Parallel and Distributed Algorithms and Programs
TD n°5 - Scheduling (1)

Etienne Mauffret Anthony Dugois

16/12/2021

Part 1
Scheduling without communication costs

1.1
Preliminaries

Definition 1 (ρ-approximation) *Let $\mathcal{P}$ be an optimisation problem with integer objective function $f_\mathcal{P}$. Writing $OPT(I)$ for an optimal solution of the problem $\mathcal{P}$ on instance I, we say a polynomial algorithm A is a ρ-approximation for the problem $\mathcal{P}$ if and only if $\forall I : f_\mathcal{P}(A(I)) \leq \rho \cdot f_\mathcal{P}(OPT(I))$.*

Theorem 1 (Impossibility theorem) *Let $\mathcal{P}$ be an optimisation problem with integer objective function $f_\mathcal{P}$ and c be a nonnegative integer. If the decision problem associated to $\mathcal{P}$ and value c (namely, "does there exist a solution x such that $f_\mathcal{P}(x) \leq c$?") is NP-complete, then, the existence of a ρ-approximation of $\mathcal{P}$ for any $\rho < (c+1)/c$ implies $P = NP$.*

Question 1

a) Prove the impossibility theorem.

Solution 1

a) Suppose that the decision problem associated to $\mathcal{P}$ and c is NP-complete. Suppose that there exists a polynomial algorithm A that is a ρ-approximation for $\mathcal{P}$ such that $\rho < (c+1)/c$. Let I be an instance of $\mathcal{P}$. We consider two cases:

   (i) $f_\mathcal{P}(OPT(I)) > c$. Then $f_\mathcal{P}(A(I)) > c$ because $f_\mathcal{P}(A(I)) > f_\mathcal{P}(OPT(I))$ (by definition).

   (ii) $f_\mathcal{P}(OPT(I)) \leq c$. We have
   $$f_\mathcal{P}(A(I)) \leq \rho \cdot f_\mathcal{P}(OPT(I)) < \frac{c+1}{c} \cdot f_\mathcal{P}(OPT(I)),$$
   thus, $f_\mathcal{P}(A(I)) < c+1$. Therefore, $f_\mathcal{P}(A(I)) \leq c$.

   We proved that

   (i) $f_\mathcal{P}(OPT(I)) > c$ implies $f_\mathcal{P}(A(I)) > c$;

   (ii) $f_\mathcal{P}(OPT(I)) \leq c$ implies $f_\mathcal{P}(A(I)) \leq c$;

   which is equivalent (by contraposition) to

   (i) $f_\mathcal{P}(A(I)) \leq c$ implies $f_\mathcal{P}(OPT(I)) \leq c$;

   (ii) $f_\mathcal{P}(A(I)) > c$ implies $f_\mathcal{P}(OPT(I)) > c$.

   Therefore, A, which is polynomial, gives the answer to the NP-complete decision problem associated to $\mathcal{P}$ and c. This implies $P = NP$.

Let us recall two classical NP-complete problems which we are going to use in the tutorial:

Definition 2 (2-Partition) *Given a set $\mathcal{A}$ of n integers $a_1, \ldots, a_n$, find a partition of $\mathcal{A}$, i.e., two subsets $\mathcal{I}_1$ and $\mathcal{I}_2$ of indices $\{1, \ldots, n\}$, such that*
$$\sum_{i \in \mathcal{I}_1} a_i = \sum_{i \in \mathcal{I}_2} a_i = \frac{1}{2} \sum_{i=1}^{n} a_i.$$

Definition 3 (Clique) *Given a graph $G = (V, E)$ and an integer k, find a subset $C \subseteq V$ of size k such that for every $u, v \in C, u \neq v, (u, v) \in E$.*

1.2

Independent tasks of various durations

If tasks are identical and independent, scheduling can obviously be done in polynomial time. However, if durations of the tasks are allowed to be different, the problem becomes NP-hard. Yet there exists a 4/3-approximation for the scheduling problem, improving on the general result for generic list algorithms (which are always 2-approximations).

Suppose we have p identical machines and n independent tasks $\{T_i\}_{1 \leq i \leq n}$. We seek a schedule σ mapping each task T_i to a machine $\mu(T_i)$ and a starting time $\tau(T_i)$, knowing that T_i takes time $w(T_i)$ to be executed. Ideally, this schedule should minimize the total running time $D(\sigma) = \max_{1 \leq i \leq n}(\tau(T_i) + w(T_i))$.

Question 2

a) Assuming $D_{OPT} < 3w(T_i)$ for every i, show that $n \leq 2p$ and give a polynomial-time algorithm to compute an optimal schedule.

b) Let us consider the following list algorithm: whenever a machine is free, we assign it the longest task available. Call σ the induced schedule; check the following bound:

$$D(\sigma) \leq D_{OPT} + \left(\frac{p-1}{p}\right) d,$$

where d denotes the duration of a(ny) task ending at instant $D(\sigma)$. Then, using the previous question, deduce:

$$D_{OPT} \leq D(\sigma) \leq \left(\frac{4}{3} - \frac{1}{3p}\right) D_{OPT}.$$

Solution 2

a) Suppose that there is an optimal schedule OPT such that $D_{OPT} < 3w(T_i)$ for all i. Let $w_{\min}$ denote the minimum execution time among all tasks. Therefore, $D_{OPT} < 3w_{\min}$, which means that the optimal schedule can execute at most 2 tasks on each machine (as the shortest task can be executed at most 2 times on one machine). We conclude that $n \leq 2p$.

Let h $(0 \leq h \leq p)$ be the number of tasks that are alone on a machine in the schedule OPT. This means that h machines are taken by these tasks, and there are at most $2(p-h)$ tasks on the $p-h$ remaining machines. We will show that we can transform OPT into another optimal schedule OPT', respecting the following rules, without changing the running time:

- the h longest tasks are alone on h machines;
- the $2(p-h)$ next tasks are grouped by two (the longest with the shortest, the second longest with the second shortest, etc) and executed on the $p-h$ remaining machines.

h tasks are alone on a machine in OPT. We can simply replace these tasks by the h longest without increasing the total running time (because $D_{OPT} \geq \max w_i$).

Now let us consider 2 machines P_i, P_j executing 2 tasks each in OPT. Suppose that machine P_i executes tasks T_{i_1}, T_{i_2} (in any order) and machine P_j executes tasks T_{j_1}, T_{j_2} (also in any order), such that $w(T_{i_1}) \geq w(T_{i_2})$, $w(T_{j_1}) \geq w(T_{j_2})$, $w(T_{i_1}) \geq w(T_{j_1})$ and $w(T_{i_2}) \geq w(T_{j_2})$. We can swap T_{i_2} and T_{j_2} without increasing the total running time, because $w(T_{i_1}) + w(T_{j_2}) \leq w(T_{i_1}) + w(T_{i_2})$, and $w(T_{j_1}) + w(T_{i_2}) \leq w(T_{i_1}) + w(T_{i_2})$.

Thus, by swapping tasks repeatedly, we get an optimal schedule OPT' that follows the previously described algorithm, which is polynomial. Note that, on this instance, this algorithm is strictly equivalent to sorting tasks by non-increasing order of execution times, and executing them one by one in-order as soon as a machine is free.

b) Let T_j be the task that finishes the last, i.e., $D(\sigma) = s + d$, where $s = \tau(T_j)$ and $d = w(T_j)$. All machines must complete after time s, otherwise T_j would have been started before s (according to the considered algorithm). Therefore, for each machine P_k, the sum of execution times of all tasks (except T_j) processed by P_k is greater than s, and we have

$$\sum_{i \neq j} w(T_i) \geq ps = p(D(\sigma) - d).$$

Hence,

$$D(\sigma) \leq \frac{1}{p} \sum_{i \neq j} w(T_i) + d,$$

which implies

$$D(\sigma) \leq \frac{1}{p}\left(\sum_i w(T_i) - d\right) + d = \frac{1}{p}\sum_i w(T_i) + \left(\frac{p-1}{p}\right) d \leq D_{OPT} + \left(\frac{p-1}{p}\right) d, \tag{1.2.1}$$

because $D_{OPT} \geq \frac{1}{p}\sum_i w(T_i)$.

Now suppose by contradiction that $D(\sigma) > \left(\frac{4}{3} - \frac{1}{3p}\right) D_{OPT}$. Then, according to (1.2.1), we have

$$D_{OPT} + \left(\frac{p-1}{p}\right) d > \left(\frac{4}{3} - \frac{1}{3p}\right) D_{OPT},$$

thus $D_{OPT} < 3d$. There are two cases:

(i) The last task T_j is the shortest task, which means that $D_{OPT} < 3w(T_i)$ for all i. We proved in the previous question that the proposed algorithm is optimal in this case. So $D(\sigma) = D_{OPT}$, which contradicts our hypothesis $D(\sigma) > \left(\frac{4}{3} - \frac{1}{3p}\right) D_{OPT}$.

(ii) T_j is not the shortest task in σ. We consider the subschedule σ' by removing all tasks shorter than T_j from σ. In σ, these shorter tasks necessarily start after s (according to the considered algorithm) and finish before $s + d$ (because T_j is the task that finishes the last), therefore $D(\sigma') = D(\sigma)$ and $D_{OPT'} \leq D_{OPT}$ (where $D_{OPT'}$ is the optimal schedule restricted to tasks in σ'). Hence, as we supposed that $D(\sigma) > \left(\frac{4}{3} - \frac{1}{3p}\right) D_{OPT}$, we have $D(\sigma') > \left(\frac{4}{3} - \frac{1}{3p}\right) D_{OPT'}$.

Moreover, $D_{OPT'} < 3d$, and as T_j is the shortest task in σ', we have $D_{OPT'} < 3w(T_i)$ for all i in σ'. The situation is similar to case (i), therefore σ' is optimal (i.e., $D(\sigma') = D_{OPT'}$), which contradicts the fact that $D(\sigma') > \left(\frac{4}{3} - \frac{1}{3p}\right) D_{OPT'}$.

We conclude that $D(\sigma) \leq \left(\frac{4}{3} - \frac{1}{3p}\right) D_{OPT}$.

1.3 Identical tasks with dependencies

Now we want to schedule n tasks $\{T_i\}_{1 \leq i \leq n}$ requiring one step of execution while respecting dependency constraints given by an order $\prec$ with p identical processors.

Question 3

a) Show that deciding the existence of a schedule with makespan 3 is an NP-complete problem. (Hint: use clique. You should try to build a set of jobs corresponding to vertices and edges. Fake jobs may also be used. Add dependencies between jobs such that a schedule of length 3 is doable if and only if you can process $\frac{1}{2}k(k-1)$ edges at time 1, while processing at most k vertices at time 0.)

b) What can you deduce on the existence of good approximation algorithms for this problem?

Solution 3

a) The decision problem related to our scheduling problem is clearly in NP. Now we prove its NP-hardness. Let $G = (V, E)$ be a graph and k an arbitrary positive integer. We first show that we can construct an instance of our scheduling problem from G and k in polynomial time.

Building instance. Let $\bar{k} = |V| - k$ (to be used as number of vertices that will not be in the clique), $l = \frac{1}{2}k(k-1)$ (to be used as number of edges that will be in the clique) and $\bar{l} = |E| - l$ (to be used as number of edges that will not be in the clique).

We consider $p = \max\left(k, \bar{k} + l, \bar{l}\right)$ machines and $n = 3p$ unit tasks.

We associate a task J_v to each vertex $v \in V$ and a task K_e to each edge $e \in E$. If two vertices u and v are linked by an edge e in G, we set $J_u \prec K_e$ and $J_v \prec K_e$. Moreover, we create $p - k$ filling tasks $\mathcal{X} = \{X_i\}_{1 \leq i \leq p-k}$, $p - \bar{k} - l$ filling tasks $\mathcal{Y} = \{Y_i\}_{1 \leq i \leq p - \bar{k} - l}$, and $p - \bar{l}$ filling tasks $\mathcal{Z} = \{Z_i\}_{1 \leq i \leq p - \bar{l}}$, such that all tasks $\mathcal{X}$ must be scheduled before tasks $\mathcal{Y}$, and tasks $\mathcal{Y}$ must be scheduled before tasks $\mathcal{Z}$.

Now we show that there exists a clique of size k in G if and only if there exists a schedule with makespan 3 for the built instance.

Equivalence of problems.

$\boxed{\Rightarrow}$ Suppose there is a clique of size k in G.

Then, at time 0, we schedule the k tasks among tasks $\{J_v\}$ that correspond to vertices in the clique. We also schedule all tasks $\mathcal{X}$ at time 0 on the $p-k$ remaining machines.

Then, at time 1, we schedule the l tasks among tasks $\{K_e\}$ that correspond to edges in the clique, and the $\bar{k}$ tasks among tasks $\{J_v\}$ that correspond to vertices not in the clique. We also schedule all tasks $\mathcal{Y}$ at time 1 on the $p-\bar{k}-l$ remaining machines.

Finally, at time 2, we schedule all remaining tasks among tasks $\{K_e\}$ (that correspond to the $\bar{l}$ edges not in the clique), and we also schedule all tasks $\mathcal{Z}$ on the $p-\bar{l}$ remaining machines. These tasks will necessarily finish at time 3.

$\boxed{\Leftarrow}$ Suppose there is a schedule σ with makespan 3.

Then, by dependencies between tasks $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$, we know that tasks $\mathcal{X}$ are necessarily scheduled at time 0, tasks $\mathcal{Y}$ are scheduled at time 1, and tasks $\mathcal{Z}$ are scheduled at time 2.

There are k remaining slots at time 0. By dependencies between tasks $\{J_v\}$ and $\{K_e\}$, we know that the k tasks scheduled at time 0 are tasks among $\{J_v\}$. There are $\bar{k}+l$ remaining slots at time 1. As all tasks are scheduled before time 3, we know that the $\bar{k}$ remaining tasks among tasks $\{J_v\}$ are necessarily scheduled at time 1. So there must be l tasks among tasks $\{K_e\}$ scheduled at time 1.

Each of these tasks corresponds to an edge linking 2 vertices represented by 2 tasks scheduled at time 0. There are $l=\frac{1}{2}k(k-1)$ such edge tasks, and k such vertex tasks. As all edges are distinct, this means that the subgraph induced by the k vertices and l edges is complete.

Therefore, there exists a clique of size k in G.

b) From the previous question, we can deduce by the impossibility theorem that there is no ρ-approximation algorithm such that $\rho < \frac{4}{3}$ unless $P = NP$.

Part 2

Scheduling with communications

2.1

Scheduling of a FORK graph (with communications)

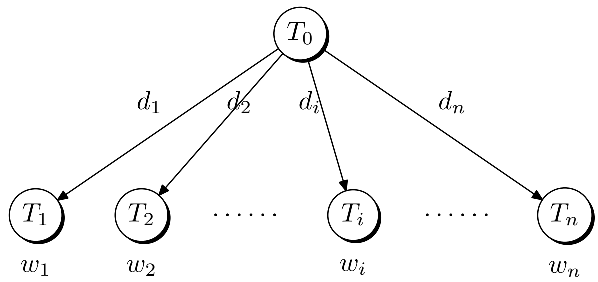

Figure 1: FORK graph with n children

Definition 4 (FORK with n children) *A FORK graph with n children is a task graph made of $n+1$ vertices labelled by $T_0, T_1, \ldots, T_n$, as depicted in Figure 1. It has edges between the node T_0 and each of its children T_i, $1 \le i \le n$. Every node has a weight w_i representing the execution time of the task T_i. Each edge (T_0, T_i) also has a weight d_i corresponding to the communication cost. Communication costs are incurred only if T_0 and T_i are not scheduled on the same processor.*

We first assume that we have infinitely many processors which are multi-port (i.e., can send multiple messages at once). Let us define the following optimization problem:

Definition 5 (FORK-SCHED-$\infty(G)$) *Given a FORK graph G with n children and infinitely many processors, what is the schedule σ minimizing the running time?*

Question 4

a) Give a polynomial-time algorithm to solve FORK-SCHED-∞.

Solution 4

a) Let G be a FORK graph with n children. Obviously, T_0 must be scheduled first. Without loss of generality, let us schedule T_0 on machine P_0 at time 0. We have an infinite number of machines.

Suppose that 2 tasks T_i and T_j are scheduled in-order on the same machine P_k ($k > 0$) in an optimal schedule OPT. T_i will not start before time $w_0 + d_i$ and T_j must start after time $w_0 + \max(d_i + w_i, d_j)$ (either it waits for the completion of T_i, or it waits for its own communication time). Executing T_j alone on another machine $P_{k'}$ ($k' > 0, k' \neq k$) would not increase the completion time of T_i or T_j (the completion time of T_j can only decrease). Therefore, an optimal schedule consists in finding two subsets $\mathcal{T}_0$ and $\mathcal{T}_\infty$ such that tasks in $\mathcal{T}_0$ are scheduled on P_0 and each task of $\mathcal{T}_\infty$ is alone on a given machine.

Suppose that 2 tasks $T_i \in \mathcal{T}_0$ and $T_j \in \mathcal{T}_\infty$ are scheduled in OPT such that $d_i + w_i < d_j + w_j$. Clearly, we can move T_i on a different machine (such that T_i is alone) without increasing the total running time, because T_j would still necessarily complete after T_i.

Therefore, we can transform OPT in another optimal schedule OPT' such that, for any tasks $T_i \in \mathcal{T}_0$ and $T_j \in \mathcal{T}_\infty$, we have $d_i + w_i > d_j + w_j$.

Thus, it suffices to sort all tasks T_i in non-decreasing order of $d_i + w_i$, i.e., $d_1 + w_1 \leq d_2 + w_2 \leq \ldots \leq d_n + w_n$, and to find the number k of lonely tasks that minimizes

$$\max\left(\sum_{i=k}^{n} w_i, d_{k-1} + w_{k-1}\right),$$

where $\sum_{i=k}^{n} w_i$ is the total running time on P_0 and $d_{k-1} + w_{k-1}$ is the highest running time among lonely tasks. This is clearly feasible in polynomial-time.

We tackle the same problem with a bounded number of processors:

Definition 6 (FORK-SCHED-BOUNDED(G, p)) *Given a FORK graph G with n children and p processors, what is the schedule σ minimizing the running time?*

Question 5

a) Show that the associated decision problem is NP-complete.

Solution 5

a) The decision problem related to our scheduling problem is clearly in NP. Now we prove its NP-hardness. Let $\mathcal{A} = \{a_1, \ldots, a_n\}$ be a set of n positive integers. We first show that we can construct an instance of our scheduling problem from $\mathcal{A}$ in polynomial time.

Building instance. Let us build a FORK graph G with n children from $\mathcal{A}$:

- the first task T_0 has a null execution time ($w_0 = 0$);
- each child T_i ($1 \leq i \leq n$) has an execution time equal to a_i ($w_i = a_i$);
- all communication costs are null ($d_i = 0$ for all i).

Moreover, we consider only 2 machines. Obviously, we can build this instance in polynomial time.

Now we show that there exists a 2-Partition of $\mathcal{A}$ if and only if there is a schedule on 2 machines with makespan lower than or equal to $\frac{1}{2}\sum_{i=1}^{n} w_i$.

Equivalence of problems.

$\boxed{\Rightarrow}$ Suppose there exists a 2-Partition of $\mathcal{A}$. By definition, there are two subsets $\mathcal{I}_1$ and $\mathcal{I}_2$ of indices $\{1, \ldots, n\}$ such that $\sum_{i\in\mathcal{I}_1} a_i = \sum_{i\in\mathcal{I}_2} a_i = \frac{1}{2}\sum_{i=1}^{n} a_i$. For each $i \in \mathcal{I}_1$, we schedule T_i on the first machine, and for each $j \in \mathcal{I}_2$, we schedule T_j on the second machine. Hence, the makespan is $\max\left(\sum_{i\in\mathcal{I}_1} w_i, \sum_{i\in\mathcal{I}_2} w_i\right) = \frac{1}{2}\sum_{i=1}^{n} w_i$.

$\boxed{\Leftarrow}$ Suppose there exists a schedule on 2 machines with makespan lower than or equal to $\frac{1}{2}\sum_{i=1}^{n} w_i$. Then there are two subsets $\mathcal{I}_1$ and $\mathcal{I}_2$ such that tasks $\{T_i\}_{i\in\mathcal{I}_1}$ are scheduled on the first machine and tasks $\{T_i\}_{i\in\mathcal{I}_2}$ are scheduled on the second machine. We have $\max\left(\sum_{i\in\mathcal{I}_1} w_i, \sum_{i\in\mathcal{I}_2} w_i\right) \leq \frac{1}{2}\sum_{i=1}^{n} w_i$, i.e., $\sum_{i\in\mathcal{I}_1} w_i \leq \frac{1}{2}\sum_{i=1}^{n} w_i$ and $\sum_{i\in\mathcal{I}_2} w_i \leq \frac{1}{2}\sum_{i=1}^{n} w_i$. Moreover, $\sum_{i\in\mathcal{I}_1} w_i + \sum_{i\in\mathcal{I}_2} w_i = \sum_{i=1}^{n} w_i$. Hence, $\sum_{i\in\mathcal{I}_1} w_i = \sum_{i\in\mathcal{I}_2} w_i = \frac{1}{2}\sum_{i=1}^{n} w_i$; in other words, $\mathcal{I}_1$ and $\mathcal{I}_2$ give a 2-Partition of $\mathcal{A}$.

We come back to the problem with infinitely many identical processors, but we no longer suppose them to be multi-port: a processor can only communicate with a single peer at a time.

Definition 7 (FORK-SCHED-1-PORT-$\infty(G)$) *Given a FORK graph G with n children and infinitely many* 1-*port processors, what is the schedule σ minimizing the running time?*

Question 6

a) Show that the associated decision problem is NP-complete. (Hint: one can use 2-Partition-Eq, which is a variant of 2-Partition where both subsets are required to be of the same size.)