Approximation algorithms for energy, reliability and makespan optimization problems

Guillaume Aupy, Anne Benoit

Approximation algorithms for energy, reliability and makespan optimization problems

Guillaume Aupy, Anne Benoit

RESEARCH REPORT
N° 8107
October 2012
Project-Team ROMA

ISSN 0249-6399   ISRN INRIA/RR--8107--FR+ENG

Approximation algorithms for energy, reliability and makespan optimization problems

Guillaume Aupy[*†], Anne Benoit[*‡†]

Project-Team ROMA

Research Report n° 8107 — October 2012 — 29 pages

Abstract: In this paper, we consider the problem of scheduling an application on a parallel computational platform. The application is a particular task graph, either a linear chain of tasks, or a set of independent tasks. The platform is made of identical processors, whose speed can be dynamically modified. It is also subject to failures: if a processor is slowed down to decrease the energy consumption, it has a higher chance to fail. Therefore, the scheduling problem requires to re-execute or replicate tasks (i.e., execute twice a same task, either on the same processor, or on two distinct processors), in order to increase the reliability. It is a tri-criteria problem: the goal is to minimize the energy consumption, while enforcing a bound on the total execution time (the makespan), and a constraint on the reliability of each task.
Our main contribution is to propose approximation algorithms for these particular classes of task graphs. For linear chains, we design a fully polynomial time approximation scheme. However, we show that there exists no constant factor approximation algorithm for independent tasks, unless P=NP, and we are able in this case to propose an approximation algorithm with a relaxation on the makespan constraint.

Key-words: Scheduling, energy, reliability, makespan, models, approximation algorithms

[*] LIP, École Normale Supérieure de Lyon, France
[†] INRIA
[‡] Institut Universitaire de France

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Algorithmes d'approximation pour des problèmes d'optimisation énergie/fiabilité/temps d'exécution

Résumé : Dans ce papier, nous considérons le problème d'ordonnancement d'une application sur une plateforme parallèle de calcul. L'application est un graphe de tâches particulier: soit une chaîne de tâche, soit un ensemble de tâches indépendantes. La plateforme est constituée de processeurs identiques, dont la vitesse peut être modifiée dynamiquement. Cette plateforme est aussi sujette à des fautes: lorsque l'on réduit la vitesse d'exécution d'un processeur pour diminuer la consommation d'énergie, ce processeur a une plus grande chance de faillir. C'est pourquoi, pour augmenter la fiabilité du processus, l'ordonnanceur va devoir choisir de re-exécuter ou répliquer certaines tâches (les exécuter deux fois, soit sur le même processeur, soit sur deux processeurs distincts). Le problème est donc tri-critère: nous cherchons à minimiser la consommation d'énergie, tout en préservant une limite sur le temps d'exécution, ainsi qu'une borne sur la fiabilité de chaque tâche.

Nos contributions résident en l'écriture d'algorithmes d'approximation efficaces pour les deux classes de graphes étudiées. Dans le cas des chaînes linéaires, nous proposons un schéma d'approximation entièrement polynomial (FPTAS). Puis nous prouvons qu'il n'existe pas d'algorithmes d'approximation à facteur constant dans le cas des tâches indépendantes, sauf si P=NP, mais nous sommes cependant capable d'exhiber un algorithme d'approximation lorsque l'on autorise une relaxation de la contrainte sur le temps d'exécution.

Mots-clés : Ordonnancement, énergie, fiabilité, temps d'exécution, modèles, algorithmes d'approximation

1 Introduction

Energy-awareness is now recognized as a first-class constraint in the design of new scheduling algorithms. To help reduce energy dissipation, current processors from AMD, Intel and Transmetta allow the speed to be set dynamically, using a dynamic voltage and frequency scaling technique (DVFS). Indeed, a processor running at speed s dissipates s^3 watts per unit of time [6]. However, it has been recognized that reducing the speed of a processor has a negative effect on the reliability of a schedule: if a processor is slowed down, it has a higher chance to be subject to transient failures, caused for instance by software errors [20, 11].

Motivated by the application of speed scaling on large scale machines [15], we consider a tri-criteria problem energy/reliability/makespan: the goal is to minimize the energy consumption, while enforcing a bound on the makespan, i.e., the total execution time, and a constraint on the reliability of each task. The application is a particular task graph, either a linear chain of tasks, or a set of independent tasks. The platform is made of identical processors, whose speed can be dynamically modified.

In order to make up for the loss in reliability due to the energy efficiency, we consider two standard techniques: *re-execution* consists in re-executing a task twice on a same processor [20, 19], while *replication* consists in executing a same task on two distinct processors simultaneously [2]. We do not consider *checkpointing*, which consists in "saving" the work done at some points, hence reducing the amount of work lost when a failure occurs [14, 18].

The schedule therefore requires to (i) decide which tasks are re-executed or replicated; (ii) decide on which processor(s) each task is executed; (iii) decide at which speed each processor is processing each task. For a given schedule, we can compute the total execution time, also called *makespan*, and it should not exceed a prescribed deadline. Each task has a reliability that can be computed given its execution speed and its eventual replication or re-execution, and we must enforce that the execution of each task is reliable enough. Finally, we aim at minimizing the energy consumption. Note that we consider a set of homogeneous processors, but each processor may run at a different speed; this corresponds to typical current platforms with DVFS.

Related work. The problem of minimizing the energy consumption without exceeding a given deadline, using DVFS, has been widely studied, without accounting for reliability issues. The problem for a linear chain of tasks is known to be solvable in polynomial time in this case, see [3]. [1] showed that the problem of scheduling independent tasks can be approximated by a factor $(1 + \varepsilon)$: they exhibit a polynomial time approximation scheme (PTAS). [9] studied the performance of greedy algorithms for the problem of scheduling independent tasks, with the objective of minimizing the energy consumption, and proposed some approximation algorithms.

All these work do not account for reliability issues. However, [20] showed that reducing the speed of a processor increases the number of transient failure rates of the system; the probability of failures increases exponentially, and this probability cannot be neglected in large-scale computing [15]. Few authors have tackled the tri-criteria problem including reliability, and to the best of our knowledge, there are no approximation algorithms for this problem. [19] initiated the study of this problem, using re-execution. However, they restrict their study to the scheduling problem on a single processor, and do not try to find any approximation ratio on their algorithm. [2] have recently proposed an off-line tri-criteria scheduling heuristic (TSH), which uses replication to minimize the makespan, with a threshold on the global failure rate and the maximum power consumption. TSH is an improved critical-path list scheduling heuristic that takes into account power and reliability before deciding which task to assign and to replicate onto the next free processors. However, the complexity of this heuristic is unfortunately exponential in the number of processors, and the authors did not try to give an approximation ratio on their heuristic. Finally, [4] also study the tri-criteria problem, but from an heuristic point of view, without trying to ensure any approximation ratio on their heuristics. Moreover, they do not consider replication of tasks, but only re-execution as in [19]. However, they present a formal model of the tri-criteria problem, re-used in this paper.

Finally, there is some related work specific to the problem of independent tasks, since several approximation algorithms have been proposed for variants of the problem. One may try to minimize the ℓ_k norm, i.e., the quantity $(\sum_{q=1}^{p}(\sum_{i \in load(q)} a_i)^k)^{1/k}$, with p processors, where $i \in load(q)$ means that task T_i is assigned to processor q, and a_i is the weight of task T_i [1]. Minimizing the power consumption then amounts to minimize the ℓ_3 norm [9], and the problem of makespan minimization is equivalent to minimizing the ℓ_∞ norm, i.e., minimize $\max_{1 \leq q \leq p} \sum_{i \in load(q)} a_i$ [13, 5]. These problems are typical *load balancing* problems, in which the load (computation requirement of the tasks) must be balanced between processors, according to various criteria.

Main contributions. In this paper, we investigate the tri-criteria problem of minimizing the energy with a bound on the makespan and a constraint on the reliability. First in Section 2, we formally introduce this tri-criteria scheduling problem, based on the previous models proposed by [19] and [4]. To the best of our knowledge, this is the first model including both re-execution and replication in order to deal with failures. The main contribution of this paper is then to provide approximation algorithms for some particular instances of this tri-criteria problem.

For linear chains of tasks, we propose a fully polynomial time approximation scheme (Section 3). Then in Section 4, we show that there exists no constant factor approximation algorithm for the tri-criteria problem with independent tasks, unless P=NP. We prove that by relaxing the constraint on the makespan, we are able to give a polynomial time constant factor approximation algorithm. To the best of our knowledge, these are the first approximation algorithms for the tri-criteria problem.

2 Framework

Consider an application task graph $\mathcal{G} = (V, \mathcal{E})$, where $V = \{T_1, T_2, \dots, T_n\}$ is the set of tasks, $n = |V|$, and where $\mathcal{E}$ is the set of precedence edges between tasks. For $1 \leq i \leq n$, task T_i has a weight w_i, that corresponds to the computation requirement of the task. $S = \sum_{i=1}^{n} w_i$ is the sum of the computation requirements of all tasks.

The goal is to map the task graph onto p identical processors, with the objective of minimizing the total energy consumption, while enforcing a bound on the total execution time (makespan), and matching a reliability constraint. Processors can have arbitrary speeds, determined by their frequency, that can take any value in the interval $[f_{\min}, f_{\max}]$ (dynamic voltage and frequency scaling with continuous speeds). Higher frequencies, and hence faster speeds, allow for a faster execution, but they also lead to a much higher (supra-linear) power consumption. Moreover, reducing the frequency of a processor increases the number of transient failures of the system. Therefore, some tasks are executed once at a speed high enough to satisfy the reliability constraint, while some other tasks are executed several times (either on the same processor, or on different processors), at a lower speed. We detail below the conditions that are enforced on the corresponding execution speeds. The problem is therefore to decide which tasks should be executed several times, on which processor, and at which speed to run each execution of a task, as well as the schedule, i.e., in which order the tasks are executed on each processor. Note that [4] showed that it is always better to execute a task at a single speed, and therefore we assume in the following that each execution of a task is done at a single speed.

We now detail the three objective criteria (makespan, reliability, energy), and then define formally the problem.

2.1 Makespan

The makespan of a schedule is its total execution time. The first task is scheduled at time 0, so that the makespan of a schedule is simply the maximum time at which one of the processors finishes its computations. Given a schedule, the makespan should not exceed the prescribed deadline D.

Let $\mathcal{E}xe(w_i, f)$ be the execution time of a task T_i of weight w_i at speed f. We assume that the cache size is adapted to the application, therefore ensuring that the execution time is linearly related to the frequency [14]: $\mathcal{E}xe(w_i, f) = \frac{w_i}{f}$. Note that we consider a worst-case scenario, and the deadline D must be matched even in the case where all tasks that are scheduled to be executed several times fail during their first executions, hence all execution times for a same task should be accounted for.

2.2 Reliability

To define the reliability, we use the failure model of [20] and [19]. *Transient* failures are failures caused by software errors for example. They invalidate only the execution of the current task and the processor subject to that failure will be able to recover and execute the subsequent tasks assigned to it (if any). In addition, we use the reliability model introduced by [17], which states that the radiation-induced transient failures follow a Poisson distribution. The parameter λ of the Poisson distribution is then $\lambda(f) = \tilde{\lambda_0}\ e^{\tilde{d}\frac{f_{\max}-f}{f_{\max}-f_{\min}}}$, where $f_{\min} \leq f \leq f_{\max}$ is the processing speed, the exponent $\tilde{d} \geq 0$ is a constant, indicating the sensitivity of failure rates to dynamic voltage and frequency scaling, and $\tilde{\lambda_0}$ is the average failure rate at speed $f_{\max}$. We see that reducing the speed for energy saving increases the failure rate exponentially. The reliability of a task T_i executed once at speed f is

$$R_i(f) = e^{-\lambda(f)\times\mathcal{E}xe(w_i,f)}.$$

Because the failure rate $\tilde{\lambda_0}$ is usually very small, of the order of 10^{-5} per time unit [2], or even 10^{-6} [7, 16], we can use the first order approximation of $R_i(f)$ as

$$\begin{aligned} R_i(f) &= 1 - \lambda(f) \times \mathcal{E}xe(w_i, f) \\ &= 1 - \tilde{\lambda_0}\ e^{\tilde{d}\frac{f_{\max}-f}{f_{\max}-f_{\min}}} \times \frac{w_i}{f} \\ &= 1 - \lambda_0\ e^{-df} \times \frac{w_i}{f}, \end{aligned}$$

where $d = \frac{\tilde{d}}{f_{\max}-f_{\min}}$ and $\lambda_0 = \tilde{\lambda_0} e^{df_{\max}}$.

Note that this equation holds if $\varepsilon_i = \lambda(f) \times \frac{w_i}{f} \ll 1$. With, say, $\lambda(f) = 10^{-5}$, we need $\frac{w_i}{f} \leq 10^3$ to get an accurate approximation with $\varepsilon_i \leq 0.01$: the task should execute within 16 minutes. In other words, large (computationally demanding) tasks require reasonably high processing speeds with this model (which makes full sense in practice).

We want the reliability R_i of each task T_i to be greater than a given threshold, namely $R_i(f_{\texttt{rel}})$, hence enforcing a local constraint dependent on the task: $R_i \geq R_i(f_{\texttt{rel}})$. If task T_i is executed only once at speed f, then the reliability of T_i is $R_i = R_i(f)$. Since the reliability increases with speed, we must have $f \geq f_{\texttt{rel}}$ to match the reliability constraint. If task T_i is executed twice (speeds $f^{(1)}$ and $f^{(2)}$), then the execution of T_i is successful if and only if one of the attempts do not fail, so that the reliability of T_i is $R_i = 1 - (1 - R_i(f^{(1)}))(1 - R_i(f^{(2)}))$, and this quantity should be at least equal to $R_i(f_{\texttt{rel}})$.

We restrict in this work to a maximum of two executions of a same task, either on the same processor (what we call *re-execution*), or on two distinct processors (what we call *replication*). This is based on the following observation on the two cases in which a third execution of a task may be useful.

1. The deadline is such that even if all tasks are executed twice at the slowest possible speed, the execution time is still lower than the deadline. Then, the problem is to decide which task should be executed three times, and it is quite similar to the problem that we discuss in this paper.
2. Some tasks are too big to be re-executed while there remains some time such that some small tasks can be executed at least three times at a speed even slower. In this case, the gain in energy consumption is negligible compared to the energy consumption of the big tasks at speed $f_{\texttt{rel}}$.

Note that if both execution speeds are equal, i.e., $f^{(1)} = f^{(2)} = f$, then the reliability constraint writes $1 - (\lambda_0 w_i \frac{e^{-df}}{f})^2 \geq R_i(f_{\texttt{rel}})$, and therefore

$$\lambda_0 w_i \frac{e^{-2df}}{f^2} \leq \frac{e^{-df_{\texttt{rel}}}}{f_{\texttt{rel}}} \ .$$

In the following, $f_{\text{inf},i}$ is the solution to the equation $\lambda_0 w_i \frac{e^{-2df_{\text{inf},i}}}{(f_{\text{inf},i})^2} = \frac{e^{-df_{\texttt{rel}}}}{f_{\texttt{rel}}}$, and hence task T_i can be executed twice at a speed greater than or equal to $f_{\text{inf},i}$ while meeting the reliability constraint. In practice, $f_{\text{inf},i}$ is small enough so that tasks are usually executed faster than this speed, hence reinforcing the argument that it is meaningful to restrict to two executions of a same task.

2.3 Energy

The total energy consumption corresponds to the sum of the energy consumption of each task. Let E_i be the energy consumed by task T_i. For one execution of T_i at speed f, the corresponding energy consumption is $E_i(f) = \mathcal{E}xe(w_i, f) \times f^3 = w_i \times f^2$, which corresponds to the dynamic part of the classical energy models of the literature [6, 8]. Note that we do not take static energy into account, because all processors are up and alive during the whole execution.

If task T_i is executed only once at speed f, then $E_i = E_i(f)$. Otherwise, if task T_i is executed twice at speeds $f^{(1)}$ and $f^{(2)}$, it is natural to add up the energy consumed during both executions, just as we consider both execution times when enforcing the deadline on the makespan. Again, this corresponds to the worst-case execution scenario. We obtain $E_i = E_i(f_i^{(1)}) + E_i(f_i^{(2)})$. Note that some authors [19] consider only the energy spent for the first execution in the case of re-execution, which seems unfair: re-execution comes at a price both in the makespan and in the energy consumption. Finally, the total energy consumed by the schedule, which we aim at minimizing, is $E = \sum_{i=1}^{n} E_i$.

2.4 Optimization problem

Given an application graph $\mathcal{G} = (V, \mathcal{E})$ and p identical processors, TRI-CRIT is the problem of finding a schedule that specifies which tasks should be executed twice, on which processor and at which speed each execution of a task should be processed, such that the total energy consumption E is minimized, subject to the deadline D on the makespan and to the local reliability constraints $R_i \geq R_i(f_{\texttt{rel}})$ for each $T_i \in V$.

We focus in this paper on the two following sub-problems that are restrictions of TRI-CRIT to special application graphs:

- TRI-CRIT-CHAIN: the graph is such that $\mathcal{E} = \cup_{i=1}^{n-1}\{T_i \rightarrow T_{i+1}\}$;
- TRI-CRIT-INDEP: the graph is such that $\mathcal{E} = \emptyset$.

3 Linear chains

In this section, we focus on the TRI-CRIT-CHAIN problem, that was shown to be NP-hard even on a single processor [4]. We derive an FPTAS (Fully Polynomial Time Approximation Scheme) to solve the general problem with replication and re-execution on p processors. We start with some preliminaries in Section 3.1 that allow us to characterize the shape of an optimal solution, and then we detail the FPTAS algorithm and its proof in Section 3.2.

3.1 Characterization

First, we note that while TRI-CRIT-CHAIN is NP-hard even on a single processor, the problem has polynomial complexity if no replication nor re-execution can be used. Indeed, each task is executed only once, and the energy is minimized when all tasks are running at the same speed. Note that this result can be found in [3].

Lemma 1. *Without replication or re-execution, solving* TRI-CRIT-CHAIN *can be done in polynomial time, and each task is executed at speed* $\max\left(f_{\texttt{rel}}, \frac{S}{D}\right)$.

Proof. For a linear chain of tasks, all tasks can be mapped on the same processor, and scheduled following the dependencies. No task may start earlier by using another processor, and all tasks run at the same speed. Since there is no replication nor re-execution, each task must be executed at least at speed $f_{\texttt{rel}}$ for the reliability constraint. If $S/f_{\texttt{rel}} > D$, then the tasks should be executed at speed S/D so that the deadline constraint is matched (recall that $S = \sum_{i=1}^{n} w_i$), hence the result. □

Next, accounting for replication and re-execution, we characterize the shape of an optimal solution. For linear chains, it turns out that with a single processor, only re-execution will be used, while with more than two processors, there is an optimal solution that do not use re-execution, but only replication.

Lemma 2 (Replication or re-execution)**.** *When there is only one processor, it is optimal to only use re-execution to solve* TRI-CRIT-CHAIN. *When there are at least two processors, it is optimal to only use replication to solve* TRI-CRIT-CHAIN.

Proof. With one processor, the result is obvious, since replication cannot be used. With more than one processor, if re-execution was used on task T_i, for $1 \leq i \leq n$, we can derive a solution with the same energy consumption and a smaller execution time by using replication instead of re-execution. Indeed, all instances of tasks T_j, for $j < i$, must finish before T_i starts its execution, and similarly, all instances of tasks T_j, for $j > i$, cannot start before both copies of T_i has finished its execution. Therefore, there are always at least two processors available when executing T_i for the first time, and the execution time is reduced when executing both copies of T_i in parallel (replication) rather than sequentially (re-execution). □

We further characterize the shape of an optimal solution by showing that two copies of a same task can always be executed at the same speed.

Lemma 3 (Speed of the replicas)**.** *For a linear chain, when a task is executed two times, it is optimal to have both replicas executed at the same speed.*

Proof. The proof for re-execution has been done by [4]: by convexity of the energy and reliability functions, it is always advantageous to execute two times the task at the same speed, even if the application is not a linear chain.

For replication, this lemma is only true in the case of linear chains. Indeed, because of the structure of the chain, as explained in the proof of Lemma 2, both copies of a task have the same constraints on starting and ending time, and hence it is better to execute them exactly at the same time. □

We can further characterize an optimal solution by providing detailed information about the execution speed of the tasks, depending whether they are executed only once, re-executed, or replicated.

Proposition 1. *If $D > \frac{S}{f_{\mathtt{rel}}}$, then in any optimal solution of* TRI-CRIT-CHAIN*, all tasks that are neither re-executed nor replicated are executed at speed $f_{\mathtt{rel}}$. Furthermore, let $V_r \subseteq V$ be the subset of tasks that are either re-executed or replicated. Then, these tasks are all executed at the same speed $f_{\mathtt{re-ex}}$, if $f_{\mathtt{re-ex}} \geq \max(f_{\min}, \max_{T_i \in V_r} f_{\inf,i})$.*

Proof. The proof for $p = 1$ (re-execution) can be found in [4]. We prove the result for $p \geq 2$, which corresponds to the case with replication and no re-execution (see Lemma 2). Note first that since $D > \frac{S}{f_{\mathtt{rel}}}$, if no task is replicated, we have enough time to execute all tasks at speed $f_{\mathtt{rel}}$.

Now, let us consider that task T_i is replicated at speed f_i (recall that both replicas are executed at the same speed, see Lemma 3), and task T_j is executed only once at speed f_j. Then, we have $f_j \geq f_{\mathtt{rel}}$ (reliability constraint on T_j), and $\frac{1}{\sqrt{2}} f_{\mathtt{rel}} \geq f_i$ (otherwise, executing T_i only once at speed $f_{\mathtt{rel}}$ would improve both the energy and the execution time while matching the reliability constraint).

If $f_j > f_{\mathtt{rel}}$, let us show that we can rather execute T_j at speed $f_{\mathtt{rel}}$ and T_i at a new speed $f'_i > f_i$, while keeping the same deadline: $\frac{w_i}{f'_i} + \frac{w_j}{f_{\mathtt{rel}}} = \frac{w_i}{f_i} + \frac{w_j}{f_j}$. The energy consumption is then $2w_i {f'_i}^2 + w_j f_{\mathtt{rel}}^2$. Moreover, we know that the minimum of the function $2w_i f_1^2 + w_j f_2^2$, given that $\frac{w_i}{f_1} + \frac{w_j}{f_2}$ is a constant (where f_1 and f_2 are the unknowns), is obtained for $f_1 = \frac{1}{2^{1/3}} f_2$ (see Theorem 1 by [3]). Therefore, if the optimal speed of T_j (i.e., f_2) is strictly greater than $f_{\texttt{rel}}$, then the optimal speed for T_i is $f'_i = f_1 = \frac{1}{2^{1/3}} f_2 > \frac{1}{2^{1/2}} f_2 > \frac{1}{2^{1/2}} f_{\texttt{rel}}$, that means that we can improve both energy and execution time by executing T_i only once at speed $f_{\texttt{rel}}$. Otherwise, the speed of T_j is further constrained by $f_{\texttt{rel}}$, hence the previous inequality ($f_1 = \frac{1}{2^{1/3}} f_2$) does not hold anymore, and the function is minimized for $f_2 = f_{\texttt{rel}}$. The value of f'_i can be easily deduced from the constraint on the deadline. This proves that all tasks that are not replicated are executed at speed $f_{\texttt{rel}}$.

Let $M = \max(f_{\min}, \max_{T_i \in V_r} f_{\inf,i})$. We now prove that if two tasks are replicated at a speed greater than M, then both tasks are executed at the same speed. Suppose that T_i and T_j are executed twice at speeds $f_i > f_j \geq M$. Let $\tilde{f} = f_i f_j \frac{w_i + w_j}{w_i f_j + w_j f_i}$. Then $f_i > \tilde{f} > f_j \geq M$, and therefore we can execute both tasks at speed $\tilde{f}$ while keeping the same deadline and matching the reliability constraints. By convexity, such an execution gives a better energy consumption. We can iterate on all the tasks that are replicated, hence obtaining the speed at which each task will be re-executed, $f_{\texttt{re-ex}}$. This concludes the proof. □

Following Proposition 1, we are able to precisely define $f_{\texttt{re-ex}}$, and give a closed form expression of the energy of a schedule.

Corollary 1. *Given a subset V_r of tasks re-executed or replicated, let $X = \sum_{T_i \in V_r} w_i$, and*

$$f_{re\text{-}ex} = \begin{cases} \max\left(f_{\min}, \frac{2X}{D f_{rel} - S + X} f_{rel}\right) & \text{if } p = 1; \\ \max\left(f_{\min}, \frac{X}{D f_{rel} - S + X} f_{rel}\right) & \text{if } p \geq 2. \end{cases}$$

Then, if $f_{re\text{-}ex} \geq \max_{T_i \in V_r} f_{\inf,i}$, the optimal energy consumption is

$$(S - X) f_{rel}^2 + 2X f_{re\text{-}ex}^2. \quad (1)$$

Note that the energy consumption only depends on X, and therefore TRI-CRIT-CHAIN *is equivalent in this case to the problem of finding the optimal set of tasks that have to be re-executed or replicated.*

Proof. Given a deadline D, the problem is to find the set of tasks re-executed (or replicated), and the speed of each task. Thanks to Proposition 1, we know that the tasks that are not in this set are executed at speed $f_{\texttt{rel}}$, and given the set of tasks re-executed or replicated, we can easily compute the optimal speed to execute each task in order to minimize the energy consumption: all tasks are executed at the same speed, and we have $\lambda \frac{X}{f_{\texttt{re-ex}}} + \frac{S-X}{f_{\texttt{rel}}} = D$, with $\lambda = 1$ in the case of replication ($p \geq 2$), and $\lambda = 2$ in the case of re-execution ($p = 1$). Hence the corollary. □

Remark. Note that if there is a task $T_i \in V_r$ such that $f_{\inf,i} > f_{\texttt{re-ex}}$, then the optimal solution for this set of replicated tasks is obtained by executing T_i at speed $f_{\inf,i}$, and by executing all the other tasks at a new speed $f_{\texttt{re-ex}}^{\texttt{new}} \leq f_{\texttt{re-ex}}$, such that D is exactly met. We can do this recursively until there are no more tasks T_i such that $f_{\inf,i} > f_{\texttt{re-ex}}^{\texttt{new}}$. Using the procedure COMPUTE_$V_l(V_r)$ (see Algorithm 1), we can compute the optimal energy consumption in a time polynomial in $|V_r|$.

Let $(V_l, f_{\texttt{re-ex}})$ be the result of COMPUTE_$V_l(V_r)$. Then the optimal energy consumption is $(S - X) f_{\texttt{rel}}^2 + \sum_{T_i \in V_l} 2 w_i f_{\inf,i}^2 + \sum_{T_i \in V_r \setminus V_l} 2 w_i f_{\texttt{re-ex}}^2$.

Corollary 2. *If $D > \frac{S}{f_{rel}}$,* TRI-CRIT-CHAIN *can be solved using an exponential time exact algorithm.*

Algorithm 1: Computing re-execution speeds; tasks in V_r are re-executed.

procedure COMPUTE_$V_l(V_r)$
begin

$V_l^{(0)} = \emptyset$;

$$f_{\texttt{re-ex}}^{(0)} = \begin{cases} \max\left(f_{\min}, \frac{2X}{Df_{\texttt{rel}}-S+X} f_{\texttt{rel}}\right) & \text{if } p = 1; \\ \max\left(f_{\min}, \frac{X}{Df_{\texttt{rel}}-S+X} f_{\texttt{rel}}\right) & \text{if } p \geq 2. \end{cases}$$

$j = 0$;

while $j = 0$ *or* $V_l^{(j)} \neq V_l^{(j-1)}$ **do**

$j := j + 1$;

$V_l^{(j)} = V_l^{(j-1)} \cup \{T_i \in V_r \mid f_{\inf,i} > f_{\texttt{re-ex}}^{(j-1)}\}$;

$$f_{\texttt{re-ex}}^{(j)} = \begin{cases} \max\left(f_{\min}, \frac{\sum_{T_i \in V_r \setminus V_l^{(j)}} 2w_i}{D - \frac{S-X}{f_{\texttt{rel}}} - \sum_{T_i \in V_l^{(j)}} \frac{2w_i}{f_{\inf,i}}}\right) & \text{if } p = 1; \\ \max\left(f_{\min}, \frac{\sum_{T_i \in V_r \setminus V_l^{(j)}} w_i}{D - \frac{S-X}{f_{\texttt{rel}}} - \sum_{T_i \in V_l^{(j)}} \frac{w_i}{f_{\inf,i}}}\right) & \text{if } p \geq 2. \end{cases}$$

return $(V_l^{(j)}, f_{re\text{-}ex}^{(j)})$;

Proof. The algorithm computes for every subset V_r of tasks the energy consumption if all tasks in this subset are re-executed, and it chooses one with the minimal energy consumption, that corresponds to an optimal solution. It takes exponential time to compute every subset $V_r \subseteq V$, with $|V| = n$. □

Thanks to Corollary 1, we are also able to identify problem instances that can be solved in polynomial time.

Theorem 1. TRI-CRIT-CHAIN *can be solved in polynomial time in the following cases:*

1. $D \leq \frac{S}{f_{rel}}$ *(no re-execution nor replication);*

2. $p = 1$, $D \geq \frac{1+c}{c}\frac{S}{f_{rel}}$, *where c is the only positive solution to the polynomial* $7X^3 + 21X^2 - 3X - 1 = 0$, *and hence* $c = 4\sqrt{\frac{2}{7}}\cos\frac{1}{3}(\pi - \tan^{-1}\frac{1}{\sqrt{7}}) - 1$ (≈ 0.2838), *and for* $1 \leq i \leq n$, $f_{\inf,i} \leq \frac{2c}{1+c} f_{rel}$ *(all tasks can be re-executed);*

3. $p \geq 2$, $D \geq 2\frac{S}{f_{rel}}$, *and for* $1 \leq i \leq n$, $f_{\inf,i} \leq \frac{1}{2} f_{rel}$ *(all tasks can be replicated).*

Proof. First note that when $D \leq \frac{S}{f_{\texttt{rel}}}$, the optimal solution is to execute each task only once, at speed $\frac{S}{D}$, since $S/D \geq f_{\texttt{rel}}$. Indeed, this solution matches both reliability and makespan constraints, and it was proven to be the optimal solution in Proposition 2 by [3] (it is easy to see that replication or re-execution would only increase the energy consumption).

Let us now consider that $D > \frac{S}{f_{\texttt{rel}}}$. We aim at showing that the minimum of the energy function is reached when the total weight of the re-executed or replicated tasks is

$$\begin{cases} c(Df_{\texttt{rel}} - S) & \text{if } p = 1; \\ (Df_{\texttt{rel}} - S) & \text{if } p \geq 2. \end{cases}$$

Then necessarily, when this total weight is greater than S, the optimal solution is to re-execute or replicate all the tasks. Hence the theorem. We differentiate the two cases in the following ($p = 1$ or $p = 2$).

Case 1 ($p = 1$). We want to show that the minimum energy is reached when the total weight of the subset of tasks is exactly $c(Df_{\texttt{rel}} - S)$. Let $I = \{i \mid T_i$ is executed twice in the solution$\}$, and let $X = \sum_{i \in I} a_i$.

We saw in Corollary 1 that the energy consumption cannot be lower than $(S - X)f_{\texttt{rel}}^2 + 2Xf_{\texttt{re-ex}}^2$ where $f_{\texttt{re-ex}} = \frac{2X}{Df_{\texttt{rel}} - S + X} f_{\texttt{rel}}$. Therefore, we want to minimize $E(X) = (S - X)f_{\texttt{rel}}^2 + 2X \left(\frac{2X}{Df_{\texttt{rel}} - S + X} f_{\texttt{rel}} \right)^2$.

If we differentiate E, we can see that the minimum is reached when $-1 + \frac{24X^2}{(Df_{\texttt{rel}} - S + X)^2} - \frac{16X^3}{(Df_{\texttt{rel}} - S + X)^3} = 0$, that is, $-(Df_{\texttt{rel}} - S + X)^3 + 24X^2(Df_{\texttt{rel}} - S + X) - 16X^3 = 0$, or

$$7X^3 + 21(Df_{\texttt{rel}} - S)X^2 - 3(Df_{\texttt{rel}} - S)^2 X - (Df_{\texttt{rel}} - S)^3 = 0.$$

The only positive solution to this equation is $X = c(Df_{\texttt{rel}} - S)$, and therefore the minimum is reached for this value of X, and then $f_{\texttt{re-ex}} = \frac{2c}{1+c} f_{\texttt{rel}}$.

When $X \geq S$, re-executing each task is the best strategy to minimize the energy consumption, and that corresponds to the case $D \geq \frac{1+c}{c} \frac{S}{f_{\texttt{rel}}}$. The re-execution speed may then be lower than $\frac{2c}{1+c} f_{\texttt{rel}}$. Therefore, it may happen that $f_{\inf,i} > f_{\texttt{re-ex}}$ for some task T_i. However, even with a tighter deadline, it would be better to re-execute T_i at speed $\frac{2c}{1+c} f_{\texttt{rel}}$ rather than to execute it only once at speed $f_{\texttt{rel}}$. Therefore, since $f_{\inf,i} \leq \frac{2c}{1+c} f_{\texttt{rel}}$, it is optimal to re-execute T_i, at the lowest possible speed, i.e., $f_{\inf,i}$. Note that this changes the value of $f_{\texttt{re-ex}}$, and the call to COMPUTE_$V_l(V)$ (see Algorithm 1) returns tasks that are executed at $f_{\inf,i}$, together with the re-execution speed for all the other tasks.

Case 2 ($p \geq 2$). Similarly, we want to show that, in this case, the minimum energy is reached when the total weight of the subset of tasks that are replicated is exactly $Df_{\texttt{rel}} - S$. Let $I = \{i \mid T_i$ is executed twice in the solution$\}$, and let $X = \sum_{i \in I} a_i$.

We saw in Corollary 1 that the energy consumption cannot be lower than $(S - X)f_{\texttt{rel}}^2 + 2Xf_{\texttt{re-ex}}^2$ where $f_{\texttt{re-ex}} = \frac{X}{Df_{\texttt{rel}} - S + X} f_{\texttt{rel}}$. Therefore, we want to minimize $E(X) = (S - X)f_{\texttt{rel}}^2 + 2X \left(\frac{X}{Df_{\texttt{rel}} - S + X} f_{\texttt{rel}} \right)^2$.

If we differentiate E, we can see that the minimum is reached when

$$-1 + \frac{6X^2}{(Df_{\texttt{rel}} - S + X)^2} - \frac{4X^3}{(Df_{\texttt{rel}} - S + X)^3} = 0,$$

that is, $-(Df_{\texttt{rel}} - S + X)^3 + 6X^2(Df_{\texttt{rel}} - S + X) - 4X^3 = 0$, or

$$X^3 + 3(Df_{\texttt{rel}} - S)X^2 - 3(Df_{\texttt{rel}} - S)^2 X - (Df_{\texttt{rel}} - S)^3 = 0.$$

The only positive solution to this equation is $X = Df_{\texttt{rel}} - S$, and therefore the minimum is reached for this value of X, and then $f_{\texttt{re-ex}} = \frac{1}{2} f_{\texttt{rel}}$.

When $X \geq S$, replicating each task is the best strategy to minimize the energy consumption, and that corresponds to the case $D \geq \frac{2S}{f_{\texttt{rel}}}$. Similarly to Case 1, it is easy to see that each task should be replicated, even if $f_{\inf,i} > f_{\texttt{re-ex}}$, since $f_{\inf,i} \leq \frac{1}{2} f_{\texttt{rel}}$. The optimal solution can also be obtained with a call to COMPUTE_$V_l(V)$. □

3.2 FPTAS for TRI-CRIT-CHAIN

We derive in this section a fully polynomial time approximation scheme (FPTAS) for TRI-CRIT-CHAIN, based on the FPTAS for SUBSET-SUM [10], and the results of Section 3.1. Without loss of generality, we use the term *replication* for either re-execution or replication, since both scenarios have already been clearly identified. The problem consists in identifying the set of replicated tasks V_r, and then the optimal solution can be derived from Corollary 1; it depends only on the total weight of these tasks, $\sum_{T_i \in V_r} w_i$, denoted in the following as $w(V_r)$.

Note that we do not account in this section for $f_{\inf,i}$ or $f_{\min}$ for readability reasons: $f_{\inf,i}$ can usually be neglected because $\lambda_0 w_i / f$ is supposed to be very small whatever f, and $f_{\min}$ simply adds subcases to the proofs (rather than an execution at speed f, the speed should be $\max(f, f_{\min})$).

First we introduce a few preliminary functions in Algorithm 2, and we exhibit their properties. These are the basis of the approximation algorithm.

When $D > \frac{S}{f_{\text{rel}}}$, X-OPT$(V, D, p)$ returns the optimal value for the weight $w(V_r)$ of the subset of replicated tasks V_r, i.e., the value that minimizes the energy consumption for TRI-CRIT-CHAIN. The optimality comes directly from the proof of Theorem 1.

Given a value X, which corresponds to $w(V_r)$, ENERGY(V, D, p, X) returns the optimal energy consumption when a subset of tasks V_r is replicated.

Then, the function TRIM(L, ε, X) trims a sorted list $L = [L_0, \cdots, L_{m-1}]$ in time $O(m)$, given L and ε. L is sorted into non decreasing order. The function returns a trimmed list, where two consecutive elements differ from at least a factor $(1 + \varepsilon)$, except the last element, that is the smallest element of L strictly greater than X. This trimming procedure is quite similar to that used for SUBSET-SUM [10], except that the latter keeps only elements lower than X. Indeed, SUBSET-SUM can be expressed as follows: given n strictly positive integers $a_1, \ldots, a_n$, and a positive integer X, we wish to find a subset I of $\{1, \ldots, n\}$ such that $\sum_{i \in I} a_i$ is as large as possible, but not larger than X. In our case, the optimal solution may be obtained either by approaching X by below or by above.

Finally, the approximation algorithm is APPROX-CHAIN(V, D, p, ε) (see Algorithm 2), where $0 < \varepsilon < 1$, and it returns an energy consumption E that is not greater than $(1 + \varepsilon)$ times the optimal energy consumption. Note that if $L = [L_0, \ldots, L_{m-1}]$, then ADD-LIST$(L, x)$ adds element x at the end of list L (i.e., it returns the list $[L_0, \ldots, L_{m-1}, x]$); $L + w$ is the list $[L_0 + w, \ldots, L_{m-1} + w]$; and MERGE-LISTS$(L, L')$ is merging two sorted lists (and returns a sorted list).

We now prove that this approximation scheme is an FPTAS:

Theorem 2. APPROX-CHAIN *is a fully polynomial time approximation scheme for* TRI-CRIT-CHAIN.

Proof. We assume that

- if $p = 1$, then $\frac{S}{f_{\text{rel}}} < D < \frac{1+c}{c} \frac{S}{f_{\text{rel}}} < 5 \frac{S}{f_{\text{rel}}}$;
- if $p \geq 2$, then $\frac{S}{f_{\text{rel}}} < D < 2 \frac{S}{f_{\text{rel}}}$;

otherwise the optimal solution is obtained in polynomial time (see Theorem 1).

Let $I_{\inf} = \{V' \subseteq V \mid w(V') \leq$ X-OPT$(V, D, p)\}$, and $I_{\sup} = \{V'' \subseteq V \mid w(V'') >$ X-OPT$(V, D, p)\}$. Note that $I_{\inf}$ is not empty, since $\emptyset \in I_{\inf}$.

First we characterize the solution with the following lemma:

Algorithm 2: Approximation algorithm for TRI-CRIT-CHAIN.

function X-OPT(V, D, p)
begin
  $S = \sum_{T_i \in V} w_i$;
  if $p = 1$ **then return** $c(Df_{rel} - S)$;
  else return $Df_{rel} - S$;

function ENERGY(V, D, p, X)
begin
  $S = \sum_{T_i \in V} w_i$;
  if $p{=}1$ **then return** $(S{-}X)f_{rel}^2{+}2X\left(\max\left(f_{\min}, \frac{2X}{Df_{rel}-S+X} f_{rel}\right)\right)^2$;
  else return $(S - X)f_{rel}^2 + 2X\left(\max\left(f_{\min}, \frac{X}{Df_{rel}-S+X} f_{rel}\right)\right)^2$;

function TRIM(L, ε, X)
begin
  $m = |L|$; $L = [L_0, \ldots, L_{m-1}]$; $L' = [L_0]$; $last = L_0$;
  for $i = 1$ **to** $m - 1$ **do**
    if *(*$last \leq X$ *and* $L_i > X$*) or* $L_i > last \times (1 + \varepsilon)$ **then**
      L' = ADD-LIST(L', L_i); $last = L_i$;
  return L'*;*

function APPROX-CHAIN(V, D, p, ε)
begin
  $X = \lfloor$X-OPT$(V, D, p)\rfloor$; $n = |V|$; $L^{(0)} = [0]$;
  for $i = 1$ **to** n **do**
    $L^{(i)}$ = MERGE-LISTS$(L^{(i-1)}, L^{(i-1)} + w_i)$;
    $L^{(i)}$ = TRIM$(L^{(i)}, \varepsilon/(28 \times 2n), X)$;
  Let $Y_1 \leq Y_2$ be the two largest elements of $L^{(n)}$;
  return $\min($ENERGY(V, D, p, Y_1), ENERGY$(V, D, p, Y_2))$*;*

Lemma 4. *Suppose $D > \frac{S}{f_{rel}}$. Then in the solution of* TRI-CRIT-CHAIN*, the subset of replicated tasks V_r is either an element $V' \in I_{\inf}$ such that $w(V')$ is maximum, or an element $V'' \in I_{\sup}$ such that $w(V'')$ is minimum.*

Proof. Recall first that according to Proposition 1, the energy consumption of a linear chain is not dependent on the number of tasks replicated, but only on the sum of their weights.

Then the lemma is obvious by convexity of the functions, and since X-OPT returns the optimal value of $w(V_r)$, the weight of the replicated tasks. Therefore, the closest the weight of the set of replicated tasks is to the optimal weight, the better the solution is. Finally, any element in $I_{\inf}$ is a solution (since we have a solution for X-OPT), and if the minimal element (if it exists) of $I_{\sup}$ is not a solution, ($f_{\texttt{re-ex}}$ too large because of time constraints), then no element of $I_{\sup}$ can be a better solution. □

We are now ready to prove Theorem 2. Let $X_1 = \max_{V_1 \in I_{\inf}} w(V_1)$, and $X_2 = \max_{V_2 \in I_{\sup}} w(V_2)$. Thanks to Lemma 4, the optimal set of replicated tasks V_o is such that $X_o = w(V_o) = X_1$ or $X_o = X_2$. The corresponding energy consumption is (Corollary 1):

$$E_{opt} = \begin{cases} (S - X_o) f_{\texttt{rel}}^2 + \frac{(2X_o)^3}{(D f_{\texttt{rel}} - S + X_o)^2} f_{\texttt{rel}}^2 & \text{if } p = 1 \\ (S - X_o) f_{\texttt{rel}}^2 + \frac{2X_o^3}{(D f_{\texttt{rel}} - S + X_o)^2} f_{\texttt{rel}}^2 & \text{if } p \geq 2 \end{cases}.$$

The solution returned by APPROX-CHAIN corresponds either to Y_1 or to Y_2, where Y_1 and Y_2 are the two largest elements of the trimmed list. We first prove that at least one of these two elements, denoted X_a, is such that $X_a \leq X_o \leq (1+\varepsilon')X_a$, where $\varepsilon' = \frac{\varepsilon}{28}$.

Existence of X_a such that $X_a \leq X_o \leq (1+\varepsilon')X_a$. We differentiate two cases.

(a) If $Y_2 > X$, then Y_1 is the value obtained by the FPTAS for SUBSET-SUM [10] with the approximation ratio ε', since it is the largest value not greater than X, and our algorithm is identical for such values. Moreover, note that X_1 is the optimal solution of SUBSET-SUM by definition, and therefore $Y_1 \leq X_1 < (1+\varepsilon')Y_1$. If $X_o = X_1$, the value $X_a = Y_1$ satisfies the property.

If $X_o = X_2$, we prove that the property remains valid, by considering the SUBSET-SUM problem with a bound X_2 instead of X. Then, since $Y_2 > X$, we have $Y_2 \geq X_2$ by definition of X_2. Moreover, APPROX-CHAIN is not removing any element of the list greater than Y_2, and therefore all elements between X and X_2 are kept, similarly to the FPTAS for SUBSET-SUM. If $Y_2 = X_2$, then $X_a = Y_2$ satisfies the property. Otherwise, Y_1 is the result of the FPTAS for SUBSET-SUM with a bound X_2, whose optimal solution is X_2, and therefore Y_1 is such that $Y_1 \leq X_2 < (1+\varepsilon')Y_1$; $X_a = Y_1$ satisfies the property.

(b) If $Y_2 \leq X$, no elements greater than X have been removed from the lists, and APPROX-CHAIN has been identical to the FPTAS for SUBSET-SUM. Then, $X_a = Y_2$ is the solution, that is valid both for SUBSET-SUM applied with the original bound X (optimal solution X_1), and with the modified bound X_2 (optimal solution X_2). Therefore, $Y_2 \leq X_1 < (1+\varepsilon')Y_2$ and $Y_2 \leq X_2 < (1+\varepsilon')Y_2$, which concludes the proof.

We have shown that there always is X_a (either Y_1 or Y_2) such that $X_a \leq X_o < (1+\varepsilon')X_a$. Next, we show that the energy E_a obtained with this value X_a is such that $E_{opt} \leq E_a \leq (1+\varepsilon)E_{opt}$.

Approximation ratio on the energy: $E_a \leq (1+\varepsilon)E_{opt}$. Let us consider first that $p \geq 2$. Then we have $E_a = (S-X_a)f_{\mathtt{rel}}^2 + \frac{2X_a^3}{(Df_{\mathtt{rel}}-S+X_a)^2} f_{\mathtt{rel}}^2$. Re-using the previous inequalities on X_a, we obtain: $\frac{E_a}{f_{\mathtt{rel}}^2} \leq S - \frac{X_o}{1+\varepsilon'} + \frac{2X_o^3}{(Df_{\mathtt{rel}}-S+\frac{X_o}{1+\varepsilon'})^2}$. Then, this can be rewritten so that E_{opt} appears:

$$\frac{E_a}{f_{\mathtt{rel}}^2} \leq \left(\frac{1}{1+\varepsilon'}(S-X_o) + \frac{\varepsilon'}{1+\varepsilon'}S\right) + \left((1+\varepsilon')^2 \frac{2X_o^3}{((1+\varepsilon')(Df_{\mathtt{rel}}-S)+X_o)^2}\right)$$

$$\begin{aligned}\frac{E_a}{f_{\mathtt{rel}}^2} &\leq ((S-X_o)+\varepsilon' S) \\ &\quad + \left((1+\varepsilon')^2 \frac{2X_o^3}{(Df_{\mathtt{rel}}-S+X_o)^2}\right) \\ &\leq ((S-X_o)+\varepsilon' S) \\ &\quad + \left((1+\varepsilon')^2 (\frac{E_{opt}}{f_{\mathtt{rel}}^2} - (S-X_o))\right) \\ &\leq (1+\varepsilon')^2 \frac{E_{opt}}{f_{\mathtt{rel}}^2} \\ &\quad - ((1+\varepsilon')^2-1)(S-X_o) + \varepsilon' S \\ &\leq (1+\varepsilon')^2 \frac{E_{opt}}{f_{\mathtt{rel}}^2} + \varepsilon' S.\end{aligned}$$

The case $p=1$ leads to the same inequality; the only difference is in the energy E_a, where $2X_a^3$ is replaced by $(2X_a)^3$, and the same difference holds for E_{opt} ($2X_o^3$ is replaced by $(2X_o)^3$).

Finally, note that with no reliability constraints, each task is executed only once at speed S/D, and therefore the energy consumption is at least $E_{opt} \geq S\frac{S^2}{D^2}$. Moreover, by hypothesis, $D < \frac{5S}{f_{\mathtt{rel}}}$ (for $p \geq 1$). Therefore, $S < \frac{25E_{opt}}{f_{\mathtt{rel}}^2}$ and $\frac{E_a}{f_{\mathtt{rel}}^2} < (1+\varepsilon')^2\frac{E_{opt}}{f_{\mathtt{rel}}^2} + \varepsilon'\frac{25E_{opt}}{f_{\mathtt{rel}}^2}$.

We conclude that

$$\frac{E_a}{E_{opt}} < 1 + 27\varepsilon' + \varepsilon'^2 < 1 + 28\varepsilon' = 1 + \varepsilon.$$

Conclusion. The energy consumption returned by APPROX-CHAIN, denoted as E_{algo}, is such that $E_{algo} \leq E_a$, since we take the minimum out of the consumption obtained for Y_1 or Y_2, and X_a is either Y_1 or Y_2. Therefore, $E_{algo} \leq (1+\varepsilon)E_{opt}$.

It is clear that the algorithm is polynomial both in the size of the instance and in $\frac{1}{\varepsilon}$, given that the trimming function and APPROX-CHAIN have the same complexity as in the original approximation scheme for SUBSET-SUM (see [10]), and all other operations are polynomial in the problem size (X-OPT, ENERGY). □

4 Independent tasks

In this section, we focus on the problem of scheduling independent tasks, TRI-CRIT-INDEP. Similarly to TRI-CRIT-CHAIN, we know that TRI-CRIT-INDEP is NP-hard, even on a single processor. We first prove in Section 4.1 that there exists no constant factor approximation algorithm for this problem, unless P=NP. We discuss and characterize solutions to TRI-CRIT-INDEP in Section 4.2, while highlighting the intrinsic difficulty of the problem. The core result is a constant factor approximation algorithm with a relaxation on the constraint on the makespan (Section 4.3).

4.1 Inapproximability of TRI-CRIT-INDEP

Lemma 5. *For all $\lambda > 1$, there does not exist any λ-approximation of* TRI-CRIT-INDEP*, unless $P = NP$.*

Proof. Let us assume that there is a λ-approximation algorithm for TRI-CRIT-INDEP. We consider an instance $\mathcal{I}_1$ of 2-PARTITION: given n strictly positive integers $a_1, \dots, a_n$, does there exist a subset I of $\{1, \dots, n\}$ such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$? Let $S = \sum_{i=1}^n a_i$.

We build the following instance $\mathcal{I}_2$ of our problem. We have n independent tasks T_i to be mapped on $p = 2$ processors, and:

- task T_i has a weight $w_i = a_i$;
- $f_{\min} = f_{\texttt{rel}} = f_{\max} = S/2$;
- $D = 1$.

We use the λ-approximation algorithm to solve $\mathcal{I}_2$, and the solution of the algorithm E_{algo} is such that $E_{algo} \leq \lambda E_{opt}$, where E_{opt} is the optimal solution. We consider the two following cases.

(i) If the λ-approximation algorithm returns a solution, then necessary all tasks are executed exactly once at speed $f_{\max}$, since $\sum_{i=1}^n w_i / f_{\max} = 2$ and there are two processors. Moreover, because of the makespan constraint, the load on each processor is equal. Let I be the indices of the tasks executed on the first processor. We have $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$, and therefore I is also a solution to $\mathcal{I}_1$.

(ii) If the λ-approximation algorithm does not return a solution, then there is no solution to $\mathcal{I}_1$. Otherwise, if I is a solution to $\mathcal{I}_1$, there is a solution to $\mathcal{I}_2$ such that tasks of I are executed on the first processor, and the other tasks are executed on the second processor. Since $E_{algo} \leq \lambda E_{opt}$, the approximation algorithm should have returned a valid solution.

Therefore, the result of the algorithm for $\mathcal{I}_2$ allows us to conclude in polynomial time whether there is a solution to the instance $\mathcal{I}_1$ of 2-PARTITION or not. Since 2-PARTITION is NP-complete [12], the inapproximability result is true unless P=NP. □

4.2 Characterization

As discussed in Section 1, the problem of scheduling independent tasks is usually close to a problem of load balancing, and can be efficiently approximated for various mono-criterion versions of the problem (minimizing the makespan or the energy, for instance). However, the tri-criteria problem turns out to be much harder, and cannot be approximated, as seen in Section 4.1, even when reliability is not a constraint.

Adding reliability further complicates the problem, since we no longer have the property that on each processor, there is a constant execution speed for the tasks executed on this processor. Indeed, some processors may process both tasks that are not replicated (or re-executed), hence at speed $f_{\texttt{rel}}$, and replicated tasks at a slower speed. Similarly to Section 3.2, we use the term *replication* for either re-execution or replication; if a task is replicated, it means it is executed two times, and it appears two times in the load of processors, be it the same processor or two distinct processors.

Furthermore, contrary to the TRI-CRIT-CHAIN problem, we do not always have the same execution speed for both executions of a task, as in Lemma 3:

Proposition 2. *In an optimal solution of* TRI-CRIT-INDEP*, if a task T_i is executed twice:*

- *if both executions are on the same processor, then both are executed at the same speed, lower than $\frac{1}{\sqrt{2}}f_{\texttt{rel}}$;*
- *however, when the two executions of this task are on distinct processors, then they are not necessarily executed at the same speed. Furthermore, one of the two speeds can be greater than $\frac{1}{\sqrt{2}}f_{\texttt{rel}}$.*

Moreover, we have $w_i < \frac{1}{\sqrt{2}}Df_{\texttt{rel}}$.

Proof. We start by proving the properties on the speeds. When both executions occur on the same processor, this property was shown by [4]: a single execution at speed $f_{\texttt{rel}}$ leads to a better energy consumption (and a lower execution time).

In the case of distinct processors, we give an example in which the optimal solution uses different speeds for a replicated task, with one speed greater than $\frac{1}{\sqrt{2}}f_{\texttt{rel}}$. Note that one of the speeds is necessary lower than $\frac{1}{\sqrt{2}}f_{\texttt{rel}}$, otherwise a solution with only one execution of this task at speed $f_{\texttt{rel}}$ would be better, similarly to the case with re-execution.

Consider a problem instance with two processors, $f_{\texttt{rel}} = f_{\max}$, $D = \frac{6.4}{f_{\max}}$, and three tasks such that $w_1 = 5$, $w_2 = 3$, and $w_3 = 1$. Because of the time constraints, T_1 and T_2 are necessarily executed on two distinct processors, and neither of them can be re-executed on its processor. The problem consists in scheduling task T_3 to minimize the energy consumption. There are three possibilities:

- T_3 is executed only once on any of the processors, at speed $f_{\texttt{rel}} = f_{\max}$;
- T_3 is executed twice on the same processor; it is executed on the same processor than T_2, hence having an execution time of $D - \frac{w_2}{f_{\max}} = \frac{3.4}{f_{\max}}$, and therefore both executions are done at a speed $\frac{2}{3.4}f_{\max}$;
- T_3 is executed once on the same processor than T_1 at a speed $\frac{1}{1.4}f_{\max}$, and once on the other processor at a speed $\frac{1}{3.4}f_{\max}$.

It is easy to see that the minimum energy consumption is obtained with the last solution, and that $\frac{1}{1.4}f_{\max} > \frac{1}{\sqrt{2}}f_{\texttt{rel}}$, hence the result.

Finally, note that since at least one of the executions of the task should be at a speed lower than $\frac{1}{\sqrt{2}}f_{\texttt{rel}}$, and since the deadline is D, in order to match the deadline, the weight of the replicated task has to be strictly lower than $\frac{1}{\sqrt{2}}Df_{\texttt{rel}}$. □

Because of this proposition, usual load balancing algorithms are likely to fail, since processors handling only non-replicated tasks should have a much higher load, and speeds of replicated tasks may be very different from one processor to another in the optimal solution.

We now derive lower bounds on the energy consumption, that will be useful to design an approximation algorithm in the next section.

Proposition 3 (Lower bound without reliability)**.** *The optimal solution of* TRI-CRIT-INDEP *cannot have an energy lower than* $\frac{S^3}{(pD)^2}$.

Proof. Let us consider the problem of minimizing the energy consumption, with a deadline constraint D, but without accounting for the constraint on reliability. A lower bound is obtained if the load on each processor is exactly equal to $\frac{S}{p}$, and the speed of each processor is constant and equal to $\frac{S}{pD}$. The corresponding energy consumption is $S \times \left(\frac{S}{pD}\right)^2$, hence the bound. □

However, if the speed $\frac{S}{pD}$ is small compared to $f_{\texttt{rel}}$, the bound is very optimistic since reliability constraints are not matched at all. Indeed, replication must be used in such a case. We investigate bounds that account for replication in the following, using the optimal solution of the TRI-CRIT-CHAIN problem.

Proposition 4 (Lower bound using linear chains)**.** *For the* TRI-CRIT-INDEP *problem, the optimal solution cannot have an energy lower than the optimal solution to the* TRI-CRIT-CHAIN *problem on a single processor with a deadline pD, where the weight of the re-executed tasks is lower than $\frac{1}{\sqrt{2}} D f_{\texttt{rel}}$.*

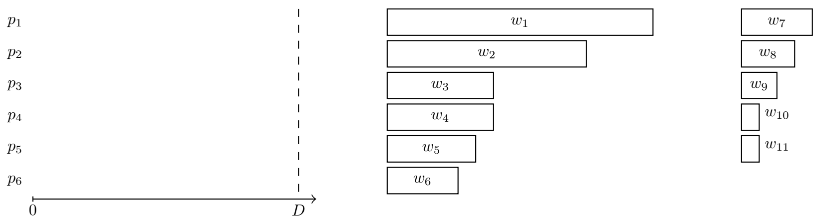
Proof. We can transform any solution to the TRI-CRIT-INDEP problem into a solution to the TRI-CRIT-CHAIN problem with deadline pD and a single processor. Tasks are arbitrarily ordered as a linear chain, and the solution uses the same number of executions and the same speed(s) for each task. It is easy to see that the TRI-CRIT-INDEP problem is more constrained, since the deadline on each processor must be enforced. The constraint on the weights of the re-executed tasks comes from Proposition 2. Therefore, the solution to the TRI-CRIT-CHAIN problem is a lower bound for TRI-CRIT-INDEP. □

The optimal solution may however be far from this bound, since we do not know if the tasks that are re-executed on a chain with a long deadline pD can be executed at the same speed when the deadline is D. The constraint on the weight of the re-executed tasks allows us to improve slightly the bound, and this lower bound is the basis of the approximation algorithm that we design for TRI-CRIT-INDEP.

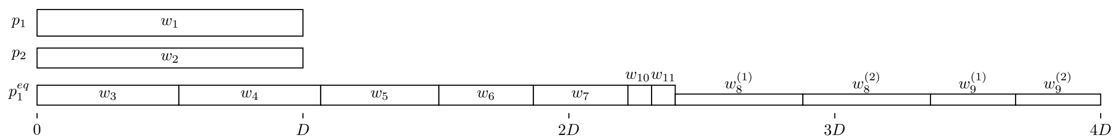
4.3 Approximation algorithm for TRI-CRIT-INDEP

We have seen in Section 4.1 that there exists no constant factor approximation algorithm for TRI-CRIT-INDEP, unless P=NP, even without accounting for the reliability constraint. This is due to the constraint on the makespan and the maximum speed $f_{\max}$. Therefore, in order to provide a constant factor approximation algorithm, we relax the constraint on the makespan and propose an (α, β)-approximation algorithm. The solution E_{algo} is such that $E_{algo} \leq \alpha \times E_{opt}$, where E_{opt} is the optimal solution with the deadline constraint D, and the makespan of the algorithm M_{algo} is such that $M_{algo} \leq \beta \times D$.

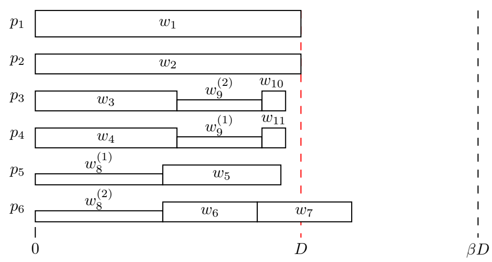
The result of Section 4.1 means that for all $\alpha > 1$, there is no $(\alpha, 1)$-approximation algorithm for TRI-CRIT-INDEP, unless $P = NP$. Therefore, we present an algorithm that realizes a $(1 + \frac{1}{\beta^2}, \beta)$-approximation, where the minimum relaxation on



(a) Input: six processors and eleven tasks



(b) Schedule the *big* tasks on p_1 and p_2, and call APPROX-CHAIN with deadline $(6-2)D$ on the remaining tasks



(c) Greedy algorithm to schedule the new tasks

Figure 1: $\left(1+\frac{1}{\beta^2},\beta\right)$-approximation algorithm for independent tasks

the deadline is smaller than 2. It is of course possible to run the algorithm with larger values of β, leading to a better guarantee on the energy consumption.

Sketch of the algorithm. In the first step of the algorithm, we schedule each task with a big weight alone on one processor, with no replication. A task T_i is considered as *big* if $w_i \geq \max(\frac{S}{p}, Df_{\texttt{rel}})$. This step is done in polynomial time: we sort the tasks by non-increasing weights, and then we check whether the current task is such that $w_i \geq \max(\frac{S}{p}, Df_{\texttt{rel}})$. If it is the case, we schedule the task alone on a processor and we let $S = S - w_i$ and $p = p - 1$. The procedure ends when the current task is small enough, i.e., all remaining tasks are such that $w_i < \max(\frac{S}{p}, Df_{\texttt{rel}})$, with the updated values of S and p.

- If $S > pDf_{\texttt{rel}}$, i.e., the load is *large enough*, we do not use replication, but we schedule the tasks at speed $\frac{S}{pD}$, using a simple scheduling heuristic, DECREASING-FIRST-FIT [13]. Tasks are sorted by non increasing weights, and at each time step, we schedule the current task on the least loaded processor. Thanks to the lower bound of Proposition 3, the energy consumption is not greater than the optimal energy consumption, and we determine β such that the deadline is enforced.
- If $S \leq pDf_{\texttt{rel}}$, the previous bound is not good enough, and therefore we use the

FPTAS on a linear chain of tasks with deadline pD for TRI-CRIT-CHAIN (see Theorem 2). The FPTAS is called with

$$\varepsilon = \min\left(\frac{2w_{min}}{3S}\left(\frac{f_{\min}}{f_{\texttt{rel}}}\right)^2, \frac{1}{3\beta^2}\right), \tag{2}$$

where $w_{min} = \min_{1 \leq i \leq n} w_i$. Note that it is slightly modified so that only tasks of weight $w < \frac{1}{\sqrt{2}} D f_{\texttt{rel}}$ can be replicated, and that we enforce a minimum speed $f_{\min}$. The FPTAS therefore determines which tasks should be executed twice, and it fixes all execution speeds.
We then use DECREASING-FIRST-FIT in order to map the tasks onto the p processors, at the speeds determined earlier. The new set of tasks includes both executions in case of replication, and tasks are sorted by non increasing execution times (since all speeds are fixed). At each time step, we schedule the current task on the least loaded processor. If some tasks cannot fit in one processor within the deadline βD, we re-execute them at speed $\frac{w_i}{\beta D}$ on two processors. Thanks to the lower bound of Proposition 4, we can bound the energy consumption in this case.

We illustrate the algorithm on an example in Figure 1, where eleven tasks must be mapped on six processors. For each task, we represent its execution speed as its height, and its execution time as its width. There are two *big* tasks, of weights w_1 and w_2, that are each mapped on a distinct processor. Then, we have $p = 4$ and we call APPROX-CHAIN with deadline $4D$; tasks T_8 and T_9 are replicated. Finally, DECREASING-FIRST-FIT greedily maps all instances of the tasks, slightly exceeding the original bound D, but all tasks fit within the extended deadline.

This algorithm leads to the following theorem:

Theorem 3. *For the problem* TRI-CRIT-INDEP*, there are* $\left(1 + \frac{1}{\beta^2}, \beta\right)$*-approximation algorithms, for all* $\beta \geq 2 - \Theta(\frac{1}{p})$*, that run in polynomial time.*

Before proving Theorem 3, we give some preliminary results: we prove below the optimality of the first step of the algorithm, i.e., the optimal solution would schedule tasks of weight greater than $\max(\frac{S}{p}, D f_{\texttt{rel}})$ alone on a processor:

Proposition 5. *In any optimal solution to* TRI-CRIT-INDEP*, each task* T_i *such that* $w_i \geq \max(\frac{S}{p}, D f_{rel})$ *is executed only once, and it is alone on its processor.*

Proof. Let us prove the result by contradiction. Suppose that there exists a task T_i such that $w_i \geq \max(\frac{S}{p}, D f_{\texttt{rel}})$, and that this task is executed on processor p_1. Suppose also that there is another task T_j executed on p_1, with $w_j \leq w_i$. Necessarily, there exists a processor, say p_2, whose load is smaller than $\frac{S}{p}$, since the load of p_1 is strictly greater than $\frac{S}{p}$. Consider the energy of the tasks executed on processors p_1 and p_2. Because of the convexity of the energy function, it is strictly better to execute task T_j on processor p_2, and then T_i is executed alone on processor p_1, at a speed $\frac{w_i}{D} \geq f_{\texttt{rel}}$. □

Next, we prove a lemma that will allow us to tackle the case where the load is *large enough* ($S > pDf_{\texttt{rel}}$), and we obtain a minimum on the approximation ratio of the deadline β.

Lemma 6. *For the problem* TRI-CRIT-INDEP *where each task* T_i *is such that* $w_i < \max(\frac{S}{p}, Df_{\texttt{rel}})$*, scheduling each task only once at speed* $\max(f_{\texttt{rel}}, \frac{S}{pD})$ *with the* DECREASING-FIRST-FIT *heuristic leads to a makespan of at most* βD*, with* $\beta = \max\left(2-\frac{3}{2p+1}, 2-\frac{p+2}{4p+2}\right)$.

Note that we introduce $\max(\frac{S}{p}, Df_{\texttt{rel}})$ since the lemma is also used in the case $S \leq pDf_{\texttt{rel}}$. Also, since β is increasing with p and the bound is computed in fact for a number of processors smaller than the original one (some processors are dedicated to *big* tasks), the value of β computed with the total number of processors p is not smaller and it is possible to achieve a makespan of at most βD.

Proof. Let l_{dff} be the maximal load of the processors after applying DECREASING-FIRST-FIT on the weights of the tasks. Let us find β such that $l_{\text{dff}}\frac{pD}{S} \leq \beta D$: this means that within a time βD, we can schedule all tasks at speed $\frac{S}{pD}$, and therefore at speed $\max(f_{\texttt{rel}}, \frac{S}{pD})$, since the most loaded processor succeeds to be within the deadline βD.

Let l_{opt} be the maximal load of the processors in an optimal solution, and let T_i be the last task executed on the processor with the maximal load l_{dff} by DECREASING-FIRST-FIT. We have either $w_i \leq l_{\text{opt}}/3$ or $w_i > l_{\text{opt}}/3$.

- If $w_i \leq l_{\text{opt}}/3$, we know that $l_{\text{opt}} \leq l_{\text{dff}} \leq \left(\frac{4}{3} - \frac{1}{3p}\right) l_{\text{opt}}$, since DECREASING-FIRST-FIT is a $\left(\frac{4}{3} - \frac{1}{3p}\right)$-approximation [13]. We want to compare l_{opt} to S/p (average load). We consider the solution of DECREASING-FIRST-FIT. At the time when T_i was scheduled, all the processors were at least as loaded as the one on which T_i was scheduled, and hence we obtain a lower bound on S: $S \geq (p-1)(l_{\text{dff}} - w_i) + l_{\text{dff}}$. Furthermore, $l_{\text{dff}} - w_i \geq \frac{2}{3} l_{\text{opt}}$ (because $l_{\text{dff}} \geq l_{\text{opt}}$ and $w_i \leq l_{\text{opt}}/3$). Finally, $S \geq (p-1)\frac{2}{3}l_{\text{opt}} + l_{\text{opt}}$, which means that $l_{\text{opt}} \leq \frac{S}{p}\frac{3p}{2p+1}$, and $l_{\text{dff}} \leq \left(\frac{4}{3} - \frac{1}{3p}\right)\frac{3p}{2p+1}\frac{S}{p} = \left(2 - \frac{3}{2p+1}\right)\frac{S}{p}$.

  In this case, with $\beta = 2 - \frac{3}{2p+1}$, we can execute all the tasks at speed $\max(f_{\texttt{rel}}, \frac{S}{pD})$ within the deadline βD.

- If $w_i > l_{\text{opt}}/3$, it is known that DECREASING-FIRST-FIT is optimal for the execution time [13], i.e., $l_{\text{opt}} = l_{\text{dff}}$, and we aim at finding an upper bound on l_{opt}. We assume in the following that tasks are sorted by non increasing weights.

  If $w_i \geq \frac{S}{p}$, then we show that T_i is the only task executed on its processor (recall that T_i is the last task executed on the processor with the maximal load by DECREASING-FIRST-FIT). Indeed, there cannot be p tasks of weight not smaller than $\frac{S}{p}$, hence $i < p$, and T_i is the first task scheduled on its processor. Moreover, if DECREASING-FIRST-FIT were to schedule another task on the processor of T_i, then this would mean that the $p-1$ other processors all have a load greater than w_i, and hence the total load would be greater than S. Then, since $w_i < \max(\frac{S}{p}, Df_{\texttt{rel}})$ and $w_i \geq \frac{S}{p}$, we have $w_i < Df_{\texttt{rel}}$ and we can execute each task at speed $f_{\texttt{rel}} = \max(f_{\texttt{rel}}, \frac{S}{pD})$ within a deadline D. Indeed, the maximal load is then w_i, by definition of T_i. Therefore, the result holds (with $\beta = 1$).

  Now suppose that $w_i < \frac{S}{p}$. In that case, if T_i was the only task executed on its processor, then we would have $l_{\text{opt}} = l_{\text{dff}} < \frac{S}{p}$, which is impossible since $S = \sum_{k=1}^{p} l_k \leq p l_{\text{opt}}$. Therefore, T_i is not the only task executed on its processor. A direct consequence of this fact is that $p+1 \leq i$. Indeed, DECREASING-FIRST-FIT schedules the p largest tasks on p distinct processors; since T_i is the last task scheduled on its processor, but not the only one, then T_i is not among the p first scheduled tasks. Also, there are only two tasks on the processor executing T_i, since $w_i > l_{\text{opt}}/3$ and the tasks scheduled before T_i have a weight at least equal to w_i. Finally, $p+1 \leq i \leq 2p$.

After scheduling task T_j on processor j for $1 \leq j \leq p$, DECREASING-FIRST-FIT schedules task T_{p+j} on processor $p-j+1$ for $1 \leq j \leq i-p$, and T_i is therefore scheduled on processor p_{2p-i+1}, together with task T_{2p-i+1}, and we have $w_i + w_{2p-i+1} = l_{\text{opt}}$. Note that because the w_j are sorted, $S \geq \sum_{j \leq i} w_j \geq i w_i$. We also have $w_{2p-i+1} < \frac{S}{p}$: indeed, when T_i was scheduled, the load of the p processors was at least equal to the load of the processor where T_{2p+i-1} was scheduled. Hence, w_{2p-i+1} cannot be greater than $\frac{S}{p}$. Then, since $w_{2p-i+1} = l_{\text{opt}} - w_i$, $w_i > l_{\text{opt}} - \frac{S}{p}$, and finally $l_{\text{opt}} - \frac{S}{p} < w_i \leq \frac{S}{i}$.

In order to find an upper bound on l_{opt}, we provide a lower bound to S, as a function of w_i:

$$\begin{aligned} S = \sum_{j=1}^{n} w_j \geq \sum_{j=1}^{i} w_j &= \sum_{j=1}^{2p-i+1} w_j + \sum_{j=2p-i+2}^{i} w_j \\ &\geq (2p-i+1) w_{2p-i+1} + (2(i-p)-1) w_i \\ &= (2p-i+1)(l_{\text{opt}} - w_i) + (2(i-p)-1) w_i \\ &= (2p-i+1) l_{\text{opt}} + (3i-4p-2) w_i = f(w_i). \end{aligned}$$

We then have $f'(w_i) = 3i - 4p - 2$, and we consider two cases.

If $f'(w_i) \geq 0$, then we have $i \geq \frac{4p+2}{3}$, and finally $S \geq i w_i > \frac{4p+2}{3}\left(l_{\text{opt}} - \frac{S}{p}\right)$. We can conclude that $l_{\text{opt}} < \frac{S}{p}\left(1 + \frac{3p}{4p+2}\right) = \frac{S}{p}\left(2 - \frac{p+2}{4p+2}\right)$.

Otherwise, $f'(w_i) < 0$ and f is a decreasing function of w_i, i.e., its minimum is reached when w_i is maximal, and $S \geq f(\frac{S}{i})$. Hence, $S \geq (2p-i+1) l_{\text{opt}} + (3i - 4p - 2)\frac{S}{i}$. Since $i \leq 2p$, $2p - i + 1 > 0$ and

$$l_{\text{opt}} \leq \frac{S}{i}\left(\frac{i - 3i + 4p + 2}{2p - i + 1}\right) = \frac{2S}{i}.$$

Finally, since $i \geq p+1$, $l_{\text{opt}} \leq \frac{2S}{p+1} = \frac{S}{p}\left(2 - \frac{2}{p+1}\right)$.

Overall, if $w_i > l_{\text{opt}}/3$, we have the bound

$$l_{\text{opt}} \leq \frac{S}{p} \times \max\left(2 - \frac{p+2}{4p+2}, 2 - \frac{2}{p+1}\right).$$

Therefore, for $\beta \geq \max\left(2 - \frac{p+2}{4p+2}, 2 - \frac{2}{p+1}\right)$, we can execute all the tasks on the processor of maximal load (and hence all the tasks) at speed $\max(f_{\texttt{rel}}, \frac{S}{pD})$ within the deadline βD in the case $w_i > l_{\text{opt}}/3$.

We can now conclude the proof of Lemma 6 by saying that for $\beta = \max\left(2 - \frac{3}{2p+1}, 2 - \frac{p+2}{4p+2}, 2 - \frac{2}{p+1}\right)$, i.e., $\beta = \max\left(2 - \frac{3}{2p+1}, 2 - \frac{p+2}{4p+2}\right)$, scheduling each task only once at speed $\max(f_{\texttt{rel}}, \frac{S}{pD})$ with the DECREASING-FIRST-FIT heuristic leads to a makespan of at most βD. □

We are now ready to prove Theorem 3.

Proof of Theorem 3. First, thanks to Proposition 5, we know that the first step of the algorithm takes decisions that are identical to the optimal solution, and therefore these tasks that are executed once, alone on their processor, have the same energy consumption than the optimal solution and the same deadline. We can therefore safely ignore them in the remaining of the proof, and consider that for each task T_i, $w_i < \max(\frac{S}{p}, Df_{\texttt{rel}})$.

In the case where $S > pDf_{\texttt{rel}}$, we use the fact that $S(\frac{S}{pD})^2$ is a lower bound on the energy (Proposition 3). Each task is executed once at speed $\max(f_{\texttt{rel}}, \frac{S}{pD}) = \frac{S}{pD}$, and therefore the energy consumption is equal to the lower bound $S(\frac{S}{pD})^2$. The bound on the deadline is obtained by applying Lemma 6.

We now focus on the case $S \leq pDf_{\texttt{rel}}$. Therefore, in the following, $\max(\frac{S}{pD}, f_{\texttt{rel}}) = f_{\texttt{rel}}$. The algorithm runs the FPTAS on a linear chain of tasks with deadline pD, and ε as defined in Equation (2). The FPTAS returns a solution on the linear chain with an energy consumption E_{FPTAS} such that $E_{\text{FPTAS}} \leq (1+\varepsilon)^2 E_{\text{chain}}$, where E_{chain} is the optimal energy consumption for TRI-CRIT-CHAIN with deadline pD on a single processor. According to Proposition 4, since the solution for the linear chain is a lower bound, the optimal solution of TRI-CRIT-INDEP is such that $E_{opt} \geq E_{\text{chain}}$.

For each task T_i, let f_i^{chain} be the speed of its execution returned by the FPTAS for TRI-CRIT-CHAIN. Note that in case of re-execution, then both executions occur at the same speed (Lemma 3). We now consider the TRI-CRIT-INDEP problem with the set of tasks $\tilde{V}$: for each task T_i, $\tilde{T}_i \in \tilde{V}$ and its weight is $\tilde{w}_i = w_i \frac{f_{\texttt{rel}}}{f_i^{\text{chain}}}$; moreover, if T_i is re-executed, we add two copies of $\tilde{T}_i$ in $\tilde{V}$. Then, $\sum_{\tilde{T}_i \in \tilde{V}} \frac{\tilde{w}_i}{f_{\texttt{rel}}} = pD$ by definition of the solution of TRI-CRIT-CHAIN.

Let $\beta = \max(2 - \frac{3}{2p+1}, 2 - \frac{p+2}{4p+2})$ be the relaxation on the deadline that we have from Lemma 6. The goal is to map all the tasks of $\tilde{V}$ at speed $f_{\texttt{rel}}$ within the deadline βD, which amounts at mapping the original tasks at the speeds assigned by the FPTAS:

- If there are tasks $\tilde{T}_i$ such that $\frac{\tilde{w}_i}{f_{\texttt{rel}}} > \beta D$, we execute them at speed $\frac{\tilde{w}_i}{\beta D}$ alone on their processor, so that they reach exactly the deadline βD. Note that in this case, the energy consumption of the algorithm becomes greater than E_{FPTAS}, since we execute these tasks faster than the FPTAS to fit on the processor.

- Tasks $\tilde{T}_i$ such that $D \leq \frac{\tilde{w}_i}{f_{\texttt{rel}}} \leq \beta D$ are executed alone on their processor at speed $f_{\texttt{rel}}$.

- For the remaining tasks and processors, we use DECREASING-FIRST-FIT as in Lemma 6. Since the previous tasks take a time of at least D in the solution of the FPTAS, and they are mapped alone on a processor, we can safely remove them and apply the lemma. Note that the number of processors may now be smaller than p, hence leading to a smaller bound β.

In the end, all tasks are mapped within the deadline βD (where β is computed with the original number of processors). There remains to check the energy consumption of the solution returned by this algorithm.

If all tasks are such that $\tilde{w}_i \leq \beta D f_{\texttt{rel}}$, $E_{algo} = E_{\text{FPTAS}} \leq (1+\varepsilon)^2 E_{\text{chain}} \leq (1+\varepsilon)^2 E_{opt}$.

According to Equation (2), $\varepsilon \leq \frac{1}{3\beta^2}$, and therefore

$$E_{algo} \leq \left(1 + \frac{2}{3\beta^2} + \frac{1}{9\beta^4}\right) E_{opt} \leq \left(1 + \frac{1}{\beta^2}\right) E_{opt}.$$

Otherwise, let $\tilde{V'}$ be the set of tasks $\tilde{T_i}$ such that $\tilde{w_i} > \beta D f_{\texttt{rel}}$. For $\tilde{T_i} \in \tilde{V'}$, $w_i > \beta D f_i^{\text{chain}}$. Since $w_i < D f_{\texttt{rel}}$ (larger tasks have been processed in the first step of the algorithm), we have $f_i^{\text{chain}} < f_{\texttt{rel}}$. This means that T_i belongs to the set of the tasks that are re-executed by the FPTAS. Hence, since we enforced an additional constraint, we have $w_i < \frac{1}{\sqrt{2}} D f_{\texttt{rel}}$. The least energy consumed for this task by any solution to TRI-CRIT-INDEP is therefore obtained when re-executing task T_i on two distinct processors at speed $\frac{w_i}{D}$, in order to fit within the deadline D. Task T_i appears two times in $\tilde{V'}$, and we let $\tilde{E}$ be the minimum energy consumption required in the optimal solution for tasks of $\tilde{V'}$: $\tilde{E} = \sum_{\tilde{T_i} \in \tilde{V'}} w_i \left(\frac{w_i}{D}\right)^2$.

The algorithm leads to the same energy consumption as the FPTAS except for the tasks of $\tilde{V'}$ that are removed from the set X of replicated tasks, and that are executed at speed $\frac{w_i}{\beta D}$:

$$E_{algo} = (S - X) f_{\texttt{rel}}^2 + (2X - \textstyle\sum_{\tilde{T_i} \in \tilde{V'}} w_i) f_{\texttt{re-ex}}^2 + \textstyle\sum_{\tilde{T_i} \in \tilde{V'}} w_i \left(\frac{w_i}{\beta D}\right)^2.$$

Since $E_{\text{FPTAS}} = (S - X) f_{\texttt{rel}}^2 + 2X f_{\texttt{re-ex}}^2$, we obtain

$$E_{algo} = E_{\text{FPTAS}} + \tfrac{1}{\beta^2}\tilde{E} - \textstyle\sum_{\tilde{T_i} \in \tilde{V'}} w_i f_{\texttt{re-ex}}^2.$$

Furthermore, $\tilde{E} \leq E_{opt}$ since it considers only the optimal energy consumption of a subset of tasks. We have $E_{\text{FPTAS}} \leq (1+\varepsilon)^2 E_{opt}$, and from Proposition 1, it is easy to see that $E_{\text{FPTAS}} \leq S f_{\texttt{rel}}^2$, i.e., E_{FPTAS} is smaller than the energy of every task executed once at speed $f_{\texttt{rel}}$. Hence, $E_{\text{FPTAS}} \leq (1+\varepsilon)^2 \min(E_{opt}, S f_{\texttt{rel}}^2)$, and since $\varepsilon < 1$, $(1+\varepsilon)^2 \leq 1 + 3\varepsilon$. Finally, $E_{\text{FPTAS}} \leq E_{opt} + 3\varepsilon S f_{\texttt{rel}}^2$. Thanks to Equation (2), $3\varepsilon S f_{\texttt{rel}}^2 \leq 2 w_{min} f_{\min}^2 \leq \sum_{\tilde{T_i} \in \tilde{V'}} w_i f_{\texttt{re-ex}}^2$ (note that there are at least two tasks in $\tilde{V'}$, since tasks are duplicated).

Finally, reporting in the expression of E_{algo},

$$\begin{aligned} E_{algo} &\leq E_{opt} + 3\varepsilon S f_{\texttt{rel}}^2 + \tfrac{1}{\beta^2} E_{opt} - \textstyle\sum_{\tilde{T_i} \in \tilde{V'}} w_i f_{\texttt{re-ex}}^2 \\ &\leq \left(1 + \tfrac{1}{\beta^2}\right) E_{opt}. \end{aligned}$$

To conclude, we point out that this algorithm is polynomial in the size of the input and in $\frac{1}{\varepsilon}$. □

We can improve the approximation ratio on the energy for large values of p. The idea is to avoid the case in which tasks are replicated by the chain but are not fitting within βD because the speed at which they are re-executed is too small. To do so, we fix a value $\varepsilon^* = \Theta\left(\frac{1}{p}\right)$, such that $0 < \varepsilon^* < 1$ for $p \geq 24$. The variant of the algorithm is used only when $p \geq 24$ (after scheduling the big tasks). The algorithm decides that the load is large enough when $S > pDf_{\texttt{rel}} \frac{1}{1+\varepsilon^*}$, leading to a $((1+\varepsilon^*)^2, \beta)$-approximation in this case. In the other case ($S \leq pDf_{\texttt{rel}} \frac{1}{1+\varepsilon^*}$), it is possible to prove that when there are tasks such that $\frac{\tilde{w_i}}{f_{\texttt{rel}}} > \beta D$, then necessarily all tasks are re-executed. Next we apply Theorem 1 while fixing values for the $f_{\text{inf},i}$'s, so as to obtain in polynomial time the optimal solution with new execution speeds, that can all be scheduled within βD using Lemma 6. Details can be found in the appendix.

5 Conclusion

In this paper, we have designed efficient approximation algorithms for the tri-criteria energy/reliability/makespan problem, using replication and re-execution to increase the reliability, and dynamic voltage and frequency scaling to decrease the energy consumption. Because of the antagonistic relation between processor speeds and reliability, this tri-criteria problem is much more challenging than the standard bi-criteria problem, which aims at minimizing the energy consumption with a bound on the makespan, without accounting for a constraint on the reliability of tasks.

We have tackled two classes of applications. For linear chains of tasks, we propose a fully polynomial time approximation scheme. However, we show that there exists no constant factor approximation algorithm for independent tasks, unless P=NP, and we are able in this case to propose an approximation algorithm with a relaxation on the makespan constraint: with a deadline at most two times larger than the original one, we can approach the optimal solution for energy consumption.

As future work, it may be possible to improve the deadline relaxation by using a FPTAS to schedule independent tasks [5] rather than DECREASING-FIRST-FIT [13]. Also, an open problem is to find approximation algorithms for the tri-criteria problem with an arbitrary graph of tasks. Even though efficient heuristics have been designed with re-execution of tasks (but no replication) by [4], it is not clear how to derive approximation ratios from these heuristics. It would be interesting to design efficient algorithms using replication and re-execution for the general case, and to prove approximation ratios on these algorithms. A first step would be to tackle fork and fork-join graphs, inspired by the study on independent tasks.

Acknowledgements: This work was supported in part by the ANR *RESCUE* project.

References

[1] N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. Approximation schemes for scheduling. In *Proceedings of SODA'97, the 8th annual ACM-SIAM Symposium On Discrete Algorithms*, pages 493–500, Philadelphia, PA, USA, 1997. Society for Industrial and Applied Mathematics.

[2] I. Assayad, A. Girault, and H. Kalla. Tradeoff exploration between reliability power consumption and execution time. In *Proceedings of SAFECOMP, the Conf. on Computer Safety, Reliability and Security*, Washington, DC, USA, 2011. IEEE CS Press.

[3] G. Aupy, A. Benoit, F. Dufossé, and Y. Robert. Reclaiming the energy of a schedule: models and algorithms. *Concurrency and Computation: Practice and Experience*, 2012.

[4] G. Aupy, A. Benoit, and Y. Robert. Energy-aware scheduling under reliability and makespan constraints. In *Proceedings of HiPC'2012, the IEEE Int. Conf. on High Performance Computing*, 2012. Also available at `gaupy.org/?paper` as INRIA Research report 7757.

[5] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer Verlag, 1999.

[6] H. Aydin and Q. Yang. Energy-aware partitioning for multiprocessor real-time systems. In *Proceedings of IPDPS, the Int. Parallel and Distributed Processing Symposium*, pages 113–121. IEEE CS Press, 2003.

[7] M. Baleani, A. Ferrari, L. Mangeruca, A. Sangiovanni-Vincentelli, M. Peri, and S. Pezzini. Fault-tolerant platforms for automotive safety-critical applications. In *Proceedings of Int. Conf. on Compilers, Architectures and Synthesis for Embedded Systems*, pages 170–177. ACM Press, 2003.

[8] N. Bansal, T. Kimbrel, and K. Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM*, 54(1):1 – 39, 2007.

[9] A. Benoit, P. Renaud-Goud, and Y. Robert. On the performance of greedy algorithms for power consumption minimization. In *Proceedings of ICPP 2011, the Int. Conf. on Parallel Processing*, pages 454 –463, Sept. 2011.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to algorithms, third edition, 2009.

[11] V. Degalahal, L. Li, V. Narayanan, M. Kandemir, and M. J. Irwin. Soft errors issues in low-power caches. *IEEE Transactions on Very Large Scale Integration Systems*, 13:1157–1166, October 2005.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[13] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429, 1969.

[14] R. Melhem, D. Mossé, and E. Elnozahy. The interplay of power management and fault recovery in real-time systems. *IEEE Transactions on Computers*, 53:2004, 2003.

[15] A. J. Oliner, R. K. Sahoo, J. E. Moreira, M. Gupta, and A. Sivasubramaniam. Fault-aware job scheduling for bluegene/l systems. In *Proceedings of IPDPS, the Int. Parallel and Distributed Processing Symposium*, pages 64–73, 2004.

[16] P. Pop, K. H. Poulsen, V. Izosimov, and P. Eles. Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems. In *Proceedings of CODES+ISSS, the IEEE/ACM Int. Conf. on Hardware/software codesign and system synthesis*, pages 233–238, 2007.

[17] S. M. Shatz and J.-P. Wang. Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems. *IEEE Transactions on Reliability*, 38:16–27, 1989.

[18] Y. Zhang and K. Chakrabarty. Energy-aware adaptive checkpointing in embedded real-time systems. In *Proceedings of DATE, the Conf. on Design, Automation and Test in Europe*, page 10918, 2003.

[19] D. Zhu and H. Aydin. Energy management for real-time embedded systems with reliability requirements. In *Proceedings of ICCAD, the IEEE/ACM Int. Conf. on Computer-Aided Design*, pages 528–534, 2006.

[20] D. Zhu, R. Melhem, and D. Mossé. The effects of energy management on reliability in real-time embedded systems. In *Proceedings of ICCAD, the IEEE/ACM Int. Conf. on Computer-Aided Design*, pages 35–40, 2004.

Appendix: $(1+\Theta(\frac{1}{p}), 2-\Theta(\frac{1}{p}))$-approximation algorithm for TRI-CRIT-INDEP

This algorithm is used only for $p \geq 24$, and we define:

$$K = 1 - \frac{1}{c(2\beta\sqrt{2}-1)};$$

$$\varepsilon^* = \frac{1}{\sqrt{2}cpK - 1}.$$

Recall that $\beta = \max(2 - \frac{3}{2p+1}, 2 - \frac{p+2}{4p+2})$. The value β is therefore increasing with p, and for $p \geq 24$, we have $\beta \geq 1.9$. Furthermore, $c \approx 0.2838$ and $K \geq 0.2$. Finally, since $p \geq 24$, $0 < \varepsilon^* < 1$.

Modifications to the original algorithm.
The handling of *big* tasks is identical. However, we do not use replication when $S > pDf_{\texttt{rel}}\frac{1}{1+\varepsilon^*}$: we schedule tasks at speed $\max(f_{\texttt{rel}}, \frac{S}{pS})$ using DECREASING-FIRST-FIT. Proposition 6 below shows that we obtain the desired guarantee in this case. In the other case ($S \leq pDf_{\texttt{rel}}\frac{1}{1+\varepsilon^*}$), we apply the FPTAS with the parameter ε^*. It is now possible to show that (i) either we can schedule all tasks with the speeds returned by the FPTAS within the deadline βD; (ii) or there is at least one task that does not fit, but then all tasks are re-executed and we can find an optimal solution that can be scheduled thanks to Theorem 1. The correction of this case is proven in Proposition 7.

Proposition 6. *For the problem* TRI-CRIT-INDEP *where each task T_i is such that $w_i < \max(\frac{S}{p}, Df_{rel})$, if $(1+\varepsilon^*)\frac{S}{pD} > f_{rel}$, then scheduling each task only once at speed $\max(f_{rel}, \frac{S}{pD})$ with* DECREASING-FIRST-FIT *is a $\left((1+\varepsilon^*)^2, \beta\right)$-approximation algorithm, with $\beta = \max\left(2-\frac{3}{2p+1}, 2-\frac{p+2}{4p+2}\right)$.*

Proof. We use the fact that $S(\frac{S}{pD})^2$ is a lower bound on the energy (Proposition 3). If each task is executed once at speed $\max(f_{\texttt{rel}}, \frac{S}{pD})$, since $f_{\texttt{rel}} < (1+\varepsilon)\frac{S}{pD}$, then the energy consumption is at most at a ratio $(1+\varepsilon^*)^2$ of the value of the optimal energy consumption. The bound on the deadline is obtained by applying Lemma 6. □

Proposition 7. *For the problem* TRI-CRIT-INDEP *where each task T_i is such that $w_i < \max(\frac{S}{p}, Df_{rel})$, if $S \leq pDf_{rel}\frac{1}{1+\varepsilon^*}$, then there is a $\left((1+\varepsilon^*)^2, \beta\right)$-approximation algorithm, with $\beta = \max\left(2-\frac{3}{2p+1}, 2-\frac{p+2}{4p+2}\right)$.*

Proof. Similarly to the original algorithm, we use the FPTAS and we obtain a $\left((1+\varepsilon^*)^2, \beta\right)$-approximation algorithm unless there is a task T_i such that $\frac{\tilde{w_i}}{f_{\texttt{rel}}} > \beta D$, and hence $\frac{w_i}{f_i^{\text{chain}}} > \beta D$. Since $w_i < Df_{\texttt{rel}}$ (larger tasks have been processed in the first step of the algorithm), we have $f_i^{\text{chain}} < f_{\texttt{rel}}$. This means that T_i belongs to the set of the tasks that are re-executed by APPROX-CHAIN. Hence, since we enforced an additional constraint, we have $w_i < \frac{1}{\sqrt{2}}Df_{\texttt{rel}}$. Finally,

$$f_i^{\text{chain}} = f_{\texttt{re-ex}} < \frac{w_i}{\beta D} < \frac{1}{\sqrt{2}\beta}f_{\texttt{rel}}. \quad (3)$$

Let X_{chain} be the total weight of the re-executed tasks (X_1 or X_2 in APPROX-CHAIN), and let $X_{\text{opt}} = c(pDf_{\texttt{rel}} - S)$ be the optimal weight to solve TRI-CRIT-CHAIN with one processor. We compute $X_{\text{opt}} - X_{\text{chain}}$. By definition of $f_{\texttt{re-ex}}$ (Corollary 1), the optimal speed at which each re-execution should occur, we have:

$$pD = \frac{S - X_{\text{chain}}}{f_{\texttt{rel}}} + \frac{2X_{\text{chain}}}{f_{\texttt{re-ex}}} = \frac{S - X_{\text{opt}}}{f_{\texttt{rel}}} + \frac{2X_{\text{opt}}}{f_{\text{opt}}},$$

where $f_{\text{opt}} = \frac{2c}{1+c} f_{\texttt{rel}}$ (Corollary 1 applied to X_{opt}). We now express $X_{\text{opt}} - X_{\text{chain}}$:

$$\left(\frac{2}{f_{\texttt{re-ex}}} - \frac{1}{f_{\texttt{rel}}}\right) X_{\text{chain}} = \left(2\frac{1+c}{2c}\frac{1}{f_{\texttt{rel}}} - \frac{1}{f_{\texttt{rel}}}\right) X_{\text{opt}},$$

and therefore $X_{\text{chain}} = \frac{f_{\texttt{re-ex}}}{c(2f_{\texttt{rel}} - f_{\texttt{re-ex}})} X_{\text{opt}}$, and finally $X_{\text{opt}} - X_{\text{chain}} = \left(1 - \frac{f_{\texttt{re-ex}}}{c(2f_{\texttt{rel}} - f_{\texttt{re-ex}})}\right) X_{\text{opt}}$, that is minimized when $f_{\texttt{re-ex}}$ is maximized. Applying the upper bound on $f_{\texttt{re-ex}}$ from Equation (3), we obtain:

$$X_{\text{opt}} - X_{\text{chain}} > \left(1 - \frac{1}{c(2\beta\sqrt{2} - 1)}\right) X_{\text{opt}} = K \times X_{\text{opt}} \ .$$

Since $\frac{S}{pD} \leq \frac{1}{1+\varepsilon^*} f_{\texttt{rel}}$, we have $\frac{S}{pD} \leq \left(1 - \frac{1}{\sqrt{2}cpK}\right) f_{\texttt{rel}}$, and $f_{\texttt{rel}} - \frac{S}{pD} \geq \frac{f_{\texttt{rel}}}{\sqrt{2}cpK}$. Since $X_{\text{opt}} = c(pDf_{\texttt{rel}} - S)$ and $K > 0$, we obtain $K \times X_{\text{opt}} \geq \frac{1}{\sqrt{2}} Df_{\texttt{rel}}$, and therefore we have $X_{\text{opt}} - X_{\text{chain}} > \frac{1}{\sqrt{2}} Df_{\texttt{rel}}$. This means that each task that can be re-executed in any solution to TRI-CRIT-INDEP is indeed re-executed in the solution given by APPROX-CHAIN, since all these tasks have a weight lower than $\frac{1}{\sqrt{2}} Df_{\texttt{rel}}$. Since X_{opt} is greater than the total weight of the tasks that can be re-executed, we can use Theorem 1 in the case $p = 1$, on the subset of tasks T_i such that $w_i \leq \frac{1}{\sqrt{2}} Df_{\texttt{rel}}$. The other tasks are executed once at speed $f_{\texttt{rel}}$. We define $f_{\text{inf},i} = \frac{w_i}{1.9D}$, so that $f_{\text{inf},i} < \frac{1}{1.9\sqrt{2}} f_{\texttt{rel}} < \frac{2c}{1+c} f_{\texttt{rel}}$ and we can apply Theorem 1. Then, in polynomial time, we have the optimal solution with new execution speeds: $\tilde{f}_i^{\text{chain}}$. Furthermore for each task T_i, necessarily

$$\frac{w_i}{\tilde{f}_i^{\text{chain}}} \leq \frac{w_i}{f_{\text{inf},i}} = 1.9D.$$

Note that since $p \geq 24$, we have $\beta \geq 1.9$, and $\frac{w_i}{\tilde{f}_i^{\text{chain}}} \leq \beta D$. We can therefore schedule the new tasks $\tilde{T}_i$ within the deadline relaxation using DECREASING-FIRST-FIT, as a direct consequence of Lemma 6. □

We can conclude by stating that thanks to Propositions 6 and 7, since ε^* is in $\Theta(\frac{1}{p})$ and β is in $2 - \Theta(\frac{1}{p})$, this algorithm is a $(1 + \Theta(\frac{1}{p}), 2 - \Theta(\frac{1}{p}))$-approximation. Indeed, $\varepsilon^* < 1$ and therefore $(1 + \varepsilon^*)^2 < 1 + 3\varepsilon^*$.

Furthermore, the algorithm is polynomial in the size of the input and in $\frac{1}{\varepsilon^*}$.

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399