Theoretical Computer Science 420 (2012) 89–98

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Single and multiple device DSA problems, complexities and online algorithms

Weiwei Wu [a,b,c], Minming Li [b,*], Wanyong Tian [a,b,c], Jason Chun Xue [b], Enhong Chen [a]

[a] School of Computer Science, University of Science and Technology of China, China
[b] Department of Computer Science, City University of Hong Kong, Hong Kong
[c] USTC-CityU Joint Research Institute, China

ARTICLE INFO

Article history:
Received 20 October 2010
Accepted 4 November 2011
Communicated by D.-Z. Du

Keywords:
Dynamic Storage Allocation
Online algorithms
Multiple device
NP-completeness

ABSTRACT

We study the single-device Dynamic Storage Allocation (DSA) problem and the multi-device Balancing DSA problem in this paper. The goal is to dynamically allocate the job into memory to minimize the usage of space without concurrency. The SRF problem is just a variant of the DSA problem. Our results are as follows.

- The NP-completeness for the 2-SRF problem, 3-DSA problem, and DSA problem for jobs with agreeable deadlines.
- An improved 3-competitive algorithm for jobs with agreeable deadlines on single-device DSA problems. A 4-competitive algorithm for jobs with agreeable deadlines on multi-device Balancing DSA problems.
- Lower bounds for jobs with agreeable deadlines: any non-clairvoyant algorithm cannot be $(2-\epsilon)$-competitive and any clairvoyant algorithm cannot be $(1.54-\epsilon)$-competitive.
- The first $O(\log L)$-competitive algorithm for general jobs on multi-device Balancing DSA problems without any assumption.

1. Introduction

This paper studies the Dynamic Storage Allocation (DSA) problem, which is a classic problem in computer science. The problem description is as follows. Given a set of n jobs $\mathcal{J}$, each job J_i is characterized by a *size* s_i, an *arrival time* r_i, and a *departure time/deadline* d_i. Each job should be allocated in a contiguous location at its arrival time. Once placed into address $a(J_i)$, the job occupies the allocated space from address $a(J_i)$ to $a(J_i) + s_i - 1$ (with size s_i) in the whole time interval $[r_i, d_i]$. The occupied space is available for other jobs after time d_i. Two jobs i, j are assigned with *conflict* if $[r_i, d_i] \cap [r_j, d_j] \neq \emptyset$ and $[a(J_i), a(J_i) + s_i - 1] \cap [a(J_j), a(J_j) + s_j - 1] \neq \emptyset$. We need to place all the jobs into the memory without conflict at any time. The objective is to minimize the memory space used by all jobs. Another interpretation of DSA is to consider the following variant of the Strip Packing Problem. Packing into a strip can be interpreted as packing into a 2D geometric plane $X \times Y$ with fixed width X. In the Strip Packing Problem, each rectangle is a $(d_i - r_i + 1) \times s_i$ area. The objective is to pack all the rectangles into a strip that has a fixed width, while minimizing the maximum height of this strip. Obviously, DSA is a variant of the Strip Packing Problem (we would not survey the work in this field here) where the position of rectangles in X-axis is immovable. DSA has received much attention since 1950s. Knuth proposed some basic methods in [12]. Stockmeyer proved its NP-completeness by a reduction from the 3-partition problem in 1976. He also proved DSA to be

* Corresponding author. Tel.: +852 27889538; fax: +852 27888614.
E-mail addresses: wweiwei2@gmail.com (W. Wu), minming.li@cityu.edu.hk (M. Li), frog@mail.ustc.edu.cn (W. Tian), jason@cityu.edu.hk (J.C. Xue), cheneh@ustc.edu.cn (E. Chen).

0304-3975/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2011.11.005

strongly NP-complete even when sizes of jobs are restricted to {1,2} (The proof was given in Larry Stockmeyer's private communication with David Johnson and included in [3]). For an *offline* version of DSA, the first constant-factor approximation algorithm—an 80-approximation first-fit mechanism was proposed by Kierstead in [10] where jobs are sorted according to their sizes, whereafter Kierstead reduced the approximation ratio to 6 in [11]. Subsequently, Gergov presented a 5-approximation algorithm in [7] and then improved to a 3-approximation algorithm in [8]. Narayanaswamy gave another 3-approximation algorithm in [16]. The algorithm with an approximation ratio $2 + \epsilon$ is proposed by Buchsbaum et al. in [2]. For the special case of DSA with only two job sizes, Gergov [7] presented a 2-approximation algorithm. When job sizes are restricted to small values, Li et al. [13] proposed a $\frac{4}{3}$-approximation algorithm when the sizes are $\{1, 2\}$ and a 1.7-approximation algorithm when the sizes are $\{1, 2, 3\}$. For an *online* version of DSA, Robson [17] showed that first-fit algorithm had a competitive ratio of $O(\log(S_{max}))$ where S_{max} is the maximum size among all jobs. Then Luby et al. [14] proved that first-fit algorithms could achieve the competitive ratio of $O(\min\{\log(S_{max}), \log(\chi)\})$, where χ denotes the maximum number of concurrent jobs for all time instants. This is an improvement over Robson [17] since generally S_{max} and χ are incomparable. They also studied the multi-device Balancing DSA problem (minimizing the maximum occupied space on m devices) and gave an optimal competitive algorithm under the assumption that $m \leq \frac{\omega^*}{S_{max}}$ where ω^* represents the maximum total size of alive jobs at any time, which is also denoted as $LOAD(\mathcal{J})$ in this paper. Gergov [7] improved the competitive ratio of first-fit strategy of online DSA to $\Theta(max\{1, \log(S_{max} \cdot \chi/\omega^*)\})$. For cases that the live-ranges (which equals the difference of deadline and arrival time) of jobs are known when they arrive (this scenario is also referred to as *clairvoyant* scheduling), Naor et al. [15] provided an online 8-competitive algorithm for jobs with agreeable deadlines (where later released jobs have no earlier deadlines) and an $O(\min\{(\log(L), \log(\tau))\})$-competitive algorithm for general jobs, where L is the ratio between the longest and the shortest live-ranges of the jobs, and τ denotes the maximum number of concurrent jobs that have different live-ranges. [9] provides a lower bound $\Omega(\frac{\log x}{\log\log x})$-competitive for online algorithms where x can be n, S_{max}, χ, τ or $\log L$.

DSA is investigated in many applications such as memory or register allocation in operating systems and bandwidth allocation in communication networks etc. In operating systems, compiler algorithms should decide where to place items (*arrays* or *matrices etc.*) in memory. To deal with register allocation problems or DSA problems, another conventional mechanism is to use graph coloring. Graph coloring has been extensively investigated and specifically studied for register allocation by Chaitin [4]. From [4] on, many register allocation algorithms based on graph coloring have been proposed such as [5,1,6]. Our study of DSA is motivated by stream register file (SRF) allocation problem. SRF is a software-managed on-chip memory. Since it is a critical resource, optimizing SRF utilization becomes crucial. Recently in [18], by dividing streams (can be referred to as *jobs* here) into two types (short streams with live-ranges $l \leq 2$ and long streams with live-ranges $l \geq 3$), they modeled the problem into comparability graph coloring. They considered a special case where there are at most two long items at any time instant and gave a near-optimal solution, without showing the complexity of this problem. We will refer to their model as 2-*SRF problem*.

We study both the complexities and algorithms for the DSA problem and its variants. Note that the SRF allocation problem studied in [18] can be transformed into DSA notation. When considering the duration of the jobs, one interesting question is that how large the value L should be in order to make the problem hard. We will refer to the problem where every job has live-range at most L as L-*DSA problem*. For the general DSA problem, we improve the performance for the single-device DSA problem studied in [15] and give the first competitive algorithm for the multi-device Balancing DSA problem without any assumption. We start by investigating the jobs with *agreeable deadlines*, which was also studied in [15]. This kind of jobs has received much attention in other domains of scheduling problems. We derive the complexity for this type of jobs for completeness. Moreover, although our method leads to the same performance as [15] for the offline setting, it is proved more extendable to the online setting both for single-device and multi-device. Our results are as follows,

- The NP-completeness for 2-SRF problem, 3-DSA problem, and DSA problem for jobs with agreeable deadlines.
- An improved 3-competitive algorithm over [15] for jobs with agreeable deadlines on single-device DSA problem. A 4-competitive algorithm for jobs with agreeable deadlines on multi-device Balancing DSA problem.
- Lower bounds for jobs with agreeable deadlines: any non-clairvoyant algorithm cannot be $(2 - \epsilon)$-competitive and any clairvoyant algorithm cannot be $(1.54 - \epsilon)$-competitive.
- The first $O(\log L)$-competitive algorithm for general jobs on multi-device Balancing DSA problem without any assumption.

The rest of the paper is organized as follows. In Section 2, we review the model for DSA problem and multi-device Balancing DSA problem. In Section 3, we settle the complexities for L-DSA problem and 2-SRF problem. In Section 4, we investigate the single-device DSA problem. We propose the laminar decomposition for jobs with agreeable deadlines. The complexity for jobs with agreeable deadlines is settled. We have a simple 2-approximation algorithm through laminar decomposition. This decomposition can be used to design an improved 3-competitive algorithm (for jobs with agreeable deadlines). We also show some lower bounds for both clairvoyant and non-clairvoyant online algorithms. In Section 5, we design competitive algorithms for multi-device Balancing DSA problem. A 4-competitive algorithm for jobs with agreeable deadlines is proposed and also an $O(\log L)$-competitive algorithm for general jobs. To make the paper smooth, some proof is put in the Appendix.

2. Preliminaries

In operating systems, memory allocation algorithms need to consider where to place a set of items (often arrays, matrices etc.) so that the overall usage of memory is minimized. In DSA (Dynamic Storage Allocation) notation, we are given jobs $\mathcal{J}$ where each job i (or J_i) has a *size* s_i and a *live-range/length* starting from the *arrival time* r_i and departing at *deadline* d_i, i.e. $J_i = (r_i, d_i, s_i)$. A job i is said *alive at time* t if $t \in [r_i, d_i]$. The time is partitioned into units. For each unit of time t, if we say i has *arrival time* t, we mean job i is released at the beginning of time t. Correspondingly, if we say i has *deadline* t, we mean job i expires at the end of time t. Upon arrival of each job, we need to allocate a contiguous memory location for it. We assume that the memory is starting from address 1 instead of address 0. By $a(J_i)$ we denote that J_i is assigned or allocated to address $a(J_i)$ throughout its live-range. Once placed into address $a(J_i)$, J_i occupies the space from address $a(J_i)$ to $a(J_i)+s_i-1$ (with size s_i) in the whole time interval $[r_i, d_i]$. The occupied space is available for other jobs after time d_i. Two jobs i, j are assigned with *conflict* if $[r_i, d_i] \cap [r_j, d_j] \neq \emptyset$ and $[a(J_i), a(J_i)+s_i-1] \cap [a(J_j), a(J_j)+s_j-1] \neq \emptyset$. We need to place all the jobs into the memory without conflict at any time. Once allocated, the job cannot be moved or deleted until its deadline. Only when the job leaves, its occupied memory location can be available for other jobs. The objective is to minimize the overall used memory size. For the multi-device *Balancing DSA problem*, we are given m devices and the goal is to allocate the job in a balanced manner such that the maximum occupied space on the m devices is minimum. By S_{max} we denote the maximum size over all jobs. Let $LOAD(t)$ be the total size of alive jobs at time t. We use $LOAD(\mathcal{J})$ to represent the maximum total size over all time t. Obviously $max\{LOAD(\mathcal{J}), S_{max}\}$ is a lower bound for the optimal solution for single-device DSA problem, while multi-device Balancing DSA problem has a lower bound $max\{\frac{LOAD(\mathcal{J})}{m}, S_{max}\}$.

Observing that in the applications such as SRF problem each item usually has a short live-range, we use *L-DSA problem* to denote the restricted DSA problem where all jobs have live-ranges at most L. The jobs with *agreeable deadlines* which receive much attention in scheduling literature are defined to be jobs where later released jobs always have no earlier deadlines.

We use $OPT(\mathcal{J})$ to denote the optimal memory space occupied by the optimal solution for jobs $\mathcal{J}$. An offline algorithm is said to be c-approximation if it outputs a solution which occupies memory space at most $c \cdot OPT(\mathcal{J})$. In the online setting, the algorithm should make decision upon the arrival of jobs. There are two versions. In the *clairvoyant* scheduling, the algorithm knows the deadline when a job is released, while in the *non-clairvoyant* scheduling the algorithm does not know this information. We say an online algorithm is c-competitive if it always outputs a solution within c times the optimal offline solution.

3. Settling the complexity for special cases of DSA/SRF problem

The decision version of *Partition problem* is as follows. Given finite set $U = \{u_1, u_2, \ldots, u_n\}$ with $\sum_{i=1}^{n} u_i = 2B$. The question is to find a subset $U' \subset U$ such that the sum of the elements in U' is exactly B. In the following we will show that the 2-SRF problem and L-DSA problem are NP-complete. The construction is reduced from Partition problem.

The problem defined in [18] is referred to as 2-SRF problem in this paper. After transforming their paradigm to the DSA notation, the problem can be considered equivalently as follows. There are two kinds of jobs, *short* jobs and *long* jobs. All the short jobs have live-ranges at most 2. The long jobs have live-ranges larger than 2, but each time there are at most two such long alive jobs. The justification of such a problem is simple because we can split the long live-range stream (job) into streams with short live-ranges by live-range splitting technique. This allows us to break long jobs into jobs limited to some constant length L. They discussed this problem by five sub-cases, among which four of these cases can be solved optimally by their proposed algorithm, while the fifth case is shown to be within the optimal solution plus 2 times the maximum size of the long jobs. Thus the complexity of this problem still remains open. The following theorem answers this question.

Theorem 1. *The decision version of 2-SRF problem is NP-complete even when all jobs have live-ranges at most $L = 3$.*

Proof. Given any instance (n items) of Partition problem, we construct an instance of the 2-SRF problem with $n + 8$ jobs. Moreover, all the long jobs we construct have live-ranges $L = 3$ and at any time at most two long jobs are alive.

The jobs we construct are as follows. We set $B = \sum_{i=1}^{n} u_i/2$.

$$J_i = (3, 3, u_i) \quad \text{for all } 1 \leq i \leq n;$$
$$J_{n+1} = (1, 3, B), \quad J_{n+2} = (3, 4, 2B), \quad J_{n+3} = (4, 4, 8B),$$
$$J_{n+4} = (2, 4, 2B), \quad J_{n+5} = (1, 1, 8B), \quad J_{n+6} = (1, 2, 3B), J_{n+7} = (2, 2, B), \quad J_{n+8} = (2, 3, 5B).$$

In this instance, the decision version of the 2-SRF problem is to find a solution with memory size $M \leq 12B$. We will show that computing a solution with memory size $M \leq 12B$ (namely we need to pack the jobs into a $4 \times 12B$ box) is equivalent to partitioning the items into two sets with the same size $B = \sum_{i=1}^{n} u_i/2$. We discuss by cases.

Case 1: J_{n+1} is placed between J_{n+5} and J_{n+6}.

W.l.o.g we assume that J_{n+6} is placed on the top, as shown in Fig. 1(a). This makes J_{n+4} impossible to be placed on the top. Two sub-cases should be considered. If J_{n+2} is on top, then J_{n+3} is forced to be placed between J_{n+2} and J_{n+4}. In this situation, no matter what order J_{n+7} and J_{n+8} are placed, there are two separated gaps with size B at time 3. The remaining n short jobs can be packed into the box if and only if we could partition their job sizes (u_i where $1 \leq i \leq n$) into two sets each with size exactly B. We already complete the proof in this case and will show contradiction for other cases. If J_{n+3} is on top,

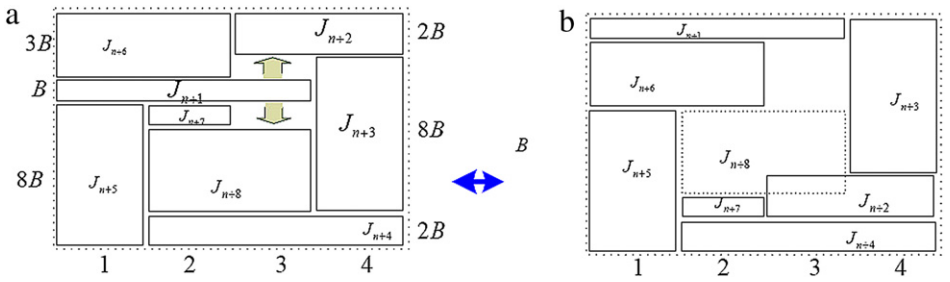

Fig. 1. Reduction 1.

then obviously there are two orders for J_{n+2} and J_{n+4} to be placed at the bottom. One natural way is to place J_{n+2} above J_{n+4}. But then no matter what orders J_{n+7} and J_{n+8} are placed, J_{n+8} will conflict with J_{n+2}. Also, if J_{n+4} is above J_{n+2}, then J_{n+8} will conflict with J_{n+4}.

Case 2: J_{n+1} is placed above J_{n+5} and J_{n+6}.

In this case, both J_{n+2} and J_{n+4} can only be placed below J_{n+3}. Clearly this forces J_{n+5} to be placed below J_{n+6} or otherwise J_{n+6} will conflict with J_{n+4}. We should discuss the two sub-cases that J_{n+2} is above J_{n+4} or not. Careful discussions will show that J_{n+8} will conflict with at least one of J_{n+2} and J_{n+4}, as shown in Fig. 1(b).

Thus our construction only allows the latter 8 jobs to be arranged as shown in Fig. 1(a). To further place the first n jobs at time 3, we need to partition them into two sets each with size exactly B. This finishes our reduction. □

Note that the reduction in Theorem 1 uses jobs that have live-ranges at most 3, thus 3-DSA problem is also NP-complete. Simple extension of this construction can show the NP-completeness for every $L \geq 3$.

Corollary 1. *The decision version of L-DSA problem is NP-complete for $L \geq 3$.*

4. Better online algorithms for DSA problem

In this section, we start by studying jobs that have *agreeable deadlines*. In this setting, all the later released jobs will have no earlier deadlines. This kind of jobs has also received much attention in other domains of scheduling problems. We will first show that the problem (with agreeable deadlines) is still NP-complete in the offline setting. Then we design a 2-approximation algorithm for jobs with agreeable deadlines in the offline version. This matches the result in [15]. However, we will show that the concept we use is more extendable to the online problem both in single-device and multi-device setting.

4.1. Approximation algorithm through laminar decomposition

The following reduction shows that the problem is still NP-complete for jobs with agreeable deadlines. The reduction uses a symmetric structure of the jobs.

Theorem 2. *DSA problem for jobs with agreeable deadlines is NP-complete.*

Proof. Given any instance (n items and total value $u_1 + u_2 + \cdots + u_n = 2B$) of Partition problem, we construct an instance with $2n + 7$ jobs. Distinguishing with the previous reduction, an important observation of this construction is that all the constructed jobs form a symmetric structure w.r.t time 3. All the jobs we construct have agreeable deadlines. We will define two groups of small jobs that have live-ranges 1. The first group is jobs $\{J_i : J_i = (1, 1, u_i), 1 \leq i \leq n\}$ while the second group is jobs $\{J_i : J_i = (5, 5, u_i), n + 8 \leq i \leq 2n + 7\}$. The remaining seven jobs we construct are as follows.

$$J_{n+1} = (1, 2, 3B), \quad J_{n+2} = (1, 3, B), \quad J_{n+3} = (2, 3, B),$$
$$J_{n+4} = (2, 4, B), \quad J_{n+5} = (3, 4, B), \quad J_{n+6} = (3, 5, B), \quad J_{n+7} = (4, 5, 3B).$$

We will show that there is a solution no larger than $6B$ for our instance if and only if either the first group or the second group of jobs can be equally divided into two sub-sets each with equal size B. Equivalently, we need to pack the jobs into a $5 \times 6B$ box. Thus we focus on creating two equal gaps each with size B either at time 1 or time 5 in the following. We discuss by cases.

Case 1: J_{n+1} is placed at address $B + 1$.

The simple case is that J_{n+2} is at address $4B + 1$ or $5B + 1$, then we successfully obtain two equal gaps at time 1. If J_{n+2} is at the bottom (address 1), then J_{n+3}, J_{n+4} should occupy the $2B$ space on the top at time 3 and J_{n+5}, J_{n+6} should occupy the $3B$ space in the middle at time 3. This makes J_{n+7} impossible to be placed without conflict.

Case 2: J_{n+1} is placed at the bottom (at address 1).

Job $J_{n+2}, J_{n+3}, J_{n+4}$ should occupy the $3B$ space on the top at time 2. If J_{n+2} is at address $4B + 1$, then we already have two equal gaps at time 1. If J_{n+2} is at address $3B + 1$, then there are two sub-cases. For the first one, if J_{n+4} is on top, then job J_{n+7}

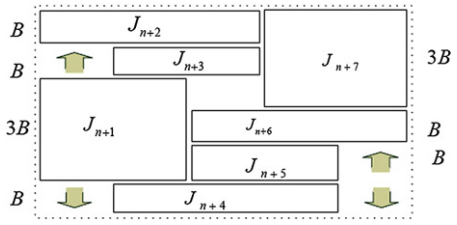

Fig. 2. Reduction 2.

could only be at address $2B + 1$ (Such an allocation can be demonstrated by rotating Fig. 2 180° clockwise and renaming job J_{n+i} to be J_{n+8-i} where $1 \leq i \leq 7$ since it is a symmetric structure). No matter how J_{n+5}, J_{n+6} are placed, we have two equal gaps at time 5 in this situation. For the second case, if J_{n+3} is on top, then at least one of $J_{n+5}, J_{n+6}, J_{n+7}$ will concurrent with J_{n+4}. It remains to show the case that J_{n+2} is at address $5B + 1$. Again in this situation, at least one of $J_{n+5}, J_{n+6}, J_{n+7}$ is concurrent with J_{n+4}.

Case 3: J_{n+1} is placed at address $2B + 1$ or $3B + 1$.

Note that the space excluding J_{n+1} in Case 2 is $6B - 3B = 3B$. Thus the case that J_{n+1} is placed at address $3B + 1$ (or $2B + 1$) is the same as Case 2 (or Case 1).

In all cases we have shown that the constructed instance has a solution $6B$ if and only if either there are two equal gaps at time 1 or time 5. Namely, we have a solution $6B$ if and only if Partition problem can be solved. □

To design an approximation algorithm, we introduce the *laminar jobs*. Laminar jobs at time t is composed of jobs $laminar(t) = \{j : t \in [r_j, d_j]\}$. The concept for our algorithm is to first decompose the original jobs into several small job sets, with jobs in the same set forming laminar jobs. These sets are then divided into two groups. Moreover, our decomposition and grouping ensure that jobs in different sets but the same group are never concurrent with each other. Thus this allows us to concentrate on the separated single set of laminar jobs. The decomposition step loops as follows:

(1) Set $t = d_1$. Let $S_1 = laminar(t)$.

(2) Assume that j to be the first job in the remaining jobs. Let $t = d_j$ and $S_2 = laminar(t)$.

(3) Update j to be the first job in the remaining jobs and repeat to find all sets $S_3, S_4, \ldots, S_b$ until no jobs are left.

For the grouping step, we group the sets with odd index into group $G_1 = \{S_1, S_3, S_5, \ldots, \}$ while the others form group G_2. Note that both the decomposition and grouping can be operated in online manner since we know both the release times and deadlines of the jobs when they arrive. An offline 2-approximation algorithm can be easily obtained.

Lemma 1. *The DSA problem for agreeable jobs has a 2-approximation algorithm.*

Proof. An important observation of such a grouping strategy is that the sets in the same group are pairwise non-concurrent at any time. Take the first three sets S_1, S_2, S_3 for example. Let j be the first job in S_2. Let t_a be the largest deadline of jobs in S_1 and t_b be the smallest arrival time of jobs in S_3. We note that $d_j \geq t_a$ according to the decomposition step (2). Moreover, the first job in S_3 has release time larger than d_j because otherwise this job would be grouped into S_2, hence $t_b > d_j$. Therefore we have $t_a < t_b$ and S_1, S_3 are not concurrent. Similarly this can be extended to every two sets in the same group.

Getting this, for all sets in G_1 (or G_2), we can solve each set optimally by FirstFit policy since laminar jobs is crossing at least one common time. The jobs in group G_1 (or G_2) can also be solved optimally due to the pairwise independence property of the sets. Finally, by applying $OrderedFit(\cdot)$ to $G_1 \cup G_2$ and Lemma 2, the new algorithm framework uses memory at most $2OPT(\mathcal{J})$. □

Algorithm 1 OrderedFit(STR,$\mathcal{J}$)

for each round $r = 1, 2, \ldots, k$ **do**

1. Simulate $STR(G_r)$ in virtual device r. Denote the maximum occupied address on device r to be M_r.

2. Assign all jobs in the practical memory as if they are combined by the allocations in k virtual devices. Instead of starting from address 1 on the virtual device, device r ($r \geq 2$) starts its allocation from address $\sum_{i=2}^{r} M_{i-1} + 1$ in the practical single memory.

end for

Lemma 2 (*Technical Lemma*)**.** *If a job set $\mathcal{J}$ is decomposed into k disjoint subsets $G_1, \ldots, G_k$, and every set G_i can be solved within $r \cdot OPT(G_i)$ by some strategy STR (i.e for each i we have $STR(G_i) \leq r \cdot OPT(G_i)$), then by applying OrderedFit to $\mathcal{J}$ we have $OrderedFit(STR, \mathcal{J}) \leq rk \cdot OPT(\mathcal{J})$.*

Note that our new algorithm matches the current best algorithm in [15]. However, our decomposition step not only simplifies the structure of the jobs, but will also be proved more extendable to online single-device and multi-device setting.

4.2. Better online algorithm through laminar decomposition

Online algorithm: We note that our decomposition step can be performed in an online manner. We will show how to achieve the 3-competitive clairvoyant online algorithm by using laminar decomposition. This improves the 8-competitive algorithms in [15].

Our strategy is to assign the second decomposed group pessimistically to ensure that we waste at most one gap in the memory, which is shown in Algorithm 2. Let S_i be generated by laminar jobs at time t_i, i.e. $S_i = laminar(t_i)$. We have $LOAD(S_i) \leq LOAD(t_i)$ by the decomposition step.

Algorithm 2 PessiOnline

1. Jobs are decomposed into two groups $G_1 = \{S_1, S_3, S_5, \ldots\}, G_2 = \{S_2, S_4, S_6, \ldots\}$ online by laminar property. For all jobs in the same set, the job that is earlier released will be allocated to lower address (ties are broken by assigning jobs with earlier deadline to lower address). We apply different strategies for the two groups as follows.

for each round $i = 1, 2, 3, \ldots$ **do**

2. For set S_{2i-1} in G_1, upon arrival of the jobs, it will be placed on the lowest feasible address (FirstFit). Let the maximum address used by this set be M_{2i-1}.
3. For set S_{2i} in G_2, we pessimistically assign the first one to address $M_{2i-1} + 1$. All the later jobs in the same set will be assigned one by one to the address immediately (consecutively) after the prior one, without violating the rule in Step 1.

end for

Theorem 3. *There is a* 3*-competitive algorithm for the single-device DSA problem where jobs have agreeable deadlines.*

Proof. Obviously the occupied space is maximized at some time t_i. The analysis is based on discussing the possible cases of placing job set S_{2k-1} where $k = 1, 2, \ldots$. Three observations are the key advantages of such an allocation strategy. First, when we consider jobs in S_{2k-1} (determine the allocation upon the arrival of a job), the determination should never worry about the possible conflict with S_{2k-3}, since these two sets are separated by time t_{2k-2} and thus never concurrent. Second, because of the pessimistic allocation in Step 3 and the rule in Step 1, the jobs in S_{2k-2} is consecutively allocated and only leave at most one gap (it can only be the empty space starting from address 1) at every time in interval $[t_{2k-2}, t_{2k-1}]$ (before jobs S_{2k-1} are released). Third, because we apply the FirstFit policy to set S_{2k-1} and also the two former observations, the jobs in S_{2k-1} can be placed in at most two contiguous memory spaces (namely, divided by a gap of empty space). For example, S_3, S_7 are assigned to two contiguous memory spaces, while S_5 is assigned to one contiguous memory space. Furthermore, the gap is generated only because there is a job with larger size being released. For the case that S_{2k-1} is separately allocated, the gap is created because the space of the gap (let the size be s_g) is less than the job (let its size be s_j) attempting to fit into this space. Namely, we have $s_g < s_j < LOAD(t_{2k-1})$. The occupied space at time t_{2k-1} is at most $LOAD(t_{2k-1}) + s_g \leq 2LOAD(t_{2k-1}) \leq 2LOAD(\mathcal{J}) \leq 2OPT(\mathcal{J})$. Since we assign the later set S_{2k} pessimistically, the space used at time t_{2k} is at most $LOAD(t_{2k-1}) + s_g + LOAD(t_{2k}) \leq 3OPT(\mathcal{J})$. For the case that S_{2k-1} is not separately allocated, the occupied space by assigning S_{2k-1} at time t_{2k-1} is $LOAD(S_{2k-1}) \leq LOAD(t_{2k-1})$. S_{2k} will not conflict with S_{2k-2} and is then consecutively allocated after S_{2k-1}, thus at time t_{2k} the occupied space is at most $LOAD(t_{2k-1}) + LOAD(t_{2k}) \leq 2LOAD(\mathcal{J}) \leq 2OPT(\mathcal{J})$. Combining these two cases, the algorithm is 3-competitive. □

Lower bounds on online algorithms: For jobs with agreeable deadlines, we show the following lower bounds of an online algorithm.

Theorem 4. *Any non-clairvoyant online algorithm for the DSA problem with agreeable jobs cannot be* $(2 - \epsilon)$*-competitive. Any online clairvoyant algorithm for the DSA problem with agreeable deadlines cannot be* $(1.54 - \epsilon)$*-competitive.*

Proof. We construct instances showing that no matter what the online algorithm ONLINE (non-clairvoyant and clairvoyant scheduling) determines, the adversary ADV can make its performance bad. In the non-clairvoyant scheduling, ONLINE does not know the job's deadline when a job is released, while in the clairvoyant scheduling it knows this information.

In the non-clairvoyant scheduling, let $\epsilon_1 \to 0$ and p be the minimum value larger than 1 that makes $n = \frac{2p}{\epsilon_1} - 1$ an integer. Assume that ADV releases n jobs $1, 2, \ldots, n$ at time 1. Each of these jobs has equivalent size 1. Since it is non-clairvoyant, the online algorithm does not know the deadline until the job is finished. Obviously, no matter how ONLINE assigns these jobs, the occupied size at time 1 is at least $n \cdot 1$ and all jobs should be assigned a position at time 1. Let job j be assigned on the top with largest memory address a_j by ONLINE. We have $a_j > n \cdot 1 - 1$. For ADV, all the jobs except j are finished at some time $t > 1$. Moreover, ADV releases a job $n + 1$ with size $M = a_j$ that has the same deadline with j. Since j is still alive at time t, ONLINE has to assign this job to $a_{n+1} > a_j$, and thus has solution at least $2M$. Clearly ADV can schedule these jobs with maximum occupied address $M + 1$. Therefore, the competitive ratio of any online algorithm is at least $\frac{2M}{M+1} \geq 2 - \frac{2}{n+1} = 2 - \frac{\epsilon_1}{p} \approx 2$.

Now we turn to the clairvoyant scheduling. Assume that $l, \epsilon_1, \epsilon_2$ are constant integers and $l \gg \epsilon_1, l \gg \epsilon_2$. Let the first job with size l be released at time 0 and finished at time 2. We will introduce constants u, v, v' and set their values later. W.l.o.g assume that algorithm ONLINE allocates it at address x. If $x \geq ul$, then the adversary ADV terminates. The competitive ratio

of this case is at least $\frac{x+l-1}{l} \geq 1 + \frac{ul-1}{l}$ which is approximately $1+u$ by setting l to be a large integer. If $x < ul$, then ADV releases the second job at time 1 with size $x+\epsilon_1$ and deadline 4. Obviously, ONLINE has to assign job 2 with address at least $x+l$. Let this job be allocated at address $x+l+y$. If $y \geq vl - x$, then ADV terminates. The competitive ratio of this case is at least $\frac{x+y+l+x+\epsilon_1-1}{l+x+\epsilon_1}$. This value is at least $1 + \frac{x+y-1}{x+l+\epsilon_1} > 1 + \frac{v-1/l}{1+u+\epsilon_1/l} \approx 1 + \frac{v}{1+u}$. If $y < vl - x$, then ADV will release the third job at time 3 with size $x+l+y+\epsilon_2$ and deadline 4. ONLINE has to allocate this job with address at least $l+y+2x+\epsilon_1$. Thus the competitive ratio is at least $\frac{l+y+2x+\epsilon_1+(x+l+y+\epsilon_2)-1}{x+l+y+\epsilon_2+(x+\epsilon_1)} = 1 + \frac{x+l+y-1}{2x+y+l+\epsilon_1+\epsilon_2}$. Since $x < ul$, $x+y < vl$, this value is at least $1 + \frac{x+l+y-1}{ul+l+vl+\epsilon_1+\epsilon_2}$. Furthermore, by setting $v' < v$, the value is larger than $1 + \frac{1+v'}{1+u+v}$ when $x+y > v'l$, and at least $1 + \frac{1}{1+u+v'}$ when $x+y \leq v'l$. Therefore, any online clairvoyant algorithm is at least $\min\{1+u, 1+\frac{v}{1+u}, 1+\frac{1+v'}{u+1+v}, 1+\frac{1}{1+u+v'}\}$-competitive. Setting $u = 0.54$, $v = 0.832$, $v' = 0.281$, we obtain lower bound 1.54-competitive. Our construction uses at most three jobs that have agreeable deadlines. Clearly the case with two jobs can be solved optimally by using FirstFit strategy, while releasing one more job can greatly deteriorate the performance of online algorithms. □

Extending to general jobs: [15] gave a $16\lceil log(L)\rceil$-competitive online algorithm for general jobs by calling/extending the algorithm for agreeable jobs. We note that the applications in an SRF problem (register) usually have small value L, thus we will further improve the performance for this situation (accurately $L \leq 112$) in the following. The idea is to extend the concept used in Section 4.2.

Theorem 5. *For general jobs, there is a* $\min\{16\lceil \log L\rceil, 2\lceil \frac{L}{2}\rceil\}$*-competitive algorithm with known maximum duration L.*

5. Multi-device Balancing DSA problem

In this section, we settle the multi-device Balancing DSA problem for both agreeable jobs and general jobs. In this setting, there are m devices and the objective is to balance the usage of the memories. Accurately, we aim at minimizing the maximum memory size used in the m memories. We observed that [14] studied this problem under the assumption that "$m \leq \frac{\omega^*}{S_{max}}$". With their assumption, [14] showed that their algorithm is online optimal competitive. We restudy this problem by dropping their assumption. For the multi-device DSA problem, we give a 4-competitive algorithm for jobs with agreeable deadlines and an $O(\log L)$-competitive algorithm for general jobs. The method of the online algorithm is still based on the laminar decomposition.

We also start by designing a $O(1)$-competitive algorithm for jobs with agreeable deadlines.

5.1. Balancing DSA problem for jobs with agreeable deadlines

Assume that we decompose the jobs $\mathcal{J}$ to $G_1 = \{S_1, S_3, S_5, \ldots\}$, $G_2 = \{S_2, S_4, S_6, \ldots\}$ where $S_i = laminar(t_i)$ through laminar decomposition in an online manner as in Section 4.2. The new algorithm for multi-device is shown in Algorithm 3.

Algorithm 3 MultiOnline

1. Jobs are decomposed into two groups $G_1 = \{S_1, S_3, S_5, \ldots\}$, $G_2 = \{S_2, S_4, S_6, \ldots\}$ online by laminar property. We apply different strategies for the two groups. For all jobs in the same set, the job that is earlier released will be allocated to lower address (ties are broken by assigning jobs with earlier deadline to lower address). In the following, by *top address* at time t we denote the maximum occupied address at time t that is allocated in the current round.

for each round $i = 1, 2, 3, \ldots$ **do**

2. For set S_{2i-1} in G_1, upon arrival of the jobs j, we simulate on every device by assigning j to the lowest feasible address (FirstFit policy). If j can be allocated to the empty space that is below the top address of the device l, then we assign j to l. If not, then we assign j to the device with minimum top address. Assign j to the selected device by FirstFit policy.

3. For set S_{2i} in G_2, we assign the jobs pessimistically. Upon the arrival of job $j \in S_{2i}$, we suppose that l is the current device that has the minimum top address (let the address be M_l). Assign j to address $M_l + 1$. All the later jobs in the same set that is allocated to l will be assigned consecutively without violating the rule in Step 1.

end for

Theorem 6. *Algorithm MultiOnline is* 4*-competitive for multi-device Balancing DSA problem on jobs with agreeable deadlines.*

Proof. Through laminar decomposition, one advantage is that the load of set of laminar jobs S_i at time t_i is exactly the summation of sizes over all its jobs. We first note that the lower bounds for the multi-device Balancing DSA problem are $OPT \geq \frac{LOAD(\mathcal{J})}{m}$ and $OPT \geq S_{max}$. Observing this, our allocation strategy in Algorithm 3 tries to balance the allocation for jobs in the same set S_i. Assume that the minimum (maximum) occupied address in the m devices is M_{min} (M_{max}), the allocation strategy in Step 2 and Step 3 ensures that $M_{max} - M_{min} \leq S_{max}$. First, when allocating S_1, the load at time t_1 is allocated in a balanced manner to the m devices. The maximum address used in the m devices is at most $\frac{LOAD(t_1)}{m} + S_{max}$ since the minimum address used in the m devices is at most $\frac{LOAD(t_1)}{m}$ and the difference of every two devices is at most S_{max}. When we allocate S_2, the jobs are pessimistically assigned to the current top address on the selected device (which is the device

that has the minimum occupied address). The total space occupied in t_2 is exactly $LOAD(S_1) + LOAD(S_2)$ which is at most $LOAD(t_1) + LOAD(t_2)$ since assigning S_1 did not generate any empty space at time t_1. Because we assign S_2 in a balanced way, the device with maximum occupied address at time t_2 is at most $\frac{LOAD(t_1)+LOAD(t_2)}{m} + S_{max}$. Note that the optimal solution OPT uses at least $max\{\frac{LOAD(t_1)}{m}, \frac{LOAD(t_2)}{m}, S_{max}\}$. Thus the occupied space at time t_2 is at most $3 \cdot OPT(\mathcal{J})$.

We will prove that the occupied address is at most $3 \cdot OPT(\mathcal{J})$ at time t_{2i-1} and at most $4 \cdot OPT(\mathcal{J})$ at time t_{2i}. As in the proof of Theorem 3, we discuss the possible cases of allocation for set S_{2i-1} where $i \geq 2$. Note that we allocate set S_{2i-1} by FirstFit for selected device. Consider the case that the current job has size larger than the remaining empty space that is below the selected device's top address. The allocation creates at most one gap (empty space) that has size less than S_{max}. This property holds because the strategy on a single device is the same as what we used in Algorithm 2. This job will be allocated to the top address of the selected device.

We start by the simple case that every device is at least assigned one job of S_{2i-1} to its top address (denote such a device as *saturated-device*). That is, each gap generated on the m devices is at most S_{max}. After assigning S_{2i-1}, the total occupied space in these devices at time t_{2i-1} is at most $LOAD(t_{2i-1}) + m \cdot S_{max}$. In this case, we have $M_{min} \leq \frac{LOAD(t_{2i-1})}{m} + S_{max}$. Thus $M_{max} \leq M_{min} + S_{max} \leq 3 \cdot OPT(\mathcal{J})$. When we assign S_{2i} later, we need not worry about the possible conflict with S_{2i-2}. Due to the pessimistic strategy in Step 3, the jobs in S_{2i} are assigned to the top address consecutively and no space is wasted. Thus after assigning S_{2i}, the total occupied space in these devices at time t_{2i} is at most $LOAD(t_{2i-1}) + LOAD(t_{2i}) + m \cdot S_{max}$. Hence, in this case we have $M_{max} \leq M_{min} + S_{max} \leq \frac{LOAD(t_{2i-1})+LOAD(t_{2i})}{m} + S_{max} + S_{max} \leq 4 \cdot OPT(\mathcal{J})$.

We say a device is *saturated* (or *unsaturated*) if the current job is (or not) infeasible to be assigned to the empty space that is below the top address of the selected device at Step 2. Now we consider the case that some devices are *saturated* while others are not when assigning S_{2i-1}. The existence of saturated device implies that the current job is infeasible to be allocated to the empty space on the unsaturated devices according to Step 2. Thus the empty space both on the unsaturated devices and the saturated devices has size less than S_{max}. The total occupied space at time t_{2i-1} is still $LOAD(t_{2i-1}) + m \cdot S_{max}$. Thus $M_{max} \leq \frac{LOAD(t_{2i-1})}{m} + 2S_{max} \leq 3OPT(\mathcal{J})$. Similarly at time t_{2i}, we have $M_{max} \leq \frac{LOAD(t_{2i-1})+LOAD(t_{2i})}{m} + 2S_{max} \leq 4OPT(\mathcal{J})$.

It remains to consider the case that all devices are unsaturated after assigning S_{2i-1}. All the jobs of S_{2i-1} are assigned below their top addresses of the selected devices. Note that the occupied top address for these unsaturated-devices is generated at time t_{2i-2}. This value can be bounded by $4OPT(\mathcal{J})$ by the analysis above. Thus the maximum occupied space at time t_{2i-1} is at most $4 \cdot OPT(\mathcal{J})$ in this situation. Now consider the assignment for jobs in S_{2i}. All jobs in S_{2i-2} have departed at this time by the independence property of laminar decomposition. Thus we need not worry about the possible conflict between S_{2i} and S_{2i-2}, since every load is assigned from address 0 consecutively at time t_{2i-1}. The total occupied space at time t_{2i} will be $LOAD(S_{2i-1}) + LOAD(S_{2i})$ which is at most $LOAD(t_{2i-1}) + LOAD(t_{2i})$. Thus we have $M_{max} \leq \frac{LOAD(t_{2i-1})+LOAD(t_{2i})}{m} + S_{max} \leq 3 \cdot OPT(\mathcal{J})$ at time t_{2i} in this situation. Therefore, the competitive ratio of Algorithm 3 is at most 4. □

5.2. Extending to general jobs

Now we are ready for the competitive algorithm on multi-devices for general jobs. The extension will follow some extending procedure for single-devices in [15] with further modifications. We will derive a $O(logL)$-competitive algorithm without aiming at reducing its constant factor. Every job with length $(2^{i-1}, 2^i]$ is rounded to length 2^i. Then the rounded jobs with length 2^i can be grouped into a set of type i. We will make a loss of factor 2 in this step. All the jobs in the same group has the nice property that they have agreeable deadlines. Let $\mathcal{J}'$ be all the jobs that are already released at current time t and correspondingly S'_{max} be the largest job size until time t. Let lower bound $LB(t) = max\{\frac{LOAD(\mathcal{J}')}{m}, S'_{max}\}$. According to the proof in Theorem 6, we only need to assign a slot with size $4 \cdot LB(t)$ for the set of type i which ensures the feasibility to allocate jobs $\mathcal{J}'$. There are $O(\log L)$ such sets of different types. We only need to open a new slot (starting from the maximum occupied address thus far) if the arrival job cannot be matched to a slot for its type. Note that the value $LB(t)$ will change over time, thus we need to update the slot size as follows. For the current value $LB(t)$, we allocate a slot with size $8LB(t)$ for each type i. When this value (the lower bound over time $LB(t)$) is doubled due to later released jobs, we then double the slot size that is newly opened for jobs of type i. The doubling procedure will ensure that the total size of slot that is allocated is $O(\log L \cdot LB(t))$. Thus finally the competitive ratio is $O(\log L)$.

Theorem 7. *There is an $O(\log L)$-competitive algorithm for multi-device Balancing DSA problem for general jobs.*

6. Conclusion

In this paper, we improve the algorithms for DSA problem(s) by introducing the online laminar decomposition. It not only simplified the analysis, but also proved more extendable for both single-device and multi-device setting.

Acknowledgements

The work described in this paper was partially supported by grants from City University of Hong Kong (Project No. 7002584 and 7002611) and National Major Special Science & Technology Projects (Grant No. 2011ZX04016-071), and Research Fund for the Doctoral Program of Higher Education of China (20093402110017, 20113402110024).

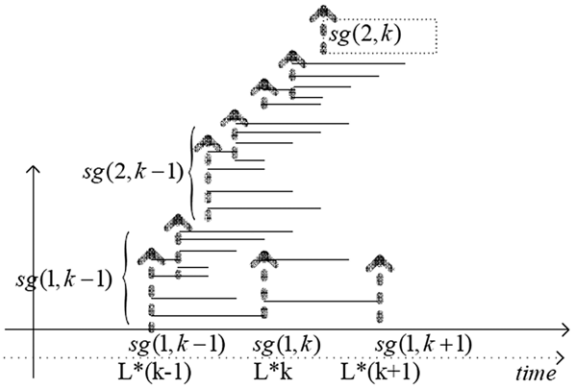

Fig. 3. Extending the pessimistic strategy of allocation.

Appendix

Proof of Theorem 5: Improved Algorithm for $L \leq 112$ Proof. Define $r(l, k)$ to be all the jobs that arrive at time $kL+l$ where $1 \leq l \leq L$ and $0 \leq L \leq K-1$. This definition can decompose the original jobs into L groups in an online manner (for fixed l, each group is composed of K disjoint sets). We note that directly extending the concept of assigning jobs pessimistically in Section 4.2 can only obtain competitive ratio $2L-1$. This is because we need to pessimistically allocate the $L-1$ groups and thus may lead to $L-1$ gaps (instead of 1 gap in Section 4.2) when we allocate the future groups, which guarantees a competitive ratio $L+L-1$ (instead of ratio $2+1$). To derive the $2\lceil \frac{L}{2} \rceil$ competitive algorithm, we should incorporate other ideas to allocate the jobs carefully. The allocation procedure is described in Algorithm 4.

Algorithm 4 PessiALG

1. Jobs are decomposed into $r(l, k)$ in an online manner.
2. For fixed k, group every two sets of $r(l, k)$ as follows: $r(1, k)$ and $r(2, k)$ form $sg(1, k)$; $r(3, k)$ and $r(4, k)$ form $sg(2, k)$; repeat the same procedure, we get $\lceil \frac{L}{2} \rceil$ such groups. For every $sg(w, k)$ with fixed k, we apply different strategies to its two sub-sets. For jobs in $r(2w-1, k)$, we assign jobs with larger deadline to lower address. While for jobs in $r(2w, k)$, we assign jobs with earlier deadline to lower address.

for upon arrival of the jobs **do**

3. For jobs in $sg(1, k)$, all the later jobs in the same group will be assigned one by one to the address immediately (consecutively) after the prior one (the job is allocated in the upper available address if its size is larger than the empty space), without violating the rule in Step 2. Let the maximum address used by this set be $M_{1,k}$.
4. For jobs in $sg(w, k)$ where $2 \leq w \leq \lceil \frac{L}{2} \rceil$, we pessimistically assign the first one to address $M_{w-1,k}+1$. All the later jobs in the same group will be assigned one by one to the address immediately (consecutively) after the prior one, without violating the rule in Step 2.

end for

For fixed k, there are L sub-sets which are possibly concurrent. Let *super_group* $sg(w, k)$ be $r(2w-1, k) \cup r(2w, k)$. In Step 2, we obtain $\lceil \frac{L}{2} \rceil$ such groups. We apply different strategies to the two sub-sets in $sg(w, k)$ for fixed w, k. While Step 3 and Step 4 apply different strategies to $sg(w, k)$ where $1 \leq w \leq \lceil \frac{L}{2} \rceil$ for fixed k. The analysis is also based on the discussion of possible number of gaps and different allocation manners for $sg(1, k)$. Fig. 3 demonstrates the allocation for the case $L = 4$. We first note that our strategy in Step 2 ensures that there is no empty space (gap) left at time $kL + 2w$ between the two sets $r(2w-1, k)$ and $r(2w, k)$ that form $sg(w, k)$. Thus the space used by each super_group $sg(w, k)$ is at most 1 times the optimal solution. Moreover, every super_group (with the same value k) may create at most one gap after time $Lk + 2w$. When we consider $sg(1, k)$, we need not worry about the possible conflict with $r(1, k-1)$. When we allocate $sg(1, k)$, this group may be separately allocated due to the $\lceil \frac{L}{2} \rceil$ gaps created by the former $sg(w, k-1)$ where $1 \leq w \leq \lceil \frac{L}{2} \rceil$. Thus at most $\lceil \frac{L}{2} \rceil$ gaps are left at time $Lk+1$ or $Lk+2$ when assigning $sg(1, k)$. Note that every gap is at most S_{max} because the gap is generated when the arrival job is infeasible to be allocated in the empty space. Thus the allocation of group $sg(1, k)$ uses memory size at most $\lceil \frac{L}{2} \rceil + 1$ times the optimal solution. Furthermore, since we pessimistically allocate the remaining $sg(w, k)$ where $2 \leq w \leq \lceil \frac{L}{2} \rceil$, we use memory size at most $\lceil \frac{L}{2} \rceil + 1 + \lceil \frac{L}{2} \rceil - 1$ times the optimal solution. Finally, the algorithm is $2\lceil \frac{L}{2} \rceil$-competitive. □

References

[1] Preston Briggs, Keith D. Cooper, Linda Torczon, Improvements to graph coloring register allocation, ACM Transactions on Programming Languages and Systems 16 (3) (1994) 428–455.
[2] A.L. Buchsbaum, H. Karloff, C. Kenyon, N. Reingold, M. Thorup, OPT versus LOAD in dynamic storage allocation, in: Proceedioings of the 35th Annual ACM Symposium on Therory of Computing, STOC, 2003, pp. 556–564.
[3] Adam L. Buchsbaum, Alon Efrat, Shaili Jain, Suresh Venkatasubramanian, Restricted strip covering and the sensor cover problem, the full version is at http://arxiv.org/PS_cache/cs/pdf/0605/0605102v1.pdf. The conference version appears in: Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms, SODA, 2007, pp. 1056–1063.
[4] G.J. Chaitin, Register allocation & spilling via graph coloring, in: Proceedings of the SIGPLAN Symposium on Compiler Construction, ACM Press, 1982, pp. 98–105.
[5] Fred C. Chow, John L. Hennessy, The priority-based coloring approach to register allocation, ACM Transactions on Programming Languages and Systems 12 (4) (1990) 501–536.
[6] Lal George, Andrew W. Appel, Iterated register coalescing, ACM Transactions on Programming Languages and Systems 18 (3) (1996) 300–324.
[7] J. Gergov, Approximation algorithms for dynamic storage allocation, in: Proceedings of the 4th European Symposium on Algorithms: Lecture Notes in Computer Science 1136, ESA, 1996, pp. 52–61.
[8] J. Gergov, Algorithms for compile-time memory optimization, in: Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms, SODA, 1999, pp. S907–S908.
[9] Bala Kalyanasundaram, Kirk R. Pruhs, Dynamic spectrum allocation: the impotency of duration notification, in: Procedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science, FST TCS 2000, in: LNCS, vol. 1974, 2000, pp. 421–428.
[10] H.A. Kierstead, The linearity of first-fit colorings of interval graphs, SIAM Journal on Discrete Mathematics 1 (4) (1988) 526–530.
[11] H.A. Kierstead, A polynomial time approximation algorithm for dynamic storage allocation, Discrete Mathematics 88 (1991) 231–237.
[12] D.E. Knuth, Foundamental Algorithms, Vol. 1, 2nd ed., Addison-Wesley, Reading, MA, 1973.
[13] Shuai Cheng Li, Hon Wai Leong, Steven K. Quek, New approximation algorithms for some dynamic storage allocation problems, in: Proceedings of the 10th Annual International Computing and Combinatorics Conference, 2004, pp. 339–348.
[14] Michael G. Luby, Joseph (seffi) Naor, Ariel Orda, Tight bounds for dynamic storage allocation, in: Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, 1994, pp. 724–732.
[15] Joseph (Seffi) Naor, Ariel Orda, Yael Petruschka, Dynamic storage allocation with known durations, in: Proceedings of the 5th Annual European Symposium on Algorithms, ESA, 1997, pp. 378–387. The journal version appreas in Discrete Applied Mathematics 100 (3), 2000, pp. 203–213.
[16] N.S. Narayanaswamy, Dynamic storage allocation and on-line colouring interval graphs, in: Proceedings of the 10th Annual International Computing and Combinatorics Conference, 2004, pp. 329–338.
[17] J.M. Robson, Worst case fragmentation of first-fit and best fit storage allocation strategies, Computer Journal 20 (1977) 242–244.
[18] Xuejun Yang, Li Wang, Jingling Xue, Yu Deng, Ying Zhang, Comparability graph coloring for optimizing utilization of stream register files in stream processors, in: Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP, 2009, pp. 111–120.