Autonomous Scheduling with Unbounded and Bounded Agents\textsuperscript{1}

Chetan Yadati Narasimha\textsuperscript{a} Cees Witteveen \textsuperscript{a} Yingqian Zhang \textsuperscript{a}
Mengxiao Wu \textsuperscript{b} Han la Poutré \textsuperscript{b}

\textsuperscript{a} Delft University of Technology, Delft
\textsuperscript{b} Centrum voor Wiskunde en Informatica, Amsterdam

Autonomous scheduling deals with the problem of enabling agents to schedule a set of interdependent tasks in such a way that their individual schedules always can be merged into a global feasible schedule without suffering any changes. 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. Furthermore, we study the performance loss on efficiency due to such flexibility of the agents.

We assume a finite set of tasks $T$, each of which takes finite time $d(t_i) \in Z^+$ to complete. 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 of $\prec$ to indicate the immediate precedence relation between tasks, i.e., $t \prec t'$ if $t \prec t'$ and there exists no $t''$ such that $t \prec t''$ and $t'' \prec t'$. We use a directed acyclic graph (DAG) $G = (T, \prec)$ to represent the task structure of $T$. We assume that $T$ has been assigned to a set of autonomous agents $A$ according to a pre-defined task allocation $\phi : T \rightarrow A$. We denote the set of tasks allocated to agent $A_i$ by $T_i = \phi^{-1}(A_i)$. 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 $\langle\{T_i\}_{i=1}^n, \prec, c, d(\cdot)\rangle$ is a scheduling instance. Given such a scheduling instance, a global schedule for it is a function $\sigma : T \rightarrow Z^+$ determining the starting time $\sigma(t)$ for each task $t \in T$. Of course, we prefer a schedule $\sigma$ which minimizes makespan, i.e., among all feasible schedules $\sigma'$, we prefer a schedule $\sigma$ such that $\max_{t \in T}(\sigma(t) + d(t)) \leq \max_{t \in T}(\sigma'(t) + d(t))$.

Our goal is to design a minimal set of constraints $C_i$ for each agent $A_i$, such that the merging of individual schedules $\sigma_i$ that satisfy $C_i$ always is a globally feasible schedule. Moreover, we would like the merging to be also makespan efficient.

Agents with unbounded concurrent

We show that there exists a simple makespan efficient autonomous scheduling, provided that the agents are capable of processing as much tasks concurrently as possible, i.e., they have unbounded concurrent capacity.

The algorithm is based on determining the earliest and latest starting times for a task in a given problem instance. To determine the earliest and the latest starting times for a task we first determine the depth and height of the task in the task graph. Based on these quantities we determine an interval $[lb(t), ub(t)]$, where $lb(t) = \text{depth}(t)$ and $ub(t) = \text{height}(t)$. These quantities alone are not enough to ensure autonomous scheduling because, tasks constrained by precedence relationships might have overlapping intervals. In such cases, it is possible for agents to violate precedence constraints between the tasks and hence, we arbitrate such overlaps between any two tasks $t, t'$ with the precedence relation $t \prec t'$ as follows: $ub(t) = lb(t) + \frac{ub(t') - lb(t') - d(t)}{2}$ and $lb(t') = ub(t) + d(t)$.

The complete description of the algorithm is given in the Algorithm 1.

\textsuperscript{1}The full version of the paper will appear in the Proceedings of Sixth German Conference on Multi-Agent system Technologies (MATES’08).
Algorithm 1 Generalised Interval based Scheduling (ISA)

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

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

1: \(\text{depth}(T) := \max_{t \in T}\{\text{depth}(t) + d(t)\}\)
2: for all \(t \in T\) do
3: \(\text{lb}(t) := \text{depth}(t)\)
4: \(\text{ub}(t) := \text{depth}(T) - \text{height}(t)\)
5: end for
6: for all \(t, t' \in T\) such that \(t \prec t'\) and \(\text{lb}(t') - \text{ub}(t) < d(t)\) do
7: \(\text{ub}(t') = \text{lb}(t') + \lceil \frac{\text{ub}(t') - \text{lb}(t') - d(t)}{2} \rceil\)
8: \(\text{lb}(t') = \text{ub}(t) + d(t)\)
9: end for
10: for all \(t \in T\) do
11: return \(C(t) = [\text{lb}(t), \text{ub}(t)]\)
12: end for

We show that: (i) ISA algorithm ensures a correct global schedule and it is efficient in terms of makespan, and furthermore, (ii) ISA gives maximum flexibility to agents without sacrificing makespan efficiency.

Agents with bounded concurrent

Algorithm ISA works with the assumption that agents have a capacity to perform an unbounded number of tasks at any given point of time. This assumption is in several cases 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.

However, it can be shown that if we limit the capacity of agents to performing a single task at any point in time, then given a sequential scheduling instance 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. This can be proved through reduction to the well known Set Partitioning problem. Therefore, we have to rely on approximation algorithms for autonomous scheduling in the sequential agent case. Our proposed algorithm first uses 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.

We use maximum matching 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. 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 \(t_i\) in the matching and a task \(t_i'\) not in the matching. Such a conflict can be resolved by adding a precedence constraint \(t_i \prec t_i'\) between \(t_i\) and \(t_i'\) 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 \(t_i\) and \(t_i'\) is removed and (ii) the global makespan \(d(T)\) might be increased. 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. 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.

We assume that 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. Hence, we can reuse the approximation algorithm to obtain a 2-approximation algorithm for autonomous scheduling of tasks with arbitrary durations.

As for our future work, 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.