec \cup \Delta \rangle$ is not plan coordinated. Then there must exist some cycle $c = (t_1, t_2, ..., t_m, t_1)$ where c contains *(i)* at least two tasks t_i and t_k belonging to different agents and *(ii)* at least two tasks connected via a local plan. However, notice 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: $depth(t) \geq depth(t')$.

Therefore, the depth of the first task t_1 occurring in c should be strictly less than the depth of the last task t_m in c or $depth(t_1) < depth(t_m)$ which contradicts the precedence constraint $t_m \prec t_1$ creating a cycle. Hence, such a cycle c cannot exist and the instance is plan coordinated. □

The drawback 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 depth partitioning needs to be applied, and those where we do not need to apply it. This means that no constraints are applied between tasks which cannot form a cycle. In most cases, this leads to a reduction in the number of coordination constraints. We call this algorithm as the *Advanced depth partitioning* algorithm or *DP**.

The basic idea is to use the agent dependency graph as such a filter:

1. Whenever the agent dependency graph is acyclic, we do not need to add any coordination constraints.

2. 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.

   (a) We can use an approximation of the minimum feedback vertex set to select an agent A_i occurring in the feedback set.

   (b) 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.

3. 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.

4. 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.

5. 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 agent dependency graph again.

We iterate this process until the agent dependency graph is acyclic.

Technically then, we need to use an approximation algorithm for the BLACKOUT FEEDBACK VERTEX SET problem to indicate which agents create the cycle.

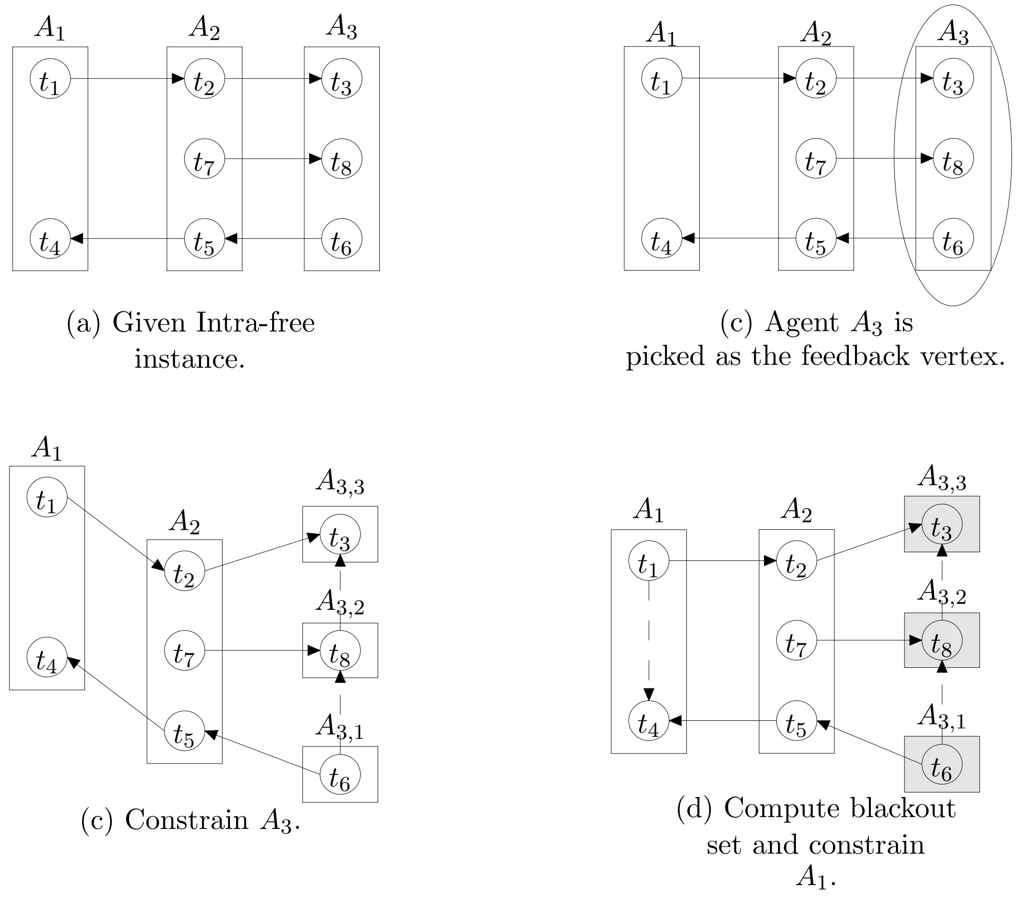
Definition 3.5. Blackout feedback vertex set (B-FVS) problem [Garey and Johnson 1979]: The Blackout feedback vertex set (B-FVS) problem is the following: given a directed graph $G = (V, E)$, a blackout set $B \subseteq V$ and a $K \in Z^+$, to decide whether there exists a subset $F \subseteq V$ with $F \cap B = \emptyset$ and $|F| \leq K$, nodes from V whose removal will render the remaining graph G acyclic. ■

An approximation algorithm for this problem has been proposed in [Even *et al.* 1998]. Their algorithm finds a feedback vertex set having an approximation ratio of $O(log|V|loglog|V|)$ where V is the set of vertices. The DP^* algorithm can be stated as in Algorithm 1.

Algorithm 1 Advanced depth partitioning algorithm (DP^*)

```
Require: An intra-free coordination instance $\Pi = \langle \{T_i\}_{i=1}^n, \prec, \emptyset \rangle$; $A = \{A_1, A_2, \ldots, A_n\}$.
Ensure: A set of coordination constraints $C$ added to $G_\Pi$ to make it coordinated.
 1: Compute $depth(t)$ for every $t \in T$;
 2: let $G_A = (V_A, E_A)$ be the agent dependency graph associated with $G_\Pi$, the task graph of $\Pi = \langle \{T_i\}_{i=1}^n, \prec, \emptyset \rangle$;
 3: $B = \{A_i$ : all tasks $t \in T_i$ having 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:    end for
13:    Split $T_i$ in $k$ subsets $T_{i,1}, \ldots T_{i,k}$ such that depth $depth(t) \forall t \in T_{i,k} = k$;
14:    $T = (T - T_i) \cup \{T_{i,1}, \ldots T_{i,k}\}$;
15:    let $G_A$ be the new dependency graph associated with $G_\Pi$;
16:    add the nodes $A_{i,j}$ corresponding to the sets $T_{i,j}$ to $B$;
17: end while
18: return $C$
```

Example 3.9. Consider the situation in Figure 3.9. 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 agent A_3, i.e., $F = \{A_3\}$. In agent A_3, we



(a) Given Intra-free instance.

(c) Agent A_3 is picked as the feedback vertex.

(c) Constrain A_3.

(d) Compute blackout set and constrain A_1.

Figure 3.9: An example of applying the Advanced depth partitioning algorithm.

first add coordination constraints $t_6 \prec t_3$, $t_6 \prec t_8$ and $t_8 \prec t_3$. We 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 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 $t_7 \prec t_5$. ■

We now show that the Advanced depth partitioning algorithm DP^* is correct.

Proposition 3.7. *Let* $\Pi = \langle \{T_i\}_{i=1}^n, \prec, \emptyset \rangle$ *be an intra-free plan coordination instance and* Δ*, the set of additional precedence constraints returned by* DP^**. Let* G_Π *be its task graph. Then, the instance* $\Pi'_T = \langle T, \prec \cup \Delta \rangle$ *is plan coordinated.*

Proof. We first show that the algorithm terminates, and then show that Δ is sufficient. For each iteration of the while loop in DP^*, the number of agents who can be constrained reduces because once A_i has been constrained, each of the $A_{i,j}$ agents that result by splitting it enter the Blackout set (all tasks within $T_{i,j}$ have the same depth). In the worst case, all nodes enter the blackout set B, resulting in an empty feedback vertex set. Thus, we can be sure that the algorithm terminates.

Termination already implies that the resulting agent dependency graph is acyclic. We know from Proposition 3.1 that if the agent dependency graph is acyclic then the associated coordination instance is coordinated. Therefore, it implies that the set of coordination constraints applied by DP^* algorithm is sufficient to ensure coordination. □

The real benefit of the DP^* algorithm is that in a large number of cases, the coordination set returned by it is much smaller than the one returned by DP algorithm. The reason for this improved performance is the following:

- The DP algorithm applies coordination constraints among tasks of *all* agents, whereas the DP^* algorithm applies it on a small number of agents chosen by a FVS algorithm.

For each agent A_i that needs to be constrained, we could potentially require $T_i - 1$ constraints in the worst case. Therefore, it naturally follows that if the feedback vertex set excludes even a single agent A_i, the potential savings in terms of number of coordination constraints could be $T_i - 1$. Typically, the feedback vertex set is a much smaller set than the set of agents in most cases. Suppose that the difference in the size of these sets is x. This would mean that the *potential saving* in the number of coordination constraints due to DP^* is $\sum_{i=1}^{m}(T_i - 1)$ where, $A_i \notin FVS$. While the exact number of coordination constraints saved due to DP^* is instance dependent, we can easily show that the coordination set derived out of it is never larger than the DP algorithm. Thus,

Proposition 3.8. *The coordination set Δ returned by the DP^* algorithm is never larger than the coordination set Δ' returned by the DP algorithm.*

Proof. Let the set of agents having tasks of different depths be denoted by $A_d : A_d \subseteq A$. It is easy to see that Δ' has coordination constraints for tasks belonging to each of the agents in A_d. In the DP^* algorithm, if an agent belongs to the set $A - A_d$, then either it belongs to the blackout set or no constraints are added between tasks in those agents because their depths are the same. Therefore, agents in $A - A_d$ can never contribute to Δ. Therefore, the only agents that can be constrained in the DP^* algorithm are also the agents in A_d. Consequently, $|\Delta| \leq |\Delta'|$. □

3.3 Tractable cases of plan coordination

In the previous sections, we have seen that real world problems such as supply chain management problems reveal simpler structures for coordination. It was shown that, coordination problems with intra-freeness property were simpler to coordinate than the general case. For example, Proposition 3.5 states that for the class of intra-free instances it is NP-complete to find a minimum coordination set while in general the problem is Σ_2^p-complete. However, motivated again by the fact that we were not only able to verify the solution for problems such as the one in Example 3.1 but also find them easily, we search further within the class of intra-free instances with a hope of finding a class of problem instances that can be coordinated in polynomial time.

Not surprisingly, real world problems with simpler structures for coordination exist. Domains such as supply chain management [Yadati *et al.* 2010] and hospital patient treatment [Yadati *et al.* 2011] have coordination problems that can be solved efficiently. However, these instances have two very special properties that allow for efficient coordination:

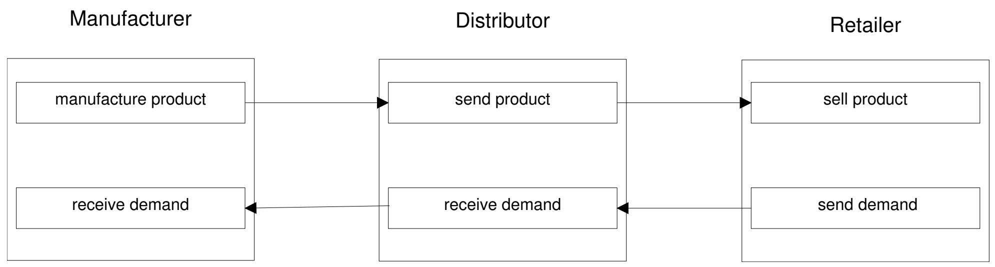


Figure 3.10: A single product supply chain system.

- These problems exhibit a *linear structure*. That is, tasks in these problems are always preceded by utmost a single task and succeeded by utmost a single task. As we will show in the next section, this linearity also is not enough to ensure that minimum coordination sets can be found.
- All chains in the instance either correspond to the ordering of agents or contradict it in a pairwise fashion (a more precise definition is presented later in this section).

As an illustration that simpler classes of coordination problems exist, consider the following example of a simple single product supply chain.

Example 3.10. Consider a simple single product supply chain comprising of three agents — a Manufacturer, a Distributor and a Retailer. The manufacturer has to decide between being a *push* system or a *pull* system.

If the manufacturer decides to be a push system, then clearly he needs to prioritise manufacturing the product over waiting for the demand. In case he wants to be a pull system, then he has to first wait for the demand and then manufacture the product. Similarly, the distributor in a push system sends the products to the retailer first so that the retailer in turn can push the product into the market. On the other hand, in a pull system the retailer first makes a demand, which is passed on through the distributor to the manufacturer.

If the manufacturer and distributor decide to be push systems, then any decision of the retailer does not create a deadlock. Similarly, it is not hard to see that if the retailer and the distributor decide to be pull systems, the instance is again coordinated. A deadlock can occur however, if the manufacturer decides to be a pull system and the distributor decides to be a push system.[1] ■

From Example 3.10, it appears as if coordinating instances where tasks with a maximum of a single predecessor and a single successor are easier to coordinate. However, it can be shown that this is still an intractable problem in general. Fortunately, there does exist a subclass — *special linear intra-free* — of instances where a minimum coordination set can be efficiently found. In the next section, we first establish the intractability of finding a minimum coordination set for linear intra-free instances and then discuss special linear intra-free instances, where the coordination set can be efficiently found.

3.3.1 Linear intra-free instances

Consider again the problem scenario presented in Example 3.10. It was already shown that this belonged to the class of intra-free instances. Notice also that maximum in and out-degree of tasks in this scenario was 1. That is, all tasks have at most a single predecessor and a single successor. Intra-free instances with such special partial orders are termed as *Linear intra-free instances.* They can be defined as follows:

Definition 3.6. Linear intra-free instance: A given intra-free plan coordination instance $\Pi = \langle \{T_i\}_{i=1}^n, \prec, \emptyset \rangle$ is *linear* if each task $t \in T$ has $in(t) \leq 1$ and $out(t) \leq 1$. Linear intra-free instances are represented as $\Pi = \langle \{T_i\}_{i=1}^n, \prec, linear \rangle$. ■

It might appear that all linear intra-free coordination instances can be easily coordinated. However, it is not so. In fact, one can reduce the decision version of the Directed feedback arc set problem (DFAS) which is a well known NP-complete problem to this problem. The decision version of DFAS can be stated as following:

Definition 3.7. Directed feedback arc set problem (DFAS) [Garey and Johnson 1979]: Given a directed graph $G = (V, A)$ and an integer K, does there

[1] A deadlock can also occur if manufacturer and retailer decide to be pull systems and the distributor decides to be a push system.

exist a set of arcs $A' \subset A$ such that A' consists of at least one arc from each directed cycle in G and $|A'| \leq K$? ■

The general idea behind the reduction is that every node x is split into two nodes x and x'. The node x together with all its predecessors $\{x_y : y \in P(x)\}$ is allocated to one agent, say A_i and the node x' together with all successors $\{x_y : y \in S(x)\}$ is allocated to a different agent A_j. Agents are connected via the arcs $(x \prec x')$ encoding nodes and $(x_y \prec y_x)$ encoding arcs. So in total we have $2|V|$ agents.

Proposition 3.9. *Given a linear intra-free instance* $\Pi = \langle \{T_i\}_{i=1}^{n}, \prec, linear \rangle$ *and an integer* M *it is* NP*-complete to find a coordination set* Δ *such that* $|\Delta| \leq M$*.*

Proof. Consider a graph $G = (V, E)$ of the DFAS problem and a positive integer K

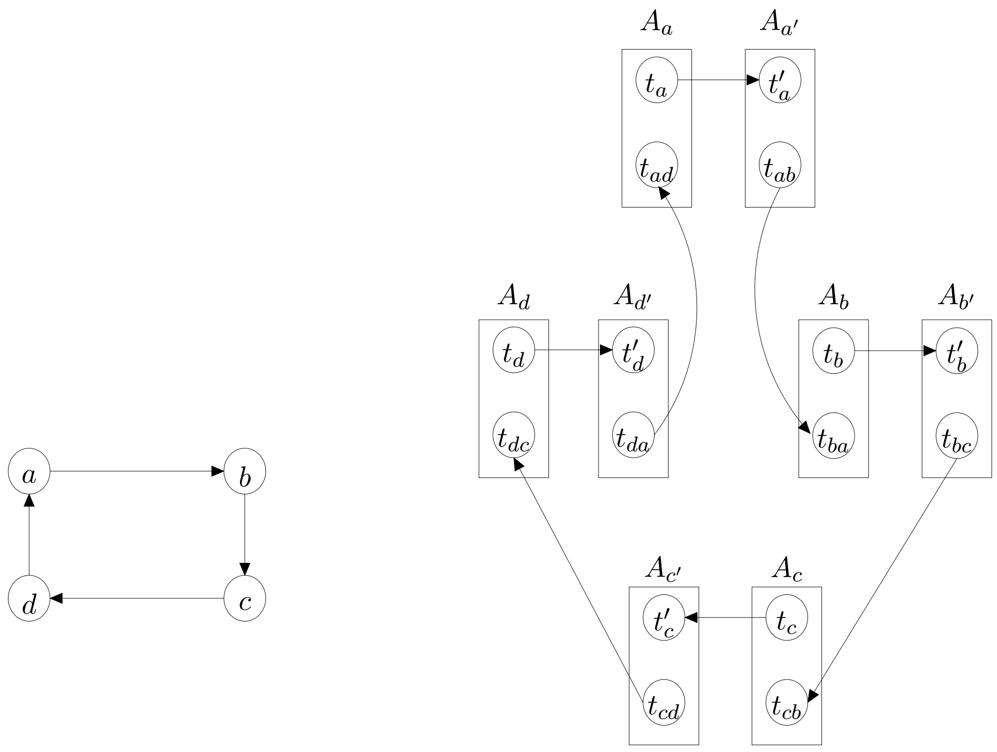
- For every $x \in V$ let $S(x) = \{y : (x, y) \in E\}$ denote the set of successors of x.
- For every $x \in V$ let $P(x) = \{y : (y, x) \in E\}$ denote the set of predecessors of x.

Next, create the coordination instance $\Pi = \langle T, A, \prec, \phi \rangle$ through the following steps:

1. For every $x \in V$, add t_x, t'_x to T,
2. For every $(x, y) \in E$, add t_{xy}, t_{yx} to T,
3. For every $x \in V$, add $t_x \prec t'_x$,
4. For every $(x, y) \in E$, add $t_{xy} \prec t_{yx}$,
5. Finally, for every $x \in V$, partition T such that $T_x = \{t_x \cup t_z : z \in P(x)\}$ and $T_{x'} = \{t'_x \cup t_z : z \in S(x)\}$.

The encoding ensures that the instance is intra-free and satisfies $in(x) = 1$ and $out(x) = 1$ for every task. It is now not that difficult to see that if (x, y) occurs in the feedback set, a coordination arc $(t_y \prec t_{yx})$ has the effect of blocking one or more cycles.[2] Conversely, if it had a coordination arc $(t_y \prec t_{yx})$ or a coordination arc $(t_{xy} \prec t'_x)$ for the intra-free instance (x, y) can be added to feedback set. Further, since we can verify if the size of the coordination set is lesser than M in polynomial time, we claim that the problem is NP-complete. □

[2]Alternatively, we could also add $(t_{xy} \prec t'_x)$.



(a) FAS instance (b) Equivalent linear intra-free instance

Figure 3.11: Reducing from the feedback arc set to the problem of coordinating linear intra-free instances.

Let us illustrate the reduction through a simple example.

Example 3.11. Consider the DAG in Figure 3.11. We are given the DAG in Figure 3.11(a). To transform it into an instance of the linear intra-free coordination problem, we first compute the set of successors S_x and the set of predecessors P_x

for each node x in the FAS problem instance:

$$S_a = \{b\}; S_b = \{c\}; S_c = \{d\}; S_d = \{a\};$$
$$P_a = \{d\}; P_b = \{a\}; P_c = \{b\}; P_d = \{c\}.$$

We then create the tasks $t_a, t'_a, t_b, t'_b, t_c, t'_c, t_d, t'_d$ because of nodes a, b, c, d in the FAS problem instance. We next create tasks $t_{ad}, t_{ab}, t_{bc}, t_{ba}, t_{cb}, t_{cd}, t_{dc}, t_{da}$ and add the edges $(t_a \prec t'_a); (t_{ab} \prec t_{b_a}); (t_b \prec t'_b); (t_{bc} \prec t_{cb}); (t_c \prec t'_c); (t_{cd} \prec t_{dc}); (t_d \prec t'_d); (t_{da} \prec t_{ad})$. Finally we partition the task set as follows:

$$T_a = \{t_a, t_{ad}\}; T_{a'} = \{t'_a, t_{ab}\}; T_b = \{t_b, t_{ba}\}; T_{b'} = \{t'_b, t_{bc}\};$$
$$T_c = \{t_c, t_{cb}\}; T_{c'} = \{t'_c, t_{cd}\}; T_d = \{t_d, t_{dc}\}; T_{d'} = \{t'_d, t_{da}\}.$$

to obtain the plan coordination instance in Figure 3.11(b). Notice that the resulting instance is a linear intra-free instance.

If a solution to the FAS problem instance was the arc (a, b), then the corresponding coordination set would be the arc $(t_b \prec t_{ba})$. ■

3.3.2 Special linear intra-free (SLIF) instances

We now know that linear intra-free instances cannot be efficiently plan coordinated and therefore, we seek subclasses of linear intra-free instances which can be coordinated efficiently. The search is motivated by the fact that in real life, situations such as the single product supply chain are coordinated quite easily.

In order to facilitate our discussion further, let us first classify task chains into two types — imitating and contradicting chains. Given an ordering of agents, an *imitating chain* is always pairwise consistent with the ordering of agents. On the other hand, if a chain has exactly the reverse order for each pair of agents, then we call it a *contradicting chain.*

Definition 3.8. Imitating and contradicting chains: Suppose, we are given a linear intra-free instance $\Pi = \langle \{T_i\}_{i=1}^n, \prec, linear \rangle$ and a total ordering $<$ on the set of agents A. If a pair of agents satisfy $A' < A''$ then each pair of tasks $t_i \in T'$ and $t_j \in T''$ such that $t_i \prec t_j$ belong to an *imitating* chain. On the other hand, if $t_j \prec t_i$ then they belong to a *contradicting* chain. ■

Notice that the supply chain scenario presented in Example 3.10, satisfies this special property that all chains are either imitating or contradicting chains. We call such linear intra-free instances as *special linear intra-free instances (*SLIF*)*. Formally then, a SLIF instance can be defined as follows:

Definition 3.9. Special linear intra-free instance (SLIF**):** A given linear intra-free instance $\Pi = \langle \{T_i\}_{i=1}^{n}, \prec, linear \rangle$ is a SLIF instance, if all the chains in Π are either imitating chains or contradicting chains. Because we also require to know the agent ordering, SLIF instances are represented as $\Pi = \langle \{T_i\}_{i=1}^{n}, \prec, < \rangle$. ■

Example 3.12. To illustrate a SLIF instance, consider the scenario in Figures 3.12(a) and (b). The order of agents is reflected in the indices of the agents in both figures. In Figure 3.12 (a) the single imitating chain comprises tasks t_1, t_3, and t_5 allocated respectively to agents A_1, A_2 and A_3. Whereas, the contradicting chain comprises of tasks t_6, t_4 and t_2 allocated to agents A_3, A_2 and A_1 respectively. In Figure 3.12 (b), the imitating chain has t_1, t_3, t_5, t_7 and the contradicting chain has tasks t_8, t_6, t_4, t_2. On the other hand in Figure 3.12 (c), the task chain $t_3 \prec t_5 \prec t_1$ is not pairwise consistent with the indices of the agents and hence is not an imitating chain. Similarly, the task chain $t_4 \prec t_2 \prec t_6$ is not a contradicting chain for the same reason. ■

Let us now proceed to develop a procedure to coordinate SLIF instances. Suppose, the coordination instance has a single imitating chain and a single contradicting chain. If the two chains have $n > 1$ agents in common, then we would require $n - 1$ constraints to stop all cycles. This is because, if we let any two of these agents unconstrained, a simple cycle can be formed between their tasks. The result of adding coordination arcs on the scenario in Figure 3.12 (a) is shown in Figure 3.13.

This basic idea of letting only a single agent free of constraints can be extended to handle multiple pairs of imitating and contradicting chains. Suppose, there are k imitating and l contradicting chains in the task graph such that each chain involves the same set of agents. In this case, we would have to add coordination constraints between $k \times l$ pairs of chains. Suppose each chain has m agents, then we would have to constrain at least $m - 1$ agents for each pair. Thus, we would in total require $(m - 1) \times k \times l$ constraints to coordinate the problem.

Now suppose that the imitating chains and the contradicting chains are all of different lengths, and also involve different subsets of agents. In this case the only

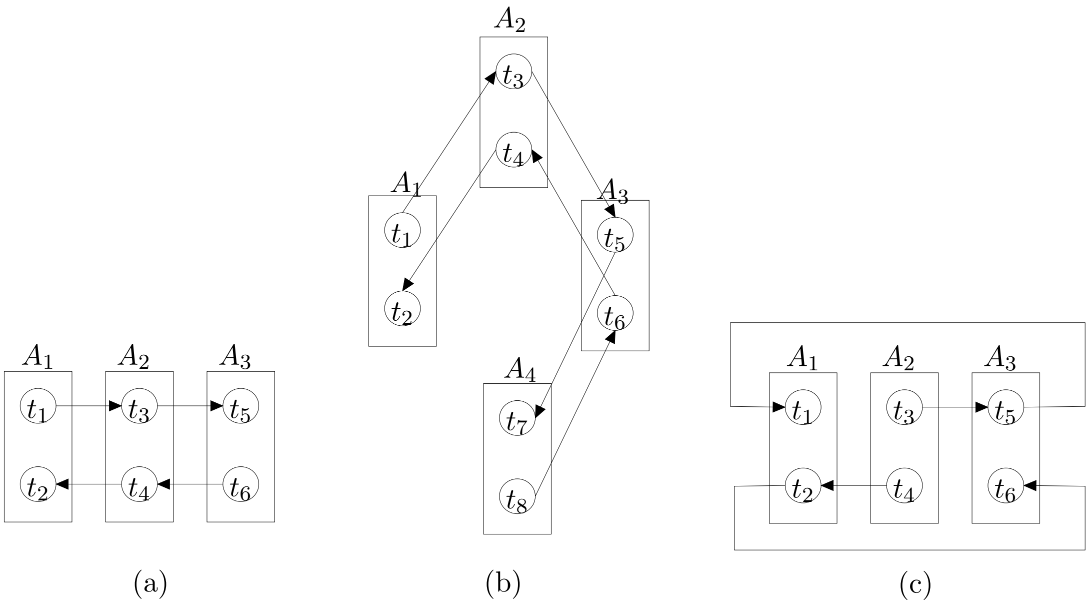


Figure 3.12: Imitating and contradicting chains.

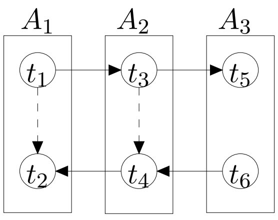


Figure 3.13: Constructing the coordination set for SLIF instances.

change is that, instead of adding $m - 1$ constraints, one would require to compute the number of agents that are common to the imitating and contradicting chains and add constraints between their tasks. Suppose there are x such agents, then one would require $x - 1$ constraints to coordinate those two chains.

Generalising further, if we had k such pairs of imitating and contradicting chains, we would have to add $\sum_{i=1}^{k} x_i - 1$ constraints. Here x_i denotes the number of common agents between the i^{th} pair of imitating and contradicting chains. WLOG, we could fix that the agent with the highest index (last in the ordering of agents) among the common agents is always left free. In theory, we could leave any one of the x_i agents free of constraints. However, doing so would require that the direction of precedence constraints be reversed for all agents that follow the agent that was left free. Therefore, for the sake of simplicity we choose to leave the last agent in the ordering free of constraints. Thus, to ensure that a given SLIF instance is coordinated, we would have to first *(i)* find out all the imitating and contradicting chain pairs and *(ii)* add coordination arcs between tasks of common agents of each pair of imitating and contradicting chains (taking care to leave out the highest indexed agent free of constraints). See Algorithm 2 for an overview of this procedure.

Algorithm 2 SLIF coordination algorithm.

```
Require: A SLIF instance $\Pi = \langle \{T_i\}_{i=1}^{n}, \prec, < \rangle$
1: List all imitating chains $I_1, \ldots, I_p$ each of length $l(I_1), \ldots, l(I_p)$.
2: List all contradicting chains $D_1, \ldots, D_q$ with lengths $l(D_1), \ldots, l(D_q)$.
3: for each pair $(I_p, D_q)$ such that they have 2 or more agents in common do
4:     Given the ordering among agents common to $(I_p, D_q)$, let $A_l$ represent the
       last agent according to the ordering;
5:     add a constraint $t_i \prec t_j$ for each $t_i \in I$ and $t_j \in D$ such that $t_i, t_j \in T_i$ and
       $T_i \neq T_l$;
6: end for
```

To illustrate this simple procedure, let us consider an example.

Example 3.13. The task graph in Figure 3.14 has 2 imitating chains and 3 contradicting chains spread over 5 agents. The order of agents is reflected in the indices of the agents in both figures. The pair of chains involving agents A_1, A_2 and A_3 require 2 constraints as shown in the figure and similarly the pair of chains involving agents A_4 and A_5 require a single constraint. Finally, the pair of chains involving A_3 and A_4 require another single constraint. So in all we would require 4 constraints to

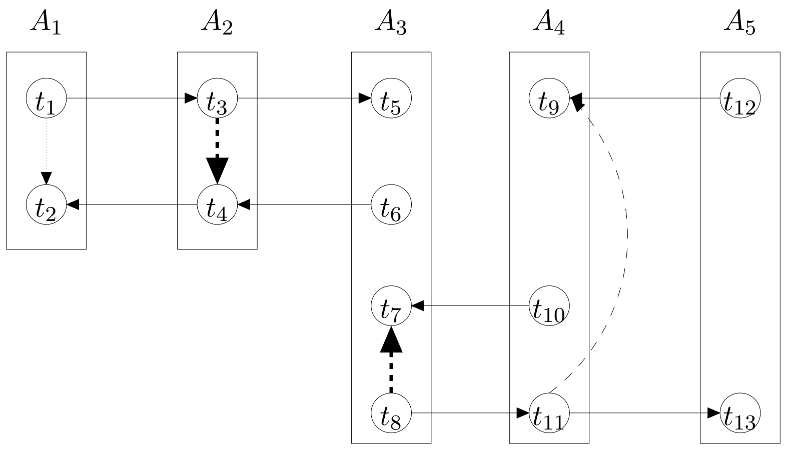


Figure 3.14: Coordinating a special linear intra-free instance.

minimally coordinate this instance. The required constraints are shown as dashed arcs in the figure.

Suppose, we were to employ the depth partitioning approach to solve the above instance. In such a case, we would apply a total of 7 coordination constraints $t_1 \prec t_2; t_6 \prec t_5; t_6 \prec t_7; t_8 \prec t_7; t_{10} \prec t_9; t_{10} \prec t_{11}$ and $t_{12} \prec t_{13}$ to solve the coordination problem. Similarly suppose, we were to use DP^* and the FVS is comprised of agents A_1, A_3 and A_4 we would find a coordination set $t_1 \prec t_2; t_6 \prec t_5; t_6 \prec t_7; t_8 \prec t_7; t_{10} \prec t_9; t_{10} \prec t_{11}$ of size 6. ■

Let us now prove that the SLIF coordination algorithm is correct.

Proposition 3.10. *The SLIF coordination algorithm is correct.*

Proof. Clearly the algorithm terminates, because there are finite number of imitating and contradicting chains in a given instance.

Assume that agent indices reflect the total order on the set of agents. Let us denote the imitating chain as I_p and the contradicting chain as D_q and let the tasks in each of the chains be:

$$I_p = \{t_p^1, t_p^2, \ldots, t_p^l\} \tag{3.1}$$

$$D_q = \{t_q^l, t_q^{l-1}, \ldots, t_q^1\} \tag{3.2}$$

The indices on task t_p^i indicate that the task belongs to the p^{th} chain and A_i. Now let $A_{p,q}$ denote the set of agents that are common to both chains and let l be the highest index for any agent in $A_{p,q}$.

The SLIF coordination algorithm adds

$$t_p^i \prec t_q^i \quad \forall A_i \quad \in \{A_{p,q}\} - \{A_l\} \tag{3.3}$$

If A_l chooses to add the constraint $t_p^l \prec t_q^l$, this in effect would create a total order among the tasks of I_p, D_q. That is, it would imply that

$$t_p^1, t_p^2, \ldots, t_p^l \prec t_q^l, t_q^{l-1}, \ldots, t_q^1 \tag{3.4}$$

Clearly, there can be no cycle between tasks of I_p and D_q if the tasks belonging to them are totally ordered.

On the other hand, if A_l chooses to impose $t_q^l \prec t_p^l$, then because of Algorithm 2 already the tasks of $I_p - t_p^l, D_q - t_q^l$ are totally ordered. Further adding $t_q^l \prec t_p^l$ implies that there t_p^l can never precede t_q^l. Therefore, a directed cycle is again impossible. The same argument holds for every pair of imitating and contradicting chains. Further, we always choose to make tasks in the imitating chain to precede those in a contradicting chain. This means that cycles involving tasks of 3 or more chains also cannot occur. Thus, we claim that the SLIF coordination algorithm results in a sufficient coordination set and hence we claim that the algorithm is correct. □

Having shown that Algorithm 2 results in a sufficient coordination set, let us now proceed to show that this set is also the minimum coordination set.

Proposition 3.11. *The SLIF coordination algorithm results in a minimum coordination set.*

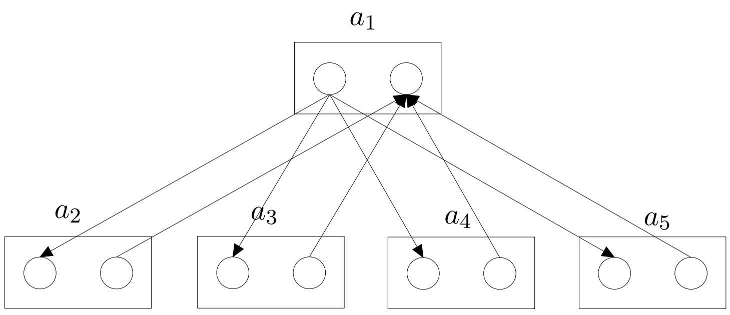


Figure 3.15: Example non-linear scenario.

Proof. To prove the minimality it is enough to notice that for any pair of imitating and contradicting chains, the maximum number of agents that can be left unconstrained is one. Otherwise, a global cycle can be formed between the two agents who are left unconstrained. Thus, the number of constraints resulting from the procedure is the minimum required to ensure coordination. □

Chain structures are relatively easier to analyse. Specially since, between each pair of chains, it does not matter whether we choose A^l to be the highest numbered agent or the lowest numbered agent.

However, the procedure we apply for constructing the minimum coordination set cannot be applied on other kinds of intra-free instances. In fact, it is easy to show counter examples where the procedure used for SLIF instances can backfire. As one such counter example, consider the structure shown in Figure 3.15.

Deciding to leave agent A_1 free of constraints would imply that we would have to constrain all the remaining agents. But, deciding to constrain A_1 would imply that no further coordination constraints need be applied. The difference between these two decisions can be very significant as evidenced by a ratio of nearly $n-1:1$, where n is the number of agents. Thus, an arbitrary choice of which agent to leave free of coordination constraints can lead to a significant improvement or degradation of the result.

Observe that although the scenario in Figure 3.15 served as a counter example, it was still possible to construct an optimal coordination set for the scenario. To find the optimal coordination set for this scenario, it was sufficient to add a coordination constraint between the fork and the join in the figure. *It is still an open problem to decide whether there exist more general structures where this idea of constraining forks and joins leads to optimal coordination sets.*

3.4 Summary

In this chapter, we stated in the introduction that our intention was to search for tractable subclasses of coordination problems. In our search, we first observed that one of the components contributing to the complexity of the general coordination problem is the fact that coordination verification was intractable. But, we observed that some real world problems had simpler structures which could be exploited.

We showed that, the property of intra-freeness allows coordination verification to be done efficiently. Intra-free instances were characterised by agents whose tasks did not have any precedence constraints between them. We showed that, if coordination instances were intra-free, complexity of solving the coordination problem reduces to being NP-complete. To efficiently solve the coordination verification problem for intra-free instances, we showed that, it was enough to determine whether a more compact representation of task graphs — agent dependency graphs — had a cycle.

We proceeded further to investigate a subclass of intra-free instances, where we could also find a minimal coordination set efficiently. Instances of this subclass were termed SLIF instances, and we developed a simple sequence of steps that could efficiently determine an optimal set of coordination constraints. We also showed that, there existed classes of intra-free instances where this procedure can yield arbitrarily bad results.

However, it was noted that finding optimal coordination sets for this class of counter examples also was quite easy. It remains an open problem to determine if there exist more general classes of intra-free instances where a polynomial time procedure can yield an optimal coordination set.

Table 3.1 summarises the various problems we studied in this chapter. The general coordination instance is known to be Σ_2^p-complete whereas we have shown that intra-free instances are NP-complete. Similarly, we have also shown that a specific class of intra-free instances — SLIF instances — can be efficiently coordinated.

Coordination instance	**CVP**	**CP**
General case	co-NP-complete	Σ_2^p complete
Intra-free instances	P	NP-complete
SLIF instances	P	P

Table 3.1: The plan coordination complexity hierarchy.

Currently, in this chapter, we presume that smaller coordination sets are better than larger ones. However, in several real situations, the number of constraints added are of lesser importance than the quality of global plans. For instance, agents might be willing to accept more constraints on their planning for the sake of a more efficient global plan or a more cost effective global plan. In such cases, reducing the number of coordination constraints may not be sufficient. In our next chapter, we design mechanisms to handle scenarios where coordination also comes with a guarantee on the quality of the global solution. Specifically, we design methods that ensure makespan minimality of the global solution.

Chapter 4

Coordinated autonomous scheduling: The unbounded concurrency case

In Chapter 3, we dealt with the plan coordination problem, and as mentioned earlier in Chapter 2, we deal with the schedule coordination problem in this chapter. The objective of this chapter is to find an efficient method to decompose a given CAS *problem instance, when capacity constraints are absent. We rely upon another well known method called the Temporal decomposition method designed by Hunsberger and obtain a faster and simpler method for decomposing* CAS *problems.*

In several systems such as distributed job shops [Kjenstad 1998] or disaster relief operations [Aldunate *et al.* 2006], a global solution that meets *additional criteria* such as makespan minimality or minimal response time is more useful than a global solution that is simply feasible. In fact, it would be better, specially in scenarios similar to disaster management, that tasks are completed as quickly as possible. Motivated by such situations, in this chapter, we focus on ensuring that the global plans are *makespan minimal*, in addition to ensuring feasibility and autonomy.

Clearly, insisting that tasks need to be finished in a finite amount of time does not require temporal information. However, insisting that tasks be completed with minimal makespan does require temporal information. As mentioned in Chapter 2, to enable such a computation, we enriched the planning framework to accommodate temporal information.

While our framework to handle schedule coordination problems is enriched, we still use the same notion of autonomy. We still maintain that, as long as agents are able to design their schedules without needing to interact with one another, autonomy needs are met. Thus in short, we have two quality objectives — *agent autonomy* and *makespan minimality.*

As an illustration of the problems encountered in this chapter consider the following example:

Example 4.1. Consider a simple distributed job shop problem instance, which requires 2 jobs J_1 and J_2 to be processed with minimal makespan. Job J_1 involves three tasks (operations) t_1, t_2 and t_3 that have to be processed by three machines M_1, M_2 and M_3 respectively. Job J_2 has two tasks t_4 and t_5 that need to be processed on machines M_1 and M_2 respectively. Machine M_1 is owned by agent A_1 and similarly machines M_2 and M_3 are owned by agents A_2 and A_3 respectively. Tasks t_1, t_3 and t_5 take 2 units in time and the remaining tasks t_2 and t_4 take a single unit of time to be processed. All tasks t_i have a release time $r(t_i) = 0$ and deadline $d(t_i) = \infty$. The tasks of the job shop problem instance have precedence constraints $t_1 \prec t_2 \prec t_3$ and $t_4 \prec t_5$. As usual, each machine can process a single task at any point in time and each task can be processed by a single machine at any point in time. The job shop problem is shown in Figure 4.2(a).

We have to now develop a mechanism that ensures that every local schedule developed by each of the agents A_1, A_2, and A_3 can always be merged to produce a makespan minimal global schedule. The corresponding task graph is shown in Figure 4.1. ■

A solution to such a problem instance is an assignment (schedule) $\sigma : T \to Z^+$ of values to tasks, meeting all the constraints specified. Intuitively, for each task $t \in T$, $\sigma(t)$ determines its starting time.

In particular, we would like to first ensure that agents can develop local schedules, such that they can be combined into a feasible global schedule, without needing to interact with each other. That is, we would like to specify for each agent a part of the scheduling instance such that each agent A_i determines a schedule σ_i for its

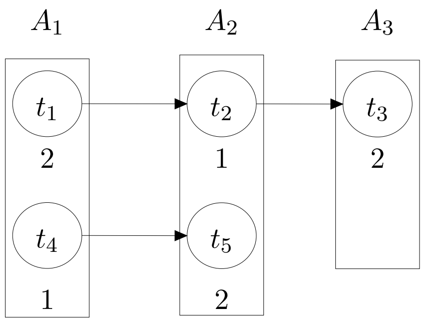


Figure 4.1: Task graph representation of the job shop problem in Example 4.1.

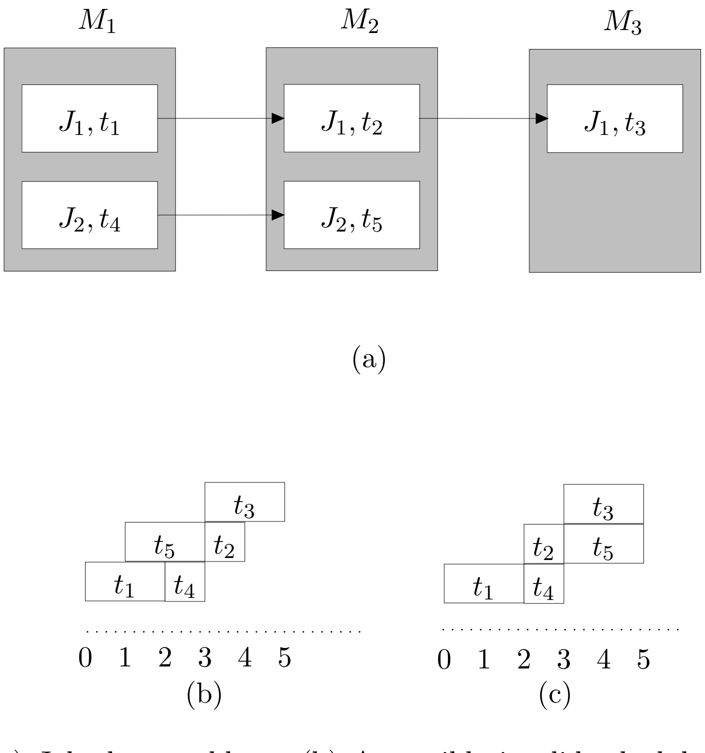


Figure 4.2: (a) Job shop problem. (b) A possible invalid schedule. (c) Optimal schedule.

part of the instance, independently from the others. We further would like to ensure that, the global schedule derived from such local schedules has minimal makespan.

The central problem in schedule coordination is that, allowing full autonomy to agents in finding their own schedule can easily result in a suboptimal or even an infeasible global schedule. As an illustration of this problem, consider the following example:

Example 4.2. Consider the situation shown in Figure 4.2(b), where agents create their own schedules without the mechanism imposing any additional restriction. Machine A_1 schedules task t_1 at time instant 0 and task t_4 at time instant 2. Agent A_2 schedules task t_5 at time instant 1 and task t_2 at time instant 3 and agent A_3 schedules to carry out task t_3 at time instant 3. These schedules of agents A_1, A_2, A_3 are all *locally feasible* (they do not violate any intra-agent constraints). However, in combination they result in a global schedule that is infeasible (because task t_4 is not completed by time instant 1 and task t_2 is not complete by time instant 3). ■

Note that it is not enough to simply ensure that schedules are coordinated. We also require that the makespan is minimum. Therefore, our goal in this chapter is to develop a coordination mechanism that solves the CAS problem but also ensures that the global makespan is minimum. Incorporating makespan minimality, our definition of the CAS problem from Chapter 2, can be restated as follows:

Definition 4.1. CAS problem: Given a scheduling instance $\Pi = \langle T, A, \prec, \phi, l(), c(), r(), d() \rangle$ and any set of locally feasible schedules σ_i chosen by each of the agents A_i for their part $\langle\ T_i, \prec_i, r_i(), d_i(), c(i), l_i()\ \rangle$ of the scheduling instance, the CAS problem is to ensure that the merged schedule $\sigma = \sigma_1 \sqcup \sigma_2 \ldots \sqcup \sigma_n$

- is always a feasible global schedule and
- the makespan of this global schedule is minimal.

■

Henceforth, we always refer to this revised definition of the CAS problem in this thesis. We refer to the scheduling problem faced by each agent as the CAS subproblem and denote it by $\Pi_i = \langle\ T_i, \prec_i, r_i(), d_i(), c(i), l_i()\ \rangle$.

Makespan minimality adds a new dimension to our effort in the sense that often autonomy and makespan minimality are at loggerheads with each other. As an illustration consider the following example:

Example 4.3. Consider another situation similar to the one in Figure 4.2(a), where a mechanism imposes the following additional restrictions to the agents: Agent A_1 has to complete task t_1 before time 2. That is, the deadline $d(t_1)$ for starting task t_1 is 0. Agent 2 receives an additional constraint $r(t_2) = 2$ and $d(t_2) \leq 5$. Finally, agent A_3 receives $d(t_3) = 3$. Then, as shown in Figure 4.2(c), we have a global schedule which is not only feasible but also has minimum makespan. However, in this schedule agents have no autonomy at all. There is a unique schedule that satisfies all (including the additional restrictions) the constraints. ■

What we are aiming for, is to design schedule coordination mechanisms that ensure makespan minimality and maximize autonomy. In this chapter, we will show that we can realize both aims in the special case where agents are not concurrency-bounded. To achieve this goal, we resort to results in the *STN* domain. Specifically, we employ Hunsberger's temporal decoupling algorithm to derive an algorithm that solves CAS problems.

This chapter is organised as follows. We will investigate the CAS problem for the case where concurrency (capacity) constraints for each of the agents are absent. Here, we will derive an efficient algorithm that solves the CAS problem perfectly. This algorithm is derived from Hunsberger's temporal [Hunsberger 2002b] decoupling algorithm. In the process of deriving our algorithm, first, we will show how to specify our CAS problem as a special case of the temporal decoupling problem in *STN*s. Next, we will show that due to the simplified structure of our problem, we can derive a surprisingly simple algorithm achieving schedule coordination as a special case of his temporal decoupling algorithm. This algorithm will also be much more efficient than the original temporal decoupling algorithm. Finally, we will show that our algorithm called the *Interval Scheduling Algorithm* ISA solves the schedule coordination problem while ensuring that agents have maximal autonomy: there is no other algorithm that guarantees more autonomy to the individual agents while ensuring minimal makespan.

However, when agents have bounds on their concurrency, makespan efficiency can no longer be guaranteed. This forms the subject matter of Chapter 5. In Chapter

5, we generalise ISA to handle concurrency constraints. This generalised version guarantees autonomy, but it also shows that the loss in makespan efficiency can be quite high.

4.1 Solving CAS problems through decoupling STNs: the unbounded concurrency case

The CAS problem specification allows agents to return 'any' feasible local schedule: a correct solution to the problem guarantees that, every *locally feasible* solution can be merged into a makespan minimal solution to the *global problem.* If the individual subproblems derived from the global scheduling problem exhibit such a property, we say that such a collection of subproblems constitutes a *decoupling* of the original CAS instance. The CAS problem therefore can be considered as specifying a decoupling problem in the scheduling domain.

Recall that in Chapter 2, we pointed out that in the STN domain, there exists an analogous problem when dealing with the decoupling of Simple temporal network (STN) instances. This corresponding problem is called the Temporal decoupling problem (TDP) and can be solved in polynomial time by applying Hunsberger's algorithm. It is not difficult to see the correspondences between these two problems (STNs were introduced in Section 2.2.3):

- In both problems, there is a set of agents $A = \{A_i\}_{i=1}^{n}$, each responsible for solving a set C of constraints.
- In both problems, there are constraints $c \in C$ that are *coupled* in the sense that, the variables occurring in c belong to different agents A_i; these constraints are called *inter-agent* constraints.
- In both problems, the goal is to achieve a decoupling of the set C of constraints. That is, each agent A_i is able to find a solution for its individual set of constraints C_i in such a way that the solution proposed by individual agents can always be merged to constitute a global solution to the original set C.

The individual sets of constraints C_i constitute the local subproblem of agent A_i. Hunsberger's algorithm succeeds in finding these sets C_i by *tightening* a subset of intra-agent constraints. That is, constraints having only variables belonging to a single agent, such that inter-agent constraints are implied by the local constraints without affecting the consistency of the original problem. This allows each agent A_i to solve its local set of constraints C_i in a completely autonomous fashion, while still ensuring that a global solution can be found by merging the local solutions returned.

Since we are looking for an algorithm that ensures a decoupling of the global CAS problem, such that every schedule of an individual agent can be merged to form a makespan efficient global solution of the original problem, it seems worthwhile to obtain such an algorithm by *reducing* the CAS problem to the TDP. This reduction should have the following properties:

1. Given a CAS problem instance Π an STN $\mathcal{S}_\Pi$ is obtained such that any solution σ of $\mathcal{S}_\Pi$ can be directly related to a solution σ' of Π.

2. Hunsberger's algorithm can be used to decouple the resulting STN $\mathcal{S}_\Pi$ into a set $\{\mathcal{S}_i\}_{i=1}^n$ of decoupled STNs.

3. For each individual $\mathcal{S}_i$ an instance of the CAS subproblem Π_i can be constructed.

4. For every set $\{\sigma_i\}_{i=1}^n$ of individual solutions σ_i of Π_i, their merge $\sigma = \sigma_1 \sqcup \ldots \sqcup \sigma_n$ is a solution to Π. That is, $\{\Pi_i\}_{i=1}^n$ is a decoupling of Π.

So, in this reduction Hunsberger's algorithm is used as a *subroutine* to obtain the set of decomposed STNs which in turn are translated back to a set $\{\Pi_i\}_{i=1}^n$ of CAS subproblems constituting a decomposition of the original CAS instance Π.

We foresee three problems in constructing this reduction.

- The first problem is that, unlike the TDP problem, the CAS problem needs a global schedule with minimum makespan. Although one could argue that in STNs, scheduling every time point variable at its earliest time will always result in a minimum makespan schedule, enforcing such a set of constraints as the result of decoupling would mean that agents are forced to choose a specific assignment of values to the variables. This is equivalent to enforcing a centralised schedule and thus would remove agent autonomy.

As we will show in the next subsection, however, this problem can be easily solved. As a result, it is not difficult to see that any global solution of the STN that can be obtained as the result of merging local solutions, is always a makespan efficient global solution. Consequently, it is also a makespan efficient global solution of the original CAS problem.

- The second problem concerns the use of bounded concurrency constraints in CAS problems. It is well-known that STNs do not allow for encoding bounded concurrency constraints and it seems that the reduction to temporal decoupling approach cannot be used in this case.

  Therefore, before we deal with the bounded concurrency case in the next chapter, we focus on dealing with the simpler problem of encoding CAS problems with unbounded concurrency in this chapter.

- The third problem concerns the overhead caused by performing the decoupling on the STN version of the problem instead of performing it on the CAS instance itself. Since CAS problems are encoded as special STNs, and temporal decoupling requires the rather expensive computation of distance matrices of an STN ($O(n^3)$) and for each temporal decoupling step an incremental update of this matrix ($O(n^2)$ per iteration), it might be that translating the decoupling algorithm directly to the special case of CAS problem instances would simplify the procedure considerably.

  We will show that indeed this is the case and there exists a linear-time algorithm to solve the original CAS problem. Furthermore, as we will show, this algorithm, the so-called *Interval splitting algorithm* (ISA), in some sense is the best we can hope for as a decoupling algorithm ensuring both maximal autonomy and minimal makespan: there is no other algorithm that is able to ensure more autonomy while guaranteeing minimal makespan.

In the next section, first we will present our reduction of CAS instances to STN instances in such a way that the only solutions of the resulting STN are minimal makespan solutions. Then, we will show how Hunsberger's temporal decoupling algorithm can be translated to a decomposition algorithm operating on CAS-instances directly, thereby avoiding the high computation overload.

4.2 Reducing the CAS problem to the temporal decoupling problem

Given a CAS problem instance $\Pi = \langle T, A, \prec, \phi, l(), c(), r(), d() \rangle$ with unbounded concurrency, we reduce it (without minimal makespan constraints) to an *STN* $\mathcal{S} = (\mathcal{T}, \mathcal{C})$ instance as follows:

Step 1 $T = \mathcal{T}$;
That is, each task t_i in the CAS instance is represented as a time point variable t_i denoting the starting time of task t_i. Note that in a CAS problem instance, the completion time of task $t_i \in T$ is determined by $t_i + l(t_i)$. In the subsequent discussion t_i will denote either the task or the starting time of the task. We hope context will provide enough support to distinguish between these closely related concepts.

Step 2 Add a constraint $t_i - t_j \leq -l(t_i) \in \mathcal{C}$, for every precedence constraint $t_i \prec t_j$ in Π;
That is, for each precedence constraint $t_i \prec t_j$ we ensure that the starting time of t_j minus the starting time of t_i is at least equal to the duration of t_i. Note that in an *STN*, all constraints are difference constraints upper bounded by a constant. Hence, we have to express $t_j - t_i \geq l(t_i)$ as $t_i - t_j \leq -l(t_i)$.

Step 3 Add a new variable z (the time reference point) to $\mathcal{T}$. This variable has a fixed value 0.

Step 4 For each $t_i \in T$, add the constraints $z - t_i \leq -r(t_i)$ and $t_i - z \leq d(t_i)$ expressing the meaning of release times and deadlines, respectively. Again, note that the deadline $d(t_i)$ of a task specifies the last moment in time t_i might start. Note that we assume for every $t_i \in T$, the release time $r(t_i)$ satisfies $r(t_i) \geq 0$. The deadline of a task might be a finite number or take the value ∞.

The following observation is a straight-forward consequence of the easy reduction stated above.

Observation 4.1. Let $\Pi = \langle T, A, \prec, \phi, l(), c(), r(), d() \rangle$ be a CAS problem instance and $\mathcal{S} = (\mathcal{T}, \mathcal{C})$ the STN obtained from Π by applying the reduction above. Then any solution σ to $\mathcal{S} = (\mathcal{T}, \mathcal{C})$ is also a solution to Π and vice-versa.

This straight forward reduction, however, does not guarantee *makespan minimality* of a global solution. To solve the problem of makespan minimality, we resort to results from [Dechter *et al.* 1991]. Using our notation this result can be stated as follows:

Observation 4.2. Let D be the distance matrix (we introduced distance matrices in section 2.2.3) belonging to a consistent STN $\mathcal{S} = (\mathcal{T}, \mathcal{C})$. Then, for every $t \in \mathcal{T}$, the earliest possible time t can be executed is given by $\sigma_0(t) = -D[t, z]$ and the latest possible time t can be executed is given by $\sigma_1(t) = D[z, t]$.

Hence, we have the following easy consequence:

Observation 4.3. If D is the distance matrix of the STN $\mathcal{S} = (\mathcal{T}, \mathcal{C})$ derived from a CAS instance Π, then

- the earliest starting time of a task $t_i \in T$ equals $-D[t_i, z]$;
- the earliest possible completion time of all the tasks $t \in T$ is given by

$$m = \max_{t_i \in \mathcal{T}} \{-D[t_i, z] + l(t_i)\}$$

. So m is the makespan of Π.

This observation implies the following result:

Proposition 4.1. *Let* $\Pi = \langle T, A, \prec, \phi, l(), c(), r(), d() \rangle$ *be a consistent* CAS *problem instance. Further, let D be the distance matrix of STN $\mathcal{S} = (\mathcal{T}, \mathcal{C}) = (\mathcal{T}, \mathcal{C})$ derived from* Π *and let* $m = \max_{t_i \in \mathcal{T}} \{-D[t_i, z] + l(t_i)\}$.
Finally, let $\mathcal{S}_\Pi = (\mathcal{T}, \mathcal{C} \cup \{t_i - z \leq m - l(t_i) : t_i \in \mathcal{T}\})$. *Then*

1. $\mathcal{S}_\Pi$ *is consistent;*
2. *Every solution* σ *of* $\mathcal{S}_\Pi$ *is a minimal makespan solution for* Π.

Proof. Given a consistent STN $\mathcal{S} = (\mathcal{T}, \mathcal{C})$, the STN $\mathcal{S}_\Pi = (\mathcal{T}, \mathcal{C} \cup \{t_i - z \leq m - l(t_i) : t_i \in \mathcal{T}\}$ is consistent, too, since at least the earliest possible time solution σ_0 for $\mathcal{S}$ satisfies all the additional (makespan) constraints.
The second condition holds since every solution σ assigns to every task $t_i \in T$ a starting time such that $\sigma(t_i) \leq m - l(t_i)$, ensuring that every task is completed before or at the minimal makespan m. □

Therefore, we can ensure that all solutions to $\mathcal{S} = (\mathcal{T}, \mathcal{C})$ are makespan minimal solutions if we perform the following additional steps:

Step 5 Construct the distance matrix D belonging to $\mathcal{S} = (\mathcal{T}, \mathcal{C})$ and let $m = \max\limits_{t_i \in \mathcal{T}} \{-D[t_i, z] + l(t_i)\}$.

Step 6 For each $t_i \in T$ such that t_i is a $\prec$-maximal element of T, add a constraint $t_i - z \leq m - l(t_i) \in \mathcal{C}$. That is, for each successor-free task t_i task, add a constraint binding its maximum completion time value to $m - l(t_i)$. We will refer to these constraints as *makespan minimising constraints* in future references to them.

Example 4.4. Consider the CAS problem instance Π introduced in Example 4.1 again. Its task graph representation has been given in Figure 4.1. By applying the reduction steps stated above, the following STN $\mathcal{S}_\Pi = (\mathcal{T}, \mathcal{C})$ is obtained:

$$\begin{array}{lllll}
\mathcal{T} & = \{ & t_1, t_2, t_3, t_4, t_5, z \ \} & & \\
\mathcal{C} & = \{ & -2 \leq t_5 - t_1 \leq 2; & 0 \leq t_2 - t_1 \leq 4; & \\
 & & 0 \leq t_3 - t_2 \leq 3; & 0 \leq t_5 - t_4 \leq 3; & \\
 & & 0 \leq t_1 - z \leq 0; & 0 \leq t_4 - z \leq 0; & \\
 & & 0 \leq t_3 - z \leq 5; & t_5 - z \leq 5 & \}
\end{array}$$

The distance graph of the STN is shown in Figure 4.3 and the distance matrix is shown in Table 4.1.

■

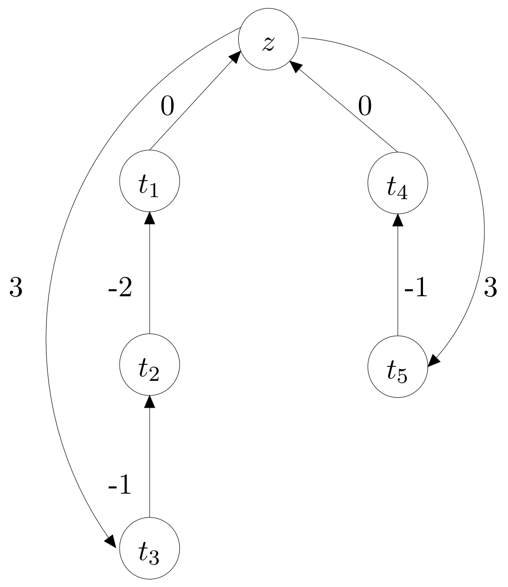


Figure 4.3: The distance graph associated with $\mathcal{S} = (\mathcal{T}, \mathcal{C})$.

Table 4.1: Distance matrix of $\mathcal{S} = (\mathcal{T}, \mathcal{C})$ given in Figure 4.3.

	z	t_1	t_2	t_3	t_4	t_5
z	0	0	2	3	2	3
t_1	0	0	2	3	2	3
t_2	-2	-2	0	1	0	1
t_3	-3	-3	-1	0	-1	0
t_4	0	0	2	3	0	3
t_5	-1	-1	1	2	-1	0

4.3 Solving the CAS problem by temporal decoupling

Given the reduction from a CAS problem instance Π to $\mathcal{S}_\Pi$, we can apply Hunsberger's temporal decoupling algorithm to the latter using the partitioning of the tasks as induced by the task assignment function ϕ.

The result of applying the Temporal decoupling algorithm on $\mathcal{S}_\Pi$ will be an STN $\mathcal{S}^*_\Pi$ such that the entries $D[z,t]$ and $D[t,z]$ are modified for all time points t involved in inter-agent constraints and all inter-agent constraints are implied by the local constraints.

From this STN $\mathcal{S}^*_\Pi$ we can easily assemble the individual (local) scheduling instances Π_i belonging to agent A_i by specializing all the elements of Π to i and updating the release times and deadlines for each task $t \in T_i$ by using the entries in the distance matrix belonging to $\mathcal{S}^*_\Pi$.

Summarizing, the following algorithm (see Algorithm 3) gives a high-level overview of solving the CAS problem by reduction to the TDP.

Algorithm 3 Solving the CAS problem by reduction to TDP

```
Require: A CAS problem instance
    $\Pi = \langle T, A, \prec, \phi, l(), c(), r(), d() \rangle$;
Ensure: A decoupling $\{\Pi_i\}_{i=1}^n$ of $\Pi$;
 1: Construct $\mathcal{S} = (\mathcal{T}, \mathcal{C})$ from $\Pi$ applying the reduction;
 2: Perform the TD algorithm to obtain a decoupled $STN$ $\mathcal{S}^*_\Pi$, let $D$ denote the
    distance matrix of such a decoupled $STN$ $\mathcal{S}^*_\Pi$;
 3: for all $A_i \in A$ do
 4:     Let $\Pi_i$ be the restriction of $\Pi$ to $i$;
 5:     for all $t \in T_i$ do
 6:         set $d_i(t) := \min\{d_i(t), D[z,t]\}$;
 7:         set $r_i(t) := \max\{r_i(t), -D[t,z]\}$;
 8:     end for
 9: end for
10: return $\{\Pi_i\}_{i=1}^n$
```

It is not difficult to see that this algorithm is correct: Note that if we consider an instance Π' composed of all the instances Π_i created by the algorithm from Π and reduced this instance to an STN using the reduction specified before, we obtain the STN $\mathcal{S}^*_\Pi$ which is the result of decoupling $\mathcal{S} = (\mathcal{T}, \mathcal{C})$. Then, using the fact that $\mathcal{S}^*_\Pi$ is a temporal decoupling of $\mathcal{S} = (\mathcal{T}, \mathcal{C})$ and the correspondence between Π and $\mathcal{S} = (\mathcal{T}, \mathcal{C})$, it follows that for every set of solutions $\{\sigma_i\}_{i=1}^n$ where $\sigma_i \models \Pi_i$ it holds that their merge σ is a model of $\mathcal{S} = (\mathcal{T}, \mathcal{C})$, and therefore $\sigma \models \Pi$. Therefore,

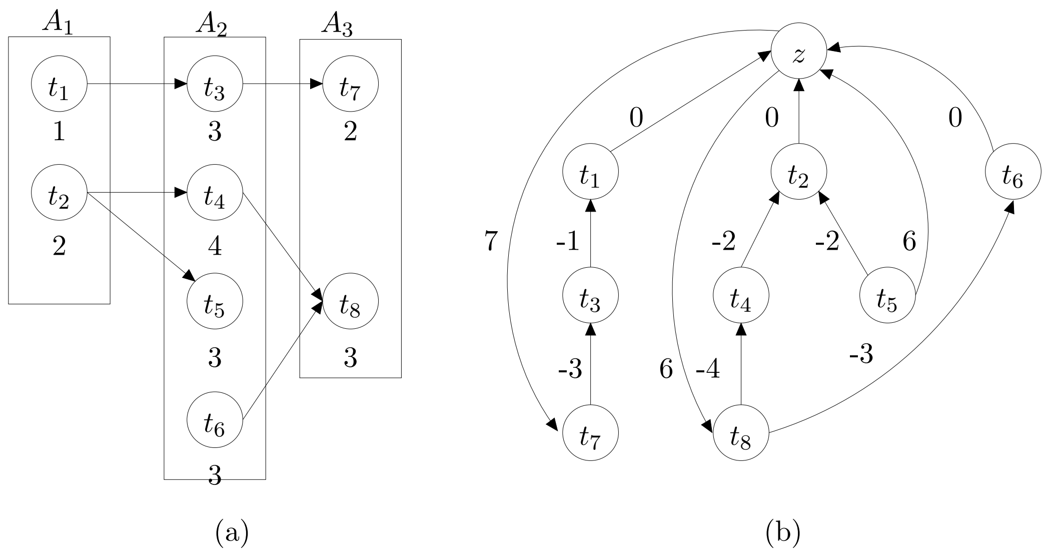


Figure 4.4: (a) An example task graph (b) the distance graph of the corresponding makespan minimal STN.

$\{\Pi_i\}_{i=1}^n$ is a decoupling of Π and we have the following result.

Proposition 4.2. *Given a* CAS *problem instance* Π*, Algorithm 3 computes a temporal decoupling* $\{\Pi\}_{i=1}^n$ *of* Π.

Example 4.5. Consider the task graph shown in Figure 4.4 (a). 8 tasks have been allocated among three agents and their durations are shown below them. The corresponding STN $\mathcal{S} = (\mathcal{T}, \mathcal{C})$ is shown in Figure 4.4 (b). The distance matrix of the same is given in Table 4.2.

There are three inter-partition edges where there is a 'slack'. This slack, termed

	z	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
z	0	3	0	4	2	6	3	7	6
t_1	0	0	0	4	2	6	3	7	6
t_2	0	3	0	4	2	6	3	7	6
t_3	-1	-1	-1	0	3	5	2	6	5
t_4	-2	1	-2	2	0	4	1	5	4
t_5	-2	1	-2	2	0	0	1	5	4
t_6	0	3	0	4	2	6	0	7	6
t_7	-4	-4	-4	-3	-2	2	-1	0	2
t_8	-6	-3	-6	-2	-4	0	-3	1	0

Table 4.2: Distance matrix for the $\mathcal{S} = (\mathcal{T}, \mathcal{C})$ in Figure 4.4(b).

the *Zero path shortfall* (ZPS) value[1] allows us to identify edges that can be tightened:

$$\begin{aligned} ZPS(t_1, t_3) =& 0 + 4 - 1 = 3 \\ ZPS(t_3, t_7) =& -1 + 7 - 3 = 3 \\ ZPS(t_6, t_8) =& 0 + 6 - 3 = 3 \end{aligned}$$

We first choose to decouple the edge (t_3, t_1). Thus, we need to find δ_1, δ_2 such that

$$\begin{aligned} -D[z, t_1] \leq \delta_1 \leq D[t_1, z] \\ -D[t_3, z] \leq \delta_2 \leq D[z, t_3] \end{aligned}$$

Substituting values we get,

$$\begin{aligned} -3 \leq \delta_1 \leq 0 \\ 1 \leq \delta_2 \leq 4 \end{aligned}$$

We choose to set $\delta_1 = -3$ and $\delta_2 = 4$. This results in two additional constraints: $z - t_1 \leq -3$ and $t_3 - z \leq 4$. Updating the distance matrix, we observe that edges (t_7, t_3) and (t_8, t_6) have a ZPS value greater than 0. After decoupling them as earlier, we derive the constraints: $z - t_3 \leq -4$; $t_7 - z \leq 7$; $z - t_6 \leq 0$; $t_8 - z \leq 6$. The

	z	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
z	0	3	0	4	2	6	3	7	6
t_1	0	0	0	4	2	6	3	7	6
t_2	0	3	0	4	2	6	3	7	6
t_3	-4	-1	-4	0	-2	2	-1	3	2
t_4	-2	1	-2	2	0	4	1	5	4
t_5	-2	1	-2	2	0	0	1	5	4
t_6	-3	3	0	4	2	6	0	7	6
t_7	-7	-4	-4	-3	-2	2	-1	0	2
t_8	-6	-3	-6	-2	-4	0	-3	1	0

Table 4.3: Distance matrix after decomposition.

final distance matrix is given in Table 4.3. The release dates and deadlines for each of the tasks in Π derived from Table 4.3 are as follows:

$$\begin{array}{ll}
r_1(t_1) = 0; & d_1(t_1) = 3 \\
r_1(t_2) = 0; & d_1(t_2) = 0 \\
r_2(t_3) = 4; & d_2(t_3) = 4 \\
r_2(t_4) = 2; & d_2(t_4) = 2 \\
r_2(t_5) = 2; & d_2(t_5) = 6 \\
r_2(t_6) = 0; & d_2(t_6) = 3 \\
r_3(t_7) = 7; & d_3(t_7) = 7 \\
r_3(t_8) = 6; & d_3(t_8) = 6
\end{array}$$

Updating the distance graph based on the new set of release times and deadlines we obtain the graph in Figure 4.5. Note that $r_i(t') \geq d_j(t) + l(t)$ whenever $t \prec t'$. Therefore, as long as the task is scheduled within its release time and deadline interval, there is no danger of any precedence constraint being violated. To illustrate

[1]ZPS of a proper edge $E : (t_j - t_i \leq \delta)$ is given by $ZPS(E) = D(t_i, z) + D(z, t_j) - \delta$

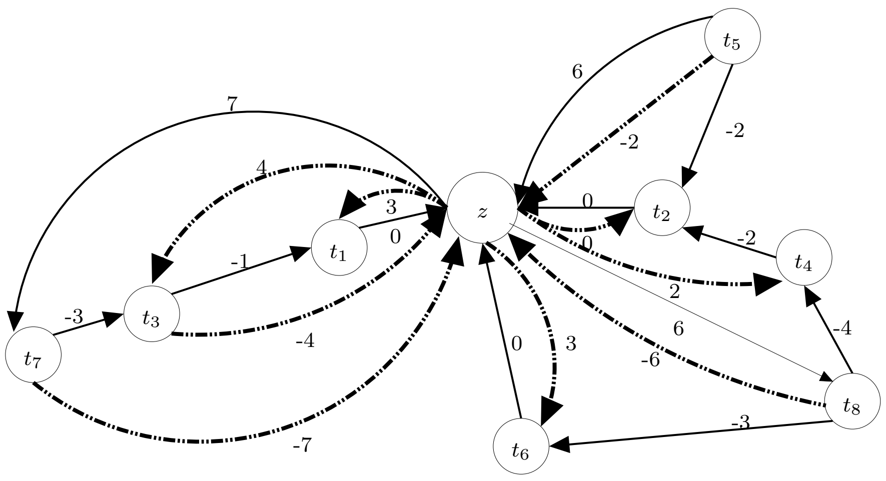


Figure 4.5: Updated distance graph. The darker, broken lines represent the updated release times and deadlines.

that, now suppose the three agents design the following three schedules:

$$\begin{array}{ll} \sigma_1(t_1) = 3 & \sigma_1(t_2) = 0; \\ \sigma_2(t_3) = 4 & \sigma_2(t_4) = 2; \\ \sigma_2(t_5) = 4 & \sigma_2(t_6) = 3; \\ \sigma_3(t_7) = 7 & \sigma_3(t_8) = 6 \end{array}$$

It is easy to see that these local schedules can be merged to obtain a valid global schedule because they do not violate any precedence constraint when merged. Further, since the deadlines do not allow for a schedule of length greater than 9. It follows that all schedules that honour the updated release times and deadlines can be merged to obtain a global schedule with a minimum makespan (which in this case is 9). ■

As mentioned before, using this temporal decoupling method requires several rather expensive computations: the distance matrix D has to be constructed (costs $O(n^3)$ where n is the number of time points in the STN) and furthermore D has to be updated after each temporal decoupling step in which an inter-agent constraint is removed (costs $O(n^2)$ per step). Therefore, we are looking for a simplification of the temporal decoupling method to solve our original CAS problem. This simplification will be accomplished stepwise, first using the translation of CAS instances to STN instances and then will be applied directly to the CAS instance.

4.3.1 Towards an efficient temporal decoupling algorithm for the CAS problem

Using Algorithm 3, requires us to apply Hunsberger's temporal decoupling algorithm. This in turn requires us to compute the distance matrix D of an STN $\mathcal{S}$. As we will show in this subsection, this computation is not needed because of the following reasons:

1. Only a small part of the distance matrix for the derived STN instance $\mathcal{S}_\Pi$ is needed to perform a decoupling of $\mathcal{S}_\Pi$;
2. This part can be computed directly by applying a simple linear algorithm to the CAS instance Π;
3. The necessary tightening of constraints to remove inter-agent constraints (the temporal decoupling steps) can be performed without recomputing other parts of the distance matrix D and requires linear time;
4. The result of the temporal decoupling can be represented directly in the form of individual CAS instances, being the result of decomposition of the original CAS instance.

As a result, we will construct an algorithm doing the same job as Algorithm 3, but not relying on a reduction to $\mathcal{S}_\Pi$ and performing the decoupling in linear time.

Computing earliest and latest starting times

We start with a derivation of the earliest and latest starting times for the tasks $t \in T$. Given a CAS problem instance Π, as a consequence of Observation 4.2, we know that the earliest and latest possible starting times of a task $t \in T$ can be obtained from the first row and first column of the distance matrix D belonging to its corresponding STN $\mathcal{S}_\Pi$. We will now show how the first row and the first column of the distance matrix D associated with $\mathcal{S}_\Pi$ can be computed in *linear time* without using the reduction to STNs.

Given a task t we define the set of predecessors and the set of successors of t as follows:

Definition 4.2. Predecessor and successor of task t: Given a CAS problem instance $\Pi = \langle T, A, \prec, \phi, l(), c(), r(), d() \rangle$, let $\ll$ denote the transitive reduction of T.[2] Then, the set $pred(t) = \{t' \mid t' \ll t\}$ is the set of all predecessor tasks of tasks t and $succ(t) = \{t' \mid t \ll t'\}$ denotes the set of all successor tasks of t. ■

Denoting the earliest starting time of a task t by $est(t)$ and the latest starting time of t by $lst(t)$, we have the following result:

Observation 4.4. For every task $t \in T$ it holds that

$$est(t) = \max\{r(t), \max_{t' \in pred(t)} \{est(t') + l(t')\}\} \tag{4.1}$$

Proof. By easy induction on the depth of task $t \in T$ w.r.t. $\prec$: clearly if $pred(t) = \emptyset$, the earliest starting time of t equals $r(t) \geq 0$. Proceeding inductively, assume that all tasks with depth k or smaller have been assigned their earliest starting time. Consider a task at depth $k+1$ and its set of predecessors $pred(t)$. Every task $t' \in pred(t)$ has depth k. Hence, $est(t)$ equals the maximum of $r(t)$ and the maximum of all earliest finishing times $est(t') + l(t')$ of tasks $t' \in pred(t)$. □

Note that the minimum makespan m of the set of tasks T equals the minimum duration required to complete all tasks. Hence, we immediately have

Observation 4.5. $m = \max_{t \in T}\{est(t) + l(t)\}$

Analogously, the latest starting times $lst(t)$ of tasks t can be determined as follows:

[2] $t \ll t'$ if there exist no $t'' \in T$ such that both $t \prec t''$ and $t'' \prec t'$.

Observation 4.6. For every task $t \in T$ it holds that

$$lst(t) = \min\{\ \min\{d(t), m - l(t)\}, \min_{t' \in succ(t)} \{lst(t') - l(t)\}\ \} \tag{4.2}$$

Proof. Analogously to the proof of Observation 4.4, but now with induction on the height of task t in the ordering $\prec$ of T, and noticing that the latest starting time of tasks without $\prec$-successor in T is determined by $m - l(t)$ and $d(t)$. □

Computing *est* and *lst* values enable us to easily check whether a given CAS problem instance Π is feasible. That is, checking whether there exists at least one solution σ satisfying Π.

Proposition 4.3. *Any given* $\Pi = \langle T, A, \prec, \phi, l(), c(), r(), d() \rangle$ *is feasible iff, for every* $t \in T$, *we have* $lst(t) - est(t) \geq 0$.

Proof. It suffices to show that if for every $t \in T$ we have $lst(t) - est(t) \geq 0$, there exists a solution σ satisfying all constraints. Consider the smallest assignment $\sigma_0(t) = est(t) \leq lst(t)$ for all $t \in T$. It is easy to see that this assignment satisfies all constraints of Π since for all $t \prec t'$ we have $\sigma_0(t) + l(t) \leq \sigma_0(t')$ by definition of $est(t)$. Moreover, $\sigma_0(t) \geq r(t)$ and $\sigma_0(t) \leq d(t)$ by definition of $est(t)$ and $lst(t)$ and the fact that $est(t) \leq lst(t)$. □

The (inductive) definitions of $est(t)$ and $lst(t)$ can be used to build easy algorithms to find the earliest and latest starting times of the tasks as they appear in the first row and column of the matrix D belonging to $\mathcal{S}_\Pi$ as discussed before:

Proposition 4.4. *The first row and column entries of the distance matrix D belonging to $\mathcal{S}_\Pi$ for a given* CAS *problem instance* $\Pi = \langle T, A, \prec, \phi, l(), c(), r(), d() \rangle$ *can be computed in linear time.*

Proof. Note that the values $est(t)$ and $lst(t)$ can be computed in linear time using a topological sort of T [Cormen *et al.* 1990]. Using the fact that the entries $-D[t_i, z]$ and $D[z, t_i]$ in the distance matrix D associated with the *STN* $\mathcal{S}_\Pi$ specify the smallest and largest makespan solutions for $t \in T$, respectively, we immediately derive that

1. $-D[t, z] = est(t)$;
2. $D[z, t] = lst(t)$.

Hence, the first row and column entries of the distance matrix D belonging to $\mathcal{S}_\Pi$ can be computed in linear time. $\square$

Incremental solvability and solution interval sets

It is well-known that the distance matrix D of a (consistent) STN $\mathcal{S}$ exhibits an important property: *incremental solvability.*[3] In its simplest form this means that for every variable t_i and for every value $v \in [-D[t_i, z], D[z, t_i]\ ]$, there exists a *complete* solution σ such that $\sigma(t_i) = v$, i.e., every value in the interval $[-D[t_i, z], D[z, t_i]\ ]$ can be chosen to extend a partial solution $\{t_i = v\}$ to a complete solution σ for $\mathcal{S}$.

As we mentioned before, our goal is to specialize the temporal decoupling method to CAS problem instances avoiding a costly reduction to STNs. As a further step, we now specify this property of incremental solvability w.r.t. the set $I_\Pi = \{[est(t), lst(t)] : t \in T\}$ of intervals for a given CAS instance Π as follows:

Definition 4.3. Solution interval set: Given a CAS problem instance $\Pi = \langle T, A, \prec, \phi, l(), c(), r(), d()\rangle$, we say that a set $I = \{[v_i, w_i\ ] : t_i \in T\}$ is a *solution interval set* for Π if there exists a solution σ for Π such that for every $t_i \in T$ it holds that $v_i \leq \sigma(t_i) \leq w_i$.
$\blacksquare$

Definition 4.4. Incremental solvability: We say that a solution interval set I allows for incremental solvability if, for every $t_i \in T$ and every choice of $\sigma(t_i) \in [v_i, w_i\ ]$ it holds that there exists a complete solution σ extending $\sigma(t_i)$, such that for every $t_j \in T$, we have $v_j \leq \sigma(t_j) \leq w_j$. $\blacksquare$

From the definitions of solution set and incremental solvability, it is immediately evident that a solution interval set $I_\Pi = \{[est(t_i), lst(t_i)] : t_i \in T\}$ allows for incremental solvability:

[3]This property is also called backtrack-free solvability.

Proposition 4.5. *Let $\mathcal{S}_\Pi$ be an STN derived from a* CAS *instance* $\Pi = \langle T, A, \prec, \phi, l(), c(), r(), d() \rangle$ *and let $est()$ and $lst()$ be the functions assigning the earliest starting time and the latest starting time for tasks in T, respectively. Then the solution interval set $I_\Pi = \{ [\, est(t_i), lst(t_i)\,] : t_i \in T\}$ is a solution interval set for* Π *allowing for incremental solvability.*

Using this idea of a solution interval set I for a CAS problem instance Π allowing for incremental solvability, we will now try to reformulate the TDP algorithm as specified by Hunsberger in such a way that decoupling is performed operating on solution interval sets instead of using the distance matrix D_Π associated with $\mathcal{S}_\Pi$.

Specifying temporal decoupling step without using the distance matrix D

Basically, Hunsberger's Temporal decoupling method for STNs consists of a series of temporal decoupling steps. In each such step, the first row and first column entries of the distance matrix D are adapted in order to ensure that, for a given pair of temporal variables t_i and t_j belonging to different agents, it holds that the inter-agent constraints they participate in are implied by the first row and first column entries they participate in. That is, it is ensured that the entries $D[t_i, z], D[t_j, z], D[z, t_i]$ and $D[z, t_j]$ (specifying the lower and upper bound for local constraints) are tightened in such a way that

$$D[t_i, z] + D[z, t_j] \leq D[t_i, t_j]$$

and

$$D[t_j, z] + D[z, t_i] \leq D[t_j, t_i].$$

At the end of each such a step, we should update other entries of D as a consequence of changing $D[t_i, z]$ and $D[z, t_i]$ in order to ensure that the modified distance matrix D' enjoys incremental solvability when starting the next step.

In our case, using the before mentioned equalities $D[t_i, z] = -est(t_i)$ and $D[z, t_j] = lst(t_j)$, the temporal decoupling step stated above simplifies to the following:

1. Adapt the values $est(t_i), lst(t_i), est(t_j)$ and $lst(t_j)$ ensuring that both $lst(t_j) - est(t_i) \leq D[t_i, t_j]$ and $lst(t_i) - est(t_j) \leq D[t_j, t_i]$;

2. Adapt the entries of D as the consequence of these local changes to ensure that the modified matrix D' enjoys incremental solvability.

Note that these equations require the values $D[t_i, t_j]$ and $D[t_j, t_i]$ to be known in order to specify whether or not these inequalities are violated or not. Of course, we would like to remove the dependency on these entries in executing a temporal decoupling step. Therefore, we distinguish two cases:

1. t_i and t_j are not $\prec$-related.
   In this case, the temporal distance $t_j - t_i$ between the two tasks is bounded above by $\max\{lst(t_j) - t(t_i), lst(t_i) - est(t_j)\}$. Hence, $D[t_i, t_j] = \max\{lst(t_j) - est(t_i), lst(t_i) - est(t_j)\}$. Then it follows that the inter-agent constraint $t_j - t_i \leq D[t_i, t_j]$ is always implied by the sum of local constraints:
   $$(t_j - z) + (z - t_i) \leq D[z, t_j] + D[t_i, z] = lst(t_j) - est(t_i) \leq D[t_i, t_j]$$
   of the two intra-agent constraints $t_j - z \leq D[z, t_j]$ and $z - t_i \leq D[t_i, z]$. Hence, the inter-agent constraint is already implied by these local intra-agent constraints. The same holds true for $t_i - t_j \leq D[t_j, t_i]$.
   We conclude that we do not need to bother about temporal decoupling if the tasks t_i and t_j are not $\prec$ related.

2. $t_i \prec t_j$ (or vice-versa).
   In this case, t_i should be completed before t_j starts, i.e., we should have that $t_i - t_j \leq -l(t_i)$, i.e., $D[t_j, t_i] \leq -l(t_i)$. Clearly, if $lst(t_i) + l(t_i) > est(t_j)$, this inter-agent constraint is not implied by the local constraints $t_i \leq lst(t_i)$ and $t_j \geq est(t_j)$. In other words, if this latter inequality holds, the inter-agent constraint will be violated whenever, for example, one agent chooses the latest possible time for starting t_i and the other agent chooses the earliest possible time for t_j. The case for $t_j \prec t_i$ is treated completely analogous to this case.

Summarizing, we conclude that

1. only in case t_i and t_j belong to different agents, and $t_i \prec t_j$, we have to check whether the local constraints imply the inter-agent constraint $t_i - t_j \leq -l(t_i)$,

2. to ensure that this inter-agent constraint is implied means that we have to adapt the values $est(t_i)$ and $lst(t_j)$ in order to ensure that $t_i + t_j \leq -l(t_i)$ is implied and,

3. for checking the decoupling conditions, we can completely remove the dependency upon the entries $D[t_i, t_j]$ in the distance matrix D.

Now it is time to look at the way the changes of the local constraints are performed in a temporal decoupling step to further simplify this step and make its execution completely independent of the entries $D[t_j, t_i]$.

According to the TD method, in executing a temporal decoupling step for t_i and t_j to imply a constraint $t_i - t_j \leq D[t_j, t_i]$, we have

1. to find two values δ_i and δ_j such that $(D[z, t_i] - \delta_i) + (D[t_j, z] - \delta_j) \leq D[t_j, t_i]$ and

2. to add the constraints $t_i - z \leq D[z, t_i] - \delta_i$ and $z - t_j \leq D[t_j, z] - \delta_j$.

Remember that in our case we only have to adapt local constraints in case $t_i \prec t_j$. Suppose that it is true then, we have

1. to find two values δ_i and δ_j such that $(lst(t_i) - \delta_i) - (est(t_j) + \delta_j) \leq -l(t_i)$. By rearranging terms we derive that $\delta_i + \delta_j \geq lst(t_i) + l(t_i) - est(t_j)$;

2. to add the constraints $t_i \leq lst(t_i) - \delta_i$ and $-t_j \leq -est(t_j) - \delta_j$.
   This means that we have to change $lst(t_i)$ to $lst(t_i) + \delta_i$ and to change $est(t_j)$ to $est(t_j) + \delta_j$.

Let us now determine suitable values δ_i and δ_j such that we can change the corresponding $est(t_j)$ and $lst(t_i)$ values.

From the required inequality

$$\delta_i + \delta_j \geq lst(t_i) + l(t_i) - est(t_j)$$

it follows that

1. by setting
$$\delta_i = \delta_j = \frac{lst(t_i) + l(t_i) - est(t_j)}{2}$$

and

2. changing $lst(t_i)$ to $lst(t_i) - \delta_i$ and then $est(t_j)$ to $est(t_j) + \delta_j$

the temporal decoupling step can be performed resulting in the implication/elimination of the inter-agent constraint $t_i + t_j \leq -l(t_i)$.

Algorithm 4 is a first specification of performing a temporal decoupling step without making any reference to the distance matrix D:

Algorithm 4 Performing a temporal decoupling step.

```
Require: An interval solution set $I = \{[est(t_i), lst(t_i)] : t_i \in T\}$, two tasks $t_i, t_j \in T$
    such that $t_i \prec t_j$ belong to different agents, and $lst(t_i) + l(t_i) > est(t_j)$;
Ensure: $est(t_j)$ and $lst(t_i)$ values are modified such that $lst(t_i) + l(t_i) \leq est(t_j)$
    and a new interval solution set $I'$ is obtained
1: let $\delta_i = \frac{lst(t_i)+l(t_i)-est(t_j)}{2}$;
2: let $est(t_j) := est(t_j) + \delta_i$;
3: let $lst(t_i) := lst(t_i) - \delta_i$;
4: return the new interval solution set $I' = \{[est(t_i), lst(t_i)] : t_i \in T\}$.
```

Refining the temporal decoupling step to ensure incremental solvability

It can easily be shown that whenever I is an interval solution set, I' is a solution interval set, too. Moreover, as a result of applying the temporal decoupling step, it is easy to see that for every solution σ based on I' we have that the inter-agent constraint $t_i + t_j \leq -l(t_i)$ is automatically satisfied: the new intervals for t_i and t_j ensure that $\sigma(t_i) + l(t_i) \leq \sigma(t_j)$. There is, however, one catch: I' does not guarantee incremental solvability, whereas I_Π did.

To show the occurrence of this lack of incremental solvability, consider the following example:

Example 4.6. Consider a set of 6 jobs: $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$. It holds that $t_1 \prec t_2 \prec t_3$ and all these jobs have a duration of 3. Furthermore, $t_4 \prec t_5 \prec t_6$ and these jobs have a duration of 1. Computing I_Π, we derive

$$\begin{array}{ll} est(t_1) = 0, & lst(t_1) = 0 \\ est(t_2) = 3, & lst(t_2) = 3 \\ est(t_3) = 6, & lst(t_3) = 6 \\ est(t_4) = 0, & lst(t_4) = 6 \\ est(t_5) = 1, & lst(t_5) = 7 \\ est(t_6) = 2, & lst(t_6) = 8 \end{array}$$

Suppose that t_4 and t_5 belong to different agents. Since we have $lst(t_4) + l(t_4) = 6 + 1 > est(t_5) = 1$, the inter-agent constraint $t_4 + 1 \leq t_5$ is not implied.

Now applying Algorithm 4 to this example with $t_4 \prec t_5$,

1. a value $\delta_4 = \frac{lst(t_4)+l(t_4)-est(t_5)}{2} = \frac{6+1-1}{2} = 3$ is determined
2. $lst(t_4)$ is updated to $6 - 3 = 3$ and $est(t_5)$ to $1 + 3 = 4$.

This results in the following interval solution set I':

$$\begin{array}{ll} est'(t_1) = 0, & lst'(t_1) = 0 \\ est'(t_2) = 3, & lst'(t_2) = 3 \\ est'(t_3) = 6, & lst'(t_3) = 6 \\ est'(t_4) = 0, & lst'(t_4) = 3 \\ est'(t_5) = 4, & lst'(t_5) = 7 \\ est'(t_6) = 2, & lst'(t_6) = 8 \end{array}$$

Although the constraint $t_4 + l(t_4) = t_4 + 1 \leq t_5$ now is implied, the resulting interval solution set I' is no longer incrementally solvable: if we choose $\sigma(t_6) = 2$, there is no solution $\sigma(t_5)$ occurring in the interval $[est'(t_5, lst'(t_5)] = [4, 7]$. ■

The careful reader might have noticed that the decoupling step as performed by the algorithm above results in interval solution sets I' where some properties I, and in particular I_Π are preserved, but others are lost. First of all, the algorithm never returns an interval solution set where intervals are not covered by the input I: such interval solution sets I' are covered by the solution interval set I given as input.

Definition 4.5. Covering: Given an interval solution set I for a CAS problem instance Π we say that an interval solution set $I' = \{[est'(t_i), lst'(t_i)] : t_i \in T\}$ is *covered* by $I = \{[est(t_i), lst(t_i)] : t_i \in T\}$ if, for every $t_i \in T$, we have $est(t_i) \leq est'(t_i) \leq lst'(t_i) \leq lst(t_i)$. ■

A property, however that is not preserved by Algorithm 4 is (est, lst)-monotonicity:

Definition 4.6. Monotonicity: Given an interval solution set I for a CAS problem instance Π we say that an interval solution set $I' = \{[est'(t_i), lst'(t_i)] : t_i \in T\}$ is (est, lst)-monotonic if for every pair $t_i, t_j \in T$ it holds that whenever $t_i \prec t_j$ then $est'(t_i) + l(t_i) \leq est'(t_j)$ and $lst'(t_i) \leq lst'(t_j) - l(t_i)$. ■

Example 4.7. To illustrate the idea of (est, lst)-monotonicity, let us consider Example 4.6 again. It is easy to check that I_Π is (est, lst)-monotonic. If we look at I', however, we see that $est'(t_3) > est'(t_4)$ while $t_3 \prec t_4$ and the same holds for the lst-values. ■

As it turns out both properties of interval solution sets are necessary to guarantee the preservation of incremental solvability:

Proposition 4.6. *Let $I' = \{[est'(t_i), lst'(t_i)] : t_i \in T\}$ be a solution interval set for a* CAS *problem instance Π such that I' is covered by I_Π and is (est, lst)-monotonic. Then I' also guarantees incremental solvability.*

Proof. Consider an arbitrary $t_i \in T$ and a value $\sigma(t_i) \in [est'(t_i), lst'(t_i)]$. Since I' is covered by I_Π, $\sigma(t_i) \in [est(t_i), lst(t_i)]$ and can be chosen arbitrarily. We extend $\sigma(t_i)$ as follows: for every $t_j \prec t_i$, we choose $\sigma(t_j) = est'(t_j)$; for every t_j such that $t_i \prec t_j$ or t_j is not $\prec$-related to t_i, we choose $\sigma(t_j) = lst'(t_j)$. By the covering property, it follows immediately that for every t_i, $\sigma(t_i) \in [est(t_i), lst(t_i)]$.

We now prove that σ satisfies Π and is makespan minimal. Consider an arbitrary $t_k, t_l \in T$ such that $t_k \prec t_l$. We distinguish the following cases:

Case (1:) $t_k, t_l \prec t_i$.
Then $\sigma(t_k) + l(t_k) = est'(t_k) + l(t_k)$. By (est, lst)-monotonicity, $est'(t_k) + l(t_k) \leq est'(t_l) = \sigma(t_l)$. Hence, $\sigma(t_k) + l(t_k) \prec \sigma(t_l)$ satisfying the constraint $t_k + l(t_k) \leq t_l$;

Case (2:) $t_k \prec t_i$ and $t_l = t_i$.
Then $\sigma(t_k) + l(t_k) = est'(t_k) + l(t_k) \leq est'(t_i) \leq \sigma(t_i)$

Case (3:) $t_k \prec t_i$ and ($t_i \prec t_l$ or t_i and t_l are not $\prec$-related).
Then $\sigma(t_k) + l(t_k) = est'(t_k) + l(t_k) \leq est'(t_l) \leq lst'(t_l) = \sigma(t_l)$.

Case (4:) ($t_k = t_i$ and $t_i \prec t_l$) or (t_i and t_l are not $\prec$-related).
Then $\sigma(t_i) \leq lst'(t_i) \leq lst'(t_j) - l(t_i) \leq lst'(t_l) = \sigma(t_l)$.

Case (5:) ($t_i \prec t_k$ or t_i and t_l are not $\prec$-related) and ($t_i \prec t_l$ or t_i and t_l are not $\prec$-related).
Then $\sigma(t_k) = lst'(t_k) \leq lst'(t_l) - l(t_k) < lst'(t_l) = \sigma(t_l)$.

We conclude that σ satisfies all constraints, showing that I' guarantees incremental solvability.

This solution is also makespan minimal: since I' is covered by I_0 it follows that the makespan m' based on I' cannot be larger than m, the makespan based on I_Π. Since there exists at least one $t_k \in T$ such that $m = est(t_k) + l(t_k) = let(t_k) + l(t_k)$, we have $[est(t_k), est(t_k)] \in I_\Pi$. Hence, by covering, $[est(t_k), est(t_k)] \in I'$ as well and $\sigma'(t_k) = est(t_k)$. Therefore, $m' \geq m$. Hence, since $m' \leq m$, it follows $m = m'$. $\square$

Since Algorithm 4 preserves covering but might return an interval solution set that is not (est, lst)-monotonic, we have to restore the latter property. This can be easily taken care of by propagating the changed est-values upward in the $\prec$-chain and propagating the changed lst-values downwards in the $\prec$-chain. The improved version of this algorithm for executing a temporal decoupling step then can be specified as follows (see Algorithm 5):

Note that the (est, lst)- propagation process restores (est, lst)-monotonicity.

Example 4.8. Let us illustrate the effect of applying Algorithm 5 to Example 4.6. After changing $lst(t_4)$ to 3 and $est(t_5)$ to 4, we have to adapt the est values of the successors of t_5. Hence, $est(t_6) = max_{t_l \in prec(t_6)}\{est'(t_l) + l(t_k)\} = est'(t_5) + 1 =$

Algorithm 5 Performing a temporal decoupling step — refined.

```
Require: An incrementally solvable interval solution set $I = \{[est(t_i), lst(t_i)] : t_i \in T\}$, two tasks $t_i, t_j \in T$ such that $t_i \prec t_j$ belong to different agents, and $lst(t_i) + l(t_i) > est(t_j)$;
Ensure: $est(t_j)$ and $lst(t_i)$ values are modified such that $lst(t_i) + l(t_i) \leq est(t_j)$ and a new incrementally solvable interval solution set $I'$ is obtained
 1: let $\delta_i = \frac{lst(t_i)+l(t_i)-est(t_j)}{2}$;
 2: let $est(t_j) := est(t_j) + \delta_i$;
 3: let $lst(t_i) := lst(t_i) - \delta_i$;
 4: let $t = t_i$
 5: for all $t \in T$ such that $t_i \prec t$, starting with $t \in succ(t_i)$ do
 6:     $est(t) = \max_{t' \in prec(t)}\{est(t') + l(t)\}$;
 7: end for
 8: for all $t \in T$ such that $t \prec t_j$, starting with $t \in prec(t_j)$ do
 9:     $lst(t) = \min_{t' \in succ(t)}\{lst(t') - l(t)\}$;
10: end for
11: return the new interval solution set $I' = \{[est(t_i), lst(t_i)] : t_i \in T\}$.
```

$4 + 1 = 5$. Now we have obtained a new interval solution set I' where

$$\begin{aligned}
est'(t_1) &= 0, & lst'(t_1) &= 0\\
est'(t_2) &= 3, & lst'(t_2) &= 3\\
est'(t_3) &= 6, & lst'(t_3) &= 6\\
est'(t_4) &= 0, & lst'(t_4) &= 3\\
est(t_5) &= 4, & lst'(t_5) &= 7\\
est(t_6) &= 5, & lst(t_6) &= 8
\end{aligned}$$

The reader might check that indeed I' is an interval solution set which satisfies (est, lst)-monotonicity. ■

As a consequence of the preceding discussion, the correctness of the temporal decoupling step as performed by Algorithm 5 immediately follows :

Proposition 4.7. *Given an incrementally solvable interval solution set $I = \{[est(t_i), lst(t_i)] : t_i \in T\}$, two tasks $t_i, t_j \in T$ such that $t_i \prec t_j$ belonging to different* *agents, and $lst(t_i) + l(t_i) > -est(t_j)$. Let I' be the solution interval set resulting from performing the temporal decoupling step to t_i, t_j by Algorithm 5. Then*

1. *I' is covered by I and I' is incrementally solvable;*
2. *For every choice of $\sigma(t_i)$ and $\sigma(t_j)$ based on I' there is a complete extension σ satisfying all constraints.*

An efficient temporal decoupling algorithm for CAS problem instances

Each temporal decoupling step effectively 'removes' an inter-agent constraint by tightening local constraints. Once a constraint is implied by such a tightening step it remains implied also if other steps might follow. Since every temporal decoupling step performed preserves incremental solvability, the temporal decoupling algorithm basically is an iteration of temporal decoupling steps. So, it suffices to specify how as a result of applying these temporal decoupling steps, the final decoupling into CAS subproblems is obtained. But this is easy: for each local instance Π_i, we only need to guarantee that changes in the *est* and *lst* values are propagated by modifying the $r(t)$ and $d(t)$ values of the tasks involved. Hence, our temporal decoupling algorithm for CAS problem instances can be specified as follows:

At first sight temporal decoupling might take more than linear time: if there are p tuples in the precedence relation $\prec$ and n tasks, a single execution of the temporal decoupling step might cost $O(n)$ time to propagate the updated *est* and *lst*-values and we would end up with an algorithm taking $O(p.n)$, i.e., quadratic in the input size. Linear time, however, can easily obtained if we perform the temporal decoupling steps in topological order. Then, following the topological order, for every pair $t_i \ll t_j$ we do either *(i)* nothing, or *(ii)* we perform a temporal decoupling step and we adapt the est, lst values locally and we keep record of these adapted values or *(iii)* we propagate a stored adapted *est*-value and adapt the current t_i, t_j values and keep record of the adapted value. Hence, we ensure that during execution of the temporal decoupling steps, the *est*-values are updated. This sweep will cost $O(n + p)$ time. Hereafter, we update the *lst*-values, performing a sweep downward. This sweep will cost $O(n + p)$ time, too. hence, the total time needed is $O(n + p)$, i.e., linear in the size of the input instance.

As a consequence, we have the following result:

Algorithm 6 Temporal Decoupling for Coordinated Scheduling.

```
Require: A CAS problem instance $\Pi = \langle T, A, \prec, \phi, l(), c(), r(), d() \rangle$
Ensure: A temporal decoupling $\{\Pi_i\}_{i=1}^{n}$ of $\Pi$
 1: Compute the set $I = \{[est(t), lst(t)] : t \in T\}$;
 2: while there exist $t_i \ll t_j$ in $T$ such that $t_i$ and $t_j$ belong to different agents and
    $lst(t_i) + l(t_i) > est(t_j)$ do
 3:    apply Algorithm 5 to $I$;
 4:    let $I := I'$;
 5: end while
 6: for all $A_i \in A$ do
 7:    Let $\Pi_i$ be the restriction of $\Pi$ to $T_i$;
 8:    for all $t \in T_i$ do
 9:       set $d_i(t) := lst(t)$ in $\Pi_i$;
10:       set $r_i(t) := est(t)$ in $\Pi_i$;
11:    end for
12: end for
13: return the decoupling $\{\Pi_i\}_{i=1}^{n}$
```

Proposition 4.8. *Temporal decoupling for* CAS *problem instances can be performed in linear time.*

Let us now illustrate the decoupling of a CAS problem instance.

Example 4.9. Consider again the job shop problem instance in Example 4.1 shown in Figure 4.6. We assume $r(t) = 0$ and $d(t) = \infty$ for every task $t \in T$. We compute a decoupling following Algorithm 6 First step, we perform a topological ordering of all the tasks to determine the *est* values of all tasks. This results in:

$$est(t_1) = 0;\ est(t_2) = 2;\ est(t_3) = 3;$$
$$est(t_4) = 0;\ est(t_5) = 1$$

Notice that the minimal makespan $m = 3 + 2 = 5$. Now we compute the *lst*-values:

$$lst(t_1) = 0;\ lst(t_2) = 2;\ lst(t_3) = 3;$$
$$lst(t_4) = 2;\ lst(t_5) = 3.$$

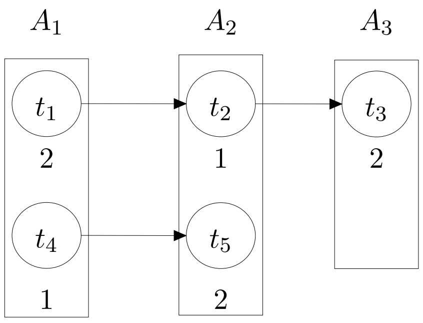


Figure 4.6: Overlapping constraints can lead to infeasible schedules.

As a result we obtain the following interval solution set I_Π:

$$t_1 = [0,0];\ t_2 = [2,2];\ t_3 = [3,3];$$
$$t_4 = [0,2];\ t_5 = [1,3].$$

Notice the starting time intervals of tasks t_4, t_5. These intervals allow agent A_1 to start t_4 at time 1, while A_2 also can choose to start t_5 at time instant 1. However, if these schedules are merged, we violate the precedence constraint between t_4 and t_5 $\sigma(t_4) < \sigma(t_5) + 1$. To ensure that this constraint is always implied, we perform a decoupling on $t_4 \prec t_5$: first we compute a suitable δ value:

$$\delta = \frac{lst(t_4) - est(t_5) + l(t_4)}{2} = 1$$

Hence, we reset the intervals for tasks t_4, t_5 as follows:

$$t_4 = [0,1]; t_5 = [2,3]$$

Now we perform the last decoupling step.

1. Agent A_1 receives the tasks t_1 and t_4 with the following constraints: $d(t_1) = 1$, $d(t_4) = 1$;

2. Agent A_2 receives the tasks t_2 and t_5 with the constraints $r(t_2) = d(t_2) = 2$ and $r(t_5) = 2$, $d(t_5) = 3$;

3. Agent A_3 receives the task t_3 with constraints $r(t_3) = d(t_3) = 3$.

Now if each agent schedules its tasks satisfying these constraints, there can be no violations of precedence constraints. ■

A very simple decoupling algorithm: the Interval scheduling algorithm (ISA)

In the previous part we developed Algorithm 6 that would ensure a complete temporal decomposition of any given CAS problem. Algorithm 6 was based on Hunsberger's temporal decoupling method and was reliant on the correspondence between CAS problem instances and STN.

Interestingly, there is an easier way to solve CAS problems. This simpler way is based on the realisation that to achieve a decomposition of the CAS problem we need to only ensure the following properties:

- whenever the original CAS problem admits a solution, all decomposed instances of the CAS problem must admit a solution and
- each set of solutions to the decomposed CAS problems can be merged to obtain a solution to the global CAS problem.

In the sequel, we show that to preserve these properties, we do not need to perform a full temporal decomposition, but can achieve the same result by *interval separation*. Here, interval separation simply means that we perform the temporal decoupling step on all the tasks that have overlapping intervals in parallel and irrespective of whether both tasks belong to the same agent or not. Therefore, instead of having to 'propagate' the updated $est()$ and $lst()$ values, we simply separate all intervals that could possibly violate incremental solvability. Based on this idea we propose the *Generalised Interval based scheduling algorithm (*ISA*).*

The output of this simple algorithm is a modified set of release dates and deadlines corresponding to each task in the CAS problem instance. With these modified release times and deadlines, it can be ensured that any solution to the CAS subproblems of this CAS problem instance can be merged to obtain a solution to the original problem instance. A complete description of the algorithm is given in Algorithm 7.

Algorithm 7 Generalised Interval based Scheduling (ISA).

```
Require: A CAS-instance $\Pi$ with the set $I_\Pi = \{[est(t), lst(t)]; t \in T\}$;
Ensure: A CAS-instance $\Pi'$ with modified release times and deadlines of tasks in $T$;
1: for all $t \in T$ do
2:     for all $t' \in T$ such that $t \ll t'$ and $lst(t) + l(t) > est(t')$ do
3:         $\delta_{t,t'} = \frac{lst(t) - est(t') + l(t)}{2}$;
4:     end for
5:     $\delta_t = max_{t \ll t'}\{\delta_{t,t'}\}$;
6:     $d(t) = lst(t) - \lfloor \delta_t \rfloor$;
7:     $r(t') = est(t') + \lceil \delta_t \rceil$;
8: end for
```

The idea of ISA is that by modifying deadline and release times of tasks in a given CAS-instance Π, for every partitioning $\{T_i\}_{i=1}^m$ of tasks, the corresponding set of instances $\{\Pi_i\}_{i=1}^m$ induced by $\{T_i\}_{i=1}^m$ is a decomposed set of CAS-instances. This means that every set of solutions to this set of instances can be merged to a solution of Π itself.

Let us now prove that ISA is correct.

Theorem 4.9. *Algorithm* ISA *is correct. That means, if* Π *is a* CAS *instance,* $\{T_i\}_{i=1}^m$ *a partitioning of* T*, and* Π' *is the* CAS *instance computed by* ISA*, then the set* $\{\Pi'_i\}_{i=1}^m$ *induced by* $\{T_i\}_{i=1}^m$ *is a decomposed set of* CAS *instances and every set* $\{\sigma_i\}$ *of solutions for* $\{\Pi'_i\}_{i=1}^m$ *can be merged to a solution* σ *for* Π*.*

Proof. Obviously, ISA halts whenever a CAS instance Π is given as input. Therefore, it suffices to prove that

1. whenever Π is consistent, each of the decomposed instances Π'_i is also consistent, and

2. every set of solutions $\{\sigma_i\}_{i=1}^m$ where σ_i is a solution for Π_i is mergeable to a solution σ for Π.

To prove the first part, suppose that Π is a consistent CAS problem instance. Then, by Proposition 4.3, we have $lst(t) \geq est(t)$ for all $t \in T$. Now consider an arbitrary part Π'_i. We show that for every $t \in T_i$ we have $est'(t) \leq lst'(t)$ where the est' and lst' values are computed from Π_i according to the modifications applied to Π by ISA.

Suppose on the contrary that there exists a task t_i in Π' such that $t_i \in T_j$ for every chain of tasks up to t_i, t_i is the first task such that $est'(t_i) > lst'(t_i)$. Since $est'(t_i) \geq r(t_i)$ and $lst'(t_i) \leq d(t_i)$, we distinguish the following two cases:

1. $r(t_i) < d(t_i)$. Then there must exist some $t' \prec t_i$ such that $est'(t') = est'(t_i) - l(t') > lst'(t_i) - l(t') = lst'(t')$. But then we have a contradiction, because t' occurs before t_i and $est'(t') > lst'(t')$. So this case cannot hold.

2. $r(t_i) \geq d(t_i)$. According to ISA we have

$$r(t_i) = est(t_i) + \frac{lst(t') + l(t') - est(t_i)}{2} = \frac{est(t_i) + lst(t') + l(t')}{2} \text{ and}$$
$$d(t_i) = lst(t_i) - \frac{lst(t_i) + l(t_i) - est(t'')}{2} = \frac{lst(t_i) + est(t'') - l(t_i)}{2}$$

Since $lst(t') + l(t') \leq lst(t_i)$ and $est(t_i) + l(t_i) \leq est(t'')$, it follows that

$$r(t_i) = \frac{est(t_i) + lst(t') + l(t')}{2} \leq \frac{est(t_i) + lst(t_i)}{2} \text{ and}$$
$$d(t_i) = \frac{lst(t_i) + est(t'') - l(t_i)}{2} \geq \frac{est(t_i) + lst(t_i)}{2}$$

Therefore, $r(t_i) \leq d(t_i)$, contradicting the assumption and this case cannot hold.

Hence, we must conclude that for all tasks $t \in \Pi_i$ we have $est'(t) \leq lst'(t)$. Therefore, every Π_i is consistent.

The proof of the second part is simple. Since we know that every decomposed part Π'_i is consistent, it has a solution σ_i. Take a set $\{\sigma_i\}_{i=1}^m$ of those solutions, arbitrarily chosen. We show that the merge $\sigma = \bigcup_{i=1}^m \sigma_i$ is always a solution to Π. First note that any solution to Π' is a solution to Π, since ISA restricts the set of possible solutions to Π. We now show that σ is a solution to Π'. Since σ is composed of solutions to the decomposed parts, it suffices to show that whenever t, t' belong to different parts and $t \prec t'$, we have $\sigma(t) + l(t) \leq \sigma(t')$. So suppose $t \in T_i$ and $t' \in T_j$. Then we have to show that $\sigma_i(t) + l(t) \leq \sigma_j(t)$. Since $t \prec t'$, we have that

$$\sigma_i(t) \leq d(t) \leq lst(t) - \lfloor \delta_{t,t'} \rfloor \text{ and}$$
$$\sigma_j(t) \geq r(t') \geq est(t') + \lceil \delta_{t,t'} \rceil$$

Now $\delta_{t,t'} = \dfrac{lst(t) - est(t') + l(t)}{2}$. Hence,

$$\sigma_i(t) \leq \frac{lst(t) + est(t') - l(t)}{2} \text{ and}$$
$$\sigma_j(t) \geq \frac{lst(t) + est(t') + l(t)}{2}$$

Therefore, $\sigma_j(t) - \sigma_i(t) \geq l(t)$. □

Example 4.10. To illustrate the procedure, consider again the situation in Example 4.6. The temporal decoupling step of ISA computes the intervals for tasks t_4 and t_5 as follows:

$$\begin{array}{lll} \delta_4 & = \frac{lst(t_4)-(est(t_5)+l(t_4)}{2} & = 3 \\ lst'(t_4) & = lst(t_4) - \lfloor \delta_4 \rfloor & = 3 \\ est'(t_5) & = est(t_4) + \lceil \delta_4 \rceil & = 4 \end{array}$$

This results in intervals $[0, 3]$ and $[4, 7]$ for tasks t_4 and t_5 respectively. The difference now is that, ISA does not stop at this point. It continues further and performs interval separation on the precedence constraint between t_5 and t_6 as well. As a result, $lst(t_5) = 6$ and $est(t_6) = 7$. No other pair of tasks has an interval overlap and the procedure halts. The updated set of values are as follows:

$$\begin{array}{llll} est'(t_1) & = 0, & lst'(t_1) & = 0 \\ est'(t_2) & = 3, & lst'(t_2) & = 3 \\ est'(t_3) & = 6, & lst'(t_3) & = 6 \\ est'(t_4) & = 0, & lst'(t_4) & = 3 \\ est(t_5) & = 4, & lst'(t_5) & = 6 \\ est(t_6) & = 7, & lst(t_6) & = 8 \end{array}$$

With these intervals, it is easy to see that incremental solvability is restored.

Proposition 4.10. *Algorithm* ISA *computes a solution in* $O(|T| + |\prec|)$ *time for any given* CAS *problem instance* $\Pi = \langle T, A, \prec, \phi, l(), c(), r(), d() \rangle$.

Proof. By performing a topological sort, we can find out the *est* and *lst* values of each task in time linear to the sum of tasks and the edges $O(|T| + |\prec|)$. The remaining computations can be done in time linear to the number of edges in the CAS problem. Thus, the overall asymptotic time complexity of finding a decoupling for STN_{CAS} through ISA is $O(n + p)$, where n is the number of tasks and p the number of tuples in the precedence relation $\prec$. $\square$

Solutions to $\mathcal{S}_\Pi$ are guaranteed to be makespan minimal and ISA which is derived from applying Hunsberger's algorithm on $\mathcal{S}_\Pi$ naturally also guarantees makespan minimality. Thus, the following theorem.

Theorem 4.11. ISA *always ensures a minimum makespan.*

4.3.2 ISA and maximal autonomy

There is another nice property of ISA as well. One can prove that the intervals generated by ISA cannot be modified without either violating a precedence constraint or violating optimality. In other words it guarantees maximal autonomy. This property can easily be proven by noticing that the ISA algorithm generates a set of constraints $C = \{C(t) \mid t \in T\}$ such that:

- For every task t such that $succ(t) = \emptyset$ we have $d(t) = depth(T) - l(t)$, in other words, no task can be scheduled such that it exceeds the depth of the task graph.
- For every t such that $pred(t) = \emptyset$ it holds that $r(t) = 0$. That is, no task can be scheduled before time 0.
- For every pair of tasks t, t' such that $t \prec t'$ it holds that $r(t') = d(t) + l(t)$. In other words, there exist no 'holes' between the release times and deadlines that are larger than the processing time of the preceding task.

This implies that any strict weakening C' of C would contain a constraint $C(t) = [est(t), lst(t)]$ such that either,

Case (a:) $est(t) < 0$ or

Case (b:) $lst(t) > est(T) - l(t)$ or

Case (c:) there exists some t' such that $t \prec t'$ and $lst(t) + l(t) > est(t')$ or

Case (d:) there exists some t' such that $t' \prec t$ and $lst(t') + l(t) > est(t)$.

Clearly, case *(a)* leads to an infeasible schedule because task processing cannot start before time point 0. Case *(b)* would imply that some C'-satisfying schedules are not makespan efficient as they would allow for schedules longer than the critical path length. Cases c or d cannot also hold, because it would lead to some C'-satisfying schedules violating a precedence constraint. Hence, such a weakening cannot exist and we have the following proposition:

Proposition 4.12. *Any strict weakening the of the set C of constraints imposed by* ISA *either leads to a infeasible schedule or leads to a non optimal makespan.*

In summary, we have the following property:

Theorem 4.13. ISA *ensures a maximally autonomous set of constraints and a makespan efficient global schedule.*

4.4 Summary

We have so far seen that there are two possible ways to decouple a CAS problem.

1. We reduce a given CAS problem instance into a $\mathcal{S}_{\Pi}$, then apply Hunsberger's algorithm to derive the decoupling constraints.

2. Derive an efficient method from Hunsberger's method which can decouple CAS problems instances directly.

The basic idea in assuring a valid decoupling is to tighten local constraints 'just enough' to make inter-agent constraints redundant. Hunsberger's method, while more general, requires repeated computations of the distance matrix, which can be expensive. Therefore we developed an algorithm ISA, based on the second approach listed above. ISA uses the idea of interval separation to achieve decoupling. ISA was shown to result in not only makespan minimal schedules but also schedules guaranteeing maximal autonomy.

The main drawback of ISA in its current form, is that it can only be applied to situations when the concurrency bounds on agents are very large (at least large enough to process as many of the tasks that need to be simultaneously processed within the intervals ISA computes for them). It is not surprising since the STN framework from which ISA is derived, cannot accommodate capacity information. However, ISA serves as an excellent starting point to explore problems of higher complexity — problems with finite bounds on the concurrency of agents.

Naturally, we would next like to solve this class of CAS problems. It is of quite common occurrence that agents can only perform a limited number of tasks simultaneously. Therefore, if we could develop a mechanism to handle CAS problems with bounded concurrency agents, then it would be useful in practical situations. Thus, in the next chapter we explore this so called class of bounded CAS (bCAS) problems and design a solution for bCAS problem instances. In doing so, we have two choices — either develop a completely new approach to handle CAS problem instances with concurrency constraints or adapt ISA to handle concurrency constraints. We choose to adapt ISA to handle these situations. The choice being motivated by the inherent simplicity of ISA.

Chapter 5

Solving CAS problems: The bounded concurrency case

Chapter 4 discussed the case of coordinating schedules when agents had unbounded concurrency. In this chapter we study the case where agents have bounds on their concurrency. We show that the CAS *problem becomes intractable once capacity constraints are introduced. We further investigate subclasses of* CAS *problem instances and show that* ISA *based algorithms can be effectively used to coordinate some very restricted subclasses.*

When agents have unbounded concurrency, the only constraints on scheduling are the precedence constraints between tasks. As was pointed out earlier, there exist many scenarios where there is a bound on the capacity of the agent. For instance, in an emergency air evacuation scenario, the agent responsible for planning helicopters may have limited number of helicopters available for evacuation. Similarly, in several logistics scenarios, there is a limitation on the number of trucks that can be used for transporting goods.

Notice that in situations as above, problems derive constraints from

- the precedence relationships of tasks and
- the concurrency constraints.

Concurrency constraints occur in many situations because resources are expensive.

For example, in a job shop, procuring a machine with higher speed or procuring more machines to augment capacity might be too costly in comparison with the benefits derived out of a smaller makespan. As we will see in this chapter, once agents have limits on concurrency (capacity), ISA is not sufficient. The intervals computed by ISA may require agents to use more capacity than is available to them to design a valid schedule. In such cases, agents might be willing to compromise on the optimality of the global makespan so that their resource (concurrency) constraints are not violated.

In this chapter, we develop a coordination mechanism that computes intervals such that local schedules of agents can be combined to derive a global schedule with a *small* makespan. As noted earlier, addition of concurrency constraints makes the problem harder (NP-complete) in general and therefore expecting a makespan efficient solution would be unwise. Therefore, we investigate the makespan efficiency and the price of autonomy,[1] of the mechanism in different subclasses of problem instances. We start by establishing the complexity in the general case and then study a subclass of problem instances known as *chain structured* bCAS *instances*. Within this subclass we further look at the class of grids and unequal chains. Our results in this chapter further strengthen the view that makespan efficient solutions to the CAS problem (in general) are impossible to derive efficiently unless P=NP.

5.1 The complexity

We start by establishing the complexity of finding a solution to the CAS problem when agents have bounds on their concurrency. To distinguish CAS problems where agents have bounded concurrency, we denote such problem instances as $\Pi_b = \langle T, A, \prec, \phi, l(), c() \leq B, r(), d() \rangle$ where, $B \in Z^+$. In the following text we will refer to this version of the CAS problems as the bCAS problem.

Recall that in the previous chapter, we were able to show that ISA results in a makespan minimal schedule. More precisely, in the unbounded case we were able to find a set C of additional constraints that could ensure a minimum makespan M for the total set of tasks. Finding such a set C, so that any set $\{\sigma_i\}_{i=1}^{n}$ of locally feasible schedules would result in a makespan M, given that agents are bounded,

[1]Price of autonomy is defined in section 5.1.

i.e., agents can perform only a fixed number of tasks at any given point in time, is an intractable problem.

Theorem 5.1. *Given a* bCAS *problem* $\Pi_b = \langle T, A, \prec, \phi, l(), c() \leq B, r(), d() \rangle$ *and a positive integer* M*, the problem to decide whether there exists a set of interval constraints* C *such that the scheduling instance ensures a solution with makespan* M *is* NP*-complete.*

Proof. We reduce the Partitioning problem (see [Garey and Johnson 1979]) (Given a set S of integers, is there a subset S' of S such that $\sum_{s \in S'} s = \sum_{s \in \bar{S}} s$, where $\bar{S} = S - S'$?) to the bCAS problem.

We convert an instance of Partitioning to an instance of the bCAS problem in the following way:

- Let $d_S = \sum_{s \in S} s$. WLOG, we can assume d_S to be even.
- Create the set of tasks $T = \{t_s | s \in S\} \cup \{t_a, t_b, t_c, t_d\}$.
- For every task $t_s \in T - \{t_a, t_b, t_c, t_d\}$, $l(t_s) = s$.
  Let $l(t_a) = \frac{d_S}{2}$, $l(t_b) = 1$, $l(t_d) = 1$ and $l(t_c) = \frac{d_S}{2}$.
- Let $\prec = \{(t_a \prec t_b \prec t_c), (t_a \prec t_d \prec t_c)\}$.
- Let agent A_1's capacity $c(A_1) = 2$ and $T_1 = T$.
- Let $r() = 0$ and $d() = \infty$ for all $t \in T$.
- Finally, let $M = d_S + 1$.

Now it is not difficult to see that there exists a set of interval constraints C allowing for a makespan (M) efficient autonomous scheduling solution iff the partition instance S has a solution: exactly in that case, agent A_1 is able to use:

1. One of its capacities to process one subset of its set of tasks in the interval $[0, l(t_a)]$, start t_b at time instant $l(t_a)$ and complete the remaining subset of tasks in the interval $[l(t_a) + 1, d_S + 1]$ and

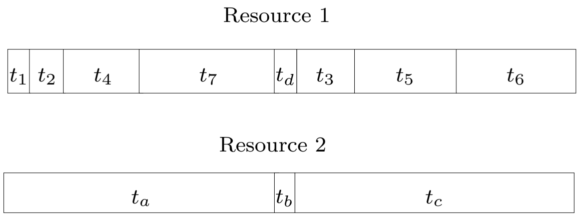


Figure 5.1: Schedules of agent A_1 in the reduction.

2. Use the other capacity to process t_a in the interval $[0, l(t_a)]$, start t_d at $l(t_a)$ and finish t_c at time instance $d_S + 1$.

Clearly, it is trivial to verify whether the resulting makespan is less than M. Therefore, we claim that our problem is NP-complete. □

Notice that the problem is NP-hard even with only one agent, so it follows that it stays NP-hard when multiple agents are involved. The difficulty lies in the fact that the agent cannot efficiently schedule its local tasks.

Let us illustrate the reduction through the following example.

Example 5.1. Suppose we have a set $S = \{1, 2, 3, 4, 5, 6, 7\}$. The partitioning problem is to now find a subset S' such that

$$\sum_{s \in S'} s = \sum_{s \in S - S'} s$$

Suppose we were able to find such a solution efficiently (i.e., in polynomial time) then we could also efficiently solve the following bCAS problem:

$T = \{t_1, \ldots, t_7, t_a, t_b, t_c, t_d\}$

$t_a \prec t_b \prec t_c$; $t_a \prec t_d \prec t_c$

$l(t_1) = 1$; $l(t_2) = 2$; $l(t_3) = 3$; $l(t_4) = 4$; $l(t_5) = 5$; $l(t_6) = 6$; $l(t_7) = 7$

$l(t_b) = 1$; $l(t_d) = 1$

The solution of the bCAS problem above would be the following:

$$
\begin{aligned}
& l(t_a) = \frac{\sum_{i=1}^{7} l(t_i)}{2} = 14 \\
& l(t_c) = \frac{\sum_{i=1}^{7} l(t_i)}{2} = 14 \\
& T_1 = T \\
& c(A_1) = 2 \\
& r(t) = 0;\ \forall t \in T \\
& d(t) = \infty;\ \forall t \in T
\end{aligned}
$$

Let $M = \sum_{i=1}^{7} l(t_i) + 1 = 29$. Now, suppose we have efficiently found the subset S' with $\sum_{s \in S'} s = 14$.

One subset of tasks that results in a total duration of 14 is $\{t_1, t_2, t_4, t_7\}$. This implies that we can schedule tasks t_1, t_2, t_4 and t_7 in any order to be performed by agent A_1 within the interval $[0, 14]$. Task t_d is performed at time instant 14 and the tasks $\{t_3, t_5, t_6\}$ can be scheduled in the interval $[15, 29]$ using the first resource. With the second resource, task t_a can be performed in the interval $[1, 14]$, task t_b at time instant 14 and task t_c in the interval $[15, 29]$. The schedules of agent A_1 are shown in Figure 5.1. ∎

Because of the intractability of the bCAS problem, we are forced to lower our expectation regarding the minimality of makespan. Thus, in this chapter we try to ensure that solutions to the bCAS problem satisfy three things:

1. Agents should be able to construct their schedules autonomously. That is, they should not be required to communicate with other agents.

2. The intervals given to the agents to schedule their tasks should allow them to construct at least a single valid local schedule within their concurrency bounds.

3. The makespan of the global schedule should be as low as possible.

We have shown it is NP-complete to find a solution for the bCAS problem. In such cases, it is important to understand the loss in terms of makespan that mechanisms such as ISA (which ensure autonomy) would incur. To measure this trade off, we define a ratio called the *price of autonomy.* Intuitively, the price of autonomy of any scheduling mechanism to solve a bCAS problem is the measure of the trade off between optimal makespan and the worst case makespan derived from the mechanism. Formally,

Definition 5.1. Price of autonomy: Given a mechanism $\mathcal{M}$ to solve a bCAS problem instance, the price of autonomy is the ratio of the worst case makespan resulting from $\mathcal{M}$ to the optimal makespan. Price of autonomy is denoted by $\mathrm{P_a}$. ■

If the price of autonomy of a mechanism to solve bCAS problems is high then we know that the mechanism may not be very desirable when makespan considerations are very critical. In such cases one might consider a slight compromise on autonomy by imposing additional restrictions on local scheduling of agents. On the other hand, if the price of autonomy is low then we can expect that the mechanism will perform well.

The ISA algorithm discussed in the previous chapter was derived based on the TD algorithm of Hunsberger. Therefore, quite naturally, our first step would be to find out if there exist results in the STN world that can assist us in solving CAS problems.

As indicated earlier, the STN framework while very useful in handling temporal constraints, cannot accommodate resource constraints. Fortunately, there exists literature in the STN community where authors have dealt with resource constraints. In [Policella *et al.* 2007], the authors devised a two phased paradigm called the *Precedence constraint posting (PCP)* paradigm to address this problem. In this paradigm, the first layer is the *time* layer. Here temporal aspects of the scheduling problem are represented through a STN, and by propagating temporal constraints, an interval $[est(t), lst(t)]$ is determined for each time point. The second layer — the *resource* layer — analyses the contentions for a set of shared resources over time. It produces a resource profile graph which indicates the request for each resource given the intervals from the first layer. Based on this resource profile analysis, the resource layer posts additional constraints to the problem ensuring that resource conflicts are absent. The time layer takes over again and computes a new set of intervals for each of the time points.

In the context of a bCAS problem, the idea that an agent A_i can perform $c(i)$ tasks simultaneously implies that A_i has enough resources to carry out $c(i)$ jobs simultaneously. In other words, concurrency constraints are essentially resource constraints. Unlike traditional resource constraints, however, resources in bCAS problems are exclusive and not shared between agents. However, the general idea of separating resource constraints and precedence constraints of [Policella *et al.* 2007] can be useful in devising solutions to the bCAS problem. We present a coordination mechanism for solving bCAS problems based on this general idea in the following subsection.

5.2 A general bCAS mechanism

In our proposed coordination mechanism, the first phase identifies time intervals for scheduling tasks based on ISA. Then in a second phase, given these time intervals for each agent A_i, due to its resource constraints, there may exist some tasks $\{t | t \in T_i\}$ that cannot be scheduled within the given intervals, i.e., this set of tasks are *conflicting*. From Theorem 5.1, we know that determining an efficient schedule for an agent is NP-hard. It follows that determining conflicting tasks is also NP-hard. Suppose conflicting tasks can be found by some exact, exponential-time algorithm. After such conflicting tasks are detected, one way to resolve a pair of conflicting tasks t and t' is to add a precedence constraint $t \prec t'$ (or $t' \prec t$). The result of such a conflict resolution is then fed into the first phase and a new set of intervals is computed. We continue this iterative process until there are no more conflicts within any agent.

The general coordination mechanism M for solving the bCAS problem is described as follows:

Phase 1: Given a bCAS instance $\Pi_b = \langle T, A, \prec, \phi, l(), c() \leq B, r(), d() \rangle$, compute time intervals $[est(t), lst(t)]$ for all the tasks $t \in T$ using ISA.

Phase 2: Resolve conflict between any two conflicting tasks t, t' of an agent by adding a precedence constraint $t \prec t'$ (or $t' \prec t$) to the partial order $\prec$.

Go to Phase 1 with the updated bCAS instance Π_b and iterate until no conflicts exist for any agent.

The output of the above mechanism is a set of time intervals for all the tasks in T that enable feasible scheduling for each agent $A_i \in A$.

There are three problems arising from this general coordination mechanism.

Question 1: In Phase 2, how to detect conflicting tasks efficiently?

Question 2: Once conflicts have been detected, what strategy should be used in order to decide which conflicts of which agent should be resolved?

Question 3: What is the consequence of using different conflict resolution strategies on the makespan of the global schedule?

Concerning Question 1, as we have shown in Theorem 5.1, it is hard for agents to even detect their own resource conflicts. That is, there is no polynomial time algorithm that can be used to answer Question 1. Therefore, it follows naturally that there is no polynomial-time algorithm to find feasible local schedules for agents that can be combined to an optimal global schedule. Some special treatments are required, which we will discuss in the later sections.

Answering questions 2 and 3 lead to questions on the price of autonomy of using any particular coordination method. One can imagine that small makespan schedules are easier to obtain if we take full control over agents' conflict resolution, without bothering about agent autonomy. As an illustration, let us consider the following example.

Example 5.2. Consider the situation in Figure 5.2. Three agents A_1, A_2, A_3 have to find local schedules such that the global schedule is optimal. Suppose also that each agent has capacity 1. All tasks of agent A_i are given an interval $[i-1, i-1]$ by ISA. Thus, each agent has to resolve three conflicting tasks.

If we now impose that agent A_1 shall always schedule t_1 first, t_2 next and thirdly t_3 and run ISA on the resulting updated partial order, then clearly the other two agents do not have any conflicts any longer. The resulting makespan is 5, which is optimal. ■

Although the above example demonstrates a possible solution, it contradicts our purpose of coordination, that is, ensuring autonomy of agents. Therefore, we need to develop a conflict resolution strategy that takes into account the autonomy of agents,

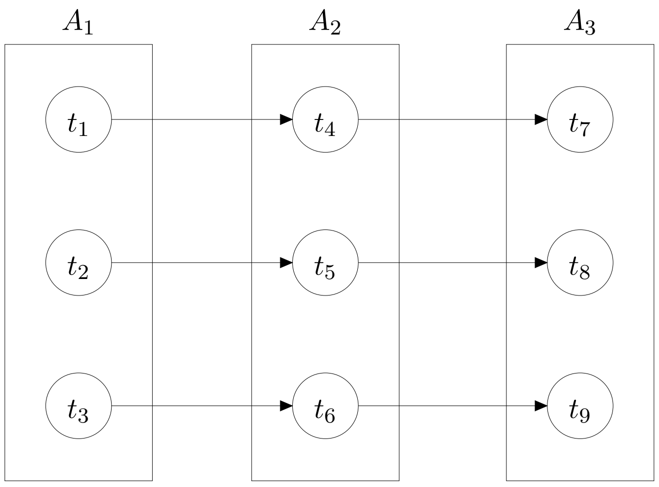


Figure 5.2: Constraining an agent completely can sometimes lead to optimal makespan.

and then analyses the price of autonomy of the resulting coordination mechanism $\mathrm{M}_{\mathrm{ISA}}$.

Definition 5.2. Autonomous coordination mechanism $\mathbf{M}_{\mathbf{ISAsq}}$: The mechanism $\mathrm{M}_{\mathrm{ISA}}$ works as an iterative two-phase mechanism as the general mechanism M, except that in Phase 2, $\mathrm{M}_{\mathrm{ISA}}$ randomly selects an agent $A_i \in A$ which has conflicting tasks t, t', and resolves conflicts between them based on the preference ($t \prec t'$ or $t' \prec t$) of A_i. ■

Example 5.3. Consider again the scenario in Figure 5.2. Instead of imposing an ordering on the tasks of agent A_1, we now ask A_1 to inform us of its preference between task t_1 and task t_2, and suppose A_1 chooses $t_2 \prec t_1$. We regenerate the time intervals using ISA, and randomly select another agent, say A_2, who has conflicts. Agent A_2 decides to execute t_4 before t_5. The time intervals are updated again. As another random choice, agent A_3 is chosen to tell its preference between conflicting tasks. Suppose A_3 prefers performing t_8 before t_9. The resulting makespan of $\mathrm{M}_{\mathrm{ISA}}$ becomes 6. ■

The mechanism $\mathrm{M}_{\mathrm{ISA}}$ considers the autonomy of participating agents. However, the price is that an agent may make *bad* local choices in determining the ordering of its tasks, and these unfortunate decisions may lead to a global schedule with its makespan far from the optimal. Thus, the price of autonomy of mechanism $\mathrm{M}_{\mathrm{ISA}}$ could be high. Actually, in the next subsection, we show that in general, the price of autonomy of $\mathrm{M}_{\mathrm{ISA}}$ is not bounded by any constant.

5.3 Solving bCAS with unit tasks and sequential agents

We have shown that it is NP-complete to ensure makespan minimality of solutions, to a general bCAS instance $\Pi_b = \langle T, A, \prec, \phi, l(), c() \leq B, r(), d() \rangle$. This is a direct conclusion from the fact that detecting resource conflicts for each agent is NP-hard.

In this section, we identify a class of bCAS instances in Π_b where the existence of conflicts raised in the coordination mechanism can be efficiently detected. Within this class, all tasks have unit duration. Furthermore, for the purpose of simplicity, we assume agents are sequential, i.e., $c(i) = 1$ for each $A_i \in A$. The bCAS instances of the class that we study in this section are defined as $\Pi_b = \langle T, A, \phi, \prec, l() = 1, c() = 1, r(), d() \rangle$. We shall deal with bCAS instances with non-sequential agents later in the chapter.

We first demonstrate that for the class of bCAS instances described by $\Pi_b = \langle T, A, \phi, \prec, l() = 1, c() = 1, r(), d() \rangle$, resource conflicts can be efficiently resolved using the idea of bipartite matching. However, unfortunately, later we prove that it is still NP-hard to find an optimal solution with minimal makespan for the general task structure. In addition, we prove that the price of autonomy of the autonomous coordination mechanism $\mathrm{M}_{\mathrm{ISA}}$ is not bounded by a constant.

Matching based ISA

In this subsection, we first use ISA to determine the time intervals for unit duration tasks. As we have shown earlier, if the agents had unbounded concurrency, they would be able to find a schedule satisfying all the constraints. If agents are sequential, this might not be possible. Due to resource constraints, an agent may have conflicting tasks. Given the earliest starting time $est(t_j)$ and the latest starting time $lst(t_j)$ of each task $t_j \in T_i$ of agent A_i, we say t_j is *conflicting* with some other task t_k of agent A_i as long as two intervals $[est(t_j), lst(t_j)]$ and $[est(t_k), lst(t_k)]$ have overlapping time points that do not allow them to be simultaneously scheduled. Fortunately, an efficient procedure to determine whether a given agent is able to find a sequential schedule for all tasks $t \in T_i$ exists.

Consider the bipartite graph $G_i = (T_i \cup I_i, E_i)$ where,

- I_i is the set of all time points occurring in the intervals $C(t) = [est(t), lst(t)]$ of tasks $t \in T_i$ and
- $(t, n) \in E_i$ iff $n \in C(t)$.

If we are able to find a *maximum matching* (see [Cormen *et al.* 1990]) involving all tasks of T_i, it immediately implies that we can schedule each of the tasks in T_i at exactly the time points they are matched with.[2] To find a matching, we use the Ford -Fulkerson method (see [Ford and Fulkerson 1957]).

The Ford-Fulkerson method for finding maximum flow in networks is a popular method to achieve maximum bipartite matching. In this method, a flow network is created for each agent with a special source u and a special sink w. To this network, directed arcs are added *(i)* from u to all predecessor free nodes and *(ii)* from all successor free nodes to w. The flow over each edge is restricted to 1. As long as there is an augmenting path (a path with available capacity is called an *augmenting path*), we send flow along one of these paths. Then we find another path and so on.

In our case, for each agent A_i we have the bipartite graph $G_i = (T_i \cup I_i, E_i)$ with directed edges from tasks in T_i to time points in I_i. We construct the maximum flow network from this bipartite graph as follows:

1. We add two more nodes u and w.
2. We next add directed edges (u, t) to every node $t \in T_i$ and
3. (n, w) for every node $n \in I_i$.
4. We then set the capacity of every edge to be equal to 1.

Figure 5.3 summarises the construction. In this figure $C(t_1) = [0, 0]$; $C(t_2) = [1, 1]$ and $C(t_3) = [2, 2]$.

[2] We assume integer values for schedules $\sigma(t)$.

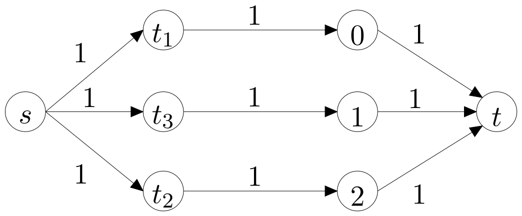


Figure 5.3: Bipartite matching through Ford-Fulkerson method for sequential agents.

Now if we are able to find a maximum flow equal to $|T_i|$, it implies that a sequential schedule is possible. On the other hand, if the maximum flow is less than $|T_i|$ then it implies that there exist at least a pair of tasks that are conflicting.

The Ford-Fulkerson method can be used to solve the maximum matching on a bipartite graph $G_i = (T_i \cup I_i, E_i)$ in $O((|T_i|+|I_i|)|E_i|)$ time (see [Cormen *et al.* 1990]).

Autonomous coordination mechanism M_{ISAsq}

The updated autonomous coordination mechanism M_{ISAsq} works as follows. In Phase 1, it calls the ISA to compute a set of time intervals for all tasks. After detecting conflicts using the proposed matching-based algorithm, in Phase 2, it asks a randomly selected agent to add a precedence constraint $t \prec t'$ (or vice-versa) between its conflicting tasks t and t'. The mechanism then calls the ISA algorithm again on the extended coordination instance.

This *(i)* calling the ISA algorithm, *(ii)* matching and *(iii)* extending the precedence relation can be repeated until we are guaranteed that for each agent there exists at least one sequential schedule. Note that this procedure must halt because conflicts can never reoccur.

The complete mechanism, for achieving a solution to bCAS problems with sequential agents and unit duration tasks, is given as Mechanism M_{ISAsq} in Algorithm 8.

Let us illustrate the complete process through the following example:

Example 5.4. Consider the situation in Figure 5.4 (a). Four unit duration tasks have been distributed among two sequential agents. The intervals derived through

Algorithm 8 Autonomous coordination mechanism $\mathrm{M}_{\mathrm{ISAsq}}$ using the matching based sequential adaptation of ISA

```
1: Inputs: constraint intervals $[est(t), lst(t)]$ for all the tasks $t \in T$ as computed by ISA.
2: Outputs: Revised $est(t)$ and $lst(t)$ for each $t \in T$ to enable sequential scheduling.
3: while there exists a $(T_i, C_i)$ not allowing for a maximal matching $M_i$ containing $T_i$ for a randomly selected agent $A_i$ do
4:    agent $A_i$ takes a task $t$ contained in $M_i$ and a conflicting task $t'$ not occurring in $M_i$;
5:    add $t \prec t'$ or $t' \prec t$ to the partial order $(T, \prec)$;
6:    run ISA on the updated partial order $(T, \prec)$ and update the set of constraint intervals $C$
7: end while
```

ISA are also shown. The corresponding bipartite graph is shown in Figure 5.4 (b) (the bipartite graphs for both the agents are similar and hence we show only one in the figure). It is easy to see that there does exist a maximum bipartite matching for the graph. This in turn implies that agent A_1 can design a valid schedule for its set of tasks within the given intervals.

Now suppose we have a new situation as shown in Figure 5.5 (a). Agent A_1 has 3 tasks instead of the two as in Figure 5.4 (a). Clearly, agent A_1 cannot find a sequential schedule which is reflected in its corresponding bipartite graph in Figure 5.5 (b). On the other hand agent A_2 can find a sequential schedule as is reflected in its bipartite graph.

One way of solving conflicts would be to add precedence constraints between conflicting tasks. The result of such an operation for the situation in Figure 5.5 is shown in Figure 5.6.

The final set of intervals for agent A_1's tasks in Figure 5.5 would be: $t_1 = [0, 0]; t_2 = [1, 1]; t_3 = [2, 2]$. ■

One may hope that since resource conflicts can be efficiently resolved, the problem of finding a minimal makespan schedule can be efficiently solved too. Unfortu-

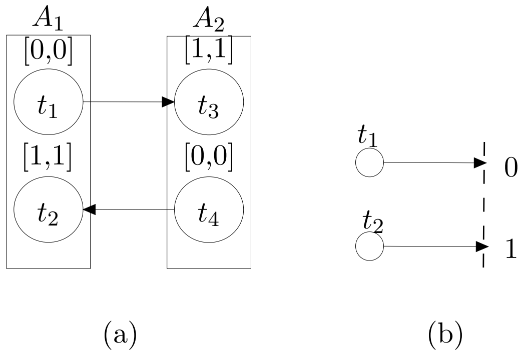


Figure 5.4: (a) After applying ISA; (b) the corresponding maximum bipartite matching.

nately, it turns out be false. We now prove that the bCAS problem with unit tasks and sequential agents stays NP-hard using the well known 3-Machine Unit-time Job-shop scheduling problem [Lenstra and Kan 1979].

Definition 5.3. 3-Machine, Unit-time Job-Shop scheduling:(adapted from [Lenstra and Kan 1979]): Given 3 machines M_1, M_2, M_3, each of which can process at most one job at a time, k jobs $J_1, \dots, J_k$ where $J_j (j = 1 \dots, n)$ consists of a chain of unit-time operations, the h^{th} of which has to be processed on machine μ_{jh} with $\mu_{jh} \neq \mu_{j,h-1}$ for $h > 1$ and an integer $M_{jobshop}$, does there exist a schedule with length at most $M_{jobshop}$? ■

Theorem 5.2. *Given a* bCAS *problem* $\Pi_b = \langle T, A, \prec, \phi, l() = 1, c() = 1, r(), d() \rangle$, *with unit tasks, sequential agents and a positive integer* M_{bCAS}*, the problem to decide whether there exists a set of constraints* C *such that the scheduling instance allows for a solution with makespan* M_{bCAS} *is* NP*-hard.*

Proof. We reduce the 3-machine job-shop scheduling problem to the bCAS problem as follows. Given an instance of the job-shop scheduling problem as defined above, define $O = \{\mu_{j,h} | j = \{1, \dots, n\}, h > 1\}$ as the set of all operations and construct the following bCAS problem.

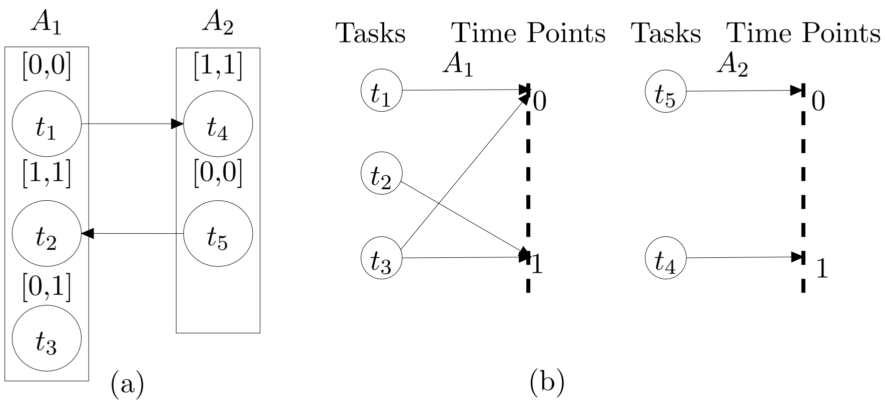


Figure 5.5: (a) Task graph with the intervals computed by ISA. (b) The corresponding bipartite graphs and possible matchings. If we can find a maximum matching for agent A_1, then it implies we have a sequential schedule for its tasks. In this case, however, it is not possible.

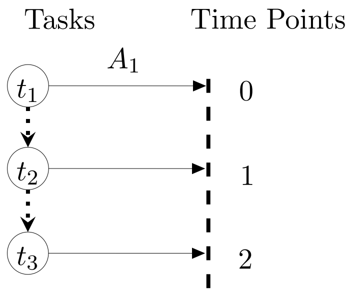


Figure 5.6: Resolve conflicts that disallow maximal matching by adding precedence constraints between tasks.

- $A = \{A_1, A_2, A_3\}$ corresponding to the three machines.
- $T = \{t_{j,h} | \mu_{j,h} \in O\}$ corresponding to each of the operations $\mu_{j,h}$.
- $T_j = \{t_{j,h}\}$ if $\mu_{j,h}$ is associated with machine M_j.

- $t_{j,h} \prec t_{j+1,h}$ for each $j \in \{1,\dots,n\}$.
- $r() = 0$; and $d() = \infty$.
- $l() = 1$; and $c() = 1$.
- $M_{\mathrm{bCAS}} = M_{jobshop}$.

We next show that there exists a solution to the job-shop problem instance if and only if there exists a solution to the bCAS instance with unit tasks and sequential agents: Let $\sigma() : O \to N$ be a scheduling function that results in a minimum makespan schedule for the job-shop problem instance. To obtain a makespan minimal solution to the bCAS problem, we add a constraint $[\sigma(\mu_{j,h}), \sigma(\mu_{j,h})]$ corresponding to each task $t_{j,h}$, within which it must be scheduled. Clearly, the precedence constraint between any two tasks $t_{j,h}$ and $t_{j+1,h}$ is not violated since $\sigma(\mu_{j,h}) \le \sigma(\mu_{j+1,h}) - 1$ and since the makespan of the job-shop problem is less than $M_{jobshop}$, the bCAS problem also has a makespan less than M_{bCAS}.

Suppose on the other hand, that we had a set of interval constraints $[est(t), lst(t)]$ that ensure that the global makespan of the bCAS problem is less than M_{bCAS}. Clearly, such a set of interval constraints cannot violate any precedence constraint. Otherwise, this would imply that agents can no longer hope to construct a schedule for their local CAS-problems, so that the local schedules can be merged into a feasible global schedule. One can easily derive a makespan minimal schedule for the job-shop problem by simply scheduling each operation $\mu_{j,h}$ at any time point within the interval $[est(t_{j,h}), lst(t_{j,h})]$. Further since $M_{\mathrm{bCAS}} = M_{jobshop}$, we claim that the bCAS problem is NP-complete even when we can efficiently detect resource conflicts. $\square$

So far, we developed an autonomous coordination mechanism for solving a given class of bCAS instances Π_b, and showed that solving this class of bCAS instances remains NP-hard, although every agent can efficiently detect its resource conflicts. To resolve conflicts, we proposed a mechanism $\mathrm{M}_{\mathrm{ISAsq}}$ that ensures autonomy of agents. We now investigate the consequence of using such an autonomous coordination mechanism on the makespan of the global schedule.

Unfortunately, as we will show shortly, the performance of $\mathrm{M}_{\mathrm{ISAsq}}$ can be arbitrarily bad in terms of the worst case makespan. More specifically, we demonstrate that: (i) there exists a set of instances within $\Pi_b = \langle T, A, \prec, \phi, l() = 1, c() = 1, r(), d() \rangle$, where applying $\mathrm{M}_{\mathrm{ISAsq}}$ results in a makespan equal to $|T|$; and moreover (ii) the price of autonomy of $\mathrm{M}_{\mathrm{ISAsq}}$ can be arbitrarily close to the number of agents $|A|$.

Proposition 5.3. $\mathrm{M}_{\mathrm{ISAsq}}$ *can result in a worst case makespan of* $|T|$.

Proof. The longest chain that any acyclic task graph can form is the number of tasks. Notice that once a total order on the tasks in T is imposed, the length of the chain is $|T|$. In such a case, task with depth i will be assigned by ISA an interval $[i, i]$ for $0 \leq i \leq |T| - 1$, and there is no conflict between any two tasks and every agent has a conflict-free schedule. Thus we conclude that a trivial upper bound on the makespan generated by ISA is $|T|$.

We now show that for a class of task graphs, the makespan generated by $\mathrm{M}_{\mathrm{ISAsq}}$ can reach this upper bound $|T|$.

Let us look at an example instance shown in Figure 5.7 with 3 agents. The tasks are classified into 6 groups with different depths from $0, 1, \ldots, 6$, i.e., group j consisting of tasks with depth $j - 1$, for $1 \leq j \leq 6$. The tasks are assigned to agents in a way such that A_i has all tasks in groups $j \cdot 3 + i$ for $j = 0, 1$. As an example, A_1 has all tasks belonging to group 1 and group 4, with depths 0 and 3 respectively. The precedence constraints of tasks are shown in the figure, where the highest numbered task in group j is constrained by every task in group $j - 1$ while the lowest numbered task in group j is only constrained by one task in group $j - 1$. For instance, t_1 precedes t_4, t_5 and t_6, t_2 precedes tasks t_5 and t_6 and t_3 precedes t_6.

In this instance, assume all agents resolve conflicts such that higher indexed tasks always precede the lower indexed ones. For instance, A_1 adds constraints $t_3 \prec t_2 \prec t_1$ and $t_{12} \prec t_{11} \prec t_{10}$. Then the makespan of the whole system would be equal to $|T| = 18$, i.e., it reaches the worst case makespan that can be generated by any mechanism that uses ISA. Note that this result holds when we increase the size of the coordination instance by adding more agents, or by adding more tasks to each task group, as long as the instance structure stays same as Figure 5.7. □

Proposition 5.4. *The price of autonomy of mechanism* $\mathrm{M}_{\mathrm{ISAsq}}$ *can be arbitrarily*

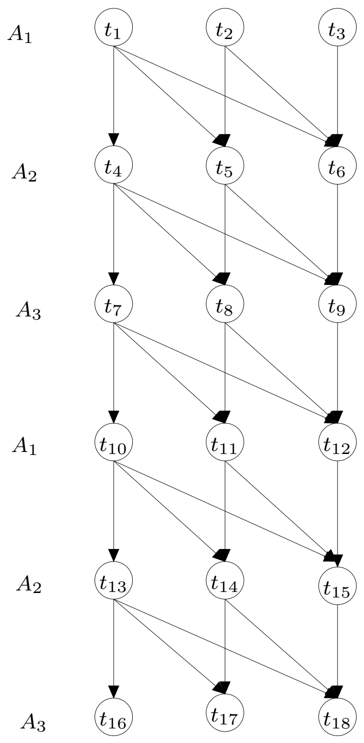


Figure 5.7: A coordination instance that shows $\mathrm{M_{ISAsq}}$ results in a worst case makespan of $|T|$.

close to $|A|$, where $|A|$ is the number of agents in Π_b.

Proof. We now show the price of autonomy that $\mathrm{M_{ISAsq}}$ can achieve with the class of instances as shown in Figure 5.7 is arbitrarily close to $|A|$. We will look at a generalisation of the coordination instance in the figure. Suppose there are $|A| = n$ agents, each having p groups of tasks. Each task group contains k tasks. The structure of the precedence constraints among tasks is same as that in Figure 5.7. So, according to Proposition 5.3 the worst case makespan $\mathrm{M_{ISAsq}}$ can obtain is $|T| = k \cdot p \cdot n$.

The makespan given such instances can not be less than $p \cdot n - 1 + k$, since $p \cdot n$

is the longest chain in the original task graph. Suppose this is not true and there is a smaller makespan $m < p \cdot n - 1 + k$. Take the last task of the last agent A_n. This task has to be completed at time m. Hence, the first task in the last task group of A_n must start at time $m - k < p \cdot n - 1$. However, a task of A_n can only start if all the $p \cdot n - 1$ tasks preceding it in the chain have been completed, that takes $p \cdot n - 1$ time in total. This leads to a contradiction.

Hence, the makespan is at least $p \cdot n - 1 + k$. In addition, this optimal makespan is only possible when $n \geq k$, since if not, agents cannot start to execute their next group's tasks until all p tasks in the previous group are completed.

We now show that the best case makespan $p \cdot n - 1 + k$ can be achieved. We use the example in Figure 5.7 where $n = k = 3$ and $p = 2$. Assume all agents resolve conflicts such that lower indexed tasks always precede the higher indexed ones, then the makespan of the conflict free schedule would be $p \cdot n - 1 + k = 8$.

Together with Proposition 5.3, we compute the price of autonomy $\mathrm{P_a}$ of $\mathrm{M_{ISAsq}}$:

$$\mathrm{P_a} = \frac{k \cdot p \cdot n}{p \cdot n - 1 + k}$$

For the worst case $\mathrm{P_a}$, $k = n$. thus we have $\mathrm{P_a} = \frac{p \cdot n^2}{p \cdot n - 1 + n}$. When p approaches infinity, i.e., when agents have more and more groups of tasks to complete, we have $\lim_{p \to \infty} \mathrm{P_a} = n = |A|$. □

As we have shown, with autonomous coordination mechanism $\mathrm{M_{ISAsq}}$, the global makespan could hit the upper bound of the makespan generated by any non-trivial algorithm. We demonstrated this by applying $\mathrm{M_{ISAsq}}$ to a class of coordination instances which have a special coordination structure. We then concluded that in general, the price of autonomy of $\mathrm{M_{ISAsq}}$ cannot be bounded by a constant and can be as close as the number of agents in the coordination instance.

There are several possible directions to circumvent such undesirable coordination outcomes. First of all, as we found out in our discussion of planning problems, not all problems are equally hard. It was possible for us to identify subclasses of problems which can be solved with better guarantees on the solution quality than in the general case. Therefore, we follow the same strategy here. More specifically, we now investigate a class of coordination instances with special task structures called *chain* *structured* bCAS *problems.* We hope that the performance of M_{ISAsq} will be better for problem instances in this special subclass.

5.4 Performance of M_{ISAsq}, given bCAS problems with special task structures

We expect the performance of M_{ISAsq} to vary for different classes of task graphs. Therefore in this section, we study a class of bCAS problem instances with a special structure — chains— and evaluate the performance of M_{ISAsq} on it. Chain structures are characterised by tasks having a maximum of one predecessor and one successor.

Definition 5.4. Chain structured bCAS **problem instances:** A given bCAS problem instance $\Pi_b = \langle T, A, \prec, \phi, l() = 1, c() = 1, r(), d() \rangle$ is considered a chain structured instance if it holds for all $t \in T$ that $in(t) \leq 1$ **and** $out(t) \leq 1$. ■

A simple subclass of chain structured instances is the class of *grids* where all the chains have the same length and all tasks of each depth are assigned to a single agent.

Definition 5.5. Grids: An (n,k) grid is a set of k task chains each of length n and a set of n agents such that agent A_i has to perform all tasks at depth i in the chains. ■

We now investigate the performance of M_{ISAsq}, on a grid structured instance Π_b. Firstly, we establish the best makespan that can be achieved for Π_b with grids.

Proposition 5.5. *Given a grid* $\Pi_b = \langle T, A, \prec, \phi, l() = 1, c() = 1, r(), d() \rangle$, *the minimal makespan for a conflict free schedule for all agents in an* (n, k) *grid is* $n + k - 1$.

Proof. First we need to show that the makespan cannot be less than $n + k - 1$. The proof is similar to the proof provided for Proposition 5.4. Suppose there is a smaller makespan $m < n + k - 1$. The first task of the last agent A_n must have been completed at time $m - (k - 1)$. Therefore, this task must start at time $m - k < n - 1$.

However, a task of agent A_n can only start if all the $n-1$ tasks preceding it in the chain have been completed. Hence it is a contradiction.

We now show that there is a critical path of length $n+k-1$ in an (n,k) grid such that no agent has a scheduling conflict for its set of tasks. Construct an arbitrary linear order $(t_{1,1},\dots,t_{1,k})$ of the set of tasks $T_1 = \{t_{1,1},\dots,t_{1,k}\}$ of agent A_1. The resulting critical path in the (n,k) set of chains has length $n+k-1$.

Given this order $(t_{1,1},\dots,t_{1,k})$, the ISA will produce for every task $t_{i,j}$ an upper and a lower bound: $est(t_{i,j}) = lst(t_{i,j}) = (i+j-2)$. This implies that for every agent A_i there is a conflict-free schedule, ensuring that the last task $t_{n,k}$ of the last agent is started at time $(n+k-2)$, and thus is completed at time $n+k-1$. □

The above proposition establishes the optimal makespan for scheduling a grid. Our next step is to establish the worst case makespan resulting from $\mathrm{M}_{\mathrm{ISAsq}}$ when dealing with grids.

Proposition 5.6. $\mathrm{M}_{\mathrm{ISAsq}}$ *results in a worst case makespan of* $n \cdot k = |T|$ *if the input is a* (n,k) *grid.*

Proof. To prove the proposition all we require to show is that there exists at least one situation where a series of *bad* decisions by $\mathrm{M}_{\mathrm{ISAsq}}$ leads to a makespan of $n \cdot k$. We next show that such an example situation exists.

Consider the situation in Figure 5.8. For ease of discussion let us view the (6,9) grid as being composed of three (6,3) sub-grids. Now suppose $\mathrm{M}_{\mathrm{ISAsq}}$ starts resolving conflicts between tasks belonging to agent A_6. It can first resolve the conflict between tasks $(t_{46} \prec t_{47})$ of the first sub-grid and then the $t_{49} \prec t_{50}$ of the second sub-grid and finally tasks $t_{52} \prec t_{53}$ of the last sub-grid. After these three steps, there are still conflicts that do not allow for a sequential schedule. Therefore, $\mathrm{M}_{\mathrm{ISAsq}}$ adds precedence constraints $t_{47} \prec t_{48}$, $t_{50} \prec t_{51}$, $t_{53} \prec t_{54}$.

After adding these constraints, A_5 has conflicting tasks since ISA provides the intervals $[4,4]$ to tasks t_{37}, t_{40} and t_{43}, $[4,5]$ to tasks t_{38}, t_{41} and t_{44}, $[5,6]$ to t_{39}, t_{42} and t_{45}. $\mathrm{M}_{\mathrm{ISAsq}}$ repeats the process of resolving conflicts of agent A_6 on agent A_5, but with a crucial difference. It now makes the predecessor of task t_{46} to *succeed* the predecessor of t_{47} and similarly, it makes the predecessor of t_{49} to succeed predecessor of t_{50} and predecessor of t_{52} to succeed predecessor of t_{53} and so on.

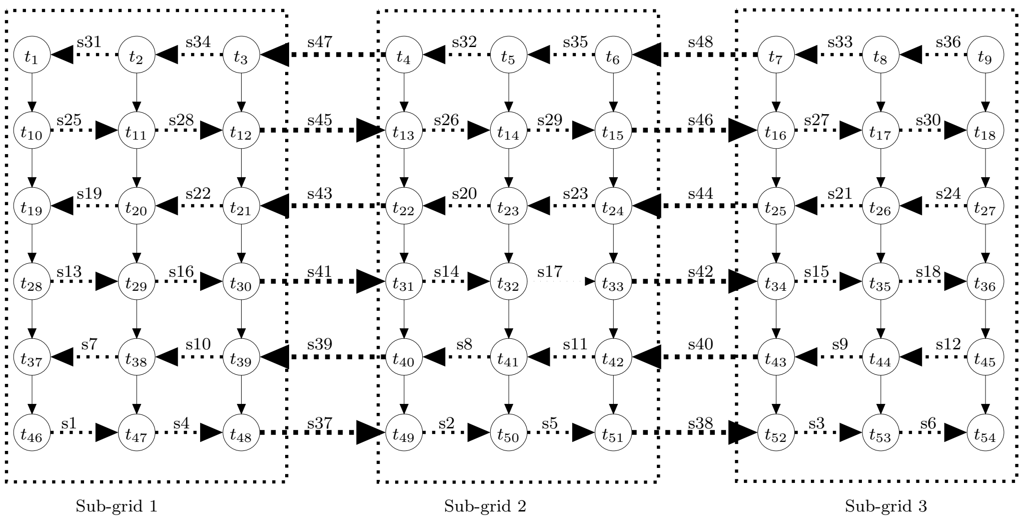


Figure 5.8: Worst case sequence of steps of M_{ISAsq} on a grid. Label *sX* indicates that a conflict resolution constraint is added between two tasks t_i, t_j in step X. The solid arcs represent the precedence constraints that are part of the original problem.

M_{ISAsq} then repeats this process of *(i)* choosing the *deepest* agent with conflicting tasks, *(ii)* strictly alternating between sub-grids for conflict resolution and finally *(iii)* alternating the direction of precedence constraints.

At this stage M_{ISAsq} has created three chains of length $\frac{k}{3} \cdot n$ from our original (6,9) grid. However, conflicts still remain. But now the conflicts are between tasks belonging to different sub-grids. These conflicts can be resolved first between tasks of the first and second sub-grids $t_{48} \prec t_{49}$ and then between second and third sub-grids $t_{51} \prec t_{52}$ in each agent, again starting from A_6 and proceeding to A_1. This results in a chain of length 54 as shown in the figure.

The construction shows clearly that M_{ISAsq} results in a makespan of $n \cdot k$ thus proving the proposition. □

It follows that the price of autonomy of M_{ISAsq} increases with the number of agents.

Lemma 5.7. *The price of autonomy of* $\mathrm{M}_{\mathrm{ISAsq}}$ *is* $\frac{|A|}{2}$ *if the input to it is a grid.*

Proof. Given the worst case performance of $\mathrm{M}_{\mathrm{ISAsq}}$ and the optimal makespan on grids, we have the price of autonomy $\mathrm{P}_{\mathrm{a}} = \frac{n \cdot k}{n+k-1}$. This ratio is maximal when $n = k$. Thus we have:

$$\mathrm{P}_{\mathrm{a}} = \frac{n^2}{2n-1} = \frac{|A|^2}{2|A|-1} \approx \frac{|A|}{2}.$$

□

Unfortunately, as we have seen, even for such simple coordination instances as grids, the price of autonomy of mechanism $\mathrm{M}_{\mathrm{ISAsq}}$ is unbounded. Therefore, it seems that we will have to compromise on agent autonomy if better global makespan is more desirable. One idea of restricting agent autonomy for efficiency is that the mechanism could make some specific choices when resolving conflicts. Such specific choices can be made based on the information of coordination instances. This idea becomes obvious when we look again at the result of Proposition 5.4, where we have proved that when the bCAS instance is a grid and we let the first agent A_1, i.e., the agent whose tasks are all at depth 0 in the grid, completely resolve its conflicts before the other agents, then the optimal makespan is obtained. Note that the order in which agent A_1 orders its tasks does not matter. In this strategy, a total order is established among all source tasks first and hence we call this strategy as the *source first heuristic.*

Recall that we are dealing with chain structured tasks in this section. Therefore, while we know that the source first heuristic results in optimal makespan for grids, we would next like to investigate whether this heuristic ensures *good* makespan for all chain structured bCAS problem instances. Thus we next investigate if the source first heuristic is effective in the case of *unequal chain* instances.

Performance of $\mathrm{M}_{\mathrm{ISAsq}}$ with source first heuristic on unequal chains We have seen that for grids, minimal makespan can be ensured if we use the source first heuristic. Buoyed by this fact, we hope to find that this heuristic would benefit more general types of chain structured instances. We now investigate the case of unequal chains. Unequal chain problem instances differ from grids in that they do not have any restriction on the lengths of the chains in the task graph.

Definition 5.6. Unequal chains: Unequal chain problems instances are bCAS problem instances such that, their task graph consists of a set $\bigcup_{i=1}^{l} m_i$ task chains, where the i^{th} task chain has a length of n_i. Furthermore, agent A_j is assigned to perform all tasks at depth j in the chains. All tasks have unit length (1). ∎

Note that in unequal chain problem instances, we still have the restriction that all tasks allocated to an agent have the same depth. Therefore, quite clearly, if we apply $\mathrm{M}_{\mathrm{ISAsq}}$ along with the source first heuristic, we would first resolve all conflicts in T_1. Further notice that, as a result of allowing different chain lengths, agent A_1 has the largest number of tasks to perform since each chain has a task at depth level 0. To analyse the performance of $\mathrm{M}_{\mathrm{ISAsq}}$ on unequal chain problem instances, let us first denote the cardinality of the agent A_1 as T_{max} and further denote the length of the longest chain of the chain graph as L_{max}.

Suppose we adopt the source first heuristic, then the following proposition shows that we cannot hope to always obtain minimal makespan schedules if $\mathrm{M}_{\mathrm{ISAsq}}$ were given an unequal chain graph as input. It also establishes that the worst case makespan using source first heuristic is *bounded by a constant.*

Proposition 5.8. *Given an unequal chain problem instance with sequential agents, the price of autonomy of* $\mathrm{M}_{\mathrm{ISAsq}}$ *is arbitrarily close to 2 if the source first heuristic is used.*

Proof. Since all tasks allocated to an agent have the same depth, totally ordering the tasks of agent A_1 is enough to remove conflicts in all other agents. To show that this indeed is the case, suppose that the tasks of agent A_1 are totally ordered. Now consider any arbitrary agent A_i and any two chains m_1, m_2 such that m_1 has a length of n_1 and m_2 has a length of n_2. Let t_1 be the first task in m_1, t_2 the first task in m_2, t_x the task in m_1 belonging to A_i and t_y the task in m_2 belonging to A_i. Suppose that the source first heuristic adds the precedence constraint $t_2 \prec t_1$. As a consequence, $depth(t_x) > depth(t_y)$ always because of the precedence constraint between t_2 and t_1. This implies that the predecessor of t_x has the same depth as t_y. However, the predecessor of t_x belongs to an agent other than A_i. Therefore, tasks t_x and t_y do not conflict. Extending this argument, we could claim that once tasks in T_1 are totally ordered, for any other agent, the depths of all their tasks are different. Consequently, the $est()$ values for their tasks are unique (within the set of tasks allocated to an agent). Since the duration is 1 for all tasks, none of the tasks have any more conflicts.

Now let us look at the makespan of such a solution. Recall that in a unequal chain instance $T_{max} = T_0$. Therefore, the longest possible makespan is equal to $|T_0| + L_{max} - 1$, which occurs when the longest chain is scheduled last. Clearly, the minimum possible makespan cannot be less than $\max\{|T_0|, L_{max}\}$. Otherwise, there would be at least one single task that cannot be accommodated in the schedule.

$$\frac{|T_0| + L_{max} - 1}{\max\{|T_0|, L_{max}\}} \approx 2 \tag{5.1}$$

Thus proving that the price of autonomy of $\mathrm{M}_{\mathrm{ISAsq}}$ is arbitrarily close to 2 if source first heuristic is used by $\mathrm{M}_{\mathrm{ISAsq}}$ on an unequal chain graph. □

We have seen that the source first heuristic cannot ensure optimal makespan in the case of unequal chains. However, it is still effective since it offers a constant factor upper bound. Unfortunately, it is not an universal strategy that works on any coordination instance. For example, for the instance in Figure 5.7, the worst makespan $|T|$ holds no matter whether we ask some specific agent to resolve conflicts first or not. This has been shown in the proof for Proposition 5.4. Therefore, there is still a case for finding heuristic strategies that can be applied over more general classes of bCAS problem instances. We leave it to future research to develop an universal strategy that achieves a good makespan while not compromising agent autonomy.

So far we have discussed the performance of $\mathrm{M}_{\mathrm{ISAsq}}$ when bCAS problem was chain structured. Disappointingly, we also showed that even with an additional heuristic for conflict resolution, $\mathrm{M}_{\mathrm{ISAsq}}$ was effective only if all tasks allocated to an agent had the same depth and the instance was chain structured, as in the case of grids and unequal chains. We have already shown in Proposition 5.3 that if the task graph is not chain structured, then there exists a class of bCAS problem instances that result in $\mathrm{M}_{\mathrm{ISAsq}}$ giving a worst case makespan of $|T|$. However, the good news is that $\mathrm{M}_{\mathrm{ISAsq}}$ always ensures that agents can construct their schedules independently.

Notice that the Ford-Fulkerson method played a crucial role in designing solutions for chain structured instances. Without the ability to detect resource conflicts efficiently, a solution to the bCAS problem would have been impossible. However, we made an important assumption before applying the Ford-Fulkerson method. We assumed that tasks were always of unit duration and that agents were sequential. Clearly, this need not be true. Therefore, in the next section we deal with bCAS problem instances where agents are non-sequential and tasks are of non unit duration.

5.5 Adapting $\mathrm{M}_{\mathbf{ISAsq}}$ for general bCAS instances Π_b

If agents are non-sequential and tasks are of various durations, $\mathrm{M}_{\mathrm{ISAsq}}$ cannot be applied directly. This is illustrated in the following example:

Example 5.5. Consider again the job shop scheduling problem of Example 4.1. Clearly task durations are not homogeneous. In fact more often than not, tasks differ in the time they take to get processed. All our effort in finding suitable schedules until now would go a waste in such scenarios. We can neither use $\mathrm{M}_{\mathrm{ISA}}$ because we cannot assume that agents have unbounded concurrency nor can we use $\mathrm{M}_{\mathrm{ISAsq}}$ because we cannot find a maximum matching if the tasks are of different lengths. ■

Hence, we need to adapt $\mathrm{M}_{\mathrm{ISAsq}}$ in order to solve general bCAS instances.

Handling non-sequential agents

Our first step is to generalise $\mathrm{M}_{\mathrm{ISAsq}}$ so that it can handle non-sequential agents. It is rather straightforward to do that. The trick to this generalisation is in the way we construct the flow network from the bipartite graph.

The construction of the flow network still relies on the same bipartite graph $G_i = (T_i \cup I_i, E_i)$. The only change would be to set the capacity of the edges to the sink node w to be equal to $c(i)$.

If we can find a maximum flow f_i whose value $|f_i|$ is equal to the number of tasks $|T_i|$, i.e., $|f_i| = |T_i|$, we can then be sure that the agent A_i can find a valid schedule such that its concurrency bound is not violated. In case we are unable to do so, we know that for some task t, $f_i(s,t) = 0$. We say that task $t \in T_i$ is not contained in the maximum flow if $f_i(s,t') = 0$, and t' is conflicting with some other task $t \in T_i$ where $f_i(s,t) = 1$. If this is the case, we resort to our earlier technique of adding precedence constraints between conflicting tasks: we take a task t contained in the maximum flow f_i and a conflicting task t' not in f_i whose time interval is overlapped with t's intervals. Whenever such a pair of tasks are found, we add $t \prec t'$ or $t' \prec t$ to resolve the conflict. We continue doing so until we are able to find a maximum flow whose value is $|T_i|$.

Let us illustrate the method through the following example.

Example 5.6. Figure 5.9 shows how to use the idea of flow network built for detecting conflicts among 4 tasks of agent A. Assume initially that ISA assigns the following time intervals for the tasks: $t_1 : [0, 0]$, $t_2 : [0, 0]$, $t_3 : [0, 0]$, $t_4 : [1, 1]$. Suppose agent A's concurrency is 2. We then build a flow network as shown in the left graph of Figure 5.9. We can use the Ford-Fulkerson method (see [Ford and Fulkerson 1957]) to find the maximum flow f. It is not difficult to see that the value of f is 3. Suppose $f(s, t_1) = f(s, t_2) = f(s, t_4) = 1$ and $f(s, t_3) = 0$. We add a precedence constraint $t_2 \prec t_3$ between the conflicting tasks t_2 and t_3. If we run ISA again, the resulting intervals of 4 tasks become: $t_1 : [0, 0]$, $t_2 : [0, 0]$, $t_3 : [1, 1]$, $t_4 : [1, 1]$. Based on these intervals, a new flow network has been constructed, as shown in the right graph of Figure 5.9. The maximum flow that can be found has value 4. Thus, no more conflicts exist within agent A.

■

As we have shown, one of the attractive features of the unit (or homogeneous) duration tasks case is the existence of a polynomial decision procedure for deciding whether there exists a schedule satisfying the constraints $c_i(t)$ for an agent A_i. Unfortunately, in most real life scenarios such an assumption is impractical. Once the assumption of homogeneous durations is dropped, with non-sequential agents, even finding a single agent schedule would be NP-hard as was shown in Theorem 5.1.

Fortunately, we can reuse $\mathrm{M}_{\mathrm{ISAsq}}$ to solve bCAS problems with heterogeneous durations through a simple transformation. The idea behind the transformation is based on the fact that in several systems, tasks can be stopped and restarted. Tasks which can be stopped and restarted at arbitrarily points during their processing are called *preemptive* tasks. This process of stopping a task during execution and resuming their execution later is called preemption. The idea of preemption has been successfully used over the years for several scheduling problems including process

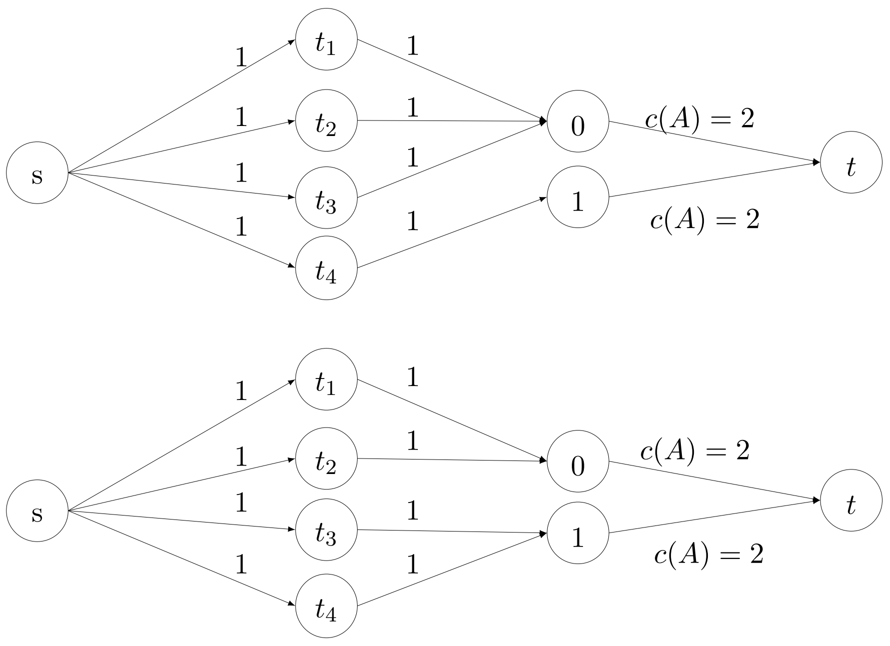


Figure 5.9: Maximum flow for detecting conflicts with a non-sequential agent.

scheduling and data transmission among others. Therefore, we hope that preempting tasks might allow us to solve bCAS problems with heterogeneous durations. We next present a straightforward adaptation of $\mathrm{M}_{\mathrm{ISAsq}}$ to compute intervals for preemptive tasks.

Dealing with heterogeneous tasks

We can reuse $\mathrm{M}_{\mathrm{ISAsq}}$ if we assume that tasks can be preempted. This allows agents to complete a part of task t, then start some other tasks and then process a next part of t and so on. If this is allowed, we can reduce a bCAS problem instance with arbitrary duration tasks to a bCAS problem instance with unit duration tasks as follows:

1. Each task $t \in T$ with duration $l(t)$ is split into unit parts $t^1, \ldots, t^{l(t)}$ and

2. We add the constraints $t^j \prec t^{j+1}$ for $j = 1, \ldots, l(t) - 1$ and finally,

3. Every precedence constraint $t \prec t'$ is replaced by the constraint $t^{l(t)} \prec t'^1$.

Note that the splitting task procedure is only polynomial for those instances where the durations $l(t)$ are not super-polynomial in the number of tasks $|T|$. Otherwise, the splitting of tasks into unit duration tasks would result in a super polynomial number of unit-duration tasks. To illustrate the consequence of allowing tasks to have large durations, suppose we have a situation where the set of tasks T contains a task t_i with $l(t_i) = 2^{|T|}$, the $\mathrm{M}_{\mathrm{ISAsq}}$ mechanism would have an input of size $2^{|T|}$ tasks. As a result it could take an exponential amount of time to compute the schedules.

Let us illustrate the above through an example.

Example 5.1. Consider the job shop problem of Example 4.1. We split task t_1 into t_1^1 and t_1^2. Similarly we split task t_5 into t_5^1 and t_5^2 and task t_3 into t_3^1 and t_3^2 (see Figure 5.10). If we run ISA with this set of tasks, we get the following starting time intervals for the tasks:

$$\begin{aligned}
&t_1^1 = [0,0];\ t_1^2 = [1,1];\ t_2 = [2,2];\ t_3^1 = [3,3];\ t_3^2 = [4,4] \\
&t_4 = [0,1];\ t_5^1 = [2,3];\ t_5^2 = [4,4]
\end{aligned}$$

To determine if agents can find a sequential schedule (we assume sequential agents for ease of explanation) we run the matching procedure. We start with agent A_1. Suppose tasks t_1^1 and t_1^2 are part of the matching but t_4 is not. We then add a precedence constraint $t_1^2 \prec t_4$ and rerun ISA. The new set of intervals are as follows:

$$\begin{aligned}
&t_1^1 = [0,0];\ t_1^2 = [1,1];\ t_2 = [2,2];\ t_3^1 = [3,3];\ t_3^2 = [4,4] \\
&t_4 = [2,2];\ t_5^1 = [3,3];\ t_5^2 = [4,4]
\end{aligned}$$

It is now possible to find a maximum matching for all agents implying that agents can find sequential schedules within the suggested intervals. ■

This idea of preemption can be developed further to also handle non-preemptive tasks. The idea is to somehow ensure that all parts of a task are scheduled consecutively. That is, we could first treat them as preemptive tasks, then break them up

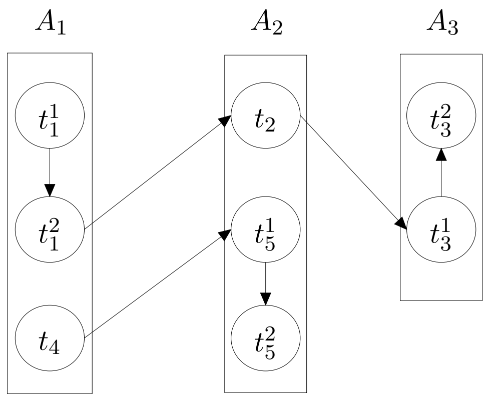


Figure 5.10: Reducing the job shop problem to an equivalent unit-duration case if preemption is allowed.

into unit duration tasks and then some how ensure that all parts of each task gets a consecutive interval and finally, combine the tasks and their intervals and execute it as one whole task. In summary, we could derive a non-preemptive schedule as follows:

1. We first preempt tasks into unit duration tasks so that we can decide whether there exists a feasible schedule for every agent.

2. If there is a conflict between two tasks t_a^p and t_b^q where t_a^p and t_b^p are the p^{th} and q^{th} parts of tasks t_a, t_b respectively, then we make all preempted parts of task t_a to precede all preempted parts of task t_b by adding a precedence constraint $t_a^{l(t_a)} \prec t_b^1$.

3. Finally we merge the preempted parts of tasks and run ISA on them again to get a conflict free schedule.

This process of first splitting and then merging tasks is called *pseudo preemption.* Due to step 2 of the procedure, all parts of each task always get consecutive intervals. Therefore, they can always be merged into a single unbroken interval. Once such an interval is derived, we can run the task within that interval in an uninterrupted way.

We have so far developed methods based on ISA that ensure scheduling autonomy. However notice that in general, bCAS problem instances need not have *nice* task graph structures such as grids. This means that heuristics such as source first heuristic will be ineffective in such scenarios. However, one could benefit from the general idea of resolving conflicts in an agent completely before proceeding to resolve conflicts in other agents. The intuition is that, in case a particular problem instance has some agent which is allocated a large portion of predecessor free tasks, then as soon as all conflicts in this agent are resolved, the whole instance would have much fewer conflicts (because many successors of these tasks are now conflict-free). This might help reduce the global makespan as it did in the case of unequal chains. In the very least it might offer a chance to complete conflict resolution faster. Thus in this updated mechanism called $\mathrm{M_{ISAn}}$, we arbitrarily choose an agent for conflict resolution, resolve all conflicts within the agent and then proceed to resolve conflicts among other agents. The complete mechanism is given as Algorithm 9.

Example 5.7. For the job shop problem of Example 4.1 the intervals computed by ISA are shown in the Figure 5.11a. Although the tasks are non-preemptive we preempt them for scheduling purposes into unit duration tasks and compute the intervals for them as shown in Figure 5.11b. We find that in the example task t_1^1, t_1^2 together with t_4 are conflicting for time points 0 and 1 and tasks t_2, t_5^1 are also conflicting for time point 2 as evidenced by the intervals shown in Figure 5.11b. The conflicts for agent A_1 are shown in the left graph of Figure 5.11c. To resolve the conflicts, we add precedence constraints between all parts of task t_1 and task t_2 as shown in Figure 5.11c to resolve the conflicts and then combine the tasks back to finally compute a non-preemptive interval for the tasks as shown in the Figure 5.11d. The final makespan is still 5. ■

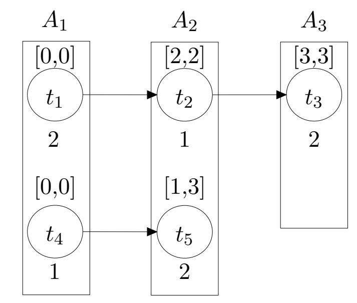
5.6 Summary

We started our study in this chapter with a hope to extend ISA so that we could ensure minimum makespan in bCAS problem instances. However, we immediately

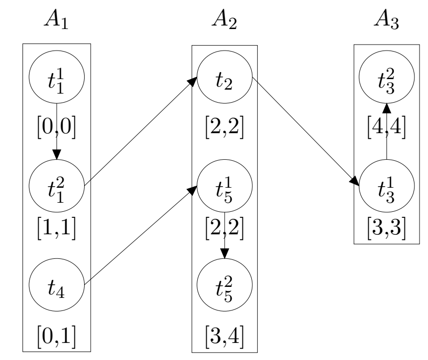
Algorithm 9 Updated mechanism M_{ISAsq} for non-preemptive tasks (M_{ISAn})

```
1: **Inputs:** constraint intervals $[est(t_k), lst(t_k)]$ for all the tasks $t_k \in T$ as computed by ISA.
2: **Outputs:** Revised $est(t_k)$ and $lst(t_k)$ for each $t_k \in T$ to enable scheduling.
3: Split each task $t_k$ into a chain of unit duration tasks $t_k^1 \prec t_k^2 \prec \ldots \prec t_k^{l(t_k)}$ and compute the constraint intervals $[est(t_k^l), lst(t_k^l)]$ for each unit duration task using ISA;
4: for each agent $A_i$ do
5:    while there exists a $(T_i, C_i)$ not allowing for a maximum flow $|f_i| = |T_i|$ do
6:       take a task $t_a^l$ contained in $f_i$ and an overlapping task $t_b^j$ not occurring in $f_i$;
7:       add $t_a^{l(t_a)} \prec t_b^1$ to the partial order $(T, \prec)$;
8:       run ISA on the updated partial order;
9:    end while
10:   combine all the parts of each task $t_a$, and add $t_a \prec t_b$ if $t_a^{l(t_a)} \prec t_b^1$;
11:   run ISA on the updated partial order $(T, \prec)$ and update the set of constraint intervals $C$
12: end for
```

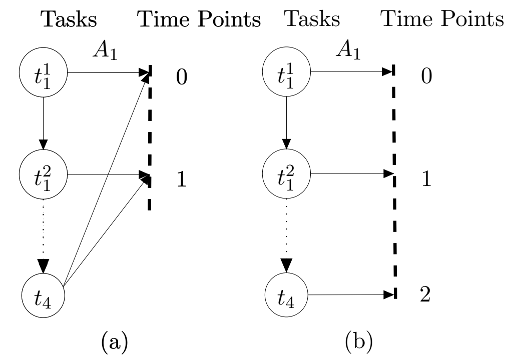
found out that solving bCAS problems to ensure makespan minimality would in general be intractable. The intractability was traced to the agent's inability to detect and resolve resource conflicts optimally. We then tried to investigate if there were any subclasses of bCAS problems with special task graph structures where conflict detection and resolution could be efficiently performed. Fortunately, for the case when agents were sequential and tasks had unit durations, resource conflicts could be efficiently detected through the use of the well known maximum matching procedure of Ford and Fulkerson. However, this extended algorithm which used ISA and maximum matching called M_{ISAsq}, still did not ensure makespan minimality. Therefore, we looked at even more special structures, namely chain structured task graph instances. However, even for this class of instances we showed that M_{ISAsq} could not ensure makespan minimality. We then investigated if additional heuristics could help reduce the makespan. We found that the source first heuristic while



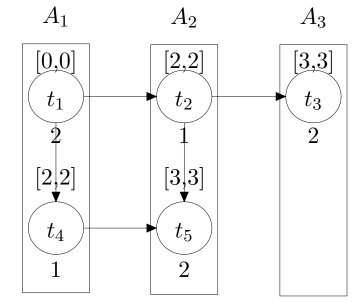
(a) Initial starting time intervals as computed by ISA.



(b) Intervals after pseudo preemption.



(c) Conflict resolution.



(d) Final task graph and intervals.

Figure 5.11: An example to illustrate $\mathrm{M}_{\mathrm{ISAsq}}$ (Algorithm 9).

effective in case of grids was less effective in the case of unequal chains.

Thus, in this chapter we have shown that ensuring autonomy comes at a cost in terms of efficiency. Even for special classes of bCAS problem instances, namely grids, the price of autonomy is not bounded by any constant. Additional heuristics such as the source first heuristic, while effective against grids and unequal chains, also cannot help much when faced with more general classes of chain structured problems. Naturally, these negative results hold for general bCAS instances as well. We leave the design and development of more advanced heuristics that could improve the performance of algorithms such as $\mathrm{M}_{\mathrm{ISAsq}}$ to future research.

Another important aspect is that of heterogeneous task durations and heterogeneous concurrency bounds. We were able to show in this chapter that $\mathrm{M}_{\mathrm{ISAsq}}$ could be generalised to $\mathrm{M}_{\mathrm{ISAn}}$ where both these issues can be effectively handled.

Although we have discovered that the price of autonomy is quite large in the worst case, all is not over yet. What is interesting still, is to understand the performance of algorithms such as $\mathrm{M}_{\mathrm{ISAn}}$ when faced with real life applications. Practical applications could have task graph structures which are neither similar to Figure 5.7 nor a grid. In such cases, specially when agents have heterogeneous capacities, we are still unaware of how $\mathrm{M}_{\mathrm{ISAn}}$ will perform. Thus, to evaluate $\mathrm{M}_{\mathrm{ISAn}}$ in practical settings we will perform an empirical analysis of $\mathrm{M}_{\mathrm{ISAn}}$ in the next chapter.

Chapter 6

Empirical analysis

At the end of Chapter 5, we developed algorithm $\mathrm{M}_{\mathrm{ISAn}}$*, which designs a set of intervals within which tasks can be autonomously scheduled. However, we also showed that* $\mathrm{M}_{\mathrm{ISAn}}$ *could not guarantee minimum global makespan in the general. Therefore, in this chapter we test its effectiveness empirically.*

In many practical scenarios, for instance when batch processing machines are involved, agents can be non-sequential (with concurrency bounds greater than 1). In such cases, the analysis we performed in the previous chapter may not be sufficient to estimate the performance of $\mathrm{M}_{\mathrm{ISAn}}$. Therefore, in this chapter we would like to understand the loss in efficiency (makespan) of $\mathrm{M}_{\mathrm{ISAn}}$ when:

1. general task graphs are input to it and

2. when agents have non-unit and heterogeneous capacities.

In this chapter, we attempt to gain this understanding empirically. To perform empirical analysis, we choose the problem of coordinating schedules among ground handling service providers in an airport as a test domain. To explain why this domain was picked for our testing, we first enumerate a set of properties that are required to be satisfied by problem domains so that we can apply $\mathrm{M}_{\mathrm{ISAn}}$:

1. Task graphs representing the problem must be acyclic.

2. Allocation of tasks to agents must be known.

3. Durations of tasks must be known and should not be super-polynomial in the number of tasks.

4. Finally, data from real instances of the problem must be available.

Several practical problems such as shop scheduling, terrain exploration and military operations, contain instances that satisfy the first three properties. We have already seen, in examples used in the previous chapter, that job shop scheduling problems can be modelled as CAS problems and hence could be also used for evaluating the performance of $\mathrm{M}_{\mathrm{ISAn}}$. While there do exist some benchmarking data sets available for shop scheduling problems (cf. [Demirkol *et al.* 1998]), the focus of shop scheduling problems is to develop a central schedule whereas our effort is to allow for autonomous scheduling. Therefore this domain was not suitable for our purpose.

Problems in terrain exploration also exhibit properties mentioned above. Multiple exploration vehicles/teams would have to coordinate their exploration tasks so that they are able to complete their exploration as soon as they possibly can. Exploration tasks would have precedence constraints emerging from the routes that exploration teams have to follow, and further since the terrain is unexplored there is little hope of establishing communication between the teams. While this problem seems exciting, lack of practical data is a big drawback. Further, in most cases, exploration vehicles in practice are centrally controlled (cf. [Seeni *et al.* 2010]).

Military operations involving different teams could also be required to coordinate in silence and possibly disarm or destroy a hostile target. Quite obviously, getting access to such data is, to horribly understate the problem, very difficult.

Consider the problem of coordinating schedules among different service providers for ground handling activities in an airport. This problem deals with operations an air craft has to undergo once it lands in an airport, so that it is ready for its next flight. This set of operations is an acyclic partial order of tasks that need to be performed by multiple service providers. Service providers, however, might each have to serve different subsets of air crafts that arrive in the airport. Therefore, service providers would prefer to develop schedules for their tasks in a decentralised way.

As we have shown earlier, allowing agents (service providers) to schedule independently might lead to unnecessary delays and hence their schedules must be coordinated. Thus, this problem meets all the properties we require for an empirical analysis of $\mathrm{M}_{\mathrm{ISAn}}$. The idea of using this problem was inspired by the work in (cf. [Leeuwen *et al.* 2007]). Here the authors use Hunsberger's method to design robust schedules for ground handling service providers. Since Hunsberger's method can be successfully applied, we assumed that $\mathrm{M}_{\mathrm{ISAn}}$, a distant cousin of Hunsberger's method, could also be applied to this scenario. Fortunately, we were also able to obtain the arrival and departure data for a single day from Amsterdam (Schiphol) airport and therefore we chose this problem for our empirical analysis.

In the next section, we first present our expectations regarding the performance of $\mathrm{M}_{\mathrm{ISAn}}$. We then present an overview of the ground handling problem in Section 6.2 and then present a formal definition of the *scheduling ground handling activities (*SGHA*)* problem. In Section 6.4 we present an ILP model to optimally solve the SGHA problem and in Section 6.5 we present our experimental results and their analyses.

6.1 Expected behaviour of ISAN

We expect several parameters such as task graph structure, number of tasks and amount of work, to affect the performance of $\mathrm{M}_{\mathrm{ISAn}}$. In particular, we expect the following parameters to have a significant impact on its performance:

Task graph structure $\mathrm{M}_{\mathrm{ISAn}}$ computes a set of starting time intervals of the form $[est(t), lst(t)]$ for each task t in the given problem instance. The *longest makespan* from these set of intervals can be computed by

$$\max_{i=1}^{|T|}(lst(t_i) + l(t_i)). \tag{6.1}$$

In the previous chapter, we showed examples of problem instances where the longest makespan generated by $\mathrm{M}_{\mathrm{ISAn}}$ for these problem instances could be as long as the number of tasks. However, not all problem instances need to be as bad. If problem instances resemble the structure in Figure 5.7, then $\mathrm{M}_{\mathrm{ISAn}}$ could result in makespan lengths close to its worst case performance. On the other hand, if the problem instances resemble grids, then $\mathrm{M}_{\mathrm{ISAn}}$ could result in optimal makespan as well. We believe that given an arbitrary problem instance, the performance of $\mathrm{M}_{\mathrm{ISAn}}$ lies somewhere between these two extremes. Recall also from the previous chapter that, tasks are rarely of unit duration in practical scenarios. These non-unit duration tasks also have a say in the ultimate input to $\mathrm{M}_{\mathrm{ISAn}}$ (because of pseudo preemption). In summary, the structure of the partial order and the duration of tasks are expected to significantly influence the performance of $\mathrm{M}_{\mathrm{ISAn}}$.

Agent order Conflict resolution process in $\mathrm{M}_{\mathrm{ISAn}}$ has a significant influence on its performance. Recall that $\mathrm{M}_{\mathrm{ISAn}}$ randomly picks an agent to completely resolve its conflicts and proceeds to resolve conflicts in another randomly picked agent and so on. If the underlying task graph is a grid or an unequal chain, and if it happens that conflict resolution picks the first agent (whose tasks are of depth 0), then we already know from our previous chapter that it results in a shorter global makespan. However, it is not the case in general. Thus, we expect that the order in which agents are picked for conflict resolution has a significant impact on the global makespan.

Agent concurrency The concurrency of an agent is another factor that determines the number of conflicts that occur during matching (within $\mathrm{M}_{\mathrm{ISAn}}$). One can easily expect that higher values of concurrency result in lower number of conflicts and hence a better global makespan. Thus we expect that $\mathrm{M}_{\mathrm{ISAn}}$ would result in lower makespan whenever concurrency bounds are raised.

Number of tasks in the system Consider the bipartite matching procedure that is part of $\mathrm{M}_{\mathrm{ISAn}}$. As the number of tasks increase, it can be expected that greater number of tasks have conflicts. Given that agents have a bound on their concurrency we expect that as the number of conflicts increase, there are greater number of opportunities for $\mathrm{M}_{\mathrm{ISAn}}$ to make bad choices during conflict resolution and hence global makespan resulting from $\mathrm{M}_{\mathrm{ISAn}}$ could also increase. Therefore, we expect that the global makespan suffers when the number of tasks increase.

Amount of work per agent We define *work* as the sum of durations of all tasks allocated to an agent. It seems sensible to expect that agents with greater amount of work influence the makespan more significantly than agents with lesser amount of work. As a consequence, providing greater number of resources (concurrency in our case) to agents with greater amount of work could be expected to reduce the global makespan more than providing other agents with greater concurrency.

Given these expectations, we would like to evaluate and learn about the performance of $\mathrm{M}_{\mathrm{ISAn}}$. Before we embark on testing its performance, in the next section, we describe the problem of coordinating the schedules of ground handling activities at airport in detail.

6.2 Ground handling problem at an airport

Ground handling in airports refers to all the activities that need to be performed on an air craft after it has landed in an airport, and prior to its take off. The International Air Transport Association (IATA), proclaims that: "Ground handling covers the complex series of processes required to separate an air craft from its load (passengers, baggage, cargo and mail) on arrival and combine it with its load prior to departure."

Broadly, ground handling processes can be grouped into two major categories - *terminal* operations and *airside* operations. Terminal operations include services inside the airport terminal such as staffing the transfer counters, customer service counters and airline lounges. Airside operations include services that are performed on the air craft itself. Airside operations involve more complex tasks as well as greater diversity of equipment. Ashford (cf. [Ashford *et al.* 1997]) provides an exhaustive list of services that make up airside operations of an air craft. Airside activities have to be performed in a specified sequence. This sequence in which airside activities are performed on an air craft is termed the *turnaround process.* Different air crafts require different turnaround processes. The specific turnaround process associated with an air craft is specified by the manufacturer of the air craft. The technical specification manuals also specify the durations of each of the activities in the turnaround process of an air craft.

Given this background, suppose now that there are several service providers in the airport who handle different activities of ground handling. For instance, there might be multiple service providers who supply food to the air crafts, based on the specific agreements the airline companies have. In such a case, developing a centralised schedule for all the ground handling activities might not be a wise decision, since a failure or delay by any one of the service providers might require the entire schedule to be recomputed. Therefore, it is desirable to allow agents autonomy in constructing their local schedules. The problem then, as we pointed out earlier, is that unless these local schedules are coordinated, the global schedule might involve significant delays. Thus, we require some mechanism that while guaranteeing autonomy to agents, also reduces the overall makespan. In this chapter, we will evaluate $\mathrm{M}_{\mathrm{ISAn}}$ to see if it can satisfy such a requirement.

To limit the scope of our experiments, we were forced to make several restricting assumptions on the test domain. The next section presents this list of assumptions that were made.

Assumptions Ground handling is not an isolated process, it is closely tied to several other processes that happen within an airport. Safety regulations restrict air crafts of different models to use specific sets of runways leading to delays in the landing and taking-off of planes. Similarly, gate allocation needs to balance between available gates and preferences of airline companies. Taxiing of air crafts can also severely affect the overall efficiency of the airport as suboptimal taxiing routes can have cascading effects. Variations in any of these processes is likely to adversely affect ground handling. Handling the entire problem is immensely complicated and beyond the scope of our research. Therefore, we are forced to make assumptions regarding the other processes that affect ground handling. Below is a list of such assumptions.

- We assume that all air crafts arrive on time. This implies that flights arrive at the airport on time and that they are able to land and taxi to the gate without any delays.

- We assume that the gate allocation problem has been solved and the most suitable gate has been allocated to air crafts and further that all this information is known prior to scheduling.

- We assume that the driving distances between air crafts have already been factored in the turnaround process. That is, either the driving distances between gates is insignificant or that it is already part of the durations of activities in the turnaround process.

- Each air craft manufacturer specifies a minimum and maximum duration for performing each of the ground handling activities. However, sometimes there is a difference in the actual amount of time required for the turnaround process. But because our model requires a single duration for a task, we assume that the time required to perform a ground handling activity is some random value within the bounds specified by the manufacturer.

- We assume that all durations are in integral minutes. Thus, the total duration over a day of operations is equal to 1440 minutes.

- Airports and airline companies allocate tasks to different service providers based on their own internal procurement mechanisms. However, since our focus is not to study the allocation process we simply allocate tasks to a set of agents in our set up based on intuition.

6.3 Modelling the ground handling operations of an airport

In performing our experiments, we choose to focus on a subset of ten ground handling activities — arrival, moving and initial set up, cargo handling and repair, galley servicing, fuelling, boarding, de-boarding, toilet cleaning, providing potable water and finally towing. The choice of these activities is based on the fact that moving and towing of the air craft happen at the very beginning and end of the turnaround process respectively and thus any delays in the turnaround process will necessarily be reflected in these activities. The remaining activities represent the processes that happen during the turnaround process.

Whenever an activity j is to be performed on flight f, we represent it as the task t^j_f. The set of all such tasks t^j_f is represented by T. We allocate the ten activities (including the arrival activity) among six agents $A_1, \ldots, A_6$ based on the type of

Activity	**Encoding**	**Agent responsible**	**Duration range** (in minutes)
Arrival	t_i^1	A_1	-
Moving and initial set up	t_i^2	A_2	1-3
Cargo handling and repair	t_i^3	A_3	1-102
Galley servicing	t_i^4	A_4	5-69
Deboarding	t_i^5	A_6	5-25
Potable water servicing	t_i^6	A_5	2-21
Toilet cleaning	t_i^7	A_5	10-21
Fuelling	t_i^8	A_5	9-70
Boarding	t_i^9	A_6	5-11
Towing	t_i^{10}	A_2	1-6

Table 6.1: Ground handling activities, their encoding as tasks and their allocation to agents.

activity. The encoding of activities and their allocation to agents is as shown in Table 6.1.

Each air craft model typically has a different turnaround process specified. However, based on available data, we were able to classify these processes into four types of turnaround processes. The first turnaround process consists of 5 activities as shown in Figure 6.1. The next turnaround process pertains to air crafts that do not require galley servicing (possibly because they are cargo air crafts). This turnaround process is shown in Figure 6.2(a). In some cases, cargo handling is not required, possibly because the air craft might have only stopped for refuelling or because of bad weather conditions. Such cases result in a third sequence of ground handling activities shown in Figure 6.2(b).

Finally, the last type of turnaround process requires all the 10 activities to be performed. This sequence is shown in Figure 6.3. Boeing provides detailed technical specifications for each of the ground handling activities on its air crafts. Thus, we choose to associate this sequence of activities with all the Boeing air crafts arriving on the day.

In performing our experiments we use the arrival and departure data on a typical

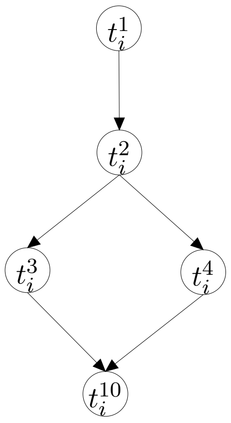


Figure 6.1: Air crafts can have a turnaround process of 5 activities.

day in Schiphol in the year 2005. The data set has close to 50 different types of air crafts each of which has been classified into one of the four types (based on the turnaround processes). If the air craft manufacturer already specifies the turnaround process then we associate it with the given turnaround process. However, in many cases, this information is not available. Therefore, we pick the most similar air craft and use its turnaround process. In case we cannot find any similar air craft, then we simply pick one of the four turnaround processes randomly and associate it with the air craft. The focus of our experiments is the ground handling process and not the arrival of the air craft itself. Further, once the air craft has arrived at the terminal, any of the ground handling activities can start (as long as they do not violate the partial order imposed by the precedence constraints between activities). Therefore, we model the release times for all of the ground handling tasks of an air craft to be equal to its arrival time.

We next model a special agent A_1 with large enough capacity (500 in our case) and allocate all arrival tasks to this agent. A_1 cannot create any delay (because of resource constraints) on its own. In all our experiments, this agent is ignored since it cannot influence the makespan obtained from $\mathrm{M_{ISAn}}$.

Modelling the problem of scheduling ground handling activities as a CAS problem is straight forward. All the ground handling tasks t_f^j to be performed in a single

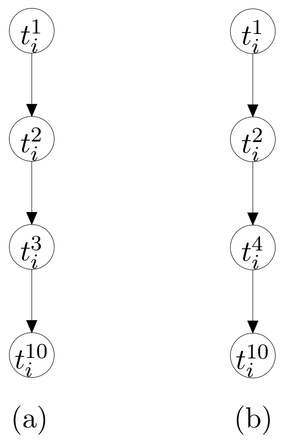


Figure 6.2: Some air crafts do not require galley servicing and some do not require cargo handling.

day can be modelled as elements of the set of tasks T. Each turnaround process imposes partial order constraints on subsets of these tasks thus resulting in a task graph $\mathcal{G} = (T, \prec)$. Each agent A_i is allocated a set of tasks T_i as already described. Each agent can perform a maximum of $c(i)$ tasks simultaneously and each task t_f^j takes a fixed duration $l(t_f^j)$ to be completed. This so called problem of *scheduling ground handling activities* or the SGHA problem therefore can now be described as $\Pi = \langle T, A, \prec, \phi, l(), c(), r(), d() \rangle$ where $r() =$ *arrival time of the air craft* and the deadline $d() = \infty$.

Recall that our intention in this chapter is to evaluate the performance of $\mathrm{M_{ISAn}}$. We aim to perform this evaluation by comparing the results (makespan) obtained from $\mathrm{M_{ISAn}}$ with that of a centralised solution. Thus, in the next section we construct an Integer linear program (ILP) to determine the optimal (centralised) makespan for the SGHA problem.

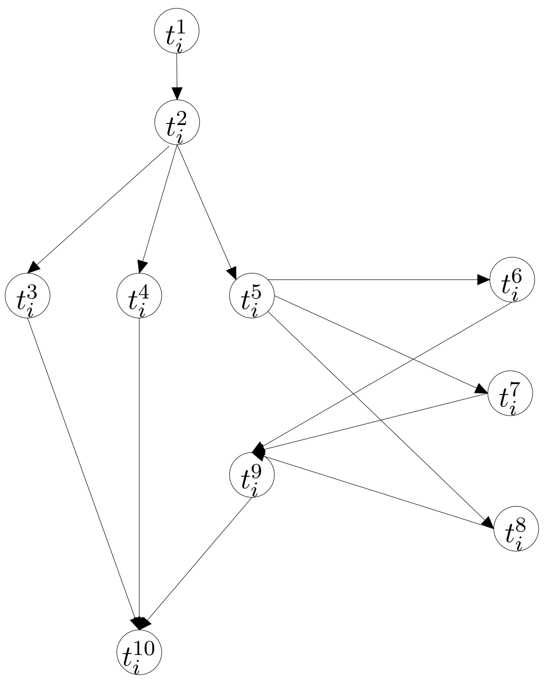


Figure 6.3: Boeing air crafts require all the 10 ground handling activities.

6.4 ILP model of the SGHA problem

The ILP formulation for the SGHA problem is similar to the classical job shop problem with arbitrary precedence constraints and a makespan minimization objective. The ILP model for the SGHA problem is designed to solve the case where agents are sequential. This restriction was necessitated by the limits on our hardware.

In formulating the ILP for the SGHA problem, we use the following indices.

$$\begin{aligned} i &= \text{agent index, } i \in \{1, \dots, N\} \\ j &= \text{task index, } j \in \{1, \dots, M\} \\ k &= \text{time index, } k \in \{1, \dots, 1440\} \end{aligned}$$

For the SGHA problem, we use the following parameters.

$$\begin{aligned} dur_j &= \text{duration for task } j \\ a_{i,j} &= \begin{cases} 1 \text{ if agent } i \text{ has to perform task } j \\ 0 \text{ otherwise} \end{cases} \\ prec_{j,g} &= \begin{cases} 1 \text{ if task } j \text{ precedes task } g \text{ where } g \in \{1, \ldots, M\} \\ 0 \text{ otherwise} \end{cases} \\ c_i &= \text{ concurrency bound of agent } i \end{aligned}$$

The decision variables that we use to determine the solution to the SGHA problem are:

$$\begin{aligned} s_j &= \text{ the starting time of task } j, \\ y_{i,j} &= \begin{cases} 1 \text{ if task } j \text{ can precede task } i \\ 0 \text{ otherwise} \end{cases} \end{aligned}$$

We use C_{max} to denote the overall makespan. We also use a constant, $L = 100000$ to ensure mutual exclusion.

The ILP formulation of the problem can be now stated as:

$$Minimise\ \ C_{max} \tag{6.2}$$

Subject to:

$$(s_j + dur_j)prec_{j,g} \leq s_g \qquad \forall j, g \in \{1 \ldots M\} \tag{6.3}$$
$$Ly_{j,g}a_{i,j} + (s_j - s_g) \geq dur_g \tag{6.4}$$
$$L(1 - y_{j,g})a_{i,j} + (s_g - s_j) \geq dur_j \qquad \forall j, g \in \{1 \ldots M\} \tag{6.5}$$
$$(s_j + dur_j)a_{i,j} \leq C_{max} \qquad \forall i \in \{1 \ldots N\} \text{ and } \forall j \in \{1 \ldots M\} \tag{6.6}$$
$$s_j \geq 0 \qquad \forall j \in \{1 \ldots M\}; \tag{6.7}$$
$$y_{j,g} \in \{0, 1\} \tag{6.8}$$

The ILP model is based on the well known Manne's model for job-shop problems (cf. [Manne 1960]). Constraint set 6.3 ensures that precedence relationships are not violated, while constraint sets 6.4 and 6.5 ensure that tasks are not pre-empted. Finally, constraint set 6.6 ensures that the global makespan is not less than the local makespan of any agent.

The integer programming model of the SGHA problem that was developed in this section is used to generate constraints for each given instance of the SGHA problem. These constraints are then input to a CPLEX solver. We use the optimal makespan computed by the solver for comparison against the longest possible makespan for the same instance obtained through $\mathrm{M_{ISAn}}$. In the next section, we shall deal in detail regarding the experiments we performed to solve the SGHA problem using $\mathrm{M_{ISAn}}$.

6.5 Experimental design and results

In Section 6.1, we listed a set of parameters, that we expect to have a significant impact on the performance of $\mathrm{M_{ISAn}}$. We also explained expected effects of varying these parameters on the performance of $\mathrm{M_{ISAn}}$. In this section we perform experiments to test whether indeed the parameters listed earlier influence the performance of $\mathrm{M_{ISAn}}$ as expected.

In general, one could use the data from the entire day to perform our experiments. However, computing an optimal solution for such a large data set was beyond our computational set up. Therefore, we divide each day into *slots* of 3 hours and compute the minimum makespan for each slot.

6.5.1 Measurements

To infer the effect of each of these parameters, we measure either the performance ratio or the longest possible makespan resulting from $\mathrm{M_{ISAn}}$ in each of our experiments.

Performance ratio In case of sequential agents, it is straight forward to obtain the performance ratio. The performance ratio for sequential agents is simply

$$\frac{\max_{i=1}^{N}(lst(t_i) + l(t_i))}{\text{optimal makespan}} \tag{6.9}$$

For agents with larger concurrency bounds, we can *only measure the longest makespan of* $\mathrm{M}_{\mathrm{ISAn}}$. Therefore, we simply compare the resulting makespan values instead of computing the performance ratio.

Hardware and software set up We use a Linux machine with 3 GB RAM and 2 GHz processor to run $\mathrm{M}_{\mathrm{ISAn}}$. $\mathrm{M}_{\mathrm{ISAn}}$ itself is coded using Java. We also use a Java program to generate all the constraints in the ILP and write it out into a .LP file (dot LP file). This .LP file is later used as input to a CPLEX solver on a machine with 16 GB RAM.

We use Algorithm $\mathrm{M}_{\mathrm{ISAn}}$ to perform all our experiments. We implement the various steps of $\mathrm{M}_{\mathrm{ISAn}}$ in the following way:

- We first create an empty master list of tasks. This list is populated in the following way. Suppose we have a flight f and its moving activity t_f^2 has a duration of 10 minutes. Suppose that this activity is followed by t_f^3. We now split t_f^2 into 10 parts each of unit duration and add into the master list. We ensure that part x of t_f^2 precedes part $x+1$ by adding part $x+1$ to the successor list of part x and part x to the predecessor list of part $x+1$. Finally, we add part 1 of task t_f^3 to the successor list of part 10 of t_f^2 (and part 10 of t_f^2 to the predecessor list of part 1 of t_f^2). The master list is ready when all activities of all flights have been split and the precedence constraints are established.

- Conflict resolution strategy prefers agents according to a random binary variable. That is, whenever a conflict between two tasks t, t' is detected, the implementation adds either $t \prec t'$ or $t' \prec t$ based on whether a random binary variable is 0 or 1.

Data overview It was already mentioned that we divide the arrivals of air crafts into 8 slots. Each such slot has a different number of plane landings and hence a different number of tasks. The number of tasks in each slot is shown in Table 6.2. It is easy to see that different slots have different number of tasks to be scheduled. Slot 4 has the largest number of tasks — 810 — while slot 7 has the fewest number of tasks — 80. In all the slots apart from the 6 and 7, A_5 has the most work to do

Slot	Number of tasks	Number of air crafts
0	215	24
1	125	14
2	720	94
3	580	74
4	810	105
5	715	91
6	605	86
7	80	10

Table 6.2: Number of tasks per slot.

as is evident from Table 6.3. In slot 6, agent A_4 has the most work to perform and in slot 7 A_6 has the maximum amount of work.

Agent	Slot 0	Slot 1	Slot 2	Slot 3
2	70	44	336	270
3	476	522	1647	1449
4	497	396	1612	1391
5	608	604	2082	1593
6	379	209	1040	900
Agent	Slot 4	Slot 5	Slot 6	Slot 7
2	365	318	328	30
3	1550	1249	709	98
4	1682	1456	1009	134
5	1841	1594	824	126
6	1222	1128	802	137

Table 6.3: Work (total time of all tasks) allocated to each agent in each slot.

6.5.2 Experimental set up

We divide the experiments into two scenarios based on whether agents are sequential or not.

Scenario 1 In this scenario, agents are *always sequential.* Using sequential agents we can determine the following aspects of $\mathrm{M}_{\mathrm{ISAn}}$'s performance:

- We can determine the performance ratio of $\mathrm{M}_{\mathrm{ISAn}}$ when confronted with more general task graphs and non-unit task durations.
- We can determine whether the number of tasks has a significant impact on the performance ratio.
- We can also determine the effect of different agent orders on the performance ratio.

The first and second points can be determined simply by running $\mathrm{M}_{\mathrm{ISAn}}$ on different instances of the problem and measuring its performance ratio.

To determine whether different agent orders result in significant improvement/deterioration of the performance ratio, we pick slot 6, and then force $\mathrm{M}_{\mathrm{ISAn}}$ to perform the matching procedure on agents in random order. The choice of slot 6 is based on the fact that slot 6 had the maximum number of tasks among all slots for which we were able to find an optimal makespan. By measuring the performance ratio each time, we can clearly determine whether agent order has a significant role to play.

Scenario 2 Primarily, in this scenario we seek to ascertain the effects of changing (increasing) agent concurrency and hence in all the experiments, we can expect to have at least one agent which has a concurrency bound greater than 1. Further, all experiments in this scenario have a fixed agent order: A_2, A_3, A_4, A_5, A_6. There are two questions with respect to concurrency that we would like to specifically answer:

- will increasing concurrency of all agents help in reducing the makespan?
- how does increasing the concurrency of a subset of agents compare with the scenario when the concurrency of all agents are increased?

To answer the first question we first fix that all agents have concurrency bounds greater than 1 and further that they are all the same. We run $\mathrm{M}_{\mathrm{ISAn}}$ and then measure the longest makespan. We increase the concurrency value gradually, and measure its effects on the makespan. This experiment should tell us two things:

1. whether increasing concurrency in general helps in improving the performance of $\mathrm{M}_{\mathrm{ISAn}}$ and
2. whether there exists some *threshold value* of concurrency such that increasing concurrency beyond this value does not lead to significant improvements in makespan.

To answer the second question we partition agents into 2 sets — the first set comprising 4 agents and the second set comprising a single agent.[1] We then fix the concurrency value of all agents in the first set to some arbitrary non-unit value and perform two experiments — in the first experiment we set the concurrency value of the agent in the second set to be greater than that of the first and then measure the resulting longest makespan and in the second experiment, we do the reverse. That is, set the concurrency value of the agent in the second set to be lesser than that of the agents in the first set and again measure the longest makespan.

We then repeat this procedure with different agents in the second set. Based on the makespan values we can easily see whether increasing or decreasing the concurrency value of each of the agents has the same effect.

6.5.3 Results of experiments in Scenario 1

The running time required for the solver (for solving the ILP formulation) and the heuristic are given in Table 6.4. For slots 2, 4 and 5, the CPLEX solver went out of memory and hence we do not consider them further. The implementation allows $\mathrm{M}_{\mathrm{ISAn}}$ to make a random choice regarding the direction of the precedence relation between conflicting tasks. Therefore, to account for this randomness, we ran 150 experiments for each time slot with the fixed agent order $A_1, A_2, A_3, A_4, A_5, A_6$ to obtain the longest makespan values from $\mathrm{M}_{\mathrm{ISAn}}$. The performance ratio for each of the slots is shown in Table 6.5.

[1]Recall that A_1 is ignored from our experiments.

Slot	Optimal	M_{ISAn}
0	13024	85
1	8124	70
2	-	213
3	65743	125
4	-	325
5	-	320
6	56765	200
7	1	2

Table 6.4: Running times required (in seconds).

	Optimal	**Average**	**Performance ratio**
Slot 0	890	1367.1	1.54
Slot 1	846	1569.83	1.87
Slot 3	2266	3877.40	1.71
Slot 6	2097	3186.3	1.52
Slot 7	1468	1734.18	1.18

Table 6.5: Performance ratio for sequential agents.

Next we experimented with random orderings of agents and recorded the longest makespan for slot 6 in Table 6.6. For each agent order we again performed 50 experiments and recorded the longest makespan. Slot 6 was chosen since it had the maximum number of tasks (i.e., 605. See Table 6.2) among slots where we were able to obtain the optimal makespan.

6.5.4 Analysis of results in Scenario 1

The table of run times is along expected lines as the problem is known to be intractable in general. It clearly shows that trying to obtain the optimal solution for larger instances of the problem may not be a practical option.

It is easy to see from Table 6.5 that the performance ratio is almost always

Agent order	Makespan	Agent order	Makespan	Agent order	Makespan
5,4,2,6,3	3156	3,2,5,4,6	3217	2,6,4,3,5	3402
6,4,3,5,2	3544	4,5,6,2,3	3160	6,3,2,5,4	3816
4,5,3,6,2	3157	5,3,6,4,2	3151	5,2,6,3,4	3160
6,4,2,5,3	3453	5,2,4,6,3	3131	6,5,2,3,4	3465
3,5,2,6,4	3151	4,2,6,5,3	3314	3,4,5,6,2	3154
4,2,5,3,6	3169	4,3,6,5,2	3630	3,5,2,4,6	3194
2,4,5,6,3	3192	5,4,3,6,2	3157	6,5,2,4,3	3483
2,6,3,5,4	3529	4,6,2,3,5	3445	6,5,3,4,2	3573
5,2,4,3,6	3144	3,2,4,5,6	3313	3,6,2,5,4	3413
5,2,3,6,4	3166	6,2,5,3,4	3501	5,4,3,2,6	3167

Table 6.6: Maximum value of the longest makespan (from 50 experiments) for slot 6 for different agent orders.

greater than 1.5. In fact, the only slot where it is less than 1.5 is the last slot — slot 7. This could be attributed to the fact that the number of plane landings in this slot (10) is very low in comparison with all other slots. The above observation might indicate that the number of tasks in a slot heavily influences M_{ISAn}. However, it is not always true. Slot 1 has the second least number of tasks (i.e., 125. See Table 6.2) but it results in the worst performance ratio (i.e., the ratio is over 1.8. See Table 6.5). Moreover, slot 6 has the largest number of tasks to perform (605) among all slots. However, M_{ISAn} is able to perform very well (i.e., 1.54, second best among all slots) on slots 6. Therefore, we wonder if the number of tasks does not influence M_{ISAn}, then what does influence it?

To understand this phenomenon, let us present a simple example. Consider the situation in Figure 6.4(a). Here there are two chains of three tasks each and each task takes unit duration for processing. We know that if a preference is made between t_1 and t_4, then the remaining tasks of the chains that follow them do not have any conflicts. On the other hand, if the chain lengths are dissimilar, as in the case of Figure 6.4(b), then choosing the longest chain ($t_1 \prec t_4 \prec t_6$) to be executed first and the chain $t_2 \prec t_6$ next and finally the task t_3 would result in a makespan of 3. If our choices were flipped, that is $t_3 \prec t_t \prec t_1$ then the makespan would be 5 (close

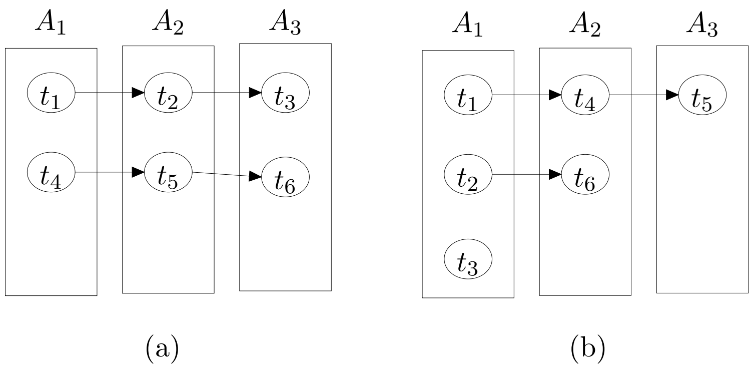


Figure 6.4: Lengths of partial orders affects approximation ratio.

to twice the minimum possible). Therefore, if the chains have equal lengths then we can expect M_{ISAn} to result in schedules of lower makespan (because it resolves conflicts arbitrarily). More generally if chain lengths are similar, then we can expect that M_{ISAn} will result in an performance ratio closer to 1.

Now we shall establish that the chains in slot 6 have a similar length. Suppose that the chain lengths are different, then there could be only two reasons — they could have widely differing turnaround durations or planes in the slot could be arriving with a large inter-arrival gap. Let us now examine if either of these two things occur in slot 6. Let us first start with turnaround durations.

A simple check of the standard deviation of servicing times of air crafts in each slot gives us an insight into the service times required for air crafts in each slot. Figure 6.6 plots the same. Notice that the standard deviation in the service times of air crafts in slot 6 is lesser than that of slot 1. Implying that the chains are of similar lengths in slot 6.

Now suppose the tasks of agent A_1 in Figure 6.4(a) are all available at different times. This in effect would imply that the situation is similar to the Figure 6.4(b), since the difference in times at which tasks are available has to be also factored in. The number of times several air crafts land simultaneously in slot 6 is 41 where as in slot 1 it is 0. This fact further strengthens the premise that large portions of the task graph are similar in slot 6 in comparison with slot 1 (a partial snapshot of the

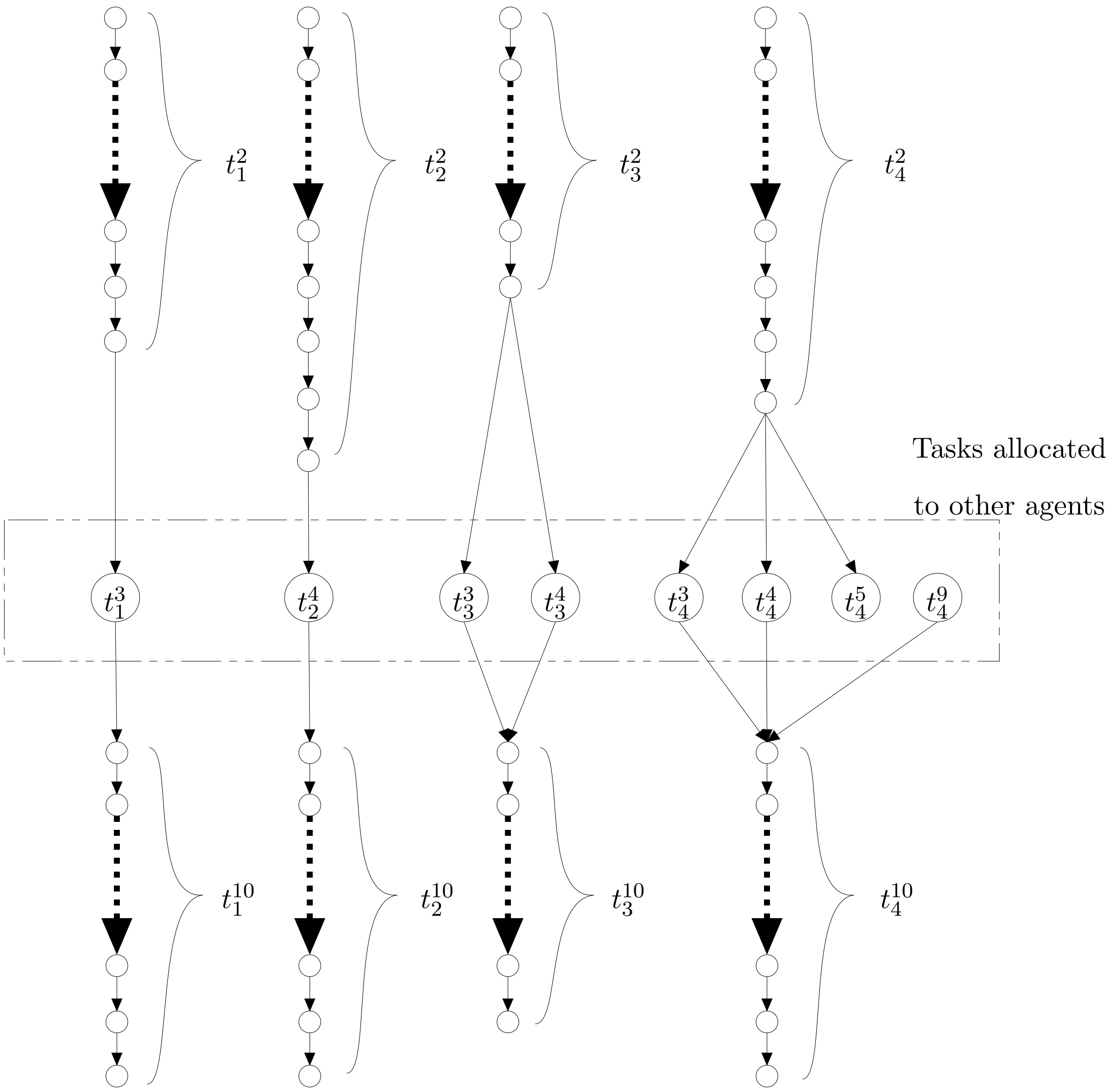


Figure 6.5: A partial snapshot of the task graph in slot 6 for agent A_2.

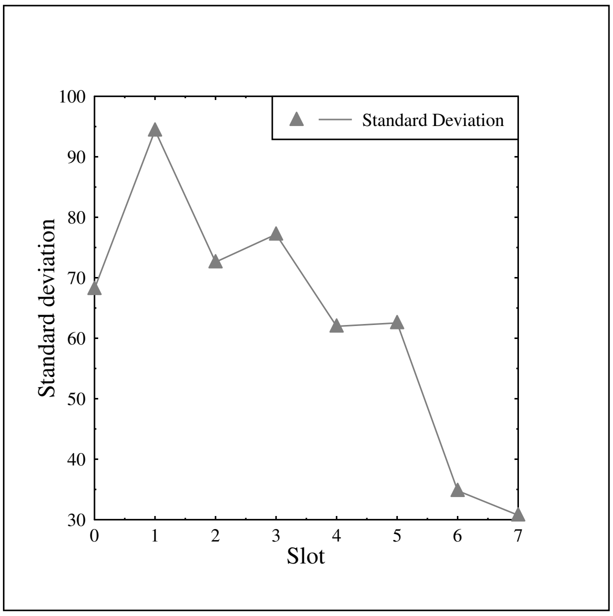


Figure 6.6: Standard deviation in service times of air crafts.

tasks in slot 6 is given in Figure 6.5).

Thus we have established that the time required to process air crafts in slot 6 are similar. This may explain the better performance ratios in slot 6 than in slot 1.

Summarising the above arguments we can conclude that the difference in partial order lengths has a significant effect on the performance of M_{ISAn}.

Let us next analyse the results of different agent orderings. Not surprisingly, the difference between the best value and the worst value is quite high as seen in Table 6.6. The worst makespan is 21.9% greater than the minimum makespan. Therefore confirming our expectation that agent order has a significant impact on the performance ratio.

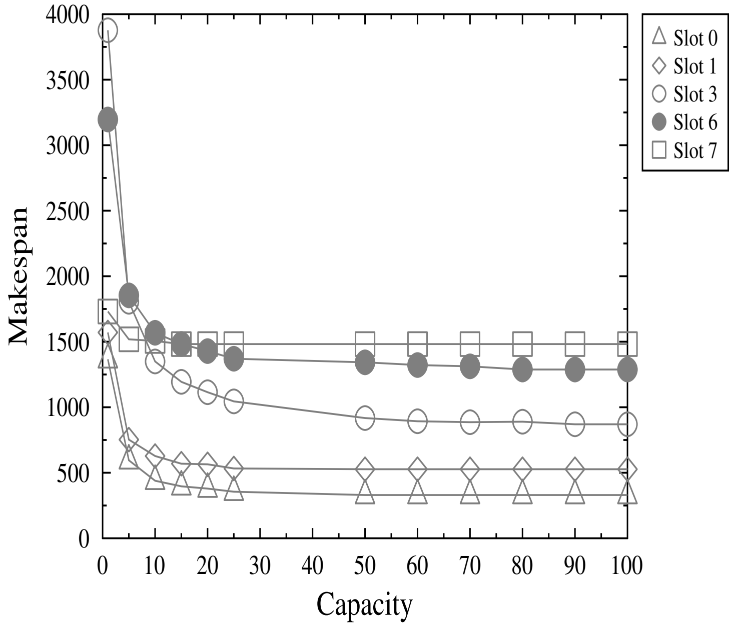


Figure 6.7: Performance of $\mathrm{M}_{\mathrm{ISAn}}$ for different capacities of agents. Each data point is the average longest makespan for a given capacity and slot computed over 50 experiments.

6.5.5 Results of experiments in Scenario 2

The result of the experiments recorded in the graph of Figure 6.7 pertain to the first set of experiments under Scenario 2 where all agents had the same value of concurrency. Concurrency bounds were increased from 1 to a maximum of 100 and at each concurrency value we ran the algorithm 50 times to obtain the results in Figure 6.7. We next experimented by allocating different capacities to agents. As explained earlier we split agents into two groups — one containing 4 agents and the second containing the remaining agent. We fixed that agents in the first group would have a concurrency value of 20 and in each experiment we evaluated the overall makespan in two scenarios — one when the single agent in the second group had a capacity of 30 and the second when the single agent had a capacity of 10. We performed experiments by altering the concurrency of each agent for slots 0, 1, 3, 6 and 7 (50 runs for each agent and for each slot). The results are shown in Figure 6.8.

6.5.6 Analysis of results in Scenario 2

In Scenario 2 we again divided the experiments into two sets — first where all agents have same concurrency and the second where agents have different amounts of concurrency.

The results from the graph of Figure 6.7 indicate that makespan values recorded a decrease when agent concurrency was increased. The graph in Figure 6.7 also shows that the makespan values *stabilise* as the concurrency of agents approaches some threshold value (≈ 50).

The idea that makespan values do not improve after the concurrency values hit a threshold value is quite easy to understand. A simple explanation for this phenomenon would be that for a given task graph if the maximum number of conflicting tasks for an agent is less than the concurrency bound, then $\mathrm{M}_{\mathrm{ISAn}}$ would essentially function as ISA and would not require any conflict resolution and hence result in minimal makespan.

For the second set of experiments, we observed that makespan values definitely decrease with an increase in concurrency. Increasing concurrency of agents with most work to do, results in greater reduction of makespan than increasing concurrency of agents with the least amount of work. For instance agent A_5 and agent A_3 make the biggest difference to the makespan values in almost all the slots. This is hardly surprising since these agents have the maximum amount of work to do in all these slots. Similarly, agent A_4 records a bigger improvement in slot 6 since it has the highest amount of work in to do in slot 6, i.e., 1682 tasks (refer Table 6.3) compared to all the other slots. Note here that while increasing the capacity of these two agents results in the biggest differences, they need not result in the best makespan. A case point being slot 3 where increasing the concurrency of agent A_6 to 30 results

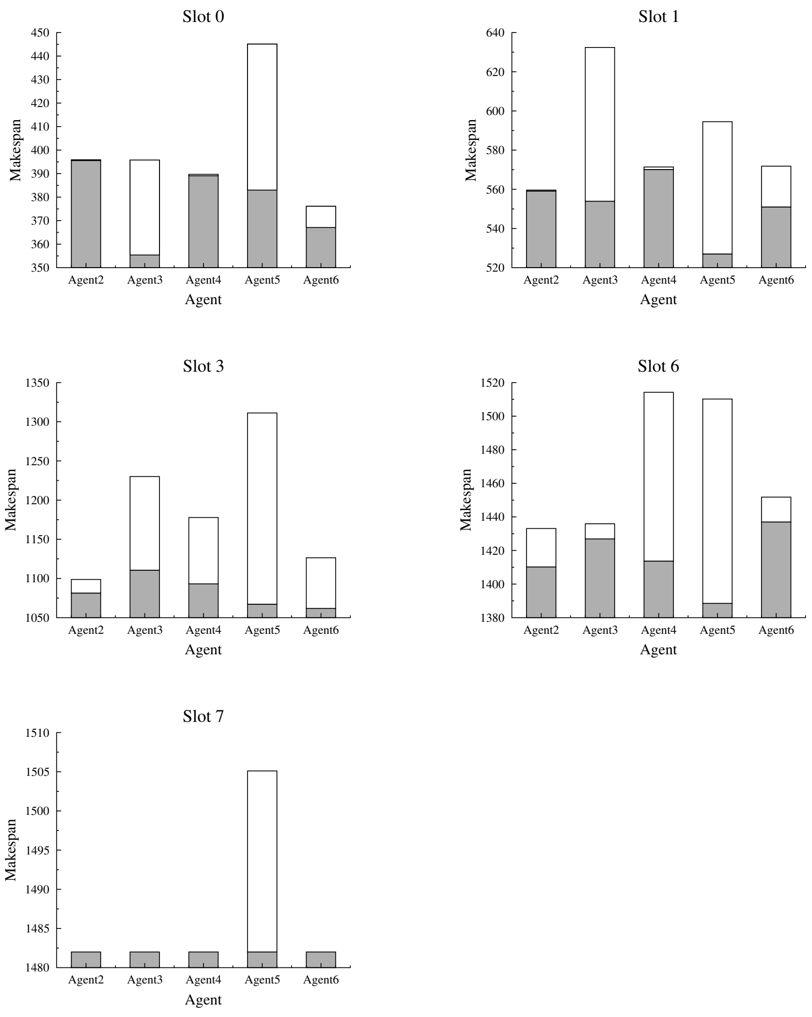


Figure 6.8: Performance of M_{ISAn} for different capacities of agents. The filled area indicates the average makespan when the capacity for the agent was increased to 30 and the size of the bar indicates the average makespan when the capacity of the agent was reduced to 10.

in better makespan than increasing the concurrency of either agent A_5 or agent A_3. A possible explanation for this could be that the agent order has a cascading effect with respect to the conflicts and agent A_6 bears the brunt of it in slot 3. Thus increasing its capacity relieved agent A_6 from a large number of conflicts and hence resulted in a better makespan.

In addition to answering our expectations, the analysis of results in Scenario 2 can be fruitfully employed to determine optimal capacity levels for agents. For instance, based on Figure 6.8, agents A_3 and A_5 could be allotted more resources in all slots apart from slot 3 where agent A_6 could be allocated more. Similarly, we also know that increasing the capacity beyond 50 may not result in considerable gain. The speed of the algorithm allows several experiments to be performed in reasonable amounts of time and hence allow for *fine tuning* the capacity levels required for agents over different time slots.

6.6 Summary

In this chapter we set out to evaluate the performance of M_{ISAn} in a general setting where task graphs could be non-chain structured and agents could have heterogeneous capacities. We found that several parameters affect the performance of M_{ISAn}. We observed through our experiments in Scenario 1 that:

- Although the number of tasks has an impact on the performance ratio, the difference in partial order lengths in a given instance has a greater impact on the performance of M_{ISAn}. We showed that when the differences between partial order lengths was smaller, the approximation ratio was better.

- Agent order affects the makespan resulting from M_{ISAn} significantly.

We next experimented with agents having concurrency bounds greater than 1 to ascertain the impact of concurrency on the makespan derived from M_{ISAn}. The results of our experiments can be summarised as follows:

- Increasing concurrency of agents in general helps in reducing makespan.

- While increasing the concurrency of the agent with most work to do helps, the results might be better if the concurrency values are directly proportional to the amount of work.

Apart from offering a useful solution directly, $\mathrm{M}_{\mathrm{ISAn}}$ could be put to use in other ways too:

- $\mathrm{M}_{\mathrm{ISAn}}$ could be used to generate useful *cuts* that a solver such as CPLEX could use to further improve upon its solution. Cuts are nothing but a set of additional constraints that can be added to an Integer program (or an MIP), so that the size of the feasible region is reduced without losing any potential integer solution. In our case, the longest makespan obtained from $\mathrm{M}_{\mathrm{ISAn}}$ can be used as a cut. That is, we could add the following constraint to the ILP model in Section 6.4:

  $$C_{max} < \text{Longest makespan for the instance derived from } \mathrm{M}_{\mathrm{ISAn}}$$

  Clearly, this results in a reduction in the number of feasible solutions for the problem and hopefully allows a solver to discover the optimal solution faster.

- $\mathrm{M}_{\mathrm{ISAn}}$ can be used to evaluate resource allocation strategies that result in better makespan values. This can be done by a method similar to our approach in the second set of experiments. That is, one could do the following:

  ▷ Increment the concurrency bound for all agents and then run $\mathrm{M}_{\mathrm{ISAn}}$ until the makespan value does not show any remarkable improvement (or until the global makespan is less than or equal to a desired value).

  ▷ Compare the global makespan resulting from the chosen resource allocation strategy with the makespan from the previous step. If the difference is small then we can claim that the resource allocation strategy is a *good* strategy.

Having evaluated the performance of $\mathrm{M}_{\mathrm{ISAn}}$ in a practical setting, we notice that there are several possible improvements to our work. Further, there are many ways our work can be extended to derive further insights into this problem of coordinating the plans and schedules of autonomous agents. Thus, in the next chapter we summarise our findings and point out future directions for research.

Chapter 7

Conclusions and future work

In this thesis, we have seen so far that when communication between agents who are trying to plan for a common goal is untenable, then ensuring coordination between their plans and schedules is an intractable problem in general. Fortunately, not all plan and schedule coordination problems are equally hard.

With the above background, we formulated a series of questions in Chapter 2. These questions pertained to *(i)* finding classes of plan coordination problem instances for which effective coordination protocols could be designed and *(ii)* designing a decoupling mechanism for schedule coordination problems.

In this final chapter, we first summarise and evaluate whether the research questions we posed in Chapter 2 have been answered. In Section 7.2, we first point out some immediate extensions to our current work. We then also briefly discuss a couple of other areas which require a more extensive effort.

7.1 Summary

It is interesting to note that in most cases of plan coordination we face in day-to-day lives, coordination is achieved through communication and cooperation. Researchers have already proved that without communication and cooperation, plan coordination problems fall in the Σ_2^p-complete class in general. However, we often find that despite lack of communication and cooperation, enterprises do manage to coordinate joint activities. This may suggest that there exist classes of problems which are somehow easier to solve than the general class of problems. Our attempt was to identify the characteristics of such classes of problems and exploit them. Therefore, we focussed on finding subclasses based on the topological structure of the task graph.

7.1.1 Plan coordination

Recall from Chapter 2 that we had asked the following questions regarding plan coordination problems:

1. Does there exist an easily identifiable class of plan coordination problems instances where CVP can be efficiently performed?
2. Can we design an approximation algorithm for CP, which exploits the fact that CVP is polynomially solvable?
3. Does there exist a class of coordination problem instances such that even CP can be solved polynomially for that class?

Suppose the task graph structure of the plan coordination instance is such that there are no precedence constraints between tasks allocated to the same agent. This property, known as the *intra-freeness property*, is the focus of Chapter 3. We found that given an intra-free instance, we could represent it as a much smaller agent dependency graph. Using these agent dependency graphs we showed that if problem instances satisfied the intra-freeness property then CVP could be efficiently solved, thus answering the first question we posed positively.

Later, we also designed an algorithm, the DP^* algorithm, which improves upon the state of the art algorithm for plan coordination. The DP^* algorithm relies on finding a blackout feedback vertex set which can be used to filter the set of tasks on which coordination constraints must be applied. The DP^* algorithm uses the fact that the underlying task graph is intra-free and hence the verification problem can be efficiently solved. Thus, we managed to find a positive answer to the second question that was posed as well.

Finally, to answer the third question, we dug deep to find a special subclass of intra-free instances called the SLIF instances. We showed that there was an efficient algorithm to solve the plan coordination problem. However, SLIF instances constitute a *severely restricted* class of intra-free instances. It would not be wrong to say that we have just managed to show that such classes do exist. Thus, we consider that we have managed to only *partially* find a solution to the third question.

The plan coordination problem inherently is a very hard problem and hence even slight generalisations of polynomially solvable subclasses can be intractable. However, looking for other topological structures that might allow for easier solving of coordination problems is still a worthy effort.

7.1.2 Schedule coordination

In practical applications of plan coordination, it is often not enough to claim that a task t would be done *after* task t'. Decision makers require more concrete information regarding the time by which a task is completed. This requirement motivated us to look into developing coordinated schedules instead of coordinating plans. In Chapter 2 we stated the following research questions regarding schedule coordination:

Can a decoupling technique be developed such that it

1. ensures that every local schedule for the decomposed problem can be merged into a feasible global solution,
2. ensures concurrency constraints are not violated and
3. also minimises the global makespan?

We answered the first question positively. The idea behind the solution is to first notice that the task graph is now slightly different from the one in plan coordination. The task graph for schedule coordination also has *lengths* corresponding to the duration of tasks. This graph of precedence constraints between tasks and duration information can be represented as a simple temporal network (STN). It has already been shown by Hunsberger (see [Hunsberger 2002a]) that any STN can be temporally decomposed in an efficient way. Therefore, we started with a basic algorithm where we transformed any given instance of the CAS problem into an instance of the temporal decoupling problem and applied Hunsberger's algorithm to obtain a set of decomposed scheduling problems which can each be autonomously solved.

The general *STN* specification allows for the specification of both a minimum and maximum interval between temporal points, but CAS problem instances only specify the duration of task. Due to this property, we showed in Chapter 4 that a faster, simpler and more efficient method can be derived from Hunsberger's approach to solve CAS problem instances. This modified algorithm called ISA, associates with each task a starting time interval derived from the *depth* and *height* of the task in the task graph. The depth and height of each node in a directed acyclic graph can be computed in time linear in the number of nodes. This allows ISA to be more efficient when compared to Hunsberger's algorithm. Furthermore, ISA guarantees that the global makespan is minimum as well as that each agent has maximal autonomy.

The second and third questions are more tricky. ISA cannot accommodate constraints on agent capacity. Chapter 5 deals with this issue. Unfortunately, once we add capacity constraints, CAS problem becomes NP-complete. One of the difficulties with this bounded capacity version of the CAS problem (bCAS problem) is that it can be shown that it is already NP-hard to detect all possible conflicts between tasks. Therefore, we extended ISA into $\mathrm{M}_{\mathrm{ISAsq}}$ so that we could handle agents with unit capacity and unit duration tasks. The idea behind this extension was to use the initial intervals suggested by ISA to construct a bipartite graph where conflicts between intervals could be efficiently detected. Unfortunately, despite the efficient conflict detection mechanism, the overall aim of achieving makespan minimal global schedules still remains NP-complete. Even when we restrict problem instances to very special structures such as *grids*, we have shown that $\mathrm{M}_{\mathrm{ISAsq}}$ like algorithms can yield very bad global makespan. We also showed that with the help of additional heuristics such as the source first heuristic, there is hope of better performance by $\mathrm{M}_{\mathrm{ISAsq}}$ like algorithms. Therefore, we were only able to *partially answer* the second and third questions.

Although the worst case performance bounds of algorithms such as $\mathrm{M}_{\mathrm{ISAsq}}$ are theoretically bad, we hoped that in a general scenario, the results would be better. Therefore, we experimentally tested the performance of $\mathrm{M}_{\mathrm{ISAn}}$ in Chapter 6. In this chapter we applied $\mathrm{M}_{\mathrm{ISAn}}$ to coordinate the schedules of ground handling operators in an airport. In a series of experiments conducted using this scenario, we found that as long as the lengths of the various partial orders in a given task graph were similar, the performance of $\mathrm{M}_{\mathrm{ISAn}}$ was reasonable. However, the real benefit of applying $\mathrm{M}_{\mathrm{ISAn}}$ for coordinating schedules in a practical setting would probably be more to provide a *first cut solution* and probably also understand the impact of adding/removing capacity.

Again, it is to be noted that CAS problems inherently are hard problems. Therefore, developing more fine tuned heuristics to achieve decoupling seems to be the most profitable solution strategy for these problems.

While it is evident that coordination problems in general are intractable, the curious researcher can explore several possible directions. Thus, in the next section we first point out immediate extensions to our work in plan and schedule coordination. We then point out a couple of other areas where a more comprehensive strategy has to be employed to even describe the problem.

7.2 Future work

Plan and schedule coordination problems occur in various contexts and settings. There could be settings similar to ours, and there could be settings where there are other kinds of restrictions such as unreliable agents, incomplete tasks etc. In our thesis, we have mostly limited ourselves to problems that could be described by Valk's framework for coordination problems (with slight extensions of course!). However, we believe that in each setting, coordination problems come with their own set of unique challenges that are very interesting for further research. This section is devoted to looking at a small set of such challenges.

Even within the immediate reach of this framework and our current set of algorithms, are several problems that are not only interesting but also practically relevant. Therefore, we next describe such a set of immediate extensions to our work. We then point out some other settings where Valk's framework might be inadequate to describe the problem.

7.2.1 Immediate extensions

There are several other extensions to our version of the coordination problem. Mainly, we could look at these extensions in two classes — improving existing solution techniques and extending problem settings. In what follows, we describe such immediate improvements and extensions for plan and schedule coordination problems.

Plan coordination: While it is true that the class of SLIF instances can be efficiently solved, it is also true that this class is *severely restricted.* A single fork or a join in the task graph could possibly render our algorithm completely ineffectual. Therefore, an immediate extension could be to find an approximate method, that solves plan coordination problem instances where task graphs have a given maximum degree of nodes.

As far as extending the existing problem setting is concerned, one could consider the case where tasks share *disjunctive constraints* between them. Pijper (see [Pijper 2010]) already has proved that finding a minimum coordination set for problem instances with disjunctive constraints is also Σ_2^p -complete. However, we still do not even have approximate methods to find coordination sets in such cases.

Another set up would be to associate *cost factors* into the mix. If each constraint *costs* some value, then finding a minimum cost coordination set could be of greater practical value than finding a minimum cardinality coordination set. We suspect that this problem is also Σ_2^p-hard, since the coordination verification problem has to be still solved. The only difference is that instead of finding a minimum cardinality coordination set we now need to find a coordination set whose sum of costs is minimum. A simple test that strengthens this view of the complexity is to associate unit cost to each constraint. Now it is easy to see that minimum cardinality coordination set is the same as a minimum cost coordination set.

Schedule coordination: An immediate improvement to the existing algorithms for solving CAS problem instances, would be to develop a more sophisticated conflict resolution strategy. Clearly, heuristics such as the source first heuristic, emphasise the fact that task graph structure can be further exploited to gain better performance. However, the real challenge would be to develop a heuristic that can be used to generate closer to optimal solutions for general classes of CAS problem instances. A more sophisticated conflict-resolution heuristic could possibly allow us to deal with the second and third questions effectively.

There are other immediate extensions to our set up that could be considered for further research. In our set up we have considered that capacities of agents are independent, that is, agents do not *share resources.* However in several cases this is not true. It would be interesting to consider coordination problems when resources are shared. While there has been some effort in the multi-agent route planning community, regarding such problems (see [terMors 2010]), there has not been much effort in ensuring that agents can simultaneously plan.

If resources required to perform a task are not exclusive to an agent then, one could employ some market mechanism to determine the usage of the resource. Market mechanisms such as auctions and negotiations also allow users to pursue individual goals which could be different from the global goal. Auction based methods for distributed scheduling might also come in handy. The paper by Wellmann et al. (see [Wellman *et al.* 2001]) describes auction protocols that can be employed to schedule resources in a distributed scheduling system.

Negotiation is another powerful idea that can be applied to derive coordinated schedules. The paper by Kaplansky and Meisels (see [Kaplansky and Meisels 2007]) uses negotiation as a basis to derive a global schedule. In their system, individual scheduling agents with their own goals need to find a coordinated global schedule. Their set up guarantees that the negotiations always end in an agreement but does not ensure the optimality of the solution derived from such negotiations. It would be interesting to study the gain in optimality if some scheduling agents had more negotiating power than the others.

In our set up we have also not considered situations where tasks could share *simultaneity constraints* between them. That is, suppose a heavy object has to be lifted by a set of agents, this task requires that all of them lift the object at the same time. If such constraints are brought into the mix, it is still an open question to determine whether we can still decompose the problem so that agents can schedule simultaneously. Just as in plan coordination, we have also not considered *disjunctive constraints* for schedule coordination problems. Similarly *cost based* coordination was also not considered. In fact, Wu et al. (see [Wu *et al.* 2009b]) consider a variant of our problem where notions of cost are introduced. They use a negotiation protocol to arrive at a final set of constraints that guarantee the best mix of profit as well as autonomy.

7.2.2 Other related areas

In the previous section we saw that there are several immediate extensions to our work on plan and schedule coordination. In this section, we shall examine a couple of areas — on-line coordination and task allocation, that have a very close connection to plan and schedule coordination problems.

On-line coordination Plans and schedules are usually made so that they can be reused. However, this is true only if the problem instance is static. It is a whole new challenge if one has to coordinate situations where tasks arrive one by one. In fact in such scenarios, the idea of coordination itself might need to change. It can no longer be the avoidance of cycles in the task graph since, many tasks that can be part of a cycle might have already been performed and are no longer relevant to the planner. Tasks also cannot be considered uniformly since each task has a different life time. Therefore, temporal effects have to be considered even during plan coordination.

There are umpteen practical problems where on-line scenarios are common. For example in a hospital, patients might arrive asynchronously over time, coordinating the plans and schedules of different departments which have to offer treatments/diagnosis to these patients is very crucial. Similarly in an airport scenario, all planes do not arrive at the same time, they require that ground handling operations must be coordinated to ensure that there are no delays.

GPGP (see [Decker and Li 1998]) has been applied in developing plans for patient treatment in hospitals. In GPGP agents need to make commitments on their activities and hence are not robust in the face of small and frequent changes. Decomposition based methods can offer agents greater degree of flexibility as well as robustness in the face of minor changes. So far, to the best of our knowledge no efforts have been made to coordinate plans or schedules in on-line scenarios using decomposition based techniques. The inherent difficulty being the changing nature of the problem instance.

One possible direction to pursue would be to look at stochastic on-line scheduling. This area has only recently gained recognition. Chou et al. (see [Chou *et al.* 2006]) have proved the asymptotic optimality of the on-line WSEPT (weighted shortest expected processing time) rule for single machine scheduling problem $1|r_j|\Sigma w_j C_j$ assuming that weights and processing times can be bounded both ways (from above and below) by constants. Megow et al. (see [Megow *et al.* 2006]) consider the machine scheduling problems $1||E[\Sigma w_j C_j]$ and $1|r_j|E[\Sigma w_j C_j]$. Here the objective represents the expected weighted completion time. They prove that a simple on-line scheduling policy can provide a performance guarantee that matches the currently best known performance guarantee for stochastic parallel machine scheduling. The common denominator in all these settings is the fact that the settings are all centralised. While this area seems to be receiving attention, the same has not been the case for distributed versions of the problem. Therefore, it might be worthwhile to pursue research in this direction.

Task Allocation Allocation of tasks plays a big role in coordination. Suppose each agent is allocated with a single task, then it is easy to see that there can be no plan deadlocks. Similarly, if task allocation was such that each agent was intra-free, then coordination verification becomes polynomially solvable. In our thesis we have so far assumed that task allocation is given to us by the problem owner. However, if task allocation is not given, then the problem of allocating tasks such that minimum number of coordination constraints are required has been shown to be Π_3^p-complete by Buzing et al. (see [Buzing *et al.* 2006]).

On the other hand, the complexity of the problem where we are given an initial allocation and allowed to make a given finite number of changes to the allocation, so as to achieve minimum cardinality coordination is still open. It is suspected that this problem is at least Σ_2^p hard since it always requires to compute a minimum cardinality coordination set. Approximation algorithms that solve this problem are also not known.

Hierarchical planning technique has been used for task allocation by a few researchers (see [Freeman 2003]). Here a top level agent breaks up its overall goal and distributes tasks among its subordinate agents, who in turn breakup their tasks to their subordinates and so on. However, the difficulty seems to be that this does not ensure makespan optimality of the derived schedules. Besides the difficulty of ensuring makespan optimality, this sort of task allocation requires that several agents are able to perform task allocation and there is some sort of a command hierarchy amongst the agents.

Another interesting idea would be to associate profits to tasks and costs to coordination constraints, and determine if there exists some allocation of tasks such that a minimum cost coordination set can coordinate plans. Market mechanisms which allow agents to negotiate about these two parameters might have interesting consequences to the general area of plan coordination. One could also take this further and evaluate the trade-off between schedules thus derived and the profits made.

Bibliography

[terMors 2010] A. W. terMors. *The world according to MARP.* PhD thesis, TUDelft, 2010.

[Agarwal *et al.* 1995] R. Agarwal, P. De and C.E. Wells. *Cooperative distributed problem solving: an investigation in the domain of jobshop scheduling.* Hawaii International Conference on System Sciences, volume 3, page 4, 1995.

[Alami *et al.* 1994] R. Alami, F. Robert, F. Ingrand and S. Suzuki. *A paradigm for plan-merging and its use for multi-robot cooperation.* In IEEE International Conference on Systems, Man, and Cybernetics, 1994, volume 1, pages 612–617, 1994.

[Alami *et al.* 1998] R. Alami, F. F. Ingrand and S. Qutub. *A Scheme for Coordinating Multi-robots Planning Activities and Plans Execution.* In European Conference on Artificial Intelligence, pages 617–621, 1998.

[Alami *et al.* 2002] R. Alami, F. Robert, F. Ingrand and S. Suzuki. *Multi-robot cooperation through incremental plan-merging.* In Proceedings of The IEEE International Conference on Robotics and Automation, 1995, volume 3, pages 2573–2579, 2002.

[Aldunate *et al.* 2006] R. Aldunate, S. F. Ochoa, F. Pe na Mora and M. Nussbaum. *Robust Mobile Ad Hoc Space for Collaboration to Support Disaster Relief Efforts Involving Critical Physical Infrastructure.* Journal of Computing in Civil Engineering, volume 20, no. 1, pages 13–27, 2006.

[Arangú *et al.* 2008] M. Arangú, A. Garrido and E. Onaindia. *A general technique for plan repair.* In 20th IEEE International Conference on Tools with Artificial Intelligence, 2008, volume 1, pages 515–518, 2008.

[Argote 1982] L. Argote. *Input uncertainty and organizational coordination in hospital emergency units.* Administrative Science Quarterly, pages 420–434, 1982.

[Ashford *et al.* 1997] N. Ashford, H. P. M.Stanton and C. Moore. Airport operations. McGraw-Hill, 2 édition, 1997.

[Azarewicz *et al.* 1989] J. Azarewicz, G. Fala and C. Heithecker. *Template-based multi-agent plan recognition for tactical situation assessment.* In Proceedings of the Fifth Conference on Artificial Intelligence Applications, 1989, pages 247–254, 1989.

[Babayan and He 2004] A. Babayan and D. He. *Solving the n-job 3-stage flexible flowshop scheduling problem using an agent-based approach.* International journal of production research, volume 42, no. 4, pages 777–799, 2004.

[Bacon 2011] Derek Bacon. *Oil and the economy The 2011 oil shock More of a threat to the world economy than investors seem to think.* The Economist, March 2011.

[Baptiste *et al.* 2004] P. Baptiste, S. Knust and V. Timkovsky. *Ten notes on equal-execution-time scheduling.* OR: Quarterly Journal of the Belgian, French and Italian Operations Research Societies, volume 2, pages 111–127, 2004.

[Baptiste 1999] P. Baptiste. *Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times.* Journal of Scheduling, volume 2, no. 6, pages 245–252, 1999.

[Burke and Prosser 1991] P. Burke and P. Prosser. *A distributed asynchronous system for predictive and reactive scheduling.* Artificial Intelligence in Engineering, volume 6, no. 3, pages 106 – 124, 1991.

[Buzing *et al.* 2006] P. C. Buzing, A. W. terMors, J. M. Valk and C. Witteveen. *Coordinating Self-Interested Planning Agents.* Autonomous Agents and Multi-Agent Systems, volume 12, no. 2, pages 199–218, March 2006.

[Caridi and Sianesi 2000] M. Caridi and A. Sianesi. *Multi-agent systems in production planning and control: An application to the scheduling of mixed-model assembly lines.* International Journal of Production Economics, volume 68, no. 1, pages 29 – 42, 2000.

[Chou *et al.* 2006] M. C. Chou, H. Liu, M. Queyranne and D. Simchi-Levi. *On the asymptotic optimality of a simple on-line algorithm for the stochastic single-machine weighted completion time problem and its extensions.* Operations Research, pages 464–474, 2006.

[Christofides 1975] N. Christofides. Graph theory: An algorithmic approach, volume 8. 1975.

[Cicirello and Smith 2004] V. A. Cicirello and S. F. Smith. *Wasp-like agents for distributed factory coordination.* Autonomous Agents and Multi-agent systems, volume 8, no. 3, pages 237–266, 2004.

[Cormen *et al.* 1990] T. T. Cormen, C. E. Leiserson and R. L. Rivest. Introduction to algorithms. 1990.

[Cox and Durfee 2003] J. S. Cox and E. H. Durfee. *Discovering and exploiting synergy between hierarchical planning agents.* In Proceedings of the second international joint conference on Autonomous agents and multiagent systems, pages 281–288, 2003.

[Cox and Durfee 2005] J. S. Cox and E. H. Durfee. *An efficient algorithm for multiagent plan coordination.* In Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, pages 828–835, 2005.

[Cox *et al.* 2005] J. S. Cox, E. H. Durfee and T. Bartold. *A distributed framework for solving the Multiagent Plan Coordination Problem.* In Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, pages 821–827, 2005.

[Dechter *et al.* 1991] R. Dechter, I. Meiri and J. Pearl. *Temporal constraint networks.* Artificial Intelligence, volume 49, no. 1-3, pages 61–95, 1991.

[Decker and Lesser 1992] K. Decker and V.R. Lesser. *Generalizing the partial global planning algorithm.* International Journal of Cooperative Information Systems, volume 2, no. 2, pages 319–346, 1992.

[Decker and Li 1998] K. Decker and J. Li. *Coordinated hospital patient scheduling.* In Proceedings of the International Conference on Multi Agent Systems, 1998, pages 104–111, Jul 1998.

[Demirkol *et al.* 1998] E. Demirkol, S. Mehta and R. Uzsoy. *Benchmarks for shop scheduling problems.* European Journal of Operational Research, volume 109, no. 1, pages 137 – 141, 1998.

[der Krogt and Weerdt 2005a] R. Van der Krogt and M. M. De Weerdt. *Plan Repair as an Extension of Planning.* In Proceedings of the International Conference on Automated Planning and Scheduling, pages 161–170, 2005.

[der Krogt and Weerdt 2005b] R. Van der Krogt and M. M. De Weerdt. *Self-interested Planning Agents Using Plan Repair.* In Proceedings of the ICAPS 2005 Workshop on Multiagent Planning and Scheduling, pages 36–44, 2005.

[Desjardins *et al.* 1999] M. E. Desjardins, E. H. Durfee, C. L. Ortiz and M. J. Wolverton. *A survey of research in distributed, continual planning.* AI Magazine, volume 20, pages 13–22, 1999.

[Dewan and Joshi 2002] P. Dewan and S. Joshi. *Auction-based distributed scheduling in a dynamic job shop environment.* International Journal of Production Research, volume 40, no. 5, pages 1173–1191, 2002.

[Dietz 2002] J.L.G Dietz. *The atoms, molecules and matter of organizations.* In Proceedings of the Seventh International Workshop on the Language/Action Perspective on Communication Modeling (LAP2002). Citeseer, 2002.

[Durfee and Lesser 1987] E. Durfee and V. Lesser. *Using Partial Global Plans to Coordinate Distributed Problem Solvers.* In Proceedings of the Tenth International Joint Conference on Artificial Intelligence, pages 875–883, August 1987.

[Durfee and Lesser 1991] E. H. Durfee and V. R. Lesser. *Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation.* IEEE Transactions on Systems, Man, and Cybernetics, volume 21, no. 5, pages 1167–1183, - 1991.

[Ephrati and Rosenschein 1993] E. Ephrati and J. S. Rosenschein. *Multi-Agent Planning as the Process of Merging Distributed Sub-Plans.* In Proceedings of the Twelfth International Workshop on Distributed Artificial Intelligence 1993, pages 115–129, May 1993.

[Ephrati and Rosenschein 1994] E. Ephrati and J. S. Rosenschein. *Divide and conquer in multi-agent planning.* In Proceedings of the twelfth national conference on Artificial intelligence, pages 375–380, 1994.

[Even *et al.* 1998] G. Even, J. S. Naor, B. Schieber and Madhusudan. *Approximating Minimum Feedback Sets and Multi-Cuts in Directed Graphs.* In Proceedings of the 4th International IPCO Conference on Integer Programming and Combinatorial Optimization, pages 14–28, 1998.

[Festa *et al.* 1999] P. Festa, P. M. Pardalos and M. G. C. Resende. *Feedback set problems.* In Handbook of combinatorial optimization, pages 209–259. 1999.

[Floyd 1962] R. W. Floyd. *Algorithm 97: Shortest path.* Communication of ACM, volume 5, no. 6, page 345, 1962.

[Ford and Fulkerson 1957] L. R. Ford and D. R. Fulkerson. *A simple algorithm for finding maximal network flows and an application to the Hitchcock problem.* Canadian Journal of Mathematics, volume 9, pages 210–218, 1957.

[Foulser *et al.* 1992] D. E. Foulser, M. Li and Q. Yang. *Theory and Algorithms for Plan Merging.* Artificial Intelligence, volume 57, no. 2–3, pages 143–181, 1992.

[Fox *et al.* 2006] M. Fox, A. Gerevini, D. Long and I. Serina. *Plan stability: Replanning versus plan repair.* In Proceedings of the International Conference on Automated Planning and Scheduling, pages 212–221, 2006.

[Freeman 2003] T. M. Freeman. Optimization techniques for task allocation and scheduling in distributed multi-agent operations. Master's thesis, Massachusetts Institute of Technology, 2003.

[Garey and Johnson 1979] M. R. Garey and D. S. Johnson. Computers and intractability - a guide to the theory of np-completeness. 1979.

[Gerkey and Mataric 2002] B. P Gerkey and M. J. Mataric. *Sold!: Auction Methods for Multirobot Coordination.* IEEE transactions on Robotics and Automation, volume 18, no. 5, pages 758–768, October 2002.

[Gerkey and Mataric 2003a] B.P. Gerkey and M. J. Mataric. *A formal framework for the study of task allocation in multi-robot systems.* International Journal of Robotics Research, volume 23, no. 9, pages 939–954, 2003.

[Gerkey and Mataric 2003b] B.P. Gerkey and M.J. Mataric. *Multi-robot task allocation: analyzing the complexity and optimality of key architectures.* In IEEE International Conference on Robotics and Automation, 2003. Proceedings, volume 3, pages 3862–3868, Sept. 2003.

[Gerson 1976] E. M. Gerson. *On" Quality of Life".* American Sociological Review, volume 41, no. 5, pages 793–806, 1976.

[Gou *et al.* 1998] L. Gou, P. B. Luh and Y. Kyoya. *Holonic manufacturing scheduling: architecture, cooperation mechanism, and implementation.* Computers in Industry, volume 37, no. 3, pages 213–231, 1998.

[Hunsberger 2002a] L. Hunsberger. *Algorithms for a Temporal Decoupling Problem in Multi-Agent Planning.* In Proceedings of the Second International Joint Conference on Autonomous Agents and MultiAgent Systems, 2002.

[Hunsberger 2002b] L. Hunsberger. *Group Decision Making and Temporal Reasoning.* Phd thesis, Harvard University, June 2002.

[Jennings 1993] N. R. Jennings. *Commitments and Conventions: The Foundation of Coordination in Multi-Agent Systems.* The Knowledge Engineering Review, volume 8, no. 3, pages 223–250, 1993.

[Jennings 1996] N. R. Jennings. *Coordination techniques for distributed artificial intelligence.* Foundations of distributed artificial intelligence, pages 187–210, 1996.

[Kaplansky and Meisels 2007] E. Kaplansky and A. Meisels. *Distributed personnel scheduling negotiation among scheduling agents.* Annals of Operations Research, volume 155, pages 227–255, 2007.

[Kim and B. C. Paulson 2003] K. Kim and Jr. B. C. Paulson. *Agent-Based Compensatory Negotiation Methodology to Facilitate Distributed Coordination of Project Schedule Changes.* Journal of Computing in Civil Engineering, volume 17, no. 1, pages 10–18, 2003.

[Kjenstad 1998] D. Kjenstad. *Coordinated Supply chain scheduling.* PhD thesis, Norwegian university of Science and Technology, 1998.

[Korf 1987] R. E. Korf. *Planning as search: A quantitative approach.* Artificial Intelligence, volume 33, no. 1, pages 65–88, 1987.

[Kouiss *et al.* 1997] K. Kouiss, H. Pierreval and N. Mebarki. *Using multi-agent architecture in FMS for dynamic scheduling.* Journal of Intelligent Manufacturing, volume 8, no. 1, pages 41–47, 1997.

[Kovács and Spens 2007] G. Kovács and K. M. Spens. *Humanitarian logistics in disaster relief operations.* International Journal of Physical Distribution & Logistics Management, volume 37, no. 2, pages 99–114, 2007.

[Kutanoglu and Wu 1999] E. Kutanoglu and S. D. Wu. *On combinatorial auction and Lagrangean relaxation for distributed resource scheduling.* IIE transactions, volume 31, no. 9, pages 813–826, 1999.

[Lau *et al.* 2005] J. S. K. Lau, G. Q. Huang, K. L. Mak and L. Liang. *Distributed project scheduling with information sharing in supply chains: part IItheoretical analysis and computational study.* International Journal of Production Research, volume 43, no. 23, pages 4899–4927, 2005.

[Lau *et al.* 2006] J. S. K. Lau, G. Q. Huang, K. L. Mak and L. Liang. *Agent-based modeling of supply chains for distributed scheduling.* IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, volume 36, no. 5, pages 847–861, Sept. 2006.

[Leeuwen *et al.* 2007] P. Van Leeuwen, L.I. Oei, P. Buzing, C. Witteveen, M. Ganzha, M. Paprzycki and T. Pelech-Pilichowski. *Adaptive Temporal Planning at Airports.* Proceedings of the International Multiconference on Computer Science and Information Technology, volume 2, pages 317–326, 2007.

[Lenstra and Kan 1979] J. K. Lenstra and A. H. G. Rinnooy Kan. *Computational Complexity of Discrete Optimization Problems.* In Discrete Optimization I Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium, volume 4 of *Annals of Discrete Mathematics*, pages 121 – 140. 1979.

[Lima *et al.* 2006] R. M. Lima, R. M. Sousa and P. J. Martins. *Distributed production planning and control agent-based system.* International Journal of Production Research, volume 44, no. 18, pages 3693–3709, 2006.

[Liu and Sycara 1995] J. S. Liu and K. P. Sycara. *Exploiting problem structure for distributed constraint optimization.* In Proceedings of the First International Conference on Multi-Agent Systems, pages 246–253, 1995.

[Malewicz *et al.* 2006] G. Malewicz, A. Russell and A. Shvartsman. *Distributed scheduling for disconnected cooperation.* Distributed Computing, volume 18, pages 409–420, 2006.

[Malone and Crowston 1994] T. W. Malone and K. Crowston. *The interdisciplinary study of coordination.* ACM Computing Survey, volume 26, no. 1, pages 87–119, 1994.

[Manne 1960] A. S. Manne. *On the job-shop scheduling problem.* Operations Research, pages 219–223, 1960.

[Maturana and Norrie 1996] F. P. Maturana and D. H. Norrie. *Multi-agent Mediator architecture for distributed manufacturing.* Journal of Intelligent Manufacturing, volume 7, no. 4, pages 257–270, 1996.

[Megow *et al.* 2006] N. Megow, M. Uetz and T. Vredeveld. *Models and algorithms for stochastic online scheduling.* Mathematics of Operations Research, pages 513–525, 2006.

[Moses and Tennenholtz 1992] Y. Moses and M. Tennenholtz. *On computational aspects of artificial social systems.* In Proceedings of the Distributed Artificial Intelligence conference-92, 1992.

[Parunak 1987] H. V. D. Parunak. *Manufacturing experience with the contract net.* Distributed artificial intelligence, volume 1, pages 285–310, 1987.

[Pijper 2010] H. J. Pijper. Coordination for agents with freedom of choice. Master's thesis, TUDelft, 2010.

[Policella *et al.* 2007] N. Policella, A. Cesta, A. Oddi and S. F. Smith. *From precedence constraint posting to partial order schedules: A CSP approach to Robust Scheduling.* Artificial Intelligence Communications, volume 20, no. 3, pages 163–180, 2007.

[Prabhu and Duffie 1994] V. Prabhu and N. Duffie. *Real-time distributed scheduling of heterarchical manufacturing system.* Journal of Manufacturing systems, volume 13, pages 94–107, 1994.

[Qutub *et al.* 1997] S. Qutub, R. Alami and F. Ingrand. *How to solve deadlock situations within the plan-merging paradigm for multi-robot cooperation.* In Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems, 1997, volume 3, pages 1610–1615, 1997.

[Ramamritham *et al.* 1989] K. Ramamritham, J. A. Stankovic and W. Zhao. *Distributed Scheduling of Tasks with Deadlines and Resource Requirements.* IEEE Transactions on Computers, volume 38, no. 8, pages 1110–1123, 1989.

[Roundy *et al.* 1991] R. O. Roundy, W. L Maxwell, Y. T. Herer, S. R. Tayur and A. W. Getzler. *A Price-Directed Approach to Real-Time Scheduling of Production Operations.* IIE Transactions, volume 23, no. 2, pages 149–160, 1991.

[Schubert and A.Gerevini 1995] L. Schubert and A.Gerevini. *Accelerating partial order planners by improving plan and goal choices.* In Proceedings of the Seventh International Conference on Tools with Artificial Intelligence, pages 442–450, 1995.

[Seeni *et al.* 2010] A. Seeni, B. Schfer and G. Hirzinger. Aerospace technologies advancements, chapitre Robot Mobility Systems for Planetary Surface Exploration State-of-the-Art and Future Outlook: A Literature Survey, pages 189–208. 2010.

[Shaw and Whinston 1988] M. J. Shaw and A. B Whinston. *A distributed knowledge-based approach to flexible automation: the contract net framework.* International Journal of Flexible Manufacturing Systems, volume 1, no. 1, pages 85–104, 1988.

[Shoham and Tennenholtz 1995] Y. Shoham and M. Tennenholtz. *On Social Laws for Artificial Agent Societies: Off-Line Design.* Artificial Intelligence, volume 73, no. 1–2, pages 231–252, 1995.

[Singh 1992] B. Singh. *Interconnected Roles (IR): A coordinated model.* Rapport technique, Technical Report CT-84-92, Microelectronics and Computer Technology Corporation, Austin, TX, 1992.

[Smith and Peot 1996] D. Smith and M. Peot. *Suspending recursion in causal-link planning.* In Proceedings of the 3rd International Conference on AI Planning Systems, 1996.

[Smith 1980] R. G. Smith. *The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver.* Transactions on Computers, volume C-29, no. 12, pages 1104–1113, December 1980.

[Solberg and Lin 1992] J. J. Solberg and G.Y. Lin. *Integrated Shop floor control using autonomous agents.* IIE Transactions, volume 24, no. 3, pages 57–71, 1992.

[Steeb *et al.* 1981] R. Steeb, S. J. Cammarata, F. Hayes-Roth, P. W. Thorndyke and R. Wesson. *Distributed intelligence for air fleet control.* 1981.

[Steenhuisen *et al.* 2006] J. R. Steenhuisen, C. Witteveen, A. W. terMors and J. M. Valk. *Framework and Complexity Results for Coordinating Non-Cooperative Planning Agents.* In Proceedings of the 4th German conference on Multi-Agent System Technologies, volume 4196 of *Lecture Notes in Artificial Intelligence*, pages 98–109, sep 2006.

[Steenhuisen *et al.* 2008] J. R. Steenhuisen, C. Witteveen and Y. Zhang. *Plan-Coordination Mechanisms and the Price of Autonomy.* In Computational Logic in Multi-Agent Systems, volume 5056 of *Lecture Notes in Artificial Intelligence*, pages 1–21. 2008.

[terMors and Witteveen 2005] A. W. terMors and C. Witteveen. *Coordinating Non Cooperative Planning Agents: Complexity Results.* In Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology, pages 407–413, 2005.

[terMors *et al.* 2010] A. terMors, C. Yadati, C. Witteveen and Y. Zhang. *Coordination by design and the price of autonomy.* Autonomous Agents and Multi-Agent Systems, volume 20, pages 308–341, 2010.

[terMors 2004] A. W. terMors. Coordinating autonomous planning agents. Master's thesis, TU Delft, April 2004.

[Toptal and Sabuncuoglu 2009] A. Toptal and I. Sabuncuoglu. *Distributed scheduling: a review of concepts and applications.* International Journal of Production Research, 2009.

[Valk 2005] J. M. Valk. *Coordination among Autonomous Planners.* PhD thesis, Department of Software technology, Electrical Engineering, Mathematics and Computer Science,TUDelft, 2005.

[von Martial 1992] F. von Martial. Coordinating plans of autonomous agents, volume 610 of *Lecture Notes on Artificial Intelligence*. 1992.

[Wang *et al.* 1997] M. Wang, S. Sethi and Velde .S. L. van de. *Scheduling a batching machine*. Operations Research, page 702, 1997.

[Wang *et al.* 2008] C. Wang, H. Ghenniwa and W. Shen. *Real time distributed shop floor scheduling using an agent-based service-oriented architecture*. International Journal of Production Research, volume 46, no. 9, pages 2433–2452, 2008.

[Wangermann and Stengel 1998] J. P. Wangermann and R. F. Stengel. *Principled negotiation between intelligent agents: a model for air traffic management*. Artificial Intelligence in Engineering, volume 12, no. 3, pages 177 – 187, 1998.

[Weerdt *et al.* 2003] M. M. De Weerdt, A. Bos, H. Tonino and C. Witteveen. *A Resource Logic For Multi-Agent Plan Merging*. Annals of Mathematics and Artificial Intelligence, volume 37, pages 93–130, January 2003.

[Weerdt 2003] M. M. De Weerdt. *Plan Merging in Multi-Agent systems*. PhD thesis, TU Delft, 2003.

[Weiss 2000] G. Weiss, editeur. Multiagent systems: a modern approach to distributed artificial intelligence. Cambridge, EUA : Massachusetts Institute of Technology, 2000.

[Wellman *et al.* 2001] M. P. Wellman, W. E. Walsh, P. R. Wurman and J. K. M. Mason. *Auction Protocols for Decentralized Scheduling,*. Games and Economic Behavior, volume 35, no. 1-2, pages 271 – 303, 2001.

[Wu *et al.* 2009a] M. Wu, M. M. De Weerdt and H. La Poutré. *Efficient Methods for Multi-agent Multi-issue Negotiation: Allocating Resources*. In Proceedings of the 12th International Conference on Principles of Practice in Multi-Agent Systems, pages 97–112, 2009.

[Wu *et al.* 2009b] M. Wu, M. M. De Weerdt, H. La Poutré, C. Yadati, Y. Zhang and C. Witteveen. *Multi-player Multi-issue Negotiation with Complete Informa-*

tion. In Proceedings of the International Workshop on Agent-based Complex Automated Negotiations, pages 87–92, 2009.

[Yadati *et al.* 2008a] C. Yadati, C. Witteveen, Y. Zhang, M. Wu and H.La Poutré. *Autonomous Scheduling.* In Proceedings of the Foundations of Computer Science, pages 73 – 79, 2008.

[Yadati *et al.* 2008b] C. Yadati, C. Witteveen, Y. Zhang, M. Wu and H. La Poutré. *Autonomous Scheduling with Unbounded and Bounded Agents.* In Proceedings of the 6th German conference on Multiagent System Technologies, MATES '08, pages 195–206, 2008.

[Yadati *et al.* 2010] C. Yadati, C. Witteveen and Y. Zhang. *COORDINATING AGENTS: An analysis of coordination in supply-chain management like tasks.* In The 2nd International Conference on Agents and Artificial Intelligence, volume 2, pages 218–223, 2010.

[Yadati *et al.* 2011] C. Yadati, C. Witteveen and Y. Zhang. *Improving Task-Based Plan Coordination.* In Collaborative Agents - Research and Development, volume 6066 of *Lecture Notes in Computer Science*, pages 175–186. 2011.

[Younes and Simmons 2003] H. L. S. Younes and R. G. Simmons. *VHPOP: Versatile heuristic partial order planner.* Journal of Artificial Intelligence Research, volume 20, no. 1, pages 405–430, 2003.

Summary

Whenever multiple agents come together to achieve a common goal, coordination between them becomes a vital part of their success. The problem of coordination between agents becomes even more vital in the absence of both cooperation and communication between them. This thesis addresses the issue of coordinating plans and schedules of agents when *(i)* they require to achieve a common goal and *(ii)* neither communication nor cooperation can be expected from them.

Earlier work on plan coordination mechanisms determined that the problem of coordinating plans is Σ_2^p complete in general. However, the problem of discovering subclasses of plan coordination problem instances, which are easier to solve was only partially addressed. Further, even when a plan can be coordinated, it does not naturally mean that it can be executed properly. That is, a coordinated plan may still be not implementable in practice. This phenomenon is evident when one is dealing with coordination instances which have temporal constraints. In the presence of temporal constraints, a coordinated plan becomes implementable only if the schedules are also coordinated. Therefore, it motivates us to look into the problem of coordinating schedules. To the best of our knowledge, there exists no previous work which studies this problem of coordinating schedules in a decentralised setting.

Our attempt is thus to address these two gaps,*(i)* finding subclasses of plan coordination problem instances which are easier to solve than the general class and *(ii)* the problem of coordinating schedules, in this thesis.

We approach the first problem by studying the structure of the graph

of precedence constraints called the *task graph* and show that there indeed is a class of plan coordination problem instances called the *Intra-free plan coordination* instances, which can be solved more easily than the general case. We design an algorithm that improves upon the state of the art algorithm for plan coordination, by exploiting the intra-free structure of problem instances and show that this algorithm is indeed better than the existing state-of-the-art algorithm for coordination. Further, we also show that there exists a subclass of intra-free instances called the class of *Special linear intra-free instances* for which the plan coordination problem can be solved efficiently.

We next study the problem of schedule coordination. Solving the schedule coordination problem is akin to decomposing a temporal network when capacity constraints are not imposed on agents. Therefore, we use and improve upon existing techniques for temporal decomposition and solve the schedule coordination problem. In the process we derive a faster and simpler algorithm called *Interval scheduling algorithm*, that solves the schedule coordination problem efficiently. Further it also ensures that the global makespan is minimum.

Unfortunately, once capacity constraints are introduced into the schedule coordination problem, we show that there exists no algorithm that can solve it efficiently and also ensure that the global makespan is minimum, unless $P = NP$.

One of the issues that we face with schedule coordination in the presence of capacity constraints is that of efficiently detecting conflicts. We show that as long as the durations of tasks are not super polynomial in the number of tasks, we can detect conflicts and resolve them efficiently. However, this still does not ensure the minimality of the global makespan. In fact, despite restricting the structure of the precedence constraints between tasks very severely, this problem remains NP-complete. Further, we show that even approximation algorithms tend to have very bad approximation ratios.

Fortunately however, we are able to show that if one could exploit the structure of the precedence relationships then, it is possible to improve the approximation ratios.

While the algorithms we develop for schedule coordination perform badly in terms of makespan minimality in the worst case, we do not have any theoretical result to predict their performance in a general setting. Therefore, we also perform empirical analysis of the schedule coordination algorithm to test its effectiveness in a practical setting.

Samenvatting

Wanneer meerdere agenten samen komen om een gemeenschappelijk doel te verwezenlijken, wordt de coördinatie tussen hen een essentieel onderdeel van hun succes. Het coördinatieprobleem wordt nog belangrijker indien er geen samenwerking en communicatie tussen de agenten is. Dit proefschrift behandelt het vraagstuk hoe de plannen en tijdsschema's van agenten te coördineren indien (*i*) ze een gemeenschappelijke doelstelling moeten bereiken, en (*ii*) coöperatie noch communicatie van hen verwacht kan worden.

Eerder werk op het gebied van plan-coördinatie mechanismen hebben vastgesteld dat het plan-coördinatieprobleem Σ_2^p-compleet is in zijn algemeenheid. Echter, het probleem om subklassen van het plan-coördinatieprobleem te ontdekken die eenvoudiger zijn op te lossen, is slechts ten dele geadresseerd. Bovendien, zelfs wanneer een plan *gecoördineerd* kan worden, volgt het niet vanzelfsprekend dat het plan ook correct *uitgevoerd* kan worden. Dit betekent dat een gecoördineerd plan nog niet in de praktijk uitvoerbaar hoeft te zijn. Dit fenomeen wordt duidelijk als men te maken heeft met coördinatie problemen die temporele *constraints* hebben. Met de aanwezigheid van temporele constraints zijn gecoördineerde plannen alleen uitvoerbaar indien de tijdsschema's ook gecoördineerd worden. Voor zover ons bekend bestaat er geen eerder werk dat dit probleem van het coördineren van tijdsschema's bestudeert in een gedecentraliseerde opzet.

Onze poging is om deze twee lacunes, namelijk (*i*) het vinden van subklassen van het plan-coördinatieprobleem die makkelijker op te lossen zijn dan het algemene geval, en (*ii*) het tijdsschema-coördinatieprobleem, in dit proefschrift te adresseren.

We benaderen het eerste probleem door de structuur van de graaf van de precedentierelatie — de *taakgraaf* genoemd — te bestuderen en we tonen aan dat er inderdaad een klasse van plan-coördinatie problemen is, genaamd *Intra-free plan coordination*, die eenvoudiger op te lossen is dan het algemene geval. We ontwerpen een algoritme dat de bestaande *state-of-the-art* voor plan-coördinatie verbetert door gebruik te maken van de *intra-free* structuur van de instanties en we tonen aan dat dit algoritme inderdaad beter is dan het bestaande state-of-the-art algoritme. Verder tonen we aan dat er ook een subklasse van *intra-free* instanties bestaat genaamd *Special linear intra-free* instanties waarvoor het plan-coördinatieprobleem efficiënt kan worden opgelost.

Vervolgens bestuderen we het probleem van tijdsschema-coördinatie. Het oplossen van het tijdsschema-coördinatieprobleem is verwant aan het opslitsen van een temporeel netwerk indien er geen capaciteitscontraints aan de agenten worden opgelegd. Daarom gebruiken en verbeteren we bestaande technieken voor temporele decompositie bij het oplossen van het tijdsschema-coördinatieprobleem. We ontwikkelen daarbij een sneller en eenvoudiger algoritme genaamd *Interval scheduling algorithm*, dat het tijdsschema-coördinatieprobleem efficiënt oplost. Verder garandeert het algoritme dat de globale tijdsspanne minimaal is.

Helaas kunnen we aantonen dat, zodra capaciteitsconstraints betrokken worden bij het tijdsschema-coördinatieprobleem, er geen algoritme bestaat dat het probleem efficiënt kan oplossen en kan garanderen dat de globale tijdspanne minimaal is, tenzij $P = NP$.

Eén van de problemen waar we tegenaan lopen bij tijdschema-coördinatie met capaciteitscontraints is het efficiënt detecteren van conflicten. We laten zien dat zolang de duur van de taken niet boven-polynomiaal is in het aantal taken, dan kunnen we conflicten efficiënt detecteren en oplossen. Echter, dit garandeert nog steeds niet de minimaliteit van de globale tijdsspanne. En ondanks de zeer sterke beperking op de structuur van de precedentierelatie blijft het probleem NP-compleet. Verder tonen we zelfs aan dat approximatie algoritmes dikwijls een erg slechte approximatie ratio's hebben.

Echter, we kunnen gelukkig aantonen dat als men erin slaagt om de structuur van de precedentierelatie aan te wenden, dan is het mogelijk om de approximatie ratio's te verbeteren.

Hoewel de algoritmes die we ontwikkelen voor tijdsschema-coördinatie slecht presteren op het gebied van tijdsspanne in het slechtste geval, hebben we geen theoretisch resultaat om te voorspellen hoe de prestaties zijn in algemene gevallen. Daartoe voeren we ook een empirische analyse uit van het tijdsschema-coördinatiealgoritme om de effectiviteit ervan te testen in een praktische omgeving.

Acknowledgements

One could compare the process of a dream fructifying to the process of growing a fruit bearing tree. My dream of completing a doctoral degree has been no exception to this comparison. My father planted the seeds for this dream by getting his own Doctorate when I was about 16 and ever since, my mother nurtured it and my brother secured it. All through the process, they have together protected me from distraction and supported me when I doubted in myself. The later on additions to our family — my sister-in-law and my niece have proved as booster doses to this process. The contribution of my family to my success has been immense to say the least. Prof. Narahari, my uncle, was one of my constant sources of technical and career advice.

One cannot but acknowledge the contribution of one's better half in any such endeavour. Shilpa was immensely patient through the whole process despite the fact that we had been recently married and we had not known each other earlier.

It is often believed that the contribution of the thesis supervisor is probably more significant to any thesis than the contribution of the candidate himself. I understand why this oft quoted statement is true. The contributions of both Yingqian and Cees to my thesis and doctoral study have been vital. Their company, patient supervision and technical insights served as the cornerstones of my thesis.

I would like to acknowledge all the support, friendship and trust I received from all my colleagues at the Algorithms group, the support I received from Vijay, currently pursuing his own doctoral dream at TUDelft, and the advice I received from VP in the erstwhile mobile and wireless technology group of TUDelft.

Finally, I would like to acknowledge the support I received from my employer — at India Science Laboratory, GMTCI. I finished the final draft of my thesis at GM and all through the process of my completing the final draft, my colleagues at GM as well as my manager supported me. There are many others I would like to thank, however, I fear that the acknowledgement section would outgrow other technical sections if I do so. Hence, I stop by saying a big THANK YOU to all of those who helped me grow into whatever I am today.

Curriculum vitae

Chetan was born in Bangalore (now Bengaluru) in 1979 and grew up into the most mischievous kid of the colony. He was most often spotted on roof tops (and sometimes treetops) of neighbours mulling and getting disappointed that there were no greater heights to climb. Sickness often forced him to be bedridden but it also introduced to him the wonderful habit of reading. His parents eventually heaved a sigh of relief when his voraciousness to climb heights meta-morphed into a voraciousness to read.

College taught him that abstract ideas were fine, but they had little relevance if they could not be precise. The two years at the erstwhile Electronic Enterprises Laboratory at Indian Institute of Science under the tutelage of Prof Narahari, taught him the value of organised thinking and also a lot of programming.

His masters at The School of Industrial Engineering and Management at Oklahoma State University under the tutelage of Prof. Manjunath Kamath, taught him many things but most importantly that "in research, the devil was always in the details". Finally, his education under Prof. Cees Witteveen at TU Delft, was responsible for ironing out a multitude of chinks in his armour — starting from the discomfort with reading formal language in computer science.

Currently he works in the general areas of mathematical programming and scheduling.

SIKS Dissertatiereeks

====
1998
====

1998-1 Johan van den Akker (CWI)
DEGAS - An Active, Temporal Database of Autonomous Objects

1998-2 Floris Wiesman (UM)
Information Retrieval by Graphically Browsing Meta-Information

1998-3 Ans Steuten (TUD)
A Contribution to the Linguistic Analysis of Business Conversations
within the Language/Action Perspective

1998-4 Dennis Breuker (UM)
Memory versus Search in Games
1998-5 E.W.Oskamp (RUL)
Computerondersteuning bij Straftoemeting

====
1999
====

1999-1 Mark Sloof (VU)
Physiology of Quality Change Modelling;
Automated modelling of Quality Change of Agricultural Products

1999-2 Rob Potharst (EUR)
Classification using decision trees and neural nets

1999-3 Don Beal (UM)
The Nature of Minimax Search

1999-4 Jacques Penders (UM)
The practical Art of Moving Physical Objects

1999-5 Aldo de Moor (KUB)
Empowering Communities: A Method for the Legitimate User-Driven
Specification of Network Information Systems

1999-6 Niek J.E. Wijngaards (VU)
Re-design of compositional systems

1999-7 David Spelt (UT)
Verification support for object database design

1999-8 Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent
Mechanism for Discrete Reallocation.

====
2000
====

2000-1 Frank Niessink (VU)
Perspectives on Improving Software Maintenance

2000-2 Koen Holtman (TUE)
Prototyping of CMS Storage Management

2000-3 Carolien M.T. Metselaar (UVA)
Sociaal-organisatorische gevolgen van kennistechnologie;
een procesbenadering en actorperspectief.

2000-4 Geert de Haan (VU)
ETAG, A Formal Model of Competence Knowledge for User Interface Design

2000-5 Ruud van der Pol (UM)
Knowledge-based Query Formulation in Information Retrieval.

2000-6 Rogier van Eijk (UU)
Programming Languages for Agent Communication

2000-7 Niels Peek (UU)
Decision-theoretic Planning of Clinical Patient Management

2000-8 Veerle Coup (EUR)
Sensitivity Analyis of Decision-Theoretic Networks

2000-9 Florian Waas (CWI)
Principles of Probabilistic Query Optimization

2000-10 Niels Nes (CWI)
Image Database Management System Design Considerations,
Algorithms and Architecture

2000-11 Jonas Karlsson (CWI)
Scalable Distributed Data Structures for Database Management

====
2001
====

2001-1 Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks

2001-2 Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models

2001-3 Maarten van Someren (UvA)
Learning as problem solving

2001-4 Evgueni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with
Instance-Based Boundary Sets

2001-5 Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style

2001-6 Martijn van Welie (VU)
Task-based User Interface Design

2001-7 Bastiaan Schonhage (VU)
Diva: Architectural Perspectives on Information Visualization

2001-8 Pascal van Eck (VU)
A Compositional Semantic Structure for Multi-Agent Systems Dynamics.

2001-9 Pieter Jan 't Hoen (RUL)
Towards Distributed Development of Large Object-Oriented Models,
Views of Packages as Classes

2001-10 Maarten Sierhuis (UvA)
Modeling and Simulating Work Practice
BRAHMS: a multiagent modeling and simulation language
for work practice analysis and design

2001-11 Tom M. van Engers (VUA)
Knowledge Management:
The Role of Mental Models in Business Systems Design

====
2002

====

2002-01 Nico Lassing (VU)
Architecture-Level Modifiability Analysis

2002-02 Roelof van Zwol (UT)
Modelling and searching web-based document collections

2002-03 Henk Ernst Blok (UT)
Database Optimization Aspects for Information Retrieval

2002-04 Juan Roberto Castelo Valdueza (UU)
The Discrete Acyclic Digraph Markov Model in Data Mining

2002-05 Radu Serban (VU)
The Private Cyberspace Modeling Electronic Environments
inhabited by Privacy-concerned Agents

2002-06 Laurens Mommers (UL)
Applied legal epistemology;
Building a knowledge-based ontology of the legal domain

2002-07 Peter Boncz (CWI)
Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications

2002-08 Jaap Gordijn (VU)
Value Based Requirements Engineering: Exploring Innovative
E-Commerce Ideas

2002-09 Willem-Jan van den Heuvel(KUB)
Integrating Modern Business Applications with Objectified Legacy Systems

2002-10 Brian Sheppard (UM)
Towards Perfect Play of Scrabble

2002-11 Wouter C.A. Wijngaards (VU)
Agent Based Modelling of Dynamics: Biological and Organisational Applications

2002-12 Albrecht Schmidt (Uva)
Processing XML in Database Systems

2002-13 Hongjing Wu (TUE)
A Reference Architecture for Adaptive Hypermedia Applications

2002-14 Wieke de Vries (UU)
Agent Interaction: Abstract Approaches to Modelling, Programming and
Verifying Multi-Agent Systems

2002-15 Rik Eshuis (UT)
Semantics and Verification of UML Activity Diagrams for Workflow Modelling

2002-16 Pieter van Langen (VU)
The Anatomy of Design: Foundations, Models and Applications

2002-17 Stefan Manegold (UVA)
Understanding, Modeling, and Improving Main-Memory Database Performance

====
2003
====

2003-01 Heiner Stuckenschmidt (VU)
Ontology-Based Information Sharing in Weakly Structured Environments

2003-02 Jan Broersen (VU)
Modal Action Logics for Reasoning About Reactive Systems

2003-03 Martijn Schuemie (TUD)
Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy

2003-04 Milan Petkovic (UT)
Content-Based Video Retrieval Supported by Database Technology

2003-05 Jos Lehmann (UVA)
Causation in Artificial Intelligence and Law - A modelling approach

2003-06 Boris van Schooten (UT)
Development and specification of virtual environments

2003-07 Machiel Jansen (UvA)
Formal Explorations of Knowledge Intensive Tasks

2003-08 Yongping Ran (UM)
Repair Based Scheduling

2003-09 Rens Kortmann (UM)
The resolution of visually guided behaviour

2003-10 Andreas Lincke (UvT)
Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture

2003-11 Simon Keizer (UT)

Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks

2003-12 Roeland Ordelman (UT)
Dutch speech recognition in multimedia information retrieval

2003-13 Jeroen Donkers (UM)
Nosce Hostem - Searching with Opponent Models

2003-14 Stijn Hoppenbrouwers (KUN)
Freezing Language: Conceptualisation Processes across ICT-Supported Organisations

2003-15 Mathijs de Weerdt (TUD)
Plan Merging in Multi-Agent Systems

2003-16 Menzo Windhouwer (CWI)
Feature Grammar Systems - Incremental Maintenance of Indexes to
Digital Media Warehouses

2003-17 David Jansen (UT)
Extensions of Statecharts with Probability, Time, and Stochastic Timing

2003-18 Levente Kocsis (UM)
Learning Search Decisions

====
2004
====

2004-01 Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic

2004-02 Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business

2004-03 Perry Groot (VU)
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving

2004-04 Chris van Aart (UVA)
Organizational Principles for Multi-Agent Architectures

2004-05 Viara Popova (EUR)
Knowledge discovery and monotonicity

2004-06 Bart-Jan Hommes (TUD)
The Evaluation of Business Process Modeling Techniques

2004-07 Elise Boltjes (UM)
Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes

2004-08 Joop Verbeek(UM)
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiële gegevensuitwisseling en digitale expertise

2004-09 Martin Caminada (VU)
For the Sake of the Argument; explorations into argument-based reasoning

2004-10 Suzanne Kabel (UVA)
Knowledge-rich indexing of learning-objects

2004-11 Michel Klein (VU)
Change Management for Distributed Ontologies

2004-12 The Duy Bui (UT)
Creating emotions and facial expressions for embodied agents

2004-13 Wojciech Jamroga (UT)
Using Multiple Models of Reality: On Agents who Know how to Play

2004-14 Paul Harrenstein (UU)
Logic in Conflict. Logical Explorations in Strategic Equilibrium

2004-15 Arno Knobbe (UU)
Multi-Relational Data Mining

2004-16 Federico Divina (VU)
Hybrid Genetic Relational Search for Inductive Learning

2004-17 Mark Winands (UM)
Informed Search in Complex Games

2004-18 Vania Bessa Machado (UvA)
Supporting the Construction of Qualitative Knowledge Models

2004-19 Thijs Westerveld (UT)
Using generative probabilistic models for multimedia retrieval

2004-20 Madelon Evers (Nyenrode)
Learning from Design: facilitating multidisciplinary design teams

====
2005
====

2005-01 Floor Verdenius (UVA)
Methodological Aspects of Designing Induction-Based Applications

2005-02 Erik van der Werf (UM))
AI techniques for the game of Go

2005-03 Franc Grootjen (RUN)
A Pragmatic Approach to the Conceptualisation of Language

2005-04 Nirvana Meratnia (UT)
Towards Database Support for Moving Object data

2005-05 Gabriel Infante-Lopez (UVA)
Two-Level Probabilistic Grammars for Natural Language Parsing

2005-06 Pieter Spronck (UM)
Adaptive Game AI

2005-07 Flavius Frasincar (TUE)
Hypermedia Presentation Generation for Semantic Web Information Systems

2005-08 Richard Vdovjak (TUE)
A Model-driven Approach for Building Distributed Ontology-based Web Applications

2005-09 Jeen Broekstra (VU)
Storage, Querying and Inferencing for Semantic Web Languages

2005-10 Anders Bouwer (UVA)
Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments

2005-11 Elth Ogston (VU)
Agent Based Matchmaking and Clustering - A Decentralized Approach to Search

2005-12 Csaba Boer (EUR)
Distributed Simulation in Industry

2005-13 Fred Hamburg (UL)
Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen

2005-14 Borys Omelayenko (VU)
Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics

2005-15 Tibor Bosse (VU)
Analysis of the Dynamics of Cognitive Processes

2005-16 Joris Graaumans (UU)
Usability of XML Query Languages

2005-17 Boris Shishkov (TUD)
Software Specification Based on Re-usable Business Components

2005-18 Danielle Sent (UU)
Test-selection strategies for probabilistic networks

2005-19 Michel van Dartel (UM)
Situated Representation

2005-20 Cristina Coteanu (UL)
Cyber Consumer Law, State of the Art and Perspectives

2005-21 Wijnand Derks (UT)
Improving Concurrency and Recovery in Database Systems by
Exploiting Application Semantics

====
2006
====

2006-01 Samuil Angelov (TUE)
Foundations of B2B Electronic Contracting

2006-02 Cristina Chisalita (VU)
Contextual issues in the design and use of information technology in organizations

2006-03 Noor Christoph (UVA)
The role of metacognitive skills in learning to solve problems

2006-04 Marta Sabou (VU)
Building Web Service Ontologies

2006-05 Cees Pierik (UU)
Validation Techniques for Object-Oriented Proof Outlines

2006-06 Ziv Baida (VU)
Software-aided Service Bundling - Intelligent Methods & Tools
for Graphical Service Modeling

2006-07 Marko Smiljanic (UT)
XML schema matching – balancing efficiency and effectiveness by means of clustering

2006-08 Eelco Herder (UT)

Forward, Back and Home Again - Analyzing User Behavior on the Web

2006-09 Mohamed Wahdan (UM)
Automatic Formulation of the Auditor's Opinion

2006-10 Ronny Siebes (VU)
Semantic Routing in Peer-to-Peer Systems

2006-11 Joeri van Ruth (UT)
Flattening Queries over Nested Data Types

2006-12 Bert Bongers (VU)
Interactivation - Towards an e-cology of people, our technological environment, and the arts

2006-13 Henk-Jan Lebbink (UU)
Dialogue and Decision Games for Information Exchanging Agents

2006-14 Johan Hoorn (VU)
Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change

2006-15 Rainer Malik (UU)
CONAN: Text Mining in the Biomedical Domain

2006-16 Carsten Riggelsen (UU)
Approximation Methods for Efficient Learning of Bayesian Networks

2006-17 Stacey Nagata (UU)
User Assistance for Multitasking with Interruptions on a Mobile Device

2006-18 Valentin Zhizhkun (UVA)
Graph transformation for Natural Language Processing

2006-19 Birna van Riemsdijk (UU)
Cognitive Agent Programming: A Semantic Approach

2006-20 Marina Velikova (UvT)
Monotone models for prediction in data mining

2006-21 Bas van Gils (RUN)
Aptness on the Web

2006-22 Paul de Vrieze (RUN)
Fundaments of Adaptive Personalisation

2006-23 Ion Juvina (UU)
Development of Cognitive Model for Navigating on the Web

2006-24 Laura Hollink (VU)

Semantic Annotation for Retrieval of Visual Resources

2006-25 Madalina Drugan (UU)
Conditional log-likelihood MDL and Evolutionary MCMC

2006-26 Vojkan Mihajlovic (UT)
Score Region Algebra: A Flexible Framework for Structured Information Retrieval

2006-27 Stefano Bocconi (CWI)
Vox Populi: generating video documentaries from semantically annotated media repositories

2006-28 Borkur Sigurbjornsson (UVA)
Focused Information Access using XML Element Retrieval

====
2007
====

2007-01 Kees Leune (UvT)
Access Control and Service-Oriented Architectures

2007-02 Wouter Teepe (RUG)
Reconciling Information Exchange and Confidentiality: A Formal Approach

2007-03 Peter Mika (VU)
Social Networks and the Semantic Web

2007-04 Jurriaan van Diggelen (UU)
Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach

2007-05 Bart Schermer (UL)
Software Agents, Surveillance, and the Right to Privacy: a Legislative
Framework for Agent-enabled Surveillance

2007-06 Gilad Mishne (UVA)
Applied Text Analytics for Blogs

2007-07 Natasa Jovanovic' (UT)
To Whom It May Concern - Addressee Identification in Face-to-Face Meetings

2007-08 Mark Hoogendoorn (VU)
Modeling of Change in Multi-Agent Organizations

2007-09 David Mobach (VU)
Agent-Based Mediated Service Negotiation

2007-10 Huib Aldewereld (UU)

Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols

2007-11 Natalia Stash (TUE)
Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System

2007-12 Marcel van Gerven (RUN)
Bayesian Networks for Clinical Decision Support: A Rational Approach
to Dynamic Decision-Making under Uncertainty

2007-13 Rutger Rienks (UT)
Meetings in Smart Environments; Implications of Progressing Technology

2007-14 Niek Bergboer (UM)
Context-Based Image Analysis

2007-15 Joyca Lacroix (UM)
NIM: a Situated Computational Memory Model

2007-16 Davide Grossi (UU)
Designing Invisible Handcuffs. Formal investigations in Institutions and
Organizations for Multi-agent Systems

2007-17 Theodore Charitos (UU)
Reasoning with Dynamic Networks in Practice

2007-18 Bart Orriens (UvT)
On the development an management of adaptive business collaborations

2007-19 David Levy (UM)
Intimate relationships with artificial partners

2007-20 Slinger Jansen (UU)
Customer Configuration Updating in a Software Supply Network

2007-21 Karianne Vermaas (UU)
Fast diffusion and broadening use: A research on residential adoption and
usage of broadband internet in the Netherlands between 2001 and 2005

2007-22 Zlatko Zlatev (UT)
Goal-oriented design of value and process models from patterns

2007-23 Peter Barna (TUE)
Specification of Application Logic in Web Information Systems

2007-24 Georgina Ramrez Camps (CWI)
Structural Features in XML Retrieval

2007-25 Joost Schalken (VU)

Empirical Investigations in Software Process Improvement

====
2008
====

2008-01 Katalin Boer-Sorbn (EUR)
Agent-Based Simulation of Financial Markets: A modular,continuous-time approach

2008-02 Alexei Sharpanskykh (VU)
On Computer-Aided Methods for Modeling and Analysis of Organizations

2008-03 Vera Hollink (UVA)
Optimizing hierarchical menus: a usage-based approach

2008-04 Ander de Keijzer (UT)
Management of Uncertain Data - towards unattended integration

2008-05 Bela Mutschler (UT)
Modeling and simulating causal dependencies on process-aware information systems from a cost perspective

2008-06 Arjen Hommersom (RUN)
On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective

2008-07 Peter van Rosmalen (OU)
Supporting the tutor in the design and support of adaptive e-learning

2008-08 Janneke Bolt (UU)
Bayesian Networks: Aspects of Approximate Inference

2008-09 Christof van Nimwegen (UU)
The paradox of the guided user: assistance can be counter-effective

2008-10 Wauter Bosma (UT)
Discourse oriented summarization

2008-11 Vera Kartseva (VU)
Designing Controls for Network Organizations: A Value-Based Approach

2008-12 Jozsef Farkas (RUN)
A Semiotically Oriented Cognitive Model of Knowledge Representation

2008-13 Caterina Carraciolo (UVA)

Topic Driven Access to Scientific Handbooks

2008-14 Arthur van Bunningen (UT)
Context-Aware Querying; Better Answers with Less Effort

2008-15 Martijn van Otterlo (UT)
The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains.

2008-16 Henriette van Vugt (VU)
Embodied agents from a user's perspective

2008-17 Martin Op 't Land (TUD)
Applying Architecture and Ontology to the Splitting and Allying of Enterprises

2008-18 Guido de Croon (UM)
Adaptive Active Vision

2008-19 Henning Rode (UT)
From Document to Entity Retrieval: Improving Precision and Performance of Focused Text Search

2008-20 Rex Arendsen (UVA)
Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven

2008-21 Krisztian Balog (UVA)
People Search in the Enterprise

2008-22 Henk Koning (UU)
Communication of IT-Architecture

2008-23 Stefan Visscher (UU)
Bayesian network models for the management of ventilator-associated pneumonia

2008-24 Zharko Aleksovski (VU)
Using background knowledge in ontology matching

2008-25 Geert Jonker (UU)
Efficient and Equitable Exchange in Air Traffic Management Plan Repair using Spender-signed Currency

2008-26 Marijn Huijbregts (UT)
Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled

2008-27 Hubert Vogten (OU)
Design and Implementation Strategies for IMS Learning Design

2008-28 Ildiko Flesch (RUN)
On the Use of Independence Relations in Bayesian Networks

2008-29 Dennis Reidsma (UT)
Annotations and Subjective Machines - Of Annotators, Embodied Agents, Users, and Other Humans

2008-30 Wouter van Atteveldt (VU)
Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content

2008-31 Loes Braun (UM)
Pro-Active Medical Information Retrieval

2008-32 Trung H. Bui (UT)
Toward Affective Dialogue Management using Partially Observable Markov Decision Processes

2008-33 Frank Terpstra (UVA)
Scientific Workflow Design; theoretical and practical issues

2008-34 Jeroen de Knijf (UU)
Studies in Frequent Tree Mining

2008-35 Ben Torben Nielsen (UvT)
Dendritic morphologies: function shapes structure

====
2009
====

2009-01 Rasa Jurgelenaite (RUN)
Symmetric Causal Independence Models

2009-02 Willem Robert van Hage (VU)
Evaluating Ontology-Alignment Techniques

2009-03 Hans Stol (UvT)
A Framework for Evidence-based Policy Making Using IT

2009-04 Josephine Nabukenya (RUN)
Improving the Quality of Organisational Policy Making using Collaboration Engineering

2009-05 Sietse Overbeek (RUN)
Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality

2009-06 Muhammad Subianto (UU)
Understanding Classification

2009-07 Ronald Poppe (UT)
Discriminative Vision-Based Recovery and Recognition of Human Motion

2009-08 Volker Nannen (VU)
Evolutionary Agent-Based Policy Analysis in Dynamic Environments

2009-09 Benjamin Kanagwa (RUN)
Design, Discovery and Construction of Service-oriented Systems

2009-10 Jan Wielemaker (UVA)
Logic programming for knowledge-intensive interactive applications

2009-11 Alexander Boer (UVA)
Legal Theory, Sources of Law & the Semantic Web

2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin)
operating Guidelines for Services

2009-13 Steven de Jong (UM)
Fairness in Multi-Agent Systems

2009-14 Maksym Korotkiy (VU)
From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)

2009-15 Rinke Hoekstra (UVA)
Ontology Representation - Design Patterns and Ontologies that Make Sense
2009-16 Fritz Reul (UvT)
New Architectures in Computer Chess
2009-17 Laurens van der Maaten (UvT)
Feature Extraction from Visual Data
2009-18 Fabian Groffen (CWI)
Armada, An Evolving Database System

2009-19 Valentin Robu (CWI)
Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets

2009-20 Bob van der Vecht (UU)
Adjustable Autonomy: Controling Influences on Decision Making

2009-21 Stijn Vanderlooy (UM)
Ranking and Reliable Classification

2009-22 Pavel Serdyukov (UT)
Search For Expertise: Going beyond direct evidence

2009-23 Peter Hofgesang (VU)
Modelling Web Usage in a Changing Environment

2009-24 Annerieke Heuvelink (VUA)
Cognitive Models for Training Simulations

2009-25 Alex van Ballegooij (CWI)
"RAM: Array Database Management through Relational Mapping"

2009-26 Fernando Koch (UU)
An Agent-Based Model for the Development of Intelligent Mobile Services

2009-27 Christian Glahn (OU)
Contextual Support of social Engagement and Reflection on the Web

2009-28 Sander Evers (UT)
Sensor Data Management with Probabilistic Models

2009-29 Stanislav Pokraev (UT)
Model-Driven Semantic Integration of Service-Oriented Applications

2009-30 Marcin Zukowski (CWI)
Balancing vectorized query execution with bandwidth-optimized storage

2009-31 Sofiya Katrenko (UVA)
A Closer Look at Learning Relations from Text

2009-32 Rik Farenhorst (VU) and Remco de Boer (VU)
Architectural Knowledge Management: Supporting Architects and Auditors

2009-33 Khiet Truong (UT)
How Does Real Affect Affect Affect Recognition In Speech?

2009-34 Inge van de Weerd (UU)
Advancing in Software Product Management: An Incremental Method Engineering Approach

2009-35 Wouter Koelewijn (UL)
Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling

2009-36 Marco Kalz (OUN)
Placement Support for Learners in Learning Networks

2009-37 Hendrik Drachsler (OUN)

Navigation Support for Learners in Informal Learning Networks

2009-38 Riina Vuorikari (OU)
Tags and self-organisation: a metadata ecology for learning resources in a multilingual context

2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin)
Service Substitution – A Behavioral Approach Based on Petri Nets

2009-40 Stephan Raaijmakers (UvT)
Multinomial Language Learning: Investigations into the Geometry of Language

2009-41 Igor Berezhnyy (UvT)
Digital Analysis of Paintings

2009-42 Toine Bogers
Recommender Systems for Social Bookmarking

2009-43 Virginia Nunes Leal Franqueira (UT)
Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients

2009-44 Roberto Santana Tapia (UT)
Assessing Business-IT Alignment in Networked Organizations

2009-45 Jilles Vreeken (UU)
Making Pattern Mining Useful

2009-46 Loredana Afanasiev (UvA)
Querying XML: Benchmarks and Recursion

====
2010
====

2010-01 Matthijs van Leeuwen (UU)
Patterns that Matter

2010-02 Ingo Wassink (UT)
Work flows in Life Science

2010-03 Joost Geurts (CWI)
A Document Engineering Model and Processing Framework for Multimedia documents
2010-04 Olga Kulyk (UT)
Do You Know What I Know? Situational Awareness of Co-located Teams in

Multidisplay Environments

2010-05 Claudia Hauff (UT)
Predicting the Effectiveness of Queries and Retrieval Systems

2010-06 Sander Bakkes (UvT)
Rapid Adaptation of Video Game AI

2010-07 Wim Fikkert (UT)
Gesture interaction at a Distance

2010-08 Krzysztof Siewicz (UL)
Towards an Improved Regulatory Framework of Free Software.
Protecting user freedoms in a world of software communities and
eGovernments

2010-09 Hugo Kielman (UL)
A Politiele gegevensverwerking en Privacy, Naar een effectieve
waarborging

2010-10 Rebecca Ong (UL)
Mobile Communication and Protection of Children 2010-11 Adriaan Ter Mors (TUD)
The world according to MARP: Multi-Agent Route Planning

2010-12 Susan van den Braak (UU)
Sensemaking software for crime analysis

2010-13 Gianluigi Folino (RUN)
High Performance Data Mining using Bio-inspired techniques

2010-14 Sander van Splunter (VU)
Automated Web Service Reconfiguration

2010-15 Lianne Bodenstaff (UT)
Managing Dependency Relations in Inter-Organizational Models

2010-16 Sicco Verwer (TUD)
Efficient Identification of Timed Automata, theory and practice

2010-17 Spyros Kotoulas (VU)
Scalable Discovery of Networked Resources: Algorithms, Infrastructure,
Applications

2010-18 Charlotte Gerritsen (VU)
Caught in the Act: Investigating Crime by Agent-Based Simulation

2010-19 Henriette Cramer (UvA)
People's Responses to Autonomous and Adaptive Systems

2010-20 Ivo Swartjes (UT)
Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative

2010-21 Harold van Heerde (UT)
Privacy-aware data management by means of data degradation

2010-22 Michiel Hildebrand (CWI)
End-user Support for Access to
Heterogeneous Linked Data

2010-23 Bas Steunebrink (UU)
The Logical Structure of Emotions
2010-24 Dmytro Tykhonov
Designing Generic and Efficient Negotiation Strategies

2010-25 Zulfiqar Ali Memon (VU)
Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective

2010-26 Ying Zhang (CWI)
XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines
2010-27 Marten Voulon (UL)
Automatisch contracteren

2010-28 Arne Koopman (UU)
Characteristic Relational Patterns

2010-29 Stratos Idreos(CWI)
Database Cracking: Towards Auto-tuning Database Kernels

2010-30 Marieke van Erp (UvT)
Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval

2010-31 Victor de Boer (UVA)
Ontology Enrichment from Heterogeneous Sources on the Web

2010-32 Marcel Hiel (UvT)
An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems

2010-33 Robin Aly (UT)
Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval

2010-34 Teduh Dirgahayu (UT)
Interaction Design in Service Compositions

2010-35 Dolf Trieschnigg (UT)
Proof of Concept: Concept-based Biomedical Information Retrieval

2010-36 Jose Janssen (OU)
Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification

2010-37 Niels Lohmann (TUE)
Correctness of services and their composition

2010-38 Dirk Fahland (TUE)
From Scenarios to components

2010-39 Ghazanfar Farooq Siddiqui (VU)
Integrative modeling of emotions in virtual agents

2010-40 Mark van Assem (VU)
Converting and Integrating Vocabularies for the Semantic Web

2010-41 Guillaume Chaslot (UM)
Monte-Carlo Tree Search

2010-42 Sybren de Kinderen (VU)
Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach

2010-43 Peter van Kranenburg (UU)
A Computational Approach to Content-Based Retrieval of Folk Song Melodies

2010-44 Pieter Bellekens (TUE)
An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain

2010-45 Vasilios Andrikopoulos (UvT)
A theory and model for the evolution of software services

2010-46 Vincent Pijpers (VU)
e3alignment: Exploring Inter-Organizational Business-ICT Alignment

2010-47 Chen Li (UT)
Mining Process Model Variants: Challenges, Techniques, Examples

2010-48 Milan Lovric (EUR)

Behavioral Finance and Agent-Based Artificial Markets

2010-49 Jahn-Takeshi Saito (UM)
Solving difficult game positions

2010-50 Bouke Huurnink (UVA)
Search in Audiovisual Broadcast Archives
2010-51 Alia Khairia Amin (CWI)
Understanding and supporting information seeking tasks in multiple sources

2010-52 Peter-Paul van Maanen (VU)
Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention

2010-53 Edgar Meij (UVA)
Combining Concepts and Language Models for Information Access

====
2011
====

2011-01 Botond Cseke (RUN)
Variational Algorithms for Bayesian Inference in Latent Gaussian Models

2011-02 Nick Tinnemeier(UU)
Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language

2011-03
Jan Martijn van der Werf (TUE)
Compositional Design and Verification of Component-Based Information Systems

2011-04 Hado van Hasselt (UU)
Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms

2011-05 Base van der Raadt (VU)
Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.

2011-06 Yiwen Wang (TUE)
Semantically-Enhanced Recommendations in Cultural Heritage

2011-07 Yujia Cao (UT)
Multimodal Information Presentation for High Load Human Computer

Interaction

2011-08 Nieske Vergunst (UU)
BDI-based Generation of Robust Task-Oriented Dialogues 2011-09 Tim de Jong (OU)
Contextualised Mobile Media for Learning

2011-10 Bart Bogaert (UvT)
Cloud Content Contention

2011-11 Dhaval Vyas (UT)
Designing for Awareness: An Experience-focused HCI Perspective

2011-12 Carmen Bratosin (TUE)
Grid Architecture for Distributed Process Mining

2011-13 Xiaoyu Mao (UvT)
Airport under Control. Multiagent Scheduling for Airport Ground Handling

2011-14 Milan Lovric (EUR)
Behavioral Finance and Agent-Based Artificial Markets

2011-15 Marijn Koolen (UvA)
The Meaning of Structure: the Value of Link Evidence for Information Retrieval

2011-16 Maarten Schadd (UM)
Selective Search in Games of Different Complexity

2011-17 Jiyin He (UVA)
Exploring Topic Structure: Coherence, Diversity and Relatedness

2011-18 Mark Ponsen (UM)
Strategic Decision-Making in complex games

2011-19 Ellen Rusman (OU)
The Mind ' s Eye on Personal Profiles

2011-20 Qing Gu (VU)
Guiding service-oriented software engineering - A view-based approach

2011-21 Linda Terlouw (TUD)
Modularization and Specification of Service-Oriented Systems

2011-22 Junte Zhang (UVA)
System Evaluation of Archival Description and Access

2011-23 Wouter Weerkamp (UVA)

Finding People and their Utterances in Social Media

2011-24 Herwin van Welbergen (UT)
Behavior Generation for Interpersonal Coordination with Virtual Humans
On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior

2011-25 Syed Waqar ul Qounain Jaffry (VU)
Analysis and Validation of Models for Trust Dynamics

2011-26 Matthijs Aart Pontier (VU)
Virtual Agents for Human Communication - Emotion Regulation and
Involvement-Distance Trade-Offs in Embodied Conversational Agents
and Robots

2011-27 Aniel Bhulai (VU)
Dynamic website optimization through autonomous management of design
patterns

2011-28 Rianne Kaptein(UVA)
Effective Focused Retrieval by Exploiting Query Context and Document
Structure

2011-29 Faisal Kamiran (TUE)
Discrimination-aware Classification

2011-30 Egon van den Broek (UT)
Affective Signal Processing (ASP): Unraveling the mystery of emotions

2011-31 Ludo Waltman (EUR)
Computational and Game-Theoretic Approaches for Modeling Bounded
Rationality
2011-32 Nees-Jan van Eck (EUR)
Methodological Advances in Bibliometric Mapping of Science

2011-33 Tom van der Weide (UU)
Arguing to Motivate Decisions

2011-34 Paolo Turrini (UU)
Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations

2011-35 Maaike Harbers (UU)
Explaining Agent Behavior in Virtual Training

2011-36 Erik van der Spek (UU)
Experiments in serious game design: a cognitive approach
2011-37 Adriana Burlutiu (RUN)
Machine Learning for Pairwise Data, Applications for Preference Learning
and Supervised Network Inference

2011-38 Nyree Lemmens (UM)
Bee-inspired Distributed Optimization

2011-39 Joost Westra (UU)
Organizing Adaptation using Agents in Serious Games

2011-40 Viktor Clerc (VU)
Architectural Knowledge Management in Global Software Development

2011-41 Luan Ibraimi (UT)
Cryptographically Enforced Distributed Data Access Control

2011-42 Michal Sindlar (UU)
Explaining Behavior through Mental State Attribution
2011-43 Henk van der Schuur (UU)
Process Improvement through Software Operation Knowledge
2011-44 Boris Reuderink (UT)
Robust Brain-Computer Interfaces
2011-45 Herman Stehouwer (UvT)
Statistical Language Models for Alternative Sequence Selection

2011-46 Beibei Hu (TUD)
Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work

2011-47 Azizi Bin Ab Aziz(VU)
Exploring Computational Models for Intelligent Support of Persons with Depression

2011-48 Mark Ter Maat (UT)
Response Selection and Turn-taking for a Sensitive Artificial Listening Agent
2011-49 Andreea Niculescu (UT)
Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality

====
2012
====

2012-01 Terry Kakeeto (UvT)
Relationship Marketing for SMEs in Uganda

2012-02 Muhammad Umair(VU)
Adaptivity, emotion, and Rationality in Human and Ambient Agent Models

2012-03 Adam Vanya (VU)
Supporting Architecture Evolution by Mining Software Repositories

2012-04 Jurriaan Souer (UU)
Development of Content Management System-based Web Applications

2012-05 Marijn Plomp (UU)
Maturing Interorganisational Information Systems

2012-06 Wolfgang Reinhardt (OU)
Awareness Support for Knowledge Workers in Research Networks

2012-07 Rianne van Lambalgen (VU)
When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions

2012-08 Gerben de Vries (UVA)
Kernel Methods for Vessel Trajectories

2012-09 Ricardo Neisse (UT)
Trust and Privacy Management Support for Context-Aware Service Platforms

2012-10 David Smits (TUE)
Towards a Generic Distributed Adaptive Hypermedia Environment

2012-11 J.C.B. Rantham Prabhakara (TUE)
Process Mining in the Large: Preprocessing, Discovery, and Diagnostics
2012-12 Kees van der Sluijs (TUE)
Model Driven Design and Data Integration in Semantic Web Information Systems

2012-13 Suleman Shahid (UvT)
Fun and Face: Exploring non-verbal expressions of emotion during playful interactions

2012-14
Evgeny Knutov(TUE)
Generic Adaptation Framework for Unifying Adaptive Web-based Systems

2012-15 Natalie van der Wal (VU)
Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes

2012-16 Fiemke Both (VU)

Helping people by understanding them - Ambient Agents supporting task execution and depression treatment

2012-17 Amal Elgammal (UvT)
Towards a Comprehensive Framework for Business Process Compliance

2012-18 Eltjo Poort (VU)
Improving Solution Architecting Practices

2012-19 Helen Schonenberg (TUE)
What's Next? Operational Support for Business Process Execution

2012-20 Ali Bahramisharif (RUN)
Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing

2012-21 Roberto Cornacchia (TUD)
Querying Sparse Matrices for Information Retrieval

2012-22 Thijs Vis (UvT)
Intelligence, politie en veiligheidsdienst: verenigbare grootheden?

2012-23 Christian Muehl (UT)
Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction

====
2013
====

2013-01 Viorel Milea (EUR)
News Analytics for Financial Decision Support

2013-02 Erietta Liarou (CWI)
MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing

2013-03 Szymon Klarman (VU)
Reasoning with Contexts in Description Logics