Autonomous Scheduling

Submitted to FCS 2008

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

***Abstract*— We discuss the problem of enabling agents to schedule a set of interdependent tasks in such a way that whatever schedule they choose the individual schedules always can be merged into a global schedule. Unlike the traditional approaches to distributed scheduling we do not enforce a schedule to every participating agent, but a set of additional constraints guaranteeing that every agent can choose its own schedule when it satisfies those constraints. 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 some heuristics that ensure a constant make-span ratio.**

Keywords: scheduling, autonomous agents, flexible scheduling, algorithm, make-span optimal

I. Introduction

Distributed scheduling has been an active area for research in the past decade. In general, one can distinguish two separate subproblems in distributed scheduling: an *allocation problem* involving tasks that need to be optimally to allocated to agents (factories, machines) and a corresponding *scheduling problem* for each of the agents. In this paper our focus will be on the second subproblem, assuming that the task allocation process has been solved. Focusing upon this subproblem, one can distinguish between two main approaches to distributed scheduling - the *cooperative* and the *non-cooperative*. Cooperative approaches assume that the participating systems or agents controlling these systems are cooperative and similarly, the non cooperative approaches assume that the participating agents/systems are non-cooperative. Examples of the former (more classical) approaches are DLS [1], HEFT [2], CPOP [2], ILHA [3] and PCT [4]. 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. Furthermore, they also attempt to optimize some performance criteria such as make-span, required communication between processors and so on.

In quite a lot of applications, however, we simply cannot assume that the participating systems are fully cooperative. For example, in grid applications, often jobs 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 all* [5] use auction mechanisms to arbitrate resource conflicts for scheduling network access to programs for various users on the internet and Wang and Xi [6] develop a two-layer optimization problem based on non-cooperative games to solve the problem when consumers have heterogeneous objectives.

A common thread in all the above algorithms (cooperative as well as non-cooperative), is the fact that (ii) they provide a single global schedule as their output and (ii) they assume that, even in the non-cooperative case, agents have a possibility to *communicate* with each other in order to establish such a globally feasible schedule. There are, however, quite some applications where these assumptions are difficult to meet. For example, in many military reconnaissance operations where several agents are involved, agents would like to have *flexible* instead of rigid schedules (to be able to cope with uncertain events), and are often not able to communicate with each other after tasks have been assigned to them. In such situations it would be important to provide each agent with a set of *constraints* for its individual schedule instead of a rigid schedule. These constraints should ensure that, whatever definite schedule σ_i is chosen by agent A_i at a later time, if σ_i meets these constraints, it is always part of a feasible global schedule.

Also in other more common situations, we often have to deal with agents that might participate in several other tasks they either don't want to provide information about or they do not have enough information about to come up with a definite schedule. Therefore, instead of choosing a schedule *now*, they would like to know what the *minimum* constraints are on a schedule they would choose *later* instead of being forced to adhere at one single schedule in advance.

A possible drawback, of course, of such an approach that aims to provide maximum *flexibility* to participating agents is that we could easily loose *global efficiency*. Therefore, in this paper we will attempt to develop methods where agents are given flexibility in scheduling their operations while still guaranteeing global efficiency. More in particular, in this paper, we deal with a set of precedence constrained tasks that have been assigned to agents and we propose methods that

- provide agents with a set of constraints for the schedules they are allowed to choose, while ensuring system-wide efficiencies if the agents have no concurrency bounds;
- provide agents with a set of scheduling constraints if they have limited capacities, more specifically, when agents can perform a single task at any given point in time, while still ensuring some performance guarantees.

[1] {c.yadati,c.witteveen,yingqian.zhang}@tudelft.nl, Phone: +31 15 2786206 Delft University of Technology, P.O. Box 5031, NL-2600 GA Delft, The Netherlands

[*] contact author

[2] {M.Wu,Han.La.Poutre}@cwi.nl, Centrum voor Wiskunde Informatica, P.O. Box 94079, NL-1090 GB Amsterdam,The Netherlands

In a sense, our approach can be seen as a generalization of the flow-shop scheduling case, where a set of precedence constrained operations have to be scheduled on a given set of machines. The generalization comes from the fact that we do not impose a rigid schedule on each of the machines (agents) but try to provide them with a (minimal) set of constraints.

We first introduce our framework in Section II. Section III proposes a method which defines a set of constraints on the tasks of each agent, assuming unbounded concurrency of the agents. The algorithm ensures the combined global schedule is optimal and leaves the agents maximal flexibility. We then study the case when the agents have bounded concurrency in Section IV. Since the problem of finding an optimal make-span schedule is NP-hard with bounded concurrency, we develop two polynomial-time heuristics and derive their worst case performance bound. We conclude and discuss some of our future work in Section V.

II. Preliminaries and Framework

We assume a finite set of tasks/operations $T = \{t_1, \ldots, t_m\}$, each of which takes a time equal to $d(t_i) \in Z^+$ to complete its processing. Furthermore, these tasks are interrelated by a partially ordered precedence relation $\prec$: $t_i \prec t_j$ states that t_i must be completed before t_j can start. We use a directed acyclic graph (DAG) $G = (T, \prec)$ to represent the task structure of T. We assume that T has been given to a set of autonomous agents $A = \{A_1, \ldots, A_n\}$ according to a pre-defined task allocation $\phi : T \rightarrow A$ inducing a partitioning $\{T_i\}_{i=1}^n$ of T, that is each task is assigned to one agent. We denote the set of tasks allocated to agent A_i by T_i. Likewise, the precedence relation $\prec_i$ is the precedence relation induced by T_i and $d_i()$ is the duration function restricted to T_i. We also assume that there is a function $c : \{1, 2, \ldots n\} \rightarrow Z^+ \cup \{\infty\}$ 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*.

We want to find a global schedule σ for a given scheduling instance $\langle \{T_i\}_{i=1}^n, \prec, c(), d() \rangle$. Such a schedule σ is a function $\sigma : T \rightarrow Z^+$ determining the starting time $\sigma(t)$ for each $t \in T$. Obviously, to be a feasible solution $\sigma()$ 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 should be respected.

We desire a schedule σ which minimizes minimal make-span, i.e., among all feasible schedules $\sigma'()$, $max_{t \in T}\{\sigma(t)\} \leq max_{t \in T}\{\sigma'(t)\}$.

More in particular, we want to guarantee that such a global schedule σ can be obtained by imposing each agent a (minimal) set of additional constraints C_i, such that if C_i is added to the scheduling instance $\langle T_i, \prec_i, d_i(), c(i) \rangle$ for agent A_i, then all locally feasible schedules $\sigma_i()$ satisfying their local scheduling instance $\langle T_i, \prec_i, d_i(), c(i) \rangle$ as well as their constraints C_i can be merged into a globally feasible schedule σ for the original scheduling instance. More precisely, for each $i = 1, 2, \ldots, n$, let Σ_i be the set of all locally feasible schedules for the tasks T_i of agent A_i that also satisfy C_i. Given any local schedule $\sigma_i \in \Sigma_i$ of any agent $A_i \in A$, the merged global schedule $\sigma = \cup_{i=1}^m \sigma_i$ must be feasible with respect to $\langle \{T_i\}_{i=1}^n, \prec, c(), d() \rangle$.

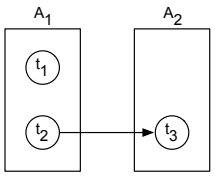

Fig. 1. An example of two agents with a precedence constraint between tasks t_2 and t_3 scheduling.

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

Example 1. Consider a simple example shown in Figure 1, where there are two agents $A = \{A_1, A_2\}$. The tasks assigned them are $T_1 = \{t_1, t_2\}$ and $T_2 = \{t_3\}$. There is a single precedence constraint: $t_2 \prec t_3$. Also, it is specified that all tasks take unit time for processing i.e. $d(t) = 1$ for all $t \in T$ and $c(1) = 2$ and $c(2) = 1$. If agents seek optimal make-span, then agent A_1 could either complete its tasks at time $\tau = 1$ or at time $\tau = 2$. Agent A_2 can start after one time unit required to process task t_2 and then complete its own task, task t_3 at time $\tau = 2$. However, allowing agent A_1 complete freedom would mean that it could complete task t_1 at time $\tau = 1$ and then task t_2 at time $\tau = 2$. But that would result in a violation of precedence constraint because agent A_2 also wants to complete its task at time $\tau = 2$. On the other hand, we could restrict agent A_1 to perform both t_1 and t_2 simultaneously in the first time unit and agent A_2 to perform t_3 in the second time unit, leading to a make-span of 2. However, this constraint is unnecessarily restrictive to the agent A_1. Notice that while task t_2 has to start at time $\tau = 0$, task t_1 can start anytime between $[0, 1]$, without affecting the make-span. Thus, by introducing the additional constraints $lb(t_1) = 0$ and $ub(t_1) = 1$, agent A_1 has some amount of flexibility on deciding its local schedule, as long as it satisfies these constraints, and the resulting global schedule that is combined always has the minimal make-span.

Our goal is to design a 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 feasible schedule that is also efficient in terms of make-span. We consider both the agents with unbounded concurrency and agents with bounded concurrency.

Unless otherwise mentioned, in this paper we assume that every task takes the same amount of time to complete and

without loss of generality, we assume that the duration $d(t)$ of any task is 1 (time unit). We start from the unbounded concurrency case and introduce an algorithm ensuring optimal autonomous scheduling in the next section.

III. Autonomous Scheduling for agents with unbounded concurrency

In this section, we discuss a simple algorithm to specify the additional constraints for each agent in order to guarantee that individual schedules σ_i meeting these constraints always can be merged into a global *make-span efficient* schedule. We name this algorithm the *Interval based Scheduling Algorithm* (or ISA). The central idea of ISA is to specify an earliest possible starting time and a latest possible starting time for each task $t \in T$, taking into account the precedence constraints between the tasks and their duration. Once the intervals are computed, the agents can autonomously create any local schedule provided that it satisfies these intervals. Thus, they offer the flexibility of creating more than one schedule but still be assured that the global make-span 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 $[lb(t), ub(t)]$ of the starting time of a task t in the given partial order $G = (T, \prec)$, we first compute, for each task $t \in T$ its depth $depth(t)$. The depth of a task t is defined as follows: $depth(t) = 1$ if $\neg\exists t' \in T[t' \prec t]$ and for all tasks t such that $\exists t'[t' \prec t]$ we define $depth(t) = 1 + max\{depth(t') \mid t' \prec t\}$.

By iteratively computing the depth of each task, we can determine the depth $depth(T)$ of the set of tasks, being the maximal depth of any task $t \in T$. Note that the depth of each task with unit duration per task determines the earliest possible starting time for each task. The depth $depth(T)$ of the set of tasks T therefore determines the maximum of all those earliest possible starting times.

The height $h(t)$ of a task in a partial order $(T, \prec)$ is defined as the depth $depth(t)$ in the *reversed* partial order $(T, \succ)$ where $t \prec t'$ if and only if $t' \succ t$. Hence, the quantity $depth(T) - h(t) + 1$ determines the latest possible starting time for task t, as $h(t)$ measures the maximum number of tasks that need to be executed after task t. The required interval $[lb(t), ub(t)]$ for task t now can be set as $lb(t) = depth(t)$ and $ub(t) = depth(T) - h(t) + 1$.

There is only one caveat: if the intervals of tasks t, t' related by $t \prec t'$ are not properly separated, then it might easily occur that their intervals $[lb(t), ub(t)]$ and $[lb(t'), ub(t')]$ have a non-empty overlap which, in turn might lead to the violation of a precedence constraint. It is easy to arrange for avoiding such overlaps because, we know that $lb(t) < ub(t)$ and $lb(t') < ub(t')$. Hence, we can easily split the intervals into two non-overlapping intervals.

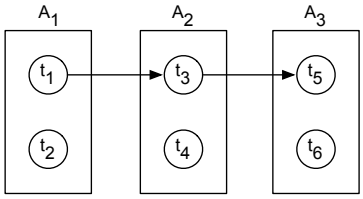

Fig. 2. Example problem instance.

A complete description of the algorithm is in Algorithm 1. The output of the ISA algorithm are the intervals $[lb(t), ub(t)]$ for all $t \in T$. The constraint set C_i for each agent A_i then is formed by collecting the intervals of all tasks in T_i thus obtained.

Algorithm 1 Interval based Scheduling Algorithm (ISA)

1: **Input:** Partially ordered set of tasks $(T, \prec)$, for every task $t \in T$ its depth $depth(t)$ and its height $h(t)$
2: **Output:** For every $t \in T$ its scheduling interval $[lb(t), ub(t)]$
3: **for all** $t \in T$ **do**
4: Set $lb(t) := depth(t)$
5: Set $ub(t) := depth(T) + 1 - h(t)$
6: **end for**
7: **for all** $t, t' \in T$ such that $t \prec t'$ and $[lb(t), ub(t)]$ overlaps with $[lb(t'), ub(t')]$ **do**
8: $ub(t) = lb(t) + \lfloor\{\frac{ub(t')-lb(t)}{2}\}\rfloor$
9: $lb(t') = ub(t) + 1$
10: **end for**

Example 2. As an example, consider the problem instance $(T, \prec)$ in Figure 2, where $T_1 = \{t_1, t_2\}$, $T_2 = \{t_3, t_4\}$, and $T_3 = \{t_5, t_6\}$. The precedence relations are $t_1 \prec t_3 \prec t_5$. The interval based algorithm ISA outputs the constraints on the tasks as follows: $C_1(t_1) = [1, 1]$, $C_1(t_2) = [1, 3]$, $C_2(t_3) = [2, 2]$, $C_2(t_4) = [1, 3]$, $C_3(t_5) = [3, 3]$, $C_3(t_6) = [1, 3]$.

One can easily verify that any local schedule of the agent meeting these constraints will give a feasible global schedule. Moreover, any combined global schedule always has the optimal make-span of 3.

One could easily prove that each interval $[lb(t), ub(t)]$ computed by ISA is always non empty. As a result any pair of precedence constrained tasks have intervals such that if $t \prec t'$ then $ub(t) < lb(t')$, thus proving the feasibility and optimality of the schedules generated by ISA.

Theorem III.1. *The interval based scheduling based coordination algorithm (*ISA*) ensures a correct global schedule and it is efficient in terms of make-span.*

ISA allows for the maximum possible flexibility for a given problem instance without hurting make-span optimality. To prove that, we need to show that by weakening any of the constraints in C we can always generate either infeasible

schedules or non optimal schedules. This fact can be easily proven once we note, as an immediate consequence of the algorithm, that the set of constraint intervals does not contain holes, that is, there exist no $\tau \in [1, depth(T)]$ such that τ is not contained in any constraint interval of C.

Proposition III.2. *Increasing the interval suggested by any constraint in C imposed by* ISA *either leads to a infeasible schedule or leads to a non optimal make-span.*

Proof: We prove the proposition by contradiction. Suppose that C is a set of constraints for a scheduling instance $\langle \{T_i\}_{i=1}^n, \prec, c(), d() \rangle$ produced by ISA and let $\{\sigma_i()\}_{i=1}^n$ be a set of local schedules that guarantees a global feasible schedule. Suppose that there exists a task $t \in T_i$ for some i with $[lb(t), ub(t)] \in C$ such that changing the constraints to $[lb(t), ub(t) + c]$ for some $c \geq 1$ also ensures a correct schedule. Suppose that $t \in T_i$ and change $\sigma_i(t)$ to $ub(t) + c$. According to ISA this schedule is still alloweable. Now either $ub(T) + c > depth(T)$ or $ub(T) + c \leq depth(T)$. In the first case, we clearly have a non-optimal make span caused by $\sigma_i(t) = ub(t) + c > depth(T)$. In the second case, clearly, there must exist a task t' such that $t \prec t'$ and $lb(t') = ub(t) + 1$. Change the schedule $\sigma_j(t')$ to $\sigma_j(t') = lb(t')$. Again, such a schedule should be allowed. But then $t \prec t'$ is violated since $\sigma_i(t) \geq \sigma_j(t')$. ■

However, the minimal global make-span is ensured by the proposed algorithm ISA only under the assumption that the participant agents have capabilities to perform an unlimited 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 does only a single task at any point in time. Since in such cases, each agent can only carry out their tasks sequentially, we call such agents (with limited concurrent execution capability) *sequential* agents. We show that with the assumption of sequential agents, the problem of finding the optimal make-span is NP-hard. Therefore, we have to rely on the construction of task constraints that guarantee approximately make-span efficient schedules.

IV. Scheduling Sequential Agents

In the previous section, we have shown that if the agents have unbounded concurrent capacity to perform assigned tasks, there exists a polynomial algorithm (i.e. ISA) which ensures a correct and make-span optimal global schedule, yet leaving maximal freedom to the agents. The assumption of unbounded concurrency can be shown to be crucial here: unless P=NP, it is not possible to come up with a set of additional constraints on the schedules of the participating agents that guarantees a make-span constrained global schedule:

Proposition IV.1. *Given a scheduling problem instance $\langle \{T_i\}_{i=1}^n, \prec, c(), d() \rangle$, where the concurrency bound $c(i)$ equals 1 for every agent A_i, and an integer K, the problem of finding a set of constraints C_i for each agent that guarantee the existence of a global schedule with make-span K is NP-hard when the number of agents is 3 or more.*

Proof: The hardness result can be straightforwardly obtained by reducing the flow shop scheduling problem to our bounded distributed scheduling problem. The flow shop problem [7] is defined as follows:

Definition 1. FLOW-SHOP SCHEDULING (FSS): Given $m \in Z^+$ of processors, a set J of jobs, each job $j \in J$ consisting of m tasks $t_1[j], t_2[j], \ldots, t_m[j]$, (where task $t_i[j]$ has to be performed by processor i), a length $l(t) \in \mathbb{N} \cup \{0\}$ for each such task t, and an overall deadline $D \in Z^+$, does there exist a flow-shop schedule $\sigma_i : J \rightarrow N \cup \{0\}$ for J, and σ_i ($1 \leq i \leq m$) is a collection of one processor schedules, that meets the overall deadline, where the following constraints hold?

- $\sigma_i(j) > \sigma_i(k)$ implies $\sigma_i(j) \geq \sigma_i(k) + l(t_i[k])$;
- the intervals$[\sigma_i(j), \sigma_i(j) + l(t_i[j]))$ are all disjoint for each $j \in J$;
- $\sigma_i(j) + l(t_i[j]) \leq D$ for $1 \leq i \leq m, 1 \leq j \leq |J|$;
- for each $j \in J$ and $1 \leq i < m, \sigma_{i+1}(j) \geq \sigma_i + l(t_i[j])$

We know from [8], that the flow shop scheduling problem is (strongly) NP-complete when there are 3 or more machines, even when we allow for preemption. In the reduction, let agents correspond to machines. The set of tasks T is the set of all $J \times m$ tasks $t_i[j]$ and for all j and $1 \leq i < m$ we have $t_i[j] \prec t_{i+1}[j]$. Agent A_i receives all tasks $t_i[j]$. To see that this trivial reduction is correct, if the flow-shop schedule admits a feasible set of schedules σ_i, let the constraints C_i for the agents simply be the set $C_i = \{[\sigma_i(j), \sigma_i(j)] : t_i[j] \in T_i\}$. Conversely, suppose that $\{C_1, \ldots C_m\}$ is the collection of constraints that guarantees that any set of locally feasible schedules can be merged into a global schedule with make-span K. Then each agent A_i can efficiently find a local schedule $\sigma_i()$ that satisfies its local constraints and the merge of them satisfies the make-span constraint. ■

Hence, we conclude that finding a set of constraints that guarantees the existence of a globally *optimal* schedule in the bounded agent case should be NP-hard.

Since the bounded capacity problem cannot be solved efficiently (unless P=NP), we will try to adapt our ISA algorithm to obtain a solution that still guarantees flexibility, but does not guarantee an optimal make-span solution.

Consider the example the the intervals generated by ISA in Example 2. Since every agent can only perform one task and since all its tasks have the same interval, not all the tasks can be successfully performed. The resulting schedule by ISA is infeasible.

We first start with a naive extension of ISA in order to develop feasible sequential schedules. We show that this naive algorithm, ISA_nSEQ, not only severely restricts the autonomy of the agents, but also could result in very bad schedules in terms of make-span. We then introduce another algorithm based on ISA, called ISA_SEQ, which gives much more freedom for the agents to schedule their own tasks. Moreover, the resulting global schedules have a constant upper bound on the make span.

Algorithm 2 The naive sequential adaptation of ISA (ISA_nSEQ)

1: **Inputs:** $[lb(t), ub(t)]$ for all the tasks $t \in T$ as computed by ISA
2: **Outputs:** Revised $lb(t)$ and $ub(t)$ for each $t \in T$
3: **for** each agent A_i **do**
4: **while** there exists a conflict between any two tasks $t, t' \in T_i$ **do**
5: add $t \prec t'$ or $t' \prec t$ to the partial order
6: Run ISA on the updated partial order
7: **end while**
8: **end for**

A. A naive algorithm for sequential ISA

The sequential agent cannot perform more than one task at any point in time. Thus, if the intervals generated by ISA for any two tasks that belong to the same agent have any overlap, then the agent may not be able to develop schedules such that these tasks can be performed sequentially. Therefore, in order to adapt ISA to the sequential cases, we need to handle such *conflicting* tasks. A straightforward way to define *conflicting tasks* is as follows.

Definition 2. Two tasks, $t, t' \in T_i$, have conflicting intervals with each other if, $[lb(t'), ub(t')] \sqcap [lb(t), ub(t)] \neq [\ ]$.

We know that the ISA algorithm ensures that tasks that have precedence relationships between them do not have overlapping intervals. Therefore, the idea now would be to add a precedence relation between conflicting tasks and rerun the ISA algorithm every time we encounter a conflict. More formally, this adapted version of ISA, called the ISA_nSEQ algorithm, can be described as in Algorithm 2.

The resulting algorithm stops after less than $O(|T|)^2$-calls of the ISA algorithm and clearly guarantees that each agent can perform its set of tasks while satisfying the constraints.

Example 3. Consider the problem instance in Figure 2, and the intervals returned by ISA as shown in Example 2. Since $C_1(t_1) = [1, 1]$ and $C_1(t_2) = [1, 3]$, according to Definition 2, the tasks t_1 and t_2 conflict. So the algorithm adds the precedence relation $t_1 \prec t_2$ between them and run ISA again. The resulting updated constraint on t_2 becomes $[2, 2]$, while others remain same. The algorithm then checks T_2 and finds a conflict again. It then adds the constraint between t_3 and t_4 as $t_3 \prec t_4$. Similarly, it continues by adding $t_5 \prec t_6$ in T_3. The updated partial order $(T, \prec')$ is shown in Figure 3.

The resulting constraints which ensure the feasible sequential schedules for all three agents are: $C'_1(t_1) = [1, 1]$, $C'_1(t_2) = [2, 2]$, $C'_2(t_3) = [2, 2]$, $C'_2(t_4) = [3, 3]$, $C'_3(t_5) = [3, 3]$, $C'_3(t_6) = [4, 4]$. One can easily see that the local schedules will give a feasible global schedule. The make-span of the global schedule is 4.

The algorithm ISA_nSEQ seems to work quite well in terms of the make-span it generates in the simple example as shown above. Unfortunately, in some cases, its worst case performance ratio: $\beta = \frac{\textit{worst case make-span}}{\textit{best case make-span}}$, can be as high as the number m of agents.

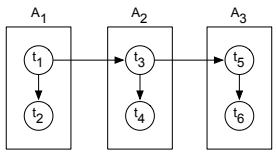

Fig. 3. Partial order generated by ISA_nSEQ.

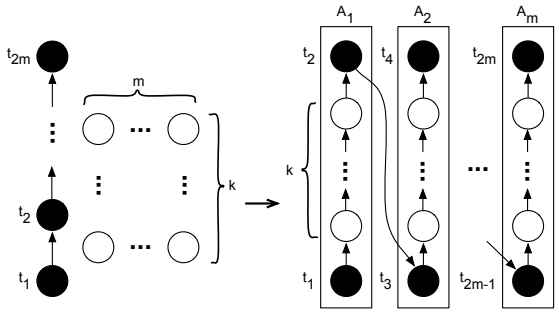

Fig. 4. An example of worst case performance of ISA_nSEQ.

To see this, consider Figure 4 and suppose that we have a set of tasks T containing just one chain of $2m$ tasks (the solid dots) $t_1 \prec t_2 \prec \ldots \prec t_{2m}$ and $k \times m$ tasks t' that are not constrained at all. Each agent A_j is assigned to two tasks in the chain: task t_{2j-1} and t_{2j} in the chain and it also receives k unconstrained tasks t'. So $|T_j| = k + 2$ for every agent A_j. Due to ISA_nSEQ, every agent could end up in ordering the chain task t_{2j-1} as its first task and the second chain task t_{2j} as its last task after all its other unconstrained tasks t'. This means that the total make-span could be as high as $\sum_{j=1}^{m} |T_j| = m \times (k + 2)$.

The optimal make-span, however, can be obtained by ordering the first chain task t_{2j-1} of agent A_j as its $(2j-1)$-th task to execute, followed by task t_{2j} followed by (if necessary) $k+2-2j$ other tasks. This would create a make-span of $max\{2m, k + 2\}$. Hence, $\beta = \frac{m \times (k+2)}{max\{2m, k+2\}} = min\{m, k + 2\} \leq m$.

B. Matching-based Sequential ISA

In the algorithm ISA_nSEQ, we defined the conflicting intervals as intervals that have some overlaps. However, using this definition the ISA_nSEQ algorithm unnecessarily restricts the flexibility of the agents. To see this, consider the intervals generated by ISA in Example 2, where, according to Definition 2, all agents have conflicting intervals. However, it is not necessary to add any precedence relation between any two tasks, because every agent can come up a feasible sequential schedule within the intervals given by C_i. A feasible schedule for this problem could have the following precedence relations $t_1 \prec t_2; t_4 \prec t_3; t_6 \prec t_5$.

Hence, if we relax the definition of conflicting tasks in Definition 2, the agent would have more freedom on

making their schedules by reducing the number of additional precedence constraints. We therefore have to find out exactly when a set of constraints C_i for a set of tasks T_i does or does not allow for a sequential schedule for agent A_i. It turns out that this decision problem in fact is a *maximum matching problem*.

Definition 3. Let C_i be a set of interval constraints for a set of tasks T_i. The matching graph associated with (T_i, C_i) is the bipartite graph $MG_i = (T_i \cup S_i, E_i)$ where $S_i = \{\tau \mid \exists[lb(t_j), ub(t_j)] \in C_i[lb(t_j) \leq \tau \leq ub(t_j)]\}$ is the set of time points occurring in the intervals making up C_i and $E_i = \{\{t_j, x_j\} | lb(t_j) \leq x_j \leq ub(t_j),\ [lb(t_j), ub(t_j)] \in C_i\}$ is the set of edges, where each edge $\{t_j, \tau_j\} \in E$ corresponds to an allowable scheduling time for task t_j.

The matching graph contains an edge between task t_j and time point τ whenever τ occurs in the interval constraint specified for task t_j. Hence, if MG_i contains a *matching* M [9], that contains *all* the tasks $t \in T_i$, we also have a sequential schedule satisfying the constraints C_i, taking $\sigma_i(t_j) = \tau_j$ iff $\{t_j, \tau_j\} \in M$. Slightly abusing language we call such an M a *matching associated* with (T_i, C_i).

Since determining a *maximum matching* is a polynomial problem [9], we can easily check whether a sequential schedule exists by determining whether $MG_i = (T_i \cup S_i, E_i)$ has a maximal matching containing all tasks $t_j \in T_i$.

If, however, such a maximum matching does not contain all tasks in T_j, we have a *conflict* between some task t in the maximum matching and a task t' not occurring in the maximal matching. To solve this conflict, (i) we take a task t not occurring in a maximum matching, (ii) we take another task t' in the matching with an overlapping interval and (iii) we add a precedence constraint between them ($t \prec t'$ or $t' \prec t$) in the partial order $(T, \prec)$. This will effectively remove the conflict between sequentially scheduling t and t'. But of course, due to the addition of this precedence tuple, some constraints might change. Therefore, we *rerun* the ISA algorithm on the extended partial order. Finally, we should obtain a set of constraints that allows sequential schedules, because every conflict detected will never reoccur due to the addition of a precedence tuple between conflicting tasks. More exactly, this adapted version of ISA called the ISA_SEQ algorithm can be described as in Algorithm 3.

The resulting algorithm stops after less than $O(|T|^2)$-calls of the ISA algorithm and clearly guarantees that each agent can construct a sequential schedule while satisfying the constraints.

Example 4. Consider again the problem instance in Figure 2, and the intervals returned by ISA are shown in Example 2. Due to the updated definition of conflicting intervals, there is no conflicting tasks within any agent. All three agents can schedule their two local tasks between intervals $[1, 3]$. The algorithm ISA_SEQ ensures the maximal freedom of autonomous scheduling of the agents. Furthermore, it also leads to the optimal global schedule with make-span 3.

The above example shows a very special case for

Algorithm 3 The matching based sequential adaptation of ISA (ISA_SEQ)

1: **Inputs:** constraint intervals $[lb(t), ub(t)]$ for all the tasks $t \in T$ as computed by ISA
2: **Outputs:** Revised $lb(t)$ and $ub(t)$ 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 **do**
4: take a task t contained in M_i and an overlapping 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**

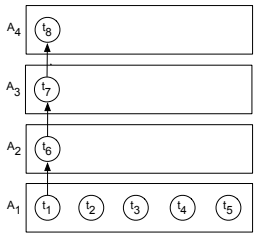

Fig. 5. An example of the worse case performance of ISA_SEQ.

ISA_SEQ, where it is able to ensure both the optimality and maximal flexibility. However, in the most of cases, due to the hardness of the problem, ISA_SEQ cannot come up with an optimal schedule. To study how bad ISA_SEQ could perform, we now analyze its worst case scenario.

Given a set of tasks $(T, \prec)$ and their allocations to the agents, let the original depth of $(T, \prec)$ be l. Assume the intervals generated by ISA cannot generate the feasible sequential schedule for some agent[1]. Every time when ISA_SEQ adds a precedence constraint, the depth of $(T, \prec')$ is either increased by 1 or stays same. Note because of ISA, the make-span of the resulting global schedule is bounded by the depth of $(T, \prec')$. ISA_SEQ keeps adding precedence constraints, and thus keeps increasing the depth of $(T, \prec')$, until the resulting constraints on each agent's tasks ensure the feasible sequential schedule. In other words, the make-span optimality is compromised with more and more added precedence constraints. Now suppose in some case $(T, \prec)$, we only need to add one precedence constraint so that the feasible sequential schedules are guaranteed with the increased make-span. In this case, the resulting make-span is $l+1$. The worst case of the algorithm ISA_SEQ is, however, to add the constraints in such a way that they keep on increasing the length of the critical path .

We now use the example in Figure 5 to illustrate this intuition and derive a bound for ISA_SEQ. In the figure, there exist a single task chains of length 4. Now if the algorithm chooses to add precedence constraints between

[1]If the intervals by ISA directly give the feasible solution, the solution is then optimal in make-span and has the maximal flexibility for the agents.

Step	Precedence Constraints added	Intervals of T_1
0		$\mathbf{t_1} : [1,1]$; $\mathbf{t_2}, \mathbf{t_3}, \mathbf{t_4}, \mathbf{t_5} : [1,4]$
1	$t_2 \prec t_3$ $t_3 \prec t_4$ $t_4 \prec t_5$	$\mathbf{t_1}, \mathbf{t_2} : [1,1]$ $t_3 : [2,2]; t_4 = [3,3]$ $t_5 : [4,4]$
4	$t_2 \prec t_1$ $t_3 \prec t_1$ $t_4 \prec t_1$ $t_5 \prec t_1$	$t_2 : [1,1]$; $t_3 : [2,2]; t_4 : [3,3]$ $t_5 : [4,4]$ $t_1 : [5,5]$

TABLE I

AN EXAMPLE OF THE WORST CASE SCHEDULING OF ISA_SEQ

tasks $t_2, \ldots t_5$ first and then adds the constraints between task t_1 and every task in $\{t_2, \ldots t_5\}$ in the end, it would have created a chain of 8 tasks leading to a make-span of 8 (the interval of task t_8 is $[8,8]$). Table I captures the decisions taken by ISA_SEQ during various steps that lead to the worst case make-span. On the other hand, if the algorithm chose to add a precedence constraint $t_2 \prec t_1$, then we would have a feasible schedule with make-span of 5.

More generally, if we have n critical paths of length l tasks, and we let the maximum number of unconstrained tasks allocated to any agent be k. Now in the worst case, if the agent with the largest number of tasks has the first tasks of the n critical paths, the algorithm could choose to resolve the conflict between the k unconstrained tasks creating a chain of length $k-1$. Finally it could resolve the conflict between the critical path tasks and the unconstrained tasks and create a chain of length $k+l+n-1$. In the best case, the algorithm could choose to add a precedence constraint between the n critical paths first and then add a precedence constraint between the last critical path task and some unconstrained task because of which the longest chain will be $l+n-1$. Therefore, the performance ratio is $\beta = \frac{k+l+n-1}{max\{k, l+n-1\}} \leq 2$. It means that by using ISA_SEQ, one can guarantee that the make-span of the combined global schedule would be at most twice as long as the optimal one, but would still afford agents with flexibility to choose between several schedules.

V. 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 the constraints on the starting time of each task. In this way, we allow flexibility for each agent to choose its schedule 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.

When the agents have unbounded capacity of handling concurrent tasks, interval based scheduling algorithm ISA generates a set of constraints on the starting time of the tasks which gives the optimal make-span for the global schedule, and also affords maximal flexibility for the agents on their local scheduling.

When the agents are sequential, we have shown that it is NP-hard to find a make-span optimal schedule. We then introduced two heuristics as extensions of ISA: the naive sequential ISA (ISA_nSEQ), and the matching based sequential ISA (ISA_SEQ). We show that ISA_SEQ outperforms ISA_nSEQ: (i) ISA_SEQ guarantees the performance bound of 2 while ISA_nSEQ may produce the solution which is m times as worse as the optimal one; (ii) ISA_SEQ allows more freedom for the agents to make their own schedules.

It is also important to note that our algorithms can return sub-optimal make-spans in several cases. This loss in performance is mainly due to the fact that agents are allowed some degree of flexibility in designing their schedules.

As an extension of the current work, we plan to formally analyze the minimal degree of flexibility that the proposed algorithms can ensure different task structures. Furthermore, we would also like to investigate the trade-off between the degree of flexibility and the loss of the make-span efficiency. Such a study would enable us to design algorithms that are better customized to specific applications.

The resulting intervals produced by ISA also introduce some other interesting issues. For instance, the agents could trade or negotiate about the lengths of the intervals locally so as to better suit their individual necessities. This situation could be easily extended to a situation where agents who perform tasks within shorter intervals are given higher share of the profits than the other agents. A very intuitive application is in multi-modal logistics domain, where the collaborating partners negotiate about time and profits.

REFERENCES

[1] G. C. Sih and E. A. Lee, "Scheduling to account for interprocessor communication within interconnection-constrained processor networks." in *ICPP (1)*, 1990, pp. 9–16.

[2] H. Topcuouglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, 2002.

[3] O. Beaumont, V. Boudet, and Y. Robert, "The iso-level scheduling heuristic for heterogeneous processors," in *Parallel, Distributed and Network-based Processing, 2002. Proceedings. 10th Euromicro Workshop on*, 9-11 Jan. 2002, pp. 335–350.

[4] M. Maheswaran and H. J. Siegel, "A dynamic matching and scheduling algorithm for heterogeneous computing systems," in *HCW '98: Proceedings of the Seventh Heterogeneous Computing Workshop*. Washington, DC, USA: IEEE Computer Society, 1998, p. 57.

[5] W. Walsh, M. Wellman, P. Wurman, and J. MacKie-Mason, "Some economics of market-based distributed scheduling," in *Distributed Computing Systems, 1998. Proceedings. 18th International Conference on*, 26-29 May 1998, pp. 612–621.

[6] C. Wang and Y. Xi, "Noncooperative game on scheduling: The single machine case," in *16th IFAC World Congress in Prague*, 2005.

[7] M. Garey and D. Johnson, *Computers and Intractability - a guide to the theory of NP-completeness.* W.H. Freeman and Company, 1979.

[8] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified np-complete problems," in *STOC '74: Proceedings of the sixth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM Press, 1974, pp. 47–63.

[9] T. T. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to algorithms*. Cambridge, MA, USA: MIT Press, 1990.