Scheduling on Two Unbounded Resources with Communication Costs

Massinissa Ait Aba[1(✉)], Alix Munier Kordon[2], and Guillaume Pallez (Aupy)[3]

[1] CEA, LIST, Computing and Design Environment Laboratory, Palaiseau, France
Massinissa.aitaba@cea.fr
[2] Sorbonne Université, CNRS-UMR 7606 LIP6, Paris, France
Alix.Munier@lip6.fr
[3] Inria, Labri & University of Bordeaux, Talence, France
guillaume.pallez@inria.fr

Abstract. Heterogeneous computing systems are popular and powerful platforms, containing several heterogeneous computing elements (e.g. CPU+GPU). In this work, we consider a platform with two types of machines, each containing an unbounded number of elements. We want to execute an application represented as a Directed Acyclic Graph (DAG) on this platform. Each task of the application has two possible execution times, depending on the type of machine it is executed on. In addition we consider a cost to transfer data from one platform to the other between successive tasks. We aim at minimizing the execution time of the DAG (also called makespan). We show that the problem is NP-complete for graphs of depth at least three but polynomial for graphs of depth at most two. In addition, we provide polynomial-time algorithms for some usual classes of graphs (trees, series-parallel graphs).

Keywords: Scheduling · DAG · Makespan · Heterogeneous platform

1 Introduction

In this work we revisit the work by Barthou and Jeannot [1]. We consider that we have two platforms, each with an unbounded number of processors. We want to execute an application represented as a Directed Acyclic Graph (DAG) using these two platforms. Each task of the application has two possible execution times, depending on the platform it is executed on. Finally, there is a cost to transfer data from one platform to another one between successive tasks.

In their work, Barthou and Jeannot [1] considered that each task could be executed on both platforms and were able to compute in polynomial time an optimal schedule. Here we study the problem where tasks cannot be re-executed. While this problem arises more from a theoretical understanding of the process, we can envision several directions linked to the usage of parallel machines where it could be useful, in High-Performance Computing or Cloud Computing.

In High-Performance Computing, one has to deal with simulations using millions of nodes. These simulations run on machines consisting often of either

R. Yahyapour (Ed.): Euro-Par 2019, LNCS 11725, pp. 117–128, 2019.
https://doi.org/10.1007/978-3-030-29400-7_9

homogeneous, or of two types of nodes (e.g. CPU+GPU)[1]. These simulations generates huge volume of data, saturating access to the Parallel File System. A recent technique to deal with this data is to analyze it *in-situ* [2], that is, while it is generated. This analysis can be done both on CPUs or GPUs, with a cost to move data around. It uses fewer nodes than the simulation by many orders of magnitude, and the only constraint is not to decelerate the main simulation. Hence one will allocate as many nodes as needed to these analysis (hence almost an unbounded number).

Another motivation in the context of Big-Data Analytics is the concept of Geo-Distributed Data-centers [3]. Information for each jobs is located in different data-centers, and the main cost is to move data-around. The number of nodes in each data-center is less an issue. Furthermore in Big-Data analytics, the data-dependencies of the graph are often linked to Map-Reduce-like applications (Hadoop, Spark etc), also called Bi-Partite Graph. This is a more general version of our problem where we have k instead of 2 unbounded resources.

Related Work: Recently, the problem of scheduling jobs on hybrid parallel platforms (k types of homogeneous machines) has attracted a lot of attention. Due to lack of space we focus on those work closest to us. More details are available in the companion report of this work [4].

The most commonly studied problem is the one when $k = 2$ (typically CPU/GPU platforms) with the objective of minimizing the makespan. The problem is in NP even when the number of each resource is bounded. In this case, several families of approximation algorithms have been studied, see for example Ait Aba et al. [5] for general graphs, or Kedad-Sidhoum et al. [6] and Marchal et al. [7] for independent tasks.

In the context of an unlimited number of processors, to limit the size of the description of the problem, one needs to consider a limited number of *performance profile* (computation/communication costs). Indeed otherwise if the size of the problem is not bounded, (almost) any algorithm is polynomial in the size of the instance. If there are no communication delays, the problem is trivial, where each task is simply assigned to the fastest machine. In the case where all processors have the same processing power and there is a cost for any communication the problem remains NP-complete. Darbha and Agrawal [8] provide an optimal solution TDS (Task Duplication based Scheduling) when the communications are not too large w.r.t the computation costs. Later, Park and Choe [9] extended this work when the communications are significantly larger than computations.

The closest to our work is the work of Barthou and Jeannot [1] who studied the problem of minimizing the makespan on two unbounded hybrid platform. They provide a $\Theta(4|E| + 2|V|)$ polynomial-time algorithm when duplication of jobs is allowed (namely, each job is executed on both platforms as soon as possible). They further discuss a possible extension of their work to the case where the number of processors of each type is limited by differentiating the allocation

[1] See for example the supercomputers at Argonne National Laboratory https://www.alcf.anl.gov/computing-resources (Accessed 09/2018).

part (using their algorithm) and the scheduling part. While the problem with duplication makes sense when the number of processors is unbounded to reduce the makespan, it may lead to other problems, such as additional energy consumption and significant memory footprint, hence motivating our study without duplication.

Finally, there is a wide range of heuristic solutions to the problem of CPU-GPU. They can be roughly partitioned in two classes: clustering algorithms and list-scheduling algorithms. Clustering algorithms [10] usually provide good solutions for communication-intensive graphs by scheduling heavily communicating tasks onto the same processor. List-scheduling heuristics such as HEFT [11] often have no performance guarantee with communication costs, but allow to handle a limited number of processors.

Results: Our main contributions are the following. We formalize the model in Sect. 2, and show that the problem is NP-complete for graphs of depth at least three but polynomial for graphs of depth at most two. We show that the problem cannot be approximated to a factor smaller than 3/2 unless $\mathcal{P} = \mathcal{NP}$. Then, we provide polynomial-time algorithms for several classes of graphs. Those results are presented in Sect. 3. Finally, in Sect. 4, we provide concluding remarks and future directions.

2 Model

An application is represented by a Directed Acyclic Graph (DAG) $\mathcal{G} = (V, E)$, such that for all $(v_1, v_2) \in E$, v_2 cannot start its execution before the end of the execution of v_1. We consider a parallel platform of two types of machines: machines of type $\mathcal{A}$ and machines of type $\mathcal{B}$. For each type of machine we consider that there are an unbounded number of them.

We define two cost functions: $t_{\mathcal{A}} : V \to \mathbb{R}+$ (resp. $t_{\mathcal{B}} : V \to \mathbb{R}+$) that define the time to execute a task $v \in V$ on a machine of type $\mathcal{A}$ (resp. $\mathcal{B}$).

We also define two communication cost functions: $c_{\mathcal{AB}} : E \to \mathbb{R}+$ (resp. $c_{\mathcal{BA}} : E \to \mathbb{R}+$), such that for all $(v_1, v_2) \in E$, if v_1 is scheduled on a machine of type $\mathcal{A}$ (resp. $\mathcal{B}$) and v_2 is scheduled on a machine of type $\mathcal{B}$ (resp. $\mathcal{A}$), then v_2 needs to wait $c_{\mathcal{AB}}(v_1, v_2)$ (resp. $c_{\mathcal{BA}}(v_1, v_2)$) units of time after the end of the execution of v_1 to start its execution. We assume that there is no communication cost within a platform of a given type ($c_{\mathcal{AA}} = c_{\mathcal{BB}} = 0$).

The goal is to find a schedule of each task that minimizes the execution time (or makespan). Since there is an unbounded number of processors of each type, it corresponds to finding an allocation $\sigma : V \to \{\mathcal{A}, \mathcal{B}\}$ of all tasks on each type of processors. For an allocation σ and a path $p = v_1 \to v_2 \to \cdots \to v_p$ of $\mathcal{G}$, we define the length of the path

$$\texttt{len}(p, \sigma) = t_{\sigma(v_1)}(v_1) + c_{\sigma(v_1)\sigma(v_2)}(v_1, v_2) + t_{\sigma(v_2)}(v_2) + \cdots + t_{\sigma(v_p)}(v_p).$$

The makespan is then obtained by computing the longest path of the graph $\mathcal{G}$ including the corresponding duration of the tasks and the computations costs: $MS(\mathcal{G}, \sigma) = \max_{p \in \{\text{paths of } \mathcal{G}\}} \texttt{len}(p, \sigma)$.

3 Results

In this section, we start by showing that the problem is strongly NP-complete for graph of depth 3, before providing some algorithms for specific graphs.

3.1 Complexity

Theorem 1. *The problem of deciding whether an instance of our main problem has a schedule of length* 2 *is strongly NP-complete even for graphs of depth* 3*.*

We perform the reduction from the 3-Satisfiability (3-SAT) problem which is known to be strongly NP-complete [12,13]: given $C_1, \cdots, C_m$ be a set of disjunctive clauses where each clause contains exactly three literals over $X = \{x_1, \cdots, x_n\}$ a set of boolean variables. Is there a truth assignment to X such that each clause is satisfied?

In the following, we write each clause $C_i = \tilde{x}_{i_1} \vee \tilde{x}_{i_2} \vee \tilde{x}_{i_3}$ where $(x_{i_1}, x_{i_2}, x_{i_3}) \in X^3$, and $\tilde{x}_k = x_k$ or $\bar{x}_k$. We are looking for a truth assignment such that $\bigwedge_{i=1}^{m} C_i$ is true.

Proof. From an instance $\mathcal{I}_1$ of 3-SAT: $C_1, \cdots, C_m$ over $\{x_1, \cdots, x_n\}$, we construct the following instance $\mathcal{I}_2$ for our problem.

For all $i \in \{1, \cdots, n\}$, we define 2 tasks v_i^0 and v_i^∞, and an edge (v_i^0, v_i^∞). Then for each clause $C_i = \tilde{x}_{i_1} \vee \tilde{x}_{i_2} \vee \tilde{x}_{i_3}$, 3 tasks $v_{i_1}^i, v_{i_2}^i, v_{i_3}^i$ are created and the following set of edges: $\{(v_{i_1}^i, v_{i_2}^i), (v_{i_2}^i, v_{i_3}^i), (v_{i_1}^i, v_{i_1}^\infty), (v_{i_2}^0, v_{i_2}^i), (v_{i_3}^0, v_{i_3}^i)\}$. For any $j \in \{1, \cdots, n\}$, $v_j^\star$ denotes the set of all the instanciations of x_j in $\mathcal{G}$.

Overall, the graph $\mathcal{G} = (V, E)$ of depth 3 has $2n + 3m$ vertices and $n + 5m$ edges.

We then define the execution and communication costs that can be written in unit size: $\forall j \in \{1, \cdots, n\}$, $t_{\mathcal{A}}(v_j^\infty) = t_{\mathcal{B}}(v_j^\infty) = t_{\mathcal{A}}(v_j^0) = t_{\mathcal{B}}(v_j^0) = 0$ and $c_{\mathcal{AB}}(v_j^0, v_j^\infty) = c_{\mathcal{BA}}(v_j^0, v_j^\infty) = 3$. For all edges $(v_j^i, v_j^\infty), (v_{j'}^0, v_{j'}^{i'}) \in E$, we add the communication costs $c_{\mathcal{AB}}(v_j^i, v_j^\infty) = c_{\mathcal{BA}}(v_j^i, v_j^\infty) = c_{\mathcal{AB}}(v_{j'}^0, v_{j'}^{i'}) = c_{\mathcal{BA}}(v_{j'}^0, v_{j'}^{i'}) = 3$. Then for $C_i = \tilde{x}_{i_1} \vee \tilde{x}_{i_2} \vee \tilde{x}_{i_3}$ we define the time costs:

$$t_{\mathcal{A}}(v_{i_j}^i) = 1 - t_{\mathcal{B}}(v_{i_j}^i) = \begin{cases} 1 & \text{if } \tilde{x}_{i_j} = \bar{x}_{i_j} \\ 0 & \text{if } \tilde{x}_{i_j} = x_{i_j} \end{cases} \tag{1}$$

and we set $c_{\mathcal{AB}}(v_{i_1}^i, v_{i_2}^i) = c_{\mathcal{BA}}(v_{i_1}^i, v_{i_2}^i) = c_{\mathcal{AB}}(v_{i_2}^i, v_{i_3}^i) = c_{\mathcal{BA}}(v_{i_2}^i, v_{i_3}^i) = 0$.

Finally, in the instance $\mathcal{I}_2$, we want to study whether there exists a schedule σ whose makespan is not greater than 2.

We show an example in Fig. 1 of the construction of the graph. Here, the clause $C_1 = x_1 \vee \bar{x}_4 \vee x_2$ is associated with the vertices v_1^1, v_4^1 and v_2^1 and the arcs set $\{(v_1^1, v_4^1), (v_4^1, v_2^1), (v_1^1, v_1^\infty), (v_4^0, v_4^1), (v_2^0, v_2^1)\}$. Moreover, $t_{\mathcal{A}}(v_1^1) = t_{\mathcal{A}}(v_2^1) = 0$, $t_{\mathcal{A}}(v_4^1) = 1$, $t_{\mathcal{B}}(v_1^1) = t_{\mathcal{B}}(v_2^1) = 1$ and $t_{\mathcal{B}}(v_4^1) = 0$. Note that $v_1^\star = \{v_1^0, v_1^\infty, v_1^1, v_1^2, v_1^3\}$, $v_2^\star = \{v_2^0, v_2^\infty, v_2^1, v_2^3\}$, $v_3^\star = \{v_3^0, v_3^\infty, v_3^2, v_3^3\}$ and $v_4^\star = \{v_4^0, v_4^\infty, v_4^1, v_4^2\}$.

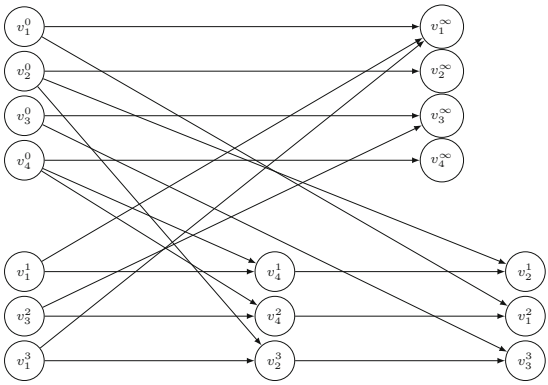


Fig. 1. Transformation of $(x_1 \vee \bar{x}_4 \vee x_2) \bigwedge (\bar{x}_3 \vee \bar{x}_4 \vee x_1) \bigwedge (x_1 \vee x_2 \vee x_3)$ ($m = 3$ clauses, $n = 4$ variables) into the associated graph $\mathcal{G} = (V, E)$.

Let $\mathcal{S}$ be the set of schedules such that, $\forall \sigma \in \mathcal{S}$, all tasks from $v_j^\star$ are scheduled by the same type of machines, *i.e*, for any couple $(v_j^\alpha, v_j^\beta) \in v_j^\star \times v_j^\star$, $\sigma(v_j^\alpha) = \sigma(v_j^\beta)$. The next lemmas provide dominance properties on feasible schedules of $\mathcal{I}_2$:

Lemma 1. *Any feasible solution σ of $\mathcal{I}_2$ belongs to $\mathcal{S}$.*

Proof. Let us suppose by contradiction that a feasible solution $\sigma \notin \mathcal{S}$. Two cases must then be considered:

- If there exists $j \in \{1, \cdots, n\}$ with $\sigma(v_j^0) \neq \sigma(v_j^\infty)$, then there is a communication delay of 3 between them and $\texttt{len}(v_j^0 \rightarrow v_j^1, \sigma) = 3$.
- Otherwise, $\forall j \in \{1, \cdots, n\}, \sigma(v_j^0) = \sigma(v_j^\infty)$. Thus, there exists a task v_j^i with $\sigma(v_j^i) \neq \sigma(v_j^0)$. If v_j^i is associated to the first term of the clause C_i, then $(v_j^0, v_j^i) \in E$ and $\texttt{len}(v_j^0 \rightarrow v_j^i, \sigma) = 3$. Otherwise, $(v_j^i, v_j^\infty) \in E$ and $\texttt{len}(v_j^i \rightarrow v_j^\infty, \sigma) = 3$.

The makespan of σ is at least 3 in both cases, the contradiction.

Lemma 2. *For any schedule $\sigma \in \mathcal{S}$, $MS(\mathcal{G}, \sigma) = \max_{i \in \{1, \cdots, m\}} \texttt{len}(v_{i_1}^i \rightarrow v_{i_2}^i \rightarrow v_{i_3}^i, \sigma)$.*

Proof. To do this, we study the length of paths of $\mathcal{G}$.

- Let $j \in \{1, \cdots, n\}$, $\texttt{len}(v_j^0 \rightarrow v_j^\infty, \sigma) = 0$ since $\sigma(v_j^0) = \sigma(v_j^\infty)$.
- Let $i \in \{1, \cdots, m\}$ associated with the clause $C_i = \tilde{x}_{i_1} \vee \tilde{x}_{i_2} \vee \tilde{x}_{i_3}$:

  1. Let us consider first the path $v_{i_1}^i \rightarrow v_{i_1}^\infty$. By Lemma 1, $\sigma(v_{i_1}^i) = \sigma(v_{i_1}^\infty)$ and thus $c_{\sigma(v_{i_1}^i)\sigma(v_{i_1}^\infty)}(v_{i_1}^i, v_{i_1}^\infty) = 0$. Since $\texttt{len}(v_{i_1}^\infty, \sigma) = 0$,

$$\texttt{len}(v_{i_1}^i \rightarrow v_{i_1}^\infty, \sigma) = \texttt{len}(v_{i_1}^i, \sigma) \leq \texttt{len}(v_{i_1}^i \rightarrow v_{i_2}^i \rightarrow v_{i_3}^i, \sigma).$$

2. Let us consider now the path $v^0_{i_2} \to v^i_{i_2} \to v^i_{i_3}$. Similarly, $\sigma(v^0_{i_2}) = \sigma(v^i_{i_2})$ hence
$$\texttt{len}(v^0_{i_2} \to v^i_{i_2} \to v^i_{i_3}, \sigma) = \texttt{len}(v^i_{i_2} \to v^i_{i_3}, \sigma) \leq \texttt{len}(v^i_{i_1} \to v^i_{i_2} \to v^i_{i_3}, \sigma).$$
3. Lastly, for the path $(v^0_{i_3} \to v^i_{i_3})$, since $\sigma(v^0_{i_3}) = \sigma(v^i_{i_3})$,
$$\texttt{len}(v^0_{i_3} \to v^i_{i_3}, \sigma) = \texttt{len}(v^i_{i_3}, \sigma) \leq \texttt{len}(v^i_{i_1} \to v^i_{i_2} \to v^i_{i_3}, \sigma),$$
which concludes the lemma.

Assume that λ *is* a solution of $\mathcal{I}_1$, *Let* us show that the schedule defined as follow, $\forall j \in \{1, \cdots, n\}$, $\forall v^\alpha_j \in v^\star_j$,
$$\sigma_\lambda : v^\alpha_j \mapsto \begin{cases} \mathcal{A} & \text{if } \lambda(x_j) = 1 \\ \mathcal{B} & \text{if } \lambda(x_j) = 0 \end{cases}$$
has a makespan not greater than 2 and thus is a solution. Following Lemma 2, we must prove that $\forall i \in \{1, \cdots, n\}$, $\texttt{len}(v^i_{i_1} \to v^i_{i_2} \to v^i_{i_3}, \sigma_\lambda) \leq 2$.

For any clause $C_i = \tilde{x}_{i_1} \vee \tilde{x}_{i_2} \vee \tilde{x}_{i_3}$, since $\lambda(C_i) = 1$, there exists $j \in \{1, 2, 3\}$ such that $\lambda(\tilde{x}_{i_j}) = 1$. Two cases must be considered:

1. If $\tilde{x}_{i_j} = x_{i_j}$, then by definition $t_{\mathcal{A}}(v^i_{i_j}) = 0$. Since $\lambda(x_{i_j}) = 1$, $\sigma_\lambda(v^i_{i_j}) = \mathcal{A}$ and thus $\texttt{len}(v^i_{i_j}, \sigma_\lambda) = t_{\mathcal{A}}(v^i_{i_j}) = 0$.
2. Otherwise, $\tilde{x}_{i_j} = \bar{x}_{i_j}$ and $t_{\mathcal{B}}(v^i_{i_j}) = 0$. Now, as $\lambda(x_{i_j}) = 0$, $\sigma_\lambda(v^i_{i_j}) = \mathcal{B}$ and thus $\texttt{len}(v^i_{i_j}, \sigma_\lambda) = t_{\mathcal{B}}(v^i_{i_j}) = 0$.

$\texttt{len}(v^i_{i_j}, \sigma_\lambda) = 0$ in both cases, so $\texttt{len}(v^i_{i_1} \to v^i_{i_2} \to v^i_{i_3}, \sigma_\lambda) \leq 2$.

Assume now that we have a solution σ *of* $\mathcal{I}_2$,*let* us show that $\lambda_\sigma(x_j) = [\sigma(v^\infty_j) = \mathcal{A}]$ *is* a solution to $\mathcal{I}_1$.

Following Lemma 1, $\sigma \in \mathcal{S}$. Moreover, for any clause $C_i = \tilde{x}_{i_1} \vee \tilde{x}_{i_2} \vee \tilde{x}_{i_3}$, the corresponding path of $\mathcal{G}$ verifies $\texttt{len}(v^i_{i_1} \to v^i_{i_2} \to v^i_{i_3}, \sigma) \leq 2$. Thus, there is $j \in \{1, 2, 3\}$ with $\texttt{len}(v^i_{i_j}, \sigma) = 0$. Two cases must be considered:

1. If $\tilde{x}_{i_j} = x_{i_j}$ then by definition $t_{\mathcal{A}}(v^i_{i_j}) = 0$ and $t_{\mathcal{B}}(v^i_{i_j}) = 1$. So, $\sigma(v^i_{i_j}) = \mathcal{A}$ and thus $\lambda_\sigma(x_{i_j}) = 1$.
2. Else, $\tilde{x}_{i_j} = \bar{x}_{i_j}$ and thus $t_{\mathcal{A}}(v^i_{i_j}) = 1$ and $t_{\mathcal{B}}(v^i_{i_j}) = 0$. So, $\sigma(v^i_{i_j}) = \mathcal{B}$ and thus $\lambda_\sigma(\bar{x}_{i_j}) = 1$.

So, at least one term of C_i is true following λ_σ, λ_σ is then a solution to $\mathcal{I}_1$.

This concludes the proof that the problem is strongly NP-complete.

Corollary 1. *There is no polynomial-time algorithm for the problem with a performance bound smaller than* $\frac{3}{2}$ *unless* $\mathcal{P} = \mathcal{NP}$.

Proof. By contradiction, let us suppose that there exists a polynomial-time algorithm with a performance ratio $\rho < \frac{3}{2}$. This algorithm can be used to decide the existence of a schedule a length at most 2 for any instance $\mathcal{I}$. We deduce that there exists a polynomial time algorithm to decide the existence of a schedule of length strictly less than 3, which contradicts Theorem 1.

3.2 Polynomial Algorithms

Bi-partite Graphs. We have shown that the problem is NP-hard if the graph has depth 3. The natural question that arises is whether it is already NP-hard for graphs of lower depth. We show that it can be solved in polynomial time for graphs of depth 2 (bipartite graphs).

Theorem 2. BiPartAlgo($\mathcal{G}$) *described below provides an optimal solution in polynomial time with a complexity of* $\Theta(n|E|)$ *when* $\mathcal{G}$ *has depth 2.*

Observe that in the case of a bipartite graph $\mathcal{G} = (V, E)$, the paths are exactly the edges of $\mathcal{G}$. The intuition of the algorithm is then to compute first the makespan of all possible allocations for all edges, and then to remove pairs associated to forbidden allocations.

For any edge $(i, j) \in E$, 4 allocations are possible: $(\sigma(i), \sigma(j)) \in \{\mathcal{A}, \mathcal{B}\}^2 = \{(\mathcal{A}, \mathcal{A}), (\mathcal{A}, \mathcal{B}), (\mathcal{B}, \mathcal{A}), (\mathcal{B}, \mathcal{B})\}$. We define the set of quintuplet of all these allocations:

$$\texttt{WgPaths} = \Big\{(\texttt{len}(i \to j, \sigma), i, j, \sigma_i, \sigma_j) \Big| \\ (i,j) \in V, (\sigma(i), \sigma(j)) \in \{\mathcal{A}, \mathcal{B}\}^2, \sigma(i) = \sigma_i, \sigma(j) = \sigma_j \Big\}.$$

This set can be constructed in linear time by a simple iteration through all the edges of the graph by a procedure that we call MkWgPaths(V, E).

Finally to minimize the makespan, we iteratively remove from `WgPaths` the allocations that would maximize the makespan and check that there still exists a possible schedule.

Algorithm 1. Polynomial algorithm for $\mathcal{G} = (V, E)$ a bipartite graph

```
1: procedure BiPartAlgo(G)
2:     WgPaths ← MkWgPaths(G)
3:     P_alg ← True; P_tmp ← True            /* Two clauses with n variables */
4:     for (t_{σ_i σ_j}, i, j, σ_i, σ_j) ∈ WgPaths, by decreasing value of t_{σ_i σ_j} do
5:         P_tmp ← P_alg ∧ ((σ(i) ≠ σ_i) ∨ (σ(j) ≠ σ_j))
6:         if P_tmp is not satisfiable then Break end if
7:         P_alg ← P_tmp
8:     end for
9:     σ(1), ··· , σ(n) ← Solve(P_alg)                /* Using a 2-SAT solver*/
10: end procedure
```

In the rest, we use the following notation for a schedule σ and a time D:

$$\mathrm{WP}(D) = \big\{(i, j, \sigma_i, \sigma_j) \text{ s.t. } \big(t_{\sigma_i\sigma_j}, i, j, \sigma_i, \sigma_j\big) \in \texttt{WgPaths} \quad \text{and } t_{\sigma_i\sigma_j} > D\big\}$$

$$P_D(\sigma) = \bigwedge_{(i,j,\sigma_i,\sigma_j)\in \mathrm{WP}(D)} [(\sigma(i) \neq \sigma_i) \vee (\sigma(j) \neq \sigma_j)]$$

Intuitively, WP(D) is the set of paths and allocations of length greater than D.

Lemma 3. *Let σ be a schedule of makespan D, then $P_D(\sigma)$ is satisfied.*

This result is a direct consequence of the fact that there should be no path of length greater than D. Hence for $(i, j, \sigma_i, \sigma_j) \in \mathrm{WP}(D)$, we know that we do not have simultaneously in the schedule $(\sigma(i) = \sigma_i)$ and $(\sigma(j) = \sigma_j)$. Hence,

$$\neg \bigvee_{(i,j,\sigma_i,\sigma_j)\in \mathrm{WP}(D)} [(\sigma(i) = \sigma_i) \wedge (\sigma(j) = \sigma_j)] = \bigwedge_{(i,j,\sigma_i,\sigma_j)\in \mathrm{WP}(D)} [(\sigma(i) \neq \sigma_i) \vee (\sigma(j) \neq \sigma_j)] = P_D(\sigma) \quad (2)$$

Proof. (Proof of Theorem 2). Consider an instance $\mathcal{G}$ of the problem. Let D_{alg} be the deadline of the schedule returned by BiPartAlgo($\mathcal{G}$). Clearly, $D_{\mathrm{alg}} = \max_{(i,j)\in E}(t_{\sigma(i)}(i) + c_{\sigma(i)\sigma(j)}(i, j) + t_{\sigma(j)}(j))$. Let P_{alg} be the set of clauses computed by it (line 9). Let $W_{\mathrm{alg}} = \{(i, j, \sigma_i, \sigma_j) | (t_{\sigma_i\sigma_j}, i, j, \sigma_i, \sigma_j) \in \texttt{WgPaths}\}$ s.t. $P_{\mathrm{alg}} = \bigwedge_{(i,j,\sigma_i,\sigma_j)\in W_{\mathrm{alg}}} [(\sigma(i) \neq \sigma_i) \vee (\sigma(j) \neq \sigma_j)]$. Then by construction of P_{alg}, we have the following properties:

1. For all $\varepsilon > 0$, $\mathrm{WP}(D_{\mathrm{alg}}) \subset W_{\mathrm{alg}} \subset \mathrm{WP}(D_{\mathrm{alg}-\varepsilon})$, because we add paths by decreasing value of makespan (line 4).
2. There exists $(D_{\mathrm{alg}}, i_0, j_0, \sigma_{i_0}, \sigma_{j_0}) \in \texttt{WgPaths}$ such that P_{alg} is satisfiable and $P_{\mathrm{alg}} \bigwedge [(\sigma(i_0) \neq \sigma_{i_0}) \vee (\sigma(j_0) \neq \sigma_{j_0})]$ is not satisfiable. This is the stopping condition on line 6.

We show the optimality of Algorithm 1 by contradiction. If it is not optimal, then $D_{\mathrm{opt}} < D_{\mathrm{alg}}$, and $W_{\mathrm{alg}} \cup (i_0, j_0, \sigma_{i_0}, \sigma_{j_0}) \subset \mathrm{WP}(D_{\mathrm{opt}})$. Furthermore, according to Lemma 3, $P_{D_{\mathrm{opt}}}(\sigma_{\mathrm{opt}})$ is satisfied, hence σ_{opt} is also a solution to $P_{\mathrm{alg}} \bigwedge [(\sigma(i_0) \neq \sigma_{i_0}) \vee (\sigma(j_0) \neq \sigma_{j_0})]$. This contradicts the fact that it does not admit a solution hence contradicting the non-optimality.

Finally, the complexity of MkWgPaths(V, E) is $\Theta(|E|)$. In Algorithm 1, we unwind the loop for (line 4) $4|E|$ times, and we verify if P_{tmp} is satisfiable in line 6 with a complexity of $\Theta(n + k)$ where k is the number of clauses is P_{tmp}. Since the number of iterations is bounded by $3|E|$, the complexity of Algorithm 1 is $\mathcal{O}(|E|^2)$.

Out-Tree Graphs. We assume now that the DAG $\mathcal{G} = (V, E)$ is an out-tree rooted by $r \in V$. For any task $u \in V$, the sub-tree rooted by u is the sub-graph $\mathcal{G}_u$ of $\mathcal{G}$ which vertices are u and the descendants of u.

For any task $u \in V$, let us denote by $D^{\mathcal{A}}(u)$ (resp. $D^{\mathcal{B}}(u)$) the lower bound of the minimal makespan of $\mathcal{G}_u$ assuming that $\sigma(u) = \mathcal{A}$ (resp. $\sigma(u) = \mathcal{B}$). Let us suppose that the arc $(u, v) \in E$. Observe that, if $D^{\mathcal{A}}(v) \leq c_{\mathcal{AB}}(u, v) + D^{\mathcal{B}}(v)$, then $D^{\mathcal{A}}(u) \geq t_{\mathcal{A}}(u) + D^{\mathcal{A}}(v)$. In the opposite, $D^{\mathcal{A}}(u) \geq t_{\mathcal{A}}(u) + c_{\mathcal{AB}}(u, v) + D^{\mathcal{B}}(v)$ and thus $D^{\mathcal{A}}(u) \geq t_{\mathcal{A}}(u) + \min(D^{\mathcal{A}}(v), c_{\mathcal{AB}}(u, v) + D^{\mathcal{B}}(v))$. Similarly, $D^{\mathcal{B}}(u) \geq t_{\mathcal{B}}(u) + \min(D^{\mathcal{B}}(v), c_{\mathcal{BA}}(u, v) + D^{\mathcal{A}}(v))$.

For any task $u \in V$, we set $\Gamma^+(u) = \{v \in V, (u, v) \in E\}$. For any allocation function σ, let $\bar{\sigma}(u) = \mathcal{A}$ if $\sigma(u) = \mathcal{B}$, $\bar{\sigma}(u) = \mathcal{B}$ otherwise. Then, for any task $u \in V$, we get $D^{\sigma(u)}(u) = t_{\sigma(u)}(u) + \max_{v\in\Gamma^+(u)} \min(D^{\sigma(u)}(v), c_{\sigma(u)\bar{\sigma}(u)} + D^{\bar{\sigma}(u)}(v))$.

Theorem 3. *For an out-tree graph $\mathcal{G} = (V, E)$ rooted by $r \in V$, an allocation σ may be built such that the corresponding schedule of length $D(r)$ verifies $D(r) = \min(D^{\mathcal{A}}(r), D^{\mathcal{B}}(r))$ and thus is optimal.*

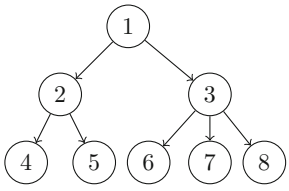
Proof. Let us suppose that lower bounds $D^{\mathcal{A}}(u)$ and $D^{\mathcal{B}}(u)$ for $u \in V$ are given. Let us define the allocation σ as $\sigma(r) = \mathcal{A}$ if $D^{\mathcal{A}}(r) \leq D^{\mathcal{B}}(r)$ and $\sigma(r) = \mathcal{B}$ in the opposite. For any task $v \neq r$ with $(u, v) \in E$, we set $\sigma(v) = \sigma(u)$ if $D^{\sigma(u)}(v) < D^{\bar{\sigma}(u)}(v) + c_{\sigma(u)\bar{\sigma}(u)}(u, v)$, and $\sigma(v) = \bar{\sigma}(u)$ otherwise.

For any task u, we prove that the length $D(u)$ of the schedule of $\mathcal{G}_u$ for the allocation σ verifies $D(u) = D^{\sigma(u)}(u)$. If u is a leaf, $D(u) = t_{\sigma(u)}(u) = D^{\sigma(u)}(u)$.

Now, let suppose that $\Gamma^+(u) \neq \emptyset$. By definition, for any arc $(u, v) \in E$, if $\sigma(u) = \sigma(v)$, $c_{\sigma(u)\sigma(v)}(u, v) = 0$. Then, if we set $\Delta^{\sigma}(u, v) = D(v) + c_{\sigma(u)\sigma(v)}(u, v)$, we get by induction $\Delta^{\sigma}(u, v) = D^{\sigma(v)}(v) + c_{\sigma(u)\sigma(v)}(u, v)$ and by definition of σ, $\Delta^{\sigma}(u, v) = \min(D^{\sigma(u)}(v), D^{\bar{\sigma}(u)}(v) + c_{\sigma(u)\bar{\sigma}(u)}(u, v))$. Now, $D(u) = t_{\sigma(u)}(u) + \max_{v \in \Gamma^+(u)} \Delta^{\sigma}(u, v)$ and thus by definition of $D^{\sigma(u)}$, $D(u) = D^{\sigma(u)}$, which concludes the proof.

A polynomial time algorithm of time complexity $\Theta(n)$ can be deduced by computing first $D^{\mathcal{A}}$, $D^{\mathcal{B}}$ and then σ.

Example 1. Let us consider as example the out-tree pictured by Fig. 2. Figure 3 shows the lower bound $D^{\mathcal{A}}$ and $D^{\mathcal{B}}$ and a corresponding optimal schedule.



$v \in V$	$t_{\mathcal{A}}(v)$	$t_{\mathcal{B}}(v)$
1	2	3
2	3	2
3	4	2
4	5	2
5	3	3
6	5	1
7	2	5
8	3	1

$a \in E$	$c_{\mathcal{AB}}(a)$	$c_{\mathcal{BA}}(a)$
$(3, 8)$	2	3
$(3, 7)$	3	3
$(3, 6)$	3	4
$(2, 4)$	2	2
$(2, 5)$	3	2
$(1, 2)$	1	4
$(1, 3)$	4	3

Fig. 2. An out-tree $\mathcal{G}$, duration of tasks and communication costs.

Series-Parallel Graphs. Let us consider a two terminal Series Parallel digraph ($2SP$ in short) as defined in [14,15]. Each element of this class has a unique source s and a unique sink t with $s \neq t$. It is formally defined as follows where $\mathcal{G}$ and $\mathcal{H}$ are two $2SP$ graphs.

- The arc $(s, t) \in 2SP$;
- The series composition of $\mathcal{G}$ and $\mathcal{H}$ is denoted by $\mathcal{G}.\mathcal{H}$ and is built by identifying the sink of $\mathcal{G}$ with the source of $\mathcal{H}$;
- The parallel composition is denoted by $\mathcal{G} + \mathcal{H}$ and identifies respectively the sinks and the sources of the two digraphs.

Figure 4 pictures a $2SP$ graph and its associated decomposition tree.

$v \in V$	$D^{\mathcal{A}}(u)$	$D^{\mathcal{B}}(u)$
1	10	10
2	7	5
3	8	7
4	5	2
5	3	3
6	5	1
7	2	5
8	3	1

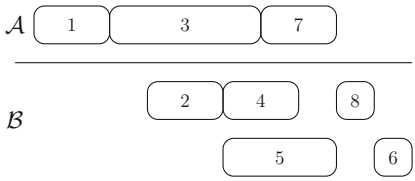


Fig. 3. Lower bounds $D^{\mathcal{A}}$ and $D^{\mathcal{B}}$. An optimal schedule is presented for the allocation $\sigma(1) = \mathcal{A}$, $\sigma(2) = \mathcal{B}$, $\sigma(3) = \mathcal{A}$, $\sigma(4) = \mathcal{B}$, $\sigma(5) = \mathcal{B}$, $\sigma(6) = \mathcal{B}$, $\sigma(7) = \mathcal{A}$ and $\sigma(8) = \mathcal{B}$.

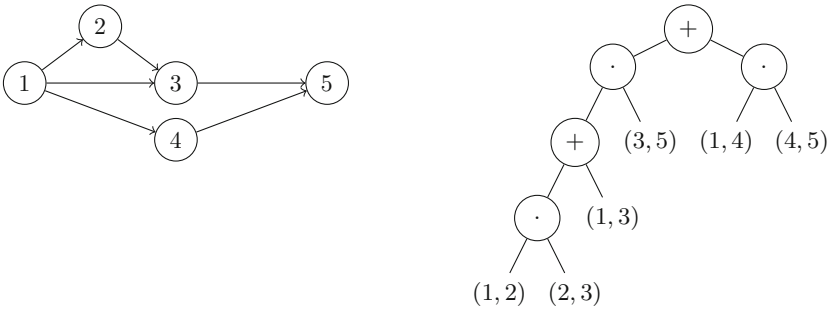


Fig. 4. A $2SP$ graph and its associated decomposition tree. Leaves correspond to arcs, while internal nodes are series or parallel compositions.

For any element $\mathcal{G} \in 2SP$ with a source s and a sink t and for any couple $(\alpha, \beta) \in \{\mathcal{A}, \mathcal{B}\}^2$, let us denote by $D^{\alpha\beta}(\mathcal{G})$ a lower bound defined as follows of the minimum length of a schedule of $\mathcal{G}$ with $\sigma(s) = \alpha$ and $\sigma(t) = \beta$. For any graph $\mathcal{G}$ with a unique arc $e = (s, t)$, for any couple $(\alpha, \beta) \in \{\mathcal{A}, \mathcal{B}\}^2$,

$$D^{\alpha\beta}(\mathcal{G}) = \begin{cases} t_\alpha(s) + t_\beta(t) + c_{\alpha\beta}(s,t) \text{ if } \alpha \neq \beta \\ t_\alpha(s) + t_\beta(t) \qquad\qquad\quad \text{otherwise.} \end{cases}$$

Now, if $\mathcal{G}$ and $\mathcal{H}$ are two $2SP$, then for the series composition, we set $D^{\alpha\beta}(\mathcal{G}.\mathcal{H}) = \min_{\gamma\in\{\mathcal{A},\mathcal{B}\}}(D^{\alpha\gamma}(\mathcal{G}) + D^{\gamma\beta}(\mathcal{H}) - t_\gamma(t))$ where t is the sink of $\mathcal{G}$. Similarly, for the parallel composition, we set $D^{\alpha\beta}(\mathcal{G} + \mathcal{H}) = \max(D^{\alpha\beta}(\mathcal{G}), D^{\alpha\beta}(\mathcal{H}))$.

We define the allocation function σ associated with a $2SP$ graph $\mathcal{G}$ and the corresponding length $D(\mathcal{G})$ as follows. We set $D(\mathcal{G}) = \min_{(\alpha,\beta)\in\{\mathcal{A},\mathcal{B}\}^2}(D^{\alpha\beta}(\mathcal{G}))$. We also set $\sigma(s)$ and $\sigma(t)$ the allocation function of the source and the sink of $\mathcal{G}$ as $D(\mathcal{G}) = D^{\sigma(s)\sigma(t)}(\mathcal{G})$. Now, for any series composition, let us suppose that s and t (resp. s' and t') are the source and the sink of $\mathcal{G}$ (resp. $\mathcal{H}$). We also suppose that $\sigma(s)$ and $\sigma(t')$ are fixed. Then, for $\mathcal{G}.\mathcal{H}$, $t = s'$ and we get $\sigma(t) = \gamma \in \{\mathcal{A}, \mathcal{B}\}$ such that $D(\mathcal{G}.\mathcal{H}) = D^{\sigma(s)\sigma(t)}(\mathcal{G}) + D^{\sigma(s')\sigma(t')}(\mathcal{H}) - t_{\sigma(t)}(t)$.

If $\mathcal{G}$ is a $2SP$ graph of source s and sink t, any vertex $v \in V - \{s, t\}$ is involved in a series composition, and thus σ is completely defined.

Theorem 4. *For any 2SP graph $\mathcal{G}$ of source s and sink t, $D(\mathcal{G}) = D^{\sigma(s)\sigma(t)}(\mathcal{G})$.*

Proof. The equality is clearly true if $\mathcal{G}$ is an arc (s,t). Indeed, we get in this case $D(\mathcal{G}) = \min_{(\alpha,\beta)\in\{\mathcal{A},\mathcal{B}\}^2}(D^{\alpha\beta}(\mathcal{G})) = D^{\sigma(s)\sigma(t)}(\mathcal{G})$.

Now, let us suppose that s and t (resp. s' and t') are the source and the sink of $\mathcal{G}$ (resp. $\mathcal{H}$) and that $D(\mathcal{G}) = D^{\sigma(s)\sigma(t)}(\mathcal{G})$ and $D(\mathcal{H}) = D^{\sigma(s')\sigma(t')}(\mathcal{H})$. For a parallel composition, $D(\mathcal{G}+\mathcal{H}) = \max(D^{\sigma(s)\sigma(t)}(\mathcal{G}), D^{\sigma(s')\sigma(t')}(\mathcal{H})) = D^{\sigma(s)\sigma(t)}(\mathcal{G}+\mathcal{H})$ as $s = s'$ and $t = t'$.

For the series composition, $D(\mathcal{G}.\mathcal{H}) = D(\mathcal{G}) + D(\mathcal{H}) - t_{\sigma(t)}(t) = D^{\sigma(s)\sigma(t)}(\mathcal{G}.\mathcal{H})$, since $t = s'$, which concludes the proof.

Corollary 2. *A polynomial-time algorithm of time complexity $\Theta(|E|)$ can be deduced by computing lower bounds $D^{\alpha\beta}$, $(\alpha,\beta) \in \{A,B\}^2$ for each graph issued from the decomposition of G and a corresponding allocation σ.*

4 Future Directions

With this work we have studied the problem of scheduling a Directed Acyclic Graph on an unbounded hybrid platform. Specifically our platform consists of two machines, each with an unbounded number of resources. Moving data from one machine to the other one has a communication cost. We have shown the intractability of the problem by reducing this problem to the 3-satisfiability problem. We have shown that there does not exist 3/2-approximation algorithms unless P=NP. We have further provided some polynomial time algorithms for special cases of graphs. While this model seems very theoretical, we can see several applications both in High-Performance Computing (*In-Situ analysis*) and in Big Data analytics in the cloud (Geo-distributed data-Centers).

There are several extensions that we can see to this work. In the context of two unbounded platforms, it would be interesting to find some polynomial time algorithms with proven bounds to the optimal. We do not expect to be able to find one in the general case, but we hope that with some constraints between the communication costs and computation cost (as is often done in the context of scheduling DAGs with communications), one may able to find such algorithms. We plan then to evaluate these algorithms with *In-Situ* frameworks. Finally, another direction we are interested by is a version of this problem where only one machine has an unbounded number of resources, and where the data is located on the other one. For example in the context of smartphone applications, we can model the frontend/backend context where the phone (Machine 1) has a limited number of available processors, but can rely on sending some of the computation on a backend machine (cloud-based), with an unbounded number of processors. Similarly to here, the problem is a data and communication problem: given the cost to transfer data from one machine to the other one, what is the most efficient strategy.

References

1. Barthou, D., Jeannot, E.: SPAGHETtI: scheduling/placement approach for task-graphs on HETerogeneous archItecture. In: Silva, F., Dutra, I., Santos Costa, V. (eds.) Euro-Par 2014. LNCS, vol. 8632, pp. 174–185. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09873-9_15
2. Dorier, M., Dreher, M., Peterka, T., Wozniak, J.M., Antoniu, G., Raffin, B.: Lessons learned from building in situ coupling frameworks. In: Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, pp. 19–24. ACM (2015)
3. Zhou, A.C., Ibrahim, S., He, B.: On achieving efficient data transfer for graph processing in geo-distributed datacenters. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pp. 1397–1407 (2017)
4. Ait Aba, M., Aupy, G., Munier-Kordon, A.: Scheduling on two unbounded resources with communication costs. Inria, Research Report RR-9264, Mar 2019. https://hal.inria.fr/hal-02076473
5. Ait Aba, M., Zaourar, L., Munier, A.: Approximation algorithm for scheduling applications on hybrid multi-core machines with communications delays. In: IPDPSW. IEEE (2018)
6. Kedad-Sidhoum, S., Monna, F., Mounié, G., Trystram, D.: A family of scheduling algorithms for hybrid parallel platforms. Int. J. Found. Comput. Sci. **29**(01), 63–90 (2018)
7. Marchal, L., Canon, L.-C., Vivien, F.: Low-cost approximation algorithms for scheduling independent tasks on hybrid platforms. Ph.D. dissertation, Inria-Research Centre Grenoble-Rhône-Alpes (2017)
8. Darbha, S., Agrawal, D.P.: Optimal scheduling algorithm for distributed-memory machines. IEEE Trans. Parallel Distrib. Syst. **9**(1), 87–95 (1998)
9. Park, C.-I., Choe, T.-Y.: An optimal scheduling algorithm based on task duplication. In: 2001 Proceedings, Eighth International Conference on Parallel and Distributed Systems, ICPADS 2001, pp. 9–14. IEEE (2001)
10. Boeres, C., Rebello, V.E., et al.: A cluster-based strategy for scheduling task on heterogeneous processors. In: 2004 16th Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2004, pp. 214–221. IEEE (2004)
11. Topcuoglu, 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)
12. Garey, M.R., Johnson, D.S.: Complexity results for multiprocessor scheduling under resource constraints. SIAM J. Comput. **4**(4), 397–411 (1975)
13. Karp, R.M.: Reducibility among combinatorial problems. Complexity of Computer Computations, pp. 85–103. Springer, Boston (1972). https://doi.org/10.1007/978-1-4684-2001-2_9
14. Valdes, J., Tarjan, R., Lawler, E.: The recognition of series parallel digraphs. SIAM J. Comput. **11**(2), 298–313 (1982). https://doi.org/10.1137/0211023
15. Schoenmakers, B.: A new algorithm for the recognition of series parallel graphs, series, CWI report. CS-R, CWI (1995)