How to Pack Better than Best Fit: Tight Bounds for Average-Case On-Line Bin Packing

Peter W. Shor

AT&T Bell Laboratories, Room 2D-149
Murray Hill, NJ 07974

Abstract

In this paper, we give an $O(n\ \log n)$-time on-line algorithm for packing items i.i.d. uniform on $[0,1]$ into bins of size 1 with expected wasted space $\Theta(n^{1/2}log^{1/2}n)$. This is better than the algorithm Best Fit, which gives $\Theta(n^{1/2}log^{3/4}n)$ expected wasted space, and matches the lower bound that no on-line algorithm can achieve $o(n^{1/2}log^{1/2}n)$ wasted space. We do this by analyzing another algorithm which involves putting balls into buckets on-line. The analysis of this second algorithm also gives us bounds on the stochastic rightward matching problem, which arises in analyzing not only the above on-line bin packing problem, but also a two-dimensional problem of packing rectangles into a half-infinite strip. Our bounds on rightward matching thus give good bounds for the two-dimensional strip packing problem.

1 Introduction

The bin packing problem is: given n items of size at most 1, pack them into the minimum number of bins of capacity 1 so that no bin has its capacity exceeded. This is an NP-complete problem which has received much study. Since it is difficult to find exact solutions to NP-complete problems, research on bin packing has concentrated on the performance of heuristics. One approach is to find algorithms that produce packings that are provably not much worse than the optimal. The best result to date along these lines is due to Karmarkar and Karp, who give a polynomial-time algorithm that never uses more than OPT $+ O(\log^2 \text{OPT})$ bins [KK], where OPT is the number of bins used in the optimal packing. Although this approach works well for *off-line* algorithms, where you are able to see all the items before starting packing, one cannot obtain comparable results for *on-line* algorithms, where you must pack each item before you learn the sizes of future items. In fact, Brown and Liang independently found a lower bound of 1.536 OPT for any on-line bin packing algorithm [B,L].

Another approach is to analyze how well algorithms do in a typical case. To do this, one must first define a typical case. One way to do this for bin packing is to pick a distribution of item sizes, and assume that the items are i.i.d. with that distribution. An obvious distribution is the uniform distribution on $[0,1]$. This was one of the first distributions studied, and is the distribution which has received the most study. It turns out that this distribution is substantially easier to pack than most distributions, so it is probably not a good test bed for the typical performance of algorithms. Nevertheless, it is at least a distribution that lends itself to analysis, and it does give insight into the efficiency of algorithms, especially when combined with results for other distributions.

The research on average-case analysis of on-line algorithms began with Coffman, Hofri, So and Yao [CHSY], who showed that the expected number of bins used by Next Fit is asymptotically 4/3 OPT. This was continued by Karmarkar [K], who analyzed the performance of Next Fit for a broader class of distributions. Lee and Lee [LL] then analyzed the Harmonic Fit algorithm, which uses 1.29 OPT bins asymptotically. Several variants of Harmonic Fit have since been used to improve this ratio to less than 1.19 OPT [RT1].

The first indication that simple on-line algorithms could be asymptotically optimal arose in massive computer experiments performed by Bentley, Johnson, Leighton and McGeoch [BJLM]. These experiments

CH3062-7/91/0000/0752$01.00 © 1991 IEEE

led to a proof that First Fit uses $O(n^{4/5})$ expected wasted space, where wasted space is the total capacity of bins used less the sum of the item sizes [BJLMM]. (For comparison, the same techniques give an $O(n^{2/3})$ bound for Best Fit.) Tight bounds have since been obtained for the performance of Best Fit and First Fit: Best Fit uses $\Theta(n^{1/2}\log^{3/4} n)$ wasted space [S,LS] and First Fit uses $\Theta(n^{2/3})$ wasted space [S,CCG*]. It was also shown that for the uniform distribution, there is an $\Omega(n^{1/2}\log^{1/2} n)$ bound on the expected wasted space of any open-ended on-line algorithm [S], where *open-ended* means that the algorithm does not know exactly how many items it will receive. More specifically, it was shown that any on-line algorithm must have an expected value of $\Omega(n^{1/2}\log^{1/2} n)$ for the average wasted space during the packing of the first n items. Surprisingly, there is an (non open-ended) on-line algorithm which achieves $\Theta(n^{1/2})$ wasted space after n items (although it has $\Theta(n)$ average wasted space while packing the first n items).

These results raised an interesting question: is there an open-ended on-line algorithm which uses asymptotically less expected wasted space than Best Fit? The $\log^{1/4} n$ gap between the wasted space used by Best Fit and the lower bound for open-ended on-line algorithms left open the possibility of such an algorithm. On the other hand, it seemed difficult to devise an algorithm which would do better than Best Fit. In this paper, we give an algorithm which uses $\Theta(n^{1/2}\log^{1/2} n)$ expected wasted space.

The result of $\Theta(n^{1/2}\log^{3/4} n)$ wasted space for Best Fit comes from the bound for unmatched points in the *up-right matching problem* [S,LS]. This matching problem not only solved the problem of Best Fit bin packing, but also had widespread applications in the average-case analysis of algorithms, and answered several open questions in probability theory [CL1,LS,RT2,SY]. Intuitively, the up-right matching problem is important because it gives a measure of how close n uniform random points chosen in the unit square come to being evenly distributed in the square. The analogous result for one dimension is the well-known and important result that if a fair coin is flipped n times, the expected difference between the number of heads and the number of tails at any time is $\Theta(\sqrt{n})$. Rightward matching gives a different measure of how far away points chosen uniformly in a square are from being evenly distributed. This paper approaches the rightward matching problem in a way that not only gives a tight bound of $\Theta(n^{1/2}\log^{1/2})$, but also shows that we can obtain this behavior on-line. What we mean by this will become clear later when we discuss these issues in detail.

As with up-right matching, our results also imply results for two other open problems: the analysis of rightward matching and the design of strip packing algorithms. We believe that other average-case analyses could also follow from these results.

Consider the following five problems:

A) Items that are i.i.d. uniform on $[0,1]$ must be packed into bins of size 1. The items arrive on-line; that is, you must pack each item before learning the size of the following items. How many bins do you need?

B) You have n buckets, labeled 1, 2, ..., n, and you receive balls labeled i.i.d. uniform with the integers between 1 and n. A ball labeled i may go only into bucket i or bucket $i+1$ (mod n). Your goal is to keep the number of balls in the buckets as close to equal as possible. How do you pack the balls?

C) You are given n rectangles, whose sides are uniform i.i.d. on $[0,1]$. You want to pack them into a unit-width strip having height as small as possible. You furthermore want to pack them in a *level* packing, that is, you want all the items to be on "shelves," as in Figure 1. How long a strip do you need?

D) Points of two colors, n red and n blue, are uniformly distributed in the unit square. You wish to match each blue point to either a red point to its right or to the bottom edge of the square, minimizing the total sum of the edge lengths in such a matching. What is the expected sum of the edge lengths?

E) Again, n red points and n blue points are uniformly distributed in the unit square. Consider the class $\mathcal{F}$ of functions $F(x,y)$ that map the unit square into $[0,1]$ such that $F(x,y_1) \geq F(x,y_2)$ if $y_1 < y_2$, and $|F(x_1,y)-F(x_2,y)| \leq |x_1-x_2|$;

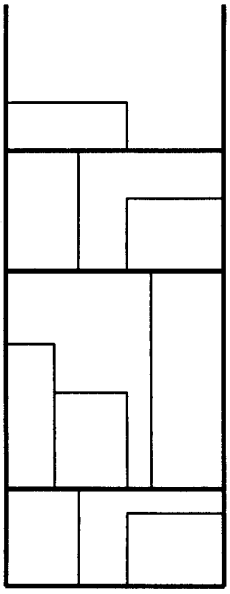

Figure 1: A level packing with four shelves.

in words, F is decreasing in y and satisfies a Lipschitz condition in x. Consider the function in $\mathcal{F}$ that minimizes the following quantity, which we call the discrepancy of F:

$$\text{discrep}(F) = \sum_{P \text{ red}} F(x_P, y_P) - \sum_{P \text{ blue}} F(x_P, y_P).$$

What is the expected value of the quantity $\min_{F \in \mathcal{F}} \text{discrep}(F)$?

Although these problems may look quite different at first sight, they are actually closely related. Problem (A) is the on-line bin packing problem discussed above. In the later sections of this paper, we will give an algorithm for problem (B), and then use this algorithm to infer improved bounds for all of the above problems.

2 The balls and buckets problem

In this section we discuss the basic problem (B in the preceding list) which we use to obtain all of our other results. We believe that, besides its use in several applications, the problem is interesting in its own right. In particular, there is a very simple greedy algorithm for this problem which seems to be far from optimal. In placing the *last* ball, however, the greedy algorithm *is* optimal; the optimal algorithm for placing a predetermined number of balls must thus become progressively "greedier" as time proceeds.

We will give two algorithms for this problem: an algorithm that produces expected square discrepancy $O(n \log^3 n)$ and is relatively easy to analyze, and an algorithm which produces expected square discrepancy $O(n \log^2 n)$ but is harder to analyze. In this draft, we give a complete analysis of the first algorithm, but only a brief sketch of the analysis of the second algorithm.

The balls and buckets problem is: we are given n buckets, labeled $1, 2, \ldots, n$. We receive balls labeled i.i.d. uniform with the integers from 1 to n. As soon as we get a ball, we must put it in a bucket, and we may put a ball labeled i only into bucket i or $i+1$ (mod n). We receive a total of kn balls, and our goal is to minimize the *square discrepancy,* which is the sum

$$\sum_{i=1}^{n} (b(i) - k)^2,$$

where $b(i)$ is the number of balls in bucket i. Note that bounds on the *discrepancy* $\sum_{i=1}^{n} |b(i) - k|$ can be obtained from bounds on the square discrepancy, since

$$\sum_{i=1}^{n} |b(i) - k| \leq n \left(\frac{1}{n} \sum_{i=1}^{n} (b(i) - k)^2 \right)^{\frac{1}{2}}.$$

The *random* strategy is to put a ball labeled i into bucket i or $i+1$ with probability $\frac{1}{2}$. It is easy to show that the expected value for the square discrepancy under the random strategy is $\Theta(nk)$.

Each ball has a choice of two possible buckets it can go into. The *greedy* strategy is to put the ball into whichever of these two buckets has the least number of balls in it, and to decide randomly if the buckets contain an equal number of balls. Experimentally,

the greedy strategy seems to give $\Theta(nk^{1/2})$ expected square discrepancy when $k < n^2$ and $\Theta(n^2)$ expected square discrepancy for $k \geq n^2$. We have heuristic arguments for why this should be true, but we do not have a rigorous proof of these bounds.

We now describe a strategy which gives expected square discrepancy $O(n \log^3 n)$ for any k. We can modify this strategy to give $O(n \log^3 k)$ expected square discrepancy for $k < n$, but the $O(n \log^3 n)$ strategy is simpler, so we shall discuss it first. We thank Joel Spencer for helping find the formulation of this strategy and the analysis that are given in the next section.

2.1 The Spencer strategy

In describing this strategy, we will assume that the number of buckets n is 2^m for some integer m. It is not difficult to eliminate this assumption, but the description of our algorithm becomes more complicated. Furthermore, for all of our applications, we can make the simplifying assumption that $n = 2^m$.

Arrange the buckets in a binary tree, so that buckets labeled $1, 2, \ldots, n/2$ are on the left half of the tree, and buckets $n/2+1, \ldots, n$ are on the right half of the tree. Each node has two subtrees, with the left subtree containing the first half of the buckets under that node and the right subtree containing the remaining ones. Let us label the root of this tree R, and let us add a further node Z above the root. For a node X of the tree, let level(X) be the level of X in the tree, where leaves have level 0, the root R has level $m-1$, and node Z has level m. For each node $X \neq Z$, we define a *left ancestor* and a *right ancestor*. For a node X, consider the path from X to the root R. The *left ancestor* of X, $l(X)$, is the first node on this path (going X to R) whose child on this path is a right child. If there is no such node, we put $l(X) = Z$. The *right ancestor* $r(X)$ is similarly the first node on this path whose child on this path is a left child, or Z if no such node exists. (See Figure 2.) Note that either $l(X)$ or $r(X)$ is the parent of X.

We are now ready to define the algorithm. Suppose that we have already placed some balls in the buckets. For each non-leaf node of the binary tree, there is exactly one label for a ball which could be placed in either the left or the right subtree of this

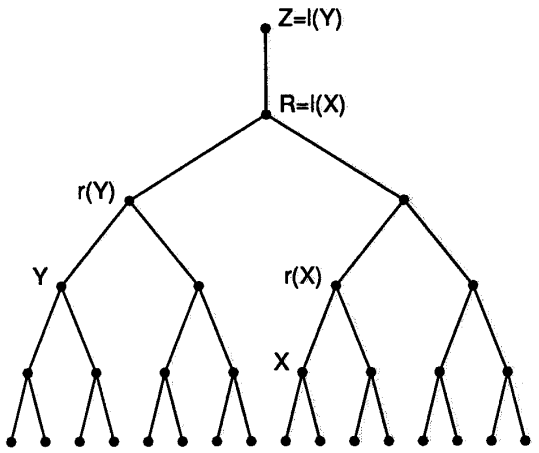

Figure 2: A binary tree showing $l(X)$ and $r(X)$

node (this is the label of the rightmost bucket in the left subtree). On each non-leaf node of the binary tree we put a probability, which will be the probability of putting a ball with this label in the right subtree. A ball labeled n corresponds to node Z. We will assign node Z the probability $1/2$, which means we will put a ball labeled n into bucket 1 or n with equal probability. Now, we inductively assign probabilities to the remaining nodes. We assign the root node R probability $p(R) = 1/2 \pm 1/2m$ depending on whether the buckets in left subtree or the buckets in right subtree of R contain more balls. (Recall that $m = \log_2 n$.) If the number of balls is equal, we decide at random between $1/2 \pm 1/2m$. Now, suppose we have a node X, and have assigned probabilities to all ancestors of X. The probability assigned to X will be $p(X) = \frac{1}{2}(p(l(X)) + p(r(X))) \pm 1/2m$ depending on whether the left subtree of X or the right subtree of X contains more balls. If the number of balls is equal, we decide at random between adding and subtracting $1/2m$. (See Figure 3 for an example with $m = 3$.)

We are now ready to analyze this algorithm. Consider a single node, and let us look at the imbalance in the number of balls in its left subtree and its right subtree. Let us denote the number of balls in the left subtree of X by $b_l(X)$ and the num-

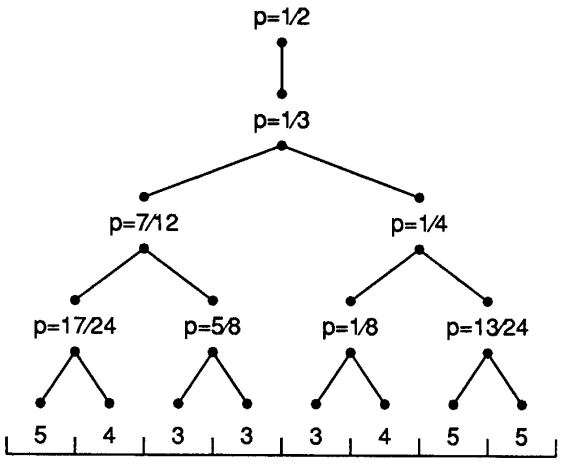

Figure 3: The probabilities on the tree nodes

ber of balls in the right subtree of X by $b_r(X)$. We add a ball to the left subtree with probability $[2^{\text{level}(X)} + p(l(X)) - p(X)]/n$ and to the right subtree with probability $[2^{\text{level}(X)} + p(X) - p(r(X))]/n$. Let us consider $b_l(X) - b_r(X)$ as a random walk. With probability $[2^{\text{level}(X)} + p(l(X)) - p(X)]/n$ we increase $b_l - b_r$ by 1 and with probability $[2^{\text{level}(X)} + p(X) - p(r(X))]/n$ we decrease it by 1. Since $p(l(X)) + p(r(X)) - 2p(X) = \pm 1/m$ depending on whether $b_l - b_r$ is negative or positive, this is a random walk with a bias of $1/mn$ towards the origin. However, a step is taken in this random walk with probability approximately $2^{\text{level}(X)+1}/n$, giving a relative bias of approximately $2^{-\text{level}(X)}/m$. It is easy to show that such a random walk has expected mean square distance from the origin $\Theta(2^{2\text{level}(X)}m^2)$. Thus, $\mathrm{E}[(b_l(X) - b_r(X))^2] = \Theta(2^{2\text{level}(X)}m^2)$.

Now, we use the identity (which can be shown by induction)

$$\sum_{i=1}^{n}(b(i) - k)^2 = \sum_{X} 2^{-\text{level}(X)}(b_l(X) - b_r(X))^2,$$

where the sum over X is the sum over all non-leaf nodes of the tree excluding Z. Recall that $b(i)$ is the number of balls in bucket i and that k is the average number of balls in a bucket. Substituting our expression for $\mathrm{E}[(b_l(X) - b_r(X)^2]$ gives

$$E\left[\sum_{i=1}^{n}(b(i) - k)^2\right] = \Theta(nm^3),$$

as desired.

In implementing this strategy, we can store the number of balls contained in every subtree. It is easy to see that when we add a ball we need only look at and update the path from the root to the bucket in which the ball is put. Our algorithm can thus be implemented in $O(n \log n)$ time. The same will hold for the more sophisticated algorithm sketched in the next section.

2.2 Improving the Spencer strategy

To improve the Spencer strategy, we would like to increase the bias returning the random walks towards the origin. Intuitively, instead of taking as our probabilities

$$p(X) = \tfrac{1}{2}\left(p(l(X)) + p(r(X))\right) \pm 1/2m,$$

we would like to take

$$p(X) = \tfrac{1}{2}\left(p(l(X)) + p(r(X))\right) \pm c_1/\sqrt{m},$$

where c_1 is a constant. This does keep the random walks closer to 0, as desired, but has the unfortunate side effect of possibly assigning probabilities of more than 1 or less than 0. We choose $c_1/\sqrt{m}$ as the bias so as to obtain the property that the probability on a node is *usually* between 0 and 1. This leaves the problem of how to keep it between 0 and 1 in the remaining cases. This is the hard part of this proof, and will only be sketched briefly in this abstract.

To keep all the probabilities between 0 and 1, intuitively what we do is use the large bias of $c_1/\sqrt{m}$ for node X if $p(X)$ is close to $\frac{1}{2}$ and otherwise use a smaller bias. We show that this gives most of the nodes of the tree the large bias, and that the remaining smaller bias nodes do not affect the expected square discrepancy.

To decide the exact bias for a node, consider the path from the root to a node X. Let $d(X)$ be the difference between the number of nodes biased to the left and biased to the right on this path. Note that at any stage of our algorithm, each node has an equal

probability of being biased to the left and to the right, so that $d(X)$ is the outcome of a random walk of $m - \text{level}(X)$ steps, and thus $|d(X)|$ has expected value $\Theta(\sqrt{m - \text{level}(X)})$. To decide the bias for a node Y, we let $\tilde{d}(Y)$ be the maximum $d(X)$ over all nodes X on the path from the root to Y. Now, if $(j-1)\sqrt{m} < \tilde{d}(Y) \leq j\sqrt{m}$, we let the bias for node Y be $c_1/(j^2\sqrt{m})$. Since $\sum_1^\infty 1/j^2$ converges, we can choose c_1, so as to keep the probabilities between 0 and 1.

We must now show that the small bias nodes do not affect the expected square discrepancy. Since the expected distance from the origin at a node with bias $c_1/(j^2\sqrt{m})$ is only j^2 times the expected distance from the origin at a node with large bias $c_1/\sqrt{m}$, and there are exponentially few nodes with small bias, this would be relatively easy if we could use just the current bias to estimate the distance from the origin in our random walk. Unfortunately, we must also account for the nodes that have recently had a small bias and now have a large bias, but where the random walk has not yet had enough time to get close to the origin again. To do this we again use the fact that at any stage of our algorithm, the bias of a node is equally likely to be to the left or to the right, as well as the fact that the direction of the bias on any node is independent of the direction of the bias for any other node. This means that we can calculate the distribution of the bias for any node, and that this distribution is independent of the stage of the algorithm. Putting all this together properly gives the desired bound on expected square discrepancy.

We now sketch the proof in more detail. We are looking at the imbalance at a node X of our tree. For simplicity, let $\text{level}(X) = 1$, so X is just above a leaf node. Recall we model the behavior of the imbalance at X as a random walk on the non-negative integers, with the probability of moving towards the origin equal to $\frac{1}{2} + c_1/j^2\sqrt{m}$, where j is determined by a random process. At any given time t, the value of j is determined by the signs of the bias on the path from R to X, so the distribution of j is the same as the maximum excursion on a random walk with $m-1$ steps. (The value of j at time t, however, is highly dependent on its value at $t-1$; this fact makes the analysis non-trivial.) We can calculate the distribution of j; what we use is that there exists a constant c such that with probability at least $1 - \exp(-cj^2)$ this excursion is at most $j\sqrt{m}$ [CL2, p. 20]. Putting this together with the definition of the bias, we find that with probability at least $1 - \exp(-cj^2)$, the bias toward the origin at any step of the random walk is at least $c_1/(j^2\sqrt{m})$. A little manipulation shows that there exists a constant c such that

$$\text{Prob}\left(\text{bias} \leq \frac{1}{2^k\sqrt{m}}\right) \leq \exp(-c2^k), \quad k \geq 0.$$

Now, let $p_i(t)$ be the probability that the random walk has value at least $2^i\sqrt{m}$ at time t. We bound $p_i(t)$ by induction. Eventually we will get the formula $p_i(t) \leq p_{i+1}(t - 2^{3i/2}m) + O(\exp(-c2^{i/2}))$, which gives by induction exponentially small bounds on $p_i(t)$ independent of t. To get these bounds, we consider the random walk at time t. This walk can exceed $2^i\sqrt{m}$ at time t in one of three ways. It could start higher than $2^{i+1}\sqrt{m}$ at time $t - 2^{3i/2}m$; this case has probability $p_{i+1}(t - 2^{3i/2}m)$. It could hit 0 between time $t - 2^{3i/2}m$ and time t and then come back and reach $2^i\sqrt{m}$; the probability it does this is less than the probability an unbiased random walk ever hits $\pm 2^i\sqrt{m}$ at some time between 0 and $2^{3i/2}m$, which is $O(\exp(-c2^{i/2}))$. It could also start below $2^{i+1}\sqrt{m}$, never hit 0, and end up higher than $2^i\sqrt{m}$. We bound the probability for this third case by dividing it into two subcases. If the sum of the biases on the random walk between time $t - 2^{3i/2}m$ and time t exceeds $2^{i+1}\sqrt{m}$, the probability the walk exceeds $2^i\sqrt{m}$ at time t is $O(\exp(-c2^{i/2})$ by Hoeffding's bound [CL2 p. 19]. Finally, the probability that the sum of the biases is small is $O(\exp(-c2^{i/2})$; this can be derived from our bound on the probability that the bias is small for a single step.

3 On-line bin packing

In this section, we show how the balls and buckets problem implies a good on-line bin packing algorithm. Recall that we are packing items uniform on $[0,1]$ into bins of size 1. Let us at first assume that we are packing N items. We classify the items into large items (those greater than $1/2$) and small items (those less than $1/2$). We then further subclassify each of these groups of items. Let $n = N^{1/2}/\log^{1/2} N$. Divide the interval $[0, 1/2]$ into n subintervals each of

size $1/2n$ and classify the small items according to which subinterval they lie in. Let S_i be the subinterval $[(i-1)/2n, i/2n)$. Similarly, divide the interval $(1/2, 1]$ into subintervals, and let L_i be the subinterval $[1-i/2n, 1-(i-1)/2n]$.

We can now describe our packing algorithm. We divide the bins into $n+3$ classes, labeled B_0, B_1, ..., B_{n+2}. Our algorithm will put at most one large item and one small item into each bin. We will put items from subinterval L_i only into bins of class B_{i-1} or B_i and items from subinterval S_i only into bins of class B_{i+1} or B_{i+2}. This ensures that an item from subinterval L_i will get paired with an item from subinterval S_{i-1} or smaller, so that the sizes of the two items will sum to less than 1. For this matching of items to bin classes, we use our balls in buckets algorithm, slightly modified to take care of edge effects (since we are no longer allowed to wrap-around and put a ball labeled n into bucket 1). We simultaneously run two such algorithms, one corresponding to the small items and the other corresponding to the large items. In the algorithm corresponding to the small items, bucket i will correspond to the set of bins B_i, and the number of balls in bucket i will be the total number of small items we have put into bins of class B_i; and similarly for the large items. It is relatively easy to check that the number of bins with only one item in them is (excluding edge effects) at most the sum of the mean discrepancies of the two balls in buckets algorithms.

We must now account for wasted space. Since the expected size of an item is $1/2$, and we never put more than two items in a bin, the expected wasted space will be half the expected number of bins with one item in them. There will be $O(N/n) = O(N^{1/2}\log^{1/2} N)$ wasted space due to edge effects (i.e., wasted space in bin classes B_0, B_1, B_{n+1} and B_{n+2}) and $O(n \log n) = O(N^{1/2}\log^{1/2} N)$ wasted space due to the discrepancy of our balls in buckets algorithm.

This algorithm is not really open-ended, i.e., the number of items we will receive, N, is used in the description of the algorithm. To make an open-ended algorithm, we dynamically change N every time that the number of items we have received doubles. Even if we start over with a new set of bins each time we change N, and leave the previously half-filled bins forever half-empty, we still get $O(N^{1/2}\log^{1/2} N)$ expected wasted space.

4 Rightward matching

The rightward matching problem, problem (D) in the list in Section 1, is as follows. You are given n points each of two colors, red and blue, uniformly distributed in the unit square. You wish to match a blue point to either the bottom of the square or to a red point to its right, minimizing the sum of the vertical lengths of the edges in the matching. What is the expected sum of vertical components of the edges?

This problem is a lower bound on the expected wasted space of an on-line bin packing algorithm [S]. In other words, because we give an upper bound on the wasted space in an on-line bin packing algorithm, we get an upper bound on the length of the edges in the optimal rightward matching. We can also use our algorithm for the balls and boxes problem to directly give an algorithm for matching the points to obtain an $O(n^{1/2}\log^{1/2} n)$ matching. Talagrand has also proved the same bound on the rightward matching problem, using a completely different method [T]. His method, although much more general and deep than the one in this paper, is not on-line, and so does not give the improved bound on bin packing. A corresponding lower bound on rightward matching can be obtained from work of Ajtai, Komlós and Tusnády [AKT,S], thus showing that the length of an optimum rightward matching is $\Theta(n^{1/2}\log^{1/2} n)$.

A bipartite matching problem, such as the rightward matching problem, can be expressed as a linear program. The dual to the rightward matching problem, considered as a linear program, is essentially problem (E) in the list in Section 1. This problem is actually the one analyzed by Talagrand. Since the value of the solutions to the primal and the dual linear programs are equal, the discrepancy of an optimum function in problem (E) is again $\Theta(n^{1/2}\log^{1/2} n)$.

If you find the rightward matching that minimizes the total sum of the edge lengths instead of the sum of just the vertical components, you can show that the expected value of this sum will not change by more than a constant factor. This fact gives a $\Theta(n^{1/2}\log^{1/2} n)$ bound for the sum of the edge lengths in a rightward matching. We can use this to give an algorithm for strip packing in the following section.

5 Strip packing

The strip packing problem is: Given n rectangular items, with sides i.i.d. uniform on $[0, 1]$, pack them into a unit-width strip having height as small as possible. Furthermore, we want a *level packing*, which is a packing in which the strip can be divided into "shelves" by horizontal lines which do not pass through any item, and where the items on each shelf are packed horizontally, i.e., a vertical line through that shelf cuts at most one item. (See Figure 1.)

To use rightward matching to obtain a level packing with $\Theta(n^{1/2}\log^{1/2} n)$ wasted space, divide the items into two classes, those with width greater than $1/2$, and those with width less than $1/2$. Now associate each item with a point in the rectangle with height 1 and width $1/2$. For an item with width $w < 1/2$ and height h, put a blue point with coordinates (w, h) in the rectangle; for width $w \geq 1/2$ and height h, put a red point with coordinates $(1 - w, h)$. Now, given a rightward matching between blue and red points with total edge length $\Theta(n^{1/2}\log^{1/2} n)$, putting items paired in the matching on the same shelf produces a packing with total wasted space $\Theta(n^{1/2}\log^{1/2} n)$. There is a lower bound of $\Omega(n^{1/2}\log^{1/2} n)$ wasted space for this problem [CS], so this algorithm is optimal up to a constant factor in wasted space.

6 Acknowledgements

I would like to thank Joel Spencer for help in coming up with what I call the Spencer strategy and in analyzing its behavior. I would also like to thank Ed Coffman and Michel Talagrand for useful discussions.

7 References

[AKT] M. Ajtai, J. Komlós, and G. Tusnády, "On optimal matchings," *Combinatorica*, **4**, pp. 259–264, 1983.

[BJLM] J.L. Bentley, D.S. Johnson, F.T. Leighton and C.C. McGeoch, "An experimental study of bin packing," *Proc. 21st Ann. Allerton Conference on Communication, Control and Computing*, U. Illinois, Urbana IL, pp. 51–60, 1983.

[BJLMM] J.L. Bentley, D.S. Johnson, F.T. Leighton, C.C. McGeoch and L. McGeoch, "Some unexpected expected behavior results for bin packing," *Proc. 16th ACM STOC*, pp. 279–288, 1984.

[B] D.J. Brown, "A lower bound for on-line one-dimensional bin packing algorithms," Technical Report R-864, Coordinated Science Laboratory, U. of Illinois, Urbana, IL, 1979.

[CCG*] E.G. Coffman, Jr., C. Courcoubetis, M.R. Garey, D.S. Johnson, L.A. McGeoch, P.W. Shor, R. Weber and M. Yannakakis, "Fundamental discrepancies between average-case analyses under discrete and continuous distributions: A bin packing case study," *Proc. 23rd ACM STOC*, pp. 230–241, 1991.

[CHSY] E.G. Coffman, Jr., M. Hofri, K. So and A.C. Yao, "A stochastic model of bin packing," *Information and Control*, **44**, pp. 105–115, 1980.

[CL1] E.G. Coffman, Jr. and F.T Leighton, "A provably efficient algorithm for dynamic storage allocation," J. Comput. System Sci., **38**, pp. 2–35.

[CL2] E.G. Coffman, Jr. and G.S. Lueker, *Probabilistic Analysis of Packing and Partitioning Algorithms*, John Wiley & Sons, New York, 1991.

[CS] E.G. Coffman, Jr. and P.W. Shor, "Packings in two dimensions: asymptotic average-case analysis of algorithms," *Math. of O. R.*, to appear.

[K] N. Karmarkar, "Probabilistic analysis of some bin-packing algorithms," *Proc. 23rd IEEE FOCS*, pp. 107–111, 1982.

[KK] N. Karmarkar and R.M. Karp, "An efficient approximation scheme for the one-dimensional bin packing problem," *Proc. 23rd IEEE FOCS*, pp. 312–320, 1982.

[LL] C.C. Lee and D.T. Lee, "A new algorithm for on-line bin packing," Technical Report 83-03-FC-02, Dept. of E. E. and Comp. Sci., Northwestern Univ., Evanston, IL 1983.

[LS] F.T. Leighton and P.W. Shor, "Tight bounds for minimax grid matching, with applications to the average-case analysis of algorithms," *Combinatorica*, **9**, pp. 161–187, 1989.

[L] F.M. Liang, "A lower bound for on-line bin packing," *Information Processing Letters*, **10**, pp. 76–79, 1980.

[RT1] P. Ramanan and K. Tsuga, "Average-case analysis of the modified harmonic algorithm," *Algorithmica*, **4**, pp. 519-533, 1989.

[RT2] W.T. Rhee and M. Talagrand, "Exact bound for the stochastic upward matching problem," *Trans. Amer. Math. Soc.*, **307**, pp. 109–125, 1988.

[S] P.W. Shor, "The average-case analysis of some on-line algorithms for bin packing," *Combinatorica*, **6**, pp. 179–200, 1986.

[SY] P.W. Shor and J. Yukich, "Minimax grid matching and empirical measures," *Ann. Probab.*, to appear.

[T] M. Talagrand, "Matching theorems and empirical discrepancy computations using majorizing measures," manuscript.