Partially Dynamic Bin Packing Can Be Solved Within $1+\epsilon$ in (Amortized) Polylogarithmic Time.[1]

Zoran Ivković[2] Errol L. Lloyd[3]

Abstract

The problem of maintaining an approximate solution for one–dimensional bin packing when items may arrive dynamically is studied. In accordance with various work on dynamic algorithms, and in contrast to prior work on bin packing, it is assumed that the packing may be arbitrarily rearranged to accommodate arriving items. In this context we show that *partially dynamic bin packing* (*Inserts* only) can be solved within $1+\epsilon$ in amortized polylogarithmic time.

1 Introduction

Recall that in the (one–dimensional) bin packing problem, a list $L = (a_1, a_2, ..., a_n)$ of items of size $size(a_i)$ in the interval (0,1] is given. The goal is to find the minimum k such that all of the items a_i can be packed into k unit–size bins. For each bin B_i, the sum of the sizes of the items packed into that bin should not exceed 1. Bin packing is *NP*–complete [14], and a number of approximation algorithms for bin packing are known. The reader is referred to [3] for background information and a survey of bin packing, together with a number of applications.

In this paper, we consider *partially dynamic* bin packing, where:

- items may arrive into the packing dynamically (*Insert*), and
- items may be moved from bin to bin as the packing is adjusted to accommodate arriving items.

We prove that, for any fixed $\epsilon > 0$, there exist approximation algorithms for partially dynamic bin packing (*Inserts* only) that run in amortized polylogarithmic time and are, in terms of the quality of approximation, within the factor of $1+\epsilon$ from an optimal solution.

2 Existing Results

Recall that the usual measure of the quality of a solution produced by a bin packing algorithm A is its *competitive ratio* $R(A)$ defined as:

$$R(A) = \lim_{n\to\infty} \sup_{OPT(L)=n} \frac{A(L)}{OPT(L)},$$

where $A(L)$ and $OPT(L)$ denote, respectively, the number of bins used for packing of the list L by A and some optimal packing of L. Usually, this property is expressed as:

[1]Partially supported by the National Science Foundation under Grant CCR-9120731.

[2]Yale School of Management, Yale University, New Haven, CT, 06520-8200, email: ivkovich@isis.som.yale.edu.

[3]Department of Computer and Information Sciences, University of Delaware, Newark, DE, 19716, email: elloyd@cis.udel.edu.

$$(\forall L)[A(L) \leq R(A)\,OPT(L) + K_A(L)],\ \ K_A(L) = o(OPT(L)).$$

With respect to running times, we say that an approximation algorithm B for bin packing has running time $O(f(n))$ if the time taken by B to process a change (an *Insert* or *Delete*, as applicable) to an instance of n items is $O(f(n))$. In order to facilitate further exposition, we introduce the following terminology. If $O(f(n))$ is a *worst case* bound on the running time required by B to process each change, then B is $f(n)$-**uniform**. If $O(f(n))$ is an *amortized* bound on the running time required by B to process each change, while the worst case bound required by B is $\omega(f(n))$,[4] then B is $f(n)$-**amortized**. Throughout this paper, we will abbreviate the above and instead refer to uniform and amortized algorithms, respectively.

In the domain of off–line algorithms, the value of R has been successively improved [3, 18, 6, 12]. Indeed, it has been shown that for any value of $R \geq 1$, there is an $O(n \log n)$ time algorithm with that competitive ratio [13]. Unfortunately, the running times for these algorithms involve exceedingly large constants (actually, these "constants" depend on how close R is to 1). Among algorithms of practical importance, the best result is an $O(n \log n)$ algorithm for which R is $\frac{71}{60}$ [12].

With respect to *on–line bin packing*, the problem has been defined strictly in terms of arrivals (*Inserts*) – items never depart from the packing (i.e., there are no *Deletes*). Further, on–line algorithms operate under the restriction that each item must be packed into some bin before the next item arrives, and it should remain in that bin permanently. In this context, Yao showed that for every on–line algorithm A, $R(A) \geq 3/2$ [18], while Brown and Liang improved that bound to $R(A) \geq 1.536...$ [1, 16]. Further, the upper bound has been improved over the years to roughly 1.6 [10, 11, 15, 17].

Another notion is *dynamic bin packing* of [2], where each item is associated not only with its size, but also with an arrival time and a departure time (interpreted in the natural way). Here, again (unlike [7]), items cannot be moved once they are assigned to some bin, unless they depart from the system permanently (at their departure time). This variant differs from *fully dynamic bin packing* in that items are not allowed to be moved once they are assigned to a bin, and through the availability of departure time information. It was shown in [2] that for any such algorithm A, $R(A) \geq 2.5$, and that for the Dynamic First Fit [2] the competitive ratio lies between 2.770 and 2.898.

The work of [7] focused on a variant of on–line bin packing, again supporting *Inserts* only, in which each item may be moved a constant number of times (from one bin to another). They provide two algorithms: One with a linear running time (linear in n, the number of *Inserts*, which is also the number of items) and a competitive ratio of 1.5, and one with an $O(n \log n)$ running time and a competitive ratio of $\frac{4}{3}$.

Fully dynamic bin packing, i.e., the problem of maintaining an approximate solution for bin packing when items may arrive (*Insert*) and depart (*Delete*) dynamically was studied in [9]. In accordance with various work on fully dynamic algorithms, and in contrast to prior work on bin packing, it was assumed that the packing may be arbitrarily rearranged to accommodate arriving and departing items. The main result is an fully dynamic approximation algorithm that is $\frac{5}{4}$–competitive and requires $\Theta(\log n)$ time per operation (i.e., for an *Insert* or a *Delete* of an item), where n is the number of items in the (current) instance.

[4] ω-notation is defined as follows: $f(n) \in \omega(g(n))$ if and only if $g(n) \in o(f(n))$ [4].

3 Partially Dynamic Bin Packing

In this section we consider algorithms for partially dynamic bin packing, i.e., algorithms that handle only *Inserts*. All of the on-line algorithms, and the algorithms reported in [7, 9] are, of course, applicable. Both the uniform and the amortized cases are discussed. Of particular interest is the amortized case, where we show the existence of a polynomial time approximation scheme [8] for partially dynamic bin packing.

3.1 Uniform Algorithms for Partially Dynamic Bin Packing

We begin by briefly considering uniform algorithms for handling only *Inserts*. Here, it is easy to see that *every* on–line bin packing algorithm is a dynamic algorithm for *Inserts* only. In that sense, all of the results on on–line bin packing reviewed in Section 2 carry over directly. Further, with the exception of one algorithm presented in [7], all of those algorithms are uniform. The best of those algorithms, for this context, is the algorithm A_2 from [7], which runs in time $\Theta(\log n)$ per operation, and is $\frac{4}{3}$–competitive. The other algorithm presented in [7], algorithm A_1, runs in linear time, i.e., amortized constant time per operation, and is $\frac{3}{2}$–competitive. Note that A_1 can easily be made uniform, by using some of the techniques developed for A_2 in the same paper and modifying the algorithm slightly.

Finally, the best uniform algorithm for insertions is the algorithm *MMP* from [9]. Recall that it has a uniform running time of $\Theta(\log n)$ per *Insert* (as well as *Delete*) operation, and is $\frac{5}{4}$–competitive.

3.2 Amortized Algorithms for Partially Dynamic Bin Packing

The algorithm presented here supports the *Insert* operation and two "lookup" queries of the following form:

- *size* – in $O(1)$ time returns the number of bins in the current packing;
- *packing* – returns a description of the packing in the form of a list of pairs $(x, \mathrm{Bin}(x))$, where $\mathrm{Bin}(x)$ denotes the bin into which an item x is packed, in time linear in n, the number of items in the current instance.

The implementation of these queries in the algorithm presented below is quite elementary, and will thus be omitted.

We next furnish the main result of this paper. More encompassing than the algorithm itself, the result establishes the existence of a polynomial time approximation scheme for partially dynamic bin packing where each *Insert* is processed in amortized polylogarithmic time. In particular, we show:

Theorem 1 *Let A be an (off–line) R_A–competitive ($R_A < 2$) algorithm for bin packing. Let $T_A(n)$ be the running time of A, and let $\epsilon > 0$. Then there exists an algorithm A_ϵ for partially dynamic bin packing that is $(R_A + \epsilon)$–competitive, and requires $O(\frac{T_A(n)}{n} \log n)$ amortized time per Insert operation.*

Proof: We begin with an informal description of the essential features of A_ϵ. A_ϵ can be produced as a combination of A and Next Fit (*NF*). *NF* is (somewhat arbitrarily) chosen because of its simplicity and its linear running time. Informally, *NF* operates as follows. At the outset, the

packing consists of 0 bins. When *NF* receives the first item a_1, it opens a bin B_1, and packs a_1 into B_1. *NF* then packs items $a_2, a_3, \ldots$ into B_1 for as long as each item would fit into B_1. As soon as the first item a_i that does not fit into B_1 appears, B_1 is "closed down" (i.e., no items will be inserted into B_1 in the future, regardless of whether or not they could fit), a new bin B_2 is opened, and a_i is packed into B_2. The subsequent items $a_{i+1}, a_{i+2}, \ldots$ will be packed into B_2 for as long as each item would fit, after which B_2 will be "closed down," a new bin B_3 would be opened, and so on. The performance of *NF* is bounded for every list L as follows: $NF(L) \leq 2OPT(L)$, and $R(NF) = 2$ [3, 10, 11].

Our algorithm A_ϵ proceeds in the following manner: *NF* is used in almost every *Insert* operation, and a certain level of supervision is built into the algorithm, so that when the desired competitive ratio $R(A_\epsilon) = R(A) + \epsilon$ is about to be exceeded, A is used to repack the entire instance inserted so far. Intuitively, this should (temporarily) improve the packing, since $R(A) < 2 = R(NF)$. After that repacking, *NF* is run again, until another repacking is required, and so on.

More formally, we begin by considering a sequence of n *Inserts*. Note that as the *Inserts* take place, the size of an optimal packing grows monotonically from 1 to $OPT(L)$, where $L = (a_1, a_2, \ldots, a_n)$ denotes the list of inserted items. Let $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ denote the *distinguished* items from L with the following property:

$$a_{i_1} = a_1; \;\; \forall j, 2 \leq j \leq k, i_j = min_l : [2\mathrm{OPT}(a_1, a_2, \ \ldots\ , a_{i_{j-1}}) = \mathrm{OPT}(a_1, a_2, \ \ldots\ , a_l)].$$

Simply put, a_{i_j}'s are the sequence of least indices such that $\mathrm{OPT}(a_1, a_2, \ldots, a_{i_j})$ is twice as large as $\mathrm{OPT}(a_1, a_2, \ \ldots\ , a_{i_{j-1}})$, $2 \leq j \leq k$.

Based on these distinguished items, we define a technical concept of *stages* thus. The first stage is trivial; it consists only of the insertion of the first item $a_{i_1} = a_1$. Stage j, $1 < j \leq k$, consists of the sequence of *Inserts* between $a_{i_{j-1}}$ and a_{i_j}, also including a_{i_j} itself. Stage $k+1$ consists of the sequence of *Inserts* between a_{i_k}, the last distinguished item, and a_n, the last item inserted so far. Since the size of an optimal packing cannot double more than $\lceil \log OPT(L) \rceil$ times, which is $O(\log OPT(L)) = O(\log n)$, the number of stages is bounded by $k + 1 = O(\log n)$.

To complete the proof it suffices to produce an algorithm A_ϵ, given an algorithm A with the following performance:

$$(\forall L)[A(L) \ \leq \ R(A)\, OPT(L) + K_A(L)], \quad K_A(L) = o(OPT(L)),$$

such that the following three conditions on A_ϵ are met:

1. $R(A_\epsilon) = R(A) + \epsilon$
2. For every L, $A_\epsilon(L) \leq R(A_\epsilon)\, OPT(L) + K_{A_\epsilon}(L)$, $K_{A_\epsilon}(L) = o(OPT(L))$
3. A_ϵ requires only a constant number, say C, of repackings via A in each stage.

We now show that the third condition insures that the overall running time of A_ϵ on a list of n items is $T_{A_\epsilon}(n) = T_A(n) \ \log n$. First, we note that each item is packed only once via *NF* (when the item is being inserted) and repacked up to $\mathrm{C}(k+1) \ = \ O(\log n)$ times via A. Since the running time required to pack each item via *NF* is $O(1)$, the total running time spent on packing via *NF* is $\Theta(n)$. Furthermore, since each repacking requires $O(T_A(n))$ running time and there is a logarithmic number of repackings, all the repackings require a total of $O(T_A(n) \ \log n)$ running time. Finally,

$A_\epsilon(x)$:
    $L' = append(L, x)$;
    pack x using *NF*;
    if $A_\epsilon(L') >$ `bound` **then**
        repack L using A;
        $\alpha = \left\lfloor \frac{A(L)-K_A(L)}{R(A)} \right\rfloor$;
        `bound` $= \lfloor (R(A)+\epsilon)\alpha \rfloor + K_{A_\epsilon}(L)$;
        pack x using *NF*;[a]
    endif;
    $L = L'$;

[a] Here only the $A(L)$-th bin of the packing most recently produced by A on L is considered to be open, while the other bins are closed for insertions via *NF*.

Figure 1: Algorithm A_ϵ.

since $T_A(n) = \Omega(n)$, the overall time required to pack n items via A_ϵ is $O(T_A(n) \log n)$, which is $O(\frac{T_A(n)}{n} \log n)$ amortized running time per *Insert* operation.

We next establish a sufficient condition to guarantee that A_ϵ requires only a constant number of repackings via A in each stage.

Lemma 1 *Let OPT_i be the size of the optimal packing immediately after the i-th repacking ($i \geq 0$) via A. If there exists a constant $\beta > 0$ such that, $OPT_{i+1} \geq OPT_i + \lceil \beta OPT_i \rceil$, then the number of repackings via A in every stage is bounded by a constant.*

Proof: The number of repackings in the first stage is 1, thus trivially bounded by a constant.

It follows from the hypotheses of the lemma that for every $i \geq 0$, $OPT_{i+1} \geq OPT_i(1+\beta)$. Now suppose that for some i_0-th repacking a stage other than the first stage has just begun. How many repackings will there be by the end of that stage? Let C denote the number of repackings in that stage. By the definition of stages and the hypotheses of the lemma,

$$2OPT_{i_0} \geq OPT_{i_0+C} \geq OPT_{i_0}(1+\beta)^C \Rightarrow C \leq \frac{1}{\log(1+\beta)} = O(1).$$

□

Algorithm A_ϵ is given in Figure 1. The variables L (the list of items) and `bound` should be initialized prior to the first execution of A_ϵ: $L = ()$, and `bound` $= 2\gamma$ (the value of γ will be specified later). The description of the algorithm outlines only the essential features; certain details regarding the maintenance of information necessary for answering the queries are omitted; their implementation can be easily done within the allowed time bounds.

Note that in A_ϵ the variable α offers, after each repacking via A, a lower bound on $\mathrm{OPT}(L)$. It is used to provide a conservative estimate of $R(A_\epsilon)\mathrm{OPT}(L) + K_{A_\epsilon}(L)$, the value that should not be exceeded in order to comply with the competitive ratio of $R(A_\epsilon)$.

The algorithm is not yet completely specified, since the values of γ and $K_{A_\epsilon}(L)$ are not yet determined. This will be remedied as the main ideas behind the algorithm A_ϵ are explained.

Recall that the goal in the design of A_ϵ is not only to achieve the desired competitive ratio, but also to insure that there are but a constant number of repackings via A at each stage. It is precisely this requirement that requires a careful selection of γ and $K_{A_\epsilon}(L)$.

The technique that is quite sufficient to insure the growth of the optimal packing between the i–th and $i+1$–st repacking by at least $\lceil \beta OPT_i \rceil$ (see Lemma 1) is to require *NF* to pack at least twice as many bins, i.e., $\geq 2\lceil \beta OPT_i \rceil$ bins between the two repackings. Since the competitive ratio of *NF* is 2, this will add at least the required number of bins (i.e., $\lceil \beta OPT_i \rceil$ bins) to the optimal packing. Unfortunately, we are not able to implement this directly in A_ϵ, since we do not know the value of OPT_i. We do accomplish this by the use of γ and $K_{A_\epsilon}(L)$ in the following way.

Let the required number of bins packed by *NF* between two repackings be denoted as Δ_i. We desire that:

$$\Delta_i \geq 2\lceil \beta OPT_i \rceil .$$

We will now refine this condition and demand a stronger inequality. The value of Δ_i will be underestimated by $\lfloor R(A_\epsilon)\alpha + K_{A_\epsilon}(L) \rfloor - \lceil R(A)\alpha + K_A(L) \rceil$ and the value of OPT_i is overestimated by $\lceil R(A)\alpha + K_A(L) \rceil$, where α denotes the variable from the description of A_ϵ. It is then desired that the following must hold:

$$\lfloor R(A_\epsilon)\alpha + K_{A_\epsilon}(L) \rfloor - \lceil R(A)\alpha + K_A(L) \rceil \geq 2\lceil \beta \lceil R(A)\alpha + K_A(L) \rceil \rceil .$$

Clearly, then $\Delta_i \geq 2\lceil \beta OPT_i \rceil$ will follow. The above inequality can be satisfied if we insist that:

$$(R(A)+\epsilon)\alpha + K_{A_\epsilon}(L) - 1 - (R(A)\alpha + K_A(L) + 1) \geq 2(\beta(R(A)\alpha + K_A(L) + 1)) + 2,$$

which after some elementary steps yields:

$$(\epsilon - 2\beta R(A))\alpha + (K_{A_\epsilon}(L) - K_A(L)) - 2 \geq 2\beta K_A(L) + 2\beta + 2.$$

If the condition $K_{A_\epsilon}(L) > K_A(L)$ is met, then a yet more refined inequality can be written:

$$(\epsilon - 2\beta R(A))\alpha \geq 2(\beta(K_A(L) + 1) + 2).$$

The above inequality suggests that there is considerable flexibility with respect to the choice of β. In particular, the value of β can be fixed to any value conforming to the inequality $\epsilon > 2\beta R(A)$, i.e.:

$$\beta < \frac{\epsilon}{2R(A)}.$$

Further, the above inequality also suggests that the required growth of the value of the optimal packing between any two repackings can be certainly achieved if:

$$\alpha \geq \left\lceil \frac{2(\beta(K_A(L)+1)+2)}{\epsilon - 2\beta R(A)} \right\rceil .$$

This requirement will be met if the initial packing via *NF* requires at least 2γ bins, where γ can be found as follows:

$$\left\lfloor \frac{\gamma - K_A(L)}{R(A)} \right\rfloor \geq \left\lceil \frac{2(\beta(K_A(L)+1)+2)}{\epsilon - 2\beta R(A)} \right\rceil .$$

This condition is the reflection of the scenario where NF would achieve its worst possible packing on the initial items, whereas A would produce an optimal packing. Some elementary steps yield the following bound on the value of integer γ:

$$\frac{\gamma - K_A(L)}{R(A)} - 1 \geq 2\frac{\beta(K_A(L)+1)+2)}{\epsilon - 2\beta R(A)} + 1$$

Further simplification yields:

$$\gamma \geq 2R(A)\left(\frac{\beta(K_A(L)+1)+2}{\epsilon - 2\beta R(A)} + 1\right) + K_A(L).$$

By the discussion about the worst possible scenario above, it will suffice to set γ and $K_{A_\epsilon}(L)$ to:

$$\gamma = K_{A_\epsilon}(L) \geq \left\lceil 2R(A)\left(\frac{\beta(K_A(L)+1)+2}{\epsilon - 2\beta R(A)} + 1\right) + K_A(L)\right\rceil.$$

Note that this condition guarantees $K_{A_\epsilon}(L) > K_A(L)$, as well as all of the other requirements. This completes the proof of Theorem 1.

□

Corollary 1 *1. For every $\epsilon > 0$ there is a $(1+\epsilon)$–competitive approximation scheme A_ϵ for partially dynamic bin packing that requires $O(\log n)$ amortized time per Insert operation.*

2. For every $\epsilon > 0$ there is a $(1+\epsilon)$–competitive fully polylogarithmic approximation scheme A_ϵ for partially dynamic bin packing that requires $O(\log^2 n)$ amortized time per Insert operation.

Proof:

1. Immediate from Theorem 1 and the results from [5].
2. Immediate from Theorem 1 and the results from [13].

□

4 Conclusion

The open questions related to this paper concern the competitive ratio achievable with uniform running time, and the practicality of the algorithms A_ϵ in the case of amortized running time. Recall that in the uniform case the best known competitive ratio for partially dynamic bin packing is $\frac{5}{4}$ [9]. However, that algorithm was not designed specifically for partially dynamic packing. It is possible that the development of partially dynamic algorithms for bin packing could benefit from the fact that *Deletes* would not be required. In the case of the amortized running time, we note that the efficiency of algorithms A_ϵ is by and large determined by the efficiency of their respective "building blocks" A. In particular, if the design goal is to develop a partially dynamic bin packing algorithm (with a good amortized running time) with a small competitive ratio, developing an entirely new algorithm which would not rely on very good but very slow building blocks (e.g., [5, 13]) might be a preferable approach.

References

[1] D.J. Brown, A lower bound for on-line one-dimensional bin packing algorithms, Tech. Rept. R-864, Coordinated Science Laboratory, University of Illinois, Urbana, IL, 1979.

[2] E.G. Coffman, M.R. Garey and D.S. Johnson, Dynamic bin packing, *SIAM J. Comput.* **12** (1983) 227–258.

[3] E.G. Coffman, M.R. Garey and D.S. Johnson, Approximation algorithms for bin packing: An updated survey, in: G. Ausiello, M. Lucertini and P. Serafini, eds., *Algorithm Design for Computer System Design* (Springer, New York, 1984) pp. 49–106.

[4] T.H. Corman, C.E. Leiserson and R.L. Rivest, *Introduction to Algorithms* (MIT Press, Cambridge, MA, 1990).

[5] W. Fernandez de la Vega and G.S. Lueker, Bin packing can be solved within $1 + \epsilon$ in linear time, *Combinatorica* **1** (4) (1981) 349–355.

[6] D.K. Friesen and M.A. Langston, Analysis of a compound bin packing algorithm, *SIAM Journal on Discrete Mathematics* **4** (1) (1991) 61–79.

[7] G. Gambosi, A. Postiglione and M. Talamo, New algorithms for on–line bin packing, in: G. Ausiello, D.P. Bovet and R. Petreschi, eds., *Algorithms and Complexity, Proc. 1st Italian Conf.* (World Scientific, Singapore, 1990) 44–59.

[8] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, San Francisco, 1979).

[9] Z. Ivković and E.L. Lloyd, Fully dynamic algorithms for bin packing: Being (mostly) myopic helps, *Proc. 1st European Symp. on Algorithms*, Lecture Notes in Computer Science **726** (Springer, New York, 1993) 224-235; also: *SIAM J. on Comput.*, to appear.

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

[11] D.S. Johnson, A. Demers, J.D. Ullman, M.R. Garey and R.L. Graham, Worst-case performance bounds for simple one-dimensional packing algorithms, *SIAM J. on Comput.* **3** (4) (1974) 299–325.

[12] D.S. Johnson and M.R. Garey, A 71/60 theorem for bin packing, *Journal of Complexity* **1** (1985) 65–106.

[13] N. Karmarkar and R.M. Karp, An efficient approximation scheme for the one-dimensional bin-packing problem, *Proc. 23rd IEEE Symp. on Foundations of Computer Science* (1982) 312–320.

[14] R.M. Karp, Reducibility among combinatorial problems, in: R.E. Miller and J.W. Thatcher, eds., *Complexity of Computations* (Plenum, New York, 1972) 85–103.

[15] C.C. Lee and D.T. Lee, A simple on-line bin-packing algorithm, *J. ACM* **3** (1985) 562–572.

[16] F.M. Liang, A lower bound for on-line bin-packing, *Inform. Process. Lett.* **10** (1980) 76-79.

[17] P. Ramanan, D.J. Brown, C.C. Lee and D.T. Lee, On-line bin-packing in linear time, *J. Algorithms* **3** (1989) 305–326.

[18] A.C.-C. Yao, New algorithms for bin packing, *J. ACM* **27** (2) (1980) 207–227.