A Graduate Course on Algorithms	**Spring Semester, 1997/8**
Lecture 3: March 25	
Lecturer: Uri Zwick	*Scribe: Eilon Reshef*

3.1 Bin Packing

3.1.1 Definitions

Recall the definition of the Bin Packing (BP) problem:

PROBLEM: BP
INSTANCE: Multiset of real numbers $I = \{s_1, \ldots, s_n\}$, $\forall i\ 1 \geq s_i > 0$.
SOLUTION: A *partition* of I into subsets $I_1, \ldots, I_k$ s.t. $\forall i\ \sum_{s \in I_i} s \leq 1$.
OPTIMAL SOLUTION: A solution with a minimum number of subsets k.

Recall that the Bin Packing problem is NP-hard [GJ79] and that there is no polynomial approximation algorithm with an approximation ratio $\alpha < 3/2$, unless $\mathcal{P} = \mathcal{NP}$.

While there is no PTAS for Bin Packing, we show that permitting a small additive constant, in addition to the multiplicative constant, allows us to obtain multiplicative constants arbitrarily close 1. More specifically, we show that for every $\epsilon > 0$, there exists an algorithm $\mathcal{A}_\epsilon$ whose running time is polynomial (and even linear) in n such that for every instance I we have

$$\mathcal{A}_\epsilon(I) \leq (1+\epsilon)OPT(I) + 1.$$

This provides us with a so-called assymptotic polynomial time approximation scheme (APTAS) for Bin Packing, along the lines of [FdlVL81].

Remark: While the algorithm presented below has a running time exponential in $1/\epsilon$, there exists an AFPTAS (asymptotic fully polynomial time approximation scheme) for Bin Packing, i.e., an algorithm that runs in time polynomial in n and in $1/\epsilon$.

Remark: There exists a polynomial algorithm, due to Karmarkar and Karp [KK82] that returns a packing such that $\mathcal{A}(I) < OPT(I) + O(\log^2 OPT(I))$. This approximation has an asymptotic approximation ratio of 1.

Open Problem: Is there a polynomial time algorithm A that guarantees $\mathcal{A}(I) \leq OPT(I) + d$ for some constant d?

3.1.2 Basic Observations

Before we proceed with the presentation of the algorithm, we note several properties of optimal solutions of Bin Packing. Although technical at first sight, these observations are simple and intuitive, and will be used later in the proofs below.

Observation 3.1.1 *If for instances I', I of BPwe have $I' \subseteq I$, then $OPT(I') \leq OPT(I)$.*

Observation 3.1.2 *If for instances $I' = \{s'_1, \ldots, s'_n\}$, $I = \{s_1, \ldots, s_n\}$ of BPwe have $s'_i \geq s_i$ for every $1 \leq i \leq n$, then $OPT(I') \geq OPT(I)$. In this case we say that I'* dominates *I and denote it by $I' \succeq I$.*

Observation 3.1.3 *$OPT(I) \geq size(I)$, where $SIZE(I) = \sum_{1 \leq i \leq n} s_i$.*

3.1.3 An APTAS for Bin Packing

Theorem 3.1 *There exists an asymptotic polynomial time approximation scheme (APTAS) for Bin Packing, i.e., $\forall \epsilon > 0$, there exists an algorithm $\mathcal{A}_\epsilon$ that runs in time linear in n and returns a packing of the input I with $\mathcal{A}_\epsilon(I) \leq (1+\epsilon)OPT(I) + 1$.*

Outline of the Algorithm

To construct the approximation algorithm, we present two constructive reductions.

First, we reduce BP into BP$[\delta]$, a variant of BP in which all the elements in the set I are larger than a threshold δ, i.e., $\forall i \; s_i \geq \delta$.

Problem: BP$[\delta]$

Instance: Multiset of real numbers $I = \{s_1, \ldots, s_n\}$, $\forall i \; 1 \geq s_i \geq \delta$.

Solution: Same.

Optimal Solution: Same.

Given an polynomial algorithm that approximates BP$[\delta]$ we construct a polynomial algorithm that approximates BP.

Then, we reduce BP$[\delta]$ into BP$[\delta, m]$, a variant of BP$[\delta]$ in which all the values $s_i \in I$ are chosen from a set of at most m different values.

Problem: BP$[\delta, m]$

Instance: Multiset of real numbers $I = \{s_1, \ldots, s_n\}$, $\forall i \; 1 \geq s_i \geq \delta$. There exists a set $V = \{v_1, \ldots, v_m\}$, s.t. $\forall i \; s_i \in V$.

Solution: Same.

Optimal Solution: Same.

Finally, we present an algorithm that gives an exact solution to BP$[\delta, m]$, completing the description of the algorithm.

Reduction from BP into BP$[\delta]$

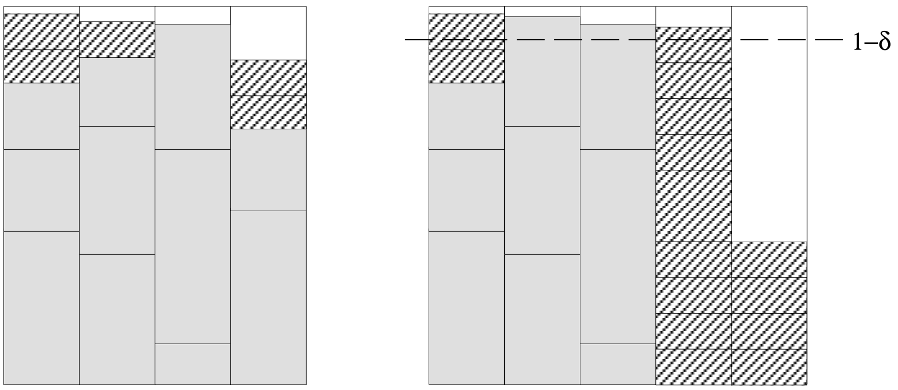
The reduction from BP into BP$[\delta]$ is amazingly simple. For some δ to be specified later, the algorithm first removes all the "small" items, i.e., the items for which $s_i < \delta$. Then, it applies the approximation algorithm for BP$[\delta]$, and finally greedily packs the "small" items into the bins.

Let $I_{\geq\delta} \triangleq \{s_i \in I \mid s_i \geq \delta\} \subseteq I$.

The heart of the redution relies on the observation that given a packing of $I_{\geq\delta}$, one of the following two cases must hold:

1. The small elements can be added to the existing bins. In this case, the approximated solution for $I_{\geq\delta}$ produced by the approximation algorithm for BP[δ] is also a good approximate solution for the instance I.

2. New bins are needed for placing the small items. In this case, all bins except perhaps a single bin are "almost" full, and therefore the packing is almost optimal.



Case 1:
All small-size elements can fit in

Case 2:
More bins are opened for the small elements

The above claim is formalized in the following lemma:

Lemma 3.2 *For $\delta \leq 1/2$, given a packing of $I_{\geq\delta}$ in β bins, one can construct a packing of I s.t.*

$$\mathcal{A}(I) \leq \max\{\beta, (1+2\delta)OPT(I) + 1\}.$$

Proof: Greedily add the small elements in $I \setminus I_\delta$ to the packing, e.g., using First-Fit. If no new bins were used, as in the first case above, then clearly $\mathcal{A}(I) = \beta$. Otherwise, there is at most one bin with δ or more unused space, so the size of the input satisfies

$$SIZE(I) \geq (\mathcal{A}(I) - 1)(1 - \delta).$$

By Observation 3.1.3, $SIZE(I) \leq OPT(I)$, so

$$\mathcal{A}(I) \leq \frac{1}{1-\delta} OPT(I) + 1.$$

For $\delta \leq 1/2$, this yields $\mathcal{A}(I) \leq (1+2\delta)OPT(I) + 1$. ∎

Theorem 3.3 *Given an algorithm $\mathcal{A}_\delta$ that approximates BP[δ], i.e., for every instance I of BP[δ],*

$$\mathcal{A}_\delta(I) \leq (1+\epsilon)OPT(I) + 1,$$

we can construct an algorithm $\mathcal{A}$ that approximates BP, i.e., for every instance I of BP,

$$\mathcal{A}(I) \leq (1+\epsilon)OPT(I) + 1.$$

Proof: Apply $\mathcal{A}_\delta$ on $I_{\geq\delta}$ for $\delta = \epsilon/2 \leq 1/2$ (recall that any greedy algorithm $\mathcal{A}_G$ satisfies $\mathcal{A}_G(I) \leq 2 \cdot OPT(I) + 1$, so we may assume $\epsilon \leq 1$). By the above lemma,

$$\mathcal{A}(I) \;\leq\; \max\{\mathcal{A}_\delta(I_{\geq\delta}), (1+\epsilon)OPT(I)+1\} \;\leq\; \max\{(1+\epsilon)OPT(I_{\geq\delta})+1, (1+\epsilon)OPT(I)+1\}.$$

Since $I_{\geq\delta} \subseteq I$, by Observation 3.1.1, $OPT(I_{\geq\delta}) \leq OPT(I)$, completing the proof. ■

Note that the additional running time of the algorithm is linear in n.

Reduction from BP$[\delta]$ **into BP**$[\delta, m]$

Theorem 3.4 *Given an algorithm $\mathcal{A}_{\delta,m}$ that gives an exact solution for BP$[\delta, m]$, we can construct an algorithm $\mathcal{A}_\delta$ such that for each instance I of BP$[\delta]$,*

$$\mathcal{A}_\delta(I) \leq (1+\epsilon)OPT(I) + 1.$$

Proof: Let $k = \lceil \epsilon\delta n \rceil$ and let $m = \lfloor \frac{n}{k} \rfloor$. Note that $m \leq (\epsilon\delta)^{-1}$, and in particular is independent of n. Assume, without loss of generality, that $s_1 \geq s_2 \leq \ldots \geq s_n$. Split the elements of I into sets $G_1, \ldots G_{m+1}$ of cardinality k (except possibly the last), such that:

$$\{\underbrace{s_1, \ldots, s_k}_{G_1}, \cdots, \underbrace{s_{(i-1)k+1}, \ldots, s_{ik}}_{G_i}, \cdots, \underbrace{s_{mk+1}, \ldots, s_n}_{G_{m+1}}\}$$

For every i, define the set H_i to be a multiset that contains $|G_i|$ copies of the largest element in G_i. Clearly,

$$H_1 \succeq G_1 \succeq H_2 \succeq \;\ldots\; \succeq G_m \succeq H_{m+1} \succeq G_{m+1}.$$

The algorithm proceeds as follows:

1. Use $\mathcal{A}_{\delta,m}$ to find an optimal packing for $\underline{I} \;=\; H_2 \bigcup H_3 \bigcup \ldots \bigcup H_{m+1}$.
2. Replace the elements of H_i by the corresponding elements in G_i. Since $H_i \succeq G_i$, the new packing is legal.
3. Greedily add the elements of G_1 to the packing.

Since $|G_1| \leq k$, we get that $\mathcal{A}(I) \leq OPT(\underline{I}) + k$. Since $\underline{I}$ is dominated by I, Observation 3.1.2 guarantees that $OPT(\underline{I}) \leq OPT(I)$. Therefore:

$$\mathcal{A}(I) \leq OPT(I) + k. \tag{3.1}$$

Furthermore, since $\forall i\; s_i \geq \delta$, $SIZE(I) \geq n\delta$, and since Observation 3.1.3 guarantees that $SIZE(I) \leq OPT(I)$,

$$k \leq \epsilon\delta n + 1 \leq \epsilon \cdot SIZE(I) + 1 \leq \epsilon \cdot OPT(I) + 1. \tag{3.2}$$

Summing up (3.1) and (3.2), we get:

$$\mathcal{A}(I) \leq (1+\epsilon)OPT(I) + 1$$

■

Note that since the number of sets $m + 1 = |\{G_i\}|$ is independent of n, there is no need to sort the input set. The sets G_i, and H_i, can be constructed using a linear time selection algorithm.

Exact Solution for BP$[\delta, m]$ using Dynamic Programming

We shall now present an algorithm for finding an optimal packing for an input I of BP$[\delta, m]$ that runs in time polynomial in n. The algorithm uses dynamic programming and relies on the following two observations.

First, note that given the set V containing the different values for $s_i \in I$, an instance I of BP$[\delta, m]$ can be represented as an m-tuple of integer numbers $\vec{I} = (n_1, \ldots, n_m)$, s.t. $n = |\vec{I}| = \sum_{1 \le i \le m} n_i$. Thus, there are at most n^m different input configurations of size $\le n$.

Second, note that since each item is of size $\ge \delta$, there are at most $r = \lfloor \frac{1}{\delta} \rfloor$ items in each bins. Thus, the number of input configurations that can be packed in a single bin q is bounded by m^r.

Let $\mathcal{P}_k$ denote the set of all input configurations of size $\le n$ that can be packed in k bins, i.e.

$$\mathcal{P}_k \triangleq \{\vec{I} \mid |\vec{I}| \le n, OPT(\vec{I}) = k\}.$$

By the above argument $q = |\mathcal{P}_1| \le m^r$, and thus $\mathcal{P}_1$ can be computed in time independent of n.

Now let us describe a simple inductive algorithm for finding the optimal packing for an input $\vec{I}$ of size n.

Suppose that the algorithm has determined the sets $\mathcal{P}_1, \ldots, \mathcal{P}_k$. In phase $k+1$, the algorithm computes the set $\mathcal{P}_{k+1}$ by adding to each input configuration $\vec{I} \in \mathcal{P}_k$ the items in $\mathcal{P}_1$, namely the different ways to pack an additional bin:

$$\mathcal{P}_{k+1} = \{\vec{I} \mid \vec{I} = \vec{I_k} + \vec{I_1}, \vec{I_k} \in \mathcal{P}_k, \vec{I_1} \in \mathcal{P}_1\}.$$

Since clearly if $|\vec{I}| = n$, $\vec{I} \in \mathcal{P}_k$ for some $k \le n$, the algorithm is guaranteed to terminate after at most n phases giving an optimal packing for I. In each phase, the number of examined states is bound by n^m, and the number of transitions is bound by $q = |\mathcal{P}_1| \le m^r$. Thus, the running time of the algorithm is polynomial in n.

Exact Solution for BP$[\delta, m]$ using an Integer Program (IP)

To complete the construction of a linear time algorithm for approximating Bin Packing, we show that BP$[\delta, m]$ can be solved in time linear in n using an integer program (IP).

To code BP$[\delta, m]$ as an integer program, note that packing an input $\vec{I}$ in k bins is equivalent to representing $\vec{I}$ as a linear combination of the vectors in $\mathcal{P}_1$ with non-negative integer scalars whose sum is k. Thus, if $\mathcal{P}_1 = \{I_1, \ldots, I_q\}$ then

$$\vec{I} \;=\; \sum_{j=1}^{q} c_j \cdot \vec{I_j},$$

where the number of bins $k = \sum_{j=1}^{q} c_j$.

To specify the integer program, define a matrix M whose columns are the vectors in $\mathcal{P}_1$,

$$M_{m \times q} \;=\; [I_1^T \;\cdots\; I_q^T].$$

Given the input $\vec{I} = (n_1, \ldots, n_m)$, where $n = |\vec{I}|$, define an integer program with variables $\vec{c} = (c_1, \ldots, c_q)$, minimizing the number of bins

$$\sum_{i=1}^{q} c_i$$

subject to the constraints:

$$M \cdot \vec{c} \geq \vec{I}^T$$

$$c_i \geq 0 \quad (1 \leq i \leq q).$$

This introduces an integer program with $q \leq m^r$ variables and $m + q$ constraints.

Using the algorithm of [Len83], one can solve the linear program in time polynomial in the number of constraints and exponential in the number of variables, but since in our computer-theoretical discussion both numbers are considered constant, the running time of the algorithm is independent of n (assuming, as usual, that arithmetic operations take one unit time).

Constructing the explicit description of the packing from the solution of the linear program can be carried out in time linear in n, so the running time of the algorithm is linear in n as needed.

3.2 Randomized Rounding for Multicommodity Flow

In this section we describe an approximation algorithm for a variant of a multicommodity flow problem.

In this problem, we are given a directed graph and a collection of (*source*, *destination*) pairs. We are asked to find a collection of directed paths linking the sources with the destinations, such that the congestion of the edges is minimized. Formally,

Problem: (Unsplittable) Multicommodity Flow

Instance: Directed graph $G = (V, E)$, list of pairs $\{(s_1, t_1), \ldots, (s_k, t_k)\} \subseteq V^2$.

Solution: List of paths $\{P_1, \ldots, P_k\}$ s.t. P_i leads from s_i to t_i.

Optimal Solution: Minimize $\max_{e \in E} c(e)$, where $c(e)$ denotes the congestion of an edge, i.e., $c(e) = |\{P_i \mid e \in P_i\}|$.

The multicommodity flow problem is NP-complete [GJ79], and the randomized approximation algorithm below is due to Raghavan and Thomposon [RT87].

3.2.1 Formulating Multicommodity Flow as an Integer Program

To represent the multicommodity flow problem as an integer program, for each (*source*, *destination*) pair and edge (u, v), define a variable $x_i(u, v)$ representing the overall flow from s_i to t_i through the edge (u, v).

The integer program is defined as follows:

$$\min W_{LP}$$

subject to the following three sets of constraints.

The first set of constraints,

$$\sum_{w:(v,w) \in E} x_i(v, w) - \sum_{u:(u,v) \in E} x_i(u, v) = \begin{cases} 1 & \text{if } v = s_i \\ -1 & \text{if } v = t_i \\ 0 & \text{otherwise} \end{cases} \quad \forall v \in V, \ \forall i = 1, \ldots, k, \tag{3.3}$$

verifies that the variables $x_i(u,v)$ indeed represent a legal flow.

The second set of constraints,

$$\sum_{1 \le i \le k} x_i(u,v) \le W_{LP} \quad \forall (u,v) \in E, \tag{3.4}$$

asserts that the congestion is indeed bound by W_{LP}.

Finally,

$$x_i(u,v) \in \{0,1\} \quad \forall (u,v) \in E,\ \forall i = 1,\ldots,k. \tag{3.5}$$

forces the flow to be a $0-1$ flow.

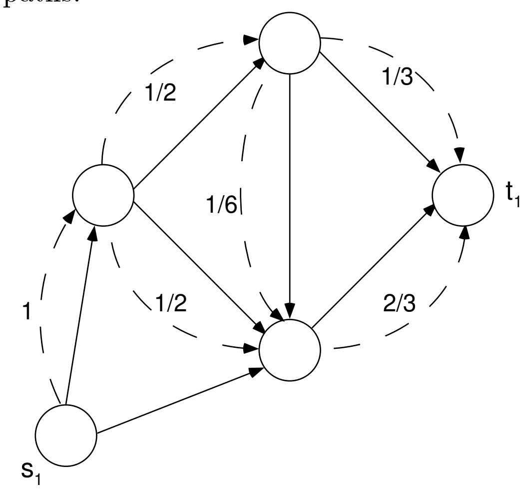
3.2.2 Approximation Algorithm

One may relax the integer program and transform it into a linear program (LP) by replacing the constraints in (3.5) by:

$$x_i(u,v) \ge 0 \quad \forall (u,v) \in E,\ \forall i = 1,\ldots,k. \tag{3.6}$$

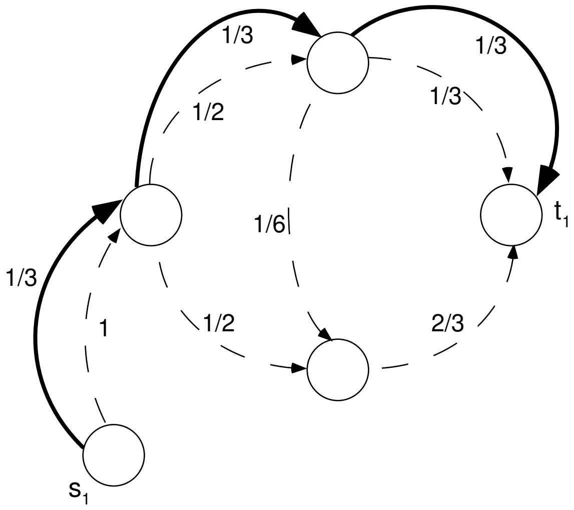
Since the linear program has a feasible solution, an optimal solution is guaranteed to exist and can be found in polynomial time.

Note, however, that the solution of the linear program is not necessarily integral, since the flow may be split along different paths.



To convert the resulting flows into a $0-1$ flow, we handle each of the k flows separately.

Consider G_i, the graph containing the non-zero edges of the i-th flow. First, remove from G_i any cycles. This operation retains the validity of the flow and may only decrease the congestion of the edges. Next, consider a non-integral path from s_i to t_i. In each such path, find an edge $e=(u,v)$ with a minimal flow $x_i(u,v)$, and remove its flow from the path.



Clearly, at each step at least one edge is discarded from G_i, so the i-th flow is thus decomposed into l_i paths $P_1, \ldots, P_{l_i}$ from s_i to t_i, such that along path P_{i_j} there are α_{i_j} units of flow from s_i to t_i, and

$$\sum_{j:(u,v)\in P_{i_j}} \alpha_{i_j} = x_i(u,v) \quad \forall (u,v) \in E, \forall i = 1, \ldots, k, \tag{3.7}$$

$$\sum_{j=1}^{l_i} \alpha_{i_j} = 1 \quad \forall i = 1, \ldots, k. \tag{3.8}$$

Now, to select a path P_i from s_i to t_i, the algorithm randomly picks P_{i_j} with probability α_{i_j}.

3.2.3 Analysis of the Algorithm

Theorem 3.5 *Let $\epsilon > 0$. If the optimal congestion W^* in a multicommodity flow problem satisfies $W^* = \Omega(\log n)$, then with probability $\geq 1 - \epsilon$, the algorithm constructs a solution with congestion $W \leq W^* + O(\sqrt{W^* \ln n})$.*

The exact constants in the terms $\Omega(\cdot)$ and $O(\cdot)$ depend on the value of ϵ, as will be evident from the proof below.

Proof:

The proof uses the following Chernoff bound:

Lemma 3.6 *Let X_i be independent random variables $x_i \in \{0,1\}$. Let $X = \sum_{i=1}^{n} X_i$, and let $\mu = E[X]$. Then, for all $\beta > 0$,*

$$Pr[X > (1+\beta)\cdot\mu] \leq \left(\frac{e^\beta}{(1+\beta)^{1+\beta}}\right)^\mu.$$

Corollary 3.7 *For $\sigma \geq \mu$,*

$$Pr[X > (1+\beta)\cdot\sigma] \leq \left(\frac{e^\beta}{(1+\beta)^{1+\beta}}\right)^\sigma.$$

Proof: Add random variables $x_{n+1}, \ldots, x_m$ such that $X' = \sum_{i=1}^{m} X_i$, $\sigma = E[X']$. Now,

$$Pr[X > (1+\beta) \cdot \sigma] \leq Pr[X' > (1+\beta) \cdot \sigma] \leq \left(\frac{e^{\beta}}{(1+\beta)^{1+\beta}} \right)^{\sigma}.$$

■

Corollary 3.8 *For $\beta \leq 1$, $\sigma \geq \mu$,*

$$Pr[X > (1+\beta) \cdot \sigma] \leq e^{-\beta^2 \sigma / 3}.$$

We are now ready to analyze the maximum congestion in the graph.

Using linearity of expectation and Equation (3.7), the expectation of the congestion of an edge (u, v) can be written as:

$$E[c(u,v)] = \sum_{i=1}^{k} \left(\sum_{j:(u,v) \in P_{i_j}} \alpha_{i_j} \right) = \sum_{i=1}^{k} x_i(u,v) \leq W_{LP} \leq W^*,$$

where the inequations follow from constraint (3.4) and from the optimality of the solution to the linear program.

Now, we can use the Chernoff bound in Corollary 3.8 to see that for $\beta \leq 1$,

$$Pr[c(e) \geq (1+\beta) W^*] \leq e^{-\beta^2 W^* / 3}. \tag{3.9}$$

Let

$$\beta = \sqrt{\frac{3 \ln(n^2/\epsilon)}{W^*}}.$$

With proper selection of constants, the conditions of the theorem guarantee that

$$W^* \geq 3 \ln(n^2/\epsilon),$$

and thus $\beta \leq 1$.

But

$$\beta^2 W^* = \frac{3 \ln(n^2/\epsilon)}{W^*} \cdot W^* = 3 \ln(n^2/\epsilon). \tag{3.10}$$

Substitution (3.10) into (3.9),

$$Pr[c(e) \geq (1+\beta) W^*] \leq e^{-\ln(n^2/\epsilon)} = \frac{\epsilon}{n^2}. \tag{3.11}$$

Therefore, the probability that there exists an edge for which (3.11) holds is thus bounded by $|E| \cdot \frac{\epsilon}{n^2} \leq \epsilon$.

Thus, with probability $\geq 1 - \epsilon$,

$$W = \max_{e \in E} \{c(e)\} < (1+\beta) \cdot W^* = W^* + O(\sqrt{W^* \ln n}).$$

■

Bibliography

[FdlVL81] W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within $1+\varepsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability, a guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[KK82] N. Karmarkar and R.M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science, Chicago, Illinois*, pages 312–320. 1982.

[Len83] H.W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

[RT87] P. Raghavan and C. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.