The Robustness of the Sum-of-Squares Algorithm for Bin Packing

Michael A. Bender* Bryan Bradley† Geetha Jagannathan‡

Krishnan Pillaipakkamnatt§

Abstract

Csirik et al. [CJK+99, CJK+00] introduced the sum-of-squares algorithm (SS) for online bin packing of integral-sized items into integral-sized bins. They showed that for discrete distributions, the expected waste of SS is sublinear if the expected waste of the offline optimal is sublinear. This algorithm SS has a time complexity of $O(nB)$ to pack n items into bins of size B. In [CJK+02] the authors present variants of this algorithm that enjoy the same asymptotic expected waste as SS (with larger multiplicative constants), but with time complexities of $O(n \log B)$ and $O(n)$.

In this paper we present three sets of results that demonstrate the robustness of the sum-of-squares approach. First, we show the results of experiments from two new variants of the SS algorithm. The first variant, which runs in time $O(n\sqrt{B} \log B)$, appears to have almost identical expected waste as the sum-of-squares algorithm on all the distributions mentioned in [CJK+99, CJK+00, CJK+02]. The other variant, which runs in $O(n \log B)$ time performs well on most, but not on all of those distributions. Both these algorithms have simple implementations.

We present results from experiments that extend the sum-of-squares algorithm to the bin-packing problem with two bin sizes (the variable-sized bin-packing problem). From our experiments comparing SS and Best Fit over uniform distributions, we observed that there are scenarios where when one bin size is 2/3 the size of the other, SS has $\Theta(\sqrt{n})$ waste while Best Fit has linear waste. We also present situations where SS has $\Theta(1)$ waste while Best Fit has $\Theta(n)$ waste. We observe an anomalous behavior in Best Fit that does not seem to affect SS.

Finally, we apply SS to the related problem of online memory allocation. Our experimental comparisons between SS and Best Fit indicate that neither algorithm is consistently better than the other. If the amount of randomness is low, SS appears to have lower waste than Best Fit, while larger amounts of randomness appear to favor Best Fit. An interesting phenomenon shows that for a given range of allocation sizes we can find ranges of allocation duration where SS has lower waste than Best Fit. In the online memory-allocation problem for the uniform and interval distributions, SS does not seem to have an asymptotic advantage over Best Fit, in contrast with the bin-packing problem.

*Department of Computer Science, State University of New York, Stony Brook, NY 11794-4400, USA, email: `bender@cs.sunysb.edu`. Supported in part by Sandia National Laboratories and NSF grants EIA-0112849 and CCR-0208670.

†Department of Computer Science, Hofstra University, Hempstead, NY 11549, USA, email: `pot8os@acm.org`.

‡Department of Computer Science, State University of New York, Stony Brook, NY 11794-4400, USA, email: `geethajagan@acm.org`.

§Department of Computer Science, Hofstra University, Hempstead, NY 11549, USA, email: `csckzp@hofstra.edu`.

1 Introduction

In classical bin packing the input is a list $L = (a_1, a_2, ..., a_n)$ of n items and an infinite supply of bins of unit capacity. Each item a_i has size $s(a_i)$, where $0 < s(a_i) \leq 1$. The objective is to pack the items into a minimum number of bins subject to the constraint that the sum of sizes of the items in each bin is no greater than 1. Bin packing has a wide range of applications including stock cutting, truck packing, commercials assignment to stations breaks in television programming, and memory allocation. Because this problem is NP-hard [GJ79], most bin-packing research concentrates on finding polynomial-time approximation algorithms for bin packing. Bin packing is among the earliest problems for which the performance of approximation algorithms were analyzed [CGJ96].

In this paper we focus on the average-case performance of online algorithms for bin packing, where item sizes are drawn according to some discrete distribution. An algorithm is said to be *online* if the algorithm packs each item as soon as it "arrives" without any knowledge of the items not yet encountered. That is, the decision to pack item a_i into some particular bin can be based only on the knowledge of items $a_1, ..., a_{i-1}$. Moreover, once an item has been assigned to a bin, it cannot be reassigned. Discrete distributions are those in which each item size is an element of some set $\{s_1, s_2, ..., s_J\}$ of integers, and such that each size has an associated rational probability. The capacity of a bin is a fixed integer $B > s_J$.

We overload notation and write $s(L)$ for the sum of the sizes of the items in the list L. For an algorithm A, we use $A(L)$ to denote the number of bins used by A to pack the items in L. We write OPT to denote an optimal packing algorithm. Let F be a probability distribution over item sizes. Then $L_n(F)$ denotes a list of n items drawn according to distribution F. A *packing* is a specific assignment of items to bins. The *size* of a packing P, written $\|P\|$, is the number of bins used by P. For an algorithm A, we use $P_n^A(F)$ to denote a packing resulting from the application of algorithm A to the list $L_n(F)$. Given a packing P of a list L, the *waste* for P, the sum of the unused bin capacities, is defined as $W(P) = B \; \|P\| - s(L)$.

The *expected waste* of an algorithm A on a distribution F is $EW_n^A(F) = E[W(P_n^A(F))]$, where the expectation is taken over all lists of length n.

1.1 Related Work.

Bin Packing. Recently progress has been made in the average-case analysis of the standard heuristics for the discrete uniform distributions $U\{j,k\}$ where the bin capacity is taken as $B = k$, and item sizes are uniformly drawn from $1, 2, ..., j < k$. When $j = k-1$, the online Best-Fit and First-Fit algorithms have $\Theta(\sqrt{n})$ expected waste.

Remarkably, when $1 \le j \le k-2$, the expected waste of the optimal is $O(1)$ [ECG+91]. An algorithm is said to be *stable* under a distribution if the expected waste remains bounded even when the number of items goes to infinity. Coffman et al. [ECG+91] proved that Best Fit is stable when $k \geq j(j+3)/2$. Kenyon et al. [KRS98] showed that Best Fit is stable under $U\{k-2,k\}$ and is also stable for some specific values of (j,k) with $k \le 14$. It has been shown experimentally that for most pairs (j,k) the expected waste of Best Fit is $O(n)$.

The Sum-of-Squares algorithm. We assume that all items have integral sizes. The *gap* of a bin is the amount of unassigned capacity in the bin. Let $N(g)$ denote the number of bins in the current packing with gap $1 \le g < B$. Initially, $N(g) = 0$ for all g. The sum-of-squares algorithm puts an item a of size $s(a)$ into a bin such that after placing the item a the value of $\sum_{g=1}^{B-1} N(g)^2$ is minimized.

Csirik et al. [CJK+99], gave experimental evidence that for discrete uniform distributions $U\{j,k\}$ (with $k = 100$ and $1 \le j \le 98$) EW_{SS}^n is $O(1)$. They also showed for $j = 99$, $EW_{SS}^n = O(\sqrt{n})$. Their results indicated that for $j = 97, 98, 99$, the expected waste of SS, EW_{SS}^n goes from $O(1)$ to $\Theta(\sqrt{n})$ whereas the expected waste of BF, EW_{BF}^n transitions from $\Theta(n)$ to $O(1)$ to $\Theta(\sqrt{n})$.

In a theoretical analysis of the sum-of-squares algorithm [CJK+00] Csirik et al. proved that for any perfectly-packable distribution F, $EW_{SS}^n = O(\sqrt{n})$. They also proved that if F is a bounded waste distribution then EW_{SS}^n is either $O(1)$ or $O(\log n)$. In particular, if F is a discrete uniform distribution $U\{j,k\}$ where $j < k-1$ then $EW_n^{SS}(F) = O(1)$. They also proved that for all lists L, $SS(L) \le 3OPT(L)$.

Variable-sized Bin Packing. In the variable-sized bin-packing problem all bins do not have the same capacity. Assume that there are infinitely many bins of sizes

$$B_1 > B_2 > ... > B_k.$$

Let L denote a list of items whose sizes are in $(0, B_1]$. Lately this problem has received more attention because of its application in stock cutting and in the assignments of commercials to variable-sized breaks in television programming.

Much of the literature for variable-sized bin-packing algorithms measures effectiveness in terms of the performance ratio of the algorithm (instead of expected waste). The *asymptotic expected performance ratio* for algorithm A on distribution F is the average ratio of the number of bins taken by A to the number of bins taken by an optimal algorithm. That is,

$$ER_\infty^A(F) = \limsup_{n\to\infty} \left(E\left[\frac{\|P_n^A(F)\|}{\|P_n^{OPT}(F)\|} \right] \right).$$

Frieson and Langston [FL86] first investigated the variable-sized bin-packing problem. Kinnerly and Langston [KL88] gave an online algorithm with performance ratio 7/4. The variable-harmonic algorithm [Csi89], which is based on the harmonic-fit algorithm, has a performance ratio of at most 1.69103.

Epstein, Seiden, and Stee [ESS01] presented two unbounded-space online algorithms for variable-sized bin packing. They focused on the cases in which there are two bin sizes. These algorithms are a combination of variable-harmonic and refined-harmonic algorithms [LL81]. Epstein et al. also proved a lower bound for any online algorithm for two bin sizes. The asymptotic performance ratio is greater than 1.33561.

Memory Allocation. Memory is modeled as an infinitely long array of storage locations. An allocator receives requests for blocks of memory of various sizes and requests for their deallocation arrive in some unknown order. Although the size of the request is known to the allocator, the deallocation time is unknown at the time of allocation. The deallocation of a block may leave a "hole" in the memory. The objective of a memory-allocation algorithm is to minimize the total amount of space wasted in these holes.

Although memory allocation has been studied since the early days of computing, only a handful of results concern the competitive ratios of the standard heuristics for the problem. For the memory-allocation problem, the competitive ratio of an online algorithm is the ratio between the total amount of memory required by the algorithm to satisfy all requests, to W, the largest amount of concurrently allocated memory. Luby, Naor and Orda [LNO96] showed that First Fit (the heuristic which assigns a request to the lowest indexed hole that can accommodate the request) has a competitive ratio of $O\{\min(\log W, \log C)\}$, where C is the largest number of concurrent allocations. By Robson's result [Rob74], this bound is the best possible value for any online algorithm.

1.2 Results.

In this paper we study the performance of the sum-of-squares algorithm and its variants. We present faster variants of the sum-of-squares algorithm for the online bin-packing problem. Next, we compare the sum-of-squares algorithm with the Best-Fit algorithm for variable-sized bin packing. Finally, we show the results of applying the sum-of-squares algorithm to memory allocation.

We performed our experiments with the uniform distribution $U\{j,k\}$. We have also run our algorithms on interval distributions, especially those that are claimed to be difficult in [CJK+02]. For the memory allocation problems we used various interval distributions for both block sizes and block durations.

- In Section 2 we present our variants for the sum-of-squares algorithm. The first is the SSmax variant, which runs in $O(n\sqrt{B}\log B)$ time. Our experiments suggest that the performance of this variant is close to the SS algorithm in absolute terms, for all the distributions mentioned in [CJK+99, CJK+00, CJK+02]. The remaining algorithms form a family called the segregated sum-of-squares algorithms (SSS). These algorithms perform well in most of the distributions mentioned in the above papers. But on some distributions they do not have the same expected waste as SS. The best runtime in this family is $O(n \log\log B)$.

- Section 3 provides experimental results for the sum-of-squares algorithm applied to a generalized version of the bin-packing problem where bins come in two sizes. For fixed bin sizes, the performance of the sum-of-squares and Best-Fit algorithms can be categorized into three ranges based on item sizes. In the first range both algorithms appear to have constant waste. In the second range, Best Fit has $\Theta(n)$ waste and SS has $\Theta(1)$ waste. In the third range, when one bin size is around 2/3 the size of the other, SS has $\Theta(\sqrt{n})$ waste while Best Fit has linear waste. When the bin sizes are not in this ratio, both algorithms have $\Theta(\sqrt{n})$ waste. We observe an anomalous behavior in Best Fit that does not seem to affect SS.

- In Section 4 we applied SS to the related problem of online memory allocation. Our experimental comparisons between SS and Best Fit indicate that neither algorithm is consistently better than the other. Smaller allocation durations appear to favor Best Fit, while larger allocation durations favor SS. Also, if the amount of randomness is low, SS appears to have lower waste than Best Fit, while larger amounts of randomness appear to favor Best Fit.

  An interesting phenomenon shows that for a given range of allocation sizes we can find ranges of allocation duration where SS has lower waste than Best Fit. In the online memory-allocation problem SS does not seem to have an asymptotic advantage over Best Fit, in contrast to the bin-packing problem.

2 Faster Variants of the Sum-of-Squares Algorithm

In this section we present variants of the sum-of-squares algorithm. The SSmax variant of Section 2.2.1 runs in $O(n\sqrt{B}\log B)$ time and appears to have an expected waste remarkably close to that of SS. Experimental results indicate that the segregated sum-of-squares family of algorithms (Section 2.2.2) run faster, but, in some cases, have $\Theta(n)$ expected waste when SS has $\Theta(\sqrt{n})$ waste.

2.1 Sum-of-Squares Algorithm.

The sum of the sizes of items in a bin is the *level* of the bin. The *gap* of a bin is the amount of its unused capacity. If the level of a bin is ℓ, then its gap is $B-\ell$. Let P be a packing of a list L of items. Let the *gap count of g*, $N(g)$, denote the number of bins in the packing that have a gap of $1 \leq g < B$. We call N the *profile vector* for the packing. We ignore perfectly packed bins (whose gap is 0) and completely empty bins. The sum-of-squares for a packing P is $ss(P) = \sum_{g=1}^{B-1} N(g)^2$.

The sum-of-squares algorithm is an online algorithm that works as follows. Let a be the next item to pack. It is packed into a legal bin (whose gap is at least $s(a)$) such that for the resulting packing P' the value of $ss(P')$ is minimum over all possible packings of a.

When an item of size s arrives, there are three possible ways of packing the item

1. Open a new bin: Here the sum of squares increases by $1 + 2N(B-s)$.

2. Perfectly fill an old bin: Here the sum of squares decreases by $2N(s) - 1$.

3. The item goes into a bin of gap g where $g > s$: Here the sum of squares increases by $2 \cdot (N(g-s) - N(g) + 1)$. This step requires finding a g which maximizes the value of $N(g) - N(g-s)$.

Each time an item arrives, the algorithm performs an exhaustive search to evaluate the change in $ss(P)$ and finds an appropriate bin in $O(B)$ time. In [CJK+02] the authors discuss variants of the original SS algorithm. They present $O(n \log B)$ and $O(n)$ variants that approximate the calculation of the sum-of-squares. The authors prove that these variants have the same asymptotic growth rate for expected waste as SS, but with larger multiplicative constants. For example, they considered the distribution $U\{400, 1000\}$. For $n = 100,000$ items, they state that the number of bins used by the variant is 400% more than optimal, whereas Best Fit uses only 0.3% more bins, and SS uses 0.25% more bins than necessary. The situation does improve in favor of their variant for larger values of n. For $n = 10^7$, the variant uses roughly 9.8% more bins than optimal, but SS uses 0.0025% more bins than necessary. The authors claim their fast variants of SS are "primarily of theoretical interest", and that they are unlikely to be competitive with Best Fit, except for large values of n.

They also looked at other generalizations for the sum-of-squares algorithm. Instead of considering the squares of the elements of the profile vector, they examined the r^{th} power, for $r \geq 1$. These generalizations yield various SrS algorithms. For all finite $r > 1$, SrS performed qualitatively the same as SS. For the limit case of $r = 1$ (S1S), the resulting algorithm is one that satisfies the *any-fit* constraint. That is, the algorithm does not open a new bin unless none of the already open bins can accommodate the new item to be packed. Best Fit and First Fit are examples of algorithms that satisfy the any-fit constraint. The S∞S algorithm chooses to minimize the largest value in the profile vector. This variant has the optimal waste for the uniform distributions $U\{j, k\}$. However, the authors provide examples of distributions for which S∞S has linear waste while the optimal and SS have sublinear waste.

2.2 Our Variants.

The variants try to approximate the behavior of the sum-of-squares algorithm. By attempting to minimize the value of $ss(P')$, SS effectively tries to ensure that it has an approximately equal number of bins of each gap g, where $1 \leq g < B$. Intuitively, the value of $ss(P')$ can usually be reduced by lowering the largest gap count. However, minimizing the largest value in the profile vector *alone* is insufficient, as the S∞S experiments indicate [CJK+02]. Nevertheless, this insight plays a key role in our variants for the sum-of-squares algorithm. Instead of examining all possible legal gaps for the best possible gap (as SS does), we examine only the k largest gap counts, for some value of k.

2.2.1 Parameterized SSmax Algorithm.

The SSmax(k) parameterized algorithm is based on the S∞S variant mentioned above. The right choice for k is discussed later in this section. To pack a given item into a bin, SSmax(k) computes the value of $ss(P)$ for the k largest gap counts (for some $k \geq 1$). The algorithm also computes $ss(P)$ for opening a new bin and for perfectly packing the new item into an already open bin. The given item is packed into a bin such that the resulting packing minimizes $ss(P)$. (Note that when we set $k = B$, we get the original SS algorithm.) Using a priority queue, we can find the k largest values in $O(k \log B)$. This can be done in $O(k \log \log B)$ time using faster priority queues such as Van Emde Boas trees.

Experimental Results. For small values of k (such as for constant k, or for $k = \log B$), the experiments suggest that the results for SSmax(k) are qualitatively similar to those of S∞S. However, for k near $2\sqrt{B}$ the average waste for SSmax(k) undergoes a substantial change. For each of the distributions mentioned in [CJK+99, CJK+00, CJK+02], we observed that the performance of SSmax($2\sqrt{B}$) matched that of SS to within a factor of 1.4. In many cases the factor is as low as 1.02. (See Table 1 and Figure 1 for results from some of the distributions.) Note that in this and other tables when it becomes clear that the growth rate of an algorithm is linear in n, we did not run experiments for $n = 10^8$ and $n = 10^9$. In all other cases, we used 100 samples for $n \in \{10^5, 10^6\}$, 25 samples for $n \in \{10^7, 10^8\}$, and 3 samples for $n = 10^9$.

We also compared the waste of this variant with SS for the distributions $U\{j, 100\}$, $U\{j, 200\}$ and $U\{j, 1000\}$. As our results in Table 2, and Figure 2 show, the performance of the variant SSmax($2\sqrt{B}$) is remarkably similar to that of SS.

2.2.2 Segregated Sum of Squares Variants.

All Segregated Sum-of-Squares algorithms (SSS) divide the profile vector N into some number t of approximately equal-sized contiguous sections of the vector, numbered $0 \ldots (t-1)$. The choice of t, and other associated data structures, differentiate the various members of the SSS family. An auxiliary vector $L[0 \ldots t-1]$ keeps track of the gap g with largest gap count in each of the t sections. The change in $ss(P)$ is evaluated for some of the legal g values in L, plus $O(t)$ additional values. Thus, the best gap size can be selected in $\Theta(t)$ steps.

Item sizes: 11-13,15-18, Bin size: 51	10^5	10^6	10^7	10^8	10^9
SS	9538	31272	97650	300121	516184
SSmax($2\sqrt{B}$)	9559	31314	97665	300172	516388
SSmax(1)	316247	3161114	3.159×10^7		
SSS(sqrt)	19259	144570	1287001	12281857	1.237×10^8
Item sizes: 18-27, Bin size: 100	10^5	10^6	10^7	10^8	10^9
SS	8639	25454	76259	229869	622198
SSmax($2\sqrt{B}$)	12333	36082	108789	327609	943798
SSmax(1)	939548	9409415	9.399×10^7		
SSS(sqrt)	146468	1447068	1.444×10^7	1.443×10^8	
Item sizes: 1,11-13,15-18, Bin size: 51	10^5	10^6	10^7	10^8	10^9
SS	585	595	635	414	628
SSmax($2\sqrt{B}$)	525	537	594	465	577
SSmax(1)	26422	2639009	2.64×10^7		
SSS(sqrt)	1130	1236	1278	1026	1648

Table 1: Comparison of waste for SS, SSmax($2\sqrt{B}$), SSmax(1), and SSS(sqrt) for various discrete distributions with the number of items $n \in \{10^5, 10^6, 10^7, 10^8, 10^9\}$. The waste of SS and SSmax($2\sqrt{B}$) are of the same order of magnitude. SSmax(1) has linear waste for all three distributions. SSS(sqrt) has linear waste for the first two distributions while SS has sublinear waste.

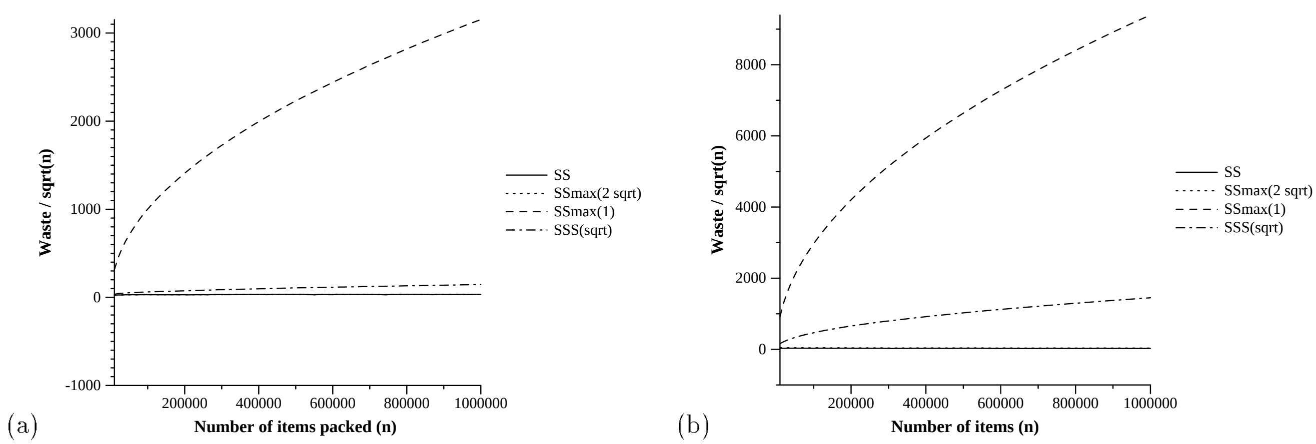


Figure 1: Comparison of waste for SS, SSmax($2\sqrt{B}$), SSmax(1), and SSS(sqrt). (a) Item sizes uniformly drawn in the range 11-13,15-18 with bin size = 51 (block 1 of Table 1). (b) Item sizes uniformly drawn in the range 18–27 with bin size = 100 (block 2 of Table 1). In both figures, the curves represent plots for the number of items packed vs waste/sqrt(number of items). In both (a) and in (b), SS and SSmax($2\sqrt{B}$) are horizontal lines and are indistinguishable, and hence they are $\Theta(\sqrt{n})$ waste. Other algorithms in these graphs have linear waste.

U{j,100}

j=12	SS	SSmax($2\sqrt{B}$)	j=75	SS	SSmax($2\sqrt{B}$)	j=99	SS	SSmax($2\sqrt{B}$)
10^5	40	42		900	974		32302	32287
10^6	47	57		916	982		108251	108176
10^7	52	52		680	776		308919	308223
10^8	45	45		923	1014		876747	876354

U{j,200}

j=24	SS	SSmax($2\sqrt{B}$)	j=150	SS	SSmax($2\sqrt{B}$)	j=198	SS	SSmax($2\sqrt{B}$)
10^5	117	117		3094	3574		82875	83680
10^6	91	91		2691	2895		193144	193332
10^7	63	63		3920	3723		541186	538786
10^8	78	78		3639	3684		1908245	1908852

U{j,1000}

j=120	SS	SSmax($2\sqrt{B}$)	j=400	SS	SSmax($2\sqrt{B}$)	j=990	SS	SSmax($2\sqrt{B}$)
10^5	253	253		4432	8062		977566	959566
10^6	876	876		5139	8328		2552676	2498732
10^7	580	580		5020	8031		5201380	5086175
10^8	672	672		4882	8081			

Table 2: Comparison of waste for SS and SSmax($2\sqrt{B}$) for various uniform distributions $U\{j,k\}$. Here the item sizes are drawn at random between $1 \ldots j$ and k denotes the bin size. The number of items $n \in \{10^5, 10^6, 10^7, 10^8\}$. The waste of SS and SSmax($2\sqrt{B}$) are of the same order of magnitude. For $j = 99$, both SS and SSmax($2\sqrt{B}$) have $\Theta(\sqrt{n})$ waste. In all the other cases both algorithms have constant waste.

Algorithms. We obtain various members of the SSS family as follows:

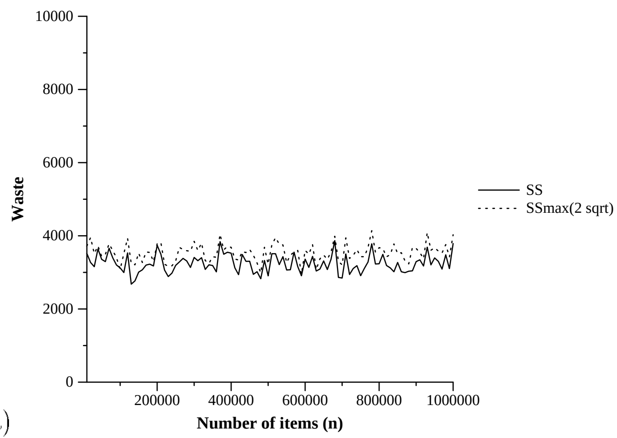
- SSS(sqrt). We set the number of intervals $t = \lceil \sqrt{B} \rceil$. Each section is of size $\lceil (B-1)/\sqrt{B} \rceil$. The algorithm runs as follows: Consider a new item of size s to be packed. We first examine gap values from section $m = \lceil s/\sqrt{B} \rceil - 1$. This is the section that contains gaps with the tightest fit for the new item. We evaluate the change in $ss(P)$ for all legal gap values of g in section m (there are $\Theta(\sqrt{B})$ elements in each section). For each section $m' > m$, we evaluate the change in $ss(P)$ at the largest gap count, $L(m')$, for section m'. There are $\Theta(\sqrt{B})$ of these sections. The gap that minimizes the value of $ss(P)$ is chosen for packing the new item. The process of finding the best gap takes $\Theta(\sqrt{B})$ time. Once a gap has been identified for the new item, we update L values for at most two sections of the profile vector (the section that contains the chosen gap g, and the section that contains the newly made gap $g-s$). This update can be done by an exhaustive scan of each section in $\Theta(\sqrt{B})$ time. Thus, the overall time complexity to pack one item is $\Theta(\sqrt{B})$.

  A simplification of this algorithm is to examine the change in $ss(P)$ at $L(m)$ alone, instead of computing the change in $ss(P)$ for all the gaps in section m. This simplified algorithm appears to have linear waste when SS has a sublinear waste. Hence both components of the algorithm seem to be necessary in order to obtain a variant that gives average waste similar to that of SS.

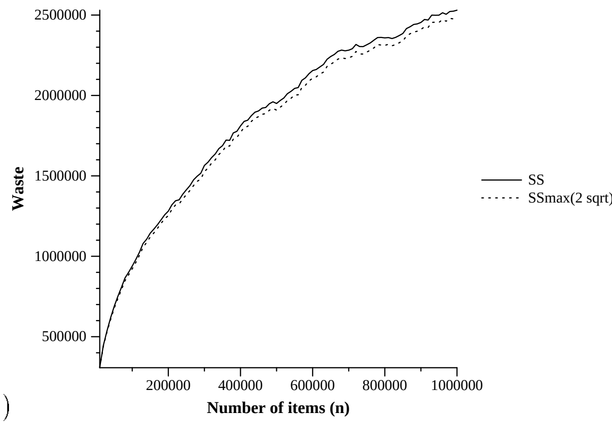
- SSS(log). In this case, we set the number of sections $t = \lceil \log B \rceil$. For each section we maintain a max heap on the gap counts of that section. Let s be the size of the new item to be packed and $m = \lceil s \log B / B \rceil - 1$. We evaluate the change in $ss(P)$ for the following gaps g:

  1. The gap with the largest count in section m, $L(m)$, and the $\log B$ gaps that follows L(m). That is, for gaps $L(m)$, $L(m)+1$, $\ldots$, $L(m) + \log B$.
  2. For each section $m' > m$, the gap with the largest count in m', $L(m')$.

  Thus, we compute the change in the value of $ss(P)$ for only $\Theta(\log B)$ values. Note that since there is a heap for each section m, the gap with the largest count in a section, $L(m)$, can be found simply by examining the root of the heap. Once the gap g with the least increase in $ss(P)$ has been identified, at most two heaps need to be updated. The heap that includes the gap $g-s$ requires an *increase_key* operation, and the heap that includes g requires a *decrease_key* on the heap. Each operation can be performed in $\Theta(\log B)$ time in the worst case.



(a)



(b)

Figure 2: Comparison of waste for SS and SSmax($2\sqrt{B}$). In both figures, the curves represent plots for the number of items packed vs waste. (a) Item sizes are uniformly drawn in the range 1–150 with bin size = 200 (block 2 of Table 2). Both SS and SSmax($2\sqrt{B}$) appear to have constant waste. (b) Item sizes are uniformly drawn in the range 1–990 with bin size = 1000 (block 3 of Table 2). The waste of SS and SSmax($2\sqrt{B}$) are of the same order of magnitude.

- SSS(loglog): We set $t = \lceil \log\log B \rceil$. For each section we maintain a maxheap of at most $\log B$ non-zero gap counts of the section. Since not all values of a section are in the heap, the top of the heap may not contain the largest gap count for the section. When a gap count in a section changes we add it to the heap and drop the last value in the heap (if the number of items in the heap exceeds $\log B$). Let s be the size of the new item to pack and $m = \lceil \frac{s \log\log B}{B} \rceil - 1$. If $H(m)$ represents the largest value in the heap for section m, we evaluate the change in $ss(P)$ at the following gap values:

  1. The gap with the largest count in the heap for section m, and the $\log\log B$ gap values that follow (that is, for gaps $H(m)$, $H(m)+1, \ldots, H(m) + \log\log B$).
  2. For each section $m' > m$, we evaluate the change at gap $H(m')$.

Experimental Results. We ran our experiments on the $U\{j,k\}$ distributions. For $B = k = 100$, for each value of j from 1 to 99, and for each of $n \in \{10^5, 10^6, 10^7, 10^8, 10^9\}$, we computed the average waste of SS, BF, SSS(sqrt), SSS(log), and SSS(loglog). We averaged the waste over 100 runs for $n \in \{10^5, 10^6\}$, over 25 runs for $n \in \{10^7, 10^8\}$, and over 3 runs for $n = 10^9$. We know from [CCG+00] that $EW^n_{OPT} = O(1)$ for $j \leq k-2 = 98$, and $EW^n_{OPT} = \Theta(\sqrt{n})$ when $j = k-1 = 99$. We have summarized the results in Table 3, showing the waste for j at 12, 25, 75, and 99.

The experiments show all versions of the SSS algorithms have the constant-waste property for $j \leq 98$, and appear to have qualitatively the same waste as SS for $j = 99$. Except for the SSS(loglog), the rest of the family appear to be within 0.80% of SS in terms of waste. We also observe that the waste for SSS(log log) seems to match SS to within a constant factor (a factor no more than 7).

The SSS variants, however, do not fare as well on some of the interval distributions. Table 1 suggests for some of the distributions, SS appears to have sublinear waste, while the SSS($\sqrt{B}$) algorithm appears to exhibit linear waste.

Our experiments thus indicate that the sum-of-squares minimization is sufficiently robust that it is not necessary to compute it precisely to receive its benefits. The variants we have presented are uncomplicated algorithms that are easily implemented.

3 Variable-sized Bin Packing

We now present results from experiments that naturally extend the sum-of-squares algorithm to accommodate variable-sized bins. We compare these results with the results from a straightforward extension of the Best-Fit algorithm.

Algorithm. Let us assume that there are infinitely many bins of integral sizes $B = B_1 > B_2 > \ldots > B_k$, where $k \geq 2$, and that items have integral sizes. Let $N(g)$ denote the number of bins in the current packing with gap $1 \leq g < B$. When an item arrives, it goes

j=12	10^5	10^6	10^7	10^8	10^9
SS	46	48	53	50	18
SSS(sqrt)	46	49	53	50	18
SSS(log)	46	48	53	50	18
SSS(log log)	302	404	357	350	402
BF	46	49	53	50	18
j=25	10^5	10^6	10^7	10^8	10^9
SS	60	62	58	53	68
SSS(sqrt)	61	63	58	53	68
SSS(log)	61	63	58	53	68
SSS(log log)	341	335	350	313	342
BF	177	834	7322	71893	728368
j=75	10^5	10^6	10^7	10^8	10^9
SS	920	902	917	873	1041
SSS(sqrt)	934	906	927	902	1025
SSS(log)	1003	953	973	933	1008
SSS(log log)	1204	1174	1213	1133	1324
BF	43448	425606	4243029	42460163	424241908
j=99	10^5	10^6	10^7	10^8	10^9
SS	33996	107707	354053	1089447	2848891
SSS(sqrt)	33772	107207	354003	1079132	2484005
SSS(log)	33651	106743	352245	1076027	2832558
SSS(log log)	33228	105379	343777	1058548	2954212
BF	26330	88169	302489	950047	27843191

Table 3: Comparison of SSS algorithms with SS and BF for various uniform distributions, where items sizes are drawn in the ranges $1 \ldots j$ for $j = 12, 25, 75, 99$ and bin size $= 100$, and the number of items $n \in \{10^5, 10^6, 10^7, 10^8, 10^9\}$. All variants of SS have waste of the same order of magnitude as SS. BF has linear waste when $j \in \{25, 75\}$, while SS has sublinear waste for those values of j.

into a legal bin such that the value of $\sum_{g=1}^{B-1} N(g)^2$ is minimized. When a new bin has to be opened, the algorithm chooses a bin of size B_i such that the value $\sum_{g=1}^{B-1} N(g)^2$ is minimized.

Experimental Results. All of our experiments are conducted under the uniform discrete distributions. We consider the case of two bin sizes. With one bin size fixed at $B_1 = 100$ we varied the other bin size B_2, $2 \leq B_2 < 100$. We generated item sizes uniformly at random between 1 and j, for each value of j between 2 and 99. We compared the modified SS algorithm against a straightforward extension of the Best-Fit algorithm for multiple bin sizes.

Our observations show that for fixed bin sizes the performance of these algorithms can be categorized into three ranges based on item sizes. In the first range, both algorithms appear to have constant waste. In the second range, Best Fit has $\Theta(n)$ waste and SS has $\Theta(1)$ waste. In the third range, when one bin size is around 2/3 the size of the other SS has $\Theta(\sqrt{n})$ waste while Best Fit has $\Theta(n)$ waste. When the bin sizes are not in the ratio mentioned above, both algorithms have $\Theta(\sqrt{n})$ waste (see Table 4 and Figure 3). The behavior of SS in these ranges is independent of the bin sizes. In the case of Best Fit the above mentioned behavior is seen as long as the largest item size is greater than the smaller bin size. When the size of the largest item generated, j, is no greater than the smaller bin, B_2 (that is, if $j \leq B_2$), then Best Fit never opens a bin of the larger size B_1. In those cases, the behavior of Best Fit is as in classical bin packing with only one bin size.

We also observed an interesting phenomenon in the performance of Best Fit for the two bin case. For values of the largest item size, j, greater than 75, the ratio of the average waste of Best Fit to average waste of SS remains constant (around 150) as the size of the second bin goes from 2 to near 60. When B_2 is about 2/3 the size of the first bin, the ratio undergoes a 15-fold increase (to around 2500), and then subsequently drops (Figure 4). That is, there are counter-intuitive situations where increasing the size of the second bin actually causes Best Fit's waste to dramatically increase. (Although a similar phenomenon can also be observed for smaller values of j, the change in waste is not as dramatic as for $j \geq 75$.)

B1=100	n	1-10		1-40		1-70		1-99	
		SS	BF	SS	BF	SS	BF	SS	BF
B2=20	10^5	50	21	105	5714	632	32687	34732	26300
	10^6	53	21	104	54750	601	317288	110091	88247
	10^7	53	29	96	543740	635	3147865	319029	285693
	10^8	26	26	114	5433252	627	31461781	1037493	952666
B2=40	10^5	51	20	121	9151	612	41208	32991	26563
	10^6	64	19	119	27823	592	404233	106553	87191
	10^7	49	21	125	86083	606	4021002	340963	289709
	10^8	32	12	132	239912	549	40243002	1005371	835138
B2=60	10^5	55	31	112	12918	616	68911	32738	228309
	10^6	50	28	131	123699	640	654135	109844	2181956
	10^7	48	30	115	1235771	695	6462667	354860	21413094
	10^8	57	30	112	12356346	467	64342607	1092447	212776914
B2=90	10^5	48	44	124	5152	514	40580	26442	65691
	10^6	45	43	120	49566	572	397687	90732	522376
	10^7	41	50	97	497937	493	3979766	293882	4794953
	10^8	50	13	106	4960739	660	39737014	895774	46178777

Table 4: Comparison of the waste for SS and BF for various uniform distributions with one bin size $B_1 = 100$ and the other bin size $B_2 = 20, 40, 60, 90$. The item sizes are from 1-10, 1-40, 1-70, 1-99. The number of items to be packed $n \in \{10^5, 10^6, 10^7, 10^8\}$.

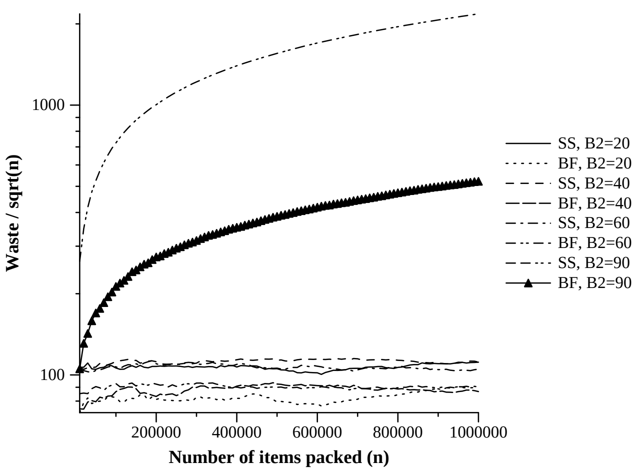


Figure 3: Comparison of SS and BF for item sizes 1...99, $B_1 = 100, B_2 \in \{20, 40, 60, 90\}$. The y-axis has a log scale. This a plot of the number of items packed vs. the waste/$\sqrt{n}$. All horizontal lines represent $\Theta(\sqrt{n})$ waste. The SS algorithm has $\Theta(\sqrt{n})$ waste for all values of B_2 listed above. The Best-Fit algorithm has linear waste for $B_2 \in \{60, 90\}$ and $\Theta(\sqrt{n})$ waste for $B_2 \in \{20, 40\}$.

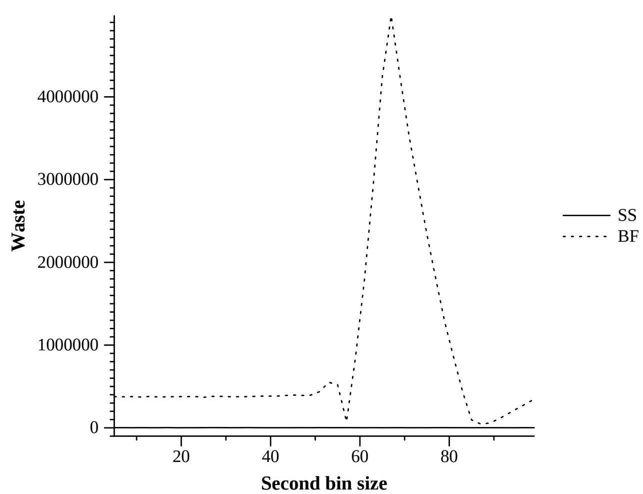


Figure 4: Best Fit's anomalous behavior for two bin sizes (item sizes 1. . .85, $B_1 = 100$). The number of items packed = 10^6. This graph indicates that the SS is able to adapt as the second bin size changes, but Best Fit is unable to adapt as the second bin size changes.

4 Memory Allocation

Researchers have long observed the similarity between bin packing and memory allocation [Dyc90, CL77, CL86]. A number of heuristics that apply to bin packing translate directly to memory allocation, and vice-versa. Since the sum-of-squares algorithm is effective for bin packing [CJK+02], we also study the algorithm in the context of memory allocation.

4.1 Bin Packing and Memory Allocation.

Dynamic bin packing [CGJ83] is a variant of the classical bin-packing problem where items are periodically deleted. At any step, an algorithm for this problem is either given a new item to pack, or is required to delete a previously packed item. As always, the objective in this problem is to minimize the total number of bins used for the entire list of items.

Closely related to the dynamic bin-packing problem is the *memory-allocation* problem in paged-memory operating systems. In this online problem, memory is represented as an infinitely long array of storage locations. Requests for blocks of memory of various sizes, and requests for their deallocation arrive overtime. Although the size of the request is known to the allocator, the deallocation time is unknown at the time of allocation. When an allocation request arrives, it must be satisfied by assigning a contiguous set of free memory locations. Once an allocation has been made, the block cannot be moved until it is deallocated. The deallocation of a block may leave a "hole" in the memory. The objective of a memory allocation-algorithm is to minimize the total amount of space wasted in these holes.

4.2 Sum-of-Squares Algorithm for Memory Allocation.

The algorithm maintains a list L of free blocks of memory. It also maintains a profile vector P which lists the number of free blocks of each size. (The block sizes that have a zero count are not explicitly stored in the vector.) When a new request for allocation arrives, the algorithm computes the sum of the squares based on the profile vector, for each legal block size. We say a block size is legal if the requested allocation fits in the block. The algorithm selects a block that has the minimum sum-of-squares value. When a free block is selected the requested memory is allocated from the beginning of the block. The selected block is deleted from the list L and a new block that represents the left-over portion of the block is inserted into L. If a block does not fit into any of the free blocks, then we "enlarge" memory to accommodate the new request. When a deallocation request arrives, we obtain a free block and it is added into L. If the newly deallocated block is contiguous with a previously deallocated block (or blocks) in L, these blocks are merged to form one larger free block. Free blocks at the end of memory are discarded and the memory "shrinks" to end at the last allocated block. In all cases, the profile vector P is updated appropriately.

4.3 Experimental Results.

There are no well defined "bin sizes" for memory allocation, in contrast with bin packing, although holes in memory correspond to gaps in bins. The parameters for our experiments in memory allocation are

- the range of block sizes, and
- the range of block durations.

For consistency with the rest of the paper, we have restricted ourselves to uniform and interval distributions for both parameters mentioned above. In our experiments the block size B represents the interval $[B, B+\delta_B]$. The value of B ranges from 1 to 1000 and δ_B varies from 1 to 1000. Similarly, a duration of t represents the interval $[t, t+\delta_t]$. The values of t ranges from 1 to 20,000 and δ_t varies from 1 to 1000.

After each request is assigned a block of memory, we measure the waste of the algorithm (as the sum of the blocks on the free list). The waste of the algorithm for a run is the maximum waste taken over all the allocation requests.

The experiments indicate that neither algorithm is consistently of lower waste than the other for these distributions. Even when an algorithm has a lower waste than the other, both algorithms seem to have waste of the same order of magnitude. Figure 5(a) is a typical example of the scenario where BF has lower waste than SS. Our experiments suggest that BF appears to have an advantage over SS when the duration of the requests are small. Figure 5(b) is a typical example of the scenario where SS has lower waste than BF. Figure 6(a) shows the difference in waste between the two algorithms as the request size increases. The difference between the waste of BF and that of SS also increases as the longevity of the allocation requests increases (Figure 6(b)). We have also observed that for every possible range of request sizes there exists a lower bound on the time duration from which point onwards the sum-of-squares algorithm has lower waste than Best Fit (Figure 7).

Our experiments suggest that in the online memory-allocation problem SS does not seem to have an asymptotic advantage over Best Fit for the uniform and interval distributions, in contrast with the bin-packing problem. In bin packing SS performs well since it tries to maintain an equal number of bins for each gap size. So

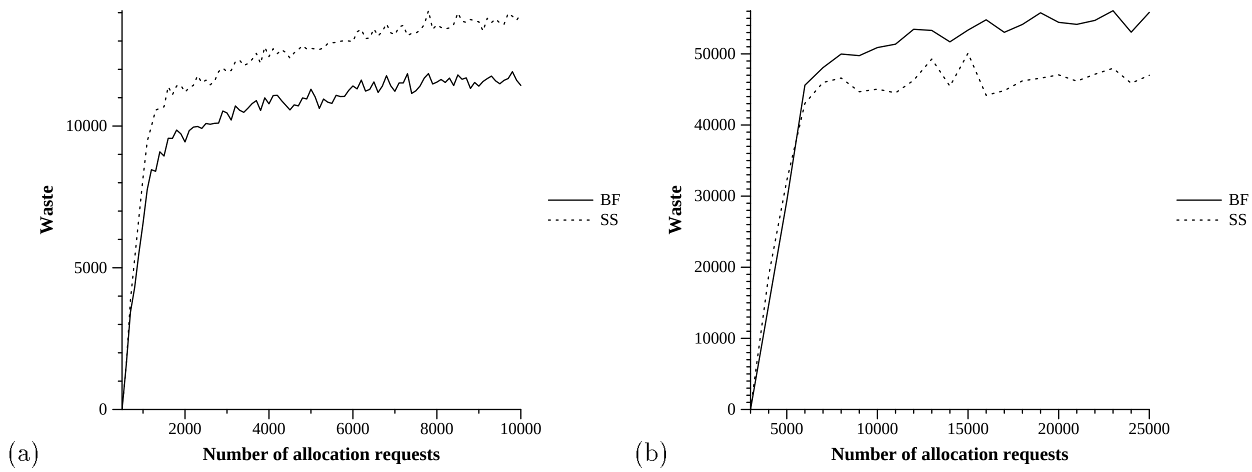


Figure 5: Comparison of the waste of Best Fit against Sum of Squares for memory allocation. These are plots of the number of allocation requests vs. waste. (a) Duration range is 500-599, and size range is 100-199. The size range and the duration range are fixed. Best Fit has lower waste, but the waste appears to be of the same order as for Sum of Squares. (b) Duration range is 3000-3099, size range is 150-249. The size range and the duration range are fixed. Best Fit has higher waste, but the waste appears to be of the same order as for Sum of Squares.

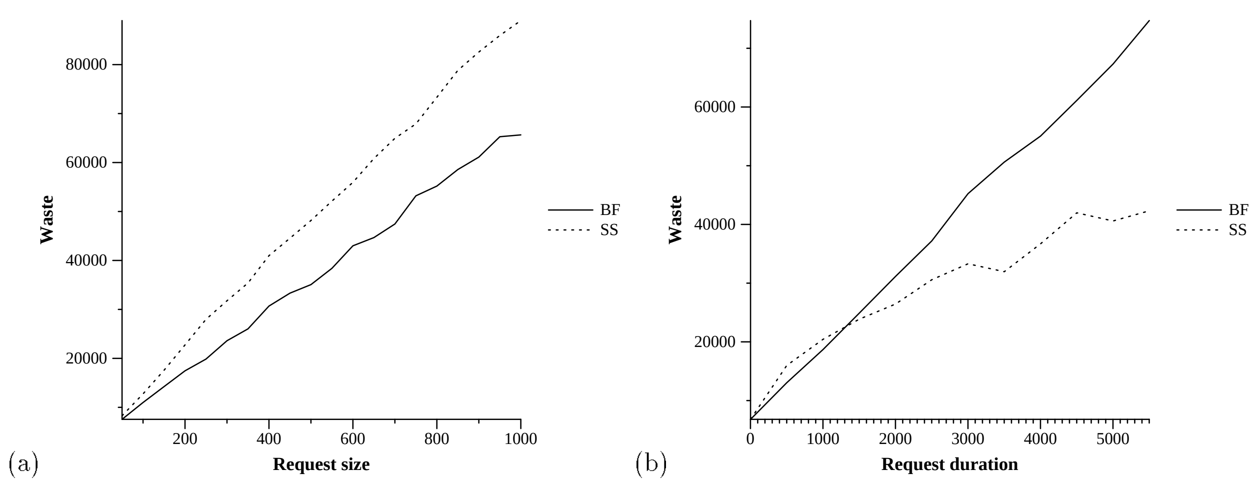


Figure 6: Comparison of waste of Best Fit and Sum of Squares for memory allocation. (a) The is a plot of request size vs. waste. Duration range is 500-599. The number of allocation requests is 5000. The duration range and the number of allocation requests are fixed. The difference in waste increases as the block size increases. (b) This is a plot of request duration vs. waste. Block size range is 101-200. The number of allocation requests is 20,000. The difference between the waste of BF and SS increases as item duration increases.

when an item arrives, the allocator finds a suitable gap size and is able to pack a bin perfectly.

A possible explanation for the behavior of the algorithms indicated by our experiments is as follows. In our experiments both the block size and the duration are uncorrelated (they are independent random variables). So when the system reaches a steady state where the rate of allocation and deallocation are the same, most of the time the Best-Fit allocator is able to find a hole of an appropriate size for any request. Deallocation provides Best Fit the same advantage that helps SS perform well in bin packing. It is too early to predict what happens when durations follow a more realistic distribution in which durations are correlated with size, such as the heavy tail distribution.

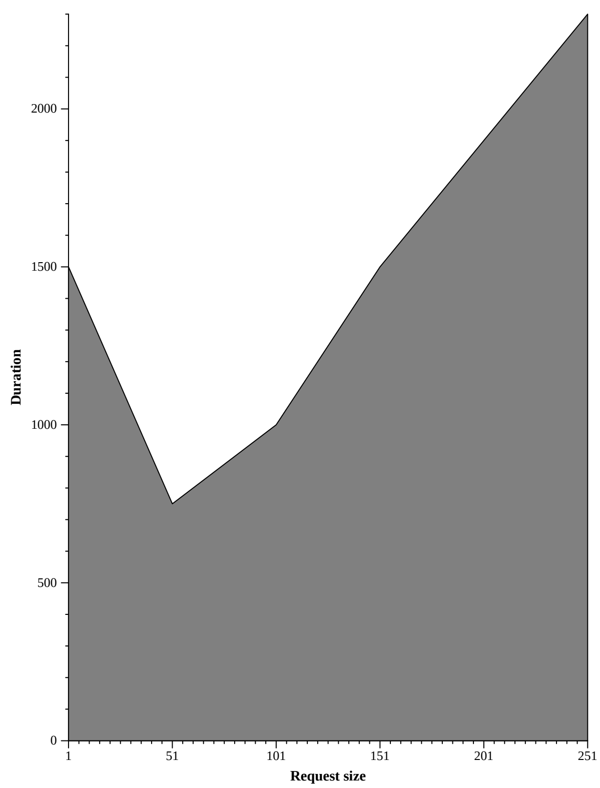


Figure 7: Comparison of SS and BF over a range of parameters of request sizes and durations. Here $\delta_B = \delta_t = 99$. For all points above the curve SS has lower waste than BF. For all points below the curve, BF has lower waste than SS.

References

[CCG+00] E. G. Coffman, C. Courcoubetis, M. R. Garey, D. S. Johnson, P. W. Shor R. R. Weber, and M. Yannakakis. Bin packing with discrete item sizes, part I: Perfect packing theorems and average case behavior of optimal packings. *SIAM Journal on Discrete Math.*, 13:384–402, 2000.

[CGJ83] E. G. Coffman, M. R. Garey, and D. S. Johnson. Dynamic bin packing. *SIAM Journal of Computing*, 12:227–258, 1983.

[CGJ96] E. G. Coffman, M. R. Garey, and D. S. Johnson. *Approximation Algorithm for NP-Hard Problems*, chapter Approximation Algorithms for Bin Packing: A Survey, pages 46–93. 1996.

[CJK+99] J. Csirik, D. S. Johnson, C. Kenyon, P. W. Shor, and R. R. Weber. A self organizing bin packing heuristic. In *Proceedings of ALENEX workshop*, pages 246–265, 1999.

[CJK+00] J. Csirik, D. S. Johnson, C. Kenyon, J. Orlin, P. W. Shor, and R. R. Weber. On the sum-of-squares algorithm for bin packing. In *Proceedings of the 32nd Annual ACM Symp. on the Theory of Computing*, pages 208–217, 2000.

[CJK+02] J. Csirik, D. S. Johnson, C. Kenyon, J. Orlin, P. W. Shor, and R. R. Weber. On the sum-of-squares algorithm for bin packing. Unpublished Manuscript, 2002.

[CL77] E. G. Coffman, Jr. and J. Y. Leung. Combinatorial analysis of an efficient algorithm for processor and storage allocation. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 214–221, 1977.

[CL86] E. G. Coffman, Jr. and F. T. Leighton. A provably efficient algorithm for dynamic storage allocation. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 77–90, 1986.

[Csi89] J. Csirik. An on-line algorithm for variable-sized bin packing. *Acta Informatica*, 26:697–709, 1989.

[Dyc90] H. Dyckhoff. A typology of cutting and packing problemsacking. *European Journal of Operational Research*, 44:145–159, 1990.

[ECG+91] E.G.Coffman, C. Courcoubetis, M. R. Garey, D. S. Johnson, L. A. McGeogh, P. W. Shor R. R. Weber, and M. Yannakakis. Fundamental discrepancies between average-case analyses under discrete and continuous distributions. In *Proceedings of the 23rd Annual Symposium on Theory of Computing*, pages 230–240, 1991.

[ESS01] L. Epstein, S. Seiden, and R. V. Stee. On the fractal beauty of bin packing. Technical Report SEN-R0104, Centrum voor Wiskunde en Informatica, 2001.

[FL86] D. K. Friesen and M. A. Langston. On variable sized bin packing. *SIAM J. Computing*, 15:222–230, 1986.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability A Guide to the Theory of NP-Completeness*. 1979.

[KL88] N. G. Kinnersley and M. A. Langston. Online variable sized bin packing. *Discrete Appl. Math.*, 22:143–148, 1988.

[KRS98] C. Kenyon, Y. Rabani, and A. Sinclair. Biases, random walks, Lyapunov functions and stochastic analysis of best fit bin packing. *J. Algorithms*, 27:218–235, 1998.

[LL81] C. C. Lee and D. T. Lee. A simple online bin packing algorithm. *J. Assoc. Comput. Mach.*, 32:562–572, 1981.

[LNO96] M. G. Luby, J. Naor, and A. Orda. Tight bounds for dynamic storage allocation. *SIAM Journal on Discrete Math.*, 9:155–166, 1996.

[Rob74] J. M. Robson. Bounds for some function concerning dynamic storage allocation. *Journal of the ACM.*, 12:491–499, 1974.