THE ACCOMMODATING FUNCTION – A GENERALIZATION OF THE COMPETITIVE RATIO *†

JOAN BOYAR ‡, KIM S. LARSEN §, AND MORTEN N. NIELSEN ¶

Abstract. A new measure, the *accommodating function*, for the quality of on-line algorithms is presented. The accommodating function, which is a generalization of both the competitive ratio and the competitive ratio on accommodating sequences, measures the quality of an on-line algorithm as a function of the resources that would be sufficient for an optimal off-line algorithm to fully grant all requests. More precisely, if we have some amount of resources n, the function value at α is the usual ratio (still on some fixed amount of resources n), except that input sequences are restricted to those where the optimal off-line algorithm will not obtain a better result by having more than the amount αn of resources.

The accommodating functions for three specific on-line problems are investigated: a variant of bin-packing in which the goal is to maximize the number of items put in n bins, the Seat Reservation Problem, and the problem of optimizing total flow time when preemption is allowed.

We also show that when trying to distinguish between two algorithms, the decision as to which one performs better cannot necessarily be made from the competitive ratio or the competitive ratio on accommodating sequences alone. For the variant of bin-packing considered, we show that Worst-Fit has a strictly better competitive ratio than First-Fit, while First-Fit has a strictly better competitive ratio on accommodating sequences than Worst-Fit.

Key words. on-line algorithms, performance measures, competitive analysis, restricted adversaries, bin packing, seat reservations, flow time

AMS subject classifications. 68Q25, 05B40, 90B35

1. Introduction.

The competitive ratio [17, 29, 23], as a measure for the quality of on-line algorithms, has been criticized for giving bounds that are unrealistically pessimistic [5, 7, 18, 22, 25], and for not being able to distinguish between algorithms with very different behavior in practical applications [7, 18, 25, 28]. Though this criticism also applies to standard worst-case analysis, it is often more disturbing in the on-line scenario [18].

The basic problem is that the adversary is too powerful compared with the on-line algorithm. For instance, it would often be more interesting to compare an on-line algorithm to other on-line alternatives than to an all-powerful off-line algorithm. A number of papers have addressed this problem [16] by making the on-line algorithm more powerful, by providing the on-line algorithm with more information, or by restricting the set of legal input sequences.

With regards to providing the on-line algorithm with more information, most progress has been made on paging problems. It has been observed [7] that programs exhibit "locality of reference". By supplying an "access graph" as part of the input to the algorithms, this behavior can be modeled. In [7, 19], a number of classes of

*A few of the results in this paper appeared in the proceedings of the Sixth International Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science, Vol. 1663, Springer-Verlag, 74–79, 1999.

†Department of Mathematics and Computer Science, University of Southern Denmark, Main campus: Odense University, Campusvej 55, DK-5230 Odense M, Denmark. Supported in part by the ESPRIT Long Term Research Programme of the EU under project number 20244 (ALCOM-IT) and by grants from the Danish Natural Sciences Research Council (SNF).

‡joan@imada.sdu.dk.

§kslarsen@imada.sdu.dk.

¶nyhave@imada.sdu.dk.

access graphs have been studied. In [11], it is shown that LRU is never worse than FIFO on any access graph.

The "loose competitive ratio" from [31] represents another attempt at improving the ratio. When determining the loose competitive ratio c, the following steps are taken. First, from an infinite set of input sequences, a set of sequences, asymptotically smaller than the whole set, may be disregarded. The remaining sequences should then either be c-competitive or have small cost. The assumption is that sequences of small cost are relatively unimportant.

With regards to making the on-line algorithm more powerful, this has been achieved through so-called "extra-resource analysis" of scheduling problems. In [22] and [28], processor speed is a resource which to some degree compensates for the on-line algorithm's lack of knowledge of the future compared with the (optimal) off-line algorithm. In [28], reduced job arrival rate is also considered as an extra resource the on-line algorithms could be allowed. Another possibility for an extra resource in scheduling problems is extra machines. Graham in [17] compares two arbitrary algorithms, allowing different numbers of processors for the two algorithms. This work was done before the competitive ratio was formally defined, but if one of these two algorithms is the optimal off-line algorithm, this result can be viewed as allowing the on-line algorithm extra machines. In [30, 31], the competitive ratio is improved by allowing some limited look-ahead.

In [25], unrealistic sequences can be removed by specifying a collection of possible distributions. The off-line adversary will choose the distribution which maximizes the ratio of the expected performance of the on-line algorithm to the expected performance of the adversary.

In this paper, we obtain new and stronger results by restricting the adversary. This can be done various ways; we move in the direction of restricting the set of input sequences, the adversary can supply. However, instead of a "fixed" restriction, we consider a function of the restriction, the *accommodating function*. Informally, in on-line problems, where requests are made for parts of some resource, we measure the quality of an on-line algorithm as a function of the resources that would be sufficient for an optimal off-line algorithm. More precisely, if we have some amount of resources n, the function value at α is the usual ratio (still on some fixed amount of resources n), except that input sequences are restricted to those where the optimal off-line algorithm will not obtain a better result by having more than the amount αn of resources.

In the limit, as α tends towards infinity, there is no restriction on the input sequence, so this is the competitive ratio. However, when α is very large, the allowed sequences cannot necessarily be handled very well by the optimal off-line algorithm and can, depending on the application, be quite unrealistic. To avoid comparing an on-line algorithm to the optimal off-line algorithm on problematic sequences of this type, we consider smaller values of α and restrict the adversary so that it can only supply sequences which the optimal off-line algorithm could handle optimally with only αn resources. In the case where increasing the amount of resources available will not improve the optimal off-line algorithm's result, the sequences are called *accommodating sequences* [9][1]. Thus, when $\alpha = 1$, the function value is the competitive ratio on accommodating sequences. Consequently, the accommodating function is a true gen-

[1] In that paper and a preliminary version of the current paper [10], this competitive ratio on accommodating sequences was called the accommodating ratio. The change is made here for consistency with common practice in the field.

eralization of the competitive ratio as well as the competitive ratio on accommodating sequences.

In addition to giving rise to new interesting algorithmic and analytical problems, which we have only begun investigating, this function, compared to just one ratio, contains more information about the on-line algorithms. For some problems, this information gives a more realistic impression of the algorithm than the competitive ratio does. Additionally, this information can be exploited in new ways. The shape of the function, for instance, can be used to warn against critical scenarios, where the performance of the on-line algorithm compared to the off-line can suddenly drop rapidly when more sequences are allowed.

In the next section, we formally define the accommodating function. In the following sections, the accommodating functions for three specific on-line problems are investigated: a variant of bin-packing in which the goal is to maximize the number of items put in n bins, the seat reservation problem, and the problem of optimizing total flow time when preemption is allowed.

In Section 3, where we consider the variant of bin-packing, we consider two specific algorithms, First-Fit and Worst-Fit. We show that although First-Fit performs worse than Worst-Fit with respect to the competitive ratio, it performs better with respect to the competitive ratio on accommodating sequences. Thus, the choice as to which algorithm to use depends on which ratio is more relevant in a specific situation. This would depend on the actual distribution of request sequences and on the accommodating functions for the algorithms.

2. The Accommodating Function. Consider an on-line problem with a fixed amount of resources n. For a *maximization problem*, $\mathbb{A}(I)$ is the *value* of running the on-line algorithm $\mathbb{A}$ on I, and $\mathrm{OPT}(I)$ is the *maximum value* that can be achieved on I by an optimal off-line algorithm, OPT.

For a *minimization* problem, $\mathbb{A}(I)$ is a *cost* and $\mathrm{OPT}(I)$ is the *minimum cost* which can be achieved.

$\mathbb{A}$ and OPT use the same amount of resources, n. For a problem with some limited resource, $\mathrm{OPT}_m(\mathbb{A}_m)$ denotes the value/cost of an optimal off-line algorithm (the on-line algorithm) when the amount m of the limited resource is available.

Definition 2.1. *Let P be an on-line problem with a fixed amount n of resources. For any $\alpha > 0$, an input sequence I to the problem P is said to be an* α-sequence, *if $OPT_{\alpha n}(I) = OPT_{n'}(I)$, for all $n' \geq \alpha n$. 1-sequences are also called* accommodating sequences.

If an input sequence is an α-sequence, then an optimal off-line algorithm does not benefit from having more than the amount αn of resources. For maximization problems, this will often mean that the optimal off-line algorithm could have fully granted all requests with the amount αn of resources. If an input sequence is an accommodating sequence, then an optimal off-line algorithm does not benefit from having more resources than the amount already available.

For a maximization problem, the algorithm $\mathbb{A}$ is *c-competitive on α-sequences* if $c \leq 1$, and for every n and every α-sequence I, $\mathbb{A}_n(I) \geq c \cdot \mathrm{OPT}_n(I) - b$, where b is a fixed constant for the given problem, and, thus, independent of I.

Let $\mathcal{C}^{\alpha}_{\mathbb{A}} = \{c \mid \mathbb{A} \text{ is } c\text{-competitive on } \alpha\text{-sequences}\}$. The *accommodating function* $\mathcal{A}$ is defined as $\mathcal{A}_{\mathbb{A}}(\alpha) = \sup \mathcal{C}^{\alpha}_{\mathbb{A}}$.

For a minimization problem, $\mathbb{A}$ is c-competitive on α-sequences if $c \geq 1$ and for every n and every α-sequence I, $\mathbb{A}_n(I) \leq c \cdot \mathrm{OPT}_n(I) + b$, and the accommodating function is defined as $\mathcal{A}_{\mathbb{A}}(\alpha) = \inf \mathcal{C}^{\alpha}_{\mathbb{A}}$.

With this definition, the competitive ratio on accommodating sequences from [9] is $\mathcal{A}(1)$ and the competitive ratio is $\lim_{\alpha\to\infty}\mathcal{A}(\alpha)$. In this paper, we only consider $\alpha \geq 1$. In Figure 2.1, these relationships are depicted using a hypothetical example for a maximization problem.

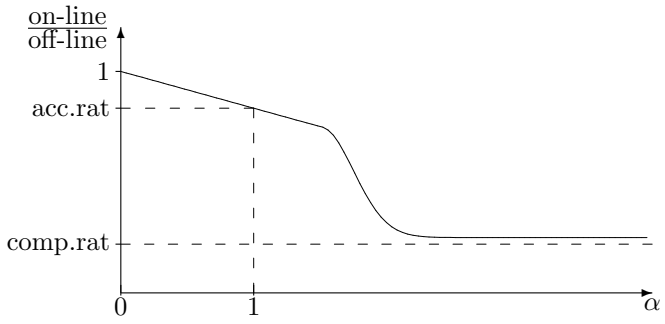

FIG. 2.1. *A typical accommodating function for a maximization problem.*

The extra information contained in the accommodating function compared with the competitive ratio can be used in different ways. If the user knows that estimates of required resources cannot be off by more than a factor three, for instance, then $\mathcal{A}(3)$ is a bound for the problem, and thereby a better guarantee than the bound given by the competitive ratio. The shape of the function is also of interest. Intervals where the function is very steep are critical, since there earnings, compared to the optimal earnings, drop rapidly. Thus, the user misses out on business. So, the function can warn against algorithms with unfortunate behavior, and, if such an algorithm must be used, the function can be used to locate resource critical areas.

3. Fair Bin Packing. Consider the following bin packing problem: Let n be the number of bins, all of size k. Given a sequence of integer-sized items of size at most k, the objective is to maximize the total number of items in these bins. This problem has been studied in the off-line setting, starting in [14], and its applicability to processor and storage allocation is discussed in [13]. (For surveys on Bin Packing, see [12, 15].) The problem we are considering is on-line, so the requests occur in a definite order. We require the packing to be *fair*, that is, an item can only be rejected if it cannot fit in any bin at the time when it is given. We refer to the problem as *Fair Bin Packing*[2]. Notice that the fairness criterion is a part of the problem specification. Thus, even though the optimal off-line algorithm knows the whole sequence of requests in advance, it must process the requests in the same order as the on-line algorithm, and do so fairly.

In this problem, for a given α, we only consider sequences which could be packed in αn bins by an optimal off-line algorithm.

3.1. Summary of Results. The following six theorems summarize our results for deterministic algorithms for Fair Bin Packing. Note that since this is a maximization problem, lower bounds are obtained by proving a bound on the worst case behavior of algorithms, and upper bounds are obtained by giving adversary arguments.

[2]In [10], where a preliminary version of some of these results was presented, the same problem was referred to as Unit Price Bin Packing.

THEOREM 3.1. *For any Fair Bin Packing algorithm, the accommodating function is at most*

$$\mathcal{A}(\alpha) \leq \begin{cases} \frac{6}{7} & : \quad \alpha = 1 \\ \frac{2}{2+(\alpha-1)(k-2)} & : \quad 1 < \alpha \leq \frac{5}{4} \\ \frac{8}{6+k} & : \quad \alpha > \frac{5}{4} \end{cases}$$

for $k \geq 3$.

THEOREM 3.2. *For any Fair Bin Packing algorithm, the accommodating function is at least*

$$\mathcal{A}(\alpha) \geq \begin{cases} \frac{1}{2} & : \quad \alpha = 1 \\ \frac{\alpha}{\max\{1+(\alpha-1)k, 2+(\alpha-1)\frac{k}{2}\}} & : \quad 1 < \alpha < 2 \\ \frac{2-\frac{1}{k}}{k} & : \quad \alpha \geq 2 \end{cases}$$

for $k \geq 3$.

The bounds presented in the previous two theorems are depicted in Figure 3.1.

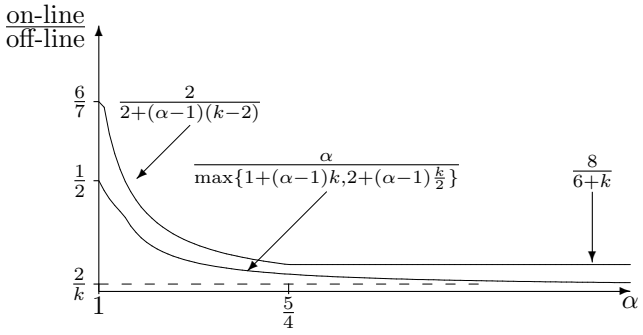

FIG. 3.1. *General upper and lower bounds on the accommodating function for Fair Bin Packing. Drawn for* $k = 60$.

The specific algorithms we consider are First-Fit (FF) and Worst-Fit (WF). First-Fit places an item in the lowest numbered bin in which it fits, while Worst-Fit places an item in one of the bins which are least full.

THEOREM 3.3. *An upper bound on the accommodating function for First-Fit is*

$$\mathcal{A}_{FF}(\alpha) \leq \begin{cases} \frac{7}{11} & : \quad \alpha = 1 \\ \frac{\alpha}{1+(\alpha-1)(k-1)} & : \quad 1 < \alpha < 2 \\ \frac{2-\frac{1}{k}}{k} & : \quad \alpha \geq 2 \end{cases}$$

The general lower bounds from Theorem 3.2 apply to all algorithms, including First-Fit. A better lower bound on the competitive ratio on accommodating sequences can be shown.

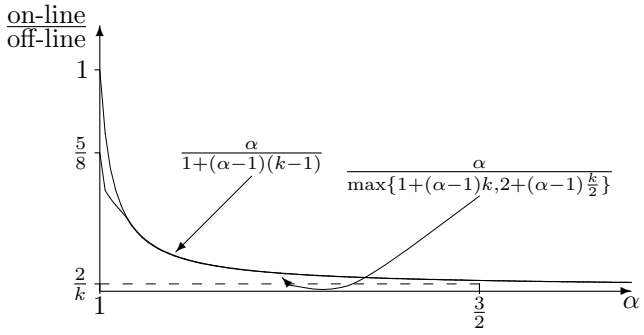

FIG. 3.2. *Upper and lower bounds on the accommodating function for First-Fit. Drawn for* $k = 60$.

THEOREM 3.4. *For First-Fit the competitive ratio on accommodating sequences is at least*

$$\mathcal{A}_{FF}(1) \geq \frac{5}{8}.$$

The bounds on First-Fit presented in the previous three theorems are depicted in Figure 3.2.

THEOREM 3.5. *An upper bound on the accommodating function for Worst-Fit is*

$$\mathcal{A}_{WF}(\alpha) \leq \begin{cases} \frac{1}{2-\frac{1}{k}} & : \quad \alpha = 1 \\ \frac{3-\frac{1}{n}}{2-\frac{1}{n}+(\alpha-1)k} & : \quad 1 < \alpha < 1 + \frac{1}{n}\left\lceil \frac{(n-1)(k-n)+(n-1)}{k} \right\rceil \\ \frac{3+\frac{1}{n-1}}{3+\frac{1}{n-1}+(1-\frac{1}{\beta})k} & : \quad \alpha \geq 1 + \frac{1}{n}\left\lceil \frac{(n-1)(k-n)+(n-1)}{k} \right\rceil,\ k \geq \beta n,\ \beta \geq 1 \end{cases}$$

The general lower bounds from Theorem 3.2 apply to all algorithms, including Worst-Fit. A better lower bound on the competitive ratio can be shown.

THEOREM 3.6. *The competitive ratio of Worst-Fit is at least* $\frac{3}{2+k}$.

3.2. Upper Bounds for All Deterministic Algorithms. Here we prove bounds which apply to all deterministic algorithms for Fair Bin Packing, beginning with an upper bound on the accommodating function. All of the results in this subsection also hold if one relaxes the restriction that all items must be integer-sized to a restriction that the bins have unit size and the smallest item has size $1/k$.

THEOREM 3.7. *For any Fair Bin Packing algorithm, if* $\alpha \leq \frac{5}{4}$ *and* $k \geq 3$*, then* $\mathcal{A}(\alpha) \leq \frac{2}{2+(\alpha-1)(k-2)}$.

Proof. Consider an arbitrary fair on-line algorithm $\mathbb{A}$. An adversary can give $\mathbb{A}$ the following request sequence, divided into three phases. Phase 1 consists of n small items of unit size. Phase 2 consists of items, one for each bin which $\mathbb{A}$ did not fill completely with size equal to the empty space in that bin, sorted in decreasing order. After these are given, $\mathbb{A}$ has filled all bins completely and so must reject the items in Phase 3, which consists of $(\alpha n - n)k$ items of unit size. Let q denote the number of empty bins in $\mathbb{A}$'s configuration after the first phase.

In the case where $q < \frac{n}{4}$, we know that $\mathbb{A}$ has at least $n - 2q \geq 2(\alpha n - n)$ bins with exactly one item after Phase 1. OPT can arrange the items from Phase 1 such that

half of the bins contain two items and half contain no items. In the second request phase, there are at least $2(\alpha n - n)$ items of size $k-1$. OPT rejects at least $\alpha n - n$, leaving room for at least $(\alpha n - n)(k-1)$ of the unit size items from Phase 3. This gives a total gain of at least $(k-2)(\alpha n - n)$, and the performance ratio is at most $\frac{2n}{2n+(\alpha n-n)(k-2)} = \frac{2}{2+(\alpha-1)(k-2)}$.

In the case where $q \geq \frac{n}{4}$, we know that $\mathbb{A}$ has at least $\alpha n - n$ empty bins after Phase 1. OPT places each of the items from Phase 1 in a different bin. This gives a performance ratio of at most $\frac{2n}{2n+(\alpha n-n)(k-1)} = \frac{2}{2+(\alpha-1)(k-1)}$, since OPT rejects $\alpha n - n$ items of size k. □

Using the value $\alpha = \frac{5}{4}$ in the above theorem, gives an upper bound on the competitive ratio.

COROLLARY 3.8. *If $k \geq 3$, no Fair Bin Packing algorithm is more than $\frac{8}{6+k}$-competitive.*

The next theorem improves the bound on the accommodating function for $\alpha = 1$.

THEOREM 3.9. *For $k \geq 7$, any Fair Bin Packing algorithm has a competitive ratio on accommodating sequences of at most $\frac{6}{7}$.*

Proof. Consider an arbitrary fair on-line algorithm $\mathbb{A}$ and assume n is even. An adversary first gives n items of size $\lceil \frac{k}{2} \rceil - 1$. Since $k \geq 7$, $\mathbb{A}$ has packed at most two items per bin. Let q denote the number of empty bins in $\mathbb{A}$'s packing. In the case where $q < \frac{2n}{7}$, the off-line algorithm can pack these n requests in the first $\frac{n}{2}$ bins. Now the adversary can give $\frac{n}{2}$ long requests of size k. The performance ratio is then $\frac{n+q}{n+\frac{n}{2}} = \frac{2n+2q}{3n} < \frac{6}{7}$.

In the case where $q \geq \frac{2n}{7}$, observe that $\mathbb{A}$ has q bins with exactly two items. Let the off-line algorithm place one item in each bin. In this case, the adversary can now give n requests of size $\lfloor \frac{k}{2} \rfloor + 1$. The performance ratio is then $\frac{2n-q}{2n} \leq \frac{6}{7}$. □

3.3. Lower Bounds for All Deterministic Algorithms. Now we prove lower bounds which apply to all deterministic algorithms for Fair Bin Packing. The first lower bound is on the competitive ratio. Fix an on-line algorithm $\mathbb{A}$, and consider the configuration of $\mathbb{A}$ after a given sequence I has been packed. Let e denote the maximal empty space in any bin. Since all items are integer-sized, if the bins are not all full, $e \geq 1$.

LEMMA 3.10. *If $e \geq 1$, the performance ratio of $\mathbb{A}$ is at least $\frac{3}{2+k}$, for $k \geq 3$.*

Proof. Let V denote the volume of the first n requests, all of which must be accepted by $\mathbb{A}$. Note that $n \leq V \leq nk$. All items of size at most e have been accepted by $\mathbb{A}$ due to the fairness criterion. Let M count the number of such small items which arrive after the first n requests.

In the case where $e = 1$, M counts the number of unit size items. As argued above, $\mathbb{A}$ must accept the first n items, plus the M unit size items, and together they have volume $V + M$. Since $\mathbb{A}$ fills every bin with volume at least $k-1$, these items account for all of what $\mathbb{A}$ places in at most $\lfloor \frac{V+M}{k-1} \rfloor$ bins. Thus, it must accept at least $n - \lfloor \frac{V+M}{k-1} \rfloor$ additional items. The optimal off-line algorithm OPT must also accept the first n items and the M unit size items. For other items, it has $nk - V - M$ space remaining. All other items in the request sequence have size at least 2, so OPT accepts at most $\frac{nk-V-M}{2}$ additional items. Hence, the performance ratio is at least

$$\frac{n + M + n - \lfloor \frac{V+M}{k-1} \rfloor}{n + M + \frac{nk-V-M}{2}} \geq \frac{n + M + n - \frac{V+M}{k-1}}{n + M + \frac{nk-V-M}{2}},$$

defined for $n \leq V \leq nk$ and $0 \leq M \leq nk - V$. By finding the minimum with respect to M, it can be shown that the following is a lower bound for the performance ratio:

$$\frac{2n - \frac{V}{k-1}}{n + \frac{nk-V}{2}} \geq \frac{2n - \frac{n}{k-1}}{n + \frac{nk-n}{2}} = \frac{4k-6}{k^2-1}.$$

The last inequality is obtained by finding the minimum with respect to V.

In the case where $e \geq 2$, the performance ratio is at least

$$\begin{aligned}\frac{n + M + (n - \lfloor \frac{V+M}{k-e} \rfloor)}{n + M + \frac{nk-V-M}{e+1}} &\geq \frac{n+M}{n + M + \frac{nk-n-M}{e+1}} \\ &= \frac{(e+1)(M+n)}{en + eM + nk} \\ &\geq \frac{(e+1)n}{en + nk} \geq \frac{3}{2+k}.\end{aligned}$$

For $k \geq 3$, the case $e \geq 2$ gives the smallest value. □

THEOREM 3.11. *The competitive ratio for any Fair Bin Packing algorithm is at least* $\frac{2-\frac{1}{k}}{k}$*, when* $k \geq 3$.

Proof. Consider an algorithm $\mathbb{A}$ for Fair Bin Packing. When $k \geq 3$, the ratio $\frac{3}{2+k}$ from Lemma 3.10 is larger than $\frac{2-\frac{1}{k}}{k}$. Thus, it suffices to consider the case where $e = 0$. Again, let V denote the volume of the first n requests, all of which must be accepted by $\mathbb{A}$, since it is fair. Since $e = 0$, every bin has been filled up by the on-line algorithm, but the first n requests filled at most $\lfloor \frac{V}{k} \rfloor$ bins. Thus, $\mathbb{A}$ accepted at least $n + (n - \lfloor \frac{V}{k} \rfloor)$ items. Since all items have size at least 1, OPT accepts at most $n + (nk - V)$ items, giving a performance ratio of at least $\frac{2n - \lfloor \frac{V}{k} \rfloor}{n+nk-V} \geq \frac{2n - \frac{V}{k}}{n+nk-V} \geq \frac{2n-\frac{n}{k}}{nk} = \frac{2-\frac{1}{k}}{k}$. □

Next we give a lower bound on the competitive ratio on accommodating sequences, $\mathcal{A}(1)$. Note that the result does not depend on the items being integer-sized.

THEOREM 3.12. *Let I be an input sequence which can be accommodated within n bins. The performance ratio is greater than $\frac{1}{2}$ for any Fair Bin Packing algorithm.*

Proof. Let A denote the set of items accepted by the on-line algorithm, and let R denote the set of items rejected. The set R could be empty, but then the on-line algorithm would have accommodated every request, giving a performance ratio of 1. If R is non-empty, at least n items are accepted. Since the performance ratio is $\frac{|A|}{|A|+|R|}$, it is enough to show that $|R| < n$. Let e denote the maximal empty space in any bin. Every item in R has size greater than e. If $|R| \geq n$, then the volume of R is greater than ne. However, this contradicts the fact that the total empty space is at most ne and the entire sequence can be packed in n bins. □

The final general lower bound is on the accommodating function.

THEOREM 3.13. *For any Fair Bin Packing algorithm, the accommodating function can be bounded by* $\mathcal{A}(\alpha) \geq \frac{\alpha}{\max\{1+(\alpha-1)k, 2+(\alpha-1)\frac{k}{2}\}}$*, for* $1 < \alpha < 2$ *and* $k \geq 4$.

Proof. Consider a worst-case sequence I for an on-line algorithm $\mathbb{A}$. We may assume that I contains no item which is rejected by both $\mathbb{A}$ and OPT, since such an item has no influence on the ratio.

Again, let A denote the set of items accepted by the on-line algorithm $\mathbb{A}$, and let R denote the set of items rejected. Let $\rho(I)$ denote the least number of bins in which the sequence can be packed and define $l = \rho(I) - n$. If $l = 0$, we can use the

result from Theorem 3.12. Assume that $l \geq 1$ and note that l is a lower bound on the number of items rejected by OPT. The first n items are accepted by both algorithms. Since, by assumption, the on-line algorithm accepts all those items OPT rejects, it must accept at least $n + l$ items. Thus, $|A| \geq n + l$. Again, let e denote the size of the maximal empty space in any bin. The proof is divided into two cases depending on e.

If $e = 0$, all n bins have been filled by the on-line algorithm, so the number of rejected items is at most lk.

In the case where $e \geq 1$, an upper bound on the number of items rejected is $|R| \leq \frac{1}{e+1}(en + lk)$ using the same arguments as in the previous theorem. We can now bound the accommodating function: $\mathcal{A}(\alpha) \geq \frac{|A|}{|A|+|R|-l} \geq \frac{n+l}{n+l+\max\{lk, \frac{en}{e+1}+l\frac{k}{e+1}\}-l}$ $\geq \frac{n+(\alpha n-n)}{n+\max\{(\alpha n-n)k, n+(\alpha n-n)\frac{k}{2}\}} = \frac{\alpha}{1+\max\{(\alpha-1)k, 1+(\alpha-1)\frac{k}{2}\}}$, for $1 < \alpha < 2$. □

3.4. Separation of First-Fit and Worst-Fit using the Accommodating Function. In this section, we prove that the accommodating function provides extra information, by showing that the best choice between two different algorithms for the same natural problem (Fair Bin Packing) cannot be made based on either the competitive ratio or the competitive ratio on accommodating sequences alone.

The two specific algorithms that we consider are First-Fit (FF) and Worst-Fit (WF). First-Fit places an item in the lowest numbered bin in which it fits, while Worst-Fit places an item in a bin which is least full. We show that Worst-Fit has a better competitive ratio ($r_{\mathrm{WF}} \geq \frac{3}{2+k}$) than First-Fit ($r_{\mathrm{FF}} \leq \frac{2-\frac{1}{k}}{k}$), while First-Fit has a better competitive ratio on accommodating sequences ($\mathcal{A}_{\mathrm{FF}}(1) \geq \frac{5}{8}$) than Worst-Fit ($\mathcal{A}_{\mathrm{WF}}(1) \leq \frac{1}{2-\frac{1}{k}}$). All of the results proven in this section for First-Fit also hold for Best-Fit (BF). Best-Fit is the algorithm which places an item in the most full among the bins where it fits. If the "most full" is not unique, then Best-Fit chooses the first among those "most full" bins.

Worst-Fit's Competitive Ratio on Accommodating Sequences. First, we prove an upper bound on Worst-Fit's competitive ratio on accommodating sequences.

THEOREM 3.14. *Worst-Fit has a competitive ratio on accommodating sequences of at most* $\frac{1}{2-\frac{1}{k}}$*, for all* k*.*

Proof. Assume that n is divisible by k. An adversary can give the following request sequence:

1. n items of unit size.
2. $n - \frac{n}{k}$ items of size k.

Worst-Fit places one small item in each bin, and must reject all the following items. The optimal algorithm behaves like First-Fit and accepts all items giving the following performance ratio: $\frac{n}{n+n-\frac{n}{k}} = \frac{1}{2-\frac{1}{k}}$. □

First-Fit's and Best-Fit's Competitive Ratios on Accommodating Sequences. Now we show that First-Fit's and Best-Fit's competitive ratios on accommodating sequences are at least $\frac{5}{8}$, which is strictly greater than Worst-Fit's competitive ratio on accommodating sequences, when $k > 2$. Thus, according to this performance measure, First-Fit and Best-Fit are better algorithms than Worst-Fit. The proof which shows this lower bound on Best-Fit's competitive ratio on accommodating sequences is essentially the same as the one for First-Fit, after the following two lemmas are proven giving lower bounds on the sizes of the first items Best-Fit places in a new bin. We let $A(\mathbb{A}, I)$ denote the set of items accepted by an on-line

algorithm $\mathbb{A}$, and we let $R(\mathbb{A}, I)$ denote the set of items it rejects. The first lemma shows that if Best-Fit packs items so that some bin is no more than half full, then the performance ratio is at least $2/3 > 5/8$, so we can safely ignore such sequences. In fact, this result holds for First-Fit, too, but we only need to apply it for Best-Fit.

LEMMA 3.15. *Suppose that OPT accepts all items in some request sequence I. Suppose further that in Best-Fit's packing of the sequence I, there is some bin which is no more than half full. Then Best-Fit accepts at least $2/3$ of the items in I.*

Proof. If Best-Fit leaves some bin b no more than half full, then every item in $R(\mathrm{BF}, I)$ must have size strictly greater than $k/2$. Suppose that among the items which Best-Fit accepts, there are x not in b which it places in bins alone. All of these x items must be larger than the empty space in bin b. Otherwise they would have been placed in bin b or some of the contents in bin b would have been placed on top of them. Thus, these x items also have size strictly greater than $k/2$. We consider two cases:

1. Case 1, bin b contains more than one item.
2. Case 2, bin b contains exactly one item.

In the first case, since OPT accepts all of the items in I, there cannot be more than n items of size strictly greater than $k/2$, so $x + |R(\mathrm{BF}, I)| \leq n$. If Best-Fit rejects any items at all, it has at least two items in all except x bins, so the number of items accepted by Best-Fit, $A(\mathrm{BF}, I) \geq 2n - x$, and the total number of items in I is $A(\mathrm{BF}, I) + R(\mathrm{BF}, I)$. Thus, Best-Fit's performance ratio is at least $\frac{A(\mathrm{BF},I)}{A(\mathrm{BF},I)+R(\mathrm{BF},I)} \geq \frac{2n-x}{2n-x+R(\mathrm{BF},I)} \geq \frac{2n-x}{3n-2x} \geq \frac{2}{3}$.

The second case is argued similarly. Since b has only one item, it cannot fit with any of the x items which are placed alone or with any of the rejected items, so $1 + x + |R(\mathrm{BF}, I)| \leq n$. Another difference is that there is now an additional bin, b, which Best-Fit does not give at least two items. In this case, $A(\mathrm{BF}, I) \geq 2n - x - 1$. Thus, Best-Fit's performance ratio is at least $\frac{A(\mathrm{BF},I)}{A(\mathrm{BF},I)+R(\mathrm{BF},I)} \geq \frac{2n-x-1}{2n-x-1+R(\mathrm{BF},I)} \geq \frac{2n-x-1}{3n-2x-2} \geq \frac{2}{3}$. □

When considering Best-Fit in the following, we only look at sequences which Best-Fit packs, such that all bins are more than half full. The next lemma shows that if Best-Fit packs more than one item in some bin, b, then either at least one of them has size greater than $k/2$ or at least two of them have size greater than the final empty space in any bin which was first used before bin b. We call a bin which was first used before that bin an "earlier" bin. This lemma can be seen to follow from Claim 2.2.2 in [21], but a direct proof is included below for completeness.

LEMMA 3.16. *Suppose that OPT accepts all items in some request sequence I, and that in Best-Fit's packing of the sequence I, all bins are more than half full. Then every bin contains either an item of size greater than $k/2$ or at least two items of size greater than the empty space in any earlier bin.*

Proof. Best-Fit only places an item in an empty bin when it will not fit in any earlier bin. Suppose that Best-Fit puts the first item in bin b at time t. This first item must be larger than the empty space in any earlier bin at time t and thus larger than the final empty space in any earlier bin. Suppose that this first item has size no more than $k/2$. Then, by the assumption that all bins are eventually more than half full, Best-Fit must put some other item x in bin b. In addition, all of the earlier bins must have been more than half full at time t. Thus, when this second item x is put in bin b, all earlier bins were more full than b, so this x was too large to fit in them. Thus, if the first item in a bin has size no more than $k/2$, the first two both have size larger than the final empty space in any earlier bin. □

Given a request sequence $I = \langle s_1, s_2, ..., s_t \rangle$, where s_i is the size of item i, we can represent the final configuration of an algorithm $\mathbb{A}$ by *conf*$(\mathbb{A}, I) = \langle S_1, S_2, ..., S_n \rangle$, a list of n multisets, where the multiset S_j contains the sizes of the items in bin j. In order to prove a lower bound on First-Fit's or Best-Fit's competitive ratio on accommodating sequences, we compare *conf*(FF, I) or *conf*(BF, I) to *conf*(OPT, I). In the following, we only write First-Fit or FF, but everything applies to Best-Fit as well. The sizes which appear in *conf*(FF, I) will correspond to items in $A(\mathrm{FF}, I)$, but since we are assuming that OPT can accommodate all t items from I, there will be t items in *conf*(OPT, I). Consider rearranging the items in *conf*(FF, I) so that the items from $A(\mathrm{FF}, I)$ are placed in exactly the same bins as they are in *conf*(OPT, I), creating a new configuration *good*$(I) = \langle S'_1, S'_2, ..., S'_n \rangle$, which also contains exactly those items in $A(\mathrm{FF}, I)$.

First, we prove a lemma which relates the number of moves necessary for changing from *conf*(FF, I) to *good*(I) to the number of items from I which First-Fit rejects. In fact, this lemma holds for any algorithm for Fair Bin Packing, not just for First-Fit and Best-Fit.

LEMMA 3.17. *For any request sequence I which could be accommodated by OPT, the minimal number of items which have to be moved, in order to change from* *conf*$(\mathbb{A}, I)$ *to* *good*(I)*, is an upper bound on the number of items from I which the algorithm $\mathbb{A}$ rejects.*

Proof. Consider any request sequence I and any algorithm $\mathbb{A}$ for Fair Bin Packing. The process of changing from *conf*$(\mathbb{A}, I)$ to *good*(I) involves moving items out of some bins and into others. Those bins that become less full after this rearrangement may have space for one or more items from $R(\mathbb{A}, I)$. Let s denote the size of the smallest item in $R(\mathbb{A}, I)$, and write the size of item i as $s_i = q_i s + r_i$, where $0 \leq r_i \leq s-1$, and the amount of empty space in bin j as $e_j = k - \sum_{s_i \in S_j} s_i$. The new empty space in bin j is $e'_j = e_j + \sum_{s_i \in S_j \setminus S'_j} s_i - \sum_{s_i \in S'_j \setminus S_j} s_i$. When item i moves from bin j to bin j', the value $s_i = q_i s + r_i$ is added to the empty space in bin j and subtracted from the empty space in bin j'. So if we sum, over all bins in the configuration *good*(I), the number of items of size s which could fit in their empty space, the value q_i is added to bin j and subtracted from bin j', giving a net gain of zero. Thus, the only real contribution comes from the original empty space, e_j, and the values r_i. Let $g(j)$ denote the net gain attributed to bin j. Then

$$\begin{aligned} g(j) &\leq \lfloor \tfrac{1}{s}(e_j + \textstyle\sum_{s_i \in S_j \setminus S'_j} r_i) \rfloor \\ &\leq \lfloor \tfrac{1}{s}(s - 1 + |S_j \setminus S'_j|(s-1)) \rfloor \\ &= \lfloor \tfrac{s-1}{s}(|S_j \setminus S'_j| + 1) \rfloor \\ &\leq |S_j \setminus S'_j| \end{aligned}$$

which is the number of items moved out of bin j. Since all of the items from $R(\mathbb{A}, I)$ fit in the empty space available in *good*(I), the total number of items in $R(\mathbb{A}, I)$ is at most the total net gain after rearranging. This net gain is $\sum_{j=1}^{n} g(j) \leq \sum_{j=1}^{n} |S_j \setminus S'_j|$, which is the total number of items moved from one bin to another to get from *conf*$(\mathbb{A}, I)$ to *good*(I). ∎

Note that the ordering of the sets in *conf*(OPT, I) is irrelevant, so we may order them in any way. The above result holds for the minimum number of moves over all of these arrangements.

In order to prove a lower bound on the competitive ratio on accommodating sequences for First-Fit, we prove a lower bound on $|A(\mathrm{FF}, I)|$, which holds for all I. This is done by proving a lower bound on the number of items which do not move

when changing from *conf*(FF, I) to *good*(I). First, we prove a general result which applies to all algorithms for Fair Bin Packing, and then we use this result and the special properties of First-Fit and Best-Fit to prove the lower bound of 5/8.

Define $B(\mathbb{A}, I)$ to be the set of items in I of size greater than $k/2$ which $\mathbb{A}$ accepts. Some of the items moved, in changing from *conf*($\mathbb{A}, I$) to *good*(I), will be moved to be in the same bins as the items in $B(\mathbb{A}, I)$ or moved out of some bin containing an item from $B(\mathbb{A}, I)$. Suppose that these moves are done first. Let $O(\mathbb{A}, I)$ denote the items in $A(\mathbb{A}, I) \setminus B(\mathbb{A}, I)$ which have not moved yet. We use Vol[$\cdot$] to denote the volume of a collection of items, e.g., Vol[$O(\mathbb{A}, I)$] is the total volume of all items in $O(\mathbb{A}, I)$.

LEMMA 3.18. *There must be an ordering of the sets in conf*(OPT, I) *in which at least* $\frac{Vol[O(\mathbb{A},I)]}{k}$ *of the items in* $A(\mathbb{A}, I) \setminus B(\mathbb{A}, I)$ *are not moved.*

Proof. Consider the bipartite graph $G = ((X, Y), E)$ defined as follows:

1. For each bin b, there are two vertices $x_b \in X$ and $y_b \in Y$.
2. For each item in $O(\mathbb{A}, I)$, there is one edge. If the item is in bin i in *conf*(A, I) and in bin j in *good*(I), the corresponding edge is (x_i, y_j).

A matching in this graph G corresponds to a partial renumbering of the bins, and the edges in the matching correspond to items which have not been moved. A well known result due to König [26, 6] states that the size of a maximum matching in a bipartite graph is equal to the size of a minimum vertex cover. However, a vertex cover of G corresponds to a set of bins (possibly with some from *conf*(A, I) and some from *good*(I)), which contain all of the items in $O(\mathbb{A}, I)$. Thus, the number of items which have not been moved is at least the minimum number of bins needed to contain all the items in $O(\mathbb{A}, I)$. The value $\frac{\text{Vol}[O(\mathbb{A},I)]}{k}$ is a lower bound on this number of bins. □

THEOREM 3.19. *The competitive ratios on accommodating sequences for First-Fit and Best-Fit are at least* $\frac{5}{8}$.

Proof. Within this proof, First-Fit and Best-Fit are the only on-line algorithms considered, and the sequence I will be fixed, but arbitrary, subject to the restriction that all requests would have been accepted by OPT (and for Best-Fit subject to the further restriction that, given I, Best-Fit packs all bins to more than half full). We use the following short-hand notation, for legibility: A is used for $A(\text{FF}, I)$ or $A(\text{BF}, I)$, B is used for $B(\text{FF}, I)$ or $B(\text{BF}, I)$, R is used for $R(\text{FF}, I)$ or $R(\text{BF}, I)$, and V is used for Vol[$O(\text{FF}, I)$] or Vol[$O(\text{BF}, I)$].

No two items from B can be placed in one bin, since they all have size greater than $k/2$. So when counting moves as in Lemma 3.17, the items from B can be assumed not to have moved, since the bins can be reordered such that these items stay in their original bins.

According to Lemma 3.17, in order to get from *conf*(BF, I) to *good*(I), at least one item must move for each item in R. All of these items must be in A, along with the items which are not moved. Lemma 3.18 shows that at least $\frac{V}{k}$ of these items in $A \setminus B$ are not moved, so $|A| \geq |R| + |B| + \frac{V}{k}$.

We consider two cases based on the value of s, the size of the smallest item in R. In each case, we first prove a lower bound on V and then use that to prove a lower bound on the performance ratio.

1. **Case $s > \frac{k}{3}$:** Let T denote those bins which do not contain items of size greater than $k/2$, and let e_i denote the size of the empty space in the ith bin in T. For First-Fit, it is clear that, for $2 \leq i \leq n - |B|$, there are at least two items in bin i in T which have size greater than e_{i-1}. For Best-Fit, since we assume that $e_i < k/2$ for all bins i in T, this follows from Lemma 3.16. The empty space in those bins

containing items from B is at most $s-1$, since no items from R could fit in them. Thus, the total volume of those items which are moved to be in the same bins as the items in B is no more than $|B|(s-1)$. The first bin in T has volume $k-e_1$. Hence, the volume $V > (k-e_1) + 2(\sum_{i=1}^{n-|B|-1} e_i) - |B|(s-1)$. The empty space above the first $n-|B|-1$ bins in T, plus the empty space above the other bins, must be large enough to contain all of the rejected items. Thus, $\sum_{i=1}^{n-|B|-1} e_i \geq |R|s-(|B|+1)(s-1)$. Hence,

$$\begin{aligned} V &> (k-e_1) + 2(\textstyle\sum_{i=1}^{n-|B|-1} e_i) - |B|(s-1) \\ &\geq k-(s-1)+2|R|s-2(|B|+1)(s-1)-|B|(s-1) \\ &= k-3s+3+2|R|s-3|B|s+3|B| \\ &\geq k-3s+2|R|s-3|B|s \end{aligned}$$

Note that either the performance ratio is greater than $\frac{5}{8}$ or $\frac{5}{8} \geq \frac{|A|}{|A|+|R|} \geq \frac{|R|+|B|}{2|R|+|B|}$, so $2|R| \geq 3|B|$. Thus, for $s \geq \frac{k}{3}$, this lower bound for V is at least the value at $s = \frac{k}{3}$. Hence, $V \geq \frac{2|R|k}{3} - |B|k$. This volume V requires at least $\frac{V}{k} \geq \frac{2}{3}|R| - |B|$ bins. The performance ratio is then $\frac{|A|}{|A|+|R|} \geq \frac{\frac{5}{3}|R|}{\frac{8}{3}|R|} = \frac{5}{8}$.

2. **Case $s \leq \frac{k}{3}$:** Each of the bins must contain items with size adding up to at least $k-s+1$, or neither First-Fit nor Best-Fit would have rejected an item of size s. As in the previous case, the total volume of those items which are moved to be in the same bins as the items in B is no more than $|B|(s-1)$. Thus, the volume $V \geq (n-|B|)(k-s+1) - |B|(s-1) = nk - ns + n - |B|k$, and $\frac{V}{k} \geq n - \frac{ns}{k} - |B|$. For $s \leq \frac{k}{3}$, this is minimized when $s = \frac{k}{3}$, giving $\frac{V}{k} \geq \frac{2n}{3} - |B|$. In the proof of Theorem 3.12, it was shown that $|R| < n$, so the performance ratio is at least $\frac{|A|}{|A|+|R|} \geq \frac{|R|+\frac{2n}{3}}{2|R|+\frac{2n}{3}} \geq \frac{\frac{5n}{3}}{\frac{8n}{3}} = \frac{5}{8}$.

Therefore, the competitive ratios on accommodating sequences for First-Fit and Best-Fit are at least $\frac{5}{8}$. □

We now show an upper bound which applies to both First-Fit's and Best-Fit's competitive ratios on accommodating sequences.

THEOREM 3.20. *For Fair Bin Packing, First-Fit's and Best-Fit's competitive ratios on accommodating sequences are at most* $\frac{7}{11}$.

Proof. Assume that n is divisible by 13 and that k is greater than 72 and divisible by 3. An adversary can give the following request sequence, divided into four phases:

1. $\frac{3n}{13}$ of size $\frac{k}{3} - 6$.
2. $\frac{6n}{13}$ pairs, one of size $\frac{k}{3} - 1$ followed by one of size $\frac{k}{3} + 3$.
3. $\frac{6n}{13}$ of size $\frac{2k}{3} + 1$.
4. $\frac{12n}{13}$ of size $\frac{k}{3}$.

FF and BF will pack Phase 1 in $\frac{n}{13}$ bins, with three items in each bin. The assumption that $k > 72$ assures that four items from this phase cannot be packed together. From Phase 2, FF and BF will pack one pair in each bin using $\frac{6n}{13}$ bins. In Phase 3, each item will be placed in its own bin, using the last $\frac{6n}{13}$ bins. There will be no space for items from Phase 4.

OPT can pack one item from Phase 1 with two of the items of size $\frac{k}{3} + 3$ from Phase 2, using a total of $\frac{3n}{13}$ bins for this. Then, it can place one item of size $\frac{k}{3} - 1$ from Phase 2 together with one item from Phase 3, using a total of $\frac{6n}{13}$ for this. There are now $\frac{4n}{13}$ empty bins which can each hold three items from Phase 4. The ratio is thus $\frac{3+12+6}{3+12+6+12} = \frac{7}{11}$. □

This bound has been improved later to $\frac{5}{8}$ [1], showing that the lower bound from Theorem 3.19 is tight.

First-Fit's Competitive Ratio. Turning to the competitive ratio, we show that according to this measure, Worst-Fit is the better algorithm. First, we prove an upper bound on First-Fit's competitive ratio. Note that this result also applies to Best-Fit.

THEOREM 3.21. *For Fair Bin Packing, First-Fit has a competitive ratio which is no more than* $\frac{2-\frac{1}{k}}{k}$*, when* k *divides* n*.*

Proof. An adversary gives the following request sequence, divided into three phases:

1. n items of unit size.
2. $n - \frac{n}{k}$ items of size k.
3. $n(k-1)$ items of unit size.

First-Fit accepts the first two phases of requests. The optimal algorithm places each of the first n items in a separate bin, and accepts all requests in phases 1 and 3, giving the following performance ratio: $\frac{n+n-\frac{n}{k}}{nk} = \frac{2-\frac{1}{k}}{k}$. □

For arbitrary n and k, a very similar request sequence gives an upper bound on the competitive ratio for First-Fit of $\frac{2-\frac{1}{k}+\frac{1}{n}}{k}$. By Theorem 3.11, this result is tight.

Worst-Fit's Competitive Ratio. Finally, we prove a lower bound on Worst-Fit's competitive ratio, but first we prove a tight upper bound, since it can provide some intuition for the lower bound $r_{\mathrm{WF}} \geq \frac{3}{2+k}$. The request sequence used to prove this upper bound involves items with sizes dependent on n, the number of bins. However, it is relatively easy to prove an upper bound of $\frac{4}{k+2}$ using a sequence where the sizes only depend on k, rather than also on n.

THEOREM 3.22. *For* $k \geq \beta n$ *and* $\beta \geq 1$*, the competitive ratio for Worst-Fit is no more than* $\frac{3+\frac{1}{n-1}}{3+\frac{1}{n-1}+(1-\frac{1}{\beta})k}$.

Proof. An adversary can give the following request sequence with four phases:

1. $n-1$ items of size $n-1$.
2. $n-1$ items of size 1.
3. n items of size $k-(n-1)$.
4. $(n-1)(k-n)+(n-1)$ items of size 1.

After Phase 2, WF has free space of size $k-(n-1)$ in every bin, and all items from Phase 3 must be accepted. OPT places items from Phase 1 in separate bins, and each item from Phase 2 on top of an item from Phase 1. OPT will then only accept one item from Phase 3, making space for all the unit size items from Phase 4. The

performance ratio is then

$$\begin{aligned}
&\frac{(n-1)+(n-1)+n}{(n-1)+(n-1)+1+(n-1)(k-n)+(n-1)} \\
&= \frac{3n-2}{3n-2+(n-1)(k-n)} \\
&= \frac{3n-3+1}{3n-3+1+(n-1)(k-n)} \\
&= \frac{3+\frac{1}{n-1}}{3+\frac{1}{n-1}+k-n} \\
&\leq \frac{3+\frac{1}{n-1}}{3+\frac{1}{n-1}+(1-\frac{1}{\beta})k}.
\end{aligned}$$

□

In order to compare this ratio to the lower bound of $\frac{3}{2+k}$, note that it can be made arbitrarily close to $\frac{3}{3+k}$ if n and β are made large enough.

We now move on to the lower bound. Consider any request sequence I for Fair Bin Packing. Throughout this section, we assume that there are no items in I which both WF and OPT reject. This cannot affect the results since any such item could simply be removed from the request sequence without affecting the competitive ratio.

The upper bound can give some intuition for the lower bound. In order to allow OPT to accept many more items than WF, the adversary must give some large items which WF will accept, but OPT will reject. In a worst case example, WF packs none of these large items alone. Since OPT is fair, OPT must pack two items in some bins, such that these bins have more contents than the WF bins just before the large items are accepted. The second small item OPT packs in each bin cannot be large enough to cause a rejection alone. Thus for each large item accepted by WF it will accept additionally two items, while OPT will accept at most k small items. Unfortunately, there are many possibilities for sequences, so it is necessary to argue that it is possible to make certain assumptions about them.

We define the following sets:

X is the set of items which both WF and OPT accept.
Y is the set of items which WF accepts, but OPT rejects.
Z is the set of items which WF rejects, but OPT accepts.

Let $y_{last} \in Y$ be the last item from Y in the request sequence I, and let i be the bin where WF places it.

We define the following additional sets:

$X^f \subseteq X$ are those items from X which appear before y_{last} in I.
$Z^f \subseteq Z$ are those items from Z which appear before y_{last} in I.

For any item $z \in Z^f$, define the following two sets:

$Y(z)$ contains those items from Y which appear after z in I.
$Z^f(z)$ contains the item z and all z' from Z^f appearing after z in I.

If necessary, when more than one sequence is involved, we subscript these sets with the name of the sequence, Z^f_I, for instance.

Let e denote the maximal empty space in any of WF's bins, after processing the request sequence I. The case $e \geq 1$ was considered in Lemma 3.10. We now turn to the case $e = 0$, beginning with some lemmas which allow us to make assumptions about the request sequences which give the worst performance ratio for Worst-Fit.

LEMMA 3.23. *If, for some sequence I, WF places two items from Y_I in the same bin, then there exists another sequence I' on which WF places all items from $Y_{I'}$ in separate bins, and on which the performance ratio of WF is smaller.*

Proof. First we show that if, for some sequence I, WF places some item $x \in X_I$ on top of some item $y \in Y_I$ (call this an inversion), then there is another sequence I', containing exactly the same items as I, for which WF and OPT accept exactly the same items as when given I, but WF never places an item from $X_{I'}$ on top of an item from $Y_{I'}$.

We modify I to obtain I', correcting one of these inversions at a time. Suppose that WF places $x \in X_I$ directly on top of $y \in Y_I$ in bin j. Clearly, y occurs before x in I. Due to fairness, y must also be larger than x since OPT accepts x, but not y. Let I' be identical to I up until the point where y appears. Replace y by x. Then x will still be placed in bin j. Let the next items from I' be the same as the next items from I up to the point where WF would put something on top of x in bin j. Assuming there was room for it, insert the item y at this point, and let the rest of I' be the same as I. Since x is smaller than y, the item y will appear in I' no later than where x appeared in I. Thus, WF will place the items from I' in exactly the same bins as the items from I, the only difference being that one item from X will be placed below an item from Y which it had been placed on top of when I was given. OPT will accept exactly the same items as before and place them exactly as before, since the only difference is that it receives an item it would accept anyway earlier and an item it would reject anyway later. Thus the performance ratio is unchanged. This process can be repeated until there are no inversions.

Suppose that for some sequence I, WF places more than one item from Y_I in some bin. From the above, we may assume without loss of generality that WF never places an item from X_I on top of an item from Y_I. Modify I to obtain I' as follows: Consider each bin which receives more than one item from Y_I, one at a time. Among the items from Y_I in the bin, choose the item O which occurs first in the request sequence I. Replace O in the sequence by a single item of size exactly equal to the sum of the sizes of all of the items from Y_I in that bin. Remove all of those other items from the request sequence. Note that all of the items which are merged had originally been placed directly on top of each other.

By induction, carrying out this modification for one bin at a time, it follows that WF places a new item in the same bin as all of the old items from Y_I that it replaced, and gives the same placement to all other items.

In addition, OPT cannot improve its ratio by accepting some of these new items, since then it could also have done it on the sequence I by accepting some of the items from Y_I. Thus, it accepts exactly the same items as from the sequence I. Hence, WF accepts fewer items from I', while OPT accepts the same number, so the performance ratio becomes smaller. □

The following proposition is used in the next lemma and in the proof of the theorem.

PROPOSITION 3.24. *Given a request sequence I and a $z \in Z^f$ and suppose that for all $w \in Z^f(z)$ it is the case that $|Z^f(w)| \leq |Y(w)|$, then there exists a 1–1 mapping $g : Z^f(z) \to Y(z)$ such that for all $w \in Z^f(z)$, $g(w)$ occurs after w in I.*

Proof. Enumerating the items in $Y(z)$ and $Z^f(z)$ separately, starting from y_{last} and working in the direction of the beginning of the sequence I, g could be defined as the mapping which takes an item from $Z^f(z)$ numbered j to the item in $Y(z)$ numbered j. □

It would be tempting to move items accepted by OPT to the end of the sequence and then convert these to unit size items. Since OPT must be fair, there is no guarantee that it would accept exactly the same items. In fact, it might be forced to accept some items from Y, which could use up more volume than the moved elements from Z. However, under some circumstances, it is possible to perform these moves which will be used in the next lemma. It shows that an additional assumption can be made on worst-case request sequences, in those cases where Worst-Fit packs all bins so that they are completely full.

LEMMA 3.25. *If, for some sequence I for which $e = 0$, there exists an item $z \in Z_I^f$ such that $|Z_I^f(z)| \geq |Y_I(z)|$, then there exists another sequence I', where for all items $z \in Z_{I'}^f$, $|Z_{I'}^f(z)| < |Y_{I'}(z)|$, and on which the performance ratio of WF is no larger.*

Proof. Let I be a sequence such that there exists a $z \in Z_I^f$ with $|Z_I^f(z)| \geq |Y_I(z)|$.

Let z be the last item such that $|Z_I^f(z)| \geq |Y_I(z)|$. It must be the case that $|Z_I^f(z)| = |Y_I(z)|$, and $|Z_I^f(w)| < |Y_I(w)|$ for all $w \in Z_I^f$ which occur after z. From Proposition 3.24, we know that there must exist a 1–1 mapping $g : Z_I^f(z) \to Y_I$ such that for all $w \in Z_I^f(z)$, $g(w)$ occurs after w in I. Since WF is fair and rejects w while accepting $g(w)$, $g(w)$ must be smaller than w. This means that the total volume of all items in $Z_I^f(z)$ must be greater than the total volume in $Y_I(z)$.

Modify I to obtain I' by removing all items from $Z_I^f(z)$ and adding that many items of size 1 right after the last item accepted by WF. This will not affect which items WF accepts, since, by assumption, $e = 0$, which means that all bins are full at that point. On the other hand, from the new sequence I', OPT will accept some items from $Y_I(z)$, and since every moved item has size one, it can accept at least one additional item for each item from $Y_I(z)$ which it rejects.

Since $|Z_I^f(z)| = |Y_I(z)|$, OPT accepts at least as many items from I' as from I, so the performance ratio of WF is no larger. □

There is no problem in assuming the forms implied by Lemmas 3.23 and 3.25 simultaneously.

COROLLARY 3.26. *Given k and n, there exists a request sequence I such that the following hold:*

1. *There is no request sequence I' which WF packs so that $e = 0$, for which we have that $\frac{WF(I')}{OPT(I')} < \frac{WF(I)}{OPT(I)}$.*
2. *WF places all items from Y_I in separate bins.*
3. *If WF packs I so that $e = 0$, then for all items $z \in Z_I^f$, $|Z_I^f(z)| < |Y_I(z)|$.*

Proof. Since one can assume that not both WF and OPT reject the same item from I, there are only a finite number of sequences to be considered; one of them must give the worst performance ratio. Begin with that request sequence. The construction from Lemma 3.23 can be applied to ensure that the second condition holds. If $e = 0$ now, the construction from Lemma 3.25 can be applied, without changing WF's behavior, so the last two properties can hold simultaneously. □

Now we are ready to prove a lower bound on the competitive ratio of Worst-Fit.

THEOREM 3.27. *For Fair Bin Packing, the competitive ratio of Worst-Fit is at least $\frac{3}{2+k}$.*

Proof. If for some sequence I, the largest empty space, e, remaining in WF is greater than zero, then by Lemma 3.10, the performance ratio r is at least $\frac{3}{2+k}$. So we assume that $e = 0$.

By Corollary 3.26, we can assume that WF places no two items from Y in the same

bin. Let B_{WF} be the set of bins which receive items from Y, and let $b = |B_{\mathrm{WF}}| = |Y|$. We first show that $|X| \geq 2(b - |Z^f|)$.

Let $U \subseteq X$ be those items from X which are placed in bins belonging to B_{WF}. For any item $x \in U$, OPT must place it with another item from X or from Z^f, or it would be unable to reject the item from Y, which WF placed with x. So if we let B_{OPT} be those bins which receive items from $U \cup Z^f$ from OPT, then every bin in B_{OPT} receives either at least two items from X or at least one from Z^f.

Consider those bins which are not in B_{OPT}. When the last item $y_{last} \in Y$ is given to OPT, such bins only contain items from $X^f \setminus U$. Recall that i is the bin where WF placed the last item $y_{last} \in Y$. Let V be the total volume of all items from U placed in bin i by WF. If j is a bin, not in B_{WF}, WF placed at least one item from X^f there. If it received more than one item from X^f, only the last one could have size greater than V. Otherwise, by its strategy, WF would have placed the next item (after the one of size greater than V) from bin j in bin i.

Thus, only this last item could be placed by OPT in a bin which has no other items from $X \cup Z^f$. The reason for this is that, since the other items have size at most V, a bin with such an item alone would have to accept the item y_{last}.

This means that only $n - b$ of OPT's bins could have no more than one item from X and no items from Z^f, so at least b bins have either at least two items from X or at least one from Z^f. Hence, even if all those from Z^f are in separate bins, $|X| \geq 2(b - |Z^f|)$.

Now WF places the first n items, all of which must be in X, in different bins, so no item in Y can be larger than $k - 1$. By Corollary 3.26, Proposition 3.24, and fairness, items in Z^f must be larger than corresponding items in Y, so since $e = 0$, $|Z \setminus Z^f| \leq (k-1)(|Y| - |Z^f|)$, which implies that $|Z| \leq |Z^f| + (k-1)(b - |Z^f|)$.

Thus, the ratio r is bounded by
$\frac{|X|+|Y|}{|X|+|Z|} \geq \frac{|X|+b}{|X|+|Z^f|+(k-1)(b-|Z^f|)} \geq \frac{1}{\frac{|X|+|Z^f|}{|X|+b} + \frac{(k-1)(b-|Z^f|)}{3b-2|Z^f|}} \geq \frac{1}{1+\frac{k-1}{3}} = \frac{3}{2+k}$. □

3.5. The Accommodating Functions for First-Fit and Worst-Fit. Finally, we prove upper bounds on the accommodating functions for First-Fit and Worst-Fit. All of the results in this subsection also hold if one relaxes the restriction that all items must be integer-sized to a restriction that the bins have unit size and the smallest item has size $1/k$. The upper bound for First-Fit is close to the general lower bound of Theorem 3.13, so it is almost tight.

THEOREM 3.28. *For Fair Bin Packing, First-Fit has an accommodating function of at most* $\frac{\alpha}{1+(\alpha-1)(k-1)}$, *for* $1 < \alpha < 2$.

Proof. The adversary's request sequence is divided into four phases. First, give $\alpha n - n$ items of unit size, and second, give one item of size $k - (\alpha n - n) \bmod k$. If k divides $\alpha n - n$, this item has size 0 and is not given. Third, give $n - \lceil \frac{\alpha n - n}{k} \rceil$ items of size k. Fourth, give $(\alpha n - n)(k-1)$ items of unit size.

First-Fit accepts those items in the first three phases. The off-line algorithm places the first n items in separate bins, rejects the remaining long items, and accepts all items from Phase 4. The performance ratio is at most $\frac{(\alpha n - n) + (n - \lfloor \frac{\alpha n - n}{k} \rfloor)}{n + (\alpha n - n)(k-1)} \leq \frac{\alpha n}{n + (\alpha n - n)(k-1)} = \frac{\alpha}{1+(\alpha-1)(k-1)}$. □

The proof of Theorem 3.22, giving an upper bound on Worst-Fit's competitive ratio, can be extended to prove an upper bound on Worst-Fit's accommodating function.

THEOREM 3.29. *For Worst-Fit, if* $1 \leq \alpha \leq 1 + \frac{1}{n}\lceil \frac{(n-1)(k-n)+(n-1)}{k} \rceil$ *and* $k \geq n$,

then $\mathcal{A}_{WF}(\alpha) \leq \frac{3-\frac{2}{n}}{3-\frac{3}{n}+(\alpha-1)k(1-\frac{1}{k-n+1})}$.

Proof. In the proof of Theorem 3.22, since all of the items in the first three phases of the request sequence fit in n bins, to determine how many bins would be necessary for an optimal off-line algorithm, one only needs to compute how many bins are necessary for Phase 4. Thus, $\alpha = 1 + \frac{1}{n}\lceil\frac{(n-1)(k-n)+(n-1)}{k}\rceil$. Hence, for any value of α less than this, replacing Phase 4 by $(\alpha-1)kn$ items of size 1, gives an α-sequence for which WF will accept exactly those requests in the first three phases, while OPT can accept all items, except some of the items from Phase 3. The number of items from Phase 3 which OPT can accept is $\lfloor\frac{kn-((n-1)(n-1)+n-1+(\alpha-1)kn)}{k-(n-1)}\rfloor = \lfloor\frac{n(k-n+1)-(\alpha-1)kn}{k-n+1}\rfloor$. Thus, the accommodating function is

$$\begin{aligned}\mathcal{A}_{\mathrm{WF}}(\alpha) &\leq \frac{3n-2}{2n-2+(\alpha-1)kn+n-\lfloor\frac{(\alpha-1)kn}{k-n+1}\rfloor} \\ &\leq \frac{3-\frac{2}{n}}{3-\frac{3}{n}+(\alpha-1)k(1-\frac{1}{k-n+1})}\end{aligned}$$

□

4. The Unit Price Seat Reservation Problem. The competitive ratio on accommodating sequences was introduced in [9][3] in connection with the Seat Reservation Problem, which was originally motivated by some ticketing systems for trains in Europe. The set-up is as follows: A train with n seats travels from a start station to an end station, stopping at $k \geq 2$ stations, including the first and last. Reservations can be made for any trip from a station s to a station t. The passenger is given a single seat number when the ticket is purchased, which can be any time before departure. The algorithms (ticket agents) attempt to maximize income, i.e., the sum of the prices of the tickets sold. For political reasons, the problem must be solved in a *fair* manner, i.e., the ticket agent may not refuse a passenger if it is possible to accommodate him when he attempts to make his reservation. In this paper, we consider only the pricing policy in which all tickets have the same price, the *unit price problem*; for the proportional price problem, where the price of the ticket is proportional to the distance traveled, there does not appear to be any significant difference between the competitive ratio and the competitive ratio on accommodating sequences. We define the accommodating function $\mathcal{A}(\alpha)$ for the Seat Reservation Problem to be the ratio of how well an on-line algorithm can do compared to the optimal off-line algorithm, OPT, when an optimal off-line algorithm could have accommodated all requests if it had had $\alpha n \geq n$ seats. The accommodating function could help the management in determining how much benefit could be gained by adding an extra car to the train, given their current distribution of request sequences. Notice that the fairness criterion is a part of the problem specification. Thus, even though the optimal off-line algorithm knows the entire sequence in advance, it too must process the sequence in the given order and do so fairly.

The Seat Reservation Problem is similar to the problem of coloring an interval graph on-line, which has been well studied because of applications to dynamic storage allocation. The difference is that with graph coloring, all vertices must be given a color and the goal is to minimize the number of colors. With the Seat Reservation Problem, there is a fixed number of colors, and the goal is to maximize the number of vertices that get colors. We use, however, an interesting result from interval graph theory: Interval graphs are *perfect* [20], so the size of the largest clique is exactly the

[3]It was called the accommodating ratio there.

number of colors needed. Thus, when there is no pair of stations $(s, s+1)$ such that the number of people who want to be on the train between stations s and $s+1$ is greater than n, the optimal off-line algorithm will be able to accommodate all requests. The contrapositive is clearly also true; if there is a pair of stations such that the number of people who want to be on the train between those stations is greater than n, the optimal off-line algorithm will be unable to accommodate all requests. We will refer to the number of people who want to be on the train between two stations as the *density* between those stations.

4.1. Bounds on the Accommodating Function. In [9], the following lower bounds for the competitive ratio and the competitive ratio on accommodating sequences were proven: Any algorithm for the Unit Price Seat Reservation Problem is $\frac{2}{k}$-competitive, and any algorithm for the Unit Price Seat Reservation Problem is $\frac{1}{2}$-competitive on accommodating sequences. The key idea for the proof of the theorem bounding the competitive ratio on accommodating sequences is also used to prove a lower bound on the accommodating function, and the result generalizes the one for the competitive ratio on accommodating sequences.

THEOREM 4.1. *$\mathcal{A}(\alpha) \geq \frac{1}{2+(k-2)(1-\frac{1}{\alpha})}$ is a lower bound for the Unit Price Seat Reservation Problem.*

Proof. Consider any algorithm $\mathbb{A}$ for the Unit Price Seat Reservation Problem and any request sequence, I, which an optimal off-line algorithm could have accommodated with $\rho(I) \geq n$ seats. Let $l = \rho(I) - n$, and suppose that $\mathbb{A}$ accepts h intervals. We first show that $\mathbb{A}$ rejects at most $h + l(k-1)$ intervals.

Let S denote the seating assignment found by the on-line algorithm, and let U be the set of unseated intervals. First, some of the intervals in U will be assigned to distinct intervals in S. Let S' be a seating assignment which is initialized to be the same as S, but which will be altered by the following process. Note that the only changes will be to increase the lengths of some intervals in S'.

First order the intervals in U by increasing left endpoint (starting station), breaking ties arbitrarily. Now process these intervals, one by one, in increasing order.

For a given interval $I \in U$, if there is no seat which is empty in S' from the point where the passenger wants to get on until at least the next station, leave I in U. Otherwise, find such a seat. Since $\mathbb{A}$ is fair and the interval I was rejected, the interval I could not be placed on that seat, so there must be a first (leftmost) interval J assigned to that seat in S' which overlaps the interval I. Assign the interval I to the interval J. Now remove I from U and replace J on this seat in S' by an interval K, which is as much of $I \cup J$ as will currently fit on that seat. Clearly, all of the intervals which are now seated in S' and all of the unseated intervals currently in U could be seated by an optimal algorithm on αn seats, since this operation cannot increase the density anywhere. This process can be repeated. The order of processing ensures that each interval $I \in U$ which gets assigned to an interval in S gets assigned to a distinct interval in S. Thus, after all of U has been processed, at most h intervals have been removed from it. For every interval I' remaining in U, the leftmost unit segment (the point where the passenger wants to get on until the next station) has density n in S', so there is now density at most l for that unit segment in U. The number of possible distinct leftmost segments in U is at most $k-1$, so the total number of leftmost segments remaining in U, and thus the total number of intervals in U, is at most $l(k-1)$. We have now shown that $\mathbb{A}$ rejects at most $h + l(k-1)$ intervals.

To compute a lower bound on the ratio of what $\mathbb{A}$ accepts to what OPT accepts, we need to have a lower bound on the number of intervals $\mathbb{A}$ accepts and an upper

bound on the number of intervals OPT accepts. We may assume that there are no intervals in the request sequence which both $\mathbb{A}$ and OPT reject, since removing them from the sequence changes nothing. Since an optimal off-line algorithm could not have accommodated all of the requests with fewer than $\rho(I)$ seats, but OPT has only n seats, OPT must reject at least l intervals, and all of these must have been accepted by $\mathbb{A}$. Clearly, the first n intervals in the request sequence must have been accepted by both $\mathbb{A}$ and OPT. Thus, $\mathbb{A}$ accepts $h \geq n + l = \rho(n)$ intervals. Of the h intervals which $\mathbb{A}$ accepts, OPT rejects at least l of them. Additionally, there are at most $h + l(k-1)$ intervals which OPT accepts, but $\mathbb{A}$ does not. Thus, a lower bound on the accommodating function is $\mathcal{A}(\alpha) \geq \frac{h}{2h+l(k-1)-l} = \frac{1}{2+(k-2)\frac{l}{h}} \geq \frac{1}{2+(k-2)\frac{\rho(n)-n}{\rho(n)}} \geq \frac{1}{2+(k-2)(1-\frac{1}{\alpha})}$. □

In [9], the following upper bounds for the competitive ratio and the competitive ratio on accommodating sequences were proven: No deterministic algorithm for the Unit Price Seat Reservation Problem is more than $\frac{8}{k+5}$-competitive, and no deterministic algorithm for the Unit Price Seat Reservation Problem is more than $\frac{8k-9}{10k-15}$-competitive on accommodating sequences, when k is divisible by 3. The proof proving an upper bound on the accommodating function for any algorithm for Unit Price Bin Packing, is very similar to the proof of the theorem in [9] giving the upper bound on the competitive ratio. The result obtained is very close to that for the competitive ratio when $\alpha > \frac{5}{4}$.

THEOREM 4.2. *$\mathcal{A}(\alpha) \leq \frac{4}{3+2(k-2)\min\{\frac{1}{4},\alpha-1\}}$ is an upper bound for the Unit Price Seat Reservation Problem.*

Proof. The following is an adversary argument, so the request sequence depends on the on-line algorithm $\mathbb{A}$'s behavior. Assume that n is divisible by 2. The adversary begins with $\frac{n}{2}$ pairs of requests for $[1,2]$ and $[k-1,k]$ intervals. Suppose that the algorithm $\mathbb{A}$ places them such that after these requests there are exactly q seats which contain two intervals. Then $n-2q$ of the seats have exactly one short interval scheduled. Next, the adversary will give q requests for $[1,k]$ intervals, followed by $\frac{n-2q}{2}$ requests for $[1,k-1]$ intervals, $\frac{n-2q}{2}$ requests for $[2,k]$ intervals, and q requests for $[2,k-1]$ intervals, all of which can be accommodated by $\mathbb{A}$. Now the train is full. One of two cases will occur:

1. Case 1: $q \geq \frac{n}{4}$, or
2. Case 2: $q < \frac{n}{4}$.

If Case 1 occurs, the adversary will give $\min\{q, \alpha n - n\}$ requests for each of the intervals $[1,2]$, $[2,3]$, $[3,4]$,..., $[k-1,k]$, none of which can be accommodated by $\mathbb{A}$. On the other hand, OPT could put each of the short intervals on a separate seat, so that it would be unable to accommodate the q $[1,k]$ intervals, but all of the other intervals would fit. The on-line algorithm $\mathbb{A}$ is able to accommodate $2n$ requests, while OPT can accommodate $2n - q + \min\{q, \alpha n - n\}(k-1)$ requests. Since $\frac{n}{4} \leq q \leq \frac{n}{2}$, this ratio is less than $\frac{2n}{2n-\frac{n}{2}+\min\{\frac{n}{4},\alpha n-n\}(k-1)} = \frac{4}{3+2(k-1)\min\{\frac{1}{4},\frac{\alpha n-n}{n}\}}$.

If Case 2 occurs, the adversary will give $\min\{\frac{n-2q}{2}, \alpha n - n\}$ requests for each of the intervals $[2,3]$, $[3,4]$,..., $[k-1,k]$, none of which can be accommodated by $\mathbb{A}$. On the other hand, OPT could pair up the short intervals, putting two per seat, so that it would be unable to accommodate the $\frac{n-2q}{2}$ $[2,k]$ intervals, but all of the other intervals would fit. The on-line algorithm $\mathbb{A}$ is able to accommodate $2n$ requests, while OPT can accommodate $2n - \frac{n-2q}{2} + (k-2)\min\{\frac{n-2q}{2}, \alpha n - n\}$ requests. Since $q < n/4$, this ratio is less than $\frac{2n}{2n-\frac{n}{4}+(k-2)\min\{\frac{n}{4},\alpha n-n\}} = \frac{8}{7+4(k-2)\min\{\frac{1}{4},\frac{\alpha n-n}{n}\}}$.

This argument assumes that $k \geq 4$, but the result clearly holds for $k = 2$ and

$k = 3$, too. □

As an example of a specific on-line algorithm, one might consider First-Fit, which always processes a new request by placing it on the first seat which is unoccupied for the length of that journey. The lower bound from Theorem 4.1, on the accommodating function for any algorithm, clearly applies to this specific algorithm. It also applies to Best-Fit, which always processes a new request by placing it on a seat so it leaves as little total free space as possible on that seat immediately before and after that passenger's trip. The following result, giving an upper bound on the accommodating function for these two specific algorithms, should be compared with the results in [9], which give upper bounds for the competitive ratio and the competitive ratio on accommodating sequences for First-Fit and Best-Fit: First-Fit and Best-Fit have competitive ratios which are no better than $\frac{2-\frac{1}{k-1}}{k-1}$ and competitive ratios on accommodating sequences no better than $\frac{k}{2k-6}$, for the Unit Price Seat Reservation Problem. The proof is very similar to the proof of the theorem in [9] which gives the upper bound on the competitive ratios for First-Fit and Best-Fit.

THEOREM 4.3. *For the Unit Price Seat Reservation Problem, First-Fit and Best-Fit have* $\mathcal{A}(\alpha) \leq \max\{\frac{2-\frac{1}{k-1}}{k-1}, \frac{2-\frac{1}{k-1}}{1+(k-1)(\alpha-1)}\}$.

Proof. We will state everything in terms of the First-Fit algorithm, but Best-Fit would behave exactly the same. We will assume that n is divisible by $k-1$. The request sequence will start with $\frac{n}{k-1}$ requests for each of the intervals $[1,2], [2,3], [3,4], ..., [k-1,k]$ which First-Fit will put in the first $\frac{n}{k-1}$ seats. Then there will be $n - \frac{n}{k-1}$ requests for $[1,k]$ intervals, which First-Fit will put in the remaining seats. At this point, the train will be full, but there will now be $\min\{n - \frac{n}{k-1}, \alpha n - n\}$ requests for each of the intervals $[1,2], [2,3], [3,4], ..., [k-1,k]$, all of which First-Fit will be unable to accommodate. It will accommodate a total of $2n - \frac{n}{k-1}$ requests. OPT will put each of the original first intervals on a different seat, thus arranging that it can reject the longest intervals. Then, it will be able to accommodate all of the additional short intervals. Thus, it will accommodate n of the original short intervals, plus $\min\{n - \frac{n}{k-1}, \alpha n - n\}(k-1)$ of the later short intervals. This gives a ratio of $\max\{\frac{2-\frac{1}{k-1}}{k-1}, \frac{2-\frac{1}{k-1}}{1+\frac{\alpha n-n}{n}(k-1)}\}$. This argument assumes that $k \geq 3$, but the result trivially holds for $k = 2$ too. □

5. Other Problems. It is natural to ask if the accommodating function can be defined for any on-line problem. This is equivalent to asking if α-sequences can be defined for every on-line problem. The answer is clearly "yes" if there is no requirement that a relevant resource be considered; then the accommodating function is constant and its value is the competitive ratio. This answer is not particularly interesting. It seems, however, that for most on-line problems, there is some relevant resource which can be used to define α-sequences and therefore also the accommodating function. For Paging, the obvious resource is the number of pages in fast memory; for Scheduling, it is the number of machines available; and for server problems, it is the number of servers.

Unfortunately, for some on-line problems, using the accommodating function with the obvious resource choice and $\alpha \geq 1$ fails to result in additional insight compared with what is already known from the competitive ratio. For example, one of the best known on-line problems is Paging. The well-known lower bound results, which show that any deterministic on-line algorithm has a competitive ratio of at least k, where k is the number of pages in main memory, holds even if there are only $k+1$ pages

in all. Thus, nothing further is said about how much it helps to have extra memory, unless one actually has enough extra memory to hold all of a program and its data. In fact, however, it was shown later [8] that the accommodating function for the Paging Problem becomes more interesting when $\alpha < 1$.

The situation is similar when considering the accommodating function for problems which generalize the Paging Problem, such as the k-Server Problem and metrical task systems.

5.1. Minimizing Flow Times on m Identical Machines. As an example of a very different type of problem where the accommodating function can be applied, we have considered a scheduling problem: the problem of minimizing flow time in a situation where there are m identical machines and preemption is allowed. Let J be the sequence of jobs. A job $j \in J$ arrives at its release time r_j, and its processing time p_j is known. The total flow time is $\sum_{j \in J}(C_j - r_j)$, where C_j denotes the completion time of job j. There are some very nice results in [27] showing that Shortest Remaining Processing Time (SRPT) has a competitive ratio of $O(\log P)$ for this problem, where P is the ratio between the processing time for the longest job and the shortest job. They also show that any randomized algorithm for the problem has a competitive ratio of $\Omega(\log P)$.

The concept of an accommodating function can be applied to this problem, even though it is a minimization problem and no rejections are allowed. Given a request sequence J, there is an absolute minimum flow time—the sum s of the processing times for all jobs in J. Thus, one can define the competitive ratio on accommodating sequences, by restricting the request sequences to those which OPT could schedule with total flow time s. This means that all jobs can be scheduled immediately when they arrive. Thus, any on-line algorithm which assigns an in-coming job to some free processor, when such a processor exists, will also schedule that sequence with total flow time s, giving a competitive ratio on accommodating sequences of 1, which is significantly different from the competitive ratio. The accommodating function $\mathcal{A}(\alpha)$ can be defined by restricting the request sequences to those in which an optimal off-line algorithm could have begun each job immediately upon arrival if it had $\alpha m \geq m$ machines available.

Using the techniques from the result in [27], proving a lower bound on the competitive ratio for SRPT, and lengthening the adversary's sequence appropriately gives a lower bound of $\Omega(\log_m P)$ on the performance ratio of SRPT, even when OPT could have handled the request sequence with only $m + 1$ processors. Thus, SRPT's behavior is similar to that of all Paging algorithms; restricting to sequences which OPT could accommodate with only one extra unit of the resource gives essentially the same result as allowing any sequence whatsoever. It is also possible to show that all other algorithms for this problem also have a sudden change from the competitive ratio on accommodating sequences to the competitive ratio.

Although some of the ideas from the $\Omega(\log P)$ lower bound on the competitive ratio in [27] are used in the proof here, their adversary uses sequences with $\alpha = \frac{3}{2}$, while our adversary only needs $\alpha m = m + 1$.

THEOREM 5.1. *For $\alpha m = m + 1$ and $m > 4$, the performance ratio of any deterministic on-line algorithm is $\Omega(\log_m P)$.*

Proof. Consider a deterministic on-line algorithm $\mathbb{A}$ and assume P is a power of m. An adversary can give a request sequence consisting of $L = \lfloor \log_m P - 1 - \log_m 4 - 2 \log_m \log_m P \rfloor$ phases. For phase $i = 0, \ldots, L-1$, let $p_i = \frac{P}{m^i}$, and $r_i = m * \sum_{j=0}^{i-1} p_j$. The adversary will repeat the same set of jobs m times. Let j denote the repetition

number, and let $r_{ij} = r_i + j * p_i, j = 0, \ldots, m-1$. The following jobs are given

1. One job of size p_i at time r_{ij}.
2. m unit size jobs at each of the times $r_{ij} + k, k = 0, \ldots, p_i(1 - \frac{1}{m}) - 1$.

Let S_{ij} denote the set of all unit size jobs given in phase i and repetition j, and let U_{ij} count the number of jobs from S_{ij} that are not finished by $\mathbb{A}$ before time $w_{ij} = r_{ij} + p_i(1 - \frac{1}{m})$.

There are two cases depending on the U_{ij}.

1. Case 1, there exist i, j, such that $U_{ij} \geq m \log_m P$.
2. Case 2, for all $i, j, U_{ij} < m \log_m P$.

If Case 1 occurs for i_0, j_0, the above release pattern is stopped at time $w_{i_0 j_0}$. Instead, the adversary gives m unit size jobs at each of the next P^3 time units.

An off-line algorithm OPT could finish all jobs from one repetition before the next starts by processing the long job on one machine, and the unit size jobs on the other machines. Beginning with phase i_0 and repetition j_0, OPT should process all unit size jobs, including the unit size jobs given after $w_{i_0 j_0}$, immediately when they are released. Then OPT only has one long unfinished job at time $w_{i_0 j_0}$, which will be delayed for time P^3.

The total flow time for OPT is then at most $m^2 P^2 \log P + (m+1)P^3 + P$, since from the first part there are less than $m^2 P \log P$ jobs, which will be delayed for at most P time.

Since Case 1 has occurred, the on-line algorithm has at least $m \log_m P$ jobs delayed at every time unit for P^3 time steps. The total flow time for the on-line algorithm is then more than $mP^3 \log_m P$.

In Case 1, the performance ratio is then more than $\frac{mP^3 \log_m P}{m^2 P^2 \log P + (m+1)P^3 + P}$.

If Case 2 occurs, an off-line algorithm, OPT, can follow a pattern similar to Case 1 and finish every job from Phase i before Phase $i+1$ starts. Call the time just after the last phase ends r_L. Starting at time r_L, the adversary gives m unit size jobs at each of the next P^3 time units. OPT can process them immediately upon arrival.

In this case, OPT has a total flow time of less than $m^2 P^2 \log P + mP^3$.

The on-line algorithm $\mathbb{A}$ will have many long jobs hanging at time r_L. Fixing a phase i, we want to calculate the possible processing time for long jobs in this and the following phases. In Phase i, the long jobs appear one at a time, so in repetition j there are $j+1$ available. After time w_{ij} there are $\frac{P}{m^{i+1}}$ time units remaining in repetition j, and since this is Case 2, there is a total of less than $m \log_m P$ time units available on all the processors together before time w_{ij}. Thus, the maximum amount of time the m long jobs from Phase i can be processed within Phase i is in total bounded by $\sum_{j=1}^{m} (j \frac{P}{m^{i+1}} + m \log_m P)$. For the following phases the total time available for processing these m jobs from Phase i is bounded by $\sum_{j=i+1}^{L-1} (m \frac{P}{m^{(j+1)}} + m^2 \log_m P)$. This adds up to at most $2m^2 \log_m^2 P + \frac{p_i}{m} \frac{m(m+1)}{2} + p_i \leq p_i \frac{m+2}{2} + p_i$, since by the definition of L, $2m^2 \log_m^2 P \leq \frac{p_i}{2}$, for all i. It follows that for a fixed phase i, at most $\lfloor \frac{m+4}{2} \rfloor$ of the long jobs could be run to completion. Since we have L phases, at least $\lceil \frac{m-4}{2} L \rceil$ long jobs are unfinished at time r_L. This gives a flow time of at least $\frac{m-4}{2} P^3 \log_m P$.

In Case 2, the performance ratio is at least $\frac{\frac{m-4}{2} P^3 \log_m P}{m^2 P^2 \log P + mP^3}$. □

The difference between this lower bound and the lower bound on the competitive ratio from [27] is quite small: $\Omega(\log_m P)$ versus $\Omega(\log_2 P)$, i.e., for any fixed m, the bounds are the same.

6. Concluding Remarks. It is now clear that in comparing on-line algorithms, the competitive ratio on accommodating sequences can give different information than the competitive ratio. This is true for Fair Bin Packing and two algorithms investigated in this paper, but these results also indicate that the competitive ratio on accommodating sequences and the accommodating function could be very useful measures generally.

With respect to Fair Bin Packing, the choice as to which algorithm to use depends on which ratio is more relevant in a specific situation. This, in turn, would depend on the actual distribution of request sequences. However, one might guess that the competitive ratio on accommodating sequences actually gives the more useful answer in most cases, since the sequences which cause First-Fit to perform so poorly with respect to the competitive ratio are in some sense rather artificial. The sequences are designed so that OPT can arrange to reject certain "difficult" requests, but continue to be "fair". This may simply be a blatant example of how some unusual request sequences can cause the competitive ratio to be excessively pessimistic.

We believe that there is a broad range of on-line problems for which analysis using the competitive ratio on accommodating sequences and the accommodating function will give interesting insights. More of these problems should be investigated. In particular, an open problem left here is finding a minimization problem which has a more gradual change from the competitive ratio on accommodating sequences to the competitive ratio.

In this paper, the accommodating function was only investigated for $\alpha \geq 1$. It was natural to use this restriction, since it spans from the competitive ratio on accommodating sequences to the standard competitive ratio, the two known interesting points. It has later been discovered [8] that the accommodating function is also interesting for $\alpha < 1$. There appear to be more problems which have interesting accommodating functions if one considers $\alpha < 1$, further distinctions can be made between known algorithms, and new interesting algorithms can be developed.

Two of the upper bounds proven in this paper have since been improved. The upper bound on First-Fit's competitive ratio on accommodating sequences for Fair Bin Packing has been improved to $\frac{5}{8}$ [1]. Thus, we can conclude that the competitive ratio of First-Fit on accommodating sequences is exactly $\frac{5}{8}$. In addition, it has been shown [3] that for any deterministic algorithm for the fair Unit Price Seat Reservation Problem with 3 seats, the competitive ratio on accommodating sequences is at most $\frac{1}{2} + \frac{3}{k+5}$, where $k \geq 7$ and $k \equiv 1 \pmod 6$. This gives an upper bound of $\frac{1}{2} + \frac{3n-3}{2k+6n-(8+2c)}$ for n seats with the same restrictions on k.

Recently, the restriction of input sequences has proven useful in another context, congestion control on the internet [24], where the competitive ratio is a function of this restriction. It does not appear that the functions in [24] can be viewed as accommodating functions.

7. Acknowledgments. We would like to thank the referees for their careful reading of the paper and their helpful suggestions for improvements.

REFERENCES

[1] Y. Azar, J. Boyar, L. Epstein, L. M. Favrholdt, K. S. Larsen, and M. N. Nielsen, *Fair versus unrestricted bin packing*, Technical Report PP-2000-20, Department of Mathematics and Computer Science, University of Southern Denmark, Odense, 2000, ftp://ftp.imada.sdu.dk/pub/papers/pp-2000/20.ps.gz, preliminary version in [2].

[2] Y. Azar, J. Boyar, L. M. Favrholdt, K. S. Larsen, and M. N. Nielsen, *Fair versus unrestricted bin packing*, in Proc. 7th Scandinavian Workshop on Algorithm Theory, volume 1851 of Lecture Notes in Computer Science, Springer-Verlag, 2000, pp. 200–213.

[3] E. Bach, J. Boyar, L. Epstein, L. M. Favrholdt, T. Jiang, K. S. Larsen, G.-H. Lin, and R. van Stee, *Tight bounds on the competitive ratio on accommodating sequences for the seat reservation problem*, Technical Report PP-2000-16, Department of Mathematics and Computer Science, University of Southern Denmark, Odense, 2000, ftp://ftp.imada.sdu.dk/pub/papers/pp-2000/16.ps.gz, preliminary version in [4].

[4] E. Bach, J. Boyar, T. Jiang, K. S. Larsen, and G.-H. Lin, *Better bounds on the accommodating ratio for the seat reservation problem*, in Proc. 6th Annual International Computing and Combinatorics Conference, volume 1858 of Lecture Notes in Computer Science, Springer-Verlag, 2000, pp. 221–231.

[5] S. Ben-David and A. Borodin, *A new measure for the study of on-line algorithms*, Algorithmica, 11 (1994), pp. 73–91.

[6] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, North Holland, 1976, pp. 72–74.

[7] A. Borodin, S. Irani, P. Raghavan, and B. Schieber, *Competitive paging with locality of reference*, Journal of Computer and System Sciences, 50 (1995), pp. 244–258.

[8] J. Boyar, L. M. Favrholdt, K. S. Larsen, and M. N. Nielsen, *Extending the accommodating function*, submitted, 2001.

[9] J. Boyar and K. S. Larsen, *The seat reservation problem*, Algorithmica, 25 (1999), pp. 403–417.

[10] J. Boyar, K. S. Larsen, and M. N. Nielsen, *The accommodating function — a generalization of the competitive ratio*, in Proc. 6th International Workshop on Algorithms and Data Structures, volume 1663 of Lecture Notes in Computer Science, Springer-Verlag, 1999, pp. 74–79.

[11] M. Chrobak and J. Noga, *LRU is better than FIFO*, Algorithmica, 23 (1999), pp. 180–185.

[12] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, *Approximation algorithms for bin packing: a survey*, in D. S. Hochbaum, editor, Approximation Algorithms for NP-Hard Problems, chapter 2, PWS Publishing Company, 1997, pp. 46–93.

[13] E. G. Coffman, Jr. and J. Y-T. Leung, *Combinatorial analysis of an efficient algorithm for processor and storage allocation*, SIAM Journal on Computing, 8 (1979), pp. 202–217.

[14] E. G. Coffman, Jr., J. Y-T. Leung, and D. W. Ting, *Bin packing: maximizing the number of pieces packed*, Acta Informatica, 9 (1978), pp. 263–271.

[15] J. Csirik and G. Woeginger, On-line packing and covering problems, in G. J. Woeginger, A. Fiat, editor, Online Algorithms, volume 1442 of Lecture Notes in Computer Science, chapter 7, Springer-Verlag, 1998, pp. 147–177.

[16] A. Fiat and G. J. Woeginger, *Competitive odds and ends*, in G. J. Woeginger, A. Fiat, editor, *Online Algorithms*, volume 1442 of Lecture Notes in Computer Science, chapter 17, Springer-Verlag, 1998, pp. 385–394.

[17] R. L. Graham, *Bounds for certain multiprocessing anomalies*, Bell Systems Technical Journal, 45 (1966), pp. 1563–1581.

[18] S. Irani and A. R. Karlin, *Online computation*, in D. S. Hochbaum, editor, Approximation Algorithms for NP-Hard Problems, chapter 13, PWS Publishing Company, 1997, pp. 521–564.

[19] S. Irani, A. R. Karlin, and S. Phillips, *Strongly competitive algorithms for paging with locality of reference*, SIAM Journal on Computing, 25 (1996), pp. 477–497.

[20] T. R. Jensen and B. Toft, *Graph Coloring Problems*, John Wiley & Sons, 1995.

[21] 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 Journal on Computing, 1974, pp. 299–325.

[22] B. Kalyanasundaram and K. Pruhs, *Speed is as powerful as clairvoyance*, in Proc. 36th Annual IEEE Foundations of Computer Science, 1995, pp. 214–221.

[23] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator, *Competitive snoopy caching*, Algorithmica, 3 (1988), pp. 79–119.

[24] R. Karp, E. Koutsoupias, C. Papadimitriou, and S. Shenker, *Optimization problems in congestion control*, in Proc. 41th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, 2000, pp. 66–74.

[25] E. Koutsoupias and C. H. Papadimitriou, *Beyond competitive analysis*, SIAM Journal on Computing, 30 (2001), pp. 300–317.

[26] D. König, *Graphs and matrices*, Mat. Fiz. Lapok, 38 (1931), pp. 116–119. In Hungarian.

[27] S. Leonardi and D. Raz, *Approximating total flow time on parallel machines*, in Proc. 29th

Annual ACM Symp. on the Theory of Computing, 1997, pp. 110–119.
[28] C. A. Phillips, C. Stein, E. Torng, and J. Wein, *Optimal time-critical scheduling via resource augmentation*, in Proc. 29th Annual ACM Symp. on the Theory of Computing, 1997, pp. 140–149.
[29] D. D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, Comm. of the ACM, 28 (1985), pp. 202–208.
[30] E. Torng, *A unified analysis of paging and caching*, Algorithmica, 20 (1998), pp. 175–200.
[31] N. Young, *The k-server dual and loose competitiveness for paging*, Algorithmica, 11 (1994), pp. 525–541.