SIAM J. COMPUT.
Vol. 7, No. 1, February 1978

AN APPLICATION OF BIN-PACKING TO MULTIPROCESSOR SCHEDULING*

E. G. COFFMAN, JR.,† M. R. GAREY‡ AND D. S. JOHNSON‡

Abstract. We consider one of the basic, well-studied problems of scheduling theory, that of nonpreemptively scheduling n independent tasks on m identical, parallel processors with the objective of minimizing the "makespan," i.e., the total timespan required to process all the given tasks. Because this problem is *NP*-complete and apparently intractable in general, much effort has been directed toward devising fast algorithms which find near-optimal schedules. The well-known LPT (Largest Processing Time first) algorithm always finds a schedule having makespan within $4/3 = 1.333\cdots$ of the minimum possible makespan, and this is the best such bound satisfied by any previously published fast algorithm. We describe a comparably fast algorithm, based on techniques from "bin-packing," which we prove satisfies a bound of 1.220. On the basis of exact upper bounds determined for each $m \leqq 7$, we conjecture that the best possible general bound for our algorithm is actually $20/17 = 1.176\cdots$.

Key words. bin packing, multiprocessor scheduling, approximation algorithms, worst-case analysis, performance bounds

1. Introduction. One of the fundamental problems of deterministic scheduling theory is that of scheduling independent tasks on a nonpreemptive multiprocessor system so as to minimize overall finishing time [1, Chap. 5], [2], [3]. Formally, we are given a set $\mathcal{T} = \{T_1, T_2, \cdots, T_n\}$ of *tasks*, each task T_i having *length* $l(T_i)$, and a set of $m \geqq 2$ identical *processors*. A *schedule* in this case can be thought of as a partition $\mathcal{P} = \langle P_1, P_2, \cdots, P_m \rangle$ of $\mathcal{T}$ into m disjoint sets, one for each processor. The ith processor, $1 \leqq i \leqq m$, executes the tasks in P_i. (Since the tasks are assumed to be independent, we may restrict our attention to schedules in which no idle time is inserted between consecutively executed tasks. For the same reason, the particular sequence in which tasks are executed on a processor is unconstrained and irrelevant for this problem.) The *finishing time* for the schedule $\mathcal{P}$ is then given by

$$f(\mathcal{P}) = \max_{1 \leqq i \leqq m} l(P_i)$$

where for any $X \subseteq \mathcal{T}$, $l(X)$ is defined to be $\sum_{T \in X} l(T)$.

An *optimum* m-processor schedule $\mathcal{P}^*$ is one that satisfies $f(\mathcal{P}^*) \leqq f(\mathcal{P})$ for all partitions $\mathcal{P}$ of $\mathcal{T}$ into m subsets. Since there are only a finite number of possible partitions, such an optimum schedule must exist, and we let $\mathcal{T}_m^* = f(\mathcal{P}^*)$ denote its finishing time.

The problem of finding an optimum schedule appears to be quite difficult in general. All known algorithms require computation time that grows exponentially with the number of tasks. This is not surprising, however, in light of the fact that

* Received by the editors September 28, 1976.

† Department of Computer Science, Pennsylvania State University, University Park, Pennsylvania 16802.

‡ Bell Laboratories, Murray Hill, New Jersey 07974.

this problem is known to be "*NP*-complete" and hence computationally "equivalent" to a host of other notoriously intractable problems [8]. These computational difficulties force us to lower our sights somewhat and seek instead reasonably efficient algorithms that find "near-optimal" schedules.

Let A be an algorithm that, when given $\mathcal{T}$ and m, constructs a partition $\mathcal{P}_A[\mathcal{T}, m]$ of $\mathcal{T}$ into m subsets. We shall use $F_A[\mathcal{T}, m]$ to denote the finishing time of $\mathcal{P}_A[\mathcal{T}, m]$, i.e., $F_A[\mathcal{T}, m] = f(\mathcal{P}_A[\mathcal{T}, m])$. The *m-processor performance ratio* for A is then defined by

$$R_m(A) = \sup\left\{\frac{F_A[\mathcal{T}, m]}{\mathcal{T}_m^*} : \text{all task sets } \mathcal{T}\right\}.$$

We would like to find an efficient algorithm A such that $R_m(A)$ is as close to 1 as possible, for all $m \geqq 2$ (the problem is trivial for $m = 1$).

In [2], [3], R. L. Graham describes a sequence $A_1, A_2, \cdots$ of algorithms such that $\lim_{i\to\infty} R_m(A_i) = 1$ for all $m \geqq 2$. Unfortunately these algorithms require computation time growing exponentially with m and become more and more like exhaustive search as the guaranteed accuracy improves. (Sahni presents similarly behaved algorithms in [7]). The best of the previously published *polynomial-time* algorithms, also in [2], [3], is the LPT algorithm, which satisfies

$$R_m(\text{LPT}) = \frac{4}{3} - \frac{1}{3m}.$$

In this paper we present a simple iterative algorithm, based on ideas from bin packing [5], [6], which substantially improves on this worst-case performance and also seems to outperform LPT on the average.[1]

In the next section we discuss the bin-packing problem and the well known first fit decreasing algorithm for it. We then describe the new results about this algorithm which have motivated the design of our scheduling algorithm and helped us produce upper bounds on its worst case behavior. The scheduling algorithm itself, called MULTIFIT, is then presented in § 3, along with our results about it. The remainder of the paper, §§ 4 and 5, are devoted to the proofs of the new bin-packing results.

2. Bin-packing. Our scheduling algorithm is based on the bin-packing algorithm first fit decreasing (abbreviated FFD). The bin-packing problem is in a sense the dual problem to the scheduling problem defined above. Given $\mathcal{T}$ as before, and a bound C, a *packing* is a partition $\mathcal{P} = \langle P_1, P_2, \cdots, P_m\rangle$ of $\mathcal{T}$ such that $l(P_i) \leqq C$, $1 \leqq i \leqq m$. The tasks T_i are here thought of as *items* with *size* $l(T_i)$, which are placed in *bins* of *capacity* C. Our goal is to minimize the number m of bins used in the packing. We let $OPT[\mathcal{T}, C]$ denote this minimum possible value of m. As before, the problem of finding an optimum packing appears to be intractable. The algorithm FFD is an attempt to find a near-optimum packing quickly. It constructs a packing $\mathcal{P}_{FFD}[\mathcal{T}, C]$ as follows.

[1] The algorithm described here is also an improvement, both in worst-case behavior and experimental average case behavior, over the earlier version described in [4].

First, the items in $\mathcal{T}$ are put in "decreasing order," that is, they are re-indexed so that $l(T_1) \geqq l(T_2) \geqq \cdots \geqq l(T_n)$. Then a packing is built up by treating each item in succession, and adding it to the lowest indexed bin into which it will fit without violating the capacity constraint. More formally, we might describe the packing procedure as follows:

1. Set $P_i \leftarrow \varphi$, $1 \leqq i \leqq n$, and $j \leftarrow 1$.
2. Set $k \leftarrow \min\{i \geqq 1: l(P_i) + l(T_j) \leqq C\}$.
3. Set $P_k \leftarrow P_k \cup \{T_j\}$, $j \leftarrow j+1$.
4. If $j \leqq n$, go to 2. Otherwise, halt.

Let us denote by $FFD[\mathcal{T}, C]$ the number m of nonempty bins in $\mathcal{P}_{FFD}[\mathcal{T}, C]$. It was proved in [5], [6] that for all $\mathcal{T}$ and C

$$FFD[\mathcal{T}, C] \leqq \frac{11}{9} OPT[\mathcal{T}, C] + 4.$$

In the following, however, we shall be interested in a different question about FFD: namely, "given $\mathcal{T}$ and m, how large does C have to be so that $FFD[\mathcal{T}, C] \leqq m$?"

This question will of course have many different answers, depending on $\mathcal{T}$ and m. We are interested, however, in finding an answer that works for *all* $\mathcal{T}$ and m. To obtain such an answer, we shall ask "how large does C have to be, *in terms of* $\mathcal{T}^*_m$, so that, for all $\mathcal{T}$ and m, $FFD[\mathcal{T}, C]$ is guaranteed to be m or less?"

Recalling our definition of $\mathcal{T}^*_m$ from § 1, we note that, in terms of the bin packing problem,

$$\mathcal{T}^*_m = \min\{C: OPT[\mathcal{T}, C] \leqq m\}.$$

Thus $\mathcal{T}^*_m$ is the smallest bin capacity which allows $\mathcal{T}$ to be packed into m or fewer bins. We now define

$$r_m = \inf\{r: \text{for all } \mathcal{T}, FFD[\mathcal{T}, r\mathcal{T}^*_m] \leqq m\}.$$

The intent of this definition is that r_m be the least "expansion factor" by which the optimum bin capacity should be enlarged to guarantee that FFD will use no more than m bins. However, the definition allows for the possibility that, while expansion factors arbitrarily close to r_m will work, the limiting value r_m itself does not. That this cannot happen (and hence that "inf" could have been replaced by "min") is proved as part of the following "monotonicity lemma."

LEMMA 2.1. *For every* $\mathcal{T}$ *and any* $r \geqq r_m$, $FFD[\mathcal{T}, r\mathcal{T}^*_m] \leqq m$.

Proof. We first show that $FFD[\mathcal{T}, r_m\mathcal{T}^*_m] \leqq m$ for every $\mathcal{T}$. Suppose, to the contrary, that $\mathcal{T}$ is a set for which $FFD[\mathcal{T}, r_m\mathcal{T}^*_m] > m$. Consider the application of the FFD algorithm to $\mathcal{T}$ with capacity $C = r_m\mathcal{T}^*_m$. Each time the algorithm places a particular item in a bin other than the first bin, this is because the size of that item would have caused the total size in each lower-indexed bin to exceed C by some positive amount. Let δ be the least such excess over all items and all such "unsuccessful" attempted placements. Then, for every capacity C', $C \leqq C' < C + \delta$, exactly the same packing will be obtained with bin capacity C' as with C, and hence $FFD[\mathcal{T}, C'] > m$. However, by the definition of r_m, we know that there exist ratios r arbitrarily close to r_m for which $FFD[\mathcal{T}, r\mathcal{T}^*_m] \leqq m$. Choosing such an

r with $r\mathcal{T}_m^* < C + \delta$, we obtain a contradiction to the assumed counter-example, proving that $FFD[\mathcal{T}, r_m\mathcal{T}_m^*] \leqq m$.

Now suppose that, for some $\mathcal{T}$ and $\alpha > 1$, $FFD[\mathcal{T}, \alpha r_m\mathcal{T}_m^*] > m$. We shall use $\mathcal{T}$ to construct a set $\bar{\mathcal{T}}$ for which $FFD[\bar{\mathcal{T}}, r_m\bar{\mathcal{T}}_m^*] > m$, contradicting what we have just proved. Consider any optimal packing of $\mathcal{T}$ into m bins of size $\mathcal{T}_m^*$. Enlarge the capacity of each optimal bin to $\alpha\mathcal{T}_m^*$ and augment $\mathcal{T}$ to form $\bar{\mathcal{T}}$ by adding new items, each smaller than the smallest item in $\mathcal{T}$, so that every enlarged bin becomes completely filled. Thus we have that $\bar{\mathcal{T}}_m^* = \alpha\mathcal{T}_m^*$. Furthermore $FFD[\bar{\mathcal{T}}, \alpha r_m\mathcal{T}_m^*] > m$ because the $(m+1)$st bin will be started before any of the new items are placed. But then we have $FFD[\bar{\mathcal{T}}, r_m\bar{\mathcal{T}}_m^*] > m$, which is the desired contradiction. The lemma follows. □

Thus r_m is indeed the desired minimum "expansion factor" and every capacity greater than $r_m\mathcal{T}_m^*$ will also work, so that we need not be concerned about possible anomalies. By determining the values of r_m, $m \geqq 2$, we will therefore be obtaining general answers to the question "how large does C have to be, in terms of $\mathcal{T}_m^*$?"

Table 1 gives the best upper and lower bounds we have discovered for r_m. As can be seen, we know the exact value of r_m for $2 \leqq m \leqq 7$, and our upper and lower bounds are quite close for $m \geqq 8$. Since $r_4 = r_5 = r_6 = r_7 = \frac{20}{17}$, and the best lower bound we have been able to find for $m \geqq 8$ is also $\frac{20}{17}$, we conjecture that in fact $r_m = \frac{20}{17}$ for all $m \geqq 4$.

The proofs of the lower bounds cited in Table 1 are straightforward, since all that is required is an example $\mathcal{T}$ such that $FFD(\mathcal{T}, C) > m$ whenever $C <$ (lower bound) $\cdot\, \mathcal{T}_m^*$. These lower bound examples for $m = 2$, $m = 3$, and $m \geqq 4$ are illustrated in Figs. 1, 2, and 3. In each case the items are represented by rectangles which are labeled by their sizes, and the bins by stacks of items.

The proofs of the upper bounds for $m \leqq 7$ involve case analyses which grow more and more complicated as m increases. These proofs have been omitted at the suggestion of the referees, but a full version of this paper containing them is available from the authors. We shall limit ourselves here to proving the general 1.220 upper bound, which in fact holds for all $m \geqq 2$. The proof will be postponed until §§ 4 and 5, however, so that we may first see how we incorporate FFD into a scheduling algorithm, and how we use the values of r_m to bound the worst-case behavior of that algorithm.

TABLE 1
Bounds on r_m

m	2	3	4, 5, 6, 7	8 or more
Upper bound on r_m	$\frac{8}{7}$	$\frac{15}{13}$	$\frac{20}{17}$	1.220
Lower bound on r_m	$\frac{8}{7}$	$\frac{15}{13}$	$\frac{20}{17}$	$\frac{20}{17} = 1.176\cdots$

3. The algorithm MULTIFIT. In the light of the results described in the preceding section, one might propose the following scheduling algorithm: Given $\mathcal{T}$ and m, set $C = r_m \cdot \mathcal{T}_m^*$ and run FFD on $\mathcal{T}$ and C. By definition of r_m,

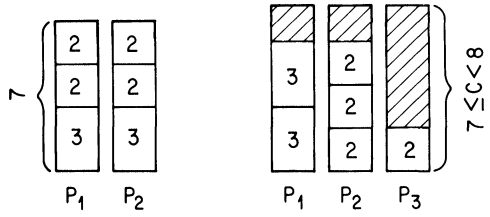

FIG. 1. *Example showing* $r_2 \geqq \frac{8}{7}$

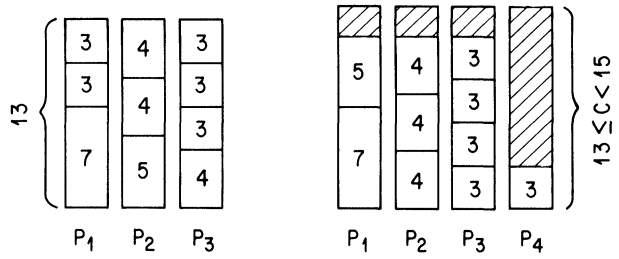

FIG. 2. *Example showing* $r_3 \geqq \frac{15}{13}$

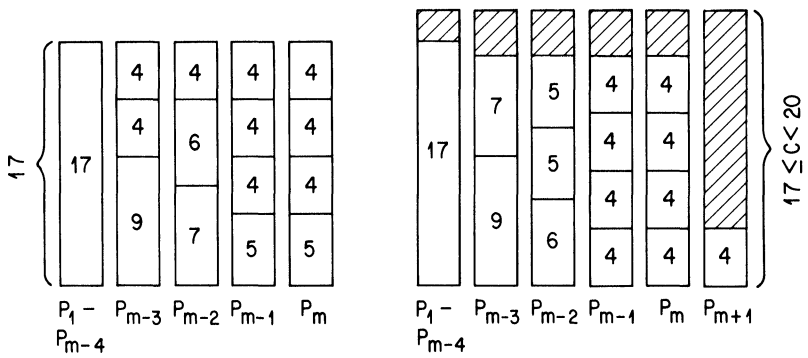

FIG. 3. *Example showing* $r_m \geqq \frac{20}{17}$, *for all* $m \geqq 4$

$\mathcal{P}_{FFD}[\mathcal{T}, C]$ will be made up of m or fewer sets, and hence will correspond to a valid schedule for $\mathcal{T}$ and m with finishing time $C = r_m \cdot \mathcal{T}^*_m$ or less, which would certainly be "near-optimal" in light of our bounds on r_m.

Unfortunately this approach depends on knowing the value of $\mathcal{T}^*_m$ in advance, and it is just as difficult to determine $\mathcal{T}^*_m$ as to find an optimum schedule. A second idea would be to make repeated trials with FFD and different values of C until we find the least C for which $FFD[\mathcal{T}, C] \leqq m$, and take the schedule resulting from this bin capacity. If we assume that all tasks have integer lengths, there are two natural ways to attempt this. One would be to try each integer C in turn, starting from some obvious lower bound on $\mathcal{T}^*_m$, until we found one for which $FFD(\mathcal{T}, C) \leqq m$; this would be the desired minimum value of C. Unfortunately, this procedure might require a large number of repeated trials of FFD and would be very costly in terms of running time. A common way of reducing the number of trials in such a search is to use *binary search*: Start with known upper and lower bounds on C, and at each step run FFD for a value of C midway between the current upper and lower bounds. If $FFD[\mathcal{T}, C] > m$, C becomes the new lower bound and we continue; if $FFD[\mathcal{T}, C] \leqq m$, C becomes the new upper bound. At each step we thus halve the potential range and so we should narrow in on the minimum value of C quite rapidly.

Unfortunately, binary search will only be guaranteed to do this if for every $C_1 < C_2$, $FFD[\mathcal{T}, C_1] \leqq m$ implies $FFD[\mathcal{T}, C_2] \leqq m$. Although this general monotonicity property does hold for $m = 2$, it fails for $m \geqq 3$. Figure 4 demonstrates that there exists a list $\mathcal{T}$ such that $FFD[\mathcal{T}, 60] = 3$ but $FFD[\mathcal{T}, 61] = 4$. Thus binary search is *not* guaranteed to find the least C such that $FFD[\mathcal{T}, C] \leqq m$. However, recall that Lemma 2.1 tells us that binary search *will* be guaranteed to narrow in on a capacity $C \leqq r_m \mathcal{T}^*_m$. Therefore, binary search can still be used to find a near-optimum schedule, and this is the approach taken in our algorithm MULTIFIT.

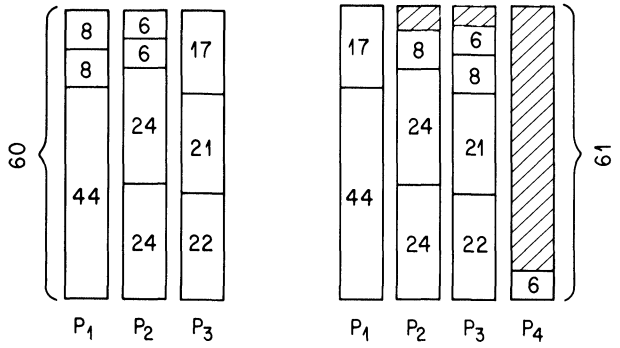

FIG. 4. $\mathcal{T}$ *such that* $FFD[\mathcal{T}, 61] > FFD[\mathcal{T}, 60]$

We begin our description of MULTIFIT by specifying the initial lower and upper bounds, C_L and C_U, used in the binary search. These are given in the following two lemmas.

LEMMA 3.1. *Let* $C_L[\mathcal{T}, m] = \max\{(1/m)l(\mathcal{T}), \max_i\{l(T_i)\}\}$. *Then for all* $C < C_L[\mathcal{T}, m]$, $FFD[\mathcal{T}, C] > m$.

Proof. This follows from the obvious fact that $\mathcal{T}^*_m \geqq C_L[\mathcal{T}, m]$.

LEMMA 3.2. *Let* $C_U[\mathcal{T}, m] = \max\{(2/m)l(\mathcal{T}), \max_i\{l(T_i)\}\}$. *Then for all* $C \geqq C_U[\mathcal{T}, m]$, $FFD[\mathcal{T}, C] \leqq m$.

Proof. Suppose $C \geqq C_U[\mathcal{T}, m]$ and $FFD[\mathcal{T}, C] > m$, and assume without loss of generality that $l(T_1) \geqq l(T_2) \geqq \cdots \geqq l(T_n)$, that is, $\mathcal{T}$ is indexed in nonincreasing order by item size. Let T_j be the first item to go in bin P_{m+1} under FFD. Clearly we must have $j \geqq m+1$. If $l(T_j) > C/2$, then since $\mathcal{T}$ is indexed in nonincreasing order $l(T_i) > C/2$, $1 \leqq i \leqq m+1$. This implies

$$l(\mathcal{T}) > \frac{mC}{2} \geqq \frac{mC_U[\mathcal{T}, m]}{2} \geqq l(\mathcal{T}),$$

a contradiction. If on the other hand $l(T_j) \leqq C/2$, then by the fact that T_j did not fit in any of the first m bins, each must have contained items whose total size exceeded $C/2$, and we again have $l(\mathcal{T}) > mC/2$ and a contradiction. □

We are now prepared to give a precise description of MULTIFIT. MULTIFIT takes as input a task set $\mathcal{T}$, a number of processors m, and a bound k on the desired number of iterations. Its first step is to put $\mathcal{T}$ into nonincreasing order, so that all subsequent applications of FFD will not have to reorder $\mathcal{T}$ themselves. It then proceeds by binary search for the desired number of iterations as follows:

1. Set $CL(0) \leftarrow C_L[\mathcal{T}, m]$;
   $CU(0) \leftarrow C_U[\mathcal{T}, m]$;
   $I \leftarrow 1$.
2. If $I > k$, halt.
   Otherwise, set $C \leftarrow [CL(I-1) + CU(I-1)]/2$.
3. If $FFD[\mathcal{T}, C] \leqq m$, set $CU(I) \leftarrow C$;
   $CL(I) \leftarrow CL(I-1)$;
   $I \leftarrow I+1$;
   and go to 2.
4. If $FFD[\mathcal{T}, C] > m$, set $CL(I) \leftarrow C$;
   $CU(I) \leftarrow CU(I-1)$;
   $I \leftarrow I+1$;
   and go to 2.

The final value $CU(k)$ gives the smallest value of C found for which $FFD[\mathcal{T}, C] \leqq m$. Either the corresponding schedule has been generated along the way (and could have been saved, storage space permitting), or else it has not yet been generated because $CU(k) = C_U[\mathcal{T}, m]$. In either case, the schedule can now be generated by a single additional application of FFD, and this schedule is the output of MULTIFIT.

Before beginning our analysis of the worst case behavior of MULTIFIT, let us first examine how long it takes to run.

We first note that the application of FFD referred to in steps 3 and 4 need not be run to completion if $FFD[\mathcal{T}, C] > m$, but can be halted as soon as the first item

enters bin P_{m+1}. This, plus the fact that $\mathcal{T}$ is already in decreasing order, means that each application of FFD need only take $O(nm)$ steps (and can actually be implemented to take $O(n \log m)$ steps using techniques described in [5]). Thus the total time for MULTIFIT, including the initial sorting of $\mathcal{T}$ by size and the k iterations of FFD is $O((n \log n)+knm)$ or $O((n \log n)+(kn \log m))$, depending on implementation. This is to be compared to the time required to run the LPT algorithm of [2], [3]. LPT involves the same initial sorting step, followed by a packing procedure which is quite comparable to one execution of FFD's packing procedure, and also can be implemented to run in either $O(nm)$ or $O(n \log m)$ time. Thus LPT does have a slight advantage over MULTIFIT in timing when $k>1$. However, as we shall see, there is apparently no reason to choose $k>7$ in applying MULTIFIT, and thus for large values of n, both algorithms will have execution times dominated by the time for the initial sort, and hence will be comparable. Of course, for small values of n both algorithms are so fast as to make matters of relative timing purely academic.

We now turn to the question of how the two algorithms compare as to the quality of the schedules they produce. We have already presented in § 1 the main result about the worst-case behavior of LPT. For MULTIFIT, our main result is presented in Theorem 3.1, where $MF(k)$ stands for the version of MULTIFIT in which k is the specified number of iterations.

THEOREM 3.1. *For all* $m \geq 2$, $k \geq 0$, $R_m(MF[k]) \leq r_m + (\frac{1}{2})^k$.

Proof. Suppose m and k are such that $R_m(MF[k])>r_m+(\frac{1}{2})^k$. Recall from the definition of $R_m(A)$ in § 1 that this means there exists a $\mathcal{T}$ such that when $MF[k]$ is applied to $\mathcal{T}$ and m, the resulting schedule has finishing time exceeding $(r_m+(\frac{1}{2})^k)\cdot \mathcal{T}^*_m$. Since this finishing time cannot exceed the final value $CU(k)$, we thus have

$$CU(k) \geq (r_m+(\tfrac{1}{2})^k)\cdot \mathcal{T}^*_m. \tag{3.1}$$

Consider the final value $CL(k)$. By the process of binary search, $CU(k)-CL(k)=(\frac{1}{2})^k\cdot(C_U[\mathcal{T},m]-C_L[\mathcal{T},m]) \leq (\frac{1}{2})^k\cdot \mathcal{T}^*_m$, since $C_U[\mathcal{T},m] \leq 2C_L[\mathcal{T},m]$ and $C_L[\mathcal{T},m] \leq \mathcal{T}^*_m$. Thus we must have

$$CL(k) \geq r_m \cdot \mathcal{T}^*_m. \tag{3.2}$$

But then, since $r_m>1$ for $m \geq 2$, this means that $CL(k)>C_L[\mathcal{T},m]$. This implies that FFD must have been performed with bin capacity $CL(k)$ at some point during the operation of the algorithm, and yielded $FFD[\mathcal{T}, CL(k)]>m$. However, this is impossible as, in light of (3.2), it violates Lemma 2.1. □

Combining Theorem 3.1 with the bounds on r_m presented in Table 1, we present in Table 2 a comparison of the worst case behavior of $MF[k]$ and LPT for various values of m and k. The lower bound on $R_m(MF[k])$ for all k which is presented in the last row of the table follows from the same examples (Figs. 1, 2, and 3) used to provide lower bounds on r_m.

From the table it can be seen that $MF[5]$ has better worst-case behavior than LPT for all $m \geq 3$ and $MF[6]$ and $MF[7]$ are better for all $m \geq 2$. Further improvement is possible by choosing larger values of k, but for $k=7$ the bound is already quite close to the limiting value, except for the cases where $m \geq 8$ and our bounds on r_m are not tight. If our conjecture that $r_m = \frac{20}{17}$ for all $m \geq 8$ is true, then

TABLE 2
A comparison of worst case bounds

$m=$	2	3	4	5	10	50
$R_m(\text{LPT})=$	1.167	1.222	1.250	1.267	1.300	1.327
$R_m(MF[5])\leqq$	1.174	1.185	1.208	1.208	1.251	1.251
$R_m(MF[6])\leqq$	1.158	1.169	1.192	1.192	1.236	1.236
$R_m(MF[7])\leqq$	1.151	1.162	1.184	1.184	1.228	1.228
$R_m(MF[k])\geqq$	1.143	1.154	1.176	1.176	1.176	1.176

the upper bounds on $R_m(MF[k])$ for $m=10$ and $m=50$ would be the same as those for $m=4$ and $m=5$.

We remind the reader that the figures in Table 2, attractive as they are, are only *worst-case* bounds. In practice the algorithms can be expected to do much better. To get a feel for how much improvement might be expected, and how the algorithms might compare as to average-case behavior , three limited simulation tests were run, each covering 10 task sets with $m=10$. In Test 1, each task set consisted of 30 tasks with sizes chosen independently according to a uniform distribution between 0 and 1. In Test 2 each task set again consisted of 30 tasks, this time with sizes being the sum of 10 independent choices from the uniform distribution, to simulate a normal distribution. For both these tests $C_L[\mathcal{T}, m]$ was taken as an estimate of $\mathcal{T}^*_m$, and since this could well have been an under-estimate, the values for $F_A[\mathcal{T}, m]/\mathcal{T}^*_m$ might be somewhat inflated. To circumvent this difficulty, in Test 3 we generated our task sets so that $\mathcal{T}^*_m$ was known in advance. We started with 10 tasks of size 1, divided each independently into 2, 3, or 4 subtasks, whose relative sizes were also randomly chosen. $\mathcal{T}$ consisted of the approximately 30 subtasks so constructed, and we automatically had $\mathcal{T}^*_{10}=1$. (As a matter of curiosity, we might note that in no case did any of the algorithms reconstruct an optimum schedule, though all came close.)

Our results are summarized in Table 3, which for each test gives the average value of $F_A[\mathcal{T}, m]/\mathcal{T}^*_m$(or our possibly inflated estimate of it) for LPT, $MF[7]$, and $MF[7]'$, where the last-named algorithm is the earlier and inferior version of MULTIFIT described in [4].

TABLE 3
Average values of $F_A[\mathcal{T}, m]/\mathcal{T}^*_m$

TEST	1	2	3
LPT	1.074	1.023	1.051
$MF[7]$	1.024	1.023	1.016
$MF[7]'$	1.033	1.065	1.021

Due to the limited and somewhat arbitrary nature of these simulations, one should not be prepared to draw far-reaching conclusions from them. However, we might note that, although $MF(7)$ and LPT came out in a dead heat in Test 2, the improvement provided by $MF[7]$ over LPT is clear in the other two tests, where in fact $MF[7]$ found a better schedule in 19 out of the 20 cases, the remaining case being a tie. It also might be noted that $MF[7]$ always found its ultimate schedule by the 6th iteration, so that $MF[6]$ could just as well have been used, for a slight saving in running time.

4. Upper bound proofs: Preliminaries. The problem of proving an upper bound on r_m is considerably simplified if we can focus our attention on a "normalized" situation. In this section we show how this can be done, and thus set up a framework for such proofs. We illustrate this framework by using it directly to prove an easy upper bound of $r_m \leqq 5/4 = 1.250$ for all $m \geqq 2$. In the next section we show how more sophisticated analysis can yield a 1.220 general bound. The same basic framework can also be used as a starting point when proving the exact upper bounds for $m \leqq 7$ summarized above in Table 1.

In what follows we shall be dealing primarily with sets $\mathcal{T}$ of items which are ordered by size. It is therefore convenient to define formally a *list* to be an ordered set $\mathcal{T} = \{T_1, T_2, \cdots, T_n\}$ such that $l(T_1) \geqq l(T_2) \geqq \cdots \geqq l(T_n)$. For any pair of integers $p \geqq q$, we define a (p/q)-*counterexample* to be a list $\mathcal{T}$ and number M of bins such that

$$FFD(\mathcal{T}, p) > M \geqq OPT(\mathcal{T}, q).$$

Thus, though it is possible to pack the items of $\mathcal{T}$ into M bins of capacity q, FFD is unable even to pack them into M bins of the larger capacity p. It is easy to see that $r_m > p/q$ implies the existence of a (p/q)-counterexample with $M = m$.

A *minimal* (p/q)-*counterexample* consists of a list $\mathcal{T}$ and number M of bins such that all of the following hold:

(a) $\mathcal{T}$ and M form a (p/q)-counterexample;

(b) For all lists $\mathcal{T}'$ satisfying (a), $|\mathcal{T}'| \geqq |\mathcal{T}|$;

(c) For all m, $1 \leqq m < M$, $r_m \leqq p/q$.

Thus a (p/q)-counterexample is minimal if there exists neither a (p/q)-counterexample for the same number of bins having fewer items nor a (p/q)-counterexample for a smaller number of bins. It is not difficult to see that the existence of a (p/q)-counterexample implies the existence of a minimal (p/q)-counterexample. It follows that if $r_m > p/q$, there must exist a minimal (p/q)-counterexample having $M \leqq m$.

In a normalized proof of an upper bound r on r_m we assume the existence of a minimal (p/q)-counterexample, for convenient p and q satisfying $r = p/q$, and derive a contradiction from that assumption. In such proofs we will be able to use many general properties which must be obeyed by such a minimal counterexample. We now turn to the task of deriving some of these properties.

We shall assume for the rest of this section that the list $\mathcal{T}$ and number m of processors provide a minimal (p/q)-counterexample. Our first observation is an immediate consequence of the definitions.

LEMMA 4.1. *$\mathcal{T}$ and m must satisfy the following*:

(a) *$FFD(\mathcal{T}, p) = m+1$ and $OPT(\mathcal{T}, q) = m$*;

(b) *All items on $\mathcal{T}$, except the last, are assigned to the first m bins by FFD.*

In all that follows, we shall let $\mathcal{P} = \langle P_1, P_2, \cdots, P_{m+1}\rangle$ be the FFD packing of $\mathcal{T}$ into bins of capacity p and $\mathcal{P}^* = \langle P_1^*, P_2^*, \cdots, P_m^*\rangle$ be the optimal packing of $\mathcal{T}$ into bins of capacity q. For subsets X and Y of $\mathcal{T}$, we shall say that *X dominates Y* if there is a 1–1 mapping f: $Y \to X$ such that $l(y) \leqq l(f(y))$ for all $y \in Y$. Using this concept and the following lemma, we can draw some interesting conclusions about $\mathcal{P}$ and $\mathcal{P}^*$.

LEMMA 4.2 (cancellation lemma). *Let I, $J \subseteq \{1, 2, \cdots, m+1\}$ be such that $|I| = |J| = k > 0$. Then the set $X = \bigcup_{i\in I} P_i$ cannot dominate the set $Y = \bigcup_{j\in J} P_j^*$.*

Proof. Suppose X dominates Y, and let f: $Y \to X$ be the mapping involved. Consider the list $\mathcal{T}' = \mathcal{T} - X$ obtained by deleting the items of X from $\mathcal{T}$ while retaining the same relative ordering of the remaining items. Since we have deleted *exactly* those items which were contained in the bins P_i, $i \in I$, the FFD packing of $\mathcal{T}'$ will be identical to that for $\mathcal{T}$ except that those bins will be missing. Thus $FFD(\mathcal{T}', p) = m - k + 1$. Now consider the packing $\mathcal{P}^*$, and construct a new packing $\mathcal{P}'$ by interchanging each item $y \in Y$ with its image $f(y)$. Since $l(y) \leqq l(f(y))$, we thus must have $l(P_i') \leqq l(P_i^*)$ for all $i \notin J$, $1 \leqq i \leqq m$. Moreover, bins P_j', $j \in J$, contain only items of X (although they may have $l(\mathcal{P}_j') > q$). Thus by deleting all elements of X from $\mathcal{P}'$, we obtain a packing of $\mathcal{T}'$ into at most $m-k$ bins of capacity q. Hence $OPT(\mathcal{T}', q) \leqq m-k$, and since $m-k < m$, this contradicts the presumed minimality of $\mathcal{T}$ and m. □

LEMMA 4.3. $|\mathcal{P}_i^*| \geqq 3$, $1 \leqq i \leqq m$.

Proof. Suppose first that $P_i^* = \{x\}$. Let j be such that $x \in P_j$. Then P_j dominates P_i^*, contrary to Lemma 4.2. Suppose next that $P_i^* = \{x, y\}$, where x precedes y in the (decreasing) ordering of $\mathcal{T}$. Suppose $x \in P_j$ and $y \in P_k$ in the FFD packing. If $j = k$, then P_j dominates P_i^*, a contradiction. Suppose $j < k$. Then the fact that y went to the right of the bin containing x, even though $l(x) + l(y) \leqq q < p$, means that P_j must have contained a second item z, in addition to x, when y was assigned. Thus z preceded y in $\mathcal{T}$ and so $l(z) \geqq l(y)$, and P_j dominates P_i^*, again a contradiction. Finally, suppose $j > k$. Then, since y follows x in $\mathcal{T}$, it cannot be the first item to go in P_k; call that first item z. Since z is the first item in a bin to the left of the one containing x, z must have preceded x in $\mathcal{T}$ and hence $l(z) \geqq l(x)$. Therefore P_k dominates P_i^*, giving us our final contradiction. □

LEMMA 4.4. $|P_i| \geqq 2$, $1 \leqq i \leqq m$.

Proof. Suppose $P_i = \{x\}$, for some i, $1 \leqq i \leqq m$. Then we must have $l(x) + l(T_n) > p$, where T_n is the last, and hence smallest, item in $\mathcal{T}$. But this means that $l(x) + l(y) > q$ for all $y \in \mathcal{T}$, so if P_j^* is the optimal bin containing x, $|P_j^*| = 1$, contrary to Lemma 4.3. □

The next lemmas obtain bounds on the item sizes. Let $\alpha = l(T_n)$ denote the size of the smallest item.

LEMMA 4.5. $\alpha > (m/(m-1))(p-q)$.

Proof. For $1 \leqq i \leqq m$, since T_n did not fit in P_i, we have $l(P_i) + \alpha > p$. Hence

$$\sum_{i=1}^{m} l(P_i) + m\alpha > mp.$$

However, since all items are contained in m bins of capacity q in $\mathscr{P}^*$,

$$\sum_{i=1}^{m} l(P_i)+\alpha < mq.$$

The lemma follows. □

LEMMA 4.6. *For all* $T_i \in \mathscr{T}$, $l(T_i) \leqq q-2\alpha$.

Proof. This follows immediately from Lemma 4.3, which implies that T_i must be in a bin with at least two other items in $\mathscr{P}^*$. □

At this point, we can already illustrate the use of these lemmas by proving an easy upper bound on r_m.

THEOREM 4.1. *For all* $m \geqq 2$, $r_m \leqq 5/4$.

Proof. Suppose $\mathscr{T}$ and m provide a minimal $(5/4)$-counterexample. By Lemma 4.5, $l(T_n)=\alpha>1$. Thus no optimal bin can contain more than three items. By Lemma 4.3 this means that $|P_i^*|=3$, $1 \leqq i \leqq m$, and hence $|\mathscr{T}|=3m$. By Lemmas 4.1 and 4.4, this means that there must be a P_i, $1 \leqq i \leqq m$, with $|P_i|=2$. Let $P_i=\{x, y\}$. Since T_n did not go in P_i, we must have $l(x)+l(y)+\alpha>5$. Thus by Lemma 4.3 we must have

$$5<2(q-2\alpha)+\alpha=8-3\alpha.$$

This implies $\alpha<1$, a contradiction. □

In order to prove stronger results, we will need to take a more detailed look at the FFD packing $\mathscr{P}$ of our minimal (p/q)-counterexample. Let us label the items of $\mathscr{T}$ according to their assigned locations in $\mathscr{P}$ as follows: If $|P_i|=k$, then the elements of P_i are denoted by $P_i[j]$, $1 \leqq j \leqq k$, in the order in which they were assigned to P_i. $P_i[1]$ is the first item assigned (the earliest in the ordering of $\mathscr{T}$), and so on. A bin P_i, $1 \leqq i \leqq m$, is a *k-bin* if it contained exactly k items when first an item was assigned to a bin to its right (i.e., when $P_{i+1}[1]$ was assigned). This is in distinction to a *k-item bin*, which is merely a bin P_i, $1 \leqq i \leqq m$, with $|P_i|=k$. The *base level* $b(P_i)$ of a k-bin P_i is defined as $\sum_{j=1}^{k} l(P_i[j])$, whereas the *final level*, or simply *level*, is just $l(P_i)$. If P_i is a k-bin, we call the items $P_i[j]$, $1 \leqq j \leqq k$, *regular items*, and all $P_i[j]$, $j>k$, are called *fallback items*. A *fallback k-bin* is a k-bin P_i such that $|P_i|>k$, that is, one that contains fallback items. A *regular k-bin* is one which contains no fallback items. (Observe that none of these definitions applies to bin P_{m+1}, a bin which, being last, is atypical and will not enter very strongly into our arguments.)

The final lemma of this section gives a list of properties that follow from these definitions and the manner in which FFD operates.

LEMMA 4.7. *In* $\mathscr{P}=\langle P_1, P_2, \cdots, P_{m+1}\rangle$,

(a) *If* $P_i[j]$ *is a regular item, then it precedes in* $\mathscr{T}$ *all* $P_{i'}[j']$ *with* $i<i' \leqq m+1$ *and* $1 \leqq j' \leqq |P_{i'}|$, *and all* $P_i[j']$ *with* $j<j' \leqq |P_i|$.

(b) *If* $1 \leqq k<k'$, *all k-bins are to the left of all k'-bins.*

(c) *If* $\{P_i: s \leqq i \leqq t\}$ *is the set of k-bins, then* $b(P_s) \geqq b(P_{s+1}) \geqq \cdots \geqq b(P_t) > (k/(k+1)) \cdot p$.

(d) *For each* $k>1$, *all regular k-bins are to the left of all fallback k-bins.*

Proof. Parts (a), (b), and (c) are all straightforward consequences of the fact that $\mathscr{T}$ is ordered by decreasing item size. We derive (d) from (b) and (c) by contradiction. Suppose P_i is a fallback k-bin, P_j is a regular k-bin, and $j>i$. Then,

since T_n, the smallest item, went in a bin to the right of P_j, we have $b(P_j)+l(T_n)=l(P_j)+l(T_n)>p$. But we also have $l(P_i[k+1])\geqq l(T_n)$ and, by (c), $b(P_i)\geqq b(P_j)$. Thus $l(P_i)\geqq b(P_i)+l(P_i[k+1])\geqq b(P_j)+l(T_n)>p$, a contradiction. □

We conclude from Lemma 4.1, Lemma 4.4, and Lemma 4.7, that if $\mathscr{P}$ is the FFD packing for a minimal (p/q)-counterexample, it consists of a (possibly vacuous) sequence of fallback 1-bins, followed by a (possibly vacuous) sequence of regular 2-bins, followed by a (possibly vacuous) sequence of fallback 2-bins, followed by a (possibly vacuous) sequence of regular 3-bins, and so on, with the last nonempty bin P_{m+1} containing the single item T_n, which is the last and hence smallest item on $\mathscr{T}$.

In § 5 we expand on this picture, using information derived from the particular values of $p=122$ and $q=100$ to derive a general bound $r_m\leqq 1.220$. More specific arguments, using values of m as well as those of p and q, are required when proving the exact upper bounds of Table 1.

5. The general upper bound.

THEOREM 5.1. $R_M\leqq 1.22$ *for all* $m\geqq 2$.

Proof. Suppose the theorem is false. Then there exists a minimal (122/100)-counterexample, provided say by $\mathscr{T}=\{T_1, T_2, \cdots, T_n\}$ and m. We first derive some constraints on the sizes of the items in $\mathscr{T}$.

By Lemma 4.5, we know that for some $\Delta>0$,

$$l(T_n)=22+\Delta. \tag{5.1}$$

Since T_n is the smallest item, this means that all items are at least this large. Furthermore, by Lemma 4.6 we can conclude that

$$l(T_i)\leqq 56-2\Delta, \qquad 1\leqq i\leqq n. \tag{5.2}$$

Our final observation gives us an upper bound on Δ. Suppose $\Delta>4$. Then every item must have size exceeding 26, and so no bin in $\mathscr{P}^*$, with its capacity of 100, can contain more than 3 items. By Lemma 4.3, this means that all bins in $\mathscr{P}^*$ contain *exactly* 3 items, and $n=3m$. On the other hand, if a bin P_i in $\mathscr{P}$, $1\leqq i\leqq m$, contained two or fewer items, we would have $l(P_i)+l(T_n)\leqq 2(56-2\Delta)+(22+\Delta)=134-3\Delta<122$, which would violate the FFD packing rule. Thus every one of the first m bins of $\mathscr{P}$ contains at least 3 items and so $n\geqq 3m+1$, a contradiction. We thus conclude that

$$0<\Delta\leqq 4. \tag{5.3}$$

The remainder of the proof consists of a weighting argument in which each item is assigned a "weight" based on its size and where it was placed in the FFD packing $\mathscr{P}$. This weighting will have the property that each P_i, $1\leqq i\leqq m$, will contain items whose total weight is *at least* $100-\Delta$. In addition, except for a very limited number of bins, in the optimal packing $\mathscr{P}^*$ each bin will contain weight *no more* than $100-\Delta$. A conservation-of-total-weight argument will then allow us to contradict the assumption that we had a counterexample.

We group the items of $\mathscr{T}$ into seven classes based on the structure of $\mathscr{P}$ as it was described at the end of the previous section. Since all items have size less than 56 by (5.2), there are no fallback 1-bins in $\mathscr{P}$. Since all items exceed 22 in size by

(5.1), there are no bins in $\mathcal{P}$ which contain more than 5 items. We classify items as to which of the remaining possible bin types contain them, and as to size. The observations made concerning item sizes will follow from the fact that $\mathcal{P}$ is an FFD packing, and $l(P_i) > 122 - l(T_n) = 100 - \Delta$ for all i, $1 \leqq i \leqq m$.

The two items in each regular 2-bin, except the last (rightmost) such bin, both exceed (100-Δ)/2 in size, and are *type*-X_2. If both items in the last regular 2-bin exceed (100-Δ)/2 in size, they are also *type*-X_2; otherwise they are both *type*-Z.

The two regular items in each fallback 2-bin must total at least 2(122)/3 in size, by Lemma 4.7(c), and are *type*-Y_2.

The three items in each regular 3-bin, except possibly the last one, all exceed (100-Δ)/3 in size and are *type*-X_3. If all three items in the last regular 3-bin exceed (100-Δ)/3 in size, they are also *type*-X_3; otherwise all three are *type*-Z.

The three regular items in each fallback 3-bin must total at least 3(122)/4 in size, and are *type*-Y_3.

The four items in each regular 4-bin, except possibly the last one, all exceed (100-Δ)/4 in size and are *type*-X_4. If all four items in the last regular 4-bin exceed (100-Δ)/4 in size, they are also *type*-X_4; otherwise all four are again *type*-Z.

The remaining items are all of size at least $22+\Delta$ and are *type*-X_5. These consist of T_n, all the fallback items in fallback 2- and 3-bins, and all the items in 5-item bins.

Now we are prepared to define our weighting function. The weight of an item depends on Δ, the item's type, and the item's size. These dependencies are described in Table 4, where l denotes the size of the item.

TABLE 4
Item weights $w(T_i)$

Item type	$0 < \Delta \leqq 12/5$	$12/5 < \Delta \leqq 4$
X_2	$50 - \frac{\Delta}{2}$	$50 - \frac{\Delta}{2}$
Y_2	$l - \Delta$	$l - \Delta$
X_3	$\frac{100}{3} - \frac{\Delta}{3}$	$\frac{100}{3} - \frac{\Delta}{3}$
Y_3	$l - \Delta$	$l - \Delta$
X_4	$25 - \Delta/4$	$25 - \Delta/4$
X_5	22	$25 - \Delta/4$
Z	l	l

Observe from the table that

$$w(T_i) \leqq l(T_i), \quad \text{for all } T_i \in \mathcal{T}. \tag{5.4}$$

Moreover, for any fixed range of Δ, the weight of a *type*-X_{i+1} item never exceeds the weight of a *type*-X_i item, $2 \leqq i \leqq 4$, and all items have weight at least 22.

For any set S of items, let $w(S)=\sum_{T_i\in S} w(T_i)$. We first show that $w(P_i)\geqq 100-\Delta$, $1\leqq i\leqq m$. Clearly this holds for bins with two *type-X_2* items, three *type-X_3* items, or four *type-X_4* items, and for all 5 item bins. Also, the (at most) three bins composed solely of *type-Z* items have this property since the sum of the item sizes in such a bin, as with all the other P_i, $1\leqq i\leqq m$, must exceed 100-Δ, and each *type-Z* item has weight equal to its size.

This leaves just the fallback 2-bins and 3-bins to be accounted for. A fallback 3-bin contains three *type-Y_3* items and one *type-X_5* item, since the three *type-Y_3* items total at least 3(122)/4 in size. Thus the total weight of such a bin is at least

$$\frac{3(122)}{4}-3\Delta+22=113.5-3\Delta>100-\Delta,$$

since $\Delta\leqq 4$ by (5.3).

A fallback 2-bin contains two *type-Y_2* items whose total size is at least 2(122)/3 and hence one *type-X_5* item. If $0<\Delta\leqq 12/5$ the total weight of such a bin is at least

$$\frac{2(122)}{3}-2\Delta+22=\frac{310}{3}-2\Delta>100-\Delta.$$

On the other hand, if $12/5<\Delta\leqq 4$, the total weight is at least

$$\frac{2(122)}{3}-2\Delta+25-\frac{\Delta}{4}=\frac{319}{3}-\frac{9}{4}\Delta>100-\Delta.$$

Thus $w(P_i)\geqq 100-\Delta$, $1\leqq i\leqq m$, and in fact we can conclude that

$$w(\mathcal{T})\geqq(100-\Delta)m+w(T_n). \tag{5.5}$$

Next we consider the optimal packing $\mathcal{P}^*$. We first claim that no optimal bin containing a *type-Y_2* or *type-Y_3* item can exceed 100-Δ in total weight. This is clear since the weight of such an element is Δ less than its size, by (5.4) no other item weighs *more* than its size, and by definition $l(P_i^*)\leqq 100$, $1\leqq i\leqq m$.

Next we consider the possible optimal bins that contain only *type-X_2*, *-X_3*, *-X_4*, and *-X_5* items. Recall that the sizes of such items are at least $(100-\Delta)/2$, $(100-\Delta)/3$, $(100-\Delta)/4$, and $22+\Delta$, respectively. Clearly no optimal bin can contain more than four of them and, by Lemma 4.3, each optimal bin must contain at least three items. Table 5 presents a partial enumeration of the conceivable configurations, for each stating whether it is permitted by the size constraints, and, if so, the maximum possible weight for such a bin. A configuration is not listed if its total size clearly exceeds that for some listed configuration which is "impossible," due to the size constraints, or if its total weight is trivially dominated by that for some permissible configuration already listed. In the table, a "—" in a configuration stands for *any type-X_i* item, $2\leqq i\leqq 5$.

In no case does the maximum total weight for a permissible configuration exceed $100-\Delta$. Thus any optimal bin whose total weight exceeds $100-\Delta$ must contain at least one of the (at most) 9 *type-Z* items. By (5.4) the total weight of any such optimal bin is at most 100, giving an "excess" weight of at most Δ. Since there are at most 9 such bins, we thus have

$$w(\mathcal{T})\leqq(100-\Delta)m+9\Delta. \tag{5.6}$$

TABLE 5
Upper bounds on $w(P_i^*)$ for bins with all type-X_i items

Configuration	$0<\Delta\leqq 12/5$	$12/5<\Delta\leqq 4$
X_2X_3—	Impossible	Impossible
$X_2X_4X_4$	$100-\Delta$	$100-\Delta$
$X_3X_3X_3$	$100-\Delta$	$100-\Delta$
X_2———	Impossible	Impossible
X_3X_4——	Impossible	Impossible
$X_3X_5X_5X_5$	$100-\frac{\Delta+2}{3}(0<\Delta\leqq 1/4)$ Impossible $(1/4<\Delta\leqq 12/5)$	Impossible
$X_4X_4X_4X_4$	$100-\Delta$	$100-\Delta$

Combining (5.5) and (5.6), we obtain

$$w(T_n)\leqq 9\Delta. \tag{5.7}$$

For $0<\Delta\leqq 12/5$, (5.7) implies that

$$w(T_n)\leqq 9\Delta\leqq 108/5<22,$$

a contradiction. When $12/5<\Delta\leqq 4$, there can be only 5 *type-Z* items, as the four items in the last regular 4-bin must all exceed $22+\Delta\geqq 25-\Delta/4$ and hence are *type-X_4* rather than *type-Z*. Thus in this case we must have

$$w(T_n)\leqq 5\Delta\leqq 20<22,$$

again a contradiction.

Having obtained contradictions in all cases, it follows that a minimal (122/100)-counterexample cannot exist, proving the theorem. □

The reader may have noted that there is a certain amount of slack left in the arguments of this proof, suggesting that better bounds could be proved using much the same methods. This is indeed the case, but all that seems possible is a very slight lowering of the bound—not enough to reduce it to 1.21, for instance. Since we conjecture that the right answer is $20/17=1.176\cdots$, we have settled for the 1.22 bound, judging that the additional effort and complication that might be introduced by an attempt to obtain such a slight improvement would not be justified.

REFERENCES

[1] K. R. BAKER, *Introduction to Sequencing and Scheduling*, John Wiley, New York, 1974.
[2] R. L. GRAHAM, *Bounds on multiprocessing timing anomalies*, SIAM J. Appl. Math., 17 (1969), pp. 416–429.

[3] ———, *Bounds on the performance of scheduling algorithms, Computer and Job/Shop Scheduling Theory*, E. G. Coffman, ed., John Wiley, New York, 1976, Chap. 5.

[4] D. S. JOHNSON, *Fast allocation algorithms*, Proceedings, 13th Annual IEEE Symposium on Switching and Automata Theory, IEEE, New York, 1972, pp. 144–154.

[5] ———, *Near-optimal bin-packing algorithms*, Ph.D. thesis, Mathematics Dept., Mass. Inst. of Tech., Cambridge, 1974.

[6] 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*, this Journal, 3(1974), pp. 299–326.

[7] S. SAHNI, *Algorithms for scheduling independent tasks*, J. Assoc. Comput. Mach., 23 (1976), pp. 116–127.

[8] J. D. ULLMAN, *Complexity of sequencing problems*, Computer and Job/Shop Scheduling Theory, E. G. Coffman, ed., John Wiley, New York, 1976, Chap. 4.