An Average-Case Analysis for Rate-Monotonic Multiprocessor Real-time Scheduling

Andreas Karrenbauer* and Thomas Rothvoß

Institute of Mathematics
École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
{*andreas.karrenbauer, thomas.rothvoss*}*@epfl.ch*

Abstract

We introduce the *First Fit Matching Periods* algorithm for rate-monotonic multiprocessor scheduling of periodic tasks with implicit deadlines and show that it yields asymptotically optimal processor assignments if utilization values are chosen uniformly at random. More precisely we prove that the *expected waste* is upper bounded by $\mathcal{O}(n^{3/4}(\log n)^{3/8})$. Here the waste denotes the ratio of idle times, cumulated over all processors and n gives the number of tasks.

The algorithm can be implemented to run in time $\mathcal{O}(n \log n)$ and even in the worst case, an asymptotic approximation ratio of 2 is guaranteed. Experiments yield an average waste proportional to $n^{0.70}$, indicating that the above upper bound on the expected waste is almost tight.

While such average-case analyses are a classical topic of Bin Packing, to the best of our knowledge, this is the first result dealing with a theoretical average-case analysis for this scheduling problem, which was described by Liu and Layland more than 35 years ago and has received a lot of attention, especially in the real-time and embedded-systems community.

1 Introduction

In this paper, we are concerned with a scheduling problem introduced by Liu and Layland [21], which is of fundamental importance in the real-time and embedded-systems community. Here one is given a set of *tasks* $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$, where each task τ is characterized by two positive values, its *period* $p(\tau)$ and its *running time* $c(\tau)$. The task τ releases a *job* requiring running time $c(\tau)$ at each integer multiple of its period. Each job has a relative deadline of $p(\tau)$, thus we have *implicit deadlines*. The *utilization* of a task τ is defined as $u(\tau) = c(\tau)/p(\tau)$, thus it gives the average fraction of processor cycles, which are consumed by τ. More general for a set $\mathcal{S}$, we denote $u(\mathcal{S}) = \sum_{\tau \in \mathcal{S}} u(\tau)$.

We consider *fixed-priority, preemptive* scheduling, i.e. priorities are assigned to the tasks and the arrival of a job of a higher priority task, preempts the execution of lower priority tasks. Liu and Layland [21] have proven that the *rate-monotonic* (RM) scheduling policy is optimal, meaning that if there is a feasible priority assignment, then the one in which the priority of a task τ equals $1/p(\tau)$ is also feasible (i.e. larger periods imply lower priorities). Therefore we only consider rate-monotonic priorities.

If several tasks $\mathcal{S}' \subseteq \mathcal{S}$ are assigned to one processor, then we call this assignment *feasible* (or RM-schedulable) if in the rate-monotonic schedule all jobs of all tasks always meet their deadlines. See Figure 1 for an example.

*Supported by the Deutsche Forschungsgemeinschaft (DFG) within Priority Programme 1307 "Algorithm Engineering"

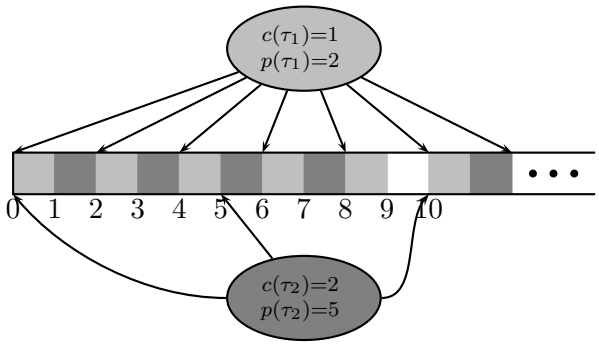

Figure 1: The picture shows a set $\mathcal{S} = \{\tau_1, \tau_2\}$ of tasks. The arrows indicate the points in time, where the two tasks τ_1 and τ_2 release jobs. At time 0, the first job of τ_1 as well as the first job of τ_2 are released. Since the period of τ_1 is smaller than the period of τ_2, the first job of τ_1 is executed, until it is finished at time 1. Now the first job of τ_2 is executed, but interrupted by the second job of τ_1 at time 2. The execution of the first job of τ_2 is resumed at time 3 and finished at time 4. Notice that the processor is idle for one time unit at time 9 and that the schedule repeats at the least common multiple of the periods which is 10. All jobs finish in time. The set $\mathcal{S}$ is feasible.

In a multiprocessor environment, the algorithmic challenge is to determine a partition of a task-set $\mathcal{S}$ into $\mathcal{S}_1, \ldots, \mathcal{S}_k$, such that each $\mathcal{S}_i$ is a feasible set of tasks for one processor and the number k of processors is minimized. The minimum possible value for k is denoted by OPT. The *rate-monotonic multiprocessor scheduling problem* has received considerable attention in the real-time and embedded-systems community [18, 17, 16, 1, 24, 5, 20, 15, 23, 2, 9, 19]. This popularity is due to the fact that more and more safety-critical control applications are carried out by microprocessors and in particular by multiprocessor environments. Such scheduling problems are for example relevant in the automotive and aviation industry.

A measure for the quality of a solution is the so-called *waste*, which is frequently used concerning the related *Bin Packing* problem. That is, the waste of a solution with k processors is the ratio of idles times, cumulated over all processors, i.e. $k - u(\mathcal{S})$. Clearly, minimizing the waste is equivalent to minimization of the number of partitions.

For the rate-monotonic single-processor scheduling Lehoczky et al. [18] gave a probabilistic analysis, indicating that the reachable processor utilization on average is much better, than the worst-case value of $\ln(2) \approx 69\%$. For example, if periods are drawn from $[1, 100]$ and the running times are scaled by the largest value, such that the system is barely schedulable, then the utilization tends to 88% for $n \to \infty$.

This motivates us to study also the average-case behavior in the multiprocessor case. Our analysis will work for an arbitrary distribution of the periods, as long as the utilization values are drawn independently and uniformly from $[0, 1]$.

Related work

For the famous *Bin Packing* problem a list of items $a_1, \ldots, a_n \in [0, 1]$ is given. The goal is to assign these items to a minimal number of bins such that the total sizes of items, assigned to each bin does not exceed 1.

We will see that if for the considered scheduling problem all periods $p(\tau)$ were multiples of each other, then the problem would be exactly Bin Packing, where the utilization values correspond to the item sizes. This is because a set of tasks $\mathcal{S}' \subseteq \mathcal{S}$ would be feasible on one processor in this case if and only if the sum

of their utilization is bounded by one.

Successful heuristics for Bin Packing are *First Fit*, *Next Fit* and *Best Fit*. In all variants the items are assigned in a consecutive manner to a bin, which has enough space (or a new one is opened). For First Fit the current item is put in the bin with the smallest index, in Best Fit it is assigned to the bin, whose item sum is maximal. For Next Fit an active bin is maintained. If the current item does not fit into it, a new bin is opened, now being the active one; old bins are never considered again. In *First Fit Decreasing* the items are first sorted by decreasing sizes and then distributed via First Fit. In the worst case Next Fit produces a 2-approximation, while First Fit needs $\lceil \frac{17}{10} OPT_{\text{BinPacking}} \rceil + 1$ [11] many bins. Asymptotically both, Best and First Fit Decreasing have an approximation ratio of $11/9$ [12].

If the items are generated randomly, the heuristics perform much better, than in the worst-case scenarios. For item sizes drawn uniformly at random from $[0, 1]$ the Best Fit algorithm yields an expected waste of $\Theta(\sqrt{n} \log^{3/4} n)$ [25], while for First Fit this value is lower bounded by $\Omega(n^{2/3})$ and upper bounded by $\mathcal{O}(n^{2/3}\sqrt{\log n})$ [25]. The upper bound even holds if First Fit is restricted to never assign more than 2 items per bin. Later we will refer to this algorithm as *Matching First Fit* (`MFF`). First Fit Decreasing yields an even smaller waste of $\Theta(\sqrt{n})$ [10, 14, 22]. If item sizes are drawn uniformly from $[0, \alpha]$, for any constant $\alpha \leq 1/2$, the waste of First Fit Decreasing is even constant with high probability.

Note that here the waste is defined similar to multiprocessor scheduling, namely as the number of bins minus the sum of all item sizes. But for Bin Packing also in the worst-case nearly optimal solutions can be computed, for example there is an asymptotic PTAS [8] and even an asymptotic FPTAS exists [13]. More on Bin Packing can be found in the excellent survey of Coffman et al. [3].

One major difference between rate-monotonic scheduling and Bin Packing is that for the latter it can be checked easily whether given items fit into one bin, whereas it is conjectured that this does not hold for a set of tasks and one processor. If a set $\mathcal{S}$ of implicit-deadline tasks is feasible (i.e. RM-schedulable), then the utilization $u(\mathcal{S})$ is at most 1. However, $\mathcal{S}$ can be infeasible, even if $u(\mathcal{S}) < 1$. Consider, for example, again the task system $\mathcal{S}$ in Figure 1. If we increase the running time of τ_2 by any $\varepsilon > 0$, then the set $\mathcal{S}$ is no longer feasible and its utilization is $u(\mathcal{S}) = (9 + 2\varepsilon)/10$. Liu and Layland [21] have shown that $\mathcal{S}$ is feasible, if $u(\mathcal{S})$ is bounded by $n(2^{1/n} - 1)$, where $n = |\mathcal{S}|$. This bound tends to $\ln(2) \approx 0.69$ and the condition is not necessary for feasibility, as the example in Figure 1 shows. Stronger, but still not necessary conditions for feasibility are given in [20, 5, 24]. Note that the first job of each task is the *critical instance* [21], thus if $p(\tau_1) \leq \ldots \leq p(\tau_n)$ then response times for τ_i in a rate-monotonic, single-processor schedule are given by the smallest value $r(\tau_i) \geq 0$ with

$$r(\tau_i) = c(\tau_i) + \sum_{j<i} \left\lceil \frac{r(\tau_i)}{p(\tau_j)} \right\rceil c(\tau_j).$$

Of course $\tau_1, \ldots, \tau_n$ are feasible if and only if $r(\tau_i) \leq p(\tau_i)$ for $i = 1, \ldots, n$ [18]. But it was proven in [7] that such response times cannot even be approximated in polynomial time within a factor of $n^{c/\log\log n}$ for a fixed constant $c > 0$, unless $\mathbf{NP} = \mathbf{P}$. Nevertheless in practice response times can be efficiently computed using a fix-point iteration approach [1]. Furthermore Baruah and Fisher [9] showed that there is an FPTAS for computing the minimum processor speed, which is needed to make a task system RM-schedulable.

Oh and Baker [23] showed that if m processors are needed to schedule $\mathcal{S}$, one must have $u(\mathcal{S}) \geq m \cdot (\sqrt{2} - 1) \approx 0.41m$. This quantity was later improved by Liebeherr et al. [20] to $m \cdot \frac{1}{1 + \sqrt[m]{2}} \approx 0.5m$ (for large m).

Most popular algorithms for rate-monotonic periodic multiprocessor scheduling first sort the tasks in a suitable way and then distribute them in a First Fit or Next Fit manner using a sufficient feasibility criterion. See the following table for an overview (with our algorithm in the last row, for the sake of comparability).

Name	sorting	distribution	ratio	run time
RMNF	inc. $p(\tau)$	Next Fit	2.67	$\mathcal{O}(n \log n)$
RMFF	inc. $p(\tau)$	First Fit	2.00	$\mathcal{O}(n \log n)$
FFDU	dec. $u(\tau)$	First Fit	2.00	$\mathcal{O}(n \log n)$
RMST	inc. $\alpha(\tau)$	Next Fit	$\frac{1}{1-u_{\max}}$	$\mathcal{O}(n \log n)$
RMGT	-	First Fit + RMST	1.75	$\mathcal{O}(n^2)$
FFMP	inc. $\alpha(\tau)$	First Fit	2.00	$\mathcal{O}(n \log n)$

Here $\alpha(\tau) = \log_2 p(\tau) - \lfloor \log_2 p(\tau) \rfloor$ and $u_{\max} = \max_{\tau \in \mathcal{S}} u(\tau)$. In the table, the column "ratio" gives the best known upper bounds on the asymptotic approximation ratio. The *rate-monotonic general task* algorithm (RMGT) [20] distributes tasks with utilization at most $1/3$ using RMST and the rest separately with First Fit. A more detailed description can be found in [19].

Furthermore there is an asymptotic PTAS under resource augmentation, computing for any fixed $\varepsilon > 0$ a solution with $(1+\varepsilon)OPT + 1$ processors, where the tasks on each processor can be feasibly scheduled after increasing the processor speed by a factor of $1 + \varepsilon$ [6]. In the same paper it was proven that unless $\mathbf{P} \neq \mathbf{NP}$ no asymptotic FPTAS can exist for this multiprocessor scheduling problem. But it is still an open question whether there might be an asymptotic PTAS and thus an algorithm that is asymptotically optimal and does not depend on any assumption about the input. Here we call an algorithm asymptotically optimal, if the approximation ratio tends to 1 for $OPT \to \infty$. We refer to the article of Baruah and Goossens [2] for an overview on complexity issues of real-time scheduling.

Our contribution

We introduce an efficient and easy to implement algorithm for the multiprocessor rate-monotonic scheduling problem called *First Fit Matching Periods* (FFMP) . We proof that it is asymptotically optimal for arbitrary periods provided that the utilizations follow a uniform distribution[1]. To this end, we show that our algorithm produces a solution with expected waste of $\mathcal{O}(n^{3/4}(\log n)^{3/8})$. Since the expected approximation ratio of $1 + \mathcal{O}(n^{-1/4}(\log n)^{3/8})$ tends to 1 for $n \to \infty$, the solution is asymptotically optimal on average. To the best of our knowledge this is the first proof that any algorithm for this problem admits this property w.r.t. a reasonable probability distribution.

To achieve our results, we use the following technique: We introduce an auxiliary algorithm FFMP* and prove that for any task set it needs at least as many processors as FFMP. Thus it suffices to derive an upper bound on the waste of this easier algorithm. We then point out that for suitable subsets of the input tasks, FFMP* behaves like a well studied Bin Packing algorithm MFF. Eventually this allows to bound the waste for FFMP* in terms of the waste of MFF.

In addition to the proof of the asymptotic optimality of our algorithm, we present experimental results showing that FFMP outperforms the algorithms known from literature already on random instances with a small number of tasks. We thereby provide an example of an algorithm that has been designed for asymptotic optimality and which is, in addition, competitive on reasonably small instances. Moreover, we present a family of instances where the average waste scales with $n^{0.70}$, which is almost tight to our theoretical upper bound and thus showing that our technique is suitable for sharp analyses.

[1] To be exact, we assume that first arbitrary periods may be given and then the utilizations are chosen randomly.

Algorithm 1 FFMP

Input: Set $\tau_1, \ldots, \tau_n$ of implicit-deadline tasks

(1) Sort tasks such that $0 \leq \alpha(\tau_1) \leq \alpha(\tau_2) \leq \ldots \leq \alpha(\tau_n) < 1$

(2) FOR $i = 1, \ldots n$ DO

(3) Assign τ_i to the processor P_j with least index j such that $u(P_j \cup \{\tau_i\}) \leq 1 - \beta(P_j \cup \{\tau_i\}) \cdot \ln(2)$

2 Preliminaries

For our algorithm we need the following sufficient (but still not necessary) schedulability condition of Burchard et al.

Lemma 1. *[20] For tasks $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ define*

$$\alpha(\tau_i) = \log_2 p(\tau_i) - \lfloor \log_2 p(\tau_i) \rfloor \quad \textit{and} \quad \beta(\mathcal{S}) := \max_{i=1,\ldots,n} \alpha(\tau_i) - \min_{i=1,\ldots,n} \alpha(\tau_i).$$

Then the tasks can be RM-scheduled on a single processor if $u(\mathcal{S}) \leq 1 - \beta(\mathcal{S}) \ln(2)$.

The intuition behind this is that a small value of $\beta(\mathcal{S})$ indicates that the periods of tasks in $\mathcal{S}$ are nearly multiples of each other and consequently the tasks are guaranteed to "harmonize".

The idea for our heuristic is now as follows: Sort the tasks w.r.t. their α-values. Then assign them in a First Fit manner using the sufficient feasibility test from Lemma 1. See Algorithm 1 for a formal description. Note that the *Rate-monotonic small tasks* algorithm (RMST) of Burchard et al. [20] is similar, just that a Next Fit assignment is used instead of First Fit. But already from average case analysis of Bin Packing, it is well know that Next Fit approaches generate linear waste for uniformly distributed item sizes [4].

3 The result of Shor

It is our aim to convey known bounds on the waste of Bin Packing algorithms to the waste of our algorithm. To this end we consider the following auxiliary algorithm *Matching First Fit* (MFF) of Shor [25], which distributes a list $L = (a_1, \ldots, a_n)$ of items to bins B_j. Denote $\text{size}(B_j) = \sum_{i \in B_j} a_i$.

Algorithm 2 Matching First Fit (MFF)

Input: Set $a_1, \ldots, a_n$ of items

(1) FOR $i = 1, \ldots, n$ DO

(2) Assign item a_i to the bin B_j with the least index j such that either B_j is empty or both of the following conditions hold

- B_j contains one item and this item has size at least $1/2$
- $\text{size}(B_j) + a_i \leq 1$

Shor [25] proved that MFF is monotonic, i.e. for all Bin Packing instances I and all items $a_i \in I$ one has

$$\texttt{MFF}(I) \geq \texttt{MFF}(I \backslash \{a_i\}) \geq \texttt{MFF}(I) - 1$$

where $\texttt{MFF}(I)$ denotes the number of bins used by MFF if applied to instance I. Furthermore MFF is never better than the pure First Fit algorithm and it has an expected waste of $\mathcal{O}(n^{2/3}\sqrt{\log n})$ for Bin Packing instances, whose item sizes are taken uniformly from $[0, 1]$.

Like MFF is a restriction to First Fit, we now state a restricted version of FFMP.

4 An auxiliary algorithm

Let $\gamma := \gamma(n)$ be an integer value, which we are going to choose later. We now define a simplified version $\texttt{FFMP}^*$ of FFMP which can be analyzed more easily. First the tasks are partitioned into *groups* $\mathcal{S}_1, \ldots, \mathcal{S}_\gamma$ with $\mathcal{S}_j = \{\tau_i \in \mathcal{S} \mid \frac{j-1}{\gamma} \leq \alpha(\tau_i) < \frac{j}{\gamma}\}$, thus the α-values of tasks from the same group differ only slightly. Next, $\texttt{FFMP}^*$ never assigns more than 2 tasks to each processor and tasks from different periods are never mixed. Here we say that an algorithm *mixes* two tasks τ_1, τ_2, if they are assigned to the same processor. The algorithm even considers a processor to be full if the first assigned task has a utilization of at most $\approx 1/2$. Note that this algorithm is precisely tailored for the used probability distribution. A formal definition of $\texttt{FFMP}^*$ now follows

Algorithm 3 $\texttt{FFMP}^*$

Input: Set $\tau_1, \ldots, \tau_n$ of implicit-deadline tasks

(1) Sort tasks such that $0 \leq \alpha(\tau_1) \leq \ldots \leq \alpha(\tau_n) < 1$
(2) Partition tasks into groups $\mathcal{S}_1, \ldots, \mathcal{S}_\gamma$ with $\mathcal{S}_j = \{\tau_i \in \mathcal{S} \mid \frac{j-1}{\gamma} \leq \alpha(\tau_i) < \frac{j}{\gamma}\}$.
(3) FOR $i = 1, \ldots n$ DO
  (4) Assign τ_i to the processor P_j with the least index j such that either P_j is empty or all following conditions are satisfied
    (a) P_j contains only one item and this item is from the same group as τ_i
    (b) the item on P_j has utilization $\geq (1 - \frac{\ln(2)}{\gamma})/2$
    (c) $u(P_j \cup \{\tau_i\}) \leq 1 - \frac{\ln(2)}{\gamma}$

Note that $1 - \frac{\ln(2)}{\gamma}$ is just slightly below 1. Observe that $\texttt{FFMP}^*$ assigns either 1 or 2 tasks to each processor. Let $\texttt{FFMP}^*(\mathcal{S})$ be the number of processors, needed when scheduling tasks $\mathcal{S}$ with algorithm $\texttt{FFMP}^*$. As a slight abuse of notation $\texttt{FFMP}^*(\mathcal{S})$ means as well the schedule, obtained when applying $\texttt{FFMP}^*$ to $\mathcal{S}$, however the meaning will be clear from the context. From Lemma 1 we see that the produced solution is always feasible since either a single task is assigned to a processor or in case that two tasks are assigned, their α-values differ by at most $1/\gamma$ and their cumulated utilization is upper bounded by $1 - \ln(2)/\gamma$.

The following observation is crucial for our analysis and allows to link the expected waste of $\texttt{FFMP}^*$ to MFF.

Observation 2. *Consider tasks $\tau_1, \ldots, \tau_m$ such that one has $\frac{j-1}{\gamma} \leq \alpha(\tau_i) < \frac{j}{\gamma}$ (i.e. all tasks fall into the same group) and $0 \leq u(\tau_i) \leq 1 - \frac{\ln(2)}{\gamma}$ for all $i = 1, \ldots, m$. Create m Bin Packing items $a_1, \ldots, a_m$ with item sizes $a_i := u(\tau_i) \cdot /(1 - \ln(2)/\gamma)$, i.e. $a_i \in [0, 1]$. Then $\texttt{FFMP}^*$ schedules $\tau_1, \ldots, \tau_m$ in exactly the same way, that MFF distributes $a_1, \ldots, a_m$, i.e. task τ_i is assigned to the ℓth processor if and only if item a_i is assigned to the ℓth bin. Especially $\texttt{FFMP}^*(\{\tau_1, \ldots, \tau_m\}) = \texttt{MFF}(\{a_1, \ldots, a_m\})$.*

The main result of this section will be to show that $\texttt{FFMP}^*(\mathcal{S}) \geq \texttt{FFMP}(\mathcal{S})$ for any set of tasks $\mathcal{S}$. The

simplicity of FFMP* will enable us to prove *monotonicity* for it, meaning that removing tasks from $\mathcal{S}$ can only lower the value of FFMP$^*(\mathcal{S})$. Although this is trivially true for algorithms yielding optimal solutions, for approximation algorithms with a complex behavior this does not necessarily hold.

Lemma 3. *For any set of tasks $\mathcal{S}$ and $\tau^* \in \mathcal{S}$ one has*

$$\texttt{FFMP}^*(\mathcal{S}) \geq \texttt{FFMP}^*(\mathcal{S}\backslash\{\tau^*\}) \geq \texttt{FFMP}^*(\mathcal{S}) - 1$$

Proof. Denote $\mathcal{S}' = \mathcal{S}\backslash\{\tau^*\}$ and let $\mathcal{S}_1, \ldots, \mathcal{S}_\gamma$ [$\mathcal{S}'_1, \ldots, \mathcal{S}'_\gamma$] be the groups of $\mathcal{S}$ [$\mathcal{S}'$, resp.]. Let i^* be the index such that $\tau^* \in \mathcal{S}_{i^*}$. Since the algorithm never mixes tasks from different groups one has FFMP$^*(\mathcal{S}'_i) =$ FFMP$^*(\mathcal{S}_i)$ for all $i \neq i^*$ and FFMP$^*(\mathcal{S}) = \sum_{i=1}^{\gamma}$ FFMP$^*(\mathcal{S}_i)$. Thus we may assume that all groups but $\mathcal{S}_{i^*}$ are empty. Furthermore tasks with utilization larger than $1 - \frac{\ln(2)}{\gamma}$ are never mixed with other tasks, thus their removal does not change the claim. Due to this we may assume that such tasks are not contained in $\mathcal{S} = \mathcal{S}_{i^*}$, hence $\mathcal{S}$ contains just tasks from the same group, all with utilization at most $1 - \frac{\ln(2)}{\gamma}$. Sticking together Observation 2 and the monotonicity of MFF [25] yields the claim. □

By iteratively applying Lemma 3 we obtain

Corollary 4. *For all task sets $\mathcal{S}$ and $\mathcal{S}' \subseteq \mathcal{S}$ one has*

$$\texttt{FFMP}^*(\mathcal{S}) \geq \texttt{FFMP}^*(\mathcal{S}').$$

We may now conclude that the restricted variant of FFMP never produces better solutions than FFMP itself.

Theorem 5. *For all task sets $\mathcal{S}$ one has*

$$\texttt{FFMP}^*(\mathcal{S}) \geq \texttt{FFMP}(\mathcal{S}).$$

Proof. Let $P_1 \dot\cup \ldots \dot\cup P_m = \mathcal{S}$ be the solution computed by FFMP and denote the groups of $\mathcal{S}$ by $\mathcal{S}_1, \ldots \mathcal{S}_\gamma$. Consider an arbitrary processor P_j and after renaming let $\tau_1, \ldots, \tau_p$ be the tasks on P_j in incoming order ($p \geq 1$). Remove $\tau_3, \ldots, \tau_p$. Given that $p \geq 2$, remove τ_2 if at least one of the following conditions is true

- τ_1 and τ_2 stem from different groups
- $u(\tau_1) < \frac{1}{2}(1 - \frac{\ln(2)}{\gamma})$
- $u(\{\tau_1, \tau_2\}) > 1 - \frac{\ln(2)}{\gamma}$

Let $\mathcal{S}' \subseteq \mathcal{S}$ the remaining tasks. Clearly FFMP* schedules $\mathcal{S}'$ in exactly the same way that FFMP schedules them in the solution leading to FFMP$(\mathcal{S})$. Thus FFMP$^*(\mathcal{S}') =$ FFMP$(\mathcal{S})$. From Corollary 4 we gain FFMP$^*(\mathcal{S}) \geq$ FFMP$^*(\mathcal{S}')$. Plugging both equations/inequalities together, yields the claim. □

5 An upper bound for FFMP*

In this section we will give an upper bound on the expected waste of FFMP*, by exploiting the bound on the waste of MFF. Again Observation 2 will be crucial.

Theorem 6. *Let $f : \mathbb{R}_{\geq 1} \to \mathbb{R}$ be a concave and monotonic increasing function, such that $f(n)$ yields an upper bound on the expected waste of* MFF *applied to n items drawn uniformly at random from $[0, 1]$. Then the expected waste of* FFMP* *is bounded by $\frac{n}{\gamma} + \gamma \cdot f(n/\gamma)$ for n tasks with arbitrary periods, but utilization values drawn uniformly at random from $[0, 1]$.*

Proof. Let $\mathcal{S}_1, \ldots, \mathcal{S}_\gamma$ be the partition of the tasks $\mathcal{S}$ into groups. Denote $n = |\mathcal{S}|$ and $n_i = |\mathcal{S}_i|$. $\texttt{FFMP}^*$ never mixes tasks from different groups, thus

$$\texttt{FFMP}^*(\mathcal{S}) = \sum_{i=1}^{\gamma} \texttt{FFMP}^*(\mathcal{S}_i).$$

Consider an arbitrary group $\mathcal{S}_i$. Call tasks τ with a utilization of $u(\tau) > 1 - \frac{\ln(2)}{\gamma}$ *full tasks* and *ordinary tasks* otherwise. Let $\mathcal{S}_i^{\text{full}}$ be the set of full tasks from $\mathcal{S}_i$ and let $\mathcal{S}_i' = \mathcal{S}_i \backslash \mathcal{S}_i^{\text{full}}$ be the ordinary tasks. Condition that $|\mathcal{S}_i'| = n_i^o$. Clearly the algorithm $\texttt{FFMP}^*$ does not mix ordinary and full tasks, thus

$$\texttt{FFMP}^*(\mathcal{S}_i) = \texttt{FFMP}^*(\mathcal{S}_i^{\text{full}}) + \texttt{FFMP}^*(\mathcal{S}_i').$$

A full task has a utilization of at least $1 - \frac{\ln(2)}{\gamma}$, thus for each full task it suffices to account a waste of $\frac{\ln(2)}{\gamma} \leq \frac{1}{\gamma}$. The expected waste stemming from the processors, owning the full tasks of group i is then

$$E[\texttt{FFMP}^*(\mathcal{S}_i^{\text{full}}) - u(\mathcal{S}_i^{\text{full}})] \leq \frac{n_i - n_i^o}{\gamma}.$$

It remains to bound the waste from the ordinary tasks. The utilization values of tasks in $\mathcal{S}_i'$ are conditioned to be in $[0, 1 - \frac{\ln(2)}{\gamma}]$. It is not difficult to see that the distribution of $u(\tau)$ for $\tau \in \mathcal{S}_i'$ is uniformly w.r.t. $[0, 1 - \frac{\ln(2)}{\gamma}]$. If we define a Bin Packing instance I_i' with an item of size $u(\tau)/(1 - \frac{\ln(2)}{\gamma})$ for each $\tau \in \mathcal{S}_i'$, then the item sizes in I_i' are distributed uniformly w.r.t. $[0, 1]$. By Observation 2

$$E[\texttt{FFMP}^*(\mathcal{S}_i')] = E[\texttt{MFF}(I_i')] \leq \frac{n_i^o}{2} + f(n_i^o).$$

The rest of the proof simply consists of summing up the achieved bounds on the waste. We can express the expected waste, stemming from the processors owning ordinary tasks from the ith group as

$$\begin{aligned} E[\texttt{FFMP}^*(\mathcal{S}_i') - u(\mathcal{S}_i')] \leq (\frac{n_i^o}{2} + f(n_i^o)) - E[u(\mathcal{S}_i')] &= (\frac{n_i^o}{2} + f(n_i^o)) - n_i^o \frac{1 - \ln(2)/\gamma}{2} \\ &\leq f(n_i^o) + \frac{n_i^o}{\gamma} \end{aligned}$$

Combining ordinary and full tasks yields

$$E[\texttt{FFMP}^*(\mathcal{S}_i) - u(\mathcal{S}_i)] \leq \frac{n_i - n_i^o}{\gamma} + (f(n_i^o) + \frac{n_i^o}{\gamma}) \leq f(n_i) + \frac{n_i}{\gamma}$$

using monotonicity of f. Hence the total expected waste for solution $\texttt{FFMP}^*(\mathcal{S})$ can be written as

$$E[\texttt{FFMP}^*(\mathcal{S}) - u(\mathcal{S})] \overset{(*)}{=} \sum_{i=1}^{\gamma} E[\texttt{FFMP}^*(\mathcal{S}_i) - u(\mathcal{S}_i)] \leq \sum_{i=1}^{\gamma} (f(n_i) + \frac{n_i}{\gamma}) \overset{(**)}{\leq} \frac{n}{\gamma} + \gamma \cdot f(n/\gamma)$$

For $(*)$ we used linearity of expectation and $(**)$ follows by Jensen's inequality and concaveness of f. □

Applying the best known bound on $f(n)$ we obtain

Theorem 7. *For the expected waste of* FFMP *one has*

$$E[\texttt{FFMP}(\mathcal{S}) - u(\mathcal{S})] = \mathcal{O}(n^{3/4}(\log n)^{3/8})$$

if $\mathcal{S}$ consists of n tasks, whose utilization values are drawn uniformly at random from $[0, 1]$.

Proof. Theorem 5 provides that bounding the waste of $\texttt{FFMP}^*$ is sufficient. Choosing $\gamma(n) = \lceil n^{1/4}/(\log n)^{3/8} \rceil$ and using the bound of $f(n) = \mathcal{O}(n^{2/3}(\log n)^{1/2})$ [25] together with Theorem 6 yields the claim (observe that $c \cdot n^{2/3} \cdot (\log n)^{1/2}$ is concave and monotonic). □

Observing that $OPT(\mathcal{S}) = \Omega(n)$ with very high probability, we conclude that

Corollary 8. *Let $\mathcal{S}$ consist of n tasks, whose utilization values are drawn uniformly at random from $[0, 1]$. Then the expected approximation ratio of* FFMP *is*

$$E\left[\frac{\texttt{FFMP}(\mathcal{S})}{OPT(\mathcal{S})}\right] \leq 1 + \mathcal{O}(n^{-1/4}(\log n)^{3/8})$$

Using essentially the same proof as [20] (see also Leung et al. [19]) one can easily show that even in the worst-case one has $\texttt{FFMP}(\mathcal{S}) \leq 2u(\mathcal{S}) + 4$, i.e. the asymptotic worst-case approximation ratio of FFMP is 2. For the sake of completeness the proof of this fact can be found in the appendix.

6 Experimental Results

We have performed simulations of our FFMP algorithm and compared it with RMFF, FFDU, and RMGT. The experimental setting is as follows. We choose the periods $p(\tau_i) \in [0, 500]$ and the utilizations $u(\tau_i) \in [0, 1]$ uniformly at random. We create random instances in the range of 10 to 100000 tasks. For each given n, we generate 100 random samples to get a good estimate of the expected value of the waste. We use the same instances to test each algorithm to allow also a direct comparison of their performance.

The log-log-plot in Figure 2 shows the power law behavior of the average waste of FFMP as predicted by Theorem 7. The regression yields an exponent of 0.70 which is close to $\frac{3}{4}$ from the Theorem showing that the theoretical analysis is almost tight, i.e. that we do not loose much by analyzing the dominated algorithm $\texttt{FFMP}^*$. In contrast to that, the average waste produced by the other algorithms shows an almost linear dependence on the number of tasks. In fact, we believe that the dependence is linear since the measurements of their average waste show a slight curvature to the left, indicating that the averages are actually growing faster than the fitted straight lines.

The simulated average processor load shown in Figure 3 supports this claim. By *average processor load*, we mean the expected value of the mean utilization of the processors. The closer this value is to 1 the less processor cycles are wasted. Hence, it comes to no surprise that the average load for FFMP tends to 1 with increasing n. For the other algorithms, there is strong evidence that they converge to respective constants strictly smaller than 1 and likely even not more than 0.9.

Interestingly, the quadratic running time of RMGT, which is due to the exact feasibility test for the large tasks, does not pay off in comparison with FFMP, which runs in $\mathcal{O}(n \log n)$ time. This does not only hold for the average waste, but also on a per instance basis: FFMP performs better than RMGT for 94 out of 100 random instances with 10 tasks and always better on our random test instances with a larger number of tasks. This is due to the splitting of the tasks into small tasks (i.e. utilization at most $1/3$) and large tasks. Thus all tasks with utilization at least $2/3$ are deterministically scheduled alone on a processor. For example in

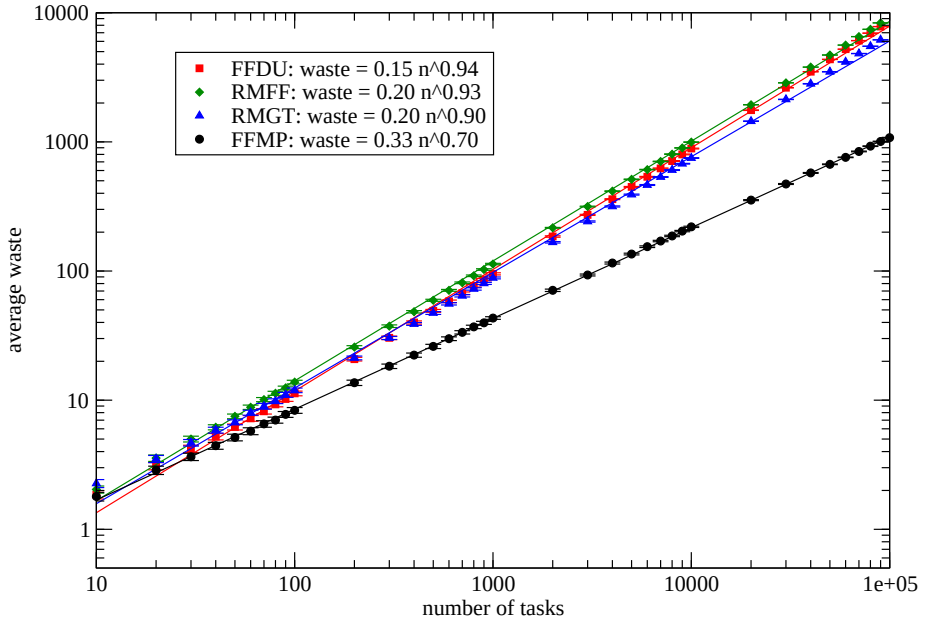

Figure 2: The average waste depending on the number of tasks are shown with 3σ error bars for `FFMP` and three algorithms from literature. For each algorithm, we fit a power function (straight lines in the log-log plot) and display the results in the legend box.

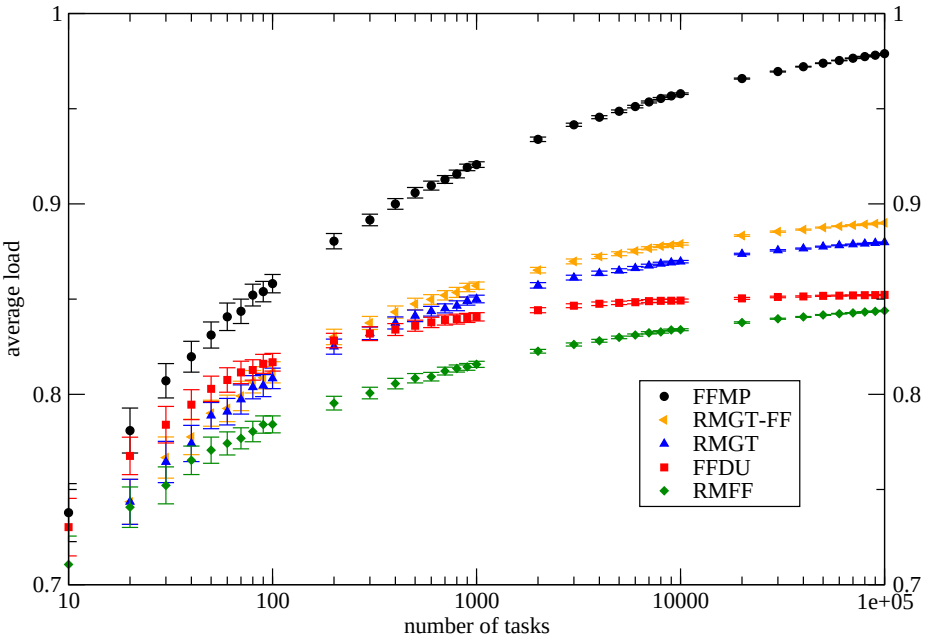

Figure 3: The average load of the processors is shown with 3σ error bars. In the algorithm RMGT-FF, the Next Fit distribution for the small tasks is replaced by First Fit.

expectation 10% of all tasks have a utilization between 0.7 and 0.8. Each of those tasks contributes at least 0.2 to the total waste. Therefore the expected waste of RMGT must be at least $0.1 \cdot 0.2 \cdot n = \Omega(n)$, even if after splitting, the algorithm would find an optimum solution for both parts. FFMP can be implemented in $\mathcal{O}(n \log n)$ using a heap data structure. For the sake of completeness a detailed description can be found in the appendix.

Acknowledgement

The authors want to thank Sanjoy K. Baruah, for reading a preliminary version of this paper and giving very helpful advices.

References

[1] Audsley, A. N., Burns, A., Richardson, M., and Tindell, K. (1993). Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, pages 284–292.

[2] Baruah, S. and Goossens, J. (2004). Scheduling real-time tasks: Algorithms and complexity. In Leung, J. Y.-T., editor, *Handbook of Scheduling — Algorithms, Models, and Performance Analysis*, Computer and Information Science Series, pages 28–28, Boca Raton-London-New York-Washington, D.C. Chapman & Hall/CRC.

[3] Coffman, Jr., E. G., Garey, M. R., and Johnson, D. S. (1984). Approximation algorithms for bin-packing—an updated survey. In *Algorithm design for computer system design*, volume 284 of *CISM Courses and Lectures*, pages 49–106. Springer, Vienna.

[4] Coffman, E.G., J., So, K., Hofri, M., and Yao, A. C. (1980). A stochastic model of bin-packing. *Inf. Control*, 44:105–115.

[5] Davari, S. and Dhall, S. K. (1995). On-line algorithms for allocating periodic-time-critical tasks on multiprocessor systems. *Informatica (Slovenia)*, 19(1).

[6] Eisenbrand, F. and Rothvoß, T. (2008a). A ptas for static priority real-time scheduling with resource augmentation. In *ICALP (1)*, pages 246–257.

[7] Eisenbrand, F. and Rothvoß, T. (2008b). Static-priority Real-time Scheduling: Response Time Computation is NP-hard. In *IEEE Real-Time Systems Symposium (RTSS)*.

[8] Fernandez de la Vega, W. and Lueker, G. S. (1981). Bin packing can be solved within $1 + \varepsilon$ in linear time. *Combinatorica*, 1(4):349–355.

[9] Fisher, N. and Baruah, S. (2005). A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines. In *ECRTS '05: Proceedings of the 17th Euromicro Conference on Real-Time Systems (ECRTS'05)*, pages 117–126, Washington, DC, USA. IEEE Computer Society.

[10] Frederickson, G. N. (1980). Probabilistic analysis for simple one- and two-dimensional bin packing algorithms. *Information Processing Letters*, 11(4/5):156–161.

[11] Garey, M. R., Graham, R. L., Johnson, D. S., and Yao, A. C. C. (1976). Resource constrained scheduling as generalized bin packing. *J. Combin. Theory Ser. A*, 21:257–298.

[12] Johnson, D. S. (1973). *Near-optimal bin packing algorithms*. PhD thesis, MIT, Cambridge, MA.

[13] Karmarkar, N. and Karp, R. M. (1982). An efficient approximation scheme for the one-dimensional bin-packing problem. In *23rd annual symposium on foundations of computer science (Chicago, Ill., 1982)*, pages 312–320. IEEE, New York.

[14] Knödel, W. (1981). A bin packing algorithm with complexity $O(n \log n)$ and performance 1 in the stochastic limit. In Gruska, J. and Chytil, M., editors, *Mathematical Foundations of Computer Science 1981*, volume 118 of *lncs*, pages 369–378, Štrbské Pleso, Czechoslovakia. Springer-Verlag.

[15] Korst, J., Aarts, E., and Lenstra, J. K. (1997). Scheduling periodic tasks with slack. *INFORMS J. Comput.*, 9(4):351–362.

[16] Korst, J., Aarts, E. H. L., Lenstra, J. K., and Wessels, J. (1991). Periodic multiprocessor scheduling. In *PARLE '91: Proceedings on Parallel architectures and languages Europe : volume I: parallel architectures and algorithms*, pages 166–178, New York, NY, USA. Springer-Verlag New York, Inc.

[17] Lehoczky, J. P. (1990). Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *IEEE Real-Time Systems Symposium*, pages 201–213.

[18] Lehoczky, J. P., Sha, L., and Ding, Y. (1989). The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *IEEE Real-Time Systems Symposium*, pages 166–171.

[19] Leung, J., Kelly, L., and Anderson, J. H. (2004). *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, Inc., Boca Raton, FL, USA.

[20] Liebeherr, J., Burchard, A., Oh, Y., and Son, S. H. (1995). New strategies for assigning real-time tasks to multiprocessor systems. *IEEE Trans. Comput.*, 44(12):1429–1442.

[21] Liu, C. L. and Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61.

[22] Lueker, G. S. (1982). An average-case analysis of bin packing with uniformly distributed item sizes. Technical Report 181, Dept. Information and Computer Science, University of California at Irvine.

[23] Oh, D.-I. and Baker, T. P. (1998). Utilization bounds for N-processor rate monotone scheduling with static processor assignment. *Real-Time Systems*.

[24] Oh, Y. and Son, S. H. (1995). Allocating fixed-priority periodic tasks on multiprocessor systems. *Real-Time Syst.*, 9(3):207–239.

[25] Shor, P. W. (1984). The average-case analysis of some on-line algorithms for bin packing. In *Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science, FOCS'84 (Singer Island, FL, October 24-26, 1984)*, pages 193–200. IEEE, IEEE.

Appendix

Implementation

In the following, we explain how to implement the FFMP algorithm with a complexity of $\mathcal{O}(n \log n)$ using a heap data-structure. To this end, we rewrite the predicate of the feasibility test as follows. Consider the current task τ_i and a processor P. Recall that the tasks are ordered by increasing α-values. Thus, the value of β always depends on the current task and the task already on P that defines the minimum α-value, say $\alpha(P) = \min\{\alpha(\tau_j) \mid \tau_j \in P\}$. Hence, task τ_i can be scheduled on processor P if

$$u(P) + u(\tau_i) \leq 1 - (\alpha(\tau_i) - \alpha(P)) \cdot \ln(2)$$

which is equivalent to

$$\underbrace{u(\tau_i) + \alpha(\tau_i)\ln(2)}_{v_i} \leq \underbrace{1 - u(P) + \alpha(P)\ln(2)}_{\ell_P}.$$

Note that the left-hand side (called v_i in the following) only depends on the current task τ_i, and that the right-hand side (called ℓ_P in the following) only depends on the tasks which are already scheduled on the processor P. For $P = \emptyset$, we define $\ell_P = \infty$. Hence, we may maintain a binary heap data-structure to find in logarithmic time the processor with the least index that passes the feasibility test. Viewed as a binary tree, we have n leaves corresponding to processors. They are labeled with the right-hand sides ℓ_P depending on their current utilization and the α-value of the first task which was scheduled on each one. Initially, they are empty and their labels are ∞. The other nodes of the tree are labeled with the maximum label of their respective children. See Figure 4 for a visualisation. Starting from the root, we proceed with the left child if the v_i value for the current task is not greater than the label of the left child. Otherwise, we turn to the right child, which then has a sufficiently large label by construction. The leaf that we eventually reach determines the processor on which we schedule the current task. Since the height of the binary tree is logarithmic in the number of leaves, i.e. n, we can schedule each task in $\mathcal{O}(\log n)$. Moreover, the update of the data-structure also takes $\mathcal{O}(\log n)$ since we only need to traverse the tree back to the root and update the labels on this path; any other label remains invariant.

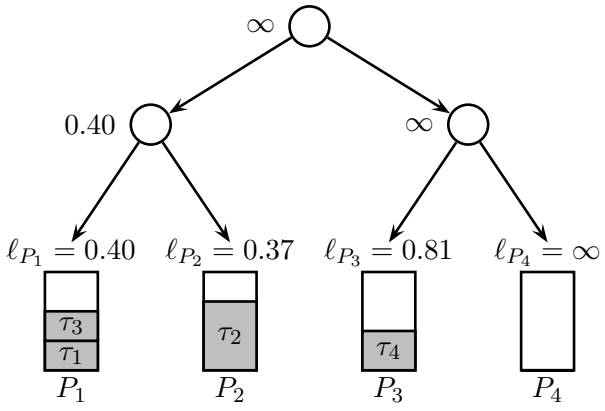

Figure 4: Example for the heap structure. We use notation $\tau_i = (u(\tau_i), \alpha(\tau_i))$. Then FFMP schedules tasks $\tau_1 = (0.3, 0.0), \tau_2 = (0.7, 0.1), \tau_3 = (0.3, 0.2), \tau_4 = (0.4, 0.3)$ as depicted.

Worst-case behaviour

In this section we outline, why the asymptotic worst-case approximation ratio is bounded by 2.

Lemma 9. *One has*

$$\texttt{FFMP}(\mathcal{S}) \leq 2 \cdot u(\mathcal{S}) + 4.$$

Proof. Let $P_1, \ldots, P_m$ be the used processors. By $\alpha(P_i) = \min\{\alpha(\tau_j) \mid \tau_j \in P\}$ we denote the α-value of the first task, assigned to P_i. Clearly $0 \leq \alpha(P_1) \leq \ldots \leq \alpha(P_m) < 1$. Let τ_j be the first task, assigned to P_{i+1} for $i \in \{1, \ldots, m-1\}$, thus $\alpha(\tau_j) = \alpha(P_{i+1})$. But the $(i+1)$th processor was only opened because τ_j did not fit on a prior processor. Especially it did not fit on P_i, thus

$$\begin{aligned} u(P_i) + u(P_{i+1}) &\geq u(P_i) + u(\tau_j) \\ &> 1 - \beta(P_i \cup \{\tau_j\}) \cdot \ln(2) \\ &\geq 1 - \ln(2) \cdot (\alpha(P_{i+1}) - \alpha(P_i)) \end{aligned}$$

Hence

$$\begin{aligned} u(\mathcal{S}) &\geq \sum_{i=1}^{\lfloor m/2 \rfloor} (u(P_{2i-1}) + u(P_{2i})) \\ &\geq \sum_{i=1}^{\lfloor m/2 \rfloor} (1 - \ln(2) \cdot (\alpha(P_{2i}) - \alpha(P_{2i-1}))) \\ &\geq \lfloor m/2 \rfloor - \ln(2) \\ &\geq m/2 - 2 \end{aligned}$$

using that the differences of the α-values sum up to at most 1. We conclude that

$$m \leq 2u(\mathcal{S}) + 4.$$

□