

Scheduling with Storage Constraints

Érik Saule

Pierre-François Dutot

Grégory Mounié

LIG*– MOAIS Team
 Grenoble Universités, France
 {firstname.lastname}@imag.fr

Abstract

Cumulative memory occupation is a quite intuitive but not so studied constraint in scheduling. The interest in such a constraint is present in multi-System-on-Chip, embedded systems for storing instruction code, or in scientific computation for storing results. Memory occupation seen as a constraint is impossible to solve with approximation algorithms. We believe that transforming the constraint into a second objective to optimize helps to deal with such constraints.

The problem addressed in this paper is to schedule tasks on identical processors in order to minimize both maximum completion time and maximum cumulative memory occupation. For independent tasks, a family of algorithms with good approximation ratios based on a PTAS is given. Several approximation ratios are proved to be impossible to achieve with any schedule. The precedence constrained case is then studied and a family of performance guaranteed algorithms based on List Scheduling is proposed. Finally, optimizing the mean completion time as a third objective is also studied and a tri-objective algorithm is given.

1 Introduction

The total execution time is the most studied objective in computer science optimization problems. From a global point of view this objective satisfies most of the users. In practice, this quantity is one of the most difficult to decrease, as adding new computing resources is not always sufficient. Some problems involving communication times are even hard for an unlimited number of resources.

Storage capacity is one of the other basic needs. Most of the time, it is much easier to increase the available capacity than to produce memory-aware schedules. The additional

financial cost is generally linear with the capacity increase. However increasing the memory is not always possible. For instance, in a multi-System-on-Chip (SoC) embedded system, every SoC has a limited storage capacity per processor for storing instructions. In such a context, code replication for online optimization can make memory constraints a key issue [5]. While the contexts of limiting time and memory usage appear to be quite different they arise jointly in some applications, such as large physics applications [4]. In all cases scheduling techniques can be applied to address both problems, extending to several models of architecture.

This article addresses the tasks multi-processor scheduling problem with two simultaneous objectives: total execution time and memory usage. The processing time of every task is not related to the memory it uses. The problem of scheduling independent tasks with strict memory constraint is a strongly related problem. However, this problem is intractable through the approximation theory point of view as detailed in Section 2. Studying the bi-objective scheduling problem will help to deal with the strictly constrained problem.

This article proposes two families of parameterized approximation algorithms to compute a guaranteed solution for tasks scheduling problem on multiprocessor, optimizing both makespan and maximum memory consumption over all processors. The first one computes a $(1 + \Delta + \epsilon, 1 + \frac{1}{\Delta} + \epsilon)$ -approximated schedule where $\Delta > 0$ is the parameter of the algorithm. This family of algorithms is only valid for independent tasks and can not be extended to other models or objective functions.

The second one computes a $(2 + \frac{1}{\Delta-2} - \frac{\Delta-1}{m(\Delta-2)}, \Delta)$ -approximated solution for $\Delta > 2$ on the DAG scheduling problem. It can be transformed into a tri-objective algorithm on independent tasks, which also optimizes the mean completion time of tasks with a performance ratio of $(2 + \frac{1}{\Delta-2})$.

The article also proposes an analysis of the best achievable performance ratio. It is proven that no algorithm can have an approximation ratio bet-

*LIG is supported by CNRS, INPG, INRIA, UJF and UPMF. See <http://lig.imag.fr>. Part of this work was supported by the "CoreGRID" Network of Excellence.

ter than $(\frac{3}{2}, \frac{3}{2})$. Approximation ratios better than $(1 + \frac{i}{km}, 1 + (m-1)(1 - \frac{i}{k}))$, $\forall m, k \geq 2, i \in \{0, \dots, k\}$ have also been proved to be impossible.

The remainder of the article is organized as follows. Section 2 states formally the bi-objective scheduling problem and discusses the model. An approximation algorithm for independent tasks is given in Section 3. Some bounds on best approximation ratio achievable are detailed in Section 4. The general cases of DAG scheduling is studied and an tri-objective extension on independent tasks is detailed in Section 5. Existing works in the field of multi-objective optimization are described in Section 6. Concluding remarks, including a discussion about the resolution of the original problem with the method we propose, are given in Section 7.

2 Preliminaries

2.1 Formal statement

As we start our study with independent tasks, we will begin by presenting the formal definitions.

Let $T = \{t_1, \dots, t_n\}$ be a set of n tasks. Task i takes p_i time units to execute and has a code of s_i memory units. Let Q be a set of m identical processors. A schedule $\pi : T \rightarrow Q$ assigns each task to a processor. We denote by $C_{max}^\pi = \max_{q \in Q} \sum_{\pi(i)=q} p_i$ the completion time of the last task to be executed. We denote by $M_{max}^\pi = \max_{q \in Q} \sum_{\pi(i)=q} s_i$ the maximum memory consumption of a processor. The parameter π will be omitted when no confusion is possible. Notice that, with independent tasks M_{max} and C_{max} are strictly equivalent and can be exchanged. Thus, we can see the memory consumption as a second time line. However, we believe that having two different notations will help the reader not to confuse both objectives. With independent tasks, all results are symmetric.

The problem we tackle is to minimize both C_{max} and M_{max} at the same time and could be denoted in the Lawler notation as $P | p_j, s_j | C_{max}, M_{max}$. Recall that $P | p_j | C_{max}$ is strongly NP-complete [7]. The optimal makespan (resp. memory consumption) is denoted by C_{max}^* (resp. M_{max}^*).

In Section 5, we will tackle the problem with precedence constraints. Thus, we introduce the starting time $\sigma(i)$ of task i and its completion time $C_i = \sigma(i) + p_i$. A processor can execute only one task at a time. A task can not be executed until all its predecessors are completed. Predecessors (resp. successors) of task i will be denoted $pred(i)$ (resp. $succ(i)$). The completion time of the last task to be executed is now denoted by $C_{max}^\pi = \max_{i \in T}(C_i)$. With precedence constraints, the problem is denoted

$$P | p_j, s_j, prec | C_{max}, M_{max}.$$

2.2 Changing Objectives into Constraints

We briefly discuss here why we chose to do bi-objective optimization while the related industrial problem is to find a schedule that minimizes C_{max} with respect to $M_{max} \leq M$. The main reason is that this problem can not be approximated within a constant factor. Indeed, deciding if a schedule exists such that $M_{max} \leq M$ is a strongly NP-complete problem as it is the decision version of $P | p_j | C_{max}$ [7]. An approximation algorithm should always be able to find a valid schedule or ensure that none exists. Thus, it is impossible (unless P=NP) to find an approximation algorithm for such a problem that runs in polynomial time.

3 Symmetric Bi-objective Approximation Algorithm

3.1 Principle

The idea of the algorithm is to combine the results of two algorithms, each dedicated to a single objective. Since each algorithm will be run with all the tasks as input, we eventually have to choose between two possible allocations for each task. This choice will be made by setting a threshold on the ratio between execution time and memory. Intuitively, if a task needs a lot of memory and is quickly executed, it needs to be scheduled with memory as a primary concern. Conversely, a long task which does not require a lot of memory has to be scheduled with an algorithm optimizing the makespan.

Since memory and execution times taken separately are two similar objectives, we can use the same algorithm for both schedules replacing the p_i values used to minimize the makespan in the first schedule by s_i values. Some good approximation algorithms for the $P | p_j | C_{max}$ problem are known. Graham proposed List Scheduling [8], which is a $2 - \frac{1}{m}$ approximation algorithm which is recalled in Section 5. A PTAS based on the Subset Sum problem has also been proposed in [9].

3.2 Algorithm

The algorithm described above is more formally written in pseudo code as Algorithm 1, and will be named SBO_Δ (for Symmetric Bi-Objective) in the rest of the paper where Δ is strictly greater than zero. As described in the algorithm, π_1 is a schedule produced by a ρ_1 approximation algorithm on the makespan and π_2 is a schedule produced

by a ρ_2 approximation algorithm on the memory consumption. To simplify the notations, we will note C the guaranteed makespan produced by the first algorithm ($C_{max}^{\pi_1}$) and M the maximum memory used by the second algorithm ($M_{max}^{\pi_2}$).

Algorithm 1 SBO_{Δ}

Input: m : an integer

$\{p_1, \dots, p_n\}$: n integers

$\{s_1, \dots, s_n\}$: n integers

Begin

Let π_1 be a ρ_1 -approximated schedule for C_{max}

Let C be $C_{max}^{\pi_1}$

Let π_2 be a ρ_2 -approximated schedule for M_{max}

Let M be $M_{max}^{\pi_2}$

For $i = 1$ **to** n

if $\frac{p_i}{C} < \Delta \frac{s_i}{M}$

then $\pi_{\Delta}(i) = \pi_2(i)$

else $\pi_{\Delta}(i) = \pi_1(i)$

End For

Return π_{Δ}

End

PROPERTY 1 *The schedule π_{Δ} generated by Algorithm 1 is a $(1 + \Delta)\rho_1$ -approximated schedule on the makespan.*

Proof: Let us note S_1 the set of tasks allocated according to π_1 , and S_2 the set of tasks allocated according to π_2 . The total execution time for each processor is the sum of the execution time of the tasks of both sets allocated to that processor. For set S_1 , this sum of execution time is easily bounded by $C_{max}^{\pi_1}$. For set S_2 , we have on the one hand the fact that for each task of S_2 the ratio between execution time and memory is lower than $\Delta \frac{C}{M}$, and on the other hand that the total memory used on each processor by allocation π_2 is bounded by M . Combining those two facts, for each processor k we obtain:

$$\begin{aligned} \sum_{i \in S_2, \pi_2(i)=k} p_i &< \sum_{i \in S_2, \pi_2(i)=k} \Delta C \frac{s_i}{M} \\ &= \Delta C \left(\sum_{i \in S_2, \pi_2(i)=k} \frac{s_i}{M} \right) \leq \Delta C_{max}^{\pi_1} \end{aligned}$$

As $C_{max}^{\pi_1}$ is less or equal to $\rho_1 C_{max}^*$, this concludes the proof. \square

PROPERTY 2 *The schedule π_{Δ} generated by Algorithm 1 is a $(1 + 1/\Delta)\rho_2$ -approximated schedule on memory.*

The proof is similar to the proof of Property 1 and is omitted.

COROLLARY 1 *For any fixed value $\epsilon > 0$, there exist polynomial algorithms with approximation ratio of $(1 + \Delta + \epsilon, 1 + \frac{1}{\Delta} + \epsilon)$. Moreover, there always exists a solution whose makespan and memory consumption are respectively lower or equal to $2C_{max}^*$ and $2M_{max}^*$.*

The corollary comes directly from the known PTAS for $P \parallel C_{max}$.

4 Impossible Approximation Ratios

In this section, we deal with the inapproximability of the problem with a single solution. We will explain the principle on a simple result in Section 4.1, which is generalized in Section 4.2. We will then provide another interesting instance which will give us additional impossibilities.

4.1 Introducing the Concept

Let us consider an instance with 2 processors and 3 tasks with: $p_1 = 1, p_2 = p_3 = \frac{1}{2}$ and $s_1 = \epsilon, s_2 = s_3 = 1$. There are only 3 possible schedules (by removing schedules with idle time and symmetric schedules): In the first one, task 1 is scheduled in parallel with task 2 and 3. In the second one, task 1 and 2 are scheduled on the same processor. In the third one, all tasks are scheduled on the same processor. Those 3 solutions lead to 3 objective values: $(1, 2)$, $(\frac{3}{2}, 1 + \epsilon)$ and $(2, 2 + \epsilon)$. The last one is Pareto dominated by the two others. In Figure 1, we represented the two dominating schedules as Gantt charts (with time as rectangle length), and the additional information of memory consumption as labels on the tasks. For this instance, we have $C_{max}^* = 1$ and $M_{max}^* = 1 + \epsilon$. Suppose that we have a $(1, \frac{7}{4})$ -approximation algorithm. It will compute on this instance a solution with $C_{max} \leq 1$ and $M_{max} \leq \frac{7}{4}(1 + \epsilon)$. No such solution exists for this instance. Thus, such an algorithm can not exist. In these results, ϵ may be as small as needed. More generally, there are no algorithms with a performance ratio better than $(1, 2)$. Once again, the result is symmetric.

LEMMA 1 *No approximation algorithm can be better than $(1, 2)$ or $(2, 1)$.*

4.2 Extending the Idea to m Processors

To go further, this result can be expressed for more than two processors and more than three tasks. With m processors, we construct a similar instance using $km + m - 1$ tasks. As previously, we use two types of tasks. The $m - 1$ first tasks are identical and are defined such that $p_1 = \dots = p_{m-1} = 1$ and $s_1 = \dots = s_{m-1} = \epsilon$, while the km other

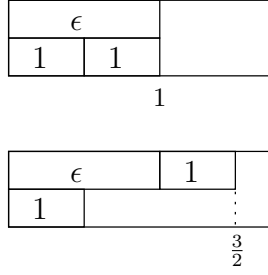


Figure 1. The two Pareto optimal schedules for the first instance (sizes are according to durations).

tasks are defined such that $p_m = \dots = p_{km+m-1} = \frac{1}{km}$ and $s_m = \dots = s_{km+m-1} = 1$. The optimal makespan is therefore 1, and the optimal memory consumption is $k + \epsilon$.

A straightforward case analysis shows that there are $k+1$ Pareto optimal schedules. Solution $i \in \{0, 1 \dots k\}$ schedules i tasks of the second type and one of the first type on each of the first $m-1$ processors, scheduling the remaining tasks (which are $km - i(m-1)$ tasks of the second type) on the last processor. Solution i has a makespan of $1 + \frac{i}{km}$ and a memory consumption of $k + (k-i)(m-1)$ unless i equals to k . Solution k has a maximum memory consumption of $k + \epsilon$. Again, there are no algorithms with an approximation ratio better than $(1 + \frac{i}{km}, 1 + (m-1)(1 - \frac{i}{k}))$.

LEMMA 2 $\forall m, k \geq 2, i \in \{0, \dots, k\}$ there are no algorithms with an approximation ratio better than $(1 + \frac{i}{km}, 1 + (m-1)(1 - \frac{i}{k}))$.

Notice that $\frac{i}{k}$ can reach all values between 0 and 1. Thus, the inapproximability result is continuous. Furthermore, we can also produce symmetric results by swapping memory consumption and processing times.

Results obtained for small values of m are depicted in Figure 3 along with the results of the following section.

4.3 More Impossibilities

In this second instance, we will again consider only two processors and three tasks. The computation times and memory requirements of the tasks are $p_1 = 1, p_2 = \epsilon, p_3 = 1 - \epsilon$ and $s_1 = \epsilon, s_2 = 1, s_3 = 1 - \epsilon$. As in Section 4.1, there are only a few possible schedules. The three possibly Pareto optimal schedules are obtained by grouping two tasks on one processor and executing the third one alone. Executing all three tasks on one processor is dominated by each of these schedules. The values of the three schedules are $(1, 2 - \epsilon)$ when tasks 1 is alone, $(1 + \epsilon, 1 + \epsilon)$ when task 3 is alone and $(2 - \epsilon, 1)$ when task 2 is alone. Since C_{max}^* and M_{max}^* are both equal to 1, these values are also

minimum possible approximation ratios. Remark that the $(1 + \epsilon, 1 + \epsilon)$ point is Pareto optimal only when ϵ is less than $\frac{1}{2}$. For values of ϵ close to $\frac{1}{2}$ this instance proves the following lemma:

LEMMA 3 No algorithm can have a better approximation ratio than $(\frac{3}{2}, \frac{3}{2})$.

The corresponding Pareto optimal schedules are represented in Figure 2, with respective memory consumption written as labels on the tasks.

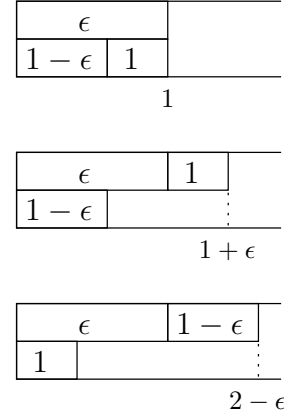


Figure 2. The three Pareto optimal schedules for the second instance.

This particular result cannot be extended as we did for the previous one by introducing a large number of tasks since all the tasks are different and splitting a task would lead to better schedules (which in turn leads to less interesting inapproximability results).

This lemma and the previous one (for values of m between 2 and 6) are illustrated in Figure 3. Note that in this figure, it may be possible to find algorithms achieving the values on the border of the designated domain. To complete the illustration, we included as a dashed curve the approximation ratio curve obtained in Section 3.

5 General Case

While the independent tasks case models some grid computing problems, DAG scheduling is more suitable for embedded system applications. SBO_{Δ} presented in Section 3 can not be extended to the DAG scheduling problem. Thus, we design a new algorithm to deal with precedence constraints. This algorithm is detailed and analyzed in Section 5.1. Then, we present in Section 5.2 an extension of this algorithm that is able to optimize a third objective, namely the sum of completion time, on independent tasks.

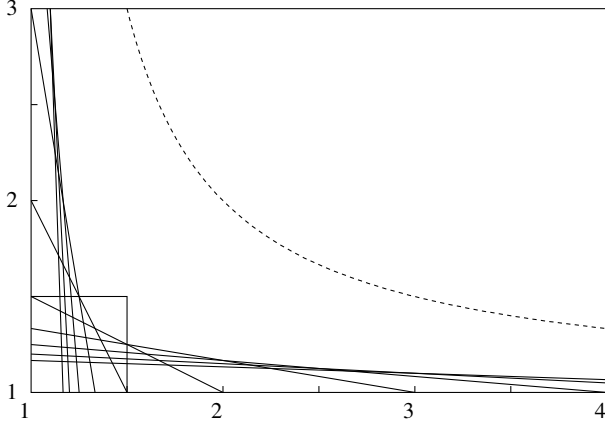


Figure 3. Impossibility domain for minimizing both makespan and memory consumption ratios. The dashed curve represents approximation ratios from Section 3.

5.1 An Algorithm Based on List Scheduling

The List Scheduling algorithm was the first heuristic which have been proved to be efficient with respect to the makespan, with a fixed performance ratio on C_{max} . This proof was then extended to the precedence constraints case by Graham in [8]. Its principle is to ensure that when processors are experiencing idle times there are no available tasks. The crucial point is that the approximation ratio came from the sum of two basic lower bounds of the achievable makespan: the critical path and the total work divided by the number of processors. The algorithm we propose is inspired from List Scheduling.

Algorithm RLS_{Δ} (for Restricted List Scheduling), given in Algorithm 2, computes a lower bound LB on the optimum value of M_{max} which is the well known Graham lower bound. Then a degradation factor of Δ is allowed on M_{max} : no processor is allowed to use more memory than ΔLB . We schedule iteratively the task which can start the soonest without violating the memory constraint. We can use an arbitrary total ordering of tasks to break ties.

For the analysis of this algorithm, we are first interested in the number of processors that have been marked, that is the number of processor that at some point have not been chosen by the algorithm because they were too loaded on M_{max} .

LEMMA 4 *The number of marked processor is less or equal to $\lfloor \frac{m}{\Delta-1} \rfloor$.*

Proof: For each marked processor j , a task i exists such that $memsiz[e[j] > \Delta LB - s_i \geq (\Delta - 1)LB$ (the

Algorithm 2 RLS_{Δ}

Input: m : an integer

$\{p_1, \dots, p_n\}$: n integers

$\{s_1, \dots, s_n\}$: n integers

Begin

compute $LB = \max(\max_i s_i, \sum_i \frac{s_i}{m})$

$load[1] = \dots = load[m] = 0$

$memsiz[e[1] = \dots = memsize[m] = 0$

Until all tasks are scheduled

For each ready task i

Let j be the processor minimizing $load[j]$

Let $proc[i] = j$

such that $memsiz[e[j] + s_i \leq \Delta LB$

Let $ready[i] = \max(\max_{i' \in prec(i)} \sigma(i') + p_{i'}, load[j])$

*/*for analysis only : */*

mark processor j' such that $load[j'] < load[j]$

End For

Let i^* be the task minimizing $ready[i]$

$\pi(i^*) = proc[i^*]$

$\sigma(i^*) = ready[i^*]$

$load[proc[i^*]] = \sigma(i^*) + p_{i^*}$

$memsiz[e[proc[i^*]] += s_{i^*}$

End Until

Return (π, σ)

End

last inequality came from $s_i \leq LB$). Suppose there are more than $\frac{m}{\Delta-1}$ marked processors. As they were marked, $\sum_{j \in \text{marked}} memsize[j] > \frac{m}{\Delta-1}(\Delta - 1)LB = mLB$. But $\sum_{j \in \text{marked}} memsize[j] \leq \sum_i s_i \leq mLB$. Thus, $mLB \geq \sum_{j \in \text{marked}} memsize[j] > mLB$ which is impossible. Thus, there are less than $\frac{m}{\Delta-1}$ marked processor.

□

This last lemma directly implies that the algorithm can not take as input values of Δ lower or equal to 2. Indeed, for those small values there might be a task which cannot be placed on any processor due to a large memory requirement, and consequently marking all processors. Thus, we have to choose values of Δ greater than 2 for which the algorithm yields schedules with M_{max} lower or equal to ΔLB . This leads to the following corollary.

COROLLARY 2 RLS_{Δ} is Δ -approximate on M_{max} if $\Delta \geq 2$.

Since $\lfloor \frac{m}{\Delta-1} \rfloor$ processors could have been discarded due to memory constraints, it means that at least $\lceil m \frac{\Delta-2}{\Delta-1} \rceil$ processors are freely used to optimize C_{max} which is sufficient to be guaranteed on C_{max} .

LEMMA 5 RLS_{Δ} is $(2 + \frac{1}{\Delta-2} - \frac{\Delta-1}{m(\Delta-2)})$ -approximate on C_{max} if $\Delta > 2$.

Proof: We consider a partition $CP \cup W$ of the time units between 0 and C_{max} . A time unit t belongs to CP if a processor is idle at t and all tasks i scheduled after were not ready at t : $\max_{i' \in prec(i)} \sigma(i') + p_{i'} > t$. Remark that the cardinality of CP is less than the processing time of any chain in the DAG. Thus, $|CP| \leq C_{max}^*$.

All the processing requirement $\sum p_i$ is scheduled in W except the part which has been executed in CP which is greater than $|CP|$ (this lower bound is tight if only one processor is active during CP). Recall that Lemma 4 implies that at least $m \frac{\Delta-2}{\Delta-1}$ processors are never constrained on the memory. Thus, the length of W is smaller than $\frac{\sum p_i - |CP|}{m \frac{\Delta-2}{\Delta-1}}$.

Since the schedule is partitioned into two sets of time units W and CP , we can write the length of the schedule as the sum of the length of both parts:

$$C_{max} = |W| + |CP| \leq \left(1 + \frac{1}{\Delta-2}\right) \frac{\sum p_i}{m} + \left(1 - \frac{\Delta-1}{m(\Delta-2)}\right) |CP|$$

Recall that $\frac{\sum p_i}{m}$ and $|CP|$ are lower bounds of C_{max}^* . Replacing them in the previous equation, we get:

$$C_{max} \leq \left(2 + \frac{1}{\Delta-2} - \frac{\Delta-1}{m(\Delta-2)}\right) C_{max}^*$$

□

We can now state the approximation ratio of RLS_{Δ} :

COROLLARY 3 RLS_{Δ} is a $(2 + \frac{1}{\Delta-2} - \frac{\Delta-1}{m(\Delta-2)}, \Delta)$ approximation algorithm, with $\Delta > 2$. Its time complexity is $O(n^2m)$.

Note that since n is greater than m this complexity is polynomial in the size of the instance. For an easier comparison to the results obtained in Section 3, we can replace Δ with $2 + \Delta'$ and write the approximation ratio as $(2 + \frac{1}{\Delta'} - \frac{\Delta'+1}{m\Delta'}, 2 + \Delta')$.

5.2 A Three Criteria Extension on Independent Tasks

It is sometimes useful in grid computing to obtain some early results in order to ensure that an application behaves as the user expects. From a scheduling point of view, this behavior can be achieved by optimizing the mean completion time of jobs which is equivalent to optimizing the sum of completion time. Thus, we are interested in this Section in optimizing three objectives namely, makespan, maximum memory consumption and sum of completion time,

when scheduling independent tasks. Once again, the algorithm presented in Section 3 can not be adapted to optimize the sum of completion time. However, the List Scheduling based algorithm presented previously allows to consider tasks in the SPT order which ensure a guarantee on $\sum C_i$. Recall that a List Scheduling using SPT is optimal on $\sum C_i$ on an arbitrary number of processors.

We first consider the degradation in $\sum C_i$ when a fraction of processor is forbidden.

LEMMA 6 Let π_1, π_2 be two SPT schedules on m and ρm processors of the same set of tasks ($0 < \rho \leq 1$) then $\sum C_i^{\pi_2} \leq (\frac{1}{\rho} + 1) \sum C_i^{\pi_1}$.

Proof: We prove the lemma, by proving that $\forall j, C_j^{\pi_2} \leq (\frac{1}{\rho} + 1) C_j^{\pi_1}$.

Without loss of generality, tasks are indexed according to the SPT rule. In a partial SPT schedule, when task j is scheduled, it is the last task to complete. Thus, $\frac{1}{m} \sum_{k=1, \dots, j} p_k \leq C_j^{\pi_1}$.

With ρm processors, when j starts, only the $j-1$ first tasks have been scheduled. Thus, $s_j^{\pi_2} \leq \frac{1}{\rho m} \sum_{k=1}^{j-1} p_k$, which leads to $C_j^{\pi_2} \leq \frac{1}{\rho m} \sum_{k=1}^{j-1} p_k + p_j \leq \frac{1}{\rho} C_j^{\pi_1} + p_j$. As the completion time $C_j^{\pi_1}$ of task j is greater than its execution time p_j , we have $C_j^{\pi_2} \leq (\frac{1}{\rho} + 1) C_j^{\pi_1}$ □

In RLS_{Δ} , $m \frac{\Delta-2}{\Delta-1}$ processors are always available. Thus, we could apply the previous lemma for this particular value of ρ . Remember that SPT is optimal on $\sum C_i$. Thus, having additional processors can not degrade the $\sum C_i$ objective. From all those result, we can state the approximation ratio of RLS_{Δ} if the SPT order is chosen to break ties on independent tasks.

COROLLARY 4 Using SPT as the order of tasks in RLS_{Δ} leads to a $(2 + \frac{1}{\Delta-2} - \frac{\Delta-1}{m(\Delta-2)}, \Delta, 2 + \frac{1}{\Delta-2})$ -approximated solution on $(C_{max}, M_{max}, \sum C_i)$

6 Related Works

Multi-objective optimization is a quite recent topic. In the approximation algorithms field, two main approaches are used to deal with such problems.

The first one is the absolute approximation technique. An algorithm computes a solution which approximates all objectives at the same time. In scheduling, it is usual to optimize two objectives by mixing a schedule efficient on the first objective and a second one efficient on the second objective. Such a method has been used in [12] to derive a framework for absolute approximation on both C_{max} and $\sum C_i$. It has been used to deal with a bi-objective scheduling problem with deadlines [3]. Recently, a similar approach have been used to schedule independent tasks

to optimize both sum of completion time and weighted sum of completion time on a single machine [1]. Some *ad hoc* methods also exist. For instance, Laforest [10] proposes a tri-criteria approximation algorithm for a network construction problem.

The second one is the Pareto set approximation. The idea is to give the set of all Pareto solutions. However, this set can be of exponential cardinality. Thus, we use an approximated set of solutions. For an optimization problem whose decision version belongs to NP, it is proven that there exists a polynomial approximated set [11]. This approach has been studied for scheduling problems such as [2]. The approximation of Pareto set is mainly interesting when no absolute approximation algorithm exists. In the grid computing and embedded systems community, the problem of optimizing the reliability of the system as well as its efficiency is such a problem [6].

In this work, we focused on the absolute approximation technique. Indeed, using the Pareto set approximation technique implies to choose between several interesting solutions. This choice is often difficult to do automatically and, thus, requires a human decision maker. However, all algorithms we provide can be tuned using the Δ parameter.

7 Concluding Remarks

In this article, we tackled the problem of scheduling tasks with a cumulative memory constraint on processors. The constraint on memory does not allow us to derive approximation algorithms due to the NP-completeness of finding a feasible schedule. Thus we transformed the constraint into an objective to optimize. The bi-objective problem has been studied: some impossible approximation ratios have been pointed out, and two algorithms dealing respectively with independent tasks and precedence constraints have been proposed.

Those results can help to solve the original problem. Regarding the problem with precedence constraints, it is easy to compute the Graham lower bound on the memory usage and thus to compute which parameter to use with *RLS*. This enables us to know which approximation ratio on the makespan can be obtained. We also know that using another value of the parameter can not lead to a better feasible solution as the algorithm uses a thresholding approach. However, this is not the case on independent tasks. A parameter which always leads to a feasible solution can also be computed. But then the solution can be tentatively improved by doing a binary search on the parameter. Finally, only few cases can not be handled by the algorithms we proposed, which are when it is difficult to fit the tasks due to the memory constraint. It seems difficult to guarantee performances in such cases.

The independent tasks case is, as usual, the core of

the problem. Thus, future works should focus on closing the gap between impossible approximation ratios and the known achievable ratios. The approximation ratio of the Restricted List Scheduling algorithm does not seem to be tight and List Scheduling based algorithms are often used in practice. Thus, the approximation ratios should be improved or a tight counter example should be presented. Some more realistic model extensions should be investigated such as conditional task graphs or non identical processors.

From a more general point of view, we show a problem in which transforming a constraint into an objective helps to deal with it. We also believe that this approach can help to deal with other constraints such as real-time constraints.

References

- [1] E. Angel, E. Bampis, and A. V. Fishkin. A note on scheduling to meet two min-sum objectives. *Operation Research Letters*, 35(1):69–73, 2007.
- [2] E. Angel, E. Bampis, and A. Kononov. A FPTAS for approximating the Pareto curve of the unrelated parallel machines scheduling problem with costs. In LNCS, editor, *Algorithms - ESA 2001*, volume 2161, pages 194–205, 2001.
- [3] F. Baille, E. Bampis, and C. Laforest. A note on bicriteria schedules with optimal approximations ratios. *Parallel processing letters*, 14(2):315–323, 2004.
- [4] S. Campana, D. Barberis, F. Brochu, A. De Salvo, F. Donno, L. Goossens, S. Gonzalez de la Hoz, T. Lari, D. Liko, J. Lozano, G. Negri, L. Perini, G. Poulard, S. Resconi, D. Rebatto, and L. Vaccarossa. Analysis of the atlas rome production experience on the lhc computing grid. In *E-SCIENCE '05: Proceedings of the First International Conference on e-Science and Grid Computing*, pages 82–89, Washington, DC, USA, 2005. IEEE Computer Society.
- [5] P. Choudhury, R. Kumar, and P. P. Chakrabarti. Hybrid scheduling of dynamic task graphs with selective duplication for multiprocessors under memory and time constraints. *IEEE Transactions on Parallel and Distributed Systems*, (preprint), 2007.
- [6] J. J. Dongarra, E. Jeannot, E. Saule, and Z. Shi. Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In *SPAA '07: Proceedings of the nineteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 280–288, 2007.
- [7] M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman, San Francisco, 1979.
- [8] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, March 1969.
- [9] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Practical and theoretical results. *Journal of ACM*, 34:144–162, 1987.
- [10] C. Laforest. A tricriteria approximation algorithm for steiner tree in graphs. Technical Report 69-2001, LaMI, 2001.

- [11] C. H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In FOCS, editor, *41st Annual Symposium on Foundations of Computer Science*, pages 86–92, 2000.
- [12] C. Wein and J. Stein. On the existence of schedules that are near-optimal for both makespan and total weighted completion time. *Operational research letters*, 21(3):115–122, Oct. 1997.