Using Dual Approximation Algorithms for Scheduling Problems: Theoretical and Practical Results

DORIT S. HOCHBAUM

University of California, Berkeley, California

AND

DAVID B. SHMOYS

Massachusetts Institute of Technology, Cambridge, Massachusetts

Abstract. The problem of scheduling a set of n jobs on m identical machines so as to minimize the makespan time is perhaps the most well-studied problem in the theory of approximation algorithms for NP-hard optimization problems. In this paper the strongest possible type of result for this problem, a polynomial approximation scheme, is presented. More precisely, for each ϵ, an algorithm that runs in time $O((n/\epsilon)^{1/\epsilon^2})$ and has relative error at most ϵ is given. In addition, more practical algorithms for $\epsilon = 1/5 + 2^{-k}$ and $\epsilon = 1/6 + 2^{-k}$, which have running times $O(n(k + \log n))$ and $O(n(km^4 + \log n))$ are presented. The techniques of analysis used in proving these results are extremely simple, especially in comparison with the baroque weighting techniques used previously.

The scheme is based on a new approach to constructing approximation algorithms, which is called dual approximation algorithms, where the aim is to find superoptimal, but infeasible, solutions, and the performance is measured by the degree of infeasibility allowed. This notion should find wide applicability in its own right and should be considered for any optimization problem where traditional approximation algorithms have been particularly elusive.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures*

General Terms: Theory, Verification

Additional Key Words and Phrases: Approximation algorithms, combinatorial optimization, heuristics, scheduling theory, worst-case analysis

1. *Introduction*

The problem of minimizing the makespan of the schedule for a set of jobs is one of the most well-studied in scheduling theory. For this problem, we are given a set of n jobs with designated integral processing times p_j to be scheduled on m identical machines. A schedule of jobs is an assignment of the jobs to the machines, so that each machine is scheduled for a certain total time, and the maximum time that

The work of D. S. Hochbaum was supported in part by the National Science Foundation under grant ECS 85-01988 and the work of D. B. Shmoys was supported in part by the National Science Foundation under grant DCR 83-02385.

Authors' address: D. S. Hochbaum, University of California, Berkeley, CA 94720; D. B. Shmoys, Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 62139.

any machine is scheduled for is called the *makespan* of the schedule. In the minimum makespan problem, the objective is to find a schedule that minimizes the makespan; this optimum value is denoted $OPT_{MM}(I, m)$, where I denotes the set of processing times, and m is the specified number of machines.

The minimum makespan problem is NP-complete, so that it is extremely unlikely that there exist efficient algorithms to find a schedule with makespan $OPT_{MM}(I, m)$. As a result, it is natural to consider algorithms that are *guaranteed* to produce solutions that are close to the optimum. Polynomial-time algorithms that always produce solutions of objective value at most $(1 + \epsilon)$ times the optimal value are often called *ϵ-approximation algorithms.* A family of algorithms $\{A_\epsilon\}$, such that for each $\epsilon > 0$ the algorithm A_ϵ is an ϵ-approximation algorithm, is referred to either as a *polynomial approximation scheme* or an *ϵ-approximation scheme.* We shall present the first such scheme for the minimum makespan problem.

The first work done in analyzing algorithms to show that they have provably good performance was for the minimum makespan problem. Perhaps the most natural class of algorithms for the minimum makespan problem is the class of *list processing* algorithms. In this approach, the jobs are given in a list, in a specified order, and the next job on the list is scheduled on the next machine to become idle. In 1966, Graham showed that any such algorithm always delivers a schedule that has makespan at most $(2 - 1/m)OPT_{MM}(I, m)$ [6]. Three years later Graham showed that, if the next job in the list to be scheduled is the one with the Longest Processing Time, the so-called LPT rule, then the schedule produced has makespan at most $(4/3 - 1/3m)OPT_{MM}(I, m)$ [7].

A problem that is closely related to the minimum makespan problem is the bin-packing problem. In this problem, the input consists of n pieces of size p_j, where each size is in the interval [0, 1]. The objective is to pack the pieces into bins where the sum of the sizes of the pieces packed in any bin cannot exceed 1, in such a way that the minimum number of bins is used. This minimum shall be denoted $OPT_{BP}(I)$.

Coffman et al. [1] exploited the relationship between these two problems in designing their MULTIFIT algorithm for the minimum makespan problem. They proved that this algorithm always delivered a schedule with makespan at most $1.22OPT_{MM}(I, m)$. Friesen later improved this bound to $1.20OPT_{MM}(I, m)$, but in the process, the proof became rather complicated in its intricate use of weighting function techniques [4]. This bound was then improved to $(72/61)OPT_{MM}(I, m)$ by Langston [11], who analyzed a modification of the MULTIFIT algorithm, using weighting function techniques as well. To the best of the authors' knowledge, for algorithms that are polynomial in the length of the input, this is the best previously known bound. There have existed, however, polynomial approximation schemes for the minimum makespan problem for any *fixed* value of m, but these have running times that are exponential in m [7, 13].

It is not hard to see that the bin-packing and the minimum makespan problems have essentially the same recognition problem. One would therefore expect that approximation results for one problem would easily translate to the other, by using a simple binary search approach. Since there are polynomial approximation schemes known for the bin-packing problem, this would *seem* to imply that creating a polynomial approximation scheme for the minimum makespan problem should be a trivial task. Unfortunately, it seems to be futile to relate the performance of a bin-packing algorithm to the performance of a corresponding algorithm for the minimum makespan problem; the MULTIFIT algorithm is derived in this way from the FIRST FIT DECREASING bin-packing algorithm, and yet the

MULTIFIT algorithm appears to require a completely new and different analysis to derive a bound that is seemingly unrelated as well.

More important, there is strong complexity-theoretic evidence that any approach that seeks to obtain an approximation algorithm for the minimum makespan problem using an approximation algorithm for the bin-packing problem as a "black box" is doomed to failure. For the bin-packing problem it is possible to create *fully* polynomial approximation schemes (where the running time is polynomial in $1/\epsilon$ as well) by allowing the guarantee to be $(1 + \epsilon)OPT_{BP}(I) + f(1/\epsilon)$, where f is some polynomial function [10]. The minimum makespan problem differs from the bin-packing problem in a crucial way; that is, the job sizes can be rescaled, thus increasing OPT without affecting the essential structure of the problem. The effect of the additive constant can thus be made arbitrarily small, creating a fully polynomial approximation scheme with $f = 0$. For any problem that is strongly NP-complete, the existence of a fully polynomial approximation scheme with $f = 0$ implies that P = NP [5]. Since the minimum makespan problem is strongly NP-complete, it follows that unless P = NP, there cannot exist a fully polynomial approximation scheme with *any* polynomial f.

In this paper we give a polynomial approximation scheme for the minimum makespan problem, where the algorithm that guarantees a relative error of ϵ executes in $O((n/\epsilon)^{1/\epsilon^2})$ time. Although this scheme is not practical, we develop techniques to give refinements of the scheme for $\epsilon = 1/5$ and $\epsilon = 1/6$ that are efficient. In comparison with MULTIFIT, the algorithm given here with an identical performance guarantee is faster, and the proof of the guarantee is rather simple.

The algorithms given here are all based on the notion of a *dual approximation algorithm.* Traditional approximation algorithms seek feasible solutions that are suboptimal, where the performance of the algorithm is measured by the degree of suboptimality allowed. In a dual approximation algorithm, the aim is to find an infeasible solution that is *superoptimal*, where the performance of the algorithm is measured by the degree of infeasibility allowed. In addition to employing dual approximation algorithms for the bin-packing problem, we show a general relationship between traditional (or primal) approximation algorithms, and dual approximation algorithms. We believe that the notion of a dual approximation algorithm is an important one and should be applicable to many other problems.

2. *Primal and Dual Approximation Algorithms*

In this section we explore the relationship between primal and dual approximation algorithms. In particular, we can show that finding an ϵ-approximation algorithm for the minimum makespan problem can be reduced to the problem of finding an ϵ-dual approximation algorithm for the bin-packing problem, which is, in some sense, dual to the minimum makespan problem. In addition, we indicate how these ideas are applicable to other problems.

For the bin-packing problem, an ϵ-dual approximation algorithm is a polynomial-time algorithm that constructs a bin-packing such that at most $OPT_{BP}(I)$ bins are used, and each bin is filled with pieces totaling at most $1 + \epsilon$. There are practical applications where the bin capacity is either not known precisely or simply not rigid, so that such "overflow" is a natural model of this flexibility.

As a historical note, it is perhaps significant to mention that notions similar to dual approximation algorithms have been considered before. Lawler suggests "constraint approximation" in the context of the knapsack problem [12, p. 213]

posing the following hypothetical situation: "A manager seeks to choose projects for a certain period, subject to certain resources constraints (knapsack capacity). The profits associated with items are real and hard. The constraints are soft and flexible. He certainly wants to earn [the optimal amount], if possible." Furthermore, Friesen measured the performance of several bin-packing algorithms when used for bins of size α as a ratio of the number of bins used for this size and the optimal number of bins needed when the bins have capacity 1 [3]. This approach seems similar to our own, but, in its willingness to abandon the constraint of super-optimality, most of the power of dual approximation algorithms, both as a simple tool for traditional approximation algorithms and for direct practical application, is lost.

Let $dual_\epsilon(I)$ denote an ϵ-dual approximation algorithm for the bin-packing problem. Furthermore, let $DUAL_\epsilon(I)$ denote the number of bins actually used by this algorithm. If I denotes a bin-packing instance with piece sizes $\{p_j\}$, it will be convenient to let I/d denote the instance with corresponding piece sizes scaled by d, $\{p_j/d\}$.

It is not hard to see that the optima of the bin-packing and minimum makespan problems are related in that $OPT_{BP}(I/d) \leq m$ if and only if $OPT_{MM}(I, m) \leq d$. In other words, the minimum makespan problem can be viewed as finding the minimum deadline d^* so that $OPT_{BP}(I/d^*) \leq m$. Thus, if we had a procedure for optimally solving the bin-packing problem, we could use it within a binary search procedure to obtain an optimal solution for the minimum makespan problem. A natural extension of this would be to obtain a (traditional) approximation algorithm for the minimum makespan problem by using a (traditional) approximation algorithm for the bin-packing problem within the binary search. As was discussed in the introduction, it is unlikely that such an approach can succeed. Instead, we show that the dual approximation algorithm for the bin-packing problem is precisely the tool required.

A useful measure of the size of an instance is $SIZE(I, m) = \max\{\sum_{j=1}^{n} p_j/m, \max_j p_j\}$. Since any schedule must process each job, $OPT_{MM}(I, m)$ is at least $\max_j p_j$. The average time scheduled on a processor is $\sum p_j/m$. Since some processor must achieve the average, $OPT_{MM}(I, m)$ is at least $SIZE(I, m)$. By another straightforward argument, it can be shown that the makespan of any list processing schedule is at most $2SIZE(I, m)$ [6]. These bounds serve to initialize the binary search given below.

```
procedure ε-makespan(I, m)
  begin
    upper := 2SIZE(I, m)
    lower := SIZE(I, m)
    repeat until upper = lower
      begin
        d := (upper + lower)/2
        call dual_ε(I/d)
        if DUAL_ε(I/d) > m then lower := d
          else upper := d
      end
    output d* := upper
    call dual(I/d*)
  end
```

This procedure is given with an infinite loop, and later we remove this simplifying assumption. Since $DUAL_\epsilon(I)$ is at most $OPT_{BP}(I)$, and since any list processing schedule has a makespan of at most $2SIZE(I, m)$, it follows that

$DUAL_\epsilon(I/upper) \leq m$ initially. Furthermore, by the way *upper* is updated, this remains true throughout the execution of the procedure.

Next we show that $OPT_{MM}(I, m) \geq lower$ throughout the execution of the program. Since $lower = SIZE(I, m)$ initially, the claim is certainly true before the beginning of the **repeat** loop. Furthermore, any time that *lower* is updated to d, it follows that $OPT_{BP}(I/d) \geq DUAL_\epsilon(I/d) > m$, and, therefore $OPT_{MM}(I, m) > d$. The makespan of the schedule produced is at most $(1 + \epsilon)d^*$. In this infinite version, $upper = lower$ at "termination," and therefore, the makespan is at most $(1 + \epsilon)lower \leq (1 + \epsilon)OPT_{MM}(I, m)$. In words, the algorithm is an ϵ-approximation algorithm for the minimum makespan problem, which is what we claimed.

By the fact that all of the processing times are integer we know that $\lceil lower \rceil$ is also a valid lower bound for $OPT_{MM}(I)$. As a result, when $upper - lower < 1$, the binary search can be terminated. (The procedure *dual* should be called once more, with the pieces rescaled by $\lceil lower \rceil$. If this succeeds in using at most m bins, the schedule produced should be output. Otherwise, $\lceil lower \rceil + 1$ is a lower bound on the optimum makespan, so that the schedule produced by $d = upper$, which is less than this bound, can be used instead.) This implies that the algorithm is polynomial in the binary encoding of the input. However, a more practical version of this result is obtained by considering the number of iterations of the binary search that were executed.

THEOREM 1. *If* **procedure** ϵ-*makespan*(I, m) *is executed with* k *iterations of the binary search, the resulting solution has makespan at most* $(1 + \epsilon)(1 + 2^{-k})OPT_{MM}(I, m)$.

PROOF. To prove this more precise claim, one need only note that after k iterations $upper - lower = 2^{-k}SIZE(I, m) \leq 2^{-k}OPT_{MM}(I, m)$. Since the schedule produced has length at most

$$\begin{aligned}(1 + \epsilon)upper &= (1 + \epsilon)(upper - lower + lower) \\ &\leq (1 + \epsilon)(2^{-k}OPT_{MM}(I, m) + OPT_{MM}(I, m)),\end{aligned}$$

we get the desired result. □

Notice that since the end goal of this approach is an ϵ-approximation scheme, we could equally well create an ϵ-approximation algorithm for the minimum makespan problem by using an $\epsilon/2$-dual approximation algorithm for the bin-packing problem, and then only $O(\log(1/\epsilon))$ iterations are required to get a total relative error of ϵ. Thus, the algorithm is a strongly polynomial one, in that we do not need to consider the lengths of the binary encoding of the given processing times.

In the remainder of this section, we demonstrate that the techniques used above can be applied to problems other than minimum makespan problem. In order to help motivate this generalization, we refer frequently to the original application, but in a slightly different form. We view the bin-packing problem instance as specified by integral piece sizes $\{p_j\}$ and an additional parameter d, the capacity of the bins. It is clear that this formulation is equivalent, since it amounts to no more than rescaling both the piece and bin sizes.

Consider the recognition or decision version of an optimization problem where there are two critical parameters—as before we had the capacity and the number of bins or machines allowed. There are *two* optimization problems that can be derived from the decision problem by fixing one of the two parameters and then, subject to this constraint, optimizing the other. We call one problem the *dual* of

the other. Let such a *two-parameter recognition problem* be denoted $\mathbf{R}(p_1, p_2)$. The primal problem shall be the one where p_1 is given as part of the input and p_2 is, say, minimized — an instance is specified by an ordered pair $(I, \bar{p}_1)$. Similarly, an instance of the dual problem consists of a pair $(I, \bar{p}_2)$ and the first parameter is minimized. Let $OPT_P(I, \bar{p}_1)$ and $OPT_D(I, \bar{p}_2)$ denote the optimal values of the specified primal and dual problems, respectively. An ϵ-(primal) approximation algorithm for the primal problem is an algorithm that delivers a solution where the value of the first parameter is at most $\bar{p}_1$ and the value of the second parameter, as derived by the algorithm and denoted $PRIMAL_{P,\epsilon}(I, \bar{p}_1)$, is at most $(1 + \epsilon)OPT_P(I\ \bar{p}_1)$. (A completely analogous statement could be made for the dual problem.) An ϵ-dual approximation algorithm $dual_{D,\epsilon}(I, \bar{p}_2)$ for the dual problem is an algorithm that delivers a solution where the value of the first parameter, as derived by the algorithm and denoted $DUAL_{D,\epsilon}(I, \bar{p}_2)$, is at most $OPT_D(I, \bar{p}_2)$ and the value of the second parameter is at most $(1 + \epsilon)\bar{p}_2$. (Again, a similar situation applies to the primal problem.) We claim that finding ϵ-(primal) approximation algorithm for the primal problem always can be reduced to finding an ϵ-dual approximation algorithm for the dual. The following algorithm is nearly identical to the one discussed above.

```
problem ε-primal(I, p̄1)
  begin
    upper := trivial upper bound
    lower := trivial lower bound
    repeat until (upper − lower) < precision bound
      begin
        p := (upper + lower)/2
        call dual_{D,ε}(I, p)
        if DUAL_{D,ε}(I, p) > p̄1 then lower := p
          else upper := p
      end
    output bound := upper
    {Note that at the end of the binary search, by a precision argument, upper is a lower
    bound as well.}
    call dual_{D,ε}(I, bound)
  end
```

Using arguments identical to the particular application given before, it is straightforward to see that this procedure is an ϵ-approximation algorithm for the primal problem. Simply put, a dual approximation algorithm for the dual problem can be converted into a primal approximation algorithm for the primal problem.

3. *A Polynomial Dual Approximation Scheme for Bin Packing*

In this section, we give a polynomial ϵ-dual approximation scheme for the bin-packing problem. By applying Theorem 1 we obtain a polynomial ϵ-approximation scheme for the minimum makespan problem. The spirit of this scheme is based on a generalization of the ideas used in [8] and has a flavor similar to the one given in [2] for a (primal) ϵ-approximation scheme for the bin-packing problem. Furthermore, techniques similar to the ones employed in this section date back at least to Ibarra and Kim [9], but it is only in recent work that the full power of these techniques has been understood.

The result is presented in two parts: First we argue that the problem of finding an ϵ-dual approximation algorithm can be reduced to finding an ϵ-dual approximation algorithm for the restricted class of instances where all piece sizes are

greater than ϵ, and then we give a polynomial scheme for this restricted class of instances.

Suppose that we had an ϵ-dual approximation algorithm for the bin-packing problem which worked only on instances where all piece sizes are greater than ϵ. Such an algorithm could be applied to an arbitrary instance I in the following way.

Step 1. Use the assumed algorithm to pack all of the pieces with size $>\epsilon$.

Step 2. For each remaining piece of size $\leq\epsilon$, pack it in any bin that currently contains ≤ 1. If no such bin exists, start a new bin.

First of all, it is easy to see that this procedure never packs any bin with more than $1 + \epsilon$. Since the algorithm used in Step 1 is a dual approximation algorithm and since the minimum number of bins for a subset of I is at most $OPT_{BP}(I)$, it follows that at most $OPT_{BP}(I)$ bins are used in Step 1. Thus, if no new bins are used in Step 2, the algorithm presented is an ϵ-dual approximation algorithm. Suppose now that a new bin was used in Step 2. This implies that the last piece packed could not fit in any started bin. Since the size of the piece is $\leq\epsilon$ and the extended capacity of the bin is $1 + \epsilon$, it follows that every bin is filled to at least capacity 1! In other words, in this case it follows that the number of bins used is at most $\lceil \sum p_j \rceil$ and it is clear that $\lceil \sum p_j \rceil \leq OPT_{BP}(I)$.

By the reduction given above, we need only produce an ϵ-dual approximation algorithm for instances where all pieces are greater than ϵ.

Partition the interval of allowed pieces sizes $(\epsilon, 1]$ into $s = \lceil 1/\epsilon^2 \rceil$ equal length subintervals $(\epsilon = l_1, l_2], (l_2, l_3], \ldots, (l_s, l_{s+1} = 1]$. For a given instance I, let b_i denote the number pieces with size in the interval $(l_i, l_{i+1}]$.

Consider any feasibly packed bin. Since each piece is of size greater than ϵ, there are at most $\lfloor 1/\epsilon \rfloor$ pieces in this bin. Let x_i denote the number of pieces with size in the interval $(l_i, l_{i+1}]$. Each x_i can assume one of at most $\lceil 1/\epsilon \rceil$ values in the interval $[0, 1/\epsilon)$, thus the *configuration* of the bin can be given by an s-truple $(x_1, x_2, \ldots, x_s)$. A configuration is said to be *feasible* if $\sum_{i=1}^{s} x_i l_i \leq 1$. It is easy to see that any bin that is packed feasibly (with total capacity used at most one) has a corresponding configuration that is feasible. A simple calculation shows that t, the number of feasible configurations is at most $\lceil 1/\epsilon \rceil^s$, which is a (rather large) constant for fixed ϵ.

Consider any bin B that is packed according to some feasible configuration $(x_1, \ldots, x_s)$. It is straightforward to see that

$$\sum_{j\in B} p_j \leq \sum_{i=1}^{s} x_i l_{i+1} \leq \sum_{i=1}^{s} x_i(l_i + \epsilon^2) = \sum_{i=1}^{s} x_i l_i + \epsilon^2 \sum_{i=1}^{s} x_i \leq 1 + \epsilon^2 \left(\frac{1}{\epsilon}\right) = 1 + \epsilon.$$

(Note that the last inequality follows from the fact that $\sum_{i=1}^{s} x_i$ is the number of pieces packed in B, which is at most $1/\epsilon$.) In other words, if the pieces are packed according to a feasible configuration, then the overflow in any bin is at most ϵ. Therefore, if we find a partition of the pieces into feasible configurations that has the minimum number of parts, this would yield an ϵ-approximation algorithm. From a slightly different angle, this is nothing more than the bin-packing problem, where the piece sizes are restricted to be one of the s lower bounds l_i. Fortunately, this restricted bin-packing problem can be solved in polynomial time.

Although it is well known that bin-packing with a fixed number of piece sizes can be solved in polynomial time, we give a dynamic programming algorithm here, for completeness. Let $\text{Bins}(b_1, \ldots, b_s)$ denote the minimum number of bins needed when there are b_i pieces of size l_i. Then consider the pieces in the first bin to be

packed. They must correspond to a feasible configuration, so that

$$\text{Bins}(b_1, \ldots, b_s) = 1 + \min_{\substack{\text{feasible} \\ \text{configurations} \\ (x_1, \ldots, x_s)}} \text{Bins}(b_1 - x_1, \ldots, b_s - x_s).$$

Thus, in building the dynamic programming table, there are n^s entries, each of which requires at most $t \leq \lceil 1/\epsilon \rceil^s$ time to compute. As a result, the overall running time is $O(t \cdot n^s) = O((n/\epsilon)^{\lceil 1/\epsilon^2 \rceil})$. Since for every fixed $\epsilon > 0$ this is polynomial, it follows that the family of algorithms forms a polynomial approximation scheme. It may also be useful to note that the $O(n^s)$ space requirements could be eliminated at the cost of performing the enumeration explicitly and thereby increasing the running time to $O(n^{(1/\epsilon)^{1/\epsilon^2}})$.

4. *A 1/5 Dual Approximation Algorithm for Bin Packing*

In this section we show how the central ideas of the general ϵ-approximation scheme can be refined to give a $(1/5 + 2^{-k})$-approximation algorithm for the minimum makespan problem that runs in $O(n(k + \log n))$ time. This performance guarantee is equivalent to the MULTIFIT algorithm, originally analyzed in [1]. Significantly, the analysis of our algorithm is rather simple, especially when compared to the intricate weighting function techniques used in [4] to prove the 1/5 bound for MULTIFIT. We are able to improve the efficiency of the algorithm for this case by examining the structure of feasible configurations much more closely and thus greatly reducing the number of types of configurations considered.

As was done in the previous section, in order to get the approximation algorithm for the minimum makespan problem, we construct a 1/5-dual approximation algorithm for the bin-packing problem when restricted to instances where all of the piece sizes are greater than 1/5. We use the term *k-bin* to denote a bin that is packed with k pieces. It is convenient to use $\mathbf{L}[u_1, \ldots, u_k]$ to denote the set of k distinct pieces $\{i_1, i_2, \ldots, i_k\}$, where i_l is the largest available piece of size at most u_l, where $u_1 \leq u_2 \cdots \leq u_k$ and $p_{i_1} \leq \cdots \leq p_{i_k}$. Consider the following algorithm. Unlike most other algorithms for bin packing, when a decision is made to pack a set of pieces together, they are placed in the bin and no other pieces will be added to the bin.

Stage 1. While there is a piece j with $p_j \in [0.6, 1]$, pack j with $\mathbf{L}[1 - p_j]$, if such a piece exists. Otherwise pack j by itself.

Stage 2. While there exist 2 pieces i, j with $p_i, p_j \in [0.5, 0.6)$, pack i and j together. {There may exist an odd number of pieces in [0.5, 0.6). For simplicity, we first assume that this is not the case.}

Stage 3. {All remaining pieces are <0.5.} While there exist three such pieces where the largest is at least 0.4, find $\mathbf{L}[0.3, 0.4, 0.5]$ and pack them together.

Stage 4. While there exists a piece with size in [0.4, 0.5), pack the largest two pieces together.

Stage 5. {All remaining pieces are <0.4.} Take the smallest piece j remaining. If $p_j > 0.25$, pack all remaining pieces in 3-bins. Otherwise, $p_j = 0.25 - \delta$ for some $\delta \geq 0$. If three other such pieces exist, pack j with $\mathbf{L}[0.25, + \delta/3, 0.25 + \delta, 0.25 + 3\delta]$. If such pieces do not exist, pack the remaining pieces in 3-bins.

In order to prove that the above algorithm is a 1/5-dual approximation algorithm, we must show two things—no bin is ever filled with more than 6/5 and that the number of bins used is at most $OPT_{BP}(I)$. The first is more straightforward, so we

begin with that. In Stage 1, it is clear that no bin is filled with more than 1. In Stage 2, since any two pieces are each of size less than 3/5, a bin is filled to at most 6/5. In Stage 3, by the choice imposed by the algorithm, the pieces sum to at most $0.3 + 0.4 + 0.5 = 1.2 = 6/5$. In Stage 4, since all remaining pieces sizes are less than 1/2, the two largest sum to less than 1. Finally, in Stage 5, when we add the bounds together, we get $1 + 10\delta/3$. Since all of the piece sizes are more than 1/5, $\delta < 1/4 - 1/5 = 1/20$ and $1 + 10\delta/3 < 7/6 < 6/5$. For a 3-bin packed in Stage 5, the total packed cannot exceed $3 \cdot (0.4) = 1.2$.

In order to prove that the number of bins used is at most $OPT_{BP}(I)$, we show, roughly, that whenever a set of jobs is packed together and deleted from the instance I, we get a new instance I', such that $OPT_{BP}(I') \leq OPT_{BP}(I) - 1$. Before considering the actions of each stage, we give two extremely useful principles that will enable us to carry out this strategy.

COMPRESSION PRINCIPLE. *If I_2 is obtained from I_1 by changing the size of some piece j from p_j to $\bar{p}_j$ where $p_j \geq \bar{p}_j$, then $OPT_{BP}(I_2) \leq OPT_{BP}(I_1)$.*

PROOF. The optimal packing of I_1 remains feasible for I_2, so the optimal packing for I_2 can not use more bins. □

DOMINATION PRINCIPLE. *If $\{i_1, \ldots, i_k\}$ are the only pieces in a bin in some optimal packing of the instance I, and $j_1, \ldots, j_k$ are distinct pieces such that $p_{i_l} \leq p_{j_l}$ for all $l = 1, \ldots, k$, then the instance I' formed by deleting $\{j_1, \ldots, j_k\}$ from I is such that $OPT_{BP}(I') \leq OPT_{BP}(I) - 1$.*

PROOF. (We can assume without loss of generality that, if $p_{i_l} = p_{j_l}$, then in fact, $i_l = j_l$.) We demonstrate that there is a feasible packing for I' using $OPT_{BP}(I) - 1$ bins. Take an optimal packing where $\{i_1, \ldots, i_k\}$ are the only pieces in some bin. Consider the packing of the other $OPT_{BP}(I) - 1$ bins. Let j_l be some piece that is in the packing of these other bins. Replace j_l by i_l. This packing must remain feasible, since $p_{j_l} \geq p_{i_l}$. In fact, by the additional assumption that, if $p_{j_l} = p_{i_l}$ then $j_l = i_l$, it follows that $p_{j_l} > p_{i_l}$. (Otherwise, j_l would have been packed in the $\{i_1, \ldots, i_k\}$ bin and would have been removed once and for all.) As a result, after a finite number of these replacements we get a feasible schedule for I' using $OPT_{BP}(I) - 1$ bins. □

Given the domination principle, it is easy to see that it is important to obtain upper bounds on pieces that can be feasibly packed together. The following lemma gives the required information.

BOUNDING LEMMA. *Consider a bin-packing instance where pieces are at least ϵ. Let piece i be packed in a k-bin in some feasible packing, where $p_i \geq l$. If $i_1, \ldots, i_{k-1}$ are the $k - 1$ jobs packed with i, where $p_{i_1} \leq \cdots \leq p_{i_{k-1}}$, then $p_{i_j} \leq (1 - l - (j - 1)\epsilon)/(k - j)$.*

PROOF. Consider piece i_j. Each of the pieces $i_1, \ldots, i_{j-1}$ is at least ϵ, so that the pieces $i, i_1, \ldots, i_{j-1}$ sum to at least $l + (j - 1)\epsilon$. This leaves at most $1 - (l + (j - 1)\epsilon)$ for pieces $i_j, \ldots, i_{k-1}$. Since i_j is the smallest, it has size at most the average of these pieces, which is at most $(1 - l - (j - 1)\epsilon)/(k - j)$. □

In particular, we have shown the following result.

COROLLARY. *If i is the smallest piece, and there exists a feasible packing where it is packed in a k-bin, then the pieces $i_1, \ldots, i_{k-1}$ that it is packed with have processing times satisfying $p_{i_j} \leq (1 - j \cdot p_i)/(k - j)$.*

We can view the algorithm as producing a series of instances, $I = I_0, I_1, \ldots, I_q = \emptyset$, where I_l consists of the pieces remaining to be packed, after the algorithm has packed l bins. For Stages 1, 2, and 4, we show that when a bin is packed to produce I_l from I_{l-1}, $OPT_{BP}(I_l) \leq OPT_{BP}(I_{l-1}) - 1$. For Stage 3 more careful analysis is required, and we show that either $OPT_{BP}(I_l) \leq OPT_{BP}(I_{l-1}) - 1$ or that another bin is packed by Stage 3, and $OPT_{BP}(I_{l+1}) \leq OPT_{BP}(I_{l-1}) - 2$. These claims imply that at the end of Stage 4, if p bins have been packed, $OPT(I_p) \leq OPT(I) - p$. In the last stage, we introduce another notion, $QUASI\text{-}OPT(I)$ such that $QUASI\text{-}OPT(I) \leq OPT(I)$ for all instances I. We shall show that $QUASI\text{-}OPT(I_{l+1}) \leq QUASI\text{-}OPT(I_l) - 1$ for any instance I_l where Stage 5 is applied. Thus, at the termination of the algorithm,

$$\begin{aligned} OPT(I) &\geq OPT(I_p) + p \geq QUASI\text{-}OPT(I_p) + p \\ &\geq (QUASI\text{-}OPT(I_q) + (q-p)) + p = 0 + q - p + p = q. \end{aligned}$$

In other words, the number of bins packed q is at most $OPT_{BP}(I)$. Thus, all we need to do is to consider each of the stages and verify the claimed inequalities, using the compression and domination principles.

Stage 1. Here we can apply the domination principle. Consider the piece j. Since $p_j \geq 3/5$, and all piece sizes are $>1/5$, j can be packed with at most one other piece. However, we pack j with $\mathbf{L}[1 - p_j]$, the largest piece that j fits with. Thus, the piece sizes packed by the algorithm are at least as big as those in the optimal packing, so that the domination principle applies. (Notice that this includes the case where j is packed by itself in an optimal packing, since then we can view it as being packed with a piece of size zero.)

Stage 2. We can view the action of this stage as follows. If i and j are to be packed together, first compress them both to have size 1/2, and then pack them together. Let I denote the instance initially, let I_1 denote the instance after the compression, and let I_2 denote the instance with i and j deleted. By the compression principle, $OPT_{BP}(I_1) \leq OPT_{BP}(I)$. Therefore, we need only show that $OPT_{BP}(I_2) \leq OPT_{BP}(I_1) - 1$. The following fact suffices to prove this.

FACT. *If $p_i = p_j = 1/2$, then there exists an optimal packing where i and j are packed together.*

PROOF. The proof is by a standard interchange argument. Suppose the claim is false. In any optimal packing, the pieces that i is packed with total at most 1/2, and the same is true for j. Thus we can change the packing so that i and j are packed together, and the two remaining "halves" are packed together, using no more bins than the original packing, which is a contradiction. □

Stage 3. The proof for this stage is the most involved of any of the five. It is important to note that in any feasible packing any piece of size ≥ 0.4 must be packed in a bin with at most three pieces. Furthermore, if we use the bounding lemma with $k = 3$, $l = 0.4$, and $\epsilon = 1/5$, we find that the two smaller pieces in a 3-bin with a piece of size ≥ 0.4 must have sizes at most 0.3 and 0.4, respectively. As a result, if there is any possibility that the largest remaining piece can be packed in a 3-bin, the algorithm will, in fact, pack it in a 3-bin. Suppose that the instance remaining at this stage is denoted I, and let i, j, and k be the three pieces packed together in this stage, where $p_i \leq p_j \leq p_k$. We consider several cases; in each, we assume that the previous cases do not apply.

(1) k is the only piece in I with size in [0.4, 0.5).

In this case, j and k are the two largest pieces. Thus, if k is packed in a 2-bin in an optimal packing, these pieces are clearly dominated by j and k (and i is packed by the algorithm as well!). If k is packed in a 3-bin, the calculations given by the bounding lemma imply that i, j, and k dominate those pieces as well. In either case, deleting i, j, and k ensures that the minimum number of bins decreases.

(2) There exists an optimal packing where k is packed in a 3-bin.

This is also an easy case. By the application of the bounding lemma given above, it is clear that the jobs packed with k in the optimal packing must be dominated by i and j.

(3) There exists a 4-bin in an optimal solution.

Consider the four pieces in such a 4-bin. Since all pieces are greater than 0.2, it follows that the four pieces packed must each be at most 0.4. Furthermore, the smallest two must each be at most 0.3, since otherwise, the three largest must total more than 0.9, and the smallest is more than 0.2. (These are weak upper bounds, but they will suffice.) Since there is another piece l with $p_l \geq 0.4$ (recall that Case (1) does not apply), the existence of the pieces in this 4-bin implies that the algorithm will pack three more pieces in Stage 3, say i', j', and k'. Since k and k' are the two largest pieces in I, we know that they dominate the pieces that are in the 2-bin containing k. (Recall that Case (2) does not apply.) Furthermore, we know that i', i, j', and j must dominate the pieces in the 4-bin. Therefore, by packing the two bins and deleting the six pieces, we know that minimum number of bins must decrease by at least two.

(4) Everything else.

Since (3) does not apply, we see that there is no 4-bin in any optimal solution. If there is also no 3-bin in the optimal solution, then clearly we are done, since any pair of pieces can be packed together, and by packing three pieces in one bin, we can only decrease the total number of bins used. If there is an optimal packing with a 3-bin, then, by a standard interchange argument, there is one where the smallest piece is in a 3-bin. However, since in any 3-bin the "middle" piece must be less than 0.4, we know that i, j, and k must dominate the piece sizes packed in such a 3-bin.

Stage 4. Consider the largest remaining piece i. By the bounding lemma, we know that it cannot be feasibly packed in a 3-bin. (Otherwise, i would be packed in Stage 3.) Thus, by choosing the next largest piece to be packed with it, we are assured that the piece selected dominates the piece packed with i in an optimal packing.

Stage 5. In this final stage, we use slightly more general tools. At this point, we know that any three pieces may be packed together (within the 6/5 bound) and that all pieces will be packed either in 3-bins or 4-bins (with the exception of at most one bin of "leftovers"). Thus, call a packing *quasi-feasible* if for any 4-bin, the capacity used is at most 1, but for any 3-bin the allowed capacity is extended to 6/5. Similarly, a *quasi-optimal* packing is a quasi-feasible packing that uses the minimum number of bins; let this minimum number be denoted $QUASI\text{-}OPT(I)$. It is clear that $QUASI\text{-}OPT(I) \leq OPT_{BP}(I)$. For this stage, we show that if we pack a set of pieces in a bin, then the value of *QUASI-OPT* decreases by at least one. It

is easy to see that analogous versions of the compression and domination principles hold for quasi-optimality.

By a simple interchange argument, it follows that if there is a quasi-optimal packing that uses a 4-bin, then there exists a quasi-optimal packing where the smallest piece is packed in a 4-bin. Furthermore, if the smallest piece is packed (feasibly or quasi-feasibly) in a 4-bin, then it must have size $0.25 - \delta$ for some nonnegative δ. Thus, if the smallest piece has size greater than 0.25, we can conclude that there are no 4-bins in the (quasi-) optimal solution, and thus packing three to a bin is at least as good as is possible.

Therefore, we need only worry about the case where the smallest piece has size $0.25 - \delta$. We apply the corollary to the bounding lemma, with $l = \epsilon = 0.25 - \delta$ and $k = 4$. We see that the three largest pieces in such a bin must be at most $0.25 + \delta/3$, $0.25 + \delta$, and $0.25 + 3\delta$. As remarked above, if there is a 4-bin in the quasi-optimal solution, we can consider one where the smallest piece is packed in a 4-bin and thus, by the bounds given by the bounding lemma, we know that the algorithm will succeed in packing a 4-bin. Furthermore, the 4 pieces selected by the algorithm are guaranteed to dominate the pieces packed together in the quasi-optimal packing. If there is no 4-bin in the solution, since *any* three pieces fit together, it cannot increase the total number of bins used if we succeed in packing a 4-bin.

In order to complete the description of the algorithm (and the accompanying proof that the packing produced uses at most $OPT_{BP}(I)$ bins), we must consider the case in which there are an odd number of pieces to be packed in Stage 2. Consider the single remaining piece i with $p_i \in [0.5, 0.6)$. It must be packed in a bin with at most three pieces. We can simply nondeterministically guess which kind of bin is correct and then continue accordingly. If the guess is that i is packed in a 2-bin, we simply pack i with the largest remaining piece. Domination ensures that, if the guess is correct, the number of bins in the optimal solution decreases by at least one. If the guess is that i is packed in a 3-bin, then pack i with $\mathbf{L}[0.25, 0.3]$. This can be shown to decrease the number of bins in the optimal packing by applying the bounding lemma with $l = 0.5$, $k = 3$, and $\epsilon = 1/5$, and then invoking the domination principle. Of course, since the number of possible guesses is three, we need not consider the algorithm to be nondeterministic: We can simply try all three possibilities and choose the best packing.

Finally, it is not hard to see that this algorithm can be implemented in $O(n)$ time, if the pieces are given in sorted order. This implies that, for each iteration of the binary search, only $O(n)$ time is required. By combining the results from this and previous sections, we have presented an algorithm for the minimum makespan problem that runs in time $O(n(k + \log n))$ and produces a solution with makespan at most $(6/5 + 2^{-k})OPT_{MM}(I, m)$. By comparison, MULTIFIT runs in time $O(n(k \log m + \log n))$ to achieve the same performance.

5. *A 1/6-Dual Approximation Algorithm for Bin Packing*

In this section we present a 1/6-dual approximation algorithm for the bin-packing problem restricted to instances where all piece sizes are greater than 1/6. Using the reductions presented in the earlier sections, this gives us both a 1/6-dual approximation algorithm for the unrestricted bin-packing problem and a 1/6-approximation algorithm for the minimum makespan problem. The techniques used in this section are natural generalizations of those employed in the previous section. Although the algorithm is still not entirely practical, since the running time of the

resulting approximation algorithm for the minimum makespan problem is $O(n(m^4 + \log n))$, this is still a significant improvement over the $O(n^{36})$ algorithm given by the general scheme. This suggests that with further refinements, algorithms with very small error bounds, based on ideas similar to those employed here, can be made practical.

Consider the following algorithm. We first present the algorithm as a nondeterministic algorithm; at certain points, the algorithm will be required to perform a **guess** operation, and for the proof of correctness of the algorithm, we assume that these guesses have been made correctly. For the actual implementation of the algorithm, we execute the algorithm for all possible guesses. In executing some choice for the guesses, it may become apparent that these are inappropriate, and in this case, the next guess is tried. The algorithm outputs the solution that uses the fewest number of bins. This ensures that the deterministic algorithm will use at most as many bins as the nondeterministic one.

procedure 1/6-*dual*(I)

Stage 1. While there exists i such that $p_i \in [2/3, 1]$, pack p_i with $\mathbf{L}[1 - p_i]$.

Stage 2. **Guess** the total number of 1- or 2-bins in an optimal solution of the remaining instance. For each of these bins, pack it with $\mathbf{L}[1/2, 2/3]$, if such pieces exist; otherwise pack with $\mathbf{L}[2/3]$.

{For the remainder of the procedure, we restrict our attention only to packings where each bin contains at least three pieces.}

Stage 3. {All remaining piece sizes are $<2/3$.} For each remaining piece i with size $p_i = 1/2 + \delta$, $\delta \geq 0$, pack i with $\mathbf{L}[1/4 - \delta/2, 1/3 - \delta]$.

Stage 4. {All remaining piece sizes are $<1/2$.} **Guess** the number of 4-bins that contain a piece with size in the range $[5/12, 1/2)$ in an optimal packing. Pack each bin with $\mathbf{L}[7/36, 5/24, 1/4, 1/2]$.

Stage 5. For each remaining piece of size $5/12 + \delta$, $\delta \geq 0$, pack it in a 3-bin with $\mathbf{L}[7/24 - \delta/2, 5/12 - \delta]$.

Stage 6. {All remaining piece sizes are $<5/12$.} **Guess** the number of 3-bins in an optimal packing of the remaining instance. For each of these, pack it with $\mathbf{L}[1/3, 5/12, 5/12]$.

{For the remainder of this procedure we can restrict attention to packings where each bin contains at least four pieces.}

Stage 7. For each piece i with size $1/3 + \delta$, $\delta \geq 0$, pack i with $\mathbf{L}[2/9 - \delta/3, 1/4 - \delta/2, 1/3 - \delta]$.

Stage 8. {All remaining piece sizes are $<1/3$.} **Guess** the number of 5-bins that contain a piece with size in the range $[7/24, 1/3)$ in an optimal packing. Pack each such bin with $\mathbf{L}[17/96, 13/72, 9/48, 5/24, 1/3]$.

Stage 9. Take the largest remaining piece of size $p_i = 7/24 + \delta$, $\delta \geq 0$ and pack i with $\mathbf{L}[17/72 - \delta/3, 13/48 - \delta/2, 7/24 + \delta]$. Repeat this until all piece sizes are $<7/24$.

Stage 10. Consider the smallest piece i. If $p_i > 1/5$, pack the remaining pieces arbitrarily four pieces per bin. If $p_i = 1/5 - \delta$, $\delta \geq 0$, then pack i with $\mathbf{L}[1/5 + \delta/4, 1/5 + 2\delta/3, 1/5 + 3\delta/2, 7/24]$, if such pieces exist. Otherwise, pack the remaining pieces four per bin.

To prove that this is a 1/6-dual approximation algorithm, it is necessary to show that no bin is ever filled with more than 7/6, and that at most $OPT_{BP}(I)$ bins are used. The first part of this is a straightforward exercise in arithmetic. To prove that no more than $OPT_{BP}(I)$ bins are used, we once again rely on the domination principle and the bounding lemma to show that, whenever the algorithm packs a bin, the number of bins in the optimal packing of the remaining instance decreases.

It is very important to note how the guesses are used to refine the structure of the allowed packings. For example, in Stage 2, we guess the total number of 1- and 2-bins in an optimal packing. Therefore, in the remainder of the procedure, when we consider an optimal packing of the remaining packing, we can restrict attention to optimal packings that only use bins with at least three pieces. This guess begins

paying dividends immediately. In Stage 3, we consider pieces of size at least 1/2. Since all pieces are greater than 1/6, such a piece can be packed in bins with at most two other pieces. Therefore, we can conclude that this piece is indeed packed in a 3-bin, and the bounding lemma can be applied immediately.

Initially, we know that each bin is packed with at most five pieces. The stages can be divided in the following way. In Stages 2 and 3, we pack pieces that can be either in 2- or 3-bins; in 4 and 5, we are restricted to 3- and 4-bins. Stage 6 ensures that we can later restrict attention to packings with only 4- and 5-bins. In Stage 7, we pack those pieces known to be in 4-bins. Finally, in the last three stages, we pack those pieces that can be in either 4- or 5-bins. In each case we use a judiciously selected guess to decide how to partition the pieces into k or $k + 1$ bins. Once the guess is made, it is a simple matter to apply the bounding lemma and the domination principle. To check the precise bounds is a straightforward, but tedious exercise.

The only stage that requires a little more work is the final one. This stage is very similar to last stage of the 1/5-dual approximation algorithm. As in that stage, it is convenient to define a notion of *quasi-feasibility*. In this case, we allow the 4-bins to be overpacked to 7/6, while restricting other bins to capacity 1. Using the resulting notion of quasi-optimality, it is not hard to use the domination principle and the bounding lemma to show that the number of bins in the quasi-optimal solution must decrease each time a bin is packed by the algorithm.

In implementing the algorithm, as mentioned above, we must try all possible guesses. One very tempting improvement in the practicality of the algorithm would be to show that some monotonicity property exists, so that binary search could be employed, instead of this explicit search.

When using the dual approximation algorithm within the procedure for the minimum makespan algorithm, it is clear that no guess greater than m need ever be considered. As a result, since the nondeterministic algorithm can be implemented in $O(n)$ time once the pieces are sorted, we get the following result.

THEOREM 2. *The* **procedure** *1/6-dual yields an approximation algorithm for the minimum makespan problem that delivers a solution with makespan at most* $(1/6 + 2^{-k})OPT_{MM}(I, m)$ *and runs in* $O(n(km^4 + \log n))$ *time.*

6. *Conclusions*

In this paper, we have presented several algorithms for the bin-packing and minimum makespan problems. Most important, we have shown that for any $\epsilon > 0$, there exists an efficient, that is, polynomial-time, algorithm that delivers an approximately optimal schedule for the minimum makespan problem that is guaranteed to have relative error at most ϵ. In particular, we have shown how the framework of the scheme can be used to produce reasonably practical algorithms for $\epsilon = 1/5$ and 1/6. As was noted above, since the minimum makespan problem is strongly NP-complete, we cannot hope to improve the scheme significantly, in that the existence of a fully polynomial scheme would imply that P = NP.

The key technique used in all of the algorithms presented here is that of a dual approximation algorithm. This notion is of fundamental importance, in addition to applications in the construction of primal approximation algorithms. For real-world problems, constraints are more often approximations to the real constraints, than restrictions that are rigid and inflexible. We have shown that for the bin-packing problem, the design and analysis of effective dual approximation

algorithms is significantly less difficult and tedious than the best known practical methods for primal bin-packing approximation algorithms. Furthermore, we presented a general framework for using dual approximation algorithms within traditional approximation algorithms for closely related problems. It may well turn out for other problems, especially those where researchers have been stymied in the quest for good primal approximation algorithms, that the dual approach is the way to proceed.

Appendix A

In this appendix we provide the computations needed to prove the performance guarantee for 1/6-*dual*(I). It will be convenient to let $OPT_{BP}(I, k)$ denote the optimal number of bins used when the constraint "all bins contain at least k pieces" is added to the usual bin-packing problem. It is important to note that the following generalization of the domination principle can be proved by the same argument used to prove the simpler form. We say that a set of pieces *dominates* another, if there is a 1–1 correspondence between the elements of the two sets so that each piece of the first set is at least as large as the corresponding piece of the second.

Generalized Domination Principle. Let $\{i_1, \ldots, i_k\}$ be the only pieces packed in some l bins of a feasible packing of the instance I, where excluding these bins, the packing contains n_r bins with r pieces. If $\{j_1, \ldots, j_k\}$ is a set of distinct pieces that dominate the set $\{i_1, \ldots, i_k\}$, then the instance I' formed by deleting $\{j_1, \ldots, j_k\}$ from I has a feasible packing such that n_k k-bins are used. Furthermore, there is a 1–1 correspondence between the bins of this feasible packing of I' and the bins of the specified feasible solution of I that do not contain $\{i_1, \ldots, i_k\}$, such that corresponding bins contain the same number of pieces, and the pieces of a bin in I dominate the pieces of the corresponding bin for I'.

Informally, this implies that we can add all sorts of "number of pieces-per-bin" constraints without affecting the validity of the domination principle.

As was done for the 1/5-dual approximation algorithm, we can view the algorithm as producing a sequence of progressively smaller bin-packing instances, $I = I_0, I_1, \ldots, I_p = \emptyset$, where for all $j > 0$ the pieces in $I_{j-1} - I_j$ are precisely the pieces packed in one bin.

We show again that at each point the optimal value is decreased by one in *some* sense. In the 1/5 case the situation was somewhat easier, and only the usual $OPT_{BP}(I)$ and the novel *QUASI-OPT*(I) were used. Here we use $OPT_{BP}(I, k)$ for various values of k and a variant of the *QUASI-OPT*(I) parameter used before. Finally, let j_i denote the total number of bins packed by the algorithm 1/6-*dual* after stage i.

Stage 1. This is the simplest of all the stages. Let I_l denote the current instance. If $p_i \in [2/3, 1]$, since all pieces are greater than 1/6, we know that i is packed in the optimal packing with at most one other piece. This piece can have size at most $1 - p_i$, and we pack p_i with the largest such piece. By the domination principle, the instance consisting of the remaining pieces, I_{l+1} is such that $OPT_{BP}(I_{l+1}) \leq OPT_{BP}(I_l) - 1$. Inductively, it follows that $OPT_{BP}(I_{j_1}) \leq OPT_{BP}(I) - j_1$. It is trivial to see that no bin is packed in this stage with capacity more than 1.

Stages 2 *and* 3. These two stages complement one another, so we present their analysis together as well. For the instance I_{j_1} consider an optimal packing that has as few 1-bins as possible. Suppose that this optimal packing has k bins that are packed with one or two pieces, and assume that the **guess** in Stage 2 is k. In any

2-bin, the smaller piece must have size $\leq 1/2$, and since all piece sizes are $<2/3$, the larger piece in a 2-bin has size less than 2/3. Note that, since all piece sizes are less than 2/3, it is impossible for there to be both 1-bins and 3-bins in the optimal solution that we consider. (Otherwise, there is some piece of size $\leq 1/3$ in a 3-bin that could be moved to a 1-bin, thereby reducing the number of 1-bins.)

We first assume that there is a 1-bin in the optimal solution. All of the bins in the specified optimal solution contain either one or two pieces, and the **guess** k is the number of bins in the optimal packing of the remaining instance. Suppose that this instance has p pieces with sizes in the range (1/6, 1/2] and q pieces with sizes in the range (1/2, 2/3), and suppose that the algorithm was allowed to pack bins with L[1/2, 2/3] for as long as possible, and then packed the remaining pieces one per bin. (In other words, the **guess** k is hypothetically ignored.) How many bins would the algorithm pack? If $p \geq q$, then every bin will be packed with two pieces (except possibly the last) and the total number of bins used is $\lfloor (p+q)/2 \rfloor$. Given that there is an optimal solution using only 1- and 2-bins, this must be no more than the optimal number of bins, k. Thus, in this case, the original algorithm packs all remaining pieces in a superoptimal number of bins. Suppose instead that $p < q$; in this case q bins are used by the algorithm, and this again must be at most the optimum number of bins since there can be at most one piece with size greater than 1/2 in a bin. Therefore, if there is 1-bin in the specified optimal solution, the algorithm completes the packing in Stage 2, using no more than the optimum number of bins. Since 2/3 + 1/2 is 7/6, no bin is ever filled with more than 7/6.

We must now consider the case in which there are no 1-bins in the optimal solution selected. In Stage 2, we guess the number of 1- or 2-bins, and thus the number guessed is simply the number of 2-bins in the specified optimal solution. As before, each of these 2-bins contains a piece $<2/3$ and a piece $\leq 1/2$. Using the notation of above, by considering the specified optimal solution, we see that $p \geq q$ and $p + q \geq 2k$. These conditions ensure that all bins packed in Stage 2 have two pieces. Furthermore, the $2k$ pieces selected by the algorithm must dominate the $2k$ pieces packed in the specified optimal solution, and thus applying the generalized domination principle, we know that for the instance remaining after this stage, I_{j_2}, there is a feasible packing using $OPT_{BP}(I_{j_1}) - k$ bins, where each bin has at least three pieces. As a result, $OPT_{BP}(I_{j_2}, 3) \leq OPT_{BP}(I_{j_1}) - k$.

For Stage 3, we focus on $OPT_{BP}(I_l, 3)$. Consider any piece with $p_i = 1/2 + \delta$, $\delta \geq 0$. Since all pieces are greater than 1/6, it cannot be packed in a 4-bin, so that for any restricted feasible packing (where each bin has at least three pieces), piece i is packed in a 3-bin. By applying the bounding lemma, we see that the smaller of the other pieces that i is packed with has size at most $(1 - (1/2 + \delta))/2 = 1/4 - \delta/2$ and the larger has size less than $1 - (1/2 + \delta + 1/6) = 1/3 - \delta$. Since we pack i with the largest such pieces, we can apply the generalized domination principle to get that $OPT_{BP}(I_{l+1}, 3) \leq OPT_{BP}(I_l, 3) - 1$. By repeating this inductively, we see that at the end of Stage 3, $OPT_{BP}(I_{j_3}, 3) \leq OPT_{BP}(I_{j_2}, 3) - (j_3 - j_2)$.

Stages 4 *and* 5. We shall show that the pieces packed in these two stages dominate the pieces contained in bins with some piece at least 5/12 in some optimal solution of I_{j_3} (subject to the constraint that every bin has at least three pieces). The first trivial observation is that any piece with size at least 5/12 can be feasibly packed with at most 3 other pieces of size greater than 1/6. Consider a feasibly packed 4-bin containing a piece i with $p_i \in [5/12, 1/2)$. The bounding lemma reveals that the other pieces in the bin have sizes at most 7/36, 5/24, and 1/4.

Consider a packing corresponding to the optimal value $OPT_{BP}(I_{j_3}, 3)$. There is some number, k, of 4-bins that contain a piece of size in the range [5/12, 1/2). Assume that the **guess** in Stage 4 is k. The pieces chosen in Stage 4 must dominate the pieces actually packed in the k bins of the specified optimal solution. In addition, the number of pieces of size [5/12, 1/2) packed by the algorithm in Stage 4 is k, which is the number of such pieces in the 4-bins of the specified optimal solution. This implies that there is a correspondence between the pieces with size in [5/12, 1/2) remaining to be packed in Stage 5 and the pieces in this range packed in 3-bins in the optimal solution so that the pieces in the optimal solution dominate those left to be packed.

It is easy to see that for each 3-bin in the specified optimal solution, a 3-bin will be packed in Stage 5, and the piece with $p_i \geq 5/12$ used in Stage 5 will be no larger than the largest piece in the corresponding bin in the optimal solution. A simple application of the bounding lemma shows that any piece of size $5/12 + \delta$, when packed in a 3-bin, is packed with pieces no larger than $7/24 - \delta/2$ and $5/12 - \delta$. In Stage 5, we are packing a piece of size $5/12 + \delta$, where δ is larger than the corresponding piece in the 3-bin of the specified optimal solution, and thus the bounds on the accompanying pieces are more generous. As a result, the pieces packed in Stages 4 and 5, together, must dominate the pieces occurring in bins with a piece of size at least 5/12 of the specified optimal solution. Applying the generalized domination principle, we get that $OPT_{BP}(I_{j_5}, 3) \leq OPT_{BP}(I_{j_3}, 3) - (j_5 - j_3)$. Finally, we note that $7/36 + 5/24 + 1/4 + 1/2$ is $(14 + 15 + 18 + 36)/72 = 83/72$, and $5/12 + \delta + 7/24 - \delta/2 + 5/12 - \delta$ is $27/24 - \delta/2$, both of which are less than 7/6.

Stages 6 *and* 7. These two stages are fairly straightforward. Consider any optimal solution corresponding to $OPT_{BP}(I_{j_5}, 3)$, and suppose there are k 3-bins in this solution. Once again, assume that the **guess** in Stage 6 is k. Since the smallest piece in any 3-bin is no more than 1/3, and all pieces remaining are less than 5/12, it is clear that the pieces packed in Stage 6 dominate the pieces packed in 3-bins in our specified optimal solution. By the generalized domination principle, we know that there is a feasible solution, *where every bin has at least four pieces*, of the instance I_{j_6}, which uses at most $OPT_{BP}(I_{j_5}, 3) - k$ bins. In other words, $OPT_{BP}(I_{j_6}, 4) \leq OPT_{BP}(I_{j_5}, 3) - k$.

Any piece of size $1/3 + \delta$, $\delta \geq 0$ cannot be packed with 4 other pieces of size greater than 1/6. Thus in any packing corresponding to $OPT_{BP}(I_l, 4)$ (for $l \geq j_6$) we must pack any such piece in a 4-bin. Applying the bounding lemma, we see that the sizes of the other pieces in the bin are bounded from above by $2/9 - \delta/3$, $1/4 - \delta/2$, and $1/3 - \delta$. Thus, if we pack the piece of size $1/3 + \delta$ with the largest such pieces, we can apply the generalized domination principle to see that $OPT_{BP}(I_{l+1}, 4) \leq OPT_{BP}(I_l, 4) - 1$. Repeating this procedure, inductively we see that $OPT_{BP}(I_{j_7}, 4) \leq OPT_{BP}(I_{j_6}, 4) - (j_7 - j_6)$.

To conclude these stages we must once again note that $1/3 + 5/12 + 5/12$ is $14/12 = 7/6$, and $2/9 - \delta/3 + 1/4 - \delta/2 + 1/3 - \delta + 1/3 + \delta$ is $(8 + 9 + 12 + 12)/36 - (5\delta)/6$, which is at most $41/36 < 7/6$.

Stages 8 *and* 9. Consider a packing corresponding to $OPT_{BP}(I_{j_7}, 4)$. Focus attention on the pieces of size at least 7/24 (and, of course, less than 1/3, since all other pieces have been packed). Some are in 4-bins and the remainder are in 5-bins. If such a piece is in a 5-bin, we once again apply the bounding lemma to discover that the remaining pieces are of sizes at most 17/96, 13/72, 9/48, and

5/24. (To remind the reader where these numbers come from, consider for example, the third largest piece; the smaller two pieces are each more than 1/6, and the largest piece is at least 7/24. This leaves at most 9/24 for the remaining two pieces, and thus the smaller of the two is no more than 9/48. Or one may simply plug the suitable parameters into the bounding lemma.) Thus, if the optimal solution selected has k 5-bins with a piece of size at least 7/24, and the **guess** of Stage 8 is done correctly, the pieces packed in this stage must dominate the pieces in the k bins of the optimal solution selected.

Next we invoke the strongest part of the generalized domination principle. We need something stronger than $OPT_{BP}(I_{j_8}, 4) \leq OPT_{BP}(I_{j_7}, 4) - k$. Let $OPT^*_{BP}(I, 4)$ denote the optimum value when we impose the additional constraint that any piece of size at least 7/24 must be packed in a 4-bin. In the specified optimal solution, except for the k 5-bins, all of these pieces are indeed packed in 4-bins. Thus we have a feasible solution for $\tilde{I} = I_{j_7} - \{i_1, \ldots, i_l\}$ where p bins are used, and no piece of size at least 7/24 is in a 5-bin. The generalized domination principle ensures that there is feasible packing of I_{j_8} where there is a strong correspondence with $\tilde{I}$. Thus we have a packing of I_{j_8} such that p bins are used, and for any 5-bin of this packing, the pieces are no bigger than the corresponding packing of $\tilde{I}$, and thus no 5-bin contains a piece of size at least 7/24. Simply put, $OPT^*_{BP}(I_{j_8}, 4) \leq OPT_{BP}(I_{j_7}, 4) - k$.

To complete Stage 9, the proof is fairly simple. Consider the largest piece i; if $p_i = 7/24 + \delta$, we consider packing it in a 4-bin. The bounding lemma shows that the other pieces in this bin are at most $17/72 - \delta/3$, $13/48 - \delta/2$ and p_i. Thus, if we pack the largest such pieces, we can apply the generalized domination principle to get that $OPT^*_{BP}(I_{l+1}, 4) \leq OPT^*_{BP}(I_l, 4) - 1$. Repeating this, inductively we show that $OPT^*_{BP}(I_{j_9}, 4) \leq OPT^*_{BP}(I_{j_8}, 4) - (j_9 - j_8)$. The weary reader may wish to verify that indeed the upper bounds ensure that no bin is ever packed with more than 7/6. Of course, the punch line is that since all pieces in I_{j_9} are less than 7/24, it follows that $OPT^*_{BP}(I_{j_9}, 4) = OPT_{BP}(I_{j_9}, 4)$.

Stage 10. In this stage we need to introduce a notion of quasi-feasibility. Call a bin-packing solution *quasi-feasible* if for all bins that contain five pieces, the capacity used is no more than 1, but for all other bins, the "capacity" used may be as much as 7/6. A *quasi-optimal solution* is a quasi-feasible solution that uses the minimum number of bins, *QUASI-OPT*(I_l). Clearly, *QUASI-OPT*(I_{j_9}) $\leq$ $OPT_{BP}(I_{j_9}, 4)$. It is easy to see that since all pieces are smaller than 7/24, any four pieces can be packed together quasi-feasibly. This implies that if there is a quasi-optimal solution with a 5-bin, then there is one with the smallest piece in a 5-bin.

Choose a quasi-optimal solution such that the smallest piece i is in a 5-bin, if possible. If there is no 5-bin, our algorithm must use no more bins than *QUASI-OPT*(I_l), since packing a few bins with five pieces can only help us, because any four pieces can be quasi-feasibly packed together. Thus we may assume that the quasi-optimal solution selected does have a 5-bin containing the smallest piece. For one last time, apply the bounding lemma, to see that if $p_i = 1/5 - \delta$, the other pieces of the bin have sizes at most $1/5 + \delta/4$, $1/5 + 2\delta/3$, $1/5 + 3\delta/2$, and 7/24. We select the largest such pieces, so that with i, they must dominate the pieces in the bin with piece i of the specified quasi-optimal solution. Using a variant of the domination principle for quasi-feasibility (which follows directly from the generalized domination principle) we see that the new instance I_{l+1} is such that *QUASI-OPT*(I_{l+1}) $\leq$ *QUASI-OPT*(I_l) $- 1$. Applying this inductively, we see that the number of bins packed in this last stage $j_{10} - j_9$ is at most *QUASI-OPT*(I_{j_9}).

Of course, we must add $1/5 + \delta/4$, $1/5 + 2\delta/3$, $1/5 + 3\delta/2$, $7/24$, and $1/5 - \delta$ to get $1 + 11/120 + 17\delta/12$. Since $\delta < 1/30$, we see that this sum is bounded by $1 + 5/36 < 7/6$!!!!

This completes the proof of the guarantees of 1/6-*dual*. By tracing through the inequalities proved for each stage, and combining them, the thorough reader can verify that in fact, the total number of bins packed is bounded by $OPT_{BP}(I)$. The moral of this proof is not that it is true, but that it was somewhat mechanical (and still true). It is the hope of the authors that any reader who has reached this point, could in fact produce a 1/7-dual algorithm that is moderately efficient, by using nearly identical techniques. Furthermore, although the notation used in the proof is somewhat cumbersome, the intuition behind the proof, as given in the main portion of the paper, is very easy to understand. We believe that this is in sharp contrast to the weighting function techniques, where the intuition behind the arguments is only understood when all of the cases have been worked out scores of pages later.

ACKNOWLEDGMENTS. We would like to thank Dick Karp and Alexander Rinnooy Kan for their many useful suggestions. We are also indebted to the anonymous referee who brought Reference [11] to our attention.

REFERENCES

1. COFFMAN, JR, E. G., GAREY, M. R., AND JOHNSON, D. S. An application of bin-packing to multiprocessor scheduling. *SIAM J. Comput. 7* (1978), 1–17.
2. FERNANDEZ DE LA VEGA, W., AND LUEKER, G. S. Bin packing can be solved within $1 + \epsilon$ in linear time. *Combinatorica 1* (1981), 349–355.
3. FRIESEN, D. K. Sensitivity analysis for heuristic algorithms. Tech. Rep. UIUCDCS-R-78-939, Department of Computer Science, Univ. of Illinois, Urbana-Champaign, 1978.
4. FRIESEN, D. K. Tighter bounds for the multifit processor scheduling algorithm. *SIAM J. Comput. 13* (1984), 170–181.
5. GAREY, M. R., AND JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.
6. GRAHAM, R. L. Bounds for certain multiprocessing anomalies. *Bell Syst. Tech. J. 45* (1966), 1563–1581.
7. GRAHAM, R. L. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math. 17* (1969), 263–269.
8. HOCHBAUM, D. S., AND SHMOYS, D. B. A bin packing problem you can almost solve by sitting on your suitcase. *SIAM J. Algebraic Discrete Methods 7* (1986), 247–257.
9. IBARRA, O. H., AND KIM, C. E. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM 22*, 4 (Oct. 1975), 463–468.
10. KARMARKAR, N., AND KARP, R. M. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science.* IEEE, New York, 1982, pp. 312–320.
11. LANGSTON, M. A. Processor scheduling with improved heuristic algorithms. Doctoral dissertation, Texas A&M Univ., College Station, Tex., 1981.
12. LAWLER, E. L. Fast approximation algorithms for knapsack problems. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science.* IEEE, New York, 1977, pp. 206–213.
13. SAHNI, S. K. Algorithms for scheduling independent tasks. *J. ACM 23* 1 (Jan. 1976), 116–127.

RECEIVED OCTOBER 1985; REVISED JANUARY 1986; ACCEPTED JANUARY 1986