Proof The routed flow is unsplittable by construction. The capacities are violated in Step (ii)(a) when they are rounded up. Let the demands be $d_1 \leq d_2 \leq \ldots \leq d_k$ in the increasing order. Observe that $d_1|d_2|\ldots|d_k$. We now show by induction that when the demands $d_1, \ldots, d_i$ have been routed, the capacity on each arc is at most d_i more than its initial capacity and is a multiple of d_i for each $1 \leq i \leq k$. The claim clearly holds for $i = 1$. By the induction hypothesis we assume the claim is true for $i-1$, i.e, the capacity is violated by at most d_{i-1} and is a multiple of d_{i-1}. While routing d_i, the capacity is increased to a multiple of d_i. Since $d_{i-1}|d_i$, the increase in the i^{th} step is bounded by $d_i - d_{i-1}$ which bounds the total violation at the end of i^{th} step by d_i as required. □

13.5 Bin Packing

In this section we show the Karmarkar-Karp algorithm for the bin packing problem. This is one of the earliest and is still one of the most sophisticated instances of using the iterative technique for an approximation algorithm. In an instance I of the one-dimensional bin packing problem, we are given n items each of which has a size between 0 and 1. The objective is to pack the items into a minimum number of unit-size bins. Let $\mathrm{OPT}(I)$ denote the number of bins required in an optimal solution to instance I. We present the following theorem due to Karmarkar and Karp.

Theorem 13.5.1 *There is a polynomial time algorithm which returns a solution with at most* $\mathrm{OPT}(I) + O(\log^2 \mathrm{OPT}(I))$ *bins.*

13.5.1 Linear Programming Relaxation

A natural linear programming relaxation is to require that each item is put in at least one bin and each bin can pack items of total size at most one, but this relaxation has a (multiplicative) integrality gap of 2; see exercises. To obtain an additive approximation, we need a so-called *configuration* linear programming relaxation for the problem. Consider an instance I of the bin packing problem with $n(I)$ items of $m(I)$ different types. For $1 \leq i \leq m(I)$, let b_i be the total number of items of type i and s_i be the common size of these items. Let $T_1, \ldots, T_N$ be all the possible configurations in which a single bin can be packed:

$$\{T_1, \ldots, T_N\} := \{(k_1, \ldots, k_{m(I)}) \in \mathbb{Z}_+^m : \sum_{i=1}^{m} k_i s_i \leq 1\}.$$

Note that N can be exponential in $n(I)$ and $m(I)$. Let $T_j = (t_{j1}, \ldots, t_{jm(I)})$ where t_{ji} denotes the number of items of type i in configuration j. The configuration linear programming relaxation for the bin packing problem is as follows.

$$\begin{array}{llcll} \text{minimize} & \displaystyle\sum_{j=1}^{N} x_j & & & \\ \text{subject to} & \displaystyle\sum_{j=1}^{N} t_{ji} x_j & \geq & b_i & \forall\, 1 \leq i \leq m(I) \\ & x_j & \geq & 0 & \forall\, 1 \leq j \leq N, \end{array}$$

where the constraints ensure that the configurations chosen contain at least b_i items for each type i. This linear programming has exponentially many variables, but there is a polynomial time algorithm to compute a fractional solution differing from the optimum by at most δ. Consider the dual of the linear program.

$$\begin{array}{llcll} \text{maximize} & \displaystyle\sum_{i=1}^{m(I)} b_i y_i & & & \\ \text{subject to} & \displaystyle\sum_{i=1}^{m(I)} t_{ji} y_i & \leq & 1 & \forall\, 1 \leq j \leq N \\ & y_i & \geq & 0 & \forall\, 1 \leq i \leq m(I), \end{array}$$

The dual program has m variables but exponentially many constraints, but if there is a polynomial time separation oracle to determine if a given y is a feasible solution, then the dual program can be solved by the ellipsoid algorithm. The separation problem for the dual linear program is to determine, given y, if there exists a configuration $T = \{t_1, t_2, \ldots, t_m\}$ so that $\sum_{i=1}^{m(I)} t_i y_i > 1$. This is equivalent to the following maximization problem on the variables t_i. Recall that s_i denotes the size of items of type i.

$$\begin{array}{llcll} \text{maximize} & \displaystyle\sum_{i=1}^{m(I)} y_i t_i & & & \\ \text{subject to} & \displaystyle\sum_{i=1}^{m(I)} s_i t_i & \leq & 1 & \\ & t_i & \in & \mathbb{Z}_+ & \forall\, 1 \leq i \leq m(I), \end{array}$$

where the constraint is the definition of a configuration. This is a knapsack problem where s_i and y_i correspond to the size and profit of item i respectively. So the separation problem is equivalent to a knapsack problem, which is NP-hard to solve optimally. Nevertheless, if there is a polynomial time *weak* separation oracle to determine whether a given y is a feasible dual solution with error at most δ on the dual objective function or y is an infeasible dual solution, then the dual program can be approximated by the ellipsoid algorithm with an error at most δ.

In the following we sketch how to obtain an approximate solution to the (primal) configuration linear program using the ellipsoid algorithm. For the dual program, we can solve the weak separation problem by rounding each y_i down to the nearest rational number that is a multiple of $\frac{\delta}{2n}$, and then use a dynamic programming algorithm to solve the resulting knapsack problem optimally in polynomial time. Using the ellipsoid algorithm with the weak separation oracle, we can then obtain a solution y^* to the dual problem with $y^*b \geq \text{OPT}_{dual} - \delta$, where b is the vector of the number of items of each type and OPT_{dual} is the optimal value of the dual program. Let $T'_1, T'_2, \ldots, T'_{N'}$ be the bin configurations that appeared as a separating hyperplane during the execution of the ellipsoid algorithm, where N' is bounded by a polynomial in $n(I)$ and $m(I)$. Consider the dual program restricted to the constraints that correspond to $T'_1, \ldots, T'_{N'}$ and let the optimal value of this restricted dual program be OPT'_{dual}. Then $y^*b \geq \text{OPT}'_{dual} - \delta$, since the weak separation oracle can always give the same answer as for the original problem. To obtain an approximate solution to the primal program, we obtain an optimal solution to the restricted primal program in which we use only the variables that correspond to the configurations in $T'_1, \ldots, T'_{N'}$ and delete all other variables; in other words, we obtain an optimal solution to the dual of the restricted dual program. Since there are only polynomially many variables and constraints, this resticted primal program can be solved optimally in polynomial time. Let OPT'_{primal} be the optimal value of the restricted primal program. Then we have

$$\text{OPT}'_{primal} - \delta = \text{OPT}'_{dual} - \delta \leq y^*b \leq \text{OPT}_{dual} = \text{OPT}_{primal}.$$

Therefore an optimal solution to the restricted primal program is an approximate solution to the primal program with additive error at most δ.

13.5.2 Characterization of Extreme Point Solutions

The following lemma is a direct consequence of the Rank Lemma.

Lemma 13.5.2 *Given any extreme point solution to the configuration linear program, there are at most $m(I)$ nonzero variables, where $m(I)$ is the number of different types of items in instance I.*

To illustrate the use of Lemma 13.5.2, we show a very simple algorithm with a good performance guarantee when the number of different types of items is small. Let $\text{LIN}(I)$ denote the optimal value of the configuration LP associated with instance I. Let $\text{SIZE}(I) = \sum_{i=1}^{m(I)} s_i b_i$ be the sum of the sizes of all the items in instance I.

Lemma 13.5.3 $\text{OPT}(I) \leq \text{LIN}(I) + \frac{m(I)+1}{2}$.

Proof Let x be an optimal extreme point solution of the configuration LP for instance I. Then, by Lemma 13.5.2, x has at most $m(I)$ nonzero variables. We open $\lfloor x_j \rfloor$ bins with configuration j for each j. The remaining items form an instance I'. Let $f_j = x_j - \lfloor x_j \rfloor$. Then $\text{SIZE}(I') \leq \text{LIN}(I') = \sum_j f_j$. We now find a packing for instance I' of cost at most $\text{SIZE}(I') + \frac{m(I)+1}{2}$. A packing for instance I' of cost at most $m(I)$ can be constructed by using one new bin for each configuration j with nonzero f_j, and then removing excess items (the items that appear in more than one configuration). On the other hand, any greedy packing algorithm will give a solution for instance I' of cost at most $2\text{SIZE}(I')+1$, since each bin, except possibly one, will be at least half full. Hence, the better of these two packings has cost at most the average, which is $\text{SIZE}(I') + \frac{m(I)+1}{2} \leq \sum_j f_j + \frac{m(I)+1}{2}$, which in turn gives a packing for instance I with cost at most $\sum_j \lfloor x_j \rfloor + \sum_j f_j + \frac{m(I)+1}{2} = \text{LIN}(I) + \frac{m(I)+1}{2}$, proving the lemma. □

13.5.3 Defining Residual Problems: Grouping and Elimination

Given Lemma 13.5.3, one would like to reduce the number of distinct item sizes of the input instance. The idea of grouping is to divide the items into groups of similar sizes, and create a residual problem by increasing the sizes of each item to the largest item size in its group, so as to decrease the number of distinct item sizes in the residual problem. By doing so, however, the total item size of the residual problem and hence the optimum of the residual problem increases, and so there is a tradeoff in choosing the parameters. The main idea of the Karmarkar-Karp algorithm is to define the residual problem iteratively, so that in each iteration the number of distinct item sizes decrease by a constant factor, while the optimum in the residual problem increases by only an extra $O(\log(\text{OPT}))$ number of bins. Applying this procedure inductively will lead to Theorem 13.5.1.

The following is a simple method to bound the optimum of the residual problem. Let I and J be two bin packing instances. We write $I \preceq J$ if there is an injective function f mapping items in I into items in J so that $\text{SIZE}(a) \leq \text{SIZE}(f(a))$ for each item $a \in I$, where $\text{SIZE}(a)$ is the size of item a. Clearly, if $I \preceq J$, then $\text{OPT}(I) \leq \text{OPT}(J)$, $\text{LIN}(I) \leq \text{LIN}(J)$ and $\text{SIZE}(I) \leq \text{SIZE}(J)$. This method will be used throughout to analyze the performance of the grouping techniques.

13.5.3.1 Linear Grouping

The following process is called linear grouping with parameter k. Let I be an instance of the bin packing problem and let k be a positive integer. Divide the set I into groups $G_1, G_2, \ldots, G_q$ so that G_1 contains the k largest items, G_2 contains the next k largest items and so on. Hence $G_1 \succeq G_2 \succeq \cdots \succeq G_q$ and $|G_i| = k$ for all groups except the last group. Let G'_i be the multi-set of items obtained by increasing the size of each item in group G_i to the maximum size of an item in that group. Then $G_1 \succeq G'_2 \succeq G_2 \succeq \cdots \succeq G'_q \succeq G_q$. Let $J := \cup_{i=2}^{q} G'_i$ and $J' := G_1$. Then $J \preceq I \preceq J \cup J'$. Note that the items in J' can be trivially packed in k new bins, by using one new bin for each item. Therefore, we have the following after linear grouping.

Lemma 13.5.4 *After linear grouping with parameter k, we have*

- $\text{OPT}(J) \leq \text{OPT}(I) \leq \text{OPT}(J) + k,$

- $\text{LIN}(J) \le \text{LIN}(I) \le \text{LIN}(J) + k$,
- $\text{SIZE}(J) \le \text{SIZE}(I) \le \text{SIZE}(J) + k$.

13.5.3.2 Geometric Grouping

The following refinement of linear grouping is called geometric grouping with parameter k. This is the key idea in the Karmarkar-Karp algorithm to reduce the number of distinct item sizes iteratively.

Let I be an instance of the bin packing problem. Let $\alpha(I)$ denote the size of the smallest item in instance I. For $0 \le r \le \lfloor \log_2 \frac{1}{\alpha(I)} \rfloor$, let I_r be the instance consisting of those items from I whose sizes are in the interval $[2^{-(r+1)}, 2^{-r})$. Let J_r and J'_r be the instances obtained by applying linear grouping with parameter $k \cdot 2^r$ to I_r. Let $J := \cup_r J_r$ and $J' := \cup_r J'_r$.

Lemma 13.5.5 *After geometric grouping with parameter k, we have*

- $\text{OPT}(J) \le \text{OPT}(I) \le \text{OPT}(J) + k\lceil \log_2 \frac{1}{\alpha(I)} \rceil$,
- $\text{LIN}(J) \le \text{LIN}(I) \le \text{LIN}(J) + k\lceil \log_2 \frac{1}{\alpha(I)} \rceil$,
- $\text{SIZE}(J) \le \text{SIZE}(I) \le \text{SIZE}(J) + k\lceil \log_2 \frac{1}{\alpha(I)} \rceil$,
- $m(J) \le \frac{2}{k}\text{SIZE}(I) + \lceil \log_2 \frac{1}{\alpha(I)} \rceil$.

Proof From Lemma 13.5.4 it follows that $J_r \preceq I_r \preceq J_r \cup J'_r$ and thus $J \preceq I \preceq J \cup J'$. Hence $\text{OPT}(J) \le \text{OPT}(I) \le \text{OPT}(J \cup J') \le \text{OPT}(J) + \text{OPT}(J')$. Note that $\text{OPT}(J') \le \sum_r \text{OPT}(J'_r)$. Each J'_r contains at most $k \cdot 2^r$ items each of size less than 2^{-r}. Hence J'_r can be packed into at most k bins. Thus $\text{OPT}(J') \le k\lceil \log_2 \frac{1}{\alpha(I)} \rceil$, and therefore $\text{OPT}(J) \le \text{OPT}(I) \le \text{OPT}(J) + k\lceil \log_2 \frac{1}{\alpha(I)} \rceil$. The proofs of the next two inequalities are similar.

For each r, since we apply linear grouping with parameter $k \cdot 2^r$, all except the last group in J_r have $k \cdot 2^r$ items and the items in each group are of the same size. Also, each item in I_r has size at least $2^{-(r+1)}$, and thus we have

$$\text{SIZE}(I_r) \ge 2^{-(r+1)} \cdot n(I_r) \ge 2^{-(r+1)} \Big((m(J_r) - 1) \cdot k \cdot 2^r \Big).$$

Therefore $m(J_r) - 1 \le \frac{2}{k}\text{SIZE}(I_r)$ and thus

$$m(J) \le \frac{2}{k}\text{SIZE}(I) + \lceil \log_2 \frac{1}{\alpha(I)} \rceil.$$

□

13.5.3.3 Elimination of Small Items

In the geometric grouping process, the performance depends on the smallest item size. In the following we show that we can eliminate items of small sizes without affecting the approximation performance ratio much. Let I be an instance of the bin packing problem. Let g be a real number between 0 and 1. We say an item is *large* if its size is larger than $\frac{g}{2}$

and *small* otherwise. Consider a process which starts with a given packing of large items into bins, and then inserts the small items, using a new bin only when necessary. If the cost of the given packing of the large pieces is C, then the cost of the packing resulting from the process is at most $\max\{C, (1+g)\textsc{opt}(I)+1\}$; see the exercises.

13.5.4 Iterative Algorithm

With the grouping technique developed, we present the iterative algorithm in Figure 13.5. The parameters k and g will be set as $k=4$ and $g = \frac{1}{\text{SIZE}(I)}$.

```
Iterative Bin Packing Algorithm

  (i) Eliminate all small items of size at most $g$.
 (ii) While $\textsc{size}(I) > 1 + \frac{1}{1-\frac{2}{k}}\lceil \log_2 \frac{1}{g} \rceil$ do
        (a) Perform geometric grouping with parameter $k$ to create instance $J$ and $J'$.
            Pack $J'$ using at most $k\lceil \log_2 \frac{1}{g} \rceil$ new bins.
        (b) Compute an optimal extreme point solution $x$ to the configuration LP on
            instance $J$ with error at most 1.
        (c) For each $j$ with $x_j \geq 1$, create $\lfloor x_j \rfloor$ bins with configuration $j$ and remove
            the items packed from the problem.
(iii) Pack the remaining items using at most $2 + \frac{2}{1-\frac{2}{k}}\lceil \log_2 \frac{1}{g} \rceil$ bins.
 (iv) Insert the small items eliminated in Step 1, using new bins only when necessary.
```

Fig. 13.5. Karmarkar-Karp Bin Packing Algorithm

13.5.5 Correctness and Performance Guarantee

To prove the correctness and the performance guarantee, we will bound the number of iterations of the algorithm (in particular it will terminate) and the number of bins used by the algorithm. Let t be the number of iterations of the algorithm. For $1 \leq i \leq t$, let I_i be the instance at the beginning of iteration i, and J_i, J_i' be the instances resulting from geometric grouping on I_i, and let X_i and Y_i be the number of bins created in Step (ii)(c) and Step (ii)(a) in iteration i of the algorithm.

Lemma 13.5.6 *The iterative algorithm will terminate in at most $O(\ln n)$ iterations.*

Proof Note that the total size of the residual problem decreases geometrically:

$$\textsc{size}(I_{i+1}) \leq \textsc{lin}(I_{i+1}) \leq \sum_j (x_j - \lfloor x_j \rfloor) \leq m(J_i) \leq \frac{2}{k}\text{SIZE}(I_i) + \lceil \log_2 \frac{1}{g} \rceil,$$

where the second last inequality follows from Lemma 13.5.2 and the last inequality follows from Lemma 13.5.5. Therefore

$$\text{SIZE}(I_{i+1}) \le (\frac{2}{k})^i \text{SIZE}(J) + \lceil \log_2 \frac{1}{g} \rceil [1 + \frac{2}{k} + \cdots + (\frac{2}{k})^{i-1}] \le (\frac{2}{k})^i \text{SIZE}(I) + \frac{1}{1-\frac{2}{k}} \lceil \log_2 \frac{1}{g} \rceil.$$

Since $\text{SIZE}(I_t) > 1 + \frac{1}{1-\frac{2}{k}} \lceil \log_2 \frac{1}{g} \rceil$, this implies that $(\frac{2}{k})^t \text{SIZE}(I) \ge 1$ and hence

$$t \le \frac{\ln \text{SIZE}(I)}{\ln \frac{k}{2}} + 1 = O(\ln n).$$

Therefore the running time of the algorithm is polynomial. □

Lemma 13.5.7 *The total number of bins used by the algorithm is at most* $\text{OPT}(I) + O(\ln^2 \text{OPT}(I))$.

Proof To bound the number of bins used by the algorithm, we note that $\text{LIN}(I_{i+1}) \le \text{LIN}(J_i) + 1 - X_i \le \text{LIN}(I_i) + 1 - X_i$ by Step (ii)(c) of the algorithm, which implies that

$$\sum_{i=1}^{t} X_i \le \text{LIN}(I) + t.$$

By Lemma 13.5.5 we have for each $1 \le i \le t$

$$Y_i \le k \lceil \log_2 \frac{1}{g} \rceil.$$

Note that Step (iii) is always possible by any greedy algorithm. The number of bins produced by the algorithm after Step (iii) is at most

$$\sum_{i=1}^{t} X_i + t \cdot Y_i + 2 + \frac{2}{1-\frac{2}{k}} \lceil \log_2 \frac{1}{g} \rceil.$$

After inserting the small items, the total number of bins used by the algorithm is at most the maximum of $(1+g)\text{OPT}(I) + 1$ and

$$\text{OPT}(I) + \Big(\frac{\ln \text{SIZE}(I)}{\ln \frac{k}{2}} + 1\Big)\Big(1 + k\lceil \log_2 \frac{1}{g} \rceil\Big) + 2 + \frac{2}{1-\frac{2}{k}} \lceil \log_2 \frac{1}{g} \rceil,$$

because $\sum_{i=1}^{t} X_i \le \text{LIN}(I) \le \text{OPT}(I)$ and $t \le (\frac{\ln \text{SIZE}(I)}{\ln \frac{k}{2}} + 1)$ and $Y_i \le k\lceil \log_2 \frac{1}{g} \rceil$. By setting $k = 4$ and $g = \frac{1}{\text{SIZE}(I)} \ge \frac{1}{n}$, and noting that $\text{OPT}(T) \ge \text{SIZE}(I)$, we see that the total number of bins is at most $\text{OPT}(I) + O(\ln^2 \text{OPT}(I))$, proving theorem 13.5.1. □

13.6 Iterative Randomized Rounding: Steiner Trees

In this section, we present a recent application of the iterative rounding method augmented with randomized rounding for the classical undirected Steiner tree problem. The Steiner