Scheduling Space-Sharing for Internet Advertising

Micah Adler* Phillip B. Gibbons† Yossi Matias‡

March 17, 1998

Abstract

This paper provides the first in-depth study of the algorithmic questions involved in the scheduling of space-sharing for Internet advertising. We consider the scheduling problem where each advertisement is specified by a geometry and a display frequency. Given a set of ads, a schedule of the ads specifies which ads are to be displayed at the same time. The objective is find a schedule of the ads such that (1) each ad is displayed with the correct frequency, (2) each ad is allocated enough space for the specified geometry, (3) all the ads to be displayed simultaneously can be arranged in the space available for advertising. We demonstrate that an optimal schedule can be determined by finding a solution to a new variant of the bin packing problem that we introduce. In this new variant, there are a number of copies of each item to be placed into the bins. In addition to the usual bin packing requirements, all copies of an item must be placed in different bins.

In this paper, we provide an efficient algorithm that finds the optimal solution to a restricted version of the new bin packing problem. This algorithm also provides a 2-approximation for the unrestricted case. We then apply this approximation to the problem of scheduling space-sharing for Internet advertising. We obtain a 2-approximation to the problem of minimizing the space requirements of a given set of ads, as well as to the problem of determining the best subset of the ads that can be scheduled with a given space constraint. We also consider an on-line version of the ad scheduling problem, where requests to purchase ad space arrive sequentially and after each request arrives, a decision must be made whether or not to sell the requested space without knowledge of future requests. We provide an algorithm that makes these decisions in a manner that is optimal.

*Department of Computer Science, University of Toronto, 10 King's College Road, Toronto ON M5S3G4, Canada. micah@cs.toronto.edu. Work done while visiting Bell Laboratories.

†Bell Laboratories, Lucent Technologies, Room 2D-148, 600 Mountain Avenue, Murray Hill NJ 07974. gibbons@research.bell-labs.com.

‡Department of Computer Science, Tel-Aviv University, Tel-Aviv, 69978, Israel; and Bell Labs, Murray Hill, NJ, USA. matias@math.tau.ac.il.

1 Introduction

Advertisers have recently become aware of the power of the Internet, and this has led to an enormous increase in World Wide Web sites that provide space for advertising. An important aspect of Internet advertising is *space-sharing*: using the same advertising space to display several different ads. This can be done by displaying different ads to different users, by periodically updating what is displayed to a given user, or by displaying a set of small ads that together use the available space. These techniques allow advertisers to purchase small portions of expensive spaces increasing the demand for advertising, and they allow users to be shown a greater variety of ads, increasing the overall advertising effectiveness [Dou98].

Space-sharing also allows advertisers to target specific subsets of the audience, and in fact much of the existing software for scheduling Internet advertisements focuses on audience targeting [Dou98, Net98, W3.98]. Although audience targeting is an important consideration, the concentration on this one aspect of space-sharing has left many other scheduling problems unresolved. For example, major players in the Internet advertising business will not sell space for an ad unless that ad has a single, pre-specified shape, presumably because of the difficulty in scheduling ads of different shapes. Another example is that, to our knowledge, no existing software allows the space provider to split the ad space into two or more "ad windows" that display different ads. In this paper, we address generalizations of these unresolved scheduling problems, and provide algorithms that greatly increasing the flexibility of Internet advertising.

We consider a scenario where the advertisements are specified in terms of three parameters. The *access fraction* of an ad i, denoted by $U(i)$, represents the fraction of accesses by users to the provider (e.g., the fraction of hits to a web page) that see the ad at some point during that access. The *time fraction* $F(i)$ of an ad i represents the fraction of the time that the provider displays the ad to any access that sees the ad. The third parameter we consider is the ad *geometry*. We assume that the amount of space allocated to advertising by a provider is fixed. Advertisements can also specify one or more target *market segments*, designating the target audience for the ad; the algorithms we present can be applied to each market segment independently, and hence the market segment parameter will not be considered explicitly in this paper.

We consider both off-line and on-line scheduling of ads specified by these three parameters. In the off-line problem, the entire set of ads to be scheduled is known in advance, and we are required to produce a schedule where all the ads to be shown to any given access at any given time can be displayed in the space allocated to advertising. In some cases, there is no valid schedule of a set of ads, and thus we also consider the problem of finding the optimal subset of the ads that does have a valid schedule. The optimal subset is defined to be the subset that produces the highest revenues for the space provider. Both off-line algorithms have been implemented, and a Java demonstration is available at http://www.bell-labs.com/project/collager.

In the on-line scheduling problem, requests to purchase ad space arrive sequentially, and as each request arrives, a decision must be made whether or not to sell the requested space without knowledge of future requests. We require that there exists a valid schedule containing every ad that has been accepted, and decisions on whether or not to accept an ad are irrevocable. However, the schedule containing the accepted ads can be changed at any time, provided that the resulting schedule is valid. This problem is motivated by the demand for fast response with most Internet systems. Providing advertisers an immediate decision on requests to purchase advertising space requires space providers to respond before future requests arrive. Decisions to accept or reject a request for ad space are difficult to change, but changing the specific schedule used poses no such difficulties.

1.1 New Techniques

In order to schedule ads specified by the above three parameters, we introduce a variant of the bin packing problem, which we call the *ad placement problem*: given T slots (equivalent to the bins of a bin packing problem), a slot size S, and a set of ads A, where each ad $i \in A$ consists of a size $s_i \leq S$ and a weight $w_i \leq T$, assign the ads to slots, where ad i is assigned exactly once to each of w_i slots. Let $P(j)$ be the set of ads that are assigned to slot j, and let the *fullness* of slot j be $|P(j)| = \sum_{i \in P(j)} s_i$. We say that an ad assignment is *valid* if $\max_j |P(j)| \leq S$. The goal is to efficiently find a valid assignment, if one exists. Figure 1 depicts an example instance and figure 2 depicts a valid solution.

The aspect of the ad placement problem that distinguishes it from the traditional bin packing problem, as well as related scheduling problems, is that at most one copy of an ad i is allowed to be assigned to any single slot. We view this restriction as a necessary one for Internet advertising: it is unacceptable for a space provider to satisfy an ad specification by displaying multiple copies of the same ad at the same time to the same user. For example, if an advertiser pays for an ad of size $S/2$ to be shown for half the users, it is unacceptable to display two copies of the ad side-by-side to a quarter of the users.

The ad placement problem is a fundamental bin packing problem, and we anticipate that it has applications outside the area of ad scheduling. In section 4, we demonstrate several uses of the ad placement problem for scheduling Internet advertising. For concreteness, we describe one of these uses here. We can schedule a set of ads that are each specified by an access fraction $U(i)$ and a one dimensional geometry $L(i)$, where all ads have a time fraction of 1. To do this, we set T to be the least common multiple of the denominators of the $U(i)$.[1] We set $w_i = T \cdot U(i)$, $s_i = L(i)$, and S to the total size of the space available for advertising. If we can find a valid solution to the ad placement problem, we use that solution by displaying to a given access the ads assigned to one of the T slots. This guarantees that all the ads scheduled for simultaneous display fit into the available space, while at the same time ensuring that each ad is shown to the required fraction of accesses. The choice of which slot to display to a given access can be made several different ways: we can cycle through either a deterministic or random permutation of the T slots, or to reduce bookkeeping at the expense of losing the deterministic guarantee that the access fractions are adhered to, we can display a random slot to each access.

We show that any solution to the ad placement problem can also be used directly to schedule ads in the other two scenarios where each ad is specified by two parameters. We also show that solutions to the ad placement problem can be used, with some restrictions, to schedule ads where the geometry is specified by more than 1 dimension. However, even scheduling ads with a 1-dimensional geometry has many uses. For example, this can be used when the space allocated to advertising is a horizontal bar, and the width of the ads vary but the ads all have a height equal to the advertising space. We also show that, with some restrictions, we can use solutions to the ad placement problem to find a schedule for the *Full Assignment Problem*, where each ad is specified in terms of an access fraction, a time fraction, as well as a geometry of any number of dimensions.

[1] We assume that the access fractions are all rational.

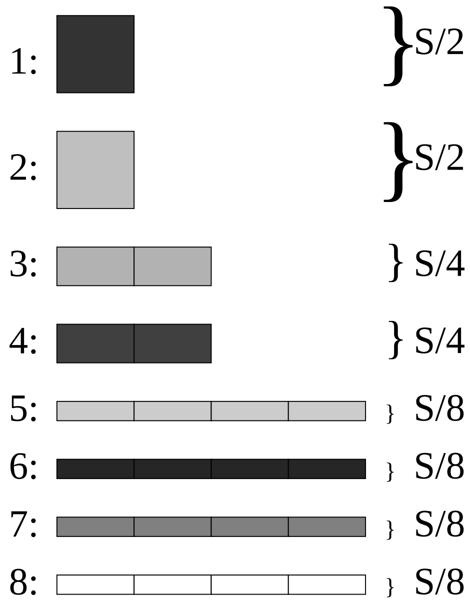

Figure 1: An example of an ad placement problem. The number of blocks represents the ad weight; the height of the blocks represents the ad size.

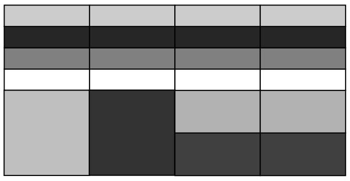

Figure 2: A valid solution for $T = 4$.

1.2 New Results

A reduction from Bin Packing to the ad placement problem (with $\forall i : w_i = 1$) demonstrates that it is NP-Hard to decide whether or not there exists a valid assignment using all the requested ads. However, this is not the case when the ad sizes are *divisible*: when the ad sizes form a sequence $S = p_1 > p_2 > p_3 \ldots$, such that for all k, p_k is an integer multiple of p_{k+1}. Given any set A of n ads with divisible sizes, we provide an algorithm that, in time $O(\sum_{i=1}^{n} w_i + sort(A))$, finds a valid assignment of ads to slots, if such an assignment exists. Here, $sort(A)$ is the time required to sort the n ads by size, and because of the divisibility assumption, this allows for use of standard integer sorting techniques. Note that a lower bound of $\Omega(\sum_{i=1}^{n} w_i)$ exists for any algorithm that outputs each assignment of an ad to a slot. This algorithm also provides us with a 2-approximation for the case where the ad sizes are arbitrary real numbers.

We also show that for the case where there does not exist a valid assignment of the entire set of ads, finding the optimal subset of the ads which does have a valid assignment is NP-Hard. This holds even when the ad sizes are required to be divisible. However, we provide an algorithm that, given a set of ads A with divisible sizes, finds a subset such that the percentage of ad space utilized is within a factor of 2 of the optimal subset of the ads. This algorithm also requires time $O(\sum_{i=1}^{n} w_i + sort(A))$.

For the on-line problem, we show that any on-line algorithm performs very poorly if the only restriction on ad sizes and weights is that the ad sizes must be divisible. Thus, the space provider must further restrict the allowed choices. We here show that if the ad sizes are divisible, Z is the maximum allowed ad size, and V is the maximum allowed ad weight, where $ZV < \frac{ST}{2}$, then we can efficiently (and deterministically) make the required on-line decisions so that the percentage of ad space utilized is within a factor of $\frac{ST}{ST - Z(2V-1)}$ of the optimal off-line choices for every sequence of requests. We also show that this is the best possible for any on-line algorithm, even when randomness is used.

1.3 Previous work

The power of using divisible sizes in a related context has been demonstrated in [CGJ87], which considers variations on the bin packing problem using the same assumption. The ad placement problem can in fact be described as a bin packing problem where, in addition to the standard notion of the size of an item s_i, every item also has a weight w_i, and a copy of item i must be placed in exactly w_i distinct bins. To our knowledge, bin packing with this additional requirement has not been previously considered. The ad placement problem can also be compared with the 2-D bin packing problem [BCR80, CGJT80, BS83], in which rectangles must be placed into a strip of fixed width, where the objective is to minimize the height of the strip. In the ad placement problem, if each ad i is viewed as a rectangle of height s_i and width w_i, then the two problems are identical. However, this is an overly restrictive version of the ad placement problem. For example, consider the problem of placing three ads, each of weight 2 and size 1, into 3 slots. There exists a placement of these ads such that each time slot contains 2 adds, but for the corresponding 2-D bin packing problem, the optimal placement has height 3.

Many authors have considered the problem of scheduling parallel jobs on a set of processors; see, e.g., [CM96, GG75, FKST93, HSW96, LT94, SWW91, TSWY94, TWY92, Sch96]. If we interpret the size of an ad s_i, as the number of processor required by a parallel job, and $w_i \cdot T$ as the time required by the same job, then the ad placement problem can be interpreted as the scheduling of independent, non-malleable jobs for identical processors, where both preemption and job migration are allowed at no cost, and the objective is to minimize the makespan. To our knowledge, this combination has not been previously considered, possibly because of the impractical nature of this set of assumptions when considered as a parallel scheduling problem. Examples similar to the 2-D bin packing problem considered above show that changing any of the requirements can change the optimal schedule.

There are a growing number of companies providing services for scheduling Internet advertising (e.g., [Bel98]). To our knowledge, none of these permit the scheduling of ads of more than one shape, and there are no available references that provide details on the techniques used or the quality of the solutions provided.

Outline. In section 2, we consider the off-line problems of finding a valid solution to the ad placement problem with a given set of ads, and of finding the optimal subset of a set of ads that has a valid schedule. In section 3 we give upper and lower bounds for the on-line ad placement problem. In section 4, we present the various uses for solutions to the ad placement problem.

2 Off-line algorithms

2.1 Finding the optimal ad placement

Theorem 2.1 *Given a parameter T and a set of ads, it is NP-Hard to decide if there exists a valid schedule of the ads using T slots.*

This follows from the fact that bin packing is a special case of the ad placement problem. We here consider the version of the problem where the allowed ad sizes form a divisible sequence; this provides us with a 2-approximation to the general problem. Before we present an algorithm for this problem, we provide additional motivation for the difficulty of the problem considered by pointing out several natural seeming algorithms for this problem that do **not** always find a valid assignment in the cases where such an assignment exists. We use the example depicted in figure 1, which when $T = 4$, has the solution depicted in figure 2.

An algorithm commonly used to approximate bin packing (see for example [KRS96]), called Best Fit, is to place each ad in the slot that would maximize the resulting fullness of the slot. However, in the given example, this places ad 1 and ad 2 in the same slot, after which ads 5,6,7 and 8 cannot be placed. Thus, this algorithm does not always find the best solution for the ad placement problem. The algorithm that places the ads in sorted order by size, from largest to smallest, placing each ad i in the w_i fullest slots, has the same difficulty as Best Fit. Next consider the algorithm that places the ads in sorted order by size, from smallest to largest, placing ad i in the w_i least full slots. This algorithm places ads 3 and 4 into disjoint subsets of the slots, after which ads 1 and 2 cannot be placed. The algorithm that places the ads in sorted order from largest weight to smallest weight, placing ad i in the w_i least full slots has the same difficulty.

However, there is an algorithm that always provides an optimal schedule. One of the main results of this paper is to demonstrate that such an algorithm is the following, which is called Largest-Size Least-Full, or **LSLF**.

Algorithm **LSLF**

- Sort the ads by size, from largest to smallest.
- Assign the ads in the sorted order, where ad i is assigned to the w_i least full slots.

This algorithm can be implemented in time $O(\sum_{i=1}^{n} w_i + sort(A))$ by maintaining linked lists of slots, where each list contains all the slots of a given fullness. The heads to these lists are chained together into a doubly-linked list, called *main-list*. We maintain the invariant that the lists in main-list are stored in increasing order of fullness. Thus, we can always find the w_i least full slots in time $O(w_i)$. Furthermore, we can always perform the required increases of slot fullness in $O(1)$ time. To see this, consider what happens when the fullness of a slot t is increased from t_{before} to t_{after}. Because we have divisible ad sizes, and the ads are placed in non-increasing order of size, the list that contains the slots of size t_{after} must be the successor in main-list of the list containing the slots of size t_{before}.

Theorem 2.2 *Given a parameter T and a set of ads A with divisible ad sizes,* **LSLF** *finds an assignment which minimizes the maximum fullness of the slots.*

Proof. We assume w.l.o.g. that ad i is the i^{th} largest ad, sorted by ad size.

Definition: *fullness profile.* For an assignment of ads Q, let $h_m(Q)$ be the m^{th} largest value of $\sum_{i \in P(j)} s_i$ taken over all j, breaking ties arbitrarily. In other words, $h_m(Q)$ is the fullness of the m^{th} fullest slot. A fullness profile $\mathcal{F}(Q)$ is a sequence of T numbers $F_1(Q), F_2(Q), \ldots F_T(Q)$, such that $F_n(Q) = \sum_{i=1}^{n} h_i(Q)$. In other words, $F_n(Q)$ is the total fullness of the n most full slots.

Given two ad assignments Q and Q', we say that the fullness profile $\mathcal{F}(Q)$ *dominates* $\mathcal{F}(Q')$, if $F_1(Q) \geq F_1(Q')$, $F_2(Q) \geq F_2(Q')$, $\ldots F_T(Q) \geq F_T(Q')$. We denote this by $\mathcal{F}(Q) \geq_D \mathcal{F}(Q')$. Theorem 2.2 follows directly from the following claim.

Claim 2.3 *For every subset $A' \subseteq A$ of the first l ads in A, let Y_l be any assignment of the ads in A', and let G_l be the assignment found by* **LSLF**. *$\mathcal{F}(Y_l) \geq_D \mathcal{F}(G_l)$.*

This claim follows by induction on l. The base case, where $l = 1$, follows from the fact that all fullness profiles for a one ad problem are the same. We assume that $\mathcal{F}(Y_{l-1}) \geq_D \mathcal{F}(G_{l-1})$ for all Y_{l-1} and show that $\mathcal{F}(Y_l) \geq_D \mathcal{F}(G_l)$, by showing that if there exists an ad assignment Y_l for the first l ads in A, such that for some k, $F_k(Y_l) < F_k(G_l)$, then this can be used to construct an ad assignment Y_{l-1} for the first $l-1$ ads in A such that for some k', $F_{k'}(Y_{l-1}) < F_{k'}(G_{l-1})$. This contradicts the assumption that $\mathcal{F}(Y_{l-1}) \geq_D \mathcal{F}(G_{l-1})$ for all Y_{l-1}.

After **LSLF** assigns ad l, the locations where ad l is assigned can be divided into 2 regions: the least full $p \leq w_l$ slots, and $w_l - p$ other slots j that all have $|P(j)| = S'$, for some fixed S'. Also, any slot j that is not assigned ad l for which $|P(j)| < S'$, must have $|P(j)| = S' - s_l$. Let q be w_i plus the number of slots that are not assigned ad l and have $|P(j)| = S' - s_l$ (see figure 3).

For an assignment Y_l of the first l ads in A, let k be the smallest integer such that $F_k(Y_l) < F_k(G_l)$. To show how to use such a Y to construct an assignment of the first $l-1$ ads that violates the inductive hypothesis, we consider 3 cases, and show that in each case we merely need to remove the ad l from the slots that it has been assigned to.

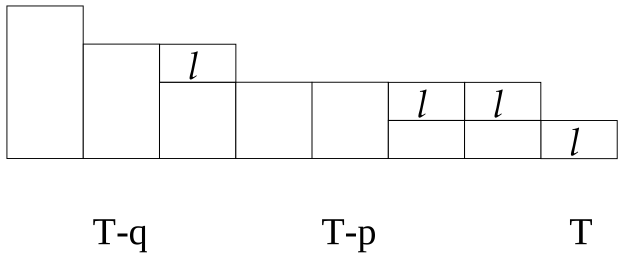

Figure 3: Fullness profile for **LSLF** after placing add l.

- $k \leq T-q$: Removing the ad l from G_l does not effect $F_r(G_{l-1})$ in the fullness profile for any $r \leq T-q$. Thus, $F_k(G_{l-1}) = F_k(G_l)$. Removing l from Y_l results in an assignment Y_{l-1} such that $F_k(Y_{l-1}) \leq F_k(Y_l)$, and thus $F_k(Y_{l-1}) < F_k(G_{l-1})$.

- $k \geq T-p$: Removing the ad l removes exactly $w_i - (T-k)$ ad assignments of size s_l from the value of the fullness profile at the k^{th} largest slot. Thus, $F_k(G_{l-1}) = F_k(G_l) - (w_i + k - T)s_l$. Since each ad can only appear in any slot once, no more than $T-k$ of the w_i ads assigned for ad l can appear in slots higher than k in the fullness profile for Y_l. Thus, removing the ad l from Y_l results in an assignment Y_{l-1} where $F_k(Y_{l-1}) \leq F_k(Y_l) - (w_i + k - T)s_l$, and again this gives an assignment Y_{l-1} of the first $l-1$ ads such that $F_k(Y_{l-1}) < F_k(G_{l-1})$.

- $T-q < k < T-p$: We show that this implies that $F_{T-p}(Y_l) < F_{T-p}(G_l)$, where the previous case holds. To see this implication, note that since k is the first slot for which $F_k(Y_l) < F_k(G_l)$, it must also be the case that $h_k(Y_l) < h_k(G_l)$, which in turn implies that $h_k(Y_l) \leq h_k(G_l) - s_l$. However, this implies that $h_{k'}(Y_l) \leq h_{k'}(G_l)$ for every slot k' after k, up to and including slot $T-p$ in the sorted order. Thus, it must also be the case that $F_{T-p}(Y_l) < F_{T-p}(G_l)$. ∎

Corollary 2.4 *Given a parameter T and a set of ads A of arbitrary size and weight, there is a 2-approximation to the question of finding a valid schedule in the following sense: if a valid assignment exists when we replace S by $\frac{S}{2}$, then we can find a valid assignment to the original problem.*

2.2 Finding the optimal subset with a valid schedule

In some cases, a valid schedule of the ads does not exist, in which case we want to find the best subset of the ads that does have a valid schedule. We wish to find the subset of the ads that maximizes advertising revenues. We here assume that the pricing scheme is proportional to the percentage of ad space utilized, but the techniques we describe apply to other pricing schemes as well. We here define the optimal subset A' to be the subset that maximizes $\sum_{i \in A'} |s_i w_i|$.

Theorem 2.5 *Given a parameter T and a set of ads A, it is NP-Hard to determine the subset $A' \subseteq A$ that maximizes the value $\sum_{i \in A'} w_i s_i$, where A' has a valid assignment of ads to slots. This holds even if the ad sizes are required to form a divisible sequence.*

Proof. This is by a reduction from the partition problem.

Recall PARTITION:
Instance: Finite set B and height $h_i \in Z^+$ for each $i \in B$.
Question: Is there a subset $B' \subseteq B$ such that $\sum_{i \in B'} h_i = \sum_{i \in B-B'} h_i$.

To reduce an instance of PARTITION to the question of finding the optimal subset, we let $T = \frac{1}{2}\sum_{i \in B} h_i$, and $S = 1$. The set of ads A has one ad for each element of B, where the weight of the ad is equal to the height of the corresponding element in B, and the size of the ad is 1. Determining the optimal subset for this T, S and set of ads A answers the corresponding partition question. ∎

We show that the following algorithm provides a 2-approximation for this problem.

Algorithm **SUBSET-LSLF**

- Let $B_s = \sum_{i, s_i = S} S w_i$. Let $B_{\bar{s}} = \sum_{i, s_i < S} s_i w_i$.
- If $B_s \geq B_{\bar{s}}$, then
  - Place the ads of size S in order by weight, discarding any ad that, when placed, would violate the size limit for some slot.
- On the ads with size $< S$, run algorithm **LSLF**, discarding any ad that, when placed, would violate the size limit for some slot.
- If $B_s < B_{\bar{s}}$, then
  - Place the ads of size S in order by weight, discarding any ad that, when placed, would violate the size limit for some slot.
- The set A' is the set of ads that are assigned to slots.

Theorem 2.6 *Given a parameter T and a set of ads A with divisible ad sizes, algorithm* **SUBSET-LSLF** *finds a subset $A' \subseteq A$, and a schedule for A', such that $\sum_{i \in A'} w_i s_i \geq \frac{OPT}{2}$, where OPT is the maximum value of $\sum_{i \in A'} w_i s_i$ over all subsets A' with valid schedules.*

The proof appears in appendix A

3 On-line algorithms

We next consider the on-line version of the ad placement problem. For any sequence of customer requests C, let $OPT(C)$ be the maximum of $\sum_{i \in A'} w_i s_i$ over all subsets A' of the customer requests that are accepted. For any on-line algorithm L, let $alg(L, C)$ be $\sum_{i \in A''} w_i s_i$, where A'' is the set of requests that are accepted by L. Our objective is to find the algorithm L that minimizes the quantity $\max_C \frac{OPT(C)}{alg(L,C)}$.

If the customer requests are restricted only in that the ad sizes must be divisible, any on-line algorithm performs poorly. For example, consider the following class of request sequences: k ads with size $\epsilon \ll S$ and weight T, which, when $k < S/\epsilon$ are followed by one final ad that has size S and weight T. Any deterministic algorithm that accepts one of the first S/ϵ ads does very poorly for the sequence where the next ad has size S. Any deterministic algorithm that does not accept one of the first S/ϵ ads does very poorly for the sequence where there is no ad with size S. Thus, the space provider needs to further restrict the allowed ad sizes and weights. We consider the scenario where there is an upper bound on both the ad size and the ad weight. We study the following algorithm.

Algorithm **OL-LSLF**
Every new customer request is accepted if it can be accepted without violating the requirement that there exists a valid assignment (which can be decided by **LSLF**).

Theorem 3.1 *If the allowed ad sizes form a divisible sequence, we have an upper bound Z on the maximum size of an ad, and we have an upper bound V on the maximum ad weight, where $2ZV < ST$, then* $\max_C \frac{OPT(C)}{alg(\mathbf{OL\text{-}LSLF},C)} \leq \frac{ST}{ST - Z(2V-1)}$.

Theorem 3.2 *For any on-line (possibly randomized) algorithm L, and any $\epsilon > 0$, there is a sequence C such that with probability at least $\left(\frac{\epsilon}{\epsilon+1}\right)^2$, it holds that $\frac{OPT(C)}{alg(L,C)} \geq \frac{ST}{ST-2ZV+Z\epsilon V+Z\max(1,\epsilon V)}$.*

These theorems demonstrate that the algorithm **OL-LSLF** is optimal, even if randomness is allowed, since ϵ can be made arbitrarily small. The proofs appear in appendix B.

4 Using solutions to the ad placement problem

In this section, we describe some of the uses for the ad placement problem to schedule ads. In the introduction, we described how to use solutions to schedule ads specified by an access fraction and a one dimensional geometry. Similarly, solutions to the ad placement problem can be used to schedule ads where each ad i is specified by a one dimensional geometry $L(i)$, and a rational time fraction $F(i)$. We assume every ad has access fraction 1. In this case, we set T to be the least common multiple of the denominators of the $F(i)$. We set $w_i = T \cdot F(i)$, $s_i = L(i)$, and S to the total available ad space. Once we have a valid ad assignment, all accesses are treated identically, and at any instant in time, what is displayed in the ad space is a set of ads that are assigned to one of the slots in the ad assignment. The choice of which slot to display can be made by cycling through either a deterministic or random permutation of the ads, or by always choosing a random slot.

Note that when an ad is specified in terms of both access and time fraction, the time fraction of an ad represents the fraction of time that the ad is displayed, given that an access sees the ad at all. For space providers where most accesses have a very short duration, the advantage gained by occasionally updating the set of ads seen by a single access is small, and thus in these scenarios the time fraction of all ads will typically be 1. However, for space providers that are accessed for longer periods of time, it becomes advantageous to update the ad space over time, and so for such providers, the time fraction becomes an important consideration. Space providers that are accessed for longer periods of time include on-line news services, and in fact the on-line news service PointCast [Poi98] updates its advertising space periodically. It is also possible that advertisements in the future will appear on web browsers, another set of potential space providers that are accessed for longer periods of time.

Solutions to the ad placement problem can be used when each ad sold by the provider has the same geometry as the ad space and is specified by a rational access fraction $U(i)$ and time fraction $F(i)$. In this case, we set T to be the least common multiple of the denominators of the $U(i)$. We set $w_i = T \cdot U(i)$, $s_i = F(i)$, and $S = 1$. Given a valid ad assignment, when a user accesses the provider, the ads seen by that access are determined by choosing one of the T time slots. Each of the ads i in that time slot j are seen a fraction of s_i of the time. This can be realized several different ways: by cycling through either a deterministic or random permutation of the ads, where each ad $i \in P(j)$ appears for a time proportional to s_i. Or, the time unit can be fixed, and the next ad is chosen by choosing each ad $i \in P(j)$ with probability s_i.

4.1 Ad geometries of higher dimension

Solutions to the ad placement problem can also be used in scenarios where the ads are specified by a size of more than one dimension. We show that any algorithm for the ad placement problem can be used to solve the following *n-D* ad placement problem, provided the ads adhere to a generalization of the divisibility requirement. In the *n-D* ad placement problem each ad i is specified by a weight w_i, and an n dimensional rectangle of size $(s_i^1, s_i^2, \ldots s_i^n)$. We again have T slots, but now each slot is an n dimensional rectangle of size $(S^1, S^2, \ldots S^n)$.

Since the ad sizes are now specified by more than 1 dimension, not all orderings of the ads within a slot require the same amount of n dimensional volume, and thus ads that are assigned to a slot also need to be assigned to a location within that slot. Thus, an ad assignment now consists of an assignment of ads to slots such that ad i is assigned exactly once to each of w_i slots, and within each assigned slot j, ad i has an assigned location $L_{ij} = (l_{ij}^1, l_{ij}^2, \ldots l_{ij}^n)$ subject to the *non-overlap* requirement: no other ad is assigned to a location within the n dimensional rectangle defined by L_{ij} and the point $L'_{ij} = (l_{ij}^1 + s_i^1, l_{ij}^2 + s_i^2, \ldots l_{ij}^n + s_i^n)$. We say that a schedule is valid, if, for each ad i and slot j, we have that $l_{ij}^1 + s_i^1 \leq S^1$, $l_{ij}^2 + s_i^2 \leq S^2$, $\ldots l_{ij}^n + s_i^n \leq S^n$.

Finding the optimal solution to the *n-D* ad placement problem, for example allows us to directly find a solution to the case where the ads are specified in terms of a height, a width, and either an access fraction or a time fraction. We shall also see in section 4.2 that solutions to the *n-D* ad placement problem can also be used to find solutions to the case where ads are specified in terms of an access fraction, a time fraction, as well as a geometry of an arbitrary number of dimensions.

The generalization of the divisibility requirement to n dimensions is as follows. There exists a series of allowed ad shapes $P_0 = (S^1, S^2, \ldots S^n), P_1 = (p_1^1, p_1^2, \ldots p_1^n), P_2 = (p_2^1, p_2^2, \ldots p_2^n), \ldots$, such that for each $t \geq 1$, there exists a dimension q_t, $1 \leq q_t \leq n$, and an integer $k_t \geq 2$, such that $p_{t-1}^q = k_t \cdot p_t^q$ and that $p_{t-1}^r = p_t^r$ for all $r \neq q$. Intuitively, this divisibility requirement states that there exists a sequence of possible ad shapes $Z_0, Z_1, Z_2, \ldots$, such that for all $t \geq 1$, Z_{t-1} is formed exactly by combining k_t of the shapes Z_t.

In order to describe our technique for the *n-D* ad placement problem, we define a set of *slot partitions*, partitions of the n dimensional slot using a divisible set of allowed ad sizes. In this set there are as many partitions as there are allowed ad sizes, which is possibly infinite. The first partition in the set divides the slot into k_1 regions, each of shape P_1. The second partition of the set divides each of the regions in the first partition into k_2 regions, each of size P_2. In general, the t^{th} partition in the set divides each region in the $t - 1^{st}$ partition into k_{t-1} regions, each of size P_t. We say that a region U in the u^{th} partition of the set is an *ancestor* of a region in the v^{th} partition, $u \leq v$, if U contains V. Note that every region is an ancestor of itself.

Given a divisible instance of the *n-D* ad placement problem, and an algorithm **1-D-ALG** for the ad placement problem that is guaranteed to find some valid ad placement when one exists, we use the following algorithm:

Algorithm **n-D-ALG**

- Replace each ad i of weight w_i and shape P_t by a 1-D ad with the same weight and with size
$$s_i = \frac{1}{\prod_{u=1}^{t} k_u}.$$
- Run algorithm **1-D-ALG** on the resulting ad problem, leaving T unchanged, and setting $S = 1$.
- Each n-D ad is assigned to the slot where the corresponding 1-D ad is assigned.
- Within a slot, the ads are assigned to locations in order of non-increasing size, where each ad of shape P_t is placed in any region R of the t^{th} partition of the set of slot partitions such that no previous ad has been placed in any ancestor of R.

Theorem 4.1 *If a valid assignment exists, then* **n-D-ALG** *finds it.*

The proof is provided in appendix C.

4.2 The Full Assignment Problem

We now turn our attention to the full assignment problem. In this problem, each ad is specified by an access fraction $U(i)$, a time fraction $F(i)$, and a geometry of any number of dimensions. We here describe the solution for the case where the geometry is specified by one dimension $L(i)$, but the results easily extend to geometries of an arbitrary number of dimensions. We are given T sets of slots, where each set consists of F slots of size S. The goal is to produce an ad assignment where ad i is assigned to $U(i) \cdot T$ sets of slots, and in each set where ad i is assigned, it is assigned exactly once to $F(i) \cdot F$ slots in that set. If $P(j,k)$ is the set of ads assigned to the k^{th} slot in set j, then we say that an ad assignment is *valid* if
$$\max_{j,k} \sum_{i \in P(j,k)} L(i) \leq S.$$

Our goal is to efficiently find a valid schedule, if one exists. Note that this solves the problem of interest: when a provider is first accessed, we choose a set of slots, and then, at each time step, a new slot from the set is chosen and displayed.

We first consider the effect of adding two additional requirements for the scheduling of ads within a set. The first is that ads be *consecutive*: that an ad must be assigned to consecutive slots within a set. The second is that the ads be *aligned*: the ads are assigned non-overlapping locations within a slot such that each ad is assigned to the same location in every slot it is assigned to. If we require ads to be both consecutive and aligned, a solution to the full assignment problem entails assigning each ad to a rectangle within each set of ad slots, where the width of the rectangle for ad i is $L(i)$, and the height of the rectangle is $F(i) \cdot F$. In this case, we can treat the full assignment problem simply as a 2 dimensional version of the n-D ad placement problem.

In some cases, there is a valid schedule that is neither consecutive or aligned but none that are either consecutive or aligned. For example, if a set of 3 slots contains three ads, each of which has time fraction $\frac{2}{3}$, and size $\frac{S}{2}$, then there exists a valid schedule for those three slots in the case where the ads need not be either consecutive or aligned, but not if either condition is required. However, the following claim, proven in appendix D, shows that if the time fraction is treated as one of the dimensions of the ad geometry, and the allowed pairs of time fraction and ad geometry are divisible, then it is sufficient to describe the full assignment problem as an n-D ad placement problem and use the algorithm provided for that problem.

Claim 4.2 *If the allowed pairs of time fraction and ad size are divisible, then if there exists any valid schedule to the full assignment problem, then there is a valid schedule that is both consecutive and aligned.*

Acknowledgments. The authors thank Ed Coffman, Sumit Ganguly and Avi Silberschatz for helpful discussions related to this work.

References

[BCR80] B. Baker, E. Coffman, and R. Rivest. Orthogonal packings in two dimensions. *SIAM Journal of Computing*, 9(4):846 – 855, 1980.

[Bel98] Bellcore. Adapt/x advertiser(tm) product overview. *See* http://www.bellcore.com/ADAPTX/ADVERTISER/index.htm, 1998.

[BS83] B. Baker and J. Schwarz. Shelf algorithms for two-dimensional packing problems. *SIAM Journal on Computing*, 12(3):508 – 525, 1983.

[CGJ87] E. Coffman, M. Garey, and D. Johnson. Bin packing with divisible item sizes. *Journal of Complexity*, 3:406 – 428, 1987.

[CGJT80] E. Coffman, M. Garey, D. Johnson, and R. Tarjan. Performance bounds for level-oriented two-dimentional packing algorithms. *SIAM Journal on Computing*, 9:808 – 826, 1980.

[CM96] S. Chakrabarti and S. Muthukrishnan. Resource scheduling for parallel database and scientific applications. In *Proc. 8th ACM Symp. on Parallel Algorithms and Architectures*, pages 329 – 335, June 1996.

[Dou98] Homepage of DoubleClick. *See:* http://www.doubleclick.net, 1998.

[FKST93] A. Feldmann, M. Kao, J. Sgall, and S. Teng. Optimal online scheduling of parallel jobs with dependencies. In *Proc. 25th ACM Symp. on the Theory of Computing*, pages 642 – 651, May 1993.

[GG75] M. Garey and R. Graham. Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, 4(2):187 – 200, 1975.

[HSW96] L. Hall, D. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line algorithms. In *Proc. 7th ACM-SIAM Symp. on Discrete Algorithms*, pages 142 – 151, January 1996.

[KRS96] Claire Kenyon, Yuval Rabani, and Alistair Sinclair. Biased random walks, lyapunov functions, and stochastic analysis of best fit bin packing. *Proc. 7th ACM-SIAM Symp. on Discrete Algorithms*, pages 351–358, 1996.

[LT94] W. Ludwig and P. Tiwari. Scheduling malleable and nonmaleable parallel tasks. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pages 167 – 176, January 1994.

[Net98] Homepage of NetGravity. *See:* http://www.netgravity.com, 1998.

[Poi98] Homepage of PointCast. *See:* http://www.pointcast.com/, 1998.

[Sch96] Uwe Schwiegelshohn. Preemptive weighted completion time scheduling of parallel jobs. *Proceedings of ESA 96, LNCS 1136*, pages 39–51, 1996.

[SWW91] D. Shmoys, J. Wein, and D. Williamson. Scheduling parallel machines online. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 131 – 140, October 1991.

[TSWY94] J. Turek, U. Schwiegelshohn, J. Wolf, and P. Yu. Scheduling parallel tasks to minimize average response time. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pages 112 – 121, January 1994.

[TWY92] J. Turek, J. Wolf, and P. Yu. Approximate algorithms for scheduling parallelizable tasks. In *Proc. 4th ACM Symp. on Parallel Algorithms and Architectures*, pages 323 – 332, June-July 1992.

[W3.98] Homepage of W3.COM. *See:* http://www.w3.com/home/, 1998.

A SUBSET-LSLF provides a 2-approximation

In this section we prove theorem 2.6, restated here for convenience.

Theorem A.1 *Given a parameter T and a set of ads A with divisible ad sizes, algorithm* **SUBSET-LSLF** *finds a subset $A' \subseteq A$, and a schedule for A', such that $\sum_{i \in A'} w_i s_i \geq \frac{OPT}{2}$, where OPT is the maximum value of $\sum_{i \in A'} w_i s_i$ over all subsets A' with valid schedules.*

Proof. We show that if $B_s \geq B_{\bar{s}}$, then even just the portion of the schedule consisting of the ads of size S provides a 2-approximation, and if $B_s < B_{\bar{s}}$, then the portion of the schedule consisting of the ads of size $< S$ provides a 2-approximation. When $B_s \geq B_{\bar{s}}$, if any ad of size S is discarded, then at least half the slots have an ad of size S assigned to them. In this case, $\sum_{i \in A'} w_i s_i \geq \min(\frac{TS}{2}, B_s)$, which is guaranteed to be a 2-approximation. In a set of ads with divisible ad sizes, any ad that does not have size S, has size $\frac{S}{2}$. Therefore, in the case that $B_s < B_{\bar{s}}$, we have from the following lemma that $\sum_{i \in A'} w_i s_i > \min(\frac{TS}{2}, B_{\bar{s}})$, which is again a 2-approximation.

Lemma A.2 *If all ads have size at most Z, and some ad cannot be assigned by* **LSLF**, *and if A' is the set of ads that have been assigned, then $\sum_{i \in A'} w_i s_i > T(S - Z)$.*

Proof. Let M be the maximum over all slots j of the value $\sum_{i \in P(j)} s_i$. Let m be the minimum over all slots j of the value $\sum_{i \in P(j)} s_i$.

Claim A.3 *For any algorithm that assigns each ad i to the w_i least full slots, after each ad has been assigned, $M - m \leq Z$.*

To prove lemma A.2, note that in the algorithm **LSLF**, if an ad cannot be assigned, then there must exist some slot j, for which $\sum_{i \in P(j)} s_i = S$. However, by claim A.3, this means that if some ad cannot be assigned, then, for every slot j, $\sum_{i \in P(j)} s_i \geq S - Z$. ■

Proof. (of claim A.3) The claim is true by induction. Let $d_i = M - n$ after the first i ads have been assigned. For the base case, note that $d_0 = 0$. For $d > 0$, $d_i \leq \max(d_{i-1}, Z)$. To see this, note that after assigning the i^{th} ad, M can be achieved at either a slot which was assigned ad i or one where it was not assigned. In the latter case, m has not decreased, and so $d_i \leq d_{i-1}$. In the former case, each slot that achieves m is also either a slot which was assigned ad i, in which case $d_i \leq d_{i-1}$, or one where it was not assigned, in which case the fact that the ad is assigned to the w_i least full slots gives us that $d_i \leq Z$. ■

■

B OL-LSLF is optimal

In this section we prove theorems 3.1, and 3.2, restated here for convenience.

Theorem B.1 *If the allowed ad sizes form a divisible sequence, we have an upper bound Z on the maximum size of an ad, and we have an upper bound V on the maximum ad weight, where $2ZV < ST$, then $\max_C \frac{OPT(C)}{alg(\textbf{OL-LSLF},C)} \leq \frac{ST}{ST - Z(2V-1)}$.*

Proof. If **OL-LSLF** is able to place every ad in a sequence C of requests, then $OPT(C) = alg(L, C)$. If some ad is rejected, then **LSLF** is not able to schedule the current set of accepted ads plus one prospective ad. In this case, let s be the size and w be the weight of the first ad that cannot be placed by **LSLF**. Since the ads are placed in order of size, and the ad sizes are divisible, we have that at least $T - w + 1 \geq T - V + 1$ of the slots must have fullness S. By claim A.3, this also implies that the remainder of the slots must have fullness at least $S - Z$. Thus, if **LSLF** is unable to place an ad, then the total fullness of the slots placed by **LSLF** must be at least $ST - ZV + Z$. These placed ads might include the one prospective ad, but the prospective ad contributes at most ZV, and thus for any C where some ad is rejected, $alg(\textbf{OL-LSLF}, C) \geq ST - 2ZV + Z$. ■

Theorem B.2 *For any on-line (possibly randomized) algorithm L, and any $\epsilon > 0$, there is a sequence C such that with probability at least $\left(\frac{\epsilon}{\epsilon+1}\right)^2$, it holds that $\frac{OPT(C)}{alg(L,C)} \geq \frac{ST}{ST - 2ZV + Z\epsilon V + Z\max(1,\epsilon V)}$.*

Proof. We show this with an adversary that requests three types of ads: type 1 ads, which have size ϵZ and weight V, type 2 ads, which have size Z and weight $\delta = \max(\lfloor \epsilon V \rfloor, 1)$, and type 3 ads of size Z and weight V. In a series of ad requests consisting only of type 1 ads, let p_i be the probability that the i^{th} such ad is the first ad accepted. We have that either there is a $k \leq 1/\epsilon$ such that $p_k \geq \frac{\epsilon}{\epsilon+1}$, or with probability $\geq \frac{\epsilon}{\epsilon+1}$, none of the first $\frac{1}{\epsilon}$ type 1 ads in such a sequence is accepted. If there exists such a k, we say the algorithm is *1-aggressive*. If the algorithm is 1-aggressive, then the adversary starts by requesting k type 1 ads. Otherwise the adversary starts by requesting $\frac{1}{\epsilon}$ type 1 ads.

If the algorithm is 1-aggressive, let q_i be the probability that in a sequence of k type 1 ads, followed by an infinite sequence of type 2 ads, the i^{th} type 2 ad is the first type 2 ad accepted, given that the k^{th} type 1 ad is the first ad accepted. If the algorithm is not 1-aggressive, let q_i be the probability that in a sequence of $\frac{1}{\epsilon}$ type 1 ads, followed by an infinite sequence of type 2 ads, the i^{th} type 2 ad is the first type 2 ad accepted, given that no type 1 ads are accepted. We have that either there is an $l \leq D = \min(V, \frac{1}{\epsilon})$ such that $q_l \geq \frac{\epsilon}{\epsilon+1}$, or with probability $\geq \frac{\epsilon}{\epsilon+1}$, none of the first D type 2 ads in such a sequence is accepted. If there exists such a l, we say the adversary is *2-aggressive*. If the algorithm is 2-aggressive, then the adversary next requests l type 2 ads. Otherwise the adversary next requests D type 2 ads.

If the algorithm is 1-aggressive and 2-aggressive, the adversary finishes by requesting $\frac{ST}{ZV}$ type 3 ads. If the algorithm is neither, the adversary finishes by requesting $\frac{ST}{ZV} - 2$ type 3 ads. Otherwise, the adversary finishes by requesting $\frac{ST}{ZV} - 1$ type 3 ads.

We analyze the 4 classes of algorithms separately. For 1- and 2-aggressive algorithms, with probability at least $\left(\frac{\epsilon}{1+\epsilon}\right)^2$, exactly one type 1 and one type 2 ads are accepted. If this occurs, the total number of type 3 ads that can be accepted is $\frac{ST}{ZV} - 2$. However, the optimal algorithm for the same sequence of requests rejects all type 1 and type 2 ads, and is thus able to accept $\frac{ST}{ZV}$ type 3 ads. Thus, $OPT(C) = ST$, but with probability at least $\left(\frac{\epsilon}{1+\epsilon}\right)^2$, $alg(L, C) \leq ST - 2ZV + Z(\epsilon V + \delta)$.

For algorithms that are neither 1- not 2-aggressive, with probability at least $\left(\frac{\epsilon}{1+\epsilon}\right)^2$, no type 1 and no type 2 ads are accepted. However, the optimal algorithm for the sequence used by the adversary accepts all ads, and thus for the C used by the adversary, we have $OPT(C) = ST$, but with probability at least $\left(\frac{\epsilon}{1+\epsilon}\right)^2$, $alg(L, C) \leq ST - 2ZV$. Similarly for the other two classes of algorithms, for the sequence C used by the adversary, $OPT(C) = ST$, but with probability at least $\left(\frac{\epsilon}{1+\epsilon}\right)^2$, $alg(L, C) \leq ST - 2ZV + Z(\epsilon V + \delta)$. ∎

C n-D-ALG finds the optimal solution

In this section, we prove theorem 4.1, restated here for convenience.

Theorem C.1 *If a valid assignment exists, then* **n-D-ALG** *finds it.*

Proof. We first point out that for any slot j, the described method of placing ads does not ever violate the non-overlap requirement. To see this, note that since the set of slot partitions is defined so that each region in the t^{th} partition is completely contained in some region of the u^{th} partition,

$u \le t$, the only way for two ads to overlap, is for an ad to be placed in a region R_1 that is completely contained in another region R_2 that is also assigned an ad. However, if R_2 has larger size than R_1, then the ad placed in R_2 was placed there first, and thus no ad could be placed in R_1, since R_2 is an ancestor of R_1. If R_1 and R_2 have the same size, then assume without loss of generality that the ad in R_1 is the second of the two ads place. Again, that ad could not have been placed there, since every region is an ancestor of itself.

We thus only need to show that if there exists a valid schedule then **n-D-ALG** finds one. Let the *volume* of an ad i be $v(i) = \prod_{l=1}^{m} s_i^l$. Let the volume of a slot be $V = \prod_{l=1}^{m} S^l$. If $P(j)$ is the set of ads assigned to slot j, then let $V(j) = \sum_{i \in P(j)} v(i)$. Since the sizes of the ads in the ad problem solved as a subroutine call to algorithm **1-D-ALG** are proportional to the volumes of the ads in the problem solved by **n-D-ALG**, the call to **1-D-ALG** finds an assignment of ads for which

$$\max_j V(j) \le V,$$

if such an assignment exists. Thus, to prove theorem 4.1, we only need to prove the following claim.

Claim C.2 *For any slot j, if $V(j) \le V$, then* **n-D-ALG** *finds a valid assignment of the ads in slot j to locations.*

Proof. We prove the counter-positive. If a valid assignment cannot be found, then for some t and i, an ad i of shape P_t cannot be assigned to a location within slot j without violating the non-overlap requirement. This means that every region in the t^{th} partition of the set of slot partitions has an ancestor that has an ad assigned to it. This is equivalent to saying that the entire volume of the slot already has ads assigned to it. Since $v(i) > 0$, this means that if ad i cannot be assigned, then $V(j) > V$. ∎

∎

D Solutions to the Full Assignment Problem

In this section, we prove claim 4.2, restated here for convenience.

Claim D.1 *If the allowed pairs of time fraction and ad size are divisible, then if there exists any valid schedule to the full assignment problem, then there is a valid schedule that is both consecutive and aligned.*

Proof. We show the claim to be true for each set of slots individually. In each set of slots, we need to assign a set of ads, where each ad i appears exactly once in each of $F(i)$ slots and requires size $L(i)$, and the allowed values of the pairs $(F(i), L(i))$ are divisible. We showed that if there exists a valid solution to this problem, then the algorithm **LSLF** finds one. It suffices to prove the claim by showing that if we specify one detail of **LSLF** correctly, then when the pairs $(F(i), L(i))$ are divisible, the algorithm always finds a schedule where the ads are consecutive and aligned.

To force **LSLF** to produce a consecutive and aligned schedule, we specify that when the w_i least full slots are chosen, ties are broken by choosing the smallest numbered slots. This does not effect the ability of the algorithm to find a valid solution, since **LSLF** minimizes the maximum fullness of the slots for any method of deciding between slots of equal fullness. The location of an ad within a slot is defined to be the fullness of the slot at the time of placement. Define a *slot size run* to be a maximal sequence of consecutive slots of equal fullness. We show that the following invariant holds: at each step of algorithm **LSLF**, the number of slots in every slot size run is a multiple of the value of $F(i)$ for every remaining ad i. This implies that the ad placement is both consecutive and aligned, since each ad is assigned to slots within a single slot size run.

We show that the invariant holds by induction on the number of ads that have been placed. Since the values of $F(i)$ are divisible, we see that the invariant holds before any ad has been placed: this provides the base case for the induction. For the inductive step, we assume that the invariant holds before an ad j is placed, and show that it must still hold after the ad n is placed. From the inductive hypothesis it follows that ad j is placed within a single slot size run, and thus ad j partitions some slot size run into two new slot size runs: one containing $F(j)$ slots and the other containing the remainder of the slots, if any. When the number of slots in the slot size run before ad j is placed is greater than $F(j)$, both of the two resulting slot size runs will be multiples of $F(j)$. However, the fact that the pairs $(F(i), L(i))$ are divisible implies that **LSLF** places the ads in non-increasing order of $F(j)$. Thus, for any remaining ad i, $F(i) \leq F(j)$. The inductive step follows from the fact that the $F(i)$ are divisible. ∎