Scheduling of independent dedicated multiprocessor tasks *

Evripidis Bampis † Massimiliano Caramia ‡ Jiří Fiala § Aleksei V. Fishkin ¶ Antonio Iovanella ||

Abstract

We study the off and on-line versions of the well known problem of scheduling a set of n independent multiprocessor tasks with prespecified processor allocations on a set of identical processors in order to minimize the makespan. Recently, in [13], it has been proven that in the case when all tasks have unit processing time the problem cannot be approximated within a factor of $m^{\frac{1}{2}-\varepsilon}$, neither for some $\varepsilon > 0$, unless P= NP; nor for any $\varepsilon > 0$, unless NP=ZPP. For this special case we give a simple algorithm based on the classical first-fit technique. We analyze the algorithm for both tasks *arrive over time* and tasks *arrive over list* on-line scheduling versions, and show that its competitive ratio is bounded by $2\sqrt{m}$ and $2\sqrt{m}+1$, respectively. Here we also use some preliminary results on (vertex) coloring of k-tuple graphs. For the case of arbitrary processing times, we show that any algorithm which uses the first-fit technique cannot be better than m competitive. Then, by using our split-round technique, we give a $3\sqrt{m}$-approximation algorithm for the off-line version of the problem. Finally, by using some ideas from [21], we adapt the algorithm to the on-line case, in the paradigm of *tasks arriving over time* in which the existence of a task is unknown until its release date, and show that its competitive ratio is bounded by $6\sqrt{m}$. Due to the conducted experimental results, which also analyzed here, we conclude that our algorithms can perform well in practice.

1 Introduction

In the traditional theory of scheduling, each task is processed by only one processor at a time. However, new theoretical approaches have emerged to model scheduling on parallel architectures. One of these is scheduling multiprocessor tasks (see [9] for some practical justifications and a survey on some main results).

In this paper we address the *dedicated* variant of the multiprocessor tasks scheduling problem. We are given a set of n tasks $T = \{1, \dots, n\}$ and a set of m processors $M = \{1, 2, \dots, m\}$. Each task $j \in T$ has a processing time p_j, a release date r_j and a prespecified set of processors

*Supported by the bilateral French-German project PROCOPE.

†LaMI, CNRS-UMR 8042, Univerité d'Evry, Boulevard Mitterrand, 91025 Evry Cedex, France. bampis@lami.univ-evry.fr. Research supported by EU APPOL II project IST-2001-32007.

‡Istituto per le Applicazioni del Calcolo "M. Picone" - CNR Viale del Policlinico, 137 - 00161 Roma - Italy. caramia@iac.rm.cnr.it. Research supported by INRC project CNRG007FC1.

§DIMATIA and ITI, Charles University, Malostranské nám. 2/25, 118 00, Prague, Czech Republic. fiala@kam.mff.cuni.cz. Research supported by EU ARACNE project HPRN-CT-1999-00112, GAČR project 201/99/0242, and by MECR project LN00A056.

¶Institut für Informatik und Praktische Mathematik, Universität zu Kiel, Olshausen Str. 40, D-24118 Kiel, Germany. avf@informatik.uni-kiel.de. Supported by Graduiertenkolleg 357 "Effiziente Algorithmen und Mehrskalenmethoden".

||Dipartimento di Informatica, Sistemi e Produzione, University of Rome "Tor Vergata", Via del Politecnico, 1 - 00133 Rome, Italy. iovanella@disp.uniroma2.it. Research supported by INRC project CNRG007FC1.

$\mathit{fix}_j \subseteq M$. Preemptions are not allowed. Each processor can work on at most one task at a time, each task must be processed simultaneously by all processors of fix_j. The objective is to minimize the *makespan* $C_{\max} = \max_j C_j$, where C_j denotes the completion time of task j.

We call fix_j the *type* and $|\mathit{fix}_j|$ the *size* of job $j \in T$, and use $\Delta_{\max}$ to denote $\max_j |\mathit{fix}_j|$ and $r_{\max}$ to denote $\max_j r_j$. To refer to the several variants of the above scheduling problem, we use the standard notation scheme by Graham at al. [16]. For example, $P|\,\mathit{fix}_j, r_j|C_{\max}$ denotes the above problem itself in the *off-line* case[1]; $P|$ *on-line*, $\mathit{fix}_j, p = 1|C_{\max}$ denotes the on-line variant where all $p_j = 1$ and the tasks *arrive over list* [2]; and $P|$ *on-line*, $\mathit{fix}_j, r_j|C_{\max}$ denotes the on-line variant where all p_j are arbitrary and the tasks *arrive over time*[3].

Multicasting in WDM networks. Optical networks employing wavelength division multiplexing (WDM) are now a viable technology for implementing a next-generation network infrastructure that will support a diverse set of existing, emerging, and future applications [14]. WDM technology initially was deployed in point-to-point links in *wide* area or *metropolitan* area distances [24]. However, the work on WDM *local area* networks[4] has also been currently under way, see e.g. [17, 19].

The main future of broadcast WDM local area networks is the so-called *one-to-many transmission* or *multicasting* ability [1]. That is, a transmission by a node in such a network on a given channel (wavelength) is received by *all* nodes listening on that channel at that point in time (see [22] for a detailed survey on WDM networks).

Since the number of channels may be less than the number of nodes and two or more nodes may want to send data packets to the same destination node, coordination among nodes that wish to communicate with each other is required. Many *access protocols* for coordinating *multicast* data transmissions have been proposed in the literature (see [23] for a survey). The approaches used range from sending a different copy of each data packet to each one of the corresponding destinations (*unicast service*), to transmitting a single copy of the data packet to all the destinations at once (*multicast service with no fanout splitting*).

Due to the relation to the later approach, there has been recently a renewed interest to the model of scheduling dedicated tasks in the *on-line* context. In this framework, *nodes* correspond to *processors* and *multi-destination* data packets to *dedicated tasks*. The goal, minimizing the overall *transmission* time, i.e. the time needed to send all data packets out, corresponds to the schedule makespan of dedicated tasks.

Known results and our contribution. Variants of the multiprocessor tasks scheduling problem have been studied, but the previous research has mainly focused on the off-line case. Hoogeveen et al. [18] showed that already the three-processor problem $P3\,|\,\mathit{fix}_j|\,C_{\max}$ is strongly NP-hard, and proved that even if all tasks have unit processing times, there exists no polynomial approximation algorithm for $P\,|\,\mathit{fix}_j, p_j = 1|\,C_{\max}$ with performance ratio smaller than $\frac{4}{3}$, unless P=NP. However, in the same work it was shown that if the number of processors m is fixed, i.e. $Pm|\,\mathit{fix}_j, p_j = 1|\,C_{\max}$, then the problem is solvable in polynomial time. Later, Amoura et al. [2] proposed a polynomial time approximation scheme (PTAS) for problem $Pm|\,\mathit{fix}_j|\,C_{\max}$, and Bampis & Kononov [3] extended this result to a PTAS for $Pm|\,\mathit{fix}_j, r_j|C_{\max}$. A $\frac{7}{6}$-approximation

[1] In *off-line* scheduling, the scheduler has full information of the problem instance. In contrast, in *on-line* scheduling, information about the problem instance is made available during the course of scheduling.

[2] The tasks of T are ordered in some list (sequence) and presented one by one to the scheduler according to this list. The existence of a job is not known until all its predecessors in the list have already been scheduled.

[3] The tasks j of T arrive at their release dates r_j. The tasks can be started at a time bigger than or equal to their release dates, and at any point of time, the scheduler only has knowledge of the released tasks.

[4] These networks are known as *single-hop* WDM networks [20].

algorithm for $P3|\,\mathit{fix}_j|C_{\max}$, which is due to Goemans [15], still remains the best low time complexity approximation algorithm. Recently, Fishkin et al. [13] extended the negative results presented in [18]. It has been proven that $P\,|\,\mathit{fix}_j, p_j = 1|\,C_{\max}$ cannot be approximated within a factor of $m^{1/2-\varepsilon}$, neither for some $\varepsilon > 0$, unless P=NP; nor for any $\varepsilon > 0$, unless NP=ZPP.

In the on-line context, there are several results known for the *parallel* variant of the multiprocessor tasks scheduling problem, see e.g. [11, 10, 5]. Up to our best knowledge, no on-line algorithms with guaranteed competitive ratio are known for the dedicated variant of the multiprocessor model, considered in this paper. Although some algorithms have been recently proposed in the literature, their performances are not evaluated analytically, but by using simulations [7, 6].

In this paper we present several results, important from both theoretical and practical point of view. First, we deal with the problems where tasks have unit processing times. Using the so-called *first-fit* technique, we give an on-line algorithm for $P|\,\textit{on-line}, fix_j, p_j = 1|C_{\max}$, which is k-competitive if the maximum task size $\Delta_{\max}$ is bounded by some constant k. It is interesting to point out that our analysis is based on a transformation of our scheduling problem to the classical (vertex) coloring problem of a particular class of graphs, the intersection graphs of k-tuples. We propose simple extensions of this algorithm to a $2\sqrt{m}$-competitive algorithm for $P\,|\,\textit{on-line}, \mathit{fix}_j, p_j = 1|\,C_{\max}$ and $(2\sqrt{m}+1)$-competitive algorithm for $P\,|\,\textit{on-line}, \mathit{fix}_j, p_j = 1, r_j|\,C_{\max}$. Clearly, the $2\sqrt{m}$-competitive algorithm and the first-fit algorithm are complementary. For instance, if $k = 2$ then the first-fit algorithm is better. From another side, whenever $k = m$ the first algorithm outperforms the second one.

Next, we switch to the general problem with arbitrary processing times. We show that any on-line algorithm which schedules tasks arriving over-list and leaves no unnecessary idles cannot be better than m-competitive. In some sense, this leaves no hope for contracting any good on-line algorithm based on the first-fit technique. However, using our *split-round* technique, we obtain an off-line $3\sqrt{m}$-approximation algorithm for $P\,|\,\mathit{fix}_j|\,C_{\max}$. Then, by using the so-called *active-passive-bins* scheduling technique by Shmoys, Wein & Williamson [21], we give a $6\sqrt{m}$-competitive algorithm for $P\,|\,\textit{on-line}, \mathit{fix}_j, r_j|\,C_{\max}$ in which the existence of a task is unknown until its release date.

Finally, we present some experimental results based on random generated instances with $m = 20$ processors; $n = 50$, 100 and 200 tasks; $\Delta_{\max} = 4$, 6, 8, 12 maximum size; $r_{\max} = 6$, 18 and 30 maximum release date. Analyzing the results we observe that our algorithms work much better than it can be expected. For all cases, the output makespan exceeds the lower bound, which is also heuristically obtained, by at most a factor of 3.5 (see Figure 4 and 5), though $\sqrt{20} \approx 4.47$. Furthermore, we find that all the algorithms have low running time, that is a very important factor for real world applications.

Last notes. We say that an algorithm A is a ρ-*approximation* algorithm for all problem instances it outputs a schedule with the makespan at most $\rho \cdot OPT$, where OPT is the makespan of the optimal schedule. If A is an on-line ρ-approximation algorithm, then we say that it is a ρ-*competitive* algorithm or A is ρ-*competitive.* (We may also that ρ is the approximation and/or competitive ratio of A.) Here, the *competitiveness* of an on-line algorithm is evaluated with respect to an off-line optimal algorithm, and hence it indicates the loss associated with not having complete information of the problem instance. Both approximation algorithms and on-line algorithms are assumed to run in polynomial time.

The paper is organized as follows. In the next section we give some preliminary results. In Section 3.1, we deal with the unit processing times case, and in Section 3.2 with the general case of arbitrary processing times. In the last Section we present the experimental results. The

Appendix includes all the statistics.

2 Preliminaries

In this section we discuss some general techniques and lemmas that apply throughout our paper. We transform our scheduling problem to the classical (vertex) coloring problem. Then, we use some standard methods from the graph theory to obtain related results. The properties we prove in this section pertain to the case of unit processing times, but the ideas will be used in modified form for arbitrary processing times as well.

2.1 Coloring of the conflict graph

Given an undirected graph $G = (V, E)$, a *clique* of G is a set of mutually adjacent vertices. A maximum clique is, naturally, a clique whose number of vertices is at least as large as that for any other clique in the graph. If the vertices have *weights* then a maximum *weighted clique* is a clique with the largest possible sum of vertex weights.

A *(vertex) coloring* of $G = (V, E)$ is an assignment of a *color* to each vertex in V. It is required that the colors on the pair of vertices incident to any edge be different. A minimum coloring of a graph is a coloring that uses as few different colors as possible.

Clique and coloring problems are very closely related. It is straightforward to see that the size of the maximum clique of G is a lower bound on the minimum number of labels needed to color a graph.

Recall our scheduling problem. We are given a task set $T = \{1, \ldots, n\}$, processor set $M = \{1, 2, \ldots, m\}$, and for each task $j \in T$ a processing time p_j, release date r_j, and type $\mathit{fix}_j \subseteq M$. The goal is to find a schedule that minimizes the makespan $C_{\max} = \max_j C_j$.

Consider the case when all $r_j = 0$ and $p_j = 1$. Then, our scheduling problem can be modeled as clique and coloring problems. It involves forming the so-called *conflict* graph G_T with vertices representing the tasks of T. An edge connects two *incompatible* tasks j and i iff $\mathit{fix}_j \cup \mathit{fix}_j \neq \emptyset$. The maximum clique problem is then to find as large a set of pairwise incompatible task as possible. The minimum coloring problem is to assign a color to each task so that every incompatible pair is assigned different colors. Accordingly, having a coloring of the conflict graph G_T one can find a schedule for the tasks of T with the makespan at most the number of colors, and vice versa, having a schedule one can find a coloring with the number of colors at most the schedule makespan. Thus, the maximum clique number of G_T, denoted $\omega(G_T)$, is a lower bound on the schedule makespan of the tasks of T.

Unfortunately, there is no such a simple transformation for the case when all $r_j = 0$, but all p_j are arbitrary. However, we can add a weight p_j to each vertex j of the conflict graph G_T and deal with the maximum weighted clique problem. In this framework, the sum of vertex weights in a maximum weighted clique of G_T, denoted $\omega_p(G_T)$, is a lower bound on the schedule makespan of the tasks of T. (We use this fact only in the last section.)

Consider the case when the maximum task size $\Delta_{\max}$ is bounded by some constant, say k. Then, the conflict graph G_T gets a very specific structure, becoming a *k-tuple graph.* More formally it can be defined as follows. Let X be a finite set. A *k-tuple* of X is a set having k (or less) elements of X. A graph $G = (V, E)$ is a *k-tuple graph* if there exists a set X such that each node of V corresponds to a k-tuple of X and there is an edge in E between any two vertices iff the intersection of the corresponding k-tuples is not empty. (Notice that several vertices can correspond to a single k-tuple.)

Different variants of the graph (vertex) coloring problem have been studied. It is widely known that in the case of general graphs the coloring problem is strongly NP-hard, and there is no ρ-approximation algorithm with $\rho = n^{1/7-\varepsilon}$ unless P=NP, where n is the number of vertices of the graph [4]. However, for restricted graph classes, there can exist either polynomial time exact or approximation algorithms, as well as on-line coloring algorithms (see [12] for a survey on on-line coloring).

For a graph $G = (V, E)$, an on-line coloring algorithm colors the vertices in V one vertex at a time in the externally determined order $v_1 \prec \cdots \prec v_n$, and at the time a color is irrevocably assigned to v_t, the algorithm sees the vertices adjacent to v_t only among $v_1, \ldots, v_{t-1}$. In the following we use a very simple variant of an on-line coloring algorithm:

First Fit Coloring (FFC):
Select vertices v from V in an arbitrary order while coloring with an initial sequence of colors $1, 2, \ldots$. Assign the vertex v the least color that has not already been assigned to any vertex adjacent to v.

Lemma 2.1 *For a k-tuple graph G, the number of labels used by the algorithm* FFC *is at most $k \cdot w(G)$, where $w(G)$ is the maximum clique number of G.*

Proof Sketch. Let X be a finite set and $G = (V, E)$ be a k-tuple graph with the corresponding k-tuples of X. Let $\{a_1, \ldots, a_k\}$ be the k-tuple of X corresponding to a vertex v. Observe that the neighborhood $N(v) = \{u \in V \mid (u, v) \in E\}$ can be split into at most k sets forming disjoint cliques. Each set of vertices in $N(v)$ whose corresponding k-tuples of X contain a common element, is a clique. Thus, we select elements from $\{a_1, \ldots, a_k\}$ one by one, extracting the corresponding cliques from $N(v)$. These cliques are disjoint and cover $N(v)$. Hence, $|N(v)| \leq k \cdot (\omega(G) - 1)$ and FFC cannot use more than $k \cdot \omega(G)$ colors on G. □

3 Scheduling of dedicated tasks

In this section we consider the on-line version of the dedicated scheduling problem. In the first part we start from the simplest case of unit processing times. In the last part we consider the general case with arbitrary processing times.

3.1 Unit processing times

Consider the case when all $p_j = 1$. Then, we can give the following simple algorithm:

First Fit Scheduling (FFS):
Schedule the tasks of T by starting the task j at the earliest interval $[t, t+1)$ in which the processors of fix_j are not busy.

By using the result of Lemma 2.1 we have:

Lemma 3.1 *If the maximum task size $\Delta_{\max}$ is bounded by some constant k, then the algorithm* FFS *is k-competitive for $P \mid$ on-line, fix_j, $p_j = 1 \mid C_{\max}$.*

Proof Sketch. First consider the case when the tasks of T arrive over-list. Let OPT be the optimum makespan for $P \mid fix_j, p_j = 1 \mid C_{\max}$. Let G_T be the corresponding conflict graph of task set T. One can see that FFC working on G_T cannot outperform FFS working on T, in the worst case analysis. (Each interval $[t, t+1)$ corresponds to color t, the makespan corresponds to the number of colors, and vice versa.) Thus, since $w(G_T)$ is a lower bound on OPT, by Lemma 2.1 the makespan of the output schedule by FFS is at most $k \cdot OPT$. □

The above lemma does not guarantee a good performance of FFS on instances with $\Delta_{\max} = m$. However, we can modify the algorithm as follows:

FIRST FIT SCHEDULING+ (FFS+):
Schedule the tasks of T by starting the task j at the earliest interval $[t, t+1)$, $t \geq r_j$ in which the processors of fix_j are not busy and there is no task in $[t, t+1)$

- of size greater than $\sqrt{m}$ if $|fix_j| \leq \sqrt{m}$, and
- of size at most $\sqrt{m}$ if $|fix_j| > \sqrt{m}$.

Less formally, the output schedule can be split into two parts — the first part includes the intervals during which the tasks have size at most $\sqrt{m}$, call these intervals "red" colored, and the second part includes the intervals during which the processed tasks have a size greater than $\sqrt{m}$, call these intervals "blue" colored. Accordingly, one can think in terms of assigning the tasks, depending on their sizes, to blue or red intervals.

Lemma 3.2 *The algorithm* FFS+ *is* $2\sqrt{m}$*-competitive for* $P\,|\,$on-line, fix_j, $p_j = 1|\,C_{\max}$ *and is* $(2\sqrt{m}+1)$*-competitive for* $P\,|\,$on-line, $fix_j, p_j = 1, r_j|\,C_{\max}$.

Proof Sketch. First consider the case where the tasks of T arrive over-list. Let $T^+ := \{j \in T : |fix_j| > \sqrt{m}\}$ and $T^- := \{j \in T : |fix_j| \leq \sqrt{m}\}$. Also, let OPT denote the optimal makespan for $P\,|\,fix_j, p_j = 1|\,C_{\max}$. Then, by Lemma 3.1 the "red" part of the output schedule corresponding to T^- has a makespan at most $\sqrt{m}OPT$. Regarding the "blue" part of the output schedule corresponding to T^+ observe the following: At least one task appears in each "blue" interval, and since for each $j \in T^+ : |fix_j| > \sqrt{m}$, one may additionally place at most $(m - \sqrt{m})/\sqrt{m} = \sqrt{m} - 1$ tasks in each "blue" interval. Hence, the blue part has a makespan at most $(\sqrt{m}-1)OPT + OPT$. Combining the two bounds we get that the output makespan is at most $2\sqrt{m}\,OPT$.

Now consider the case where the tasks of T arrive over-time. Let OPT' denote the optimal makespan for $P\,|\,fix_j, p_j = 1, r_j|\,C_{\max}$. Then, it is clear that the release date $r_j \leq OPT'$ for each task $j \in T$. Furthermore, by the above observation we can estimate the completion time of each task $j \in T$ as follows:

$$C_j \leq r_j + 2\sqrt{m}\,OPT \leq OPT' + 2\sqrt{m}\,OPT' \leq (2\sqrt{m}+1)OPT'.$$

□

3.2 Arbitrary processing times

Now we consider the general case when all p_j are arbitrary. First we analyze the first-fit technique, and then we go to our split-round technique, presenting the main results of this paper.

3.2.1 First-fit technique

Consider m tasks $T_1, T_2 \ldots, T_m$ with processing times $p_k = 1 + \frac{k}{m}\varepsilon$ and types $fix_k = \{k\}$ $(k = 1, \ldots, m)$, and, in addition, $m-1$ tasks $T_{m+1}, T_{m+2}, \ldots, T_{2m-1}$ with the same type $fix_k = \{1, \ldots, m\}$ and processing time $p_k = \frac{\varepsilon}{m-1}$ $(k = m+1, \ldots, 2m-1)$.

If all these $2m-1$ tasks arrive in the order $1, (m+1), 2, (m+2), \ldots, i, (m+i), \ldots, (m-1), (2m-1), m$, then any deterministic on-line algorithm which leaves no unnecessary idles produces a schedule of makespan greater than $m + 2\varepsilon$, while an optimal schedule has a makespan

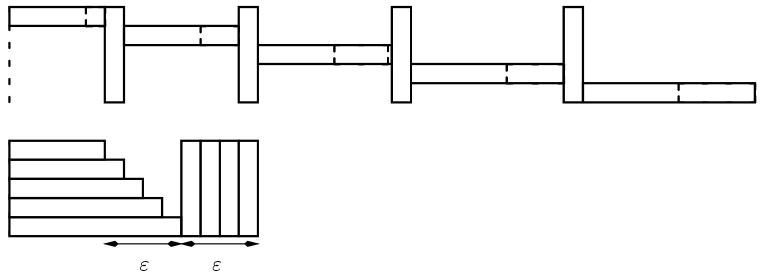

Figure 1: The output and optimal schedules

equal to $1+2\varepsilon$. (See Figure 1). Thus, m is a lower bound on the competitive ratio of any on-line algorithm which uses the first-fit technique.

We define $P(T) = \sum_{j \in T} p_j$ to be the total processing time of tasks of T, and $L(T) = \frac{1}{m}\sum_{j \in T} p_j |\mathit{fix}_j|$ to be the average load of T. Then, it holds that

$$P(T) \le m \cdot L(T) \qquad \text{and} \qquad L(T) \le OPT \le \max_{j \in T} r_j + P(T),$$

where OPT is the minimum makespan for T.

Consider the following algorithm:

General First Fit Scheduling (GFFS): Schedule the tasks of T by starting the task j at the earliest time $t \ge r_j$ such that the processors of fix_j are not busy in interval $[t, t+p_j)$.

Sure, the above algorithm uses the first-fit technique and cannot be better than m-competitive. However, it cannot be much worse than that as well.

Lemma 3.3 *The algorithm* GFFS *is m-competitive for $P\,|\,\text{on-line}, \mathit{fix}_j\,|\,C_{\max}$ and $(m+1)$-competitive for $P\,|\,\text{on-line}, \mathit{fix}_j, r_j\,|\,C_{\max}$.*

Proof Sketch. First consider the case where the tasks of T arrive over-list. Let OPT be the minimal makespan for $P\,|\,\mathit{fix}_j\,|\,C_{\max}$. Let C denote the makespan of the schedule found by GFFS working on T. Since for each task $j \in T$ the release date $r_j = 0$ and the algorithm GFFS uses the first-fit technique, $P(T)$ is an upper bound on C. Hence, it holds that

$$C \le P(T) \le m \cdot L(T) \le m \cdot OPT,$$

i.e. the output schedule's makespan is at most $m \cdot OPT$.

We complete by using the arguments in the proof of Lemma 3.2. □

3.2.2 Split-round technique

Let us consider the following off-line algorithm for $P\,|\,\mathit{fix}_j\,|\,C_{\max}$.

Split Round Scheduling (SRS):
Form $T^+ := \{j \in T : |\mathit{fix}_j| > \sqrt{m}\}$ and $T^- := \{j \in T : |\mathit{fix}_j| \leq \sqrt{m}\}$.

1. Apply GFFS to tasks $j \in T^+$ ordered in an arbitrary way.
2. For each task $j \in T^-$ round p_j to the smallest power of two, say $[p_j]$[a].
3. Apply GFFS to tasks $j \in T^-$ ordered by non-increasing $[p_j]$'s.

[a] Here $[p_j] = 2^a \geq p_j$ and $2^{a-1} < p_j$.

We can state the following.

Theorem 3.4 *The algorithm* SRS *is a* $3\sqrt{m}$*-approximation algorithm for* $P\,|\,\mathit{fix}_j|\,C_{\max}$.

Proof: Let OPT be the minimum makespan for $P\,|\,\mathit{fix}_j|\,C_{\max}$. Let C^+ be the makespan of the schedule found by GFFS while working on the tasks of T^+ ordered arbitrary (Step 1), and C^- be the makespan of the schedule found by GFFS while working on the rounded tasks $j \in T^-$ ordered by non-increasing $[p_j]$'s (Steps 2 and 3).

Since for each task $j \in T^+$ the size $|\mathit{fix}_j| > \sqrt{m}$, it holds that

$$L(T^+) = \frac{1}{m} \sum_{j \in T^+} p_j |\mathit{fix}_j| \geq \frac{\sqrt{m}}{m} \sum_{j \in T^+} p_j = \frac{P(T^+)}{\sqrt{m}}.$$

Thus,

$$C^+ \leq P(T^+) \leq \sqrt{m}\, L(T^+) \leq \sqrt{m}\, OPT.$$

Regarding the tasks of T^- observe the following. First, by applying the rounding procedure we lose at most a factor of 2 in the objective function. Hence, the new minimal makespan, say OPT', is at most $2\,OPT$. Second, the processing time of the tasks of T^- are powers of 2, and GFFS schedules tasks $j \in T^-$ in non-increasing order of $[p_j]$. Hence, a task, say k, is "blocked" by an already scheduled task, say s, if and only if $s : \mathit{fix}_k \cap \mathit{fix}_s \neq \emptyset$. (See Figure 2 for an illustration. Notice that if processing times are not powers of 2 we cannot guarantee this property, that is a base element of our analysis.)

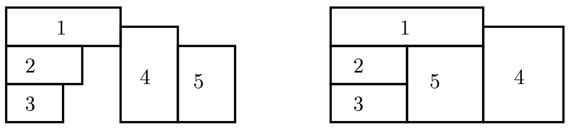

Figure 2: Schedules for non-rounded and rounded tasks

Consider the schedule found by GFFS while working on the rounded tasks $j \in T^-$ ordered by non-increasing $[p_j]$'s. Let k be a task such that $C^- = C_k$. Accordingly, $S_k = C_k - p_k = C^- - p_k$ is the starting time of task k. Then, by the above observation and by the fact that GFFS uses the first-fit technique, we can find a sequence $s_1, s_2, \ldots, s_q$ of tasks in T^- such that at each moment $t \in [0, S_k)$ for exactly one task $s(t) \in \{s_1, s_2, \ldots, s_q\}$ it holds that $\mathit{fix}_{s(t)} \cap \mathit{fix}_k \neq \emptyset$. (See Figure 3). Hence,

$$S_k = \sum_{i=1}^{q} p_{s_i} \qquad \text{and} \qquad C^- = p_k + \sum_{i=1}^{q} p_{s_i}.$$

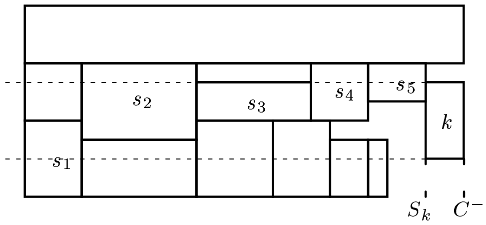

Figure 3: A task k and a sequence s_1, s_2, s_3, s_4, s_5

Since the size $|fix_k| \leq \sqrt{m}$ and each task s_i, $i = 1, \ldots, q$ requires at least one of the processors in τ_k, the makespan of any schedule for tasks $s_1, s_2, \ldots, s_q$ and k is at least:

$$L' = \frac{1}{\sqrt{m}} \left(p_k + \sum_{i=1}^{q} p_{s_i} \right) = \frac{C^-}{\sqrt{m}}.$$

Furthermore, since these tasks belong to $T^- \subseteq T$ it also holds that $L' \leq OPT' \leq 2OPT$. Hence, we get

$$C^+ + C^- \leq \sqrt{m}\, OPT + 2\sqrt{m}\, OPT \leq 3\sqrt{m}\, OPT,$$

i.e. the makespan of the output schedule by SRS is at most $3\sqrt{m}\, OPT$. □

We are ready to present the following main algorithm:

Split-Round Scheduling+ (SRS+):

1. At each moment we consider two bins, one of which is *active* and the other one is *passive*.
2. At the first release date, collect the tasks into the active bin.
3. Schedule the tasks of the active bin by SRS while collecting the tasks being released into the passive bin.
4. At each moment when SRS has no task to schedule, exchange the activities of bins.

Here we call the following result by Shmoys, Wein & Williamson in [21]: "Let A be a polynomial-time scheduling algorithm that works in an environment in which each job to scheduled is available at time 0 and which always produces a schedule of length at most ρC^*. For the analogous environment in which the existence of a job is unknown until its release date, there exists another polynomial-time algorithm A' that works in this more general setting and produces a schedule of length at most $2\rho C^*$". In fact, SRS is similar to the algorithm constructed, and we can give the following result

Theorem 3.5 SRS+ *is $6\sqrt{m}$-competitive for $P \,|\, \text{on-line}, fix_j, r_j \,|\, C_{\max}$ in which the existence of a task is unknown until its release date.*

4 Experimental analysis

The scope of this section focuses on experimental evaluating the algorithms presented above. In fact, even though an algorithm has a "bad" approximation or competitive ratio, in practice, where one meets not that many anomalistic instances, the behavior of the algorithm is better. This is exactly that happens to our algorithms.

We generated several sets of random instances, see in the next paragraph, to be used as inputs for FFS(+), GFFS and SRS(+). In order to evaluate the quality of the output schedule on some task set T, we compare the output makespan to the maximum (weighted) clique of the conflict graph G_T. (See Section 2.) In particular, we used the unweighted clique, namely ω, for the case when all $p_j = 1$ (and thus for FFS and FFS+), and we used the weighted clique, namely ω_p, for the case when all p_j are arbitrary (and thus for GFFS, SRS and SRS+). Due to the NP-hardness of the maximum clique problem, we used the heuristics by Carraghan and Pardalos [8] stopping the iterations at $10,000$.

Presentation

The instances used in the experiments have a number of tasks n =50, 100, and 200 and a number of processors $m = 20$. For each value of n we generated ten instances for different choices of two parameters, namely, the maximum task size $\Delta_{\max}$ and the maximum task release date r_{max}. The algorithms have been implemented in the C language and tested on a PC Pentium 833 MHz. In particular, for $\Delta_{\max}$ we use the values 4, 6, 8, 12, and for r_{max} the values 6, 18, 30. Results are reported in Tables 1–3 listed in the Appendix. The headings contain r_{max} and $\Delta_{\max}$ and the names of our algorithms, i.e. FFS(+), GFFS, and SRS(+).

Analysis

FFS. The algorithm FFS (together with FFS+) is the only algorithm implemented with unitary task duration. This clearly results in lower completion times than those achieved by other algorithms working with arbitrary duration. What can be easily inferred by Table 2 is that the upper bound-lower bound gap computed experimentally is smaller than the theoretical one presented in the previous section. To be more specific, one can observe that for many cases the makespan of the output schedules by FFS is at most twice the lower bound. The experimental ratio between the results of FFS and ω is depicted on Figure 4 in the Appendix.

FFS+. As FFS, the algorithm FFS+ is implemented with unitary weights. Results achieved by FFS+ are identical to those obtained by FFS (see Table 1 in the Appendix). This phenomenon can be explained by the fact that the theoretical advantage to schedule tasks according to the number of processors required (which gives a substantial difference in terms of competitive-ratio), is not so highlighted by the experimental behavior of the algorithms. For FFS+ the maximum clique values are the same as for FFS. The experimental ratio between the results of FFS+ and ω is depicted on Figure 4 in the Appendix.

GFFS. The algorithm GFFS works on tasks with arbitrarily processing times. We have compared GFFS (as well as SRS and SRS+) to the lower bound offered by ω_p. The experimental ratio between the results of GFFS and ω_p is depicted on Figure 5 in the Appendix. The results show that in some cases we get very close to the optimum, while in many cases the ratio is much better than $\sqrt{20} = 4.47$ (here $m = 20$).

SRS. The algorithm SRS achieves better results than GFFS. This in part can be motivated by the best theoretical competitive ratio of SRS with respect to GFFS. However, there are some cases in which the two algorithms, i.e., SRS and GFFS, give the same results. (One can find a similar situation for SRS and SRS+, as well as for GFFS and SRS+.) This is due to the fact that GFFS and SRS are identical on instances where tasks have sizes $|fix_j|$ greater than $\sqrt{m}$. Since $m = 20$, this happens by $\Delta_{\max} = 6$, as Table 3 clearly shows that. The experimental ratio between the results of SRS and ω_p are depicted on Figure 5 in the Appendix.

SRS+. The algorithm SRS+ can be seen as a different engineering of SRS. Here SRS becomes a subroutine of SRS+. For lower values of $\Delta_{\max}$, SRS+ outputs schedules with higher value of the makespan than SRS, while for high values of r_{max}, the two algorithms have similar performance (see again the comments done for SRS). The experimental ratio between the results of SRS+ and ω_p are depicted on Figure 5 in the Appendix.

Computational time All the algorithms have a very low running time, especially FFS and FFS+ that in the worst case for $n = 200$ and $r_{max} = 2$ employ 53 and 57 milliseconds, respectively. For GFFS, SRS and SRS+ the computational times increase but remain limited below 2 seconds even for $n = 200$. In particular, for GFFS the worst case is 1.26 seconds (for $r_{max} = 6$ and $\Delta_{\max} = 4$), for SRS the worst case is 1.18 (for $r_{max} = 18$ and $\Delta_{\max} = 4$) and for SRS+ is 1.81 seconds (for $r_{max} = 6$ and $\Delta_{\max} = 4$).

5 Acknowledgment

We thank to Maxim Sviridenko for giving us useful comments and ideas.

References

[1] M. Ammar, G. Polyzos, and S. Tripathi (Eds.). Special issue on network support for multipoint communication. *IEEE Journal Selected Areas in Communications*, 15, 1997.

[2] A. K. Amoura, E. Bampis, C. Kenyon, and Y. Manoussakis. Scheduling independent multiprocessor tasks. In *Proceedings 5th European Symposium on Algorithms*, LNCS 1284, pages 1–12. Springer Verlag, 1997.

[3] E. Bampis and A. Kononov. On the approximability of scheduling multiprocessor tasks with time dependent processing and processor requirements. In *Proceedings 15th International Parallel and Distributed Processing Symposium*, San Francisco, 2001.

[4] M. Bellare, O. Goldreich, and M. Sudan. Free bits, PCPs, and non-approximability — towards tight results. *SIAM Journal on Computing*, 27:804–915, 1998.

[5] S. Bischof and E.W. Mayr. On-line scheduling of parallel jobs with runtime restrictions. In *Proceedings 9th Annual International Symposium on Algorithms and Computation*, LNCS 1533, pages 119–129. Springer Verlag, 1998.

[6] M. Caramia, P. Dell'Olmo, and A. Iovanella. Lower bound algorithms for multiprocessor task scheduling with ready times, 2001. Personal communications.

[7] M. Caramia, P. Dell'Olmo, and A. Iovanella. On-line algorithms for multiprocessor task scheduling with ready times, 2001. Personal communications.

[8] R. Carraghan and P. M. Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9:375–382, 1990.

[9] M. Drozdowski. Scheduling multiprocessor tasks - an overview. *European Journal on Operations Research*, pages 215–230, 1996.

[10] A. Feldmann, M.-Y. Kao, J. Sgall, and S.H. Teng. Optimal online scheduling of parallel tasks with dependencies. In *Proceedings 25th ACM Symposium on the Theory of Computing*, pages 642–651, 1993.

[11] A. Feldmann, J. Sgall, and S-H. Teng. Dynamic scheduling on parallel machines. *Theoretical Computer Science*, 130:49–72, 1994.

[12] A. Fiat and G. J. Woeginger, editors. *Online algorithms. The state of the art.* LNCS 1442. Springer Verlag, 1998.

[13] A. V. Fishkin, K. Jansen, and L. Porkolab. On minimizing average weighted completion time of multiprocessor tasks with release dates. In *Proceedings 28th International Colloquium on Automata, Languages and Programming*, LNCS 2076, pages 875–886, Crete, 2001. Springer Verlag.

[14] O. Gerstel, B. Li, A. McGuire, G. N. Rouskas, K. Sivalingam, and Z. Zhang (Eds.). Special issue on protocols and architectures for next generations optical wdm networks. *IEEE Journal Selected Areas in Communications*, 18, October 2000.

[15] M. X. Goemans. An approximation algorithm for scheduling on three dedicated processors. *Discrete Applied Mathematics*, 61:49–59, 1995.

[16] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic scheduling: A survey. *Annals of Discrete Mathematics*, pages 287–326, 1979.

[17] The NGI Helious project: Regional Testbed Optical Access Network For IP Multicast and Differentiated Services. http://projects.anr.mcnc.org/Helios/, 2000.

[18] J. A. Hoogeveen, S. L. Van de Velde, and B. Veltman. Complexity of scheduling multiprocessor tasks with prespecified processor allocations. *Discrete Applied Mathematics*, 55:259–272, 1994.

[19] M. Kuznetsov, N. Froberg, S. Henion, H. Rao, J. Korn, K. Rauschenbach, E. Modiano, and V. Chan. A next-generation optical regional access networks. *IEEE Communications Magazine*, 38:66 – 72, January 2000.

[20] B. Mukherjee. Wdm-based local lightwave networks Part I: Single-hop systems. *IEEE Network Magazine*, pages 12–27, 1992.

[21] D.B. Shmoys, J. Wein, and D.P. Williamson. Scheduling parallel machines on-line. *SIAM Journal on Computing*, 24:1313–1331, 1995.

[22] K. M. Sivalingam and S. Subramaniam, editors. *Optical WDM networks: Principles and practice.* Kluwer Academic Publishers, 2000.

[23] D. Thaker and G. N. Rouskas. Multi-destination communication in broadcast WDM networks: A survey. Technical Report 2000-08, North Caroline State University, 2000.

[24] R. E. Wagner, R. C. Alferness, A. A. M. Saleh, and M. S. Goodman. MONET: Multiwavelength Optical Networking. *Journal of Lightwave Technology*, 14:1349, June 1996.

6 Appendix

Table 1: Results on n = 50, 100, and 200 tasks (averages of 10 instances)

r_{max}	$\Delta_{\max}$	ω	FFS	FFS+	ω_p	GFFS	SRS	SRS+
n = 50								
6	2	9.1	11.6	11.6	49.3	61.3	52.4	88.0
18	2	9.1	19.8	19.8	49.6	65.7	57.0	89.8
30	2	9.1	31.0	31.0	47.6	69.9	59.7	94.9
6	4	16.3	20.1	20.1	80.1	120.6	95.5	165.7
18	4	16.3	24.4	24.4	83.2	129.1	101.7	174.0
30	4	16.3	32.7	32.7	89.9	133.9	109.3	181.4
6	6	24.7	30.1	30.1	136.5	196.9	196.9	196.9
18	6	24.7	33.6	33.6	128.3	194.7	194.7	194.7
30	6	24.7	38.2	38.2	140.1	201.8	201.8	201.8
6	8	45.4	47.1	47.1	237.1	252.9	252.9	252.9
18	8	45.4	48.0	48.0	256.0	270.0	270.0	270.0
30	8	45.4	50.2	50.2	256.7	268.4	268.4	268.4
6	12	50.0	51.0	51.0	259.7	260.1	260.1	260.1
18	12	50.0	51.2	51.2	269.0	270.9	270.9	270.9
30	12	50.0	51.1	51.1	275.1	277.3	277.3	277.3
n = 100								
6	2	15.9	17.8	17.8	83.4	106.7	88.2	149.0
18	2	15.9	22.1	22.1	88.8	107.3	92.1	149.0
30	2	15.9	31.5	31.5	89.2	108.2	91.6	153.5
6	4	27.9	34.9	34.9	157.4	224.8	171.9	300.1
18	4	27.9	36.3	36.3	146.6	219.5	171.3	298.5
30	4	27.9	40.7	40.7	137.8	221.1	172.8	296.0
6	6	36.0	55.8	55.8	197.3	350.1	350.1	350.1
18	6	36.0	57.1	57.1	202.2	354.6	354.6	354.6
30	6	36.0	58.2	58.2	208.4	337.4	337.4	337.4
6	8	85.6	88.2	88.2	473.1	515.5	515.5	515.5
18	8	85.6	90.4	90.4	481.2	519.2	519.2	519.2
30	8	85.6	89.1	89.1	494.4	521.9	521.9	521.9
6	12	100.0	101.0	101.0	551.9	552.1	552.1	552.1
18	12	100.0	101.0	101.0	555.2	555.6	555.6	555.6
30	12	100.0	101.0	101.0	572.2	572.5	572.5	572.5
n = 200								
6	2	28.2	30.6	30.6	144.0	175.8	144.2	246.8
18	2	28.2	32.1	32.1	142.8	172.6	142.6	239.4
30	2	28.2	36.0	36.0	140.6	178.4	145.3	242.4
6	4	46.7	60.3	60.3	251.6	396.4	307.9	532.3
18	4	46.7	62.5	62.5	240.6	394.8	308.4	521.3
30	4	46.7	64.2	64.2	275.5	396.6	309.9	521.2
6	6	50.4	103.0	103.0	283.9	637.8	637.8	637.8
18	6	50.4	103.0	103.0	279.0	631.5	631.5	631.5
30	6	50.4	104.3	104.3	279.5	624.5	624.5	624.5
6	8	155.6	158.9	158.9	827.6	957.1	957.1	957.1
18	8	155.6	157.9	157.9	846.0	976.0	976.0	976.0
30	8	155.6	162.5	162.5	892.4	973.9	973.9	973.9
6	12	200.0	201.0	201.0	1117.5	1117.5	1117.5	1117.5
18	12	200.0	201.0	201.0	1086.1	1086.3	1086.3	1086.3
30	12	200.0	201.0	201.0	1098.8	1128.2	1128.2	1128.2

Table 2: Gap between ω_p and FFS

r_{max}	$\Delta_{\max}$	$n = 50$	$n = 100$	$n = 200$
6	2	1.27	1.12	1.09
18	2	2.18	1.39	1.14
30	2	3.41	1.98	1.28
6	4	1.23	1.25	1.29
18	4	1.50	1.30	1.34
30	4	2.01	1.46	1.37
6	6	1.22	1.55	2.04
18	6	1.36	1.59	2.04
30	6	1.55	1.62	2.07
6	8	1.04	1.03	1.02
18	8	1.06	1.06	1.01
30	8	1.11	1.04	1.04
6	12	1.02	1.01	1.01
18	12	1.02	1.01	1.01
30	12	1.02	1.01	1.01

Table 3: Gap between ω_p and GFFS (r_a), SRS (r_b) and SRS+ (r_c)

		$n = 50$			$n = 100$			$n = 200$		
r_{max}	$\Delta_{\max}$	r_a	r_b	r_c	r_a	r_b	r_c	r_a	r_b	r_c
6	2	1.24	1.06	1.78	1.14	0.94	1.60	1.14	0.93	1.60
18	2	1.32	1.15	1.81	1.21	1.04	1.68	1.12	0.93	1.56
30	2	1.47	1.25	1.99	1.21	1.03	1.72	1.17	0.95	1.59
6	4	1.51	1.19	2.07	1.43	1.09	1.91	1.58	1.22	2.12
18	4	1.55	1.22	2.09	1.50	1.17	2.04	1.64	1.28	2.17
30	4	1.49	1.22	2.02	1.60	1.25	2.15	1.44	1.12	1.90
6	6	1.44	1.44	1.44	1.77	1.77	1.77	2.25	2.25	2.25
18	6	1.52	1.52	1.52	1.75	1.75	1.75	2.26	2.26	2.26
30	6	1.44	1.44	1.44	1.62	1.62	1.62	2.23	2.23	2.23
6	8	1.07	1.07	1.07	1.09	1.09	1.09	1.16	1.16	1.16
18	8	1.05	1.05	1.05	1.08	1.08	1.08	1.15	1.15	1.15
30	8	1.05	1.05	1.05	1.06	1.06	1.06	1.09	1.09	1.09
6	12	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
18	12	1.01	1.01	1.01	1.00	1.00	1.00	1.00	1.00	1.00
30	12	1.01	1.01	1.01	0.99	0.99	0.99	1.03	1.03	1.03

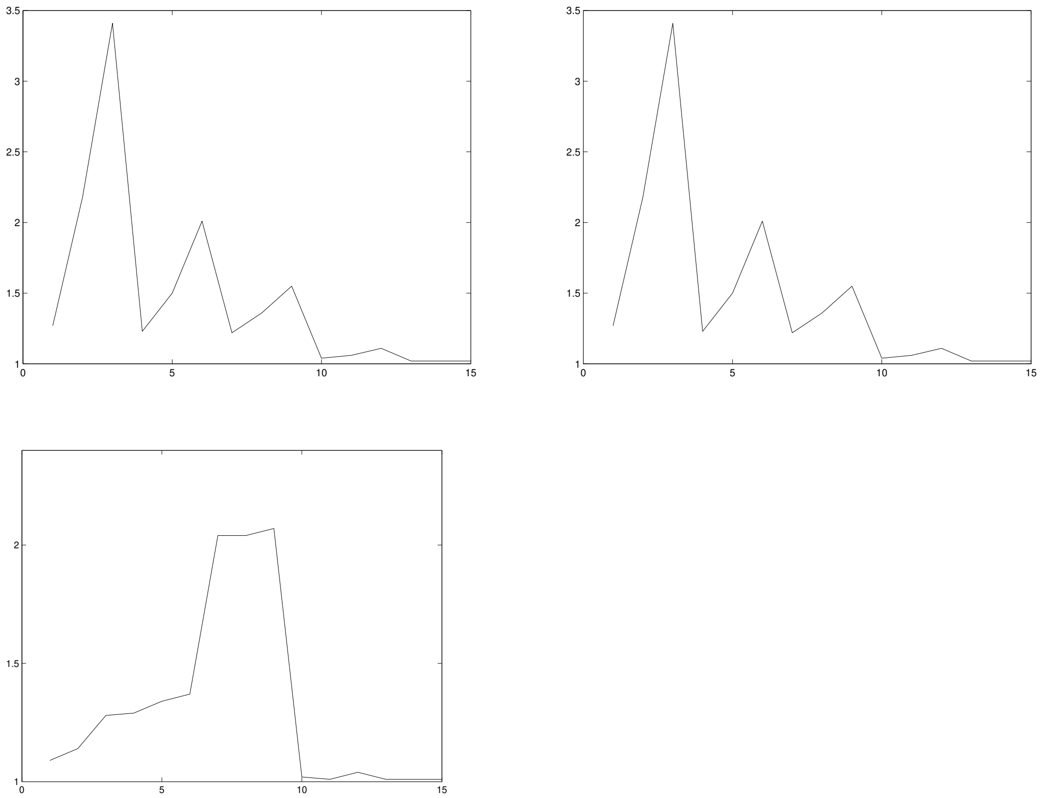

Figure 4: Percentage gaps of FFS for 50, 100 and 200 tasks

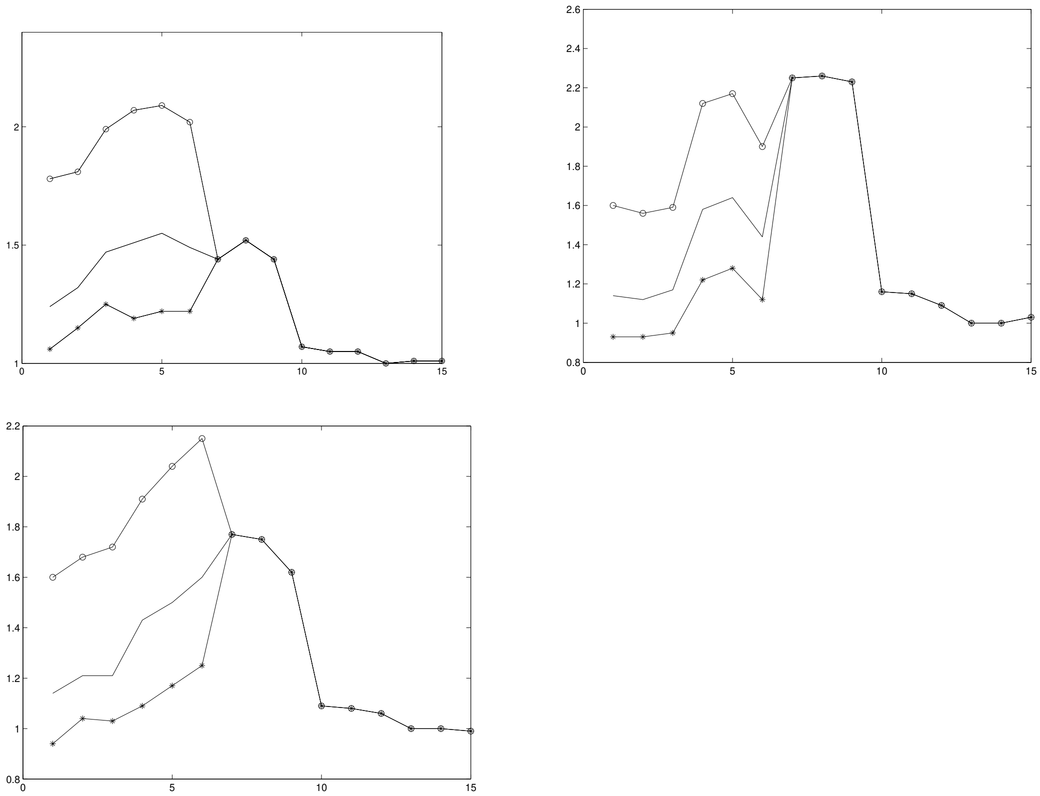

Figure 5: Percentage gaps for 50, 100 and 200 tasks ('*' -GFFS, 'o'-SRS+, '-' - SRS)