Practical Approximation Algorithms for Separable Packing Linear Programs*

Feodor F. Dragan[1], Andrew B. Kahng[2], Ion I. Măndoiu[3], Sudhakar Muddu[4], and Alexander Zelikovsky[5]

[1] Department of Computer Science, Kent State University, Kent, OH 44242
dragan@cs.kent.edu

[2] Departments of Computer Science and Engineering, and of Electrical and Computer Engineering, UC San Diego, La Jolla, CA 92093-0114
abk@cs.ucsd.edu

[3] Department of Computer Science, UC Los Angeles, Los Angeles, CA 90095-1596
mandoiu@cc.gatech.edu

[4] Sanera Systems, Inc., Santa Clara, CA
muddu@sanera.net

[5] Department of Computer Science, Georgia State University, Atlanta, GA 30303
alexz@cs.gsu.edu

Abstract. We describe fully polynomial time approximation schemes for generalized multicommodity flow problems arising in VLSI applications such as Global Routing via Buffer Blocks (GRBB). We extend Fleischer's improvement [7] of Garg and Könemann [8] fully polynomial time approximation scheme for edge capacitated multicommodity flows to multiterminal multicommodity flows in graphs with capacities on vertices and subsets of vertices. In addition, our problem formulations observe upper bounds and parity constraints on the number of vertices on any source-to-sink path. Unlike previous works on the GRBB problem [5,17], our algorithms can take into account (i) multiterminal nets, (ii) simultaneous buffered routing and compaction, and (iii) buffer libraries. Our method outperforms existing algorithms for the problem and has been validated on top-level layouts extracted from a recent high-end microprocessor design.

1 Introduction

In this paper, we address the problem of how to perform buffering of global nets *given an existing buffer block plan.* We give integer linear program (ILP) formulations of the basic Global Routing via Buffer Blocks (GRBB) problem and its extensions to (i) multiterminal nets, (ii) simultaneous buffered routing and compaction, and (iii) buffer libraries. The fractional relaxations of these ILP's are *separable packing LP's* (SP LP) which are multiterminal multicommodity flows in graphs with capacities on vertices and subsets of vertices.

* This work was partially supported by Cadence Design Systems, Inc., the MARCO Gigascale Silicon Research Center and NSF Grant CCR-9988331.

© Springer-Verlag Berlin Heidelberg 2001

The main contribution of this paper is a practical algorithm for the GRBB problem and its extensions based on a fully polynomial time approximation scheme (FPTAS) for solving SP LPs. Prior to our work, heuristics based on solving fractional relaxations followed by randomized rounding have been applied to VLSI global routing [12,16,2,9,1] As noted in [11], the applicability of this approach is limited to problem instances of relatively small size by the prohibitive cost of solving exactly the fractional relaxation. We avoid this limitation by giving an FPTAS for SP LP's based on results in [8,7]. Computational experience with industrial benchmarks shows that our approach is practical and outperforms existing algorithms.

The rest of the paper is organized as follows. In Section 3 we formulate the GRBB problem and its extensions as integer linear programs. The fractional relaxation of these ILPs is a special type of packing LP which we refer to as separable packing LP. In Sections 4 we give a practical approximation algorithm, obtained by extending the ideas of Fleischer [7] for separable packing LPs; the details of the key subroutine for finding minimum-weight feasible Steiner trees are given in Section 5; the details of randomized rounding algorithms are in Section 6. In Section 7 we describe implementations of several GRBB heuristics and give the results of an experimental comparison of these heuristics on industrial test cases.

2 Global Buffering via Buffer Blocks

Process scaling in VLSI leads to an increasingly dominant effect of interconnect on high-end chip performance. Each top-level global net must undergo repeater or buffer (inverter) insertion to maintain signal integrity and reasonable signal delay [4]. It is estimated that up to 10^6 repeaters will be needed for the next generation on-chip interconnect. To isolate repeaters from circuit block implementations, a buffer block methodology is becoming increasingly popular. Two recent works by Cong, Kong and Pan [5] and Tang and Wong [17] give algorithms to solve the *buffer block planning* problem. Their buffer block planning formulation is roughly stated as follows: Given a placement of circuit blocks, and a set of 2-pin connections with *feasible regions* for buffer insertion, plan the location of *buffer blocks* within the available free space so as to route a maximum number of connections.

In this paper we address the problem of maximizing the number of routed nets for given buffer block locations and capacities, informally defined as follows.

Given:

- a planar region with rectangular obstacles;
- a set of nets in the region, each net having:
  - a non-negative importance (criticality) coefficient;
  - a single source and multiple sinks;

- for each sink:
    - a parity requirement and an upper-bound on the number of buffers on the path connecting it to the source;
- a set of buffer blocks, each with given capacity; and
- an interval $[L, U]$ specifying lower and upper bounds on the distance between buffers.

Global Routing via Buffer Blocks (GRBB) Problem: route a subset of the given nets, with maximum total importance, such that:

- the distance between the source of a route and its first repeater, between any two consecutive repeaters, respectively between the last repeater on a route and the route's sink, are all between L and U;
- the number of routing trees passing through any given buffer block does not exceed the block's capacity;
- the number of buffers on each source-sink path does not exceed the given upper bound and has the required parity; to meet the parity constraint two buffers of the same block can be used.

We also address the following extensions of the basic GRBB problem:

- **GRBB with Set Capacity Constraints.** The basic GRBB problem assumes predetermined capacities for all buffer blocks. In practice buffer blocks are placed in the space available after placing circuit blocks, and some of the circuit blocks can still be moved within certain limits (Figure 1). The *GRBB problem with set capacity constraints* captures this freedom by allowing constraints on the total capacity of arbitrary *sets* of buffer blocks.

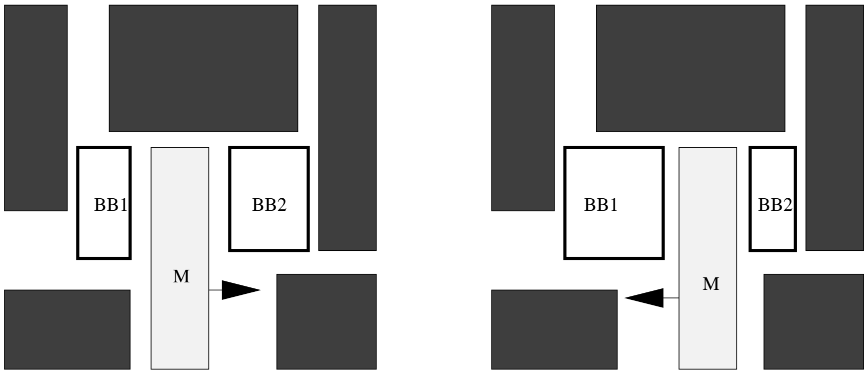

Fig. 1. Two buffer blocks BB1 and BB2 that share capacity: if the circuit block M moves to the right, then the capacity of buffer block BB1 is increasing while the capacity of buffer block BB2 is decreasing. In this example it is the sum of capacities of BB1 and BB2, rather than their individual capacities, that is bounded.

– **GRBB with Buffer Library.** To achieve better use of area and power resources, multiple buffer types can be used. The *GRBB problem with buffer library* optimally distributes the available buffer block capacity between given buffer types and simultaneously finds optimum buffered routings.

3 Integer Linear Program Formulations

Throughout this paper we let $N_k = (s_k; t_k^1, \ldots, t_k^{q_k})$, $k = 1, \ldots, K$, denote the nets to be routed; s_k is the *source*, and $t_k^1, \ldots, t_k^{q_k}$ are the *sinks* of net N_k. We denote by $g_k \geq 1$ the importance (criticality) coefficient of net N_k, and by $a_k^i \in \{\text{even, odd}\}$ and $l_k^i \geq 0$ the prescribed *parity*, respectively *upper bound*, on the number of buffers on the path between source s_k and sink t_k^i. We also let $S = \{s_1, \ldots, s_K\}$ and $S' = \{t_1^1, \ldots, t_1^{q_1}, \ldots, t_K^1, \ldots, t_K^{q_K}\}$ denote the set of sources, respectively of sinks, and $R = \{r_1, \ldots, r_n\}$ denote the given set of *buffer blocks*. For each buffer block r_i, we let $c(r_i)$ denote its *capacity*, i.e., the maximum number of buffers that can be inserted in r_i.

A *routing graph* for nets N_k, $k = 1, \ldots, K$, is an undirected graph $G = (V, E)$ such that $S \cup S' \subseteq V$. The set of vertices of G other than sources and sinks, $V \setminus (S \cup S')$, is denoted by V'. All vertices in a routing graph are associated to locations on the chip, including vertices of V' which are associated with buffer block locations. We require that the rectilinear distance with obstacles between two vertices connected by an edge in the routing graph be either between L and U or 0 (this last case corresponds to using two buffers in the same buffer block). Thus, inserting a buffer at each Steiner point ensures that every Steiner tree in the routing graph satisfies the given L/U bounds. A *feasible Steiner tree* for net N_k is a Steiner tree T_k connecting terminals $s_k, t_k^1, \ldots, t_k^{q_k}$ such that, for every $i = 1, \ldots, q_k$, the path of T_k connecting s_k to t_k^i has length at most l_k^i and parity a_k^i. We denote the set of all feasible Steiner trees for net N_k by $\mathcal{T}_k$, and let $\mathcal{T} = \bigcup_{k=1}^K \mathcal{T}_k$.

For the GRBB problem, the routing graph $G = (V, E)$ has $V = S \cup S' \cup \{r', r'' \mid r \in R\}$ (there are two vertices corresponding to each buffer block to allow for feasible Steiner trees that meet the parity constraints by using two buffers in the same buffer block) and $E = \{(r', r'') \mid r \in R\}$ $\cup\{(x, y) \mid x, y \in V,\ L \leq d(x, y) \leq U\}$, where, $d(x, y)$ is the rectilinear distance with obstacles between points x and y. Given importance coefficients $g_k = g(N_k)$ for each net N_k, let $g(T) = g_k$ for each tree $T \in \mathcal{T}_k$, $k = 1, \ldots, K$. The GRBB problem is then equivalent to the following integer linear program:

$$\begin{array}{lll}
\textbf{maximize} & \sum_{T \in \mathcal{T}} g(T) f_T & \text{(GRBB ILP)} \\
\textbf{subject to} & & \\
& \sum_{T \in \mathcal{T}} \pi_T(v) f_T \leq 1, & \forall v \in S \cup S' \\
& \sum_{T \in \mathcal{T}} (\pi_T(r') + \pi_T(r'')) f_T \leq c(r), & \forall r \in R \\
& f_T \in \{0, 1\}, & \forall T \in \mathcal{T}
\end{array}$$

where $\pi_T(v)$ is 1 if $v \in T$ and 0 otherwise.

The GRBB ILP, as well as the ILP formulations for GRBB with set constraints and buffer library (which we omit from this extended abstract) are captured by the following common generalization, referred to as the *separable packing ILP* (SP ILP):

$$\begin{array}{lll}
\textbf{maximize} & \sum_{T\in\mathcal{T}} g(T) f_T & \text{(SP ILP)} \\
\textbf{subject to} & & \\
 & \sum_{T\in\mathcal{T}} \left(\sum_{v\in X} \pi_T(v)s(v)\right) f_T \le c(X), & \forall X \in \mathcal{V} \\
 & f_T \in \{0,1\}, & \forall T \in \mathcal{T}
\end{array}$$

for given

- arbitrary sets $\mathcal{T}_k$ of Steiner trees for each net N_k;
- family $\mathcal{V}$ of subsets of V such that $\{v\} \in \mathcal{V}$ for every $v \in S \cup S'$;
- "size" function $s : V \to R_+$ such that $s(v) = 1$ for every $v \in S \cup S'$; and
- "set-capacity" function $c : \mathcal{V} \to Z_+$ such that $c(\{v\}) = 1$ for every $v \in S \cup S'$.

Our two-step approach to the GRBB problem and its extensions is to first solve the fractional relaxations obtained by replacing integrality constraints $f_T \in \{0,1\}$ with $f_T \ge 0$, and then use randomized rounding to get integer solutions. In next section we give an algorithm for approximating the fractional relaxation of the SP ILP. The algorithm relies on a subroutine for finding minimum weight feasible Steiner trees, the details of this subroutine are given in Section 5.

4 Approximating the SP ILP Relaxation

The fractional relaxation of the SP ILP can be solved exactly in polynomial time using, e.g., the ellipsoid algorithm. However, exact algorithms are highly impractical. The SP LP can be efficiently approximated within any desired accuracy using Garg and Könemann's approximation scheme for packing LPs [8]. The main step of their algorithm is computing the minimum weight column of the LP. For the special case of edge-capacitated multicommodity flow LPs, Fleischer [7] gave a significantly faster algorithm by computing in each step the minimum weight column only among columns corresponding to a single commodity. Below we generalize Fleisher's idea to separable packing LPs by partitioning the columns into groups corresponding to the nets.

4.1 The Algorithm

Our algorithm simultaneously finds feasible solutions to the SP LP and its dual. The dual LP asks for an assignment of non-negative weights $w(X)$ to every $X \in \mathcal{V}$ such that the weight of every tree $T \in \mathcal{T}$ is at least 1, where the weight of T is defined by $weight(T) = \frac{1}{g(T)} \sum_{X\in\mathcal{V}} \pi_T(X)w(X)$ and $\pi_T(X) = \sum_{v\in X} \pi_T(v)s(v)$:

Input: Nets $N_1, \ldots, N_K$, coefficients $g_1, \ldots, g_K$, routing graph $G = (V, E)$, family $\mathcal{V}$ of subsets of V, capacities $c(X)$, $X \in \mathcal{V}$, and weights $s(v)$, $v \in V$
Output: SP LP solution f_T, $T \in \mathcal{T}$

```
For every $T \in \mathcal{T}$, $f_T \leftarrow 0$
For every $X \in \mathcal{V}$, $w(X) \leftarrow \delta$
$\bar{\alpha} \leftarrow \delta/\Gamma$
For $i = 1$ to $t = \left\lfloor \log_{1+\epsilon} \frac{(1+\epsilon)\Gamma}{\delta} \right\rfloor$ do
 For $k = 1$ to $K$ do
     Find a minimum weight feasible Steiner tree $T$ in $\mathcal{T}_k$
     While $weight(T) < min\{1, (1+\epsilon)\bar{\alpha}\}$ do
         $f_T \leftarrow f_T + 1$
         For all $X \in \mathcal{V}$, $w(X) \leftarrow w(X)(1 + \epsilon\pi_T(X)/c(X))$
         Find a minimum weight feasible Steiner tree $T$ in $\mathcal{T}_k$
     End while
 End for on $k$
 $\bar{\alpha} \leftarrow (1+\epsilon)\bar{\alpha}$
End for on $i$
For every $T \in \mathcal{T}$, $f_T \leftarrow \frac{f_T}{\log_{1+\epsilon} \frac{(1+\epsilon)\Gamma}{\delta}}$
Output $f_T$, $T \in \mathcal{T}$
```

Fig. 2. The algorithm for finding approximate solutions to the SP LP.

$$\begin{array}{lll} \textbf{maximize} & \sum_{X \in \mathcal{V}} w(X)c(X) & \text{(SP LP Dual)} \\ \textbf{subject to} & & \\ & \frac{1}{g(T)} \sum_{X \in \mathcal{V}} \pi_T(X) w(X) \geq 1, \quad \forall T \in \mathcal{T} & \\ & w(X) \geq 0, \quad \forall X \in \mathcal{V} & \end{array}$$

In the following we assume that $\min\{g_k : k = 1, \ldots, K\} = 1$ (this can be easily achieved by scaling) and denote $\max\{g_k : k = 1, \ldots, K\}$ by Γ.

The algorithm (Figure 2) starts with weights $w(X) = \delta$ for every $X \in \mathcal{V}$, where δ is an appropriately chosen constant, and with a SP LP solution $f \equiv 0$. While there is a feasible tree whose weight is less than 1, the algorithm selects such a tree T and increments f_T by 1. This increase will likely violate the capacity constraints for some of the sets in $\mathcal{V}$; feasibility is achieved at the end of the algorithm by uniformly scaling down all f_T's. Whenever f_T is incremented, the algorithm also updates each weight $w(X)$ by multiplying it with $(1 + \epsilon\pi_T(X)/c(X))$, for a fixed ϵ.

According to the Garg and Könemann's approximation algorithm [8] each iteration must increment the variable f_T corresponding to a tree with minimum weight among all trees in $\mathcal{T}$. Finding this tree essentially requires K minimum-weight feasible Steiner tree computations, one for each net N_k. We reduce the total number of minimum-weight feasible Steiner tree computations during the algorithm by extending a speed-up idea due to Fleischer [7]. Instead of always finding the minimum-weight tree in $\mathcal{T}$, the idea is to settle for trees with weight within a factor of $(1+\epsilon)$ of the minimum. As shown in next section, the faster algorithm still leads to an approximation guarantee similar to that of Garg and Könemann.

4.2 Runtime and Performance Analysis

In each iteration the algorithm cycles through all nets. For each net, the algorithm repeatedly computes minimum-weight feasible Steiner tree until the weight becomes larger than $(1+\epsilon)$ times a lower-bound $\bar{\alpha}$ on the overall minimum weight, $\min\{weight(T) : T \in \mathcal{T}\}$. The lower-bound $\bar{\alpha}$ is initially set to δ/Γ, and then multiplied by a factor of $(1+\epsilon)$ from one iteration to another (note that no tree in $\mathcal{T}$ has weight smaller than $(1+\epsilon)\bar{\alpha}$ at the end of an iteration, so $(1+\epsilon)\bar{\alpha}$ is a valid lower-bound for the next iteration).

The scheme used for updating $\bar{\alpha}$ fully determines the number of iterations in the outer loop of the algorithm. Since $\bar{\alpha} = \delta/\Gamma$ in the first iteration and at most $(1+\epsilon)$ in the last one, it follows that the number of iterations is $\left\lfloor \log_{1+\epsilon} \frac{(1+\epsilon)\Gamma}{\delta} \right\rfloor$. The following lemma gives an upper-bound on the runtime of the algorithm.

Lemma 1. *Overall, the algorithm in Figure 2 requires* $O\left(K \log_{1+\epsilon} \frac{(1+\epsilon)\Gamma}{\delta}\right)$ *minimum-weight feasible Steiner tree computations.*

Proof. First, note that the number of minimum-weight feasible Steiner tree computations that do not contribute to the final fractional solution is $K \left\lfloor \log_{1+\epsilon} \frac{(1+\epsilon)\Gamma}{\delta} \right\rfloor$. Indeed, in each iteration, and for each net N_k, there is exactly one minimum-weight feasible Steiner tree computation revealing that $min_{T \in \mathcal{T}_k} weight(T) \geq (1+\epsilon)\bar{\alpha}$, all other computations trigger the incrementation of some f_T.

We claim that the number of minimum-weight Steiner trees that lead to variable incrementations is at most $K \log_{1+\epsilon} \frac{(1+\epsilon)\Gamma}{\delta}$. To see this, note that the weight of the set $\{s_k\} \in \mathcal{V}$ is updated whenever a variable f_T, $T \in \mathcal{T}_k$, is incremented. Moreover, $w(\{s_k\})$ is last updated when incrementing f_T for a tree $T \in \mathcal{T}_k$ of weight less than one. Thus, before the last update, $w(\{s_k\}) \leq \Gamma \cdot weight(T) < \Gamma$. Since $\pi_T(\{s_k\}) = c(\{s_k\}) = 1$, the weight of $\{s_k\}$ is multiplied by a factor of $1+\epsilon$ in each update, including the last one. This implies that the final value of $w(\{s_k\})$ is at most $(1+\epsilon)\Gamma$. Recalling that $w(\{s_k\})$ is initially set to δ, this gives that the number of updates to $w(\{s_k\})$ is at most $\log_{1+\epsilon} \frac{(1+\epsilon)\Gamma}{\delta}$. The lemma follows by summing this upper-bound over all nets. □

We now show that, for an appropriate value of the parameter δ, the algorithm finds a feasible solution close to optimum.

Theorem 1. *For every $\epsilon < 0.15$, the algorithm in Figure 2 computes a feasible solution to the SP LP within a factor of $1/(1+4\epsilon)$ of optimum by choosing $\delta = (1+\epsilon)\Gamma((1+\epsilon)L\Gamma)^{-\frac{1}{\epsilon}}$; the runtime of the algorithm for this value of δ is $O\left(\frac{1}{\epsilon^2}K(\log L + \log \Gamma)T_{tree}\right)$. Here, L is the maximum number of vertices in a feasible tree, and T_{tree} is the time required to compute the minimum weight feasible Steiner tree for a net.*

Proof. Our proof is an adaptation of the proofs of Garg and Könemann [8] and Fleischer [7]. We omit the proof that the solution found by the algorithm is feasible. To establish the approximation guarantee, we show that the solution computed by the algorithm is within a factor of $1/(1+4\epsilon)$ of the optimum objective value, β, of the dual LP. Let $\alpha(w)$ be the weight of a minimum weight tree from $\mathcal{T}$ with respect to weight function $w : \mathcal{V} \to R_+$, and let $D(w) = \sum_{X\in\mathcal{V}} w(X)c(X)$. A standard scaling argument shows that the dual LP is equivalent to finding a weight function w such that $D(w)/\alpha(w)$ is minimum, and that $\beta = \min_w\{D(w)/\alpha(w)\}$.

For every $X \in \mathcal{V}$, let $w_i(X)$ be the weight of set X at the end of the ith iteration and $w_0(X) = \delta$ be the initial weight of set X. For brevity, we will denote $\alpha(w_i)$ and $D(w_i)$ by $\alpha(i)$ and $D(i)$, respectively. Furthermore, let f_T^i be the value of f_T at the end of ith iteration, and $h_i = \sum_{T\in\mathcal{T}} g(T)f_T^i$ be the objective value of the SP LP at the end of this iteration.

When the algorithm increments f_T by one unit, each weight $w(X)$ is increased by $(\epsilon\pi_T(X)w(X)/c(X)$. Thus, the incrementation of f_T increases $D(w)$ by

$$\epsilon \sum_{X\in\mathcal{V}} \pi_T(X)w(X) = \epsilon\; weight(T)g(T)$$

If this update takes place in the ith iteration, then $weight(T) \le (1+\epsilon)\alpha(i-1)$. Adding this over all f_T's incremented in ith iteration gives

$$D(i) - D(i-1) \le \epsilon(1+\epsilon)\alpha(i-1)(h_i - h_{i-1})$$

which implies that

$$D(i) - D(0) \le \epsilon(1+\epsilon)\sum_{j=1}^{i}\alpha(j-1)(h_j - h_{j-1})$$

Consider the weight function $w_i - w_0$, and notice that $D(w_i - w_0) = D(i) - D(0)$. Since the minimum weight tree w.r.t. weight function $w_i - w_0$ has a weight of at most $\alpha(w_i - w_0) + L\delta$ w.r.t. w_i, $\alpha(i) \le \alpha(w_i - w_0) + L\delta$. Hence, if $\alpha(i) - L\delta > 0$, then

$$\beta \le \frac{D(w_i - w_0)}{\alpha(w_i - w_0)} \le \frac{D(i) - D(0)}{\alpha(i) - L\delta} \le \frac{\epsilon(1+\epsilon)\sum_{j=1}^{i}\alpha(j-1)(h_j - h_{j-1})}{\alpha(i) - L\delta}$$

Thus, in any case (when $\alpha(i) - L\delta \le 0$ this follows trivially) we have

$$\alpha(i) \le L\delta + \frac{\epsilon(1+\epsilon)}{\beta}\sum_{j=1}^{i}\alpha(j-1)(h_j - h_{j-1})$$

Note that, for each fixed i, the right-hand side of last inequality is maximized by setting $\alpha(j)$ to its maximum possible value, say $\alpha'(j)$, for every $0 \leq j < i$. Then, the maximum value of $\alpha(i)$ is

$$\begin{aligned}
\alpha'(i) &= L\delta \;+\; \frac{\epsilon(1+\epsilon)}{\beta}\sum_{j=1}^{i-1}\alpha'(j-1)(h_j - h_{j-1}) \;+\; \frac{\epsilon(1+\epsilon)}{\beta}\alpha'(i-1)(h_i - h_{i-1}) \\
&= \alpha'(i-1)\left(1 \;+\; \frac{\epsilon(1+\epsilon)}{\beta}(h_i - h_{i-1})\right) \\
&\leq \alpha'(i-1)e^{\frac{\epsilon(1+\epsilon)}{\beta}(h_i - h_{i-1})}
\end{aligned}$$

where the last inequality uses that $1 + x \leq e^x$ for every $x \geq 0$. Using that $\alpha'(0) = L\delta$, this gives

$$\alpha(i) \leq L\delta e^{\frac{\epsilon(1+\epsilon)}{\beta}h_i}$$

Let t be the last iteration of the algorithm. Since $\alpha(t) \geq 1$,

$$1 \leq L\delta e^{\frac{\epsilon(1+\epsilon)}{\beta}h_t}$$

and thus

$$\frac{\beta}{h_t} \leq \frac{\epsilon(1+\epsilon)}{\ln(L\delta)^{-1}}$$

Let $\gamma = \frac{\beta}{h_t}\log_{1+\epsilon}\frac{(1+\epsilon)\Gamma}{\delta}$ be the ratio between the optimum dual objective value and the objective value of the SP LP solution produced by the algorithm. By substituting the previous bound on β/h_t we obtain

$$\gamma \leq \frac{\epsilon(1+\epsilon)\log_{1+\epsilon}\frac{(1+\epsilon)\Gamma}{\delta}}{\ln(L\delta)^{-1}} = \frac{\epsilon(1+\epsilon)\ln\frac{(1+\epsilon)\Gamma}{\delta}}{\ln(1+\epsilon)\ln(L\delta)^{-1}}$$

For $\delta = (1+\epsilon)\Gamma((1+\epsilon)L\Gamma)^{-\frac{1}{\epsilon}}$,

$$\frac{\ln\frac{(1+\epsilon)\Gamma}{\delta}}{\ln(L\delta)^{-1}} = \frac{\ln\left((1+\epsilon)L\Gamma\right)^{\frac{1}{\epsilon}}}{\ln\left((1+\epsilon)L\Gamma\right)^{-1+\frac{1}{\epsilon}}} = \frac{\frac{1}{\epsilon}\ln\left(1+\epsilon\right)L\Gamma)}{\frac{1-\epsilon}{\epsilon}\ln\left(1+\epsilon\right)L\Gamma)} = \frac{1}{1-\epsilon}$$

and thus

$$\gamma \leq \frac{\epsilon(1+\epsilon)}{(1-\epsilon)\ln(1+\epsilon)} \leq \frac{\epsilon(1+\epsilon)}{(1-\epsilon)(\epsilon-\epsilon^2/2)} \leq \frac{(1+\epsilon)}{(1-\epsilon)^2}$$

Here we use the fact that $\ln(1+\epsilon) \geq \epsilon - \epsilon^2/2$ (by Taylor series expansion of $\ln(1+\epsilon)$ around the origin). The proof of the approximation guarantee is completed by observing that $(1+\epsilon)/(1-\epsilon)^2 \leq (1+4\epsilon)$ for every $\epsilon < 0.15$. The runtime follows by substituting δ in the bound given by Lemma 1. □

5 Computing Minimum-Weight Feasible Steiner Trees

The key subroutine of the approximation algorithm given in the previous section is to compute, for a fixed k and given weights $w(X)$, $X \in \mathcal{V}$, a feasible tree $T \in \mathcal{T}_k$ minimizing $weight(T) = \frac{1}{g(T)} \sum_{X \in \mathcal{V}} \pi_T(X) w(X)$. Define a weight function w' on the vertices of the routing graph $G = (V, E)$ by setting $w'(v) = \frac{1}{g(T)} \sum_{v \in X \in \mathcal{V}} w(X)$, and let $w'(T) = \sum_{v \in V(T)} w'(v)$ be the total vertex weight w.r.t. w' of T. Then $weight(T) = w'(T)$, and the problem reduces to finding a tree $T \in \mathcal{T}_k$ with minimum total vertex weight w.r.t. w'.

Recall that for the GRBB problem and its extensions, $\mathcal{T}_k$ contains all Steiner trees connecting the source s_k with the sinks $t_k^1, \ldots, t_k^{q_k}$ such that the number of intermediate vertices on each tree path between s_k and t_k^i has the parity specified by a_k^i and does not exceed l_k^i. In this case we can further reduce the problem of finding the tree $T \in \mathcal{T}_k$ minimizing $w'(T)$ to the *minimum-cost directed rooted Steiner tree* (DRST) problem in a directed acyclic graph. Unfortunately, the minimum-cost DRST problem is NP-hard, and the fact that D_k is acyclic does not help since there is a simple reduction for this problem from arbitrary directed graphs to acyclic graphs. As far as we know, the best result for the DRST problem, due to Charikar et al. [3], gives $O(\log^2 q_k)$-approximate solutions in quasi-polynomial time $O(n^{3 \log q_k})$. Note, on the other hand, that the minimum-cost DRST can be found in polynomial time for small nets (e.g., in time $O(n^{M-1})$ for nets with at most M sinks, for $M = 2, 3, 4$); most of the nets in industrial VLSI designs fall into this category [10]. For nets of small size, Theorem 1 immediately gives:

Corollary 1. *If the maximum net size is $M \leq 4$, the algorithm in Figure 2 finds, for every $\epsilon < 0.15$, a feasible solution to the SP LP within a factor of $1/(1 + 4\epsilon)$ of optimum in time $O\left(\frac{1}{\epsilon^2} K n^{M-1} (\log n + \log \Gamma)\right)$.*

We have implemented both heuristics that use approximate DRSTs instead of optimum DRSTs and heuristics that decompose larger nets into nets with 2-4 pins before applying the algorithm in Figure 2; results of experiments comparing these approaches are reported in Section 7.

6 Rounding Fractional SP LP Solutions

In the previous two sections we presented an algorithm for computing near-optimal solutions to the SP LP. In this section we give two algorithms based on the randomized rounding technique of Raghavan and Thomson [14] (see also [11]) for converting these solutions to integer SP ILP solutions.

The first algorithm is to route net N_k with probability equal to $f_k = \sum_{T \in \mathcal{T}_k} f_T$ by picking, for selected nets, one of the trees $T \in \mathcal{T}_k$ with probability f_T / f_k. A drawback of this algorithm is that it requires the explicit representation of trees $T \in \mathcal{T}$ with $f(T) \neq 0$. Although the approximate SP LP algorithm produces a polynomial number of trees with non-zero f_T, storing all such trees

```
Input: Net- and edge-cumulated $f_T$ values, $f_k = \sum_{T \in \mathcal{T}_k} f_T$ and
$f_k(e) = \sum_{T \in \mathcal{T}_k:\ e \in E(T)} f_T$, $k = 1, \ldots, K$, $e \in E(D_k)$
Output: Routed trees $T_k \in \mathcal{T}_k$
-----------------------------------------------------------------

For each $k = 1, \ldots, K$, select net $N_k$ with probability $f_k$
Route each selected net $N_k$ as follows:
 $T_k \leftarrow \{s_k\}$
 For each sink $t_k^i$ in $N_k$ do
     $P \leftarrow \emptyset$;    $v \leftarrow t_k^i$
     While $v \notin T_k$ do
         Pick arc $(u, v)$ with probability $\frac{f_k(u,v)}{\sum_{(w,v) \in E} f_k(w,v)}$
         $P \leftarrow P \cup \{(u, v)\}$;    $v \leftarrow u$
     End while
     $T_k \leftarrow T_k \cup P$
 End for
```

Fig. 3. The random walk based rounding algorithm.

is infeasible for large problem instances. Our second rounding algorithm (Figure 3) takes as input the net- and edge-cumulated f_T values, $f_k = \sum_{T \in \mathcal{T}_k} f_T$, respectively $f_k(e) = \sum_{T \in \mathcal{T}_k:\ e \in E(T)} f_T$, thus using only $O(K|E|)$ space.

As the first rounding algorithm, the algorithm in Figure 3 routes each net N_k with a probability of $f_k = \sum_{T \in \mathcal{T}_k} f_T$. The difference is in how each chosen net is routed: to route net N_k, the algorithm performs *backward random walks* from each sink of N_k until reaching either the source of N_k or a vertex already connected to the source. The random walks are performed in the directed acyclic graphs used for DRST computation, with probabilities given by the normalized $f_k(e)$ values.

On the average, the total importance of the nets routed by each of the two algorithm is $\sum_{k=1}^{K} g_k f_k = \sum_{T \in \mathcal{T}} g(T) f_T$. By Theorem 1, this is within a factor of $1/(1 + 4\epsilon)$ of the optimum SP LP solution, which in turn is an upper-bound on the optimum SP ILP solution. Ensuring that no set capacity is exceeded can be accomplished in two ways. One approach is to solve the SP LP with set capacities scaled down by a small factor which guarantees that the rounded solution meets the *original* capacities with very high probability (see [11]). A more practical approach, extending the so-called *greedy-deletion algorithm* in [6] to multiterminal nets, is to repeatedly drop routed paths passing through over-used sets until feasibility is achieved.

7 Experimental Results

We have implemented four greedy algorithms for the GRBB problem; all four greedy algorithms route nets sequentially. For a given net, the algorithms start

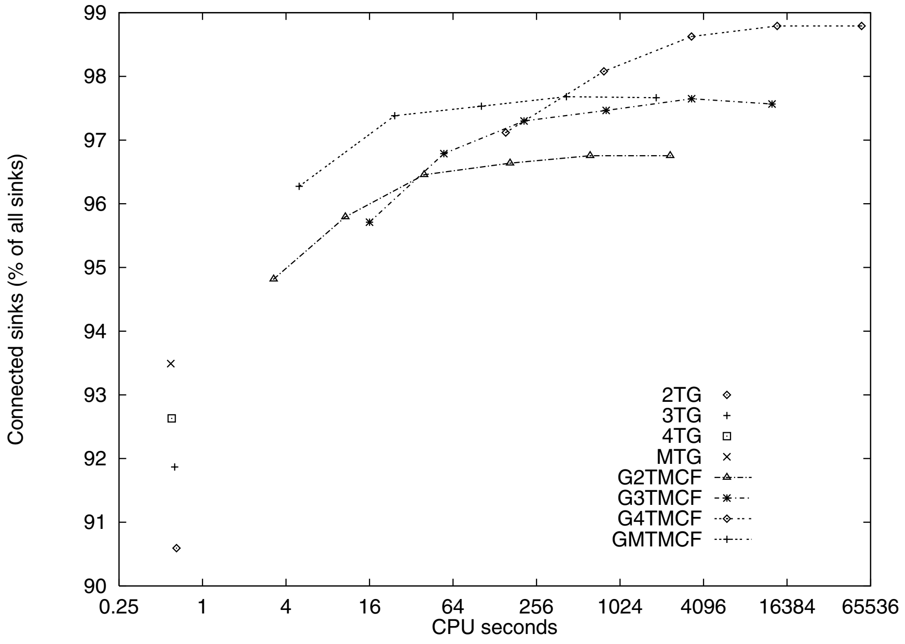

Fig. 4. Percent of sinks connected vs. CPU time.

with a tree containing only the net's source, then iteratively add shortest paths from each sink to the already constructed tree. The only difference is in whether or not net decomposition is used, and in the size of the decomposed nets. The first three algorithms—referred to as 2TG, 3TG, and 4TG, respectively—start by decomposing larger multiterminal nets into 2-, 3-, respectively 4-pin nets. The fourth algorithm, MTG, works on the original (undecomposed) nets.

We have also implemented four algorithms that approximate the fractional solution to the SP LP corresponding to GRBB problem (which generalizes the node-capacitated multiterminal multicommodity flow problem) and then apply randomized rounding. The first three algorithms (2TMCF, 3TMCF, and 4TMCF) decompose larger nets into 2-, 3-, respectively 4-pin nets then call the algorithm in Figure 2 with exact DRST computations. The fourth algorithm, MTMCF, works on the original (undecomposed) nets, using shortest-path trees as approximate DRSTs in the SP LP approximation algorithm.

Figure 4 plots the solution quality versus the CPU time (on a 195MHz SGI Origin 2000) of each implemented algorithm. The test cases used in our experiments were extracted from the next-generation (as of January 2000) microprocessor chip at SGI. The results clearly demonstrate the high quality of solutions obtained by rounding the approximate SP LP solutions. The MTMCF algorithm proves to be the best among all algorithms when the time budget is limited, providing significant improvements over greedy algorithms without undue runtime penalty. However, the best convergence to the optimum is achieved by 4TMCF, which dominates all other algorithms when high time budgets are allowed.

References

1. C. Albrecht, "Provably good global routing by a new approximation algorithm for multicommodity flow", *Proc. ISPD*, 2000.
2. R.C. Carden and C.-K. Cheng, "A global router using an efficient approximate multicommodity multiterminal flow algorithm", *Proc. DAC*, 1991, pp. 316–321.
3. M. Charikar, C. Chekuri, T. Cheung, Z. Dai, A. Goel, and S. Cheung, "Approximation algorithms for directed Steiner problems", *J. Algorithms*, 33 (1999), pp. 73–91.
4. J. Cong, L. He, C.-K. Koh and P.H. Madden, "Performance optimization of VLSI interconnect layout", *Integration* 21 (1996), pp. 1–94.
5. J. Cong, T. Kong and D.Z. Pan, "Buffer block planning for interconnect-driven floorplanning", *Proc. ICCAD*, 1999, pp. 358–363.
6. F.F. Dragan, A.B. Kahng, I.I. Măndoiu, S. Muddu and A. Zelikovsky, "Provably good global buffering using an available buffer block plan", *Proc. ICCAD*, 2000, pp. 104–109.
7. L.K. Fleischer, "Approximating fractional multicommodity flow independent of the number of commodities", *Proc. 40th Annual Symposium on Foundations of Computer Science*, 1999, pp. 24–31.
8. N. Garg and J. Könemann, "Faster and simpler algorithms for multicommodity flow and other fractional packing problems", *Proc. 39th Annual Symposium on Foundations of Computer Science*, 1998, pp. 300–309.
9. J. Huang, X.-L. Hong, C.-K. Cheng and E.S. Kuh, "An efficient timing-driven global routing algorithm", *Proc. DAC*, 1993, pp. 596-600.
10. A.B. Kahng and G. Robins. *On Optimal Interconnections for VLSI*, Kluwer Academic Publishers, Norwell, Massachusetts, 1995.
11. R. Motwani, J. Naor, and P. Raghavan, "Randomized approximation algorithms in combinatorial optimization", In *Approximation algorithms for NP-hard problems* (Boston, MA, 1997), D. Hochbaum, Ed., PWS Publishing, pp. 144–191.
12. A.P.-C. Ng, P. Raghavan, and C.D. Thomson, "Experimental results for a linear program global router". *Computers and Artificial Intelligence*, 6 (1987), pp. 229–242.
13. C.A. Phillips, "The network inhibition problem", *Proc. 25th Annual ACM Symposium on Theory of Computing*, 1993, pp. 776–785.
14. P. Raghavan and C.D. Thomson, "Randomized rounding", *Combinatorica*, 7 (1987), pp. 365–374.
15. P. Raghavan and C.D. Thomson, "Multiterminal Global Routing: A Deterministic Approximation Scheme", *Algorithmica*, 6 (1991), pp. 73–82.
16. E. Shragowitz and S. Keel, "A global router based on a multicommodity flow model", *Integration*, 5 (1987), pp. 3–16.
17. X. Tang and D.F. Wong, "Planning buffer locations by network flows", *Proc. ISPD*, 2000.