CS 860: Bounded Space On-Line Bin Packing Algorithms With Good Average Case Performance

Matthew Robertson
University of Waterloo

August 15, 2014

1 Introduction

The bin packing problem is a classic problem of computer science, and been studied extensively both in an off-line and on-line setting. This paper is interested in bounded space on-line bin packing algorithms specially with good average case performance. The bin packing problem and other basic definitions are given in Section 2 using the terminology in this paper. A brief background about the problem and relevant theorems are given in Section 3. The known algorithms in Section 4 are presented using a construction that lends itself well to describing bounded space algorithms specifically. The construction gives intuitive bounded space variants of traditionally unbounded space algorithms. The algorithms are listed in an order with relevant theorems, analysis, and experimental results that progress naturally towards the *k-Bounded Harmonic-m Match* algorithm described in Section 5. The experimental results are summarized and explained in Section 6.

2 Definitions

Definition. The *bin packing problem* is the task of *packing* a given sequence of *items* into the minimum number of unit capacity *bins*. An item is an entity with a size property. The size of an item must be greater than 0 and no more than a single unit. A bin is a container that packs items. To pack an item is to place the item into a bin. Every item must be packed in exactly one bin. Once an item has been packed, it can never be moved or duplicated. The *level* of a bin, the total size of all items packed into the bin, can never exceed a single unit. There is an endless supply of bins.

Definition. A bin packing algorithm (ALG) is an algorithm to either solve or approximate the bin packing problem. An algorithm is *on-line* if it packs items in the order with witch they are revealed, without any knowledge of the future. Each item is packed based solely on the sizes of the previous items and the levels of the *open* bins. A bin is open if the algorithm is aware of it. A *k-bounded space* algorithm (ALG_k) is an algorithm with the limitation that at no time during its operation can the number of open bins exceed k.

Definition. The *performance ratio* of an on-line algorithm is the ratio between the *cost* of the algorithm and the cost of an *optimal* algorithm. The cost of a bin packing algorithm is the total number of bins used by the algorithm to pack a given sequence of items. An optimal algorithm has unlimited computational power and knowledge of the future. The worst case performance ratio of an algorithm is called the *asymptotic competitive ratio* and is given by

$$\text{R}(\text{ALG}) = \limsup_{n\to\infty} \sup_{\sigma} \left(\frac{\text{cost}(\text{ALG}(\sigma))}{\text{cost}(\text{OPT}(\sigma))} \middle| \text{cost}(\text{OPT}(\sigma)) = n \right).$$

Definition. The expected performance ratio for a random input of items with sizes following a uniform distribution is called the *average case performance ratio* and is given by $\text{ER}(\text{ALG})$.

This paper is interested in bounded space on-line bin packing algorithms with good average case performance. Ideally, such algorithms should have the best possible worst case performance, or competitive ratio, possible for bounded space on-line algorithms.

3 Background

The bin packing problem has been studied since the early 1970's. Although this problem is known to be $\mathcal{NP}$-*complete* [6], there exists several polynomial time approximation algorithms that are very practical in reality. This paper only focuses on a portion of these algorithms; those algorithms which are on-line and use bounded space. Some unbounded space algorithms have very natural bounded space variants.

Theorem 1. No polynomial time bin packing algorithm can have a competitive ratio better than $\frac{3}{2}$ unless $\mathcal{P} = \mathcal{NP}$.

Proof of Theorem 1. If such an algorithm existed, it could be used to solve the *partition problem*, which is also known to be $\mathcal{NP}$-*complete*, in polynomial time. The partition problem is the problem of partitioning a given multiset of positive integers into two subsets such that both subsets sum up to the same total. An instance of the partition problem can be converted to an instance of the bin packing problem by creating a sequence of items with the same proportions as the given multiset, such that the items total size is exactly two units. The items will be packed into two bins if and only if the multiset can be partitioned into two sets with the same total sum. If the items can be packed into two bins, an approximation algorithm with a better competitive ratio than $\frac{3}{2}$ will definitely find that packing, because a single bin cannot possibly fit two units and 3 or more bins would give a competitive ratio of $\frac{3}{2}$ or worse. □

Theorem 2. No polynomial time on-line bin packing algorithm can have a better competitive ratio than 1.540 [8].

Theorem 3. No k-bounded space on-line bin packing algorithm ALG_k can achieve a competitive ratio better than

$$\text{R}(\text{ALG}_k) = h_\infty = \sum_{i=1}^{\infty} \frac{1}{u_i - 1} \approx 1.69103$$

where $u_1 = 2$ and $u_{i+1} = u_i(u_i - 1) + 1$ [6].

Theorem 4. No k-bounded space on-line bin packing algorithm ALG_k can have a better average case performance ratio than

$$\text{ER}(\text{ALG}_k) = 1 + \frac{1}{4(k+1)}$$

[2].

4 Algorithms

Henceforth in this paper, an algorithm refers to a polynomial time k-bounded space on-line bin packing approximation algorithm unless otherwise noted.

An algorithm (ALG_k^m) can be broken down into two components, a *classification method* (CM^m) and a *packing strategy* (PS_k^m). A CM is a method to classify an item into one of m classifications based solely on its size. A classification $\mathcal{I}_j$ is an interval of possible item sizes. Classifications must have the property that

$$\bigcup_{j=1}^{m}{\cdot} \, \mathcal{I}_j = (0, 1] \text{ units}.$$

A PS is a strategy to choose which bin to pack an item into based on its size, classification, and the current open bins. A PS strategy that is unaware of classifications is called a *simple* PS (SPS_k). The notation $\text{ALG}_k^m = (\text{CM}^m, \text{PS}_k^m)$ is used to represent an algorithm. If the algorithm uses an SPS strategy, the notation can be simplified to $\text{ALG}_k^1 = (\varnothing, \text{SPS}_k)$ or just $\text{ALG}_k = \text{SPS}_k$.

An SPS strategy and the algorithm that uses it can be though of as the same thing because such an algorithm does not have a CM method. For example, the *k-Bounded Best Fit* algorithm $\text{B-BF}_k = \text{B-BF}_k$.

4.1 Next Fit

Next Fit (NF) is a the most basic SPS strategy. In NF, the next item is packed into the current open bin. Once that bin becomes *full*, it is *flipped*. A bin is considered full when the next item to be packed would cause the bins level to exceed a single unit. To flip a bin means to close that bin, open a new bin in its place, and pack the next item into the new bin.

The NF algorithm uses 1-bounded space, has a competitive ratio $\mathrm{R}(\mathrm{NF}) = 2$ [3], and an average case performance ratio of $\mathrm{ER}(\mathrm{NF}) = \frac{4}{3} = 1.\bar{3}$ [1].

Theorem 5. The *Next Fit* algorithm has a competitive ratio of $\mathrm{R}(\mathrm{NF}) = 2$.

Proof of Theorem 5. The competitive ratio of NF is at most 2 because no more than one bin can be less than half full. The nature of NF will not flip the current open bin until it is full. If the current bin is less than half full and the next item is less than half a unit, NF will pack it into the current open bin. Therefore $\mathrm{R}(\mathrm{NF}) \leq 2$.

A sequence of n items to make NF cost twice that of OPT can easily be constructed by alternating items of sizes half a unit and ϵ, where n is a multiple of 4 and $\epsilon \leq \frac{2}{n}$. The cost of NF is $\mathrm{cost}(\mathrm{NF}) = \frac{n}{2}$ because every item of size half a unit (except the first) causes the current open bin to flip, essentially wasting half of it. An optimal packing would be to place two, half unit sized items into each bin and place all the ϵ sized items into a single bin, giving a cost of $\mathrm{cost}(\mathrm{OPT}) = \frac{n}{4} + 1$. Therefore, as $n \to \infty$, $\mathrm{R}(\mathrm{NF}) = 2$. □

The average case performance $\mathrm{ER}(\mathrm{NF})$ can be measured experimentally by repeatably running the algorithm on sequences of 1,000,000 items with a uniform distribution of sizes. The experiment is repeated 1000 times, and item sizes are represented by integers between 0 and $2^{31} - 1$. The *measured* average performance ratio is $\mathrm{MR}(\mathrm{NF}) = 1.3333$. Over all experiments, the maximum measured performance ratio $\mathrm{MR}_{\max}(\mathrm{NF}) = 1.3344$, and the minimum $\mathrm{MR}_{\min}(\mathrm{NF}) = 1.3322$. This is very close to the theoretical average case performance ratio. All measured performance ratio experiments are run similarly.

4.2 Next-*k* Fit

Next-k Fit (NF_k) is an extension of the NF strategy to allow multiple open bins. In NF_k, the next item is packed into the *oldest* of the k open bins in which it fits. If the item does not fit in any open bin, the oldest overall bin is flipped. The oldest bin is the least recently flipped bin.

The NF_k algorithm uses k-bounded space, NF_1 is equivalent to NF, and NF_∞ is equivalent to the unbounded space algorithm *First Fit* (FF). The competitive ratio of NF_k for $k \geq 2$ is

$$\mathrm{R}(\mathrm{NF}_k) = \frac{17}{10} + \frac{3}{10(k-1)}$$

[7]; worse than the competitive ratio of FF, $\mathrm{R}(\mathrm{FF}) = \frac{17}{10} = 1.7$ [4]. The results of similar MR experiments to the one performed on NF are presented below.

NF_k	$k=2$	5	10	20	40	80	160	320	640
$\mathrm{MR}_{\mathrm{avg}}$	1.2386	1.1564	1.1156	1.0858	1.0637	1.0470	1.0345	1.0252	1.0183
$\mathrm{MR}_{\max}$	1.2396	1.1572	1.1165	1.0867	1.0643	1.0477	1.0351	1.0257	1.0187
$\mathrm{MR}_{\min}$	1.2376	1.1556	1.1148	1.0852	1.0630	1.0464	1.0340	1.0247	1.0180

As k increases, $\mathrm{MR}(\mathrm{NF}_k)$ gets closer to 1, this shows for any $k > 1$, NF_k is an improvement to NF.

4.3 Best-*k* Fit

Best-k Fit (BF_k) is another SPS strategy similar to NF_k. The difference is that BF_k will place the next item into the *largest*, not oldest, open bin in which it fits. If the item does not fit anywhere, the oldest bin is still flipped.

BF_k uses k-bounded space, BF_1 is still equivalent to NF, and NF_∞ is equivalent to the unbounded space algorithm *Best Fit* (BF). The competitive ratio of BF_k is

$$\mathrm{R}(\mathrm{NF}_k) = \frac{17}{10} + \frac{3}{10k}$$

[7], slightly better than NF_k but not as good as the unbounded space algorithms FF and BF both with competitive ratio $\text{R}(\text{BF}) = \frac{17}{10} = 1.7$ [7].

4.4 *k*-Bounded Best Fit

The *k-Bounded Best Fit* (B-BF_k) strategy is very similar to BF_k. The only difference is that when the next item does not fit anywhere, B-BF_k will flip the largest, not oldest, open bin.

This small change gives B-BF_k a competitive ratio of $\frac{17}{10} = 1.7$ for any $k \geq 2$, matching the unbounded space algorithms FF and BF [2]. Experimental MR results are presented below.

B-BF_k	$k=2$	5	10	20	40	80	160	320	640
MR_{avg}	1.1783	1.1015	1.0673	1.0439	1.0280	1.0175	1.0107	1.0065	1.0041
MR_{max}	1.1790	1.1024	1.0680	1.0445	1.0287	1.0185	1.0115	1.0072	1.0049
MR_{min}	1.1776	1.1008	1.0665	1.0431	1.0273	1.0169	1.0101	1.0060	1.0035

Clearly, as k increases, $\text{MR}(\text{B-BF}_k)$ approaches 1 faster than $\text{MR}(\text{NF}_k)$. Although both of the unbounded space versions of the algorithms have $\text{ER}(\text{BF}) = \text{ER}(\text{NF}) = 1$; this experiment shows the bounded space variants $\text{ER}(\text{B-BF}_k) < \text{ER}(\text{NF}_k)$ for $k \geq 2$. This has already been known as "best is better than first" [2].

4.5 Relaxed On-Line Match

The bounded *Relaxed On-Line Match* (ROM_k) strategy is a purposed bounded space variant of *Relaxed On-Line Match* (ROM). ROM is an SPS strategy that packs *small* items with *large* items. An item is considered large if it's size is greater than half a unit, and small otherwise. If the next item is large, ROM will open a new bin and pack it there. If the next item is small, ROM will pack it with a matching item, that is pack the small item using BF into a bin that already has a large item, and close that bin. If the small item does not match anywhere, the item is packed using NF into a reserved bin. Since ROM uses unbounded space, there is no defined flipping strategy for when there is no space to pack a large item. A very natural flipping strategy for ROM_k is to flip the largest non-reserved open bin when needed to pack a large item. The MR experimental results are presented below.

ROM_k	$k=2$	5	10	20	40	80	160	320	640
MR_{avg}	1.2432	1.1448	1.0965	1.0615	1.0376	1.0224	1.0131	1.0076	1.0046
MR_{max}	1.2442	1.1456	1.0972	1.0622	1.0384	1.0232	1.0138	1.0084	1.0055
MR_{min}	1.2423	1.1441	1.0958	1.0607	1.0367	1.0217	1.0124	1.0070	1.0038

Due to the nature of ROM_k, it is possible to not utilize all of the open bins. Another purposed modification is to not close matched bins immediately but wait until they get flipped later, giving them the potential to fit several small items. This will make non-closing ROM_k (NC-ROM_k) behave similar to B-BF_k.

Theorem 6. The non-closing bounded *Relaxed On-Line Match* (NC-ROM_k) algorithm will not cost more than the normal bounded *Relaxed On-Line Match* (ROM_k) algorithm for any given sequence of items σ.

Proof of Theorem 6. ROM is *monotone*, meaning that removing an item from the sequence σ will not increase the cost of packing it [5]. In the NC-ROM_k algorithm, any item σ_j that is packed into an already matched bin is equivalent to skipping that item in ROM_k. Therefore $\text{cost}(\text{NC-ROM}_k(\sigma)) \leq \text{cost}(\text{ROM}_k(\sigma))$ for any sequence of items σ. □

The above MR experiments are repeated with the non-closing variant below.

NC-ROM_k	$k=2$	5	10	20	40	80	160	320	640
MR_{avg}	1.2338	1.1203	1.0754	1.0472	1.0293	1.0180	1.0109	1.0066	1.0041
MR_{max}	1.2347	1.1212	1.0761	1.0479	1.0301	1.0186	1.0115	1.0073	1.0049
MR_{min}	1.2329	1.1195	1.0748	1.0466	1.0286	1.0174	1.0103	1.0061	1.0036

NC-ROM$_k$ does have a better performance ratio, close to that of B-BF$_k$. But the competitive ratio of ROM$_k$ and NC-ROM$_k$ is R(ROM$_k$) = R(NC-ROM$_k$) = 2.

Theorem 7. The competitive ratio of ROM$_k$ and NC-ROM$_k$ is R(ROM$_k$) = R(NC-ROM$_k$) = 2.

Proof of Theorem 7. For any sequence of only small items, ROM$_k$ and NC-ROM$_k$ are essentially NF. The worst case of NF can be achieved with only small items. By Theorem 5, R(NF) = 2. Therefore R(ROM$_k$) ≥ 2 and R(NC-ROM$_k$) ≥ 2. Of the utilized non-reserved bins, none can be less than half full. So by an argument very similar to that in the proof of Theorem 5, R(ROM$_k$) ≤ 2 and R(NC-ROM$_k$) ≤ 2. Therefore R(ROM$_k$) = 2 and R(NC-ROM$_k$) = 2. □

4.6 Harmonic-*m*

In contrast to an SPS, a PS strategy alone does not make an algorithm; it must be accompanied by a CM method. *Harmonic-m* (HA^m) is a CM method that classifies item sizes into the m intervals

$$\mathcal{I}_j = \begin{cases} \left(\frac{1}{j+1}, \frac{1}{j}\right], & 1 \leq j < m \\ \left(0, \frac{1}{j}\right], & j = m. \end{cases}$$

A *naive* PS strategy (N-SPS$_{k/m}$) is a naive way to augment an SPS to be aware of classifications. N-SPS$_{k/m}$ will partition the k open bins into m isolated groups, and apply the SPS$_{k/m}$ strategy to each group independently.

The *Harmonic-m* algorithm is given by HA$_k^m$ = $\left(\text{HA}^m, \text{N-NF}_{k/m}\right)$ where N-NF is the naive classified version of the NF strategy, and $k = m$. Because of this, k may be omitted from the notation, giving the simpler notation HA^m = (HA^m, N-NF). In HA^m, the next item is packed into the open bin corresponding to its classification using the simple NF strategy.

Theorem 8. As m increases, the *Harmonic-m* (HA^m) algorithm has a competitive ratio that approaches R(HA^m) = $h_\infty \approx 1.69103$ [6].

Theorem 9. The *Harmonic-m* algorithm has an average case performance ratio of

$$\text{ER}(\text{HA}^m) = \sum_{j=1}^{\infty} \frac{2}{j^2(j+1)} \approx 1.2899\ .$$

Proof of Theorem 9. Each bin B_j corresponding to each classification $\mathcal{I}_j = (\frac{1}{j+1}, \frac{1}{j}]$ for each $1 \leq j < m$ can pack exactly j items before being flipped. The last bin B_m corresponding to classification $\mathcal{I}_m = (0, \frac{1}{m}]$ can pack at least m items. For a sequence σ of length n with a uniform distribution of item sizes, there is expected $n(\frac{1}{j} - \frac{1}{j+1})$ items of classification $\mathcal{I}_j$ fitting at least j items per bin. So the total cost of HA^m is expected to be

$$\text{E}(\text{cost}(\text{HM}^m(\sigma))) \leq \sum_{j=1}^{m-1} \frac{n\left(\frac{1}{j} - \frac{1}{j+1}\right)}{j} + \frac{n\left(\frac{1}{m} - 0\right)}{m} = n\left(\sum_{j=1}^{m-1} \frac{1}{j^2(j+1)} + \frac{1}{m^2}\right).$$

The cost of an an optimal algorithm is at least the sum of all the item sizes

$$\text{cost}(\text{OPT}(\sigma)) \geq \sum_{j=1}^{n} |\sigma_j|$$

where $|\sigma_j|$ is the size of item σ_j. So the expected cost of an optimal algorithm for a uniformly distributed sequence is given by

$$\text{E}(\text{cost}(\text{OPT}(\sigma))) \geq \sum_{j=1}^{n} 0.5 = \frac{n}{2}.$$

The average case performance ratio of HA^m is

$$\text{ER}(\text{HA}^m) = \frac{\text{E}(\text{cost}(\text{HM}^m(\sigma)))}{\text{E}(\text{cost}(\text{OPT}(\sigma))} \leq 2\left(\sum_{j=1}^{m-1} \frac{1}{j^2(j+1)} + \frac{1}{m^2}\right).$$

Therefore, as $m \to \infty$, the average case performance ratio of HA^m approaches

$$\text{ER}(\text{HA}^m) = \sum_{j=1}^{\infty} \frac{2}{j^2(j+1)} \approx 1.2899\ .$$

□

The results of running the MR experiment on HA^m are presented below.

HA^m	$m=2$	5	10	20	40	80	160	320	640
MR_{avg}	1.2986	1.2901	1.2899	1.2899	1.2899	1.2900	1.2900	1.2903	1.2909
MR_{max}	1.2996	1.2910	1.2908	1.2910	1.2907	1.2909	1.2910	1.2914	1.2918
MR_{min}	1.2976	1.2892	1.2889	1.2891	1.2892	1.2891	1.2892	1.2895	1.2901

Interestingly, after only $m = 10$ classifications, $\text{MR}(\text{HA}^m)$ gets very close to the average case performance ratio 1.2899, but after $m = 80$, the measured performance starts to degrade. Bins corresponding to very small items are likely to waste more space. Any $m > 6$ is enough to have a competitive ratio better than 1.7, and it has been recommended that $m \leq 12$ should be used in practice [6]. So an m value of 10 is a nice round number to use for a good guaranteed worst case, but HA^m does not have a good average case compared to B-BF_k.

4.7 Harmonic-m Match

Harmonic-m Match (HM^m) is a CM very similar to HA^{m+1}. Each classification $\mathcal{I}_j$ of HM^m for $1 \leq j \leq m$ contains the interval $\mathcal{I}_{j+1}$ of HA^{m+1} and a *matching* subinterval of $(0, \frac{1}{2}]$. The classification $\mathcal{I}_j$ of HM^m is given by

$$\mathcal{I}_j = \begin{cases} \left(\frac{1}{j+2}, \frac{1}{j+1}\right] \bigcup \left(\frac{j}{j+1}, \frac{j+1}{j+2}\right], & 1 \leq j < m \\ \left(0, \frac{1}{m+1}\right] \bigcup \left(\frac{m}{m+1}, 1\right], & j = m. \end{cases}$$

The bounded *Harmonic-m Match* (HM_k^m) algorithm is a naive bounded space variant of the unbounded space *Harmonic-m Match* (HM^m) algorithm. HM^m will naively apply an instance of the simple ROM strategy for each classification. Since HM^m is given by $(\text{HM}^m, \text{N-ROM})$, a very natural way to define the bounded space HM_k^m is $\left(\text{HM}^m, \text{N-ROM}_{k/m}\right)$. HM^m has an average case performance ratio of 1 [5]. Since HM_k^m is a k-bounded space algorithm, its average case performance ratio can not be better than $\frac{1}{4(k+1)}$.

Theorem 10. If $k \geq m+1$, then as m increases the competitive ratio of HM_k^m will approach $\text{R}(\text{HM}_k^m) = h_\infty \approx 1.69103$.

Proof of Theorem 10. Given a sequence of items σ, the cost of HM_k^m is no more than the cost of HA^{m+1} for any $k \geq m+1$. Since HM_k^m has m classifications, the N-ROM strategy has m reserved bins for small items to be packed with NF strategy. Each reserved bin of classification $\mathcal{I}_j$, $1 \leq j \leq m$ for HM_k^m corresponds to a bin of classification $\mathcal{I}_{j+1}$ for HA^{m+1}. For any sequence of only small items, HM_k^m and HA^{m+1} will pack the items exactly the same. The two algorithms are essentially the same for small items. They differ with large items.

First, consider the case $k = m+1$. The single non-reserved bin of HM_{m+1}^m corresponds to the bin of classification $\mathcal{I}_1 = (\frac{1}{2}, 1]$ for HA^{m+1}. All large items will be packed into this bin, but HM_{m+1}^m has the potential to pack a matching small item into this bin, while HA^{m+1} does not. Mark any small item in σ that matches the most recent preceding large item as *matchable*. The cost of HM_{m+1}^m on σ is equal to the cost of HA^{m+1} on σ with all the matchable items removed.

Since HA^{m+1} is monotone [5], removing the matchable items cannot increase the cost. Therefore $\text{cost}(\text{HM}^m_{m+1}) \leq \text{cost}(\text{HA}^{m+1})$.

Now, consider the case $k > m+1$. This case is just like the previous case except there are multiple non-reserved bins in HM^m_k corresponding to $\mathcal{I}_1$ of HA^{m+1}. Every large item in σ requires its own bin in both HM^m_k and HA^{m+1}, so they cost the same. For small items, everything is the same except more items in σ are matchable. By the same argument as the previous case, $\text{cost}(\text{HM}^m_k) \leq \text{cost}(\text{HA}^{m+1})$. Therefore $\text{R}(\text{HM}^m_k) \leq \text{R}(\text{HA}^{m+1})$ for any $k \geq m+1$.

By Theorem 8, $\text{R}(\text{HM}^m_k) \leq h_\infty$ and by Theorem 3, $\text{R}(\text{HM}^m_k) \geq h_\infty$. Therefore, as $m \to \infty$, $\text{R}(\text{HM}^m_k) = h_\infty$ for $k \geq m+1$.

□

The results of running the MR experiment on HM^{10}_k are presented below.

HM^{10}_k	$k=20$	40	80	160	320	640
MR_{avg}	1.2019	1.1194	1.0733	1.0447	1.0269	1.0160
MR_{max}	1.2028	1.1204	1.0743	1.0456	1.0281	1.0171
MR_{min}	1.2009	1.1184	1.0724	1.0438	1.0260	1.0150

The measured performance ratio of HM^{10}_k is actually quite bad, considering the unbounded space HM^m algorithm has an optimal average case. The poor performance is due to the N-$\text{ROM}_{k/m}$ strategy not fully utilizing all the open bins. A non-closing variant of N-$\text{ROM}_{k/m}$ would do no good because the naive strategy isolates each classification, making it impossible to match any two small items with a large item of the same classification. An unmatched bin in one classification may be flipped while free bins exist in other classifications. A more specialized PS is needed to effectively utilize all of the available space.

5 *k*-Bounded Harmonic-*m* Match

The *k-Bounded Harmonic-m Match* (B-HM^m_k) algorithm is a carefully constructed bounded space variant of the unbounded space HM^m algorithm. B-HM^m_k is given by $(\text{HM}^m, \text{S-ROM}^m_k)$ where S-ROM^m_k is a specialized augmented version of the NC-ROM_k strategy. If the next item is large, S-ROM^m_k will pack it into a new bin, flipping the largest non-reserved bin if there is no space to open a new bin. If the next item is small, the strategy will attempt to pack it into the largest, non-reserved bin, no larger than $\max(\mathcal{I}_j)$ where $\mathcal{I}_j$ is the classification of the item, in which it fits. It the item does not fit anywhere, it is packed into the m reserved bins using the HA^{m+1} strategy for small items (no bin for the classification $(\frac{1}{2}, 1]$ of HA^{m+1}).

Theorem 11. If $k \geq m+1$, then as m increases the competitive ratio of B-HM^m_k will approach $\text{R}(\text{B-HM}^m_k) = h_\infty \approx 1.69103$.

Proof of Theorem 11. The proof of Theorem 10 applies to any bounded space variant of the HM^m algorithm that has a reserved bin for each small classification of HA^{m+1} and at least 1 non-reserved bin for large items. Therefore the result also applies to B-HM^m_k for $k \geq m+1$. □

The results of running the MR experiment on B-HM^{10}_k are presented below.

B-HM^{10}_k	$k=11$	20	40	80	160	320	640
MR_{avg}	1.2251	1.0689	1.0346	1.0195	1.0114	1.0067	1.0041
MR_{max}	1.2258	1.0697	1.0353	1.0203	1.0120	1.0073	1.0050
MR_{min}	1.2241	1.0683	1.0340	1.0189	1.0107	1.0062	1.0035

The B-HM^{10}_k algorithm performances quite well. As k increases, B-HM^m_k essentially becomes closer and closer to B-BF_k while maintaining a better competitive ratio. Every time a small item is matched with a large item, the new level of the bin can be considered a single large item of a different classification, to be matched with an even smaller small item. By the time a bin is flipped, it has had the opportunity to match several small items. The classifications in HM^m span different proportions of the interval $(0, 1]$, so B-HM^m_k can expect different proportions of bins for each classification. Allowing these proportions to change dynamically is what stops B-HM^m_k from wasting bins and allows it to utilize a non-closing PS strategy.

6 Experimental Results

All the previous MR experimental results are listed in the table below.

MR_{avg}	$k=2$	5	10	20	40	80	160	320	640
NF_k	1.2386	1.1564	1.1156	1.0858	1.0637	1.0470	1.0345	1.0252	1.0183
B-BF_k	1.1783	1.1015	1.0673	1.0439	1.0280	1.0175	1.0107	1.0065	1.0041
ROM_k	1.2432	1.1448	1.0965	1.0615	1.0376	1.0224	1.0131	1.0076	1.0046
NC-ROM_k	1.2338	1.1203	1.0754	1.0472	1.0293	1.0180	1.0109	1.0066	1.0041
$\text{HA}^{m=k}$	1.2986	1.2901	1.2899	1.2899	1.2899	1.2900	1.2900	1.2903	1.2909
HM_k^{10}	-	-	-	1.2019	1.1194	1.0733	1.0447	1.0269	1.0160
B-HM_k^{10}	-	-	-	1.0689	1.0346	1.0195	1.0114	1.0067	1.0041

The B-BF_k algorithm is clearly the winner in terms of average case performance for all the measured k values, but has a competitive ratio of $\text{R}(\text{B-BF}_k) = 1.7$. The runner up is the NC-ROM_k algorithm, which has an even worse competitive ratio of $\text{R}(\text{NC-ROM}_k) = 2$. However, the B-HM_k^{10} algorithm comes in a strong third place and has a better competitive ratio of $\text{R}(\text{B-HM}_k^{10}) \approx 1.6910$.

7 Conclusion

The B-HM_k^m algorithm performs almost as well as B-BF_k on average, but has a better worst case performance. B-HM_k^m can be implemented almost as easily as B-BF_k, and in the same time and space complexity. As k increases, the difference between B-HM_k^m and B-BF_k gets less and less significant. However, an m value of 10 is still recommended for the same reason as explained for HA^m. Therefore this paper recommends any application that is current using B-BF_k to switch to B-HM_k^m for essentially the same performance on average, but better worst case performance.

Future research could be done on the performance of B-HM_k^m on sequences of items with sizes following non-uniform distributions. Because of the way B-HM_k^m allows the proportions of bins corresponding to each classification to dynamically change, it probably performs quite well on almost any distribution.

References

[1] Edward G Coffman Jr, Kimming So, Micha Hofri, and AC Yao. A stochastic model of bin-packing. *Information and Control*, 44(2):105–115, 1980.

[2] János Csirik and David S Johnson. Bounded space on-line bin packing: best is better than first. In *SODA*, pages 309–319, 1991.

[3] David S Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8(3):272–314, 1974.

[4] David S. Johnson, Alan Demers, Jeffrey D. Ullman, Michael R Garey, and Ronald L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974.

[5] Shahin Kamali and Alejandro López-Ortiz. An all-around near-optimal solution for the classic bin packing problem. *arXiv preprint arXiv:1404.4526*, 2014.

[6] Chan C Lee and Der-Tsai Lee. A simple on-line bin-packing algorithm. *Journal of the ACM (JACM)*, 32(3):562–572, 1985.

[7] W Mao. Besk-k-fit bin packing. *Computing*, 50(3):265–270, 1993.

[8] André van Vliet. An improved lower bound for on-line bin packing algorithms. *Information processing letters*, 43(5):277–284, 1992.