Autonomous Scheduling with Unbounded and Bounded Agents

Chetan Yadati[1], Cees Witteveen[1], Yingqian Zhang[1], Mengxiao Wu[2], and Han la Poutre[2]

[1] Delft University of Technology, Delft
[2] Centrum voor Wiskunde en Informatica, Amsterdam

Abstract. Autonomous scheduling deals with the problem - how to enable agents to schedule a set of interdependent tasks in such a way that whatever schedule they choose for their tasks, the individual schedules always can be merged into a global feasible schedule? Unlike the traditional approaches to distributed scheduling we do not enforce a fixed schedule to every participating agent. Instead we guarantee flexibility by offering a set of schedules to choose from in such a way that every agent can choose its own schedule independently from the others. We show that in case of agents with unbounded concurrency, optimal make-span can be guaranteed. Whenever the agents have bounded concurrency optimality cannot be guaranteed, but we present an approximation algorithm that ensures a constant make-span ratio.

Keywords: Scheduling, autonomous agents, flexible scheduling, algorithm, make-span optimality.

1 Introduction

Autonomous scheduling aims to provide autonomous agents with a set of minimal constraints on tasks such that each agent is able to make a schedule for its tasks independently from the others. This independent scheduling capability should hold even if the tasks of an agent are dependent upon the completion of tasks given to other agents. The construction of such constraints can be viewed as the result of a coordination mechanism that ensures the existence of a joint feasible schedule whenever each of the individual agent's schedules meets its constraints. In particular, such coordination mechanisms are useful whenever a set of tasks has to be completed by a number of autonomous agents who are not willing, or not able, to communicate and negotiate about the schedules for the tasks they have to process.

Autonomous scheduling problems differ from classical (distributed) scheduling problems in the sense that besides efficiency criteria like minimal makespan, also *flexibility* criteria that aim to maximize the freedom of scheduling choice (autonomy) of the participating agents play a role: Instead of forcing each agent to comply to a given schedule, autonomous scheduling offers them a choice from a *set* of schedules for the tasks proposed.

R. Bergmann et al. (Eds.): MATES 2008, LNAI 5244, pp. 195–206, 2008.

To represent such a set of possible schedules, in autonomous scheduling a set of (time) constraints for each of the tasks is constructed in such a way that each agent can choose a schedule itself, provided that it satisfies all the task constraints. Of course, such a set of constraints should be such that *(i)* whatever schedule is chosen by an agent, it never interferes with the choice made by other agents and *(ii)* the set of constraints should be *maximally flexible*, that is, it should not be possible to find a weakening of the original constraints that also satisfies the first condition. If a set C of constraints satisfies both conditions we say that it meets the criterion of *maximal flexibility*. Besides flexibility to meet the needs of autonomous agents, we are also interested in *efficiency* as a system value: In order to get the total set of tasks done by the agents, we would like to minimize the total *makespan*. One of the questions then to be answered is: Can we design a flexible makespan efficient autonomous scheduling method?

In this paper, we show that there exists a surprisingly simple makespan efficient autonomous scheduling algorithm, provided that the agents are capable to process as much tasks concurrently as possible, i.e., they have unbounded concurrent capacity. Often, however, this might seem quite unrealistic. Hence, we adapt the method to accommodate for such bounded concurrency requirements of agents. In particular, we consider the case where agents are strictly sequential. We then prove that in this latter case designing a makespan-efficient autonomous scheduling method is NP-hard. The good news, however, is that there exist good approximation algorithms for makespan efficient autonomous sequential scheduling, if we allow the tasks to be processed in a preemptive way.

The structure of this paper is as follows: In the next section, we first provide some background on distributed and autonomous scheduling. Then, the basic framework which we use to describe problem instances is presented. In Section 4, we develop an algorithm called the ISA for distributed scheduling for tasks with homogeneous durations and unbounded capacity. In Section 5, we deal with sequential agents. We summarize our findings, conclude and point to future directions for research in Section 6.

2 Background and Related Work

Distributed scheduling has been an active area for research in the past decade. Roughly speaking, one can distinguish between approaches that assume that the participating systems, or agents controlling these systems, are cooperative and approaches that assume that the participating agents/systems are non-cooperative. Examples of the former (classical) approaches are DLS [1], HEFT [2], CPOP [2], ILHA [3] and PCT [4]. All these approaches mainly focus on optimizing some performance criteria such as makespan and required communication between processors. Typically, these approaches assume that the participating systems are *(i)* fully cooperative in *(ii)* establishing a single globally feasible schedule for the complete set of tasks.

In quite a lot of applications, however, we simply cannot assume that the participating systems are fully cooperative. For example, in grid applications, jobs

often have to compete for CPU and network resources, and each agent is mainly interested in maximizing its own throughput instead of maximizing the global throughput. Thus, several researchers have adopted a game-theoretic approach for solving scheduling problems with non-cooperative agents. For example, Walsh *et al.* [5] use auction mechanisms to arbitrate resource conflicts for scheduling network access to programs for various users on the internet. Another approach to non-cooperative scheduling is based on negotiation. Recently, Li in his PhD thesis developed both static and learning based dynamic negotiation models for grid scheduling [6].

In both approaches to non-cooperative scheduling, however, the effort is directed at developing a *single global* centrally computed schedule that meets some criterion. In situations where agents are exploring unknown or hostile territory, it might be very restrictive or even impossible to enforce rigid schedules on the agents. In other situations where agents participate in more than one such system, they would require a minimum set of constraints on their schedules rather than a single rigid schedule. Recently, Hunsberger [7] has developed a temporal decoupling method for Simple Temporal Networks (STNs) to decompose an STN into a number of independent sub-STNs, each of which can be scheduled independently from the others. Although the autonomous scheduling method we will apply is related to his decoupling method, we want to design such independent subnetworks directly from a given set of simple constraints. Moreover, even if such a common temporal network would exist, unlike in the temporal decoupling method, we would allow to modify one or more of its constraints in the decomposition process as, e.g., we will do in the sequential scheduling case. Finally, the principal aim of our method is to provide a decomposition method where flexibility as well as efficiency criteria play an important role.

3 Preliminaries and Framework

We assume a finite set of tasks/operations $T = \{t_1, \ldots, t_m\}$, each of which takes finite time $d(t_i) \in Z^+$ for processing. Furthermore, these tasks are interrelated by a partially ordered precedence relation $\prec$, where $t_i \prec t_j$ indicates that t_i must be completed before t_j can start. We use the transitive reduction $\ll$ of $\prec$ to indicate the immediate precedence relation between tasks, i.e., $t \ll t'$ iff $t \prec t'$ and there exists no t'' such that $t \prec t''$ and $t'' \prec t'$. We use a directed acyclic graph (DAG) $G = (T, \ll)$ to represent the task structure of T.

We assume that T has been assigned to a set of autonomous agents $A = \{A_1, \ldots, A_n\}$ according to a pre-defined task allocation $\phi : T \to A$. We denote the set of tasks allocated to agent A_i by $T_i = \phi^{-1}(A_i)$ Note that $\{T_i\}_{i=1}^n$ is a partitioning of T. Likewise, the precedence relation $\prec_i$ is the precedence relation $\prec$ induced by T_i and $d_i()$ is the duration function restricted to T_i. We also assume that there is a function $c : \{1, 2, \ldots n\} \to Z^+ \cup \{\infty\}$ 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. We say that $\langle \{T_i\}_{i=1}^n, \prec, c(), d() \rangle$ is a *scheduling instance*.

Given such a scheduling instance $\langle \{T_i\}_{i=1}^n, \prec, c(), d() \rangle$, a global schedule for it is a function $\sigma : T \rightarrow Z^+$ determining the starting time $\sigma(t)$ for each task $t \in T$. Obviously, to be a feasible solution, σ should satisfy the following constraints:

1. for every pair $t, t' \in T$, if $t \prec t'$, then $\sigma(t) + d(t) \leq \sigma(t')$;
2. for every $i = 1, \ldots, n$ and for every $\tau \in Z^+$, $|\{t \in T_i \mid \tau \in [\sigma(t), \sigma(t) + d(t)]\}| \leq c(i)$, that is the concurrency bounds for every agent A_i should be respected.

Of course, we prefer a schedule σ which minimizes *makespan*, i.e., among all feasible schedules σ', we prefer a schedule σ such that $max_{t \in T}\{\sigma(t) + d(t)\} \leq max_{t \in T}\{\sigma'(t) + d(t)\}$. Since we are dealing with autonomous agents, we would like to offer them the possibility to choose a most adequate schedule from a set of feasible schedules in such a way that their individual choices do not interfere. More in particular, we want to guarantee that a feasible global schedule σ can be obtained by imposing upon each agent a (minimal) set of additional constraints C_i, such that if C_i is specified for the scheduling instance $\langle T_i, \prec_i, d_i(), c(i) \rangle$ 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. More precisely, for each $i = 1, 2, \ldots, n$, let Σ_i be the set of all locally feasible schedules σ_i that satisfy C_i. Given any locally feasible schedule $\sigma_i \in \Sigma_i$ of any agent $A_i \in A$, we require the merging $\sigma = \bigcup_{i=1}^n \sigma_i$ to be a globally feasible schedule for $\langle \{T_i\}_{i=1}^n, \prec, c(), d() \rangle$.

In this paper, the set of constraints C_i that we will add to each local scheduling instance for each agent A_i, is a set of *time intervals* $[lb(t), ub(t)]$ for the tasks in T_i, where each interval $[lb(t), ub(t)] \in C_i$ with $lb(t), ub(t) \in Z^+$ specifies that any individual schedule σ_i agent A_i might choose has to satisfy the constraint $lb(t) \leq \sigma_i(t) \leq ub(t)$.

Example 1. Consider a simple example shown in Figure 1, where there are three agents and 7 tasks with precedence constraints. Here, a direct precedence constraint $t \ll t'$ is represented as an arrow from t pointing to t'. The task durations $d(t)$ are indicated below the circles representing the tasks t. Suppose that each agent can perform two tasks simultaneously, that is $c(1) = c(2) = c(3) = 2$. Clearly, the minimal makespan of processing these tasks is 11. To achieve this minimal makespan, the following schedules for the agents are possible: $\sigma_1(t_1) = \sigma_1(t_2) = 0$; $\sigma_2(t_3) = 2; \sigma_2(t_4) = 3$ and $\sigma_3(t_5) = 3; \sigma_3(t_6) = 7; \sigma_3(t_7) = 9$. However, prescribing these schedules is unnecessarily restrictive to the agents. In fact, several other schedules also result in the same global makespan. For example ,A_1 could start task t_1 in the interval [0,2] while starting task t_2 starting in the interval $[0, 0]$. Agent A_2 can process tasks t_3 in the interval $[4, 7]$ while starting t_3 in $[2, 2]$. Similarly, agent A_3 can process task t_5 in the interval [8,10] with task t_6 starting in the interval $[7, 7]$ and task t_7 in $[9, 9]$. Notice here that any schedule produced by the agents such that these intervals are honoured will always lead to a global makespan of 11. Thus, by introducing the additional constraints in the form of these intervals, agents have some amount of flexibility on deciding their local schedule without affecting the minimal makespan. ■

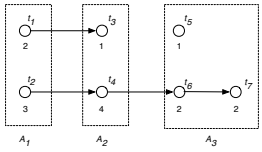

Fig. 1. A set of tasks t_i with their durations $d(t_i)$ to demonstrate the possibility of developing several makespan optimal schedules. Task durations are indicated as numbers within the circles representing the tasks.

Our goal is to design a minimal set of constraints C_i for each agent A_i, such that the merging of individual schedules σ_i that satisfy C_i always is a globally feasible schedule. Moreover, we would like the merging to be also makespan efficient. In the next sections, we consider two scenarios: the first, where agents can perform an unlimited number of tasks simultaneously and the second where they can perform only a single task at any given point in time.

4 Autonomous Scheduling for Agents with Unbounded Concurrency

In this section, we discuss a simple method to specify additional constraints for each agent in order to guarantee that individual schedules σ_i meeting these constraints can always be merged into a global *makespan efficient* schedule. The central idea is to specify as the constraint for a task t an interval consisting of its earliest possible starting time and its latest possible starting time, taking into account the precedence constraints between the tasks and their duration. Once these intervals are computed, the agents can autonomously create any local schedule provided that it satisfies these intervals. Thus, they are offered the flexibility of creating more than one schedule but still be assured that the global makespan is optimal.

The idea of using these intervals is related to the way in which the CPOP [2] algorithm by Topcuouglu *et al.* computes the priority of a task for scheduling on a machine. In CPOP, a combined value of the *depth* and the *height* of a task is used to compute the priority. We build upon this idea and compute intervals instead of priorities, within which each agent/machine is free to schedule its tasks.

To define the interval $[lb(t), ub(t)]$ for the starting time of a task t in the given partial order $\langle T, \prec, d() \rangle$, we first compute, for each task $t \in T$ its $depth(t)$ and its $height(t)$. We will need both measures to determine the earliest and the latest possible time at which a task can be started. To aid in our computation of the depth and height of a task t, we further define two sets: $pred(t) = \{t' |\ t' \ll t\}$ and $succ(t) = \{t' |\ t \ll t'\}$.

The depth of a task t is defined as follows: $depth(t) = 0$ if $pred(t) = \emptyset$ and $depth(t) = max_{t' \in pred(t)}\{depth(t') + d(t')\}$, otherwise. Note that the depth of a task t is the maximum duration of any chain of tasks preceding it, hence it directly determines the earliest time task t might start. The depth $depth(T)$ of the set of

tasks T is defined as the maximum duration required to complete all tasks taking into account the precedence relation $\prec$: $depth(T) = \max_{t \in T}\{depth(t) + d(t)\}$. So $depth(T)$ defines the minimal makespan of T.

The height $height(t)$ of a task t in a partial order $\langle T, \prec, d()\rangle$ defines the time that has to pass before all tasks occurring after t and including t have been completed: So $height(t) = d(t)$, if $succ(t) = \emptyset$ and $height(t) = max_{t' \in succ(t)}\{height(t') + d(t)\}$, otherwise. From the specifications of $depth(t), height(t)$ and $depth(T)$ the earliest ($lb(t)$) and latest ($ub(t)$) possible starting times for a task t can be derived as follows: $lb(t) = depth(t)$ and $ub(t) = depth(T) - height(t)$.

These intervals $[lb(t), ub(t)]$, however, are still not directly usable for autonomous scheduling. Since the length of task chains might differ, it can easily happen that the intervals of some precedence constrained tasks $t \prec t'$ might *overlap*, that is $lb(t') < ub(t) + d(t)$, while $t \prec t'$. Such an overlap might cause a violation of the first constraint on the joint schedule, namely that $t \prec t'$ should imply $\sigma(t) + d(t) < \sigma(t')$.

Example 2. Consider the set of tasks given in Figure 1. The depth of T is $depth(T) = 11$. Computing the depths and the heights of the tasks t_1 and t_3, we derive the constraints $C(t_1) = [0, 7]$ and $C(t_3) = [2, 9]$. Now, agent A_1 could decide to start t_1 at time $\sigma_1(t_1) = 5$, while A_2 could choose $\sigma_2(t_3) = 6$. However, if these schedules are merged, we violate the precedence constraint between t_1 and t_3 since then $\sigma(t_3) < \sigma(t_1) + 2$. ■

This implies that, in case of overlap, the agents are not free to choose any point in the interval $[lb(t), ub(t)]$ as the starting point $\sigma(t)$ for t. Therefore, we have to remove such overlap as depicted in Figure 2: In order to satisfy the scheduling constraint we should ensure that whenever $t \prec t'$, the difference between $lb(t')$ and $ub(t)$ should be at least $d(t)$. Note that, in case of an overlap between two tasks $t \prec t'$, it is always possible to remove the overlap without creating empty intervals: If $t \prec t'$, we have $lb(t) + d(t) \leq lb(t')$ and $ub(t) + d(t) \leq ub(t')$. The existence of an overlap implies that $lb(t) < lb(t') < ub(t) + d(t)$. Hence, $ub(t') - lb(t) \geq ub(t) + d(t) - (lb(t') - d(t)) > -d(t) + 2d(t) = d(t)$.

Since we require $lb(t') - ub(t) \geq d(t)$, we set $ub(t) = lb(t) + \lfloor \frac{ub(t') - lb(t) - d(t)}{2} \rfloor$ and thereafter $lb(t') = lb(t) + \lfloor \frac{ub(t') - lb(t) - d(t)}{2} \rfloor + d(t)$. In both cases, since $ub(t') - lb(t) > d(t)$, the new constraint intervals are non-empty. The complete description of the algorithm is given in the ISA [1] Algorithm (see Algorithm 1).

Example 3. Consider again the set of tasks given in Figure 1. The "raw" constraints on the tasks are computed as $C(t_1) = [0, 7]$, $C(t_3) = [2, 9]$, $C(t_5) = [4, 10]$ while $C(t_2) = [0, 0]$, $C(t_4) = [3, 3]$, $C(t_6) = [7, 7]$ and $C(t_7) = [9, 9]$. Since there is overlap between the constraints of task t_1 and t_3 and t_3 and t_5, these constraints have to be adapted. The result is the following set of constraints: $C(t_1) = [0, 3]$, $C(t_3) = [5, 7]$ and $C(t_5) = [8, 10]$. It is easily verifiable that all local schedules that adhere to their constraints are feasible and also that all such local schedules can be combined to obtain a global schedule that is correct and has a makespan of 11. ■

[1] Interval-based Scheduling Algorithm.

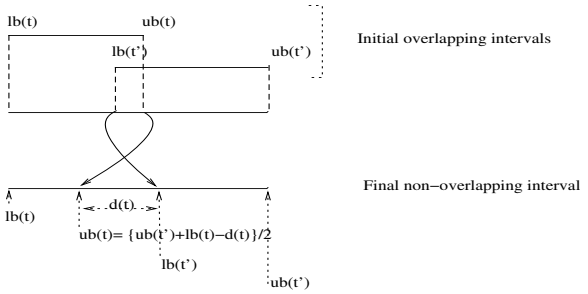

Fig. 2. Overlapping interval splitting procedure in ISA

Algorithm 1. Generalised Interval based Scheduling (ISA)

Require: Partially ordered set of tasks $(T, \prec)$, for every task $t \in T$ its depth $depth(t)$ and its height $height(t)$;

Ensure: For every $t \in T$ its scheduling interval $C(t) = [lb(t), ub(t)]$;

1: $depth(T) := max_{t \in T}\{depth(t) + d(t)\}$
2: **for all** $t \in T$ **do**
3: $\quad lb(t) := depth(t)$ and $ub(t) := depth(T) - height(t)$
4: **end for**
5: **for all** $t, t' \in T$ such that $t \prec t'$ and $lb(t') - ub(t) < d(t)$ **do**
6: $\quad ub(t) = lb(t) + \lfloor \frac{ub(t') - lb(t) - d(t)}{2} \rfloor$
7: $\quad lb(t') = ub(t) + d(t)$
8: **end for**
9: return $C(t) = [lb(t), ub(t)]$ for all $t \in T$

Obviously, this algorithm runs in polynomial time. Note that each interval $[lb(t), ub(t)]$ computed by ISA is always non-empty and for every $t \in T$, $ub(t) \leq depth(T) - d(t)$. Moreover, for every pair t, t' of tasks, $t \prec t'$ implies $ub(t) + d(t) < lb(t')$. Hence, it is not difficult to see that *(i)* every local schedule σ_i satisfying the local constraints C_i will be a feasible schedule for the set of tasks T_i and *(ii)* the merge of every set $\{\sigma_i\}_{i=1}^n$ of local schedules is a feasible global schedule, thus ensuring the correctness and make-span optimality of the algorithm:

Proposition 1. *The interval-based scheduling algorithm (*ISA*) ensures a correct global schedule and it is efficient in terms of makespan.*

With respect to flexibility of this autonomous scheduling method, it is not difficult to see that it satisfies maximal flexibility. Here, we call a set $C = \{C(t) \mid t \in T\}$ of interval constraints for a a scheduling instance $\langle \{T_i\}_{i=1}^n, \prec, c(), d() \rangle$ maximally flexible, if there does not exist any strict weakening[2] C' of C such that C' also allows for autonomous scheduling of $\langle \{T_i\}_{i=1}^n, \prec, c(), d() \rangle$ and is makespan efficient. This property can easily be proven by noticing that the ISA algorithm generates

[2] A set C' is a strict weakening of C if every schedule σ satisfying C also satisfies C' but not vice versa.

a set of constraints $C = \{C(t) \mid t \in T\}$ such that *(i)* for every task t such that $succ(t) = \emptyset$ we have $ub(t) = depth(T) - d(t)$; *(ii)* for every t such that $pred(t) = \emptyset$ it holds that $lb(t) = 0$; *(iii)* for every pair of tasks t, t' such that $t \prec t'$ it holds that $lb(t') = ub(t) + d(t)$.

This implies that any strict weakening C' of C would contain a constraint $C(t) = [lb(t), ub(t)]$ such that either *(a)* $lb(t) < 0$ or *(b)* $ub(t) > depth(T) - d(t)$ or *(c)* there exists some t' such that $t \prec t'$ and $ub(t) + d(t) > lb(t')$ or *(d)* there exists some t' such that $t' \prec t$ and $ub(t') + d(t) > lb(t)$. Clearly, case *(a)* and *(b)* would imply that some C'-satisfying schedules are not make span efficient and if case c or d holds, some C'-satisfying schedules violate a precedence constraint. Hence, such a weakening cannot exist and we have the following proposition:

Proposition 2. *Any strict weakening the set C of constraints imposed by* ISA *either leads to a infeasible schedule or leads to a non optimal makespan.*

Summarizing, we have the following property:

Theorem 1. ISA *ensures a maximally flexible set of constraints and a makespan efficient global schedule.*

Note, however, that the minimal global makespan is ensured by the proposed algorithm ISA only under the assumption that the participant agents have capabilities to perform a potentially unbounded number of tasks at the same time. Often, this assumption is not realistic as agents may only have limited resources at their disposal. Therefore, in the next section, we study the case when every agent is capable of performing only a single task at any point in time (sequential agents).

5 Scheduling Sequential Agents

A scheduling instance $\langle \{T_i\}_{i=1}^n, \prec, c(), d() \rangle$ where $c(i) = 1$ for every A_i is called a sequential scheduling instance, abbreviated as $\langle \{T_i\}_{i=1}^n, \prec, 1, d() \rangle$. Like in the unbounded case, we would like to come up with a set C of constraints $C(t) = [lb(t), ub(t)]$ for each task $t \in T$ such that the agents are able to construct their *sequential* schedule independently from the others. Any individual schedule σ_i for a sequential agent A_i with the set of tasks T_i assigned to it, has to satisfy the following conditions:

- $lb(t) \leq \sigma_i(t) \leq ub(t)$ for every $t \in T_i$ where $C(t) = [lb(t), ub(t)]$;
- for every $t, t' \in T_i$, $t \neq t'$ implies $\sigma_i(t) - \sigma_i(t') \geq d(t)$ or $\sigma_i(t') - \sigma_i(t) \geq d(t')$.

While designing such constraints for autonomous scheduling if the agents are unbounded turns out to be a feasible problem, the equivalent problem for sequential agents turns out to be infeasible, mainly because we cannot ensure that, based on a given set of constraints delivered to the individual agents, they are able to find a sequential schedule satisfying all the constraints. More precisely, while in the unbounded case we were able to find a minimum makespan M for the total set of tasks and could guarantee that given a set C of additional task constraints

any set $\{\sigma_i\}_{i=1}^n$ of locally feasible schedules would result in a makespan M complying global schedule, finding such a makespan complying schedule in the sequential case is an intractable problem:

Proposition 3. *Given a sequential scheduling instance $\langle\{T_i\}_{i=1}^n, \prec, 1, d()\rangle$ and a positive integer M, the problem to decide whether there exists a set of constraints C such that the scheduling instance allows for a solution with makespan M by autonomous scheduling is NP-hard.*

Proof. We reduce the PARTITIONING problem [8] (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 autonomous scheduling for sequential agents problem.

Take an instance S of PARTITIONING and let $d_S = \sum_{s\in S} s$. Without loss of generality, we can assume d_S to be even. Consider the following set of tasks $T = \{t_s | s \in S\} \cup \{t_a, t_b, t_c\}$. For every task $t_s \in T$, let $d(t_s) = s$, let $d(t_a) = \frac{d_S}{2}$, let $d(t_b) = 1$, $d(t_c) = \frac{d_S}{2} + 1\}$ and let $\prec = \{(t_a \prec t_b), (t_b \prec t_c)\}$. Furthermore, there are two agents A_1 and A_2, where A_1 has to perform the tasks t_a and t_b and agent A_2 has to perform all the remaining tasks $(T - \{t_a, t_b\})$. Finally, let $M = d_S + 1$.

If the agents are sequential, there exists a set of 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_2 is able to process one subset of its set of tasks in the interval $[0, d(t_a)]$, starts t_b in the interval $[d(t_a), d(t_a)]$ and completes the remaining subset of tasks in the interval $[d(t_a) + 1, d_S + 1]$. □

Note that the complexity is not dependent upon the number of agents: Already two agents suffice to render the problem hard and, in particular, the problem derives its hardness from the difficulty to determine for a *single* agent the set of constraints that would allow it to determine its own schedule without violating the global makespan.

Therefore, we have to rely on approximation algorithms for autonomous scheduling in the sequential agent case. As we have shown in some recent work [9], there exists a polynomial 2-approximation algorithm for constructing a set of constraints in the sequential agent scheduling case if all the tasks $t \in T$ have unit durations $d(t) = 1$. This algorithm constructs a maximally flexible set of constraints guaranteeing that the resulting global makespan is never more than twice the optimal makespan that can be realized by sequential scheduling agents. We will briefly discuss the outlines of this algorithm and reuse it (after some adaptations) to the general sequential agent scheduling case.

The basic idea in this algorithm is to first use the ISA algorithm to determine the set of constraints C for the unit duration tasks. As we have shown above, if the agents would be able to handle tasks concurrently, an agent would be able to find a schedule satisfying all the constraints. In the sequential agent scheduling case, this might not be possible. For example, if there are three unrelated tasks t_1, t_2 and t_3 of unit duration, where the first two are given to agent A_1 and the third to agent A_2, agent A_1 is not able to schedule both tasks given the constraints

$C(t_1) = C(t_2) = [0, 0]$. There is, however, a simple way to tell whether a given agent is able to find a sequential schedule for all tasks $t \in T_i$ with the constraints $C(t)$ given to it: Consider the bipartite graph $G_i = (T_i \cup N_i, E_i)$ where N_i is the set of all time points occurring in the intervals $C(t) = [lb(t), ub(t)]$ of tasks $t \in T_i$ and $(t, n) \in E_i$ iff $n \in C(t)$.[3] It is not difficult to see that there exists a sequential schedule for agent A_i iff the graph G_i has a *maximum matching* [10] that includes every task $t \in T_i$.

If the (polynomial) maximum matching algorithm is not able to find a complete matching for T_i, i.e., some of the tasks could not be scheduled, there must be a *scheduling conflict* between a task t in the matching and a task t' not in the matching. Such a conflict can be resolved by adding a precedence constraint $t \prec t'$ between t and t' and calling the ISA algorithm again on the extended scheduling instance. Note that the result of such extensions of the precedence relation is twofold *(i)* the conflict between t and t' is removed and *(ii)* the global makespan $d(T)$ might be increased.

Continuing our last example mentioned above: consider the two tasks t_1 and t_2 agent A_1 was not able to handle. A maximum matching for G_1 contains either t_1 or t_2. If we add a precedence constraint $t_1 \prec t_2$ to the set of tasks, agent A_1 will receive the constraints $C(t_1) = [0, 0]$ and $C(t_2) = [1, 1]$ is able to find a suitable sequential schedule for its set of tasks.

This matching, extending the precedence relation, and calling the ISA algorithm is repeated until we are guaranteed that for each agent there exists at least one sequential schedule.[4] The result is a set of constraints C guaranteeing that any schedule resulting from independently chosen schedules realizes a makespan that is at most twice as long as the optimal makespan. One of the attractive features of the unit-duration sequential scheduling case is the existence of a polynomial decision procedure (the maximum matching algorithm) for deciding whether there exists a sequential schedule satisfying the constraints $C_i(t)$ for an agent A_i. The reduction from PARTITION given above shows that, unless P=NP, we cannot hope to find a solution for the same problem in the general sequential scheduling case.

There is, however, a possibility to reuse the approximation algorithm sketched above if we assume that, although the agents are strictly sequential, the tasks can be accomplished using *preemption*. This enables an agent to complete a part of task t, then to start some other tasks, process a next part of t and so on. If this is allowed, we can easily reduce a sequential scheduling instance $\langle \{T_i\}_{i=1}^n, \prec, 1, d() \rangle$ to a sequential scheduling instance with unit durations as follows:

Each task $t \in T$ is split in unit parts $t^1, \ldots, t^{d(t)}$, we add the constraints $t^j \prec t^{j+1}$ for $j = 1, \ldots, d(t) - 1$. Finally, every precedence constraint $t \prec t'$ is replaced by the constraint $t^{d(t)} \prec t'^1$. See Figure 3 for an illustration. Note that this assumption implies that the sequential scheduling case with arbitrary task durations can be reduced to the unit duration sequential case. Hence, we can reuse the approximation algorithm for this case, too, obtaining a 2-approximation al-

[3] Note that we assume integer values for schedules $\sigma(t)$.

[4] This procedure must halt because conflicts can never reoccur.

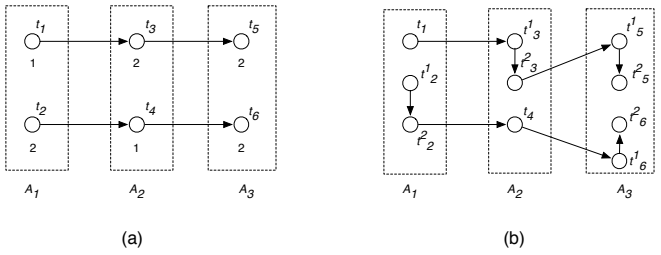

Fig. 3. A sequential scheduling instance (a) and its reduction to the equivalent unit-duration case (b) if preemption is allowed

gorithm for autonomous scheduling of tasks with arbitrary durations. There is of course a catch: The approximation algorithm is only polynomial for those instances where the durations $d(t)$ are not super-polynomial in the number of tasks $|T|$. Otherwise, the splitting of tasks in unit duration tasks would result in a super polynomial number of unit-duration tasks.

Example 4. Consider the problem instance in Figure 3. Here, the tasks are assigned to 3 agents $T_1 = \{t_1, t_2\}$, $T_2 = \{t_3, t_4\}$ and $T_3 = \{t_5, t_6\}$ with task durations $d(t_1) = d(t_4) = 1$ and $d(t_2) = d(t_3) = d(t_5) = d(t_6) = 2$. The precedence constraints are $t_1 \prec t_3 \prec t_5$ and $t_2 \prec t_4 \prec t_6$. The minimal makespan for the unbounded case is $d(T) = 5$. This implies that after task splitting of task t_2, agent A_1 will receive the constraints $C(t_1) = C(t_{2a}) = [0, 0]$ and $C(t_{2b}) = [1, 1]$. If t_1 is included in the maximum matching (and t_{2a} is not), an additional constraint $t_1 \prec t_2 a$ is added to the set of tasks. As a result, the depth $depth(T) = 6$ and the new constraints are $C(t_1) = [0, 0]$, $C(t_2^1) = [1, 1]$ and $C(t_2^2) = [2, 2]$. Continuing the procedure with agent A_2 and agent A_3 in the same way will result in a joint schedule with makespan 7, while the optimal makespan for sequential scheduling agents is 6. ■

6 Conclusions and Future Work

In this paper, we studied the distributed scheduling problem in the context of non-cooperative agents. Unlike traditional scheduling algorithms, which generate a single rigid schedules for the agents, our algorithms introduce constraints on the starting time of each task. In this way, we allow flexibility for each agent to choose among a set of allowable ones, while ensuring that the combined global schedule is feasible. We believe that for many real-world applications, specially those with uncertainty, such flexible scheduling for individual agents is highly desirable.

We have developed a polynomial time algorithm - ISA that generates a set of constraints on the possible schedules the agents can develop so that, any schedule that abides by these additional constraints is always feasible. The schedules generated by ISA also were proved to have maximal flexible and makespan efficient. We have also shown that in the the sequential case, it is NP-hard to find

a makespan optimal schedule. We then showed that preemptive task processing allowed us to reuse an approximation algorithm developed for sequential scheduling of unit duration tasks. The makespan of the schedules resulting out of using this approximation algorithm are at most twice that of the optimal.

As a follow-up of this paper first of all, we would like to use approximation algorithms for the general bounded agent case without relying on the preemptive task processing assumption and without restricting attention to the strictly sequential case. Next, we plan to formally analyze the minimal degree of flexibility that the proposed algorithms can ensure given different task structures. Furthermore, we would also like to investigate the trade-off between the degree of flexibility and the loss of the makespan efficiency. Such a study would enable us to design algorithms that are better customized to specific applications.

References

1. Sih, G.C., Lee, E.A.: Scheduling to account for interprocessor communication within interconnection-constrained processor networks. In: ICPP (1), pp. 9–16 (1990)
2. Topcuouglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. 13(3), 260–274 (2002)
3. Beaumont, O., Boudet, V., Robert, Y.: The iso-level scheduling heuristic for heterogeneous processors. In: Proc. 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing 2002, 9-11 Jan. 2002, pp. 335–350 (2002)
4. Maheswaran, M., Siegel, H.J.: A dynamic matching and scheduling algorithm for heterogeneous computing systems. In: HCW 1998: Proc. of the Seventh HCW, p. 57 (1998)
5. Walsh, W., Wellman, M., Wurman, P., MacKie-Mason, J.: Some economics of market-based distributed scheduling. In: Proceedings of 18th International Conference on Distributed Computing Systems 1998, 26-29 May 1998, pp. 612–621 (1998)
6. Li, J.: Strategic Negotiation Models for Grid Scheduling. PhD thesis, TU Dortmund (2007)
7. Hunsberger, L.: Algorithms for a temporal decoupling problem in multi-agent planning. In: Proc. of AAMAS-2003 (2002)
8. Garey, M., Johnson, D.: Computers and Intractability - a guide to the theory of NP-completeness. W.H. Freeman and Company, New York (1979)
9. Yadati, C., Witteveen, C., Zhang, Y., Wu, M., Putre, H.L.: Autonomous scheduling. In: Proc. of the FCS 2008 (2008)
10. Cormen, T.T., Leiserson, C.E., Rivest, R.L.: Introduction to algorithms. MIT Press, Cambridge (1990)