Provably Good Global Buffering by Generalized Multiterminal Multicommodity Flow Approximation

Feodor F. Dragan, Andrew B. Kahng, Ion I. Măndoiu, Sudhakar Muddu, and Alexander Zelikovsky

***Abstract*—To implement high-performance global interconnect without impacting the placement and performance of existing blocks, the use of buffer blocks is becoming increasingly popular in structured-custom and block-based application specified integrated circuit methodologies. Recent works by Cong *et al.* (1999) and Tang and Wong (2000) give algorithms to solve the *buffer block planning* problem. In this paper, we address the problem of how to perform the buffering of global multiterminal nets *given an existing buffer block plan*. We give provably good and heuristic algorithms for this problem based on a recent approach of Garg and Könemann (1998) and Fleischer (1999) [see also Albrecht (2000)]. Our method routes connections using available buffer blocks, such that required upper and lower bounds on buffer intervals are satisfied. In addition, our algorithms allow more than one buffer to be inserted into any given connection and observe upper bounds and parity constraints on the number of buffers per connection. Most importantly, and unlike previous works on the problem [Cong *et al.* (1999); Tang and Wong (2000)], we take into account: 1) multiterminal nets; 2) multiple routing layers; 3) simultaneous buffered routing and compaction; and 4) buffer libraries. Our method outperforms existing algorithms for the problem [see Cong *et al.* (1999)], based on two-pin decompositions of the nets, and has been validated on top-level layouts extracted from a recent high-end microprocessor design.**

***Index Terms*—Approximation algorithms, buffer insertion, buffer planning, global routing, multicommodity flow.**

I. Introduction

VERY LARGE scale integration (VLSI) process scaling leads to an increasingly dominant effect of interconnect on high-end chip performance. Each top-level global net must undergo repeater insertion (among other optimizations; see [4], [14], and [17]) to maintain signal integrity and reasonable signal delay.[1] Estimates of the need for repeater insertion range up to 10^6 repeaters for top-level on-chip interconnect when we reach the 50-nm technology node. These repeaters are large (anywhere from $40\times$ to $200\times$ minimum inverter size), affect global routing congestion, can entail nonstandard cell height and special power routing requirements, and can act as noise sources. In block- or reuse-based methodology, designers seek to isolate repeaters for global interconnect from individual block implementations.

For these reasons, a *buffer block* methodology has become increasingly popular in structured-custom and block-based application specified integrated circuit (ASIC) methodologies. Two recent works by Cong *et al.* [5] and Tang and Wong [22] 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 two-pin connections with *feasible regions* for buffer insertion,[2] 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 how to perform buffering of global nets *given an existing buffer block plan*. (Hence, our work is compatible with and complements the methods in [5] and [22].) We give a provably good algorithm based on a recent approach of Garg and Könemann [10] and Fleischer [9]. Our method routes the nets using the available buffer blocks, such that required upper and lower bounds on repeater intervals—as well as length upper bounds per connection—are satisfied.[3] Our algorithm allows more than one buffer to be inserted into any given connection and observes upper bounds on the number of buffers per connection. In addition, our algorithm observes *repeater parity constraints*, i.e., it will choose the number of inverters in any routing path according to the source and destination signal parity. The authors of [5] and [22] assumed that global nets have been already decomposed into two-pin connections; unlike these works our model takes into account *multiterminal nets*.

Our basic problem is informally defined as follows.

Given:

- a planar region with rectangular obstacles;

Manuscript received March 8, 2001. This paper was recommended by Associate Editor T. Yoshimura. This work was supported in part by Cadence Design Systems, Inc., by the MARCO Gigascale Silicon Research Center, by the National Science Foundation under Grant CCR-9988331, by the Moldovan Research and Development Association (MRDA) under Award MM2–3018, by the U.S. Civilian Research and Development Foundation for the Independent States of the Former Soviet Union (CRDF), and by the State of Georgia's Yamacraw Initiative.

F. F. Dragan is with the Department of Mathematics and Computer Science, Kent State University, Kent, OH 44242 USA (e-mail: dragan@mcs.kent.edu).

A. B. Kahng is with the Department of Computer Science and Engineering and Department of Electrical and Computer Engineering, University of California San Diego, La Jolla, CA 92093-0114 USA (e-mail: abk@ucsd.edu).

I. I. Măndoiu is with the Department of Computer Science and Engineering, University of California San Diego, La Jolla, CA 92093-0114 USA (e-mail: mandoiu@cs.ucsd.edu).

S. Muddu is with Sanera Systems, Inc., Sunnyvale, CA 94085 USA (e-mail: muddu@sanera.net).

A. Zelikovsky is with the Department of Computer Science, Georgia State University, Atlanta, GA 30303 USA (e-mail: alexz@cs.gsu.edu).

Publisher Item Identifier S 0278-0070(02)01787-6.

[1]Following the literature, we will use the terms *buffer* and *repeater* fairly interchangeably. When we need to be more precise: a repeater can be implemented as either an inverter or as a buffer (= two colocated inverters).

[2]In [22] only a single buffer per connection is allowed.

[3]For example, global repeater rules for a high-end microprocessor design in 0.25 μm CMOS [13] require repeater intervals of at most 4500 μm. The number of buffers needed for a given connection depends strongly on the length of the connection; as noted in [13], the repeater interval is not only required for delay reduction, but also for crosstalk noise immunity and edge slewtime control.

0278-0070/02$17.00 © 2002 IEEE

- a set of nets in the region, each net having:
  - — a single source and multiple sinks;
  - — a nonnegative importance (criticality) coefficient;
- each sink having:
  - — a parity requirement, which specifies the required parity of the number of buffers (inverters) on the path connecting it to the source;
  - — a timing-driven requirement, which specifies the maximum number of buffers on the path to the source;
- a set of buffer blocks, each with given capacity;
- 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 trees passing through any given buffer block does not exceed the block's capacity;
- the number of buffers on each source-to-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.

If possible, the optimum solution to the GRBB problem simultaneously routes all the nets. Otherwise, it maximizes the sum of the importance coefficients over routed nets. The importance coefficients can be used to model various practical objectives. For example, importance coefficients of 1 for each net correspond to maximizing the number of routed nets and importance coefficients equal to the number of sinks of the net correspond to maximizing the number of connected sinks.

We also consider the following extensions of the basic GRBB problem.

- *Multilayer GRBB:* The basic GRBB formulation imposes the same L/U bounds on the length of all buffer-to-buffer, source-to-buffer, and buffer-to-sink wire segments. The *multilayer GRBB problem* accounts for the different electrical characteristics (unit-wire resistance and capacitance) of different routing layers and takes into consideration nonuniform source driving strengths and sink input capacitances.
- *GRBB With Set Capacity Constraints:* The basic GRBB problem assumes predetermined capacities for all buffer blocks. In practice, there is some freedom for transferring capacity from a buffer block to neighboring buffer blocks by translating circuit blocks. The *GRBB problem with set capacity constraints* captures this freedom by allowing constraints on the total capacity of *sets* of buffer blocks, instead of only constraining individual buffer blocks.
- *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.

We give integer linear program (ILP) formulations for the basic GRBB problem and its extensions; these formulations generalize the vertex-capacitated integer *multiterminal multicommodity flow* (MTMCF) problem. The main contribution of the paper is a provably good algorithm for these generalizations of the MTMCF problem. Prior to our work, heuristics based on solving fractional relaxations of integer multicommodity flow formulations have been applied to VLSI global routing [1], [2], [12], [16], [21]. As noted in [15], the applicability of this approach is limited to problem instances of relatively small size by the prohibitive cost of solving exactly the fractional relaxation. As in the recent work of Albrecht [1], we avoid this limitation by using an approximation algorithm for solving the fractional relaxations. The approximation algorithm can find solutions within any desired accuracy; an important feature of the algorithm is that it allows for a smooth tradeoff between runtime and solution accuracy. Our experiments indicate that even low accuracy fractional solutions give good final solutions for the GRBB problem after rounding.

The most interesting feature of our algorithm is its ability to work with *multiterminal* nets. Note that our problem formulation addresses only the partitioning of long lines, although in the case of multipin nets buffers can also be used for decoupling off-path capacitance. Thus, the constraints on the number of buffers on source-to-sink paths are less correlated to the real timing requirements of multipin nets. However, we believe that this limitation of our model could be corrected by post-process fine tuning. Experiments on top-level layouts extracted from a recent high-end microprocessor design validate our algorithms and indicate that it significantly outperforms existing algorithms for the problem [5], [22] which are based on two-pin decompositions.

The rest of the paper is organized as follows. In Section II we give ILP formulations for the GRBB problem and its extensions and introduce a common generalization of these ILPs, referred to as the *generalized multiterminal multicommodity flow* (GMTMCF) ILP. The fractional relaxation of the GMTMCF ILP is a special type of *packing LP* and can thus be approximated within any desired accuracy using the algorithm of Garg and Könemann [10]. In Section III we give a significantly faster approximation algorithm, obtained by extending a speed-up idea due to Fleischer [9] to this special type of packing LPs. We give the details of a key subroutine of the algorithm—finding minimum-weight feasible Steiner trees—in Section IV and present algorithms for rounding near-optimal fractional GMTMCF solutions to near-optimal integral solutions in Section V. In Section VI, we describe implementations of several GRBB heuristics, some based on rounding approximate fractional GMTMCF solutions and some based on less sophisticated greedy ideas; Section VII gives the results of experiments comparing these heuristics on test cases extracted from the top-level layout of a recent high-end microprocessor. Finally, we conclude in Section VIII with a list of future research directions.

II. 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 $a_k^i \in \{\text{even}, \text{odd}\}$ 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. Let also $S = \{s_1, \ldots, s_K\}$ 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'. Specific routing graphs are defined in the following sections for the GRBB problem and each of its extensions. All vertices in these routing graphs have associated locations on the chip, including those in V' which are associated with buffer block locations. The edges are defined according to the specific L/U bounds imposed by each problem. Thus, every Steiner tree in G automatically satisfies the given L/U bounds assuming that a buffer is inserted at each Steiner point. To ensure that upper bound and parity constraints on the number of buffers on source-to-sink paths are met as well, we need to restrict the set of allowable Steiner trees as follows.

A path $p = (s_k, v_1, v_2, \ldots, v_l, t_k^i)$, connecting source s_k to sink t_k^i in routing graph G, is a *feasible* (s_k, t_k^i)*-path* if

- $v_i \in V'$ for each $i = 1, \ldots, l$;
- the parity of l is a_k^i;
- $l \leq l_k^i$.

A *feasible Steiner tree* for net N_k is a Steiner tree T_k in G 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 is a feasible (s_k, t_k^i)-path as defined above.

We will denote the set of all feasible Steiner trees for net N_k by $\mathcal{T}_k$ and let $\mathcal{T} = \bigcup_{k=1}^K \mathcal{T}_k$. Given importance coefficients $g_k = g(N_k)$ for each net N_k, we define $g(T) = g_k$ for each tree $T \in \mathcal{T}_k, k = 1, \ldots, K$.

A. ILP Formulation of GRBB

We begin by defining the routing graph $G = (V, E)$ for the GRBB problem. To allow feasible Steiner trees that meet parity constraints by using two buffers in the same buffer block, we introduce two distinct vertices, r' and r'', corresponding to each buffer block r and define $V = S \cup S' \cup \{r', r'' \mid r \in R\}$. If $d(x, y)$ denotes the length of the shortest rectilinear path connecting points x and y and avoiding all given rectangular obstacles, the edge set of G is defined by $E = E_0 \cup E_1$, where

$$E_0 = \{(r', r'') \mid r \in R\}$$
$$E_1 = \{(x, y) \mid x, y \in V, L \leq d(x, y) \leq U\}.$$

The GRBB problem is then equivalent to the following integer linear program:

$$\textbf{maximize} \quad \Sigma_{T \in \mathcal{T}} g(T) f_T \qquad \text{(ILP1)}$$
$$\textbf{subject to}$$
$$\Sigma_{T \in \mathcal{T}} \pi_T(v) f_T \leq 1, \qquad \forall v \in S \cup S'$$
$$\Sigma_{T \in \mathcal{T}} \left(\pi_T(r') + \pi_T(r'')\right) f_T \leq c(r), \qquad \forall r \in R$$
$$f_T \in \{0, 1\}, \qquad \forall T \in \mathcal{T}$$

where $\pi_T(v)$ is the number of occurrences of v in T, i.e.,

$$\pi_T(v) = \begin{cases} 0, & \text{if } v \notin T \\ 1, & \text{if } v \in T. \end{cases}$$

In (ILP1), the variable f_T is set to one if the feasible Steiner tree T is routed and to zero otherwise. Constraints of the first type (corresponding to $v \in S \cup S'$) ensure that at most one feasible Steiner tree is routed for each net; constraints of the second type (corresponding to $r \in R$) enforce buffer block capacities.

B. ILP Formulation for Multilayer GRBB

The basic version of the GRBB problem imposes identical L/U bounds on the length of all buffer-to-buffer, source-to-buffer, and buffer-to-sink wire segments. This is not appropriate when routing is done in multiple layers, since different layers have different electrical characteristics (unit-wire resistance and capacitance). In addition, signal sources typically have nonuniform driving strengths and signal sinks have nonuniform input capacitances. Thus, an accurate formulation of the GRBB problem for $z > 1$ routing layers must handle:

- *layer-dependent* lower and upper bounds, L_i/U_i, $i = 1, \ldots, z$, on the length of buffer-to-buffer wire segments;
- *source- and layer-dependent* lower and upper bounds, L_i^s/U_i^s, $s \in S$, $i = 1, \ldots, z$, on the length of source-to-buffer wire segments;
- *sink- and layer-dependent* lower and upper bounds, L_i^t/U_i^t, $t \in S'$, $i = 1, \ldots, z$, on the length of buffer-to-sink wire segments.

These additional parameters are taken into account by appropriately modifying the routing graph graph $G = (V, E)$. The vertex set of G remains the same, $V = S \cup S' \cup \{r', r'' : r \in R\}$, but we now define $E = E_0 \cup E_1 \cup E_2 \cup E_3$, where

$$E_0 = \{(r', r'') \mid r \in R\}$$
$$E_1 = \Big\{(s, r'), (s, r'') \mid s \in S, r \in R, \exists i \in \{1, \ldots, z\} \text{ s.t. } L_i^s \leq d_i(s, r) \leq U_i^s\Big\}$$
$$E_2 = \Big\{(r_1', r_2'), (r_1', r_2''), (r_1'', r_2'), (r_1'', r_2'') \mid r_1, r_2 \in R, r_1 \neq r_2, \exists i \in \{1, \ldots, z\} \text{ s.t. } L_i \leq d_i(r_1, r_2) \leq U_i\Big\}$$
$$E_3 = \Big\{(r', t), (r'', t) \mid r \in R, t \in S', \exists i \in \{1, \ldots, z\} \text{ s.t. } L_i^t \leq d_i(r, t) \leq U_i^t\Big\}.$$

Here, $d_i(x, y)$ denotes the length of the shortest rectilinear path connecting points x and y and avoiding all rectangular obstacles in layer i.

The multilayer GRBB problem is then equivalent to (ILP1) for the modified routing graph G.

C. ILP Formulation for GRBB With Set Capacity Constraints

Our basic formulation of the GRBB problem assumes predetermined capacities for all buffer blocks. In practice, buffer

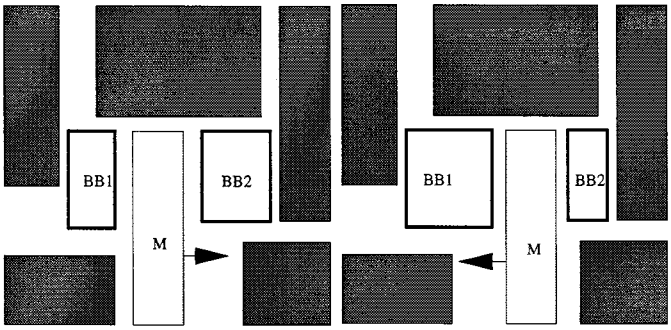

Fig. 1. Two buffer blocks BB1 and BB2 that share capacity; if the circuit block M moves right, then the capacity of buffer block BB1 is increasing while the capacity of buffer block BB2 is decreasing. In this example it is the sum of capacities of BB1 and BB2, rather than their individual capacities,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

blocks are placed in the free space available after compaction, when some of the circuit blocks can still be moved within certain limits, thus transferring capacity from a buffer block to neighboring buffer blocks (see Fig. 1). This freedom is captured by upper bounds on the total capacity of entire sets of buffer blocks, rather than individual buffer blocks.

Assume that, as the result of compaction, we have identified subsets $R_1, \ldots, R_p$ of R (some of which may consist of a single buffer block) such that there is a positive upper bound $c(R_i)$ on the total capacity of each R_i. Assuming further that circuit block movements are so small compared to the given L/U bounds that we can ignore changes in buffer block positions, it follows that the GRBB problem with set-capacity constraints given by $c(R_1), \ldots, c(R_p)$ is equivalent to (ILP1) in which the buffer block capacity constraints are replaced by

$$\sum_{T \in \mathcal{T}} \left(\sum_{r \in R_i} (\pi_T(r') + \pi_T(r'')) \right) f_T \leq c(R_i), \quad \forall i \in \{1, \ldots, p\}.$$

D. ILP Formulation for GRBB With Buffer Library

The basic GRBB problem formulation implicitly assumes the use of a single buffer type. In practice, better use of area and power resources may be achieved by using more than one type of buffer. In this section we give an integer program formulation for the GRBB problem with buffers chosen from a given buffer library. This version of the problem allows buffered routings of the nets using any mix of buffers from the given library and also allows buffers of different types (and hence, of different sizes) be placed in the same buffer block, up to the capacity of the block.

Let B be the set of buffer types in the library. We assume to be given the size, $size(b)$, for each buffer type $b \in B$, as well as lower and upper bounds $L^{(s,b)}/U^{(s,b)}$, $L^{(b,b')}/U^{(b,b')}$, respectively $L^{(b,t)}/U^{(b,t)}$, on the length of each wire segment connecting source $s \in S$ to a buffer of type b, a buffer of type b to a buffer of type b', respectively, a buffer of type b to sink $t \in S'$. To ensure that the available buffer block capacity is optimally distributed between the given buffer types and also allow feasible Steiner trees that use more than one buffer in the same buffer block, we introduce $2|B|$ vertices corresponding to each buffer block r. Formally, the routing graph $G = (V, E)$ has vertex set $V = S \cup S' \cup \{r'_b, r''_b \mid r \in R,\ b \in B\}$ and edge set $E = E_0 \cup E_1 \cup E_2 \cup E_3 \cup E_4$, where

$$E_0 = \left\{ (r'_b, r''_b) \mid r \in R, b \in B \right\}$$

$$E_1 = \left\{ \left(r'_{b_1}, r'_{b_2}\right), \left(r'_{b_1}, r''_{b_2}\right), \left(r''_{b_1}, r'_{b_2}\right), \left(r''_{b_1}, r''_{b_2}\right) \mid r \in R,\ b_1, b_2 \in B, b_1 \neq b_2 \right\}$$

$$E_2 = \left\{ (s, r'_b), (s, r''_b) \mid s \in S, r \in R, b \in B, \ L^{(s,b)} \leq d(s,r) \leq U^{(s,b)} \right\}$$

$$E_3 = \left\{ \left(r'_{b_1}, q'_{b_2}\right), \left(r'_{b_1}, q''_{b_2}\right), \left(r''_{b_1}, q'_{b_2}\right), \left(r''_{b_1}, q''_{b_2}\right) \mid r, q \in R, r \neq q, b_1, b_2 \in B, \ L^{(b_1,b_2)} \leq d(r,q) \leq U^{(b_1,b_2)} \right\}$$

$$E_4 = \left\{ (r'_b, t), (r''_b, t) \mid r \in R, b \in B, t \in S', \ L^{(b,t)} \leq d(r,t) \leq U^{(b,t)} \right\}.$$

The GRBB problem with buffer library B is then equivalent to (ILP1) in which the buffer block capacity constraints are replaced by

$$\sum_{T \in \mathcal{T}} \left(\sum_{b \in B} (\pi_T(r'_b) + \pi_T(r''_b))\, size(b) \right) f_T \leq c(r), \quad \forall r \in R.$$

E. The Generalized MTMCF ILP

Note that (ILP1) and its extensions are already strict generalizations of previous integer edge-capacitated multiterminal multicommodity flow formulations used for VLSI global routing [1], [20], since they impose capacities on vertices and/or specific sets of vertices. In this section we formulate a *generalized multiterminal multicommodity flow* (GMTMCF) ILP which allows: 1) capacities on arbitrary sets of vertices and 2) arbitrary vertex weights saying how much capacity is used by a tree visiting the vertex, thus capturing all given GRBB formulations.

Given:

- nets N_k, $k = 1, \ldots, K$, with importance coefficients g_k;
- a routing graph $G = (V, E)$ for the nets;
- arbitrary sets, $\mathcal{T}_k$, of Steiner trees for each net N_k;
- a family, $\mathcal{V}$, of subsets of V, such that $\{v\} \in \mathcal{V}$ for every $v \in S \cup S'$;
- a "size" function $s : V \to R_+$ such that $s(v) = 1$ for every $v \in S \cup S'$;
- a "set capacity" function $c : \mathcal{V} \to Z_+$ such that $c(\{v\}) = 1$ for every $v \in S \cup S'$;

the generalized multiterminal multicommodity flow ILP is as follows:

$$\begin{aligned} &\textbf{maximize} \quad \Sigma_{T\in\mathcal{T}} g(T) f_T && \text{(GMTMCF ILP)}\\ &\textbf{subject to}\\ &\qquad \Sigma_{T\in\mathcal{T}} \pi_T(X) f_T \le c(X), \qquad \forall X \in \mathcal{V}\\ &\qquad f_T \in \{0,1\}, \qquad \forall T \in \mathcal{T} \end{aligned}$$

where $\mathcal{T} = \cup_{k=1}^{K} \mathcal{T}_k$, $g(T) = g_k$ for every $T \in \mathcal{T}_k$ and

$$\pi_T(X) = \sum_{v\in X} \pi_T(v) s(v)$$

for every $T \in \mathcal{T}$ and $X \in \mathcal{V}$.

It is not difficult to see that (ILP1) and its extensions are special cases of the GMTMCF ILP. Thus, (ILP1) is obtained with $s \equiv 1$ by including in $\mathcal{V}$, besides singleton sets corresponding to sources and sinks, all sets $X_r = \{r', r''\}$, $r \in R$ and setting $c(X_r) = c(r)$. Similarly, the ILP for GRBB with set capacity constraints is obtained with $s \equiv 1$ by including in $\mathcal{V}$ singleton sets corresponding to sources and sinks, as well as the sets $X_i = \{r', r'' \mid r \in R_i\}$, $i = 1, \ldots, p$, with $c(X_i) = c(R_i)$. Finally, the ILP for GRBB with buffer library is obtained for a family $\mathcal{V}$ containing singleton sets corresponding to sources and sinks together with the sets $X_r = \{r'_b, r''_b \mid b \in B\}$, $r \in R$, for which $c(X_r) = c(r)$. In this case, $s(v) = 1$ if v is a source or a sink and $s(v) = size(b)$ if $v \in \{r'_b, r''_b \mid r \in R\}$.

III. Approximating the GMTMCF LP Relaxation

Our two-step approach to the GRBB problem and its extensions is to: 1) solve the fractional relaxations obtained by replacing integrality constraints $f_T \in \{0,1\}$ with $f_T \ge 0$ and then 2) use randomized rounding to get integer solutions. In this section we give an algorithm for approximating within any desired accuracy the fractional relaxation of the GMTMCF ILP. The algorithm relies on a subroutine for finding minimum weight feasible Steiner trees; the details of this subroutine are given in Section IV.

The fractional relaxation of the GMTMCF ILP, which will be referred to here as the *generalized multiterminal multicommodity flow linear program* (GMTMCF LP),[4] can be solved exactly in polynomial time using, e.g., the ellipsoid algorithm. However, exact algorithms are highly impractical. On the other hand, the GMTMCF LP is a *packing LP* and can thus be efficiently approximated within any desired accuracy using the recent combinatorial algorithm of Garg and Könemann [10]. In this section we give a significantly faster approximation algorithm based on a speedup idea originally proposed by Fleischer [9] for approximating the maximum edge-capacitated multicommodity flow (MCF).

A. The Algorithm

Our algorithm simultaneously finds feasible solutions to the GMTMCF LP and its *dual linear program*. The dual LP asks for an assignment of nonnegative 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 $weight(T) = 1/g(T) \sum_{X\in\mathcal{V}} \pi_T(X) w(X)$

$$\begin{aligned} &\textbf{minimize} \quad \Sigma_{X\in\mathcal{V}} w(X) c(X) && \text{(GMTMCF Dual)}\\ &\textbf{subject to}\\ &\qquad \frac{1}{g(T)} \sum_{X\in\mathcal{V}} \pi_T(X) w(X) \ge 1, \qquad \forall T \in \mathcal{T}\\ &\qquad w(X) \ge 0, \qquad \forall X \in \mathcal{V}. \end{aligned}$$

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 (Fig. 2) starts with weights $w(X) = \delta$ for every $X \in \mathcal{V}$, where δ is an appropriately chosen constant and with a GMTMCF 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_Ts. Whenever f_T is incremented, the algorithm also updates each weight $w(X)$ by multiplying it with $(1 + \varepsilon \pi_T(X)/c(X))$, for a fixed ε.

According to the Garg and Könemann's approximation algorithm [10], 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 speedup idea due to Fleischer [9]. Instead of always finding the minimum-weight tree in $\mathcal{T}$, the idea is to settle for trees with weight within a factor of $(1+\varepsilon)$ of the minimum-weight. This speeds up the computation since multiple variables f_T can now be incremented, possibly more than once each, in a single iteration. As shown in next section, the faster algorithm still leads to an approximation guarantee similar to that of Garg and Könemann.

B. 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 $(1+\varepsilon)$ times a lower-bound $\bar{\alpha}$ on the overall minimum weight, $\min\{\text{weight(T)} : \text{T} \in \mathcal{T}\}$. The lower-bound $\bar{\alpha}$ is initially set to δ/Γ and then multiplied by a factor of $(1+\varepsilon)$ from one iteration to another (note that no tree in $\mathcal{T}$ has weight smaller than $(1+\varepsilon)\bar{\alpha}$ at the end of an iteration, so $(1+\varepsilon)\bar{\alpha}$ 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. Note that the lower-bound $\bar{\alpha}$ is at most $(1+\varepsilon)$ in the last iteration (since it increases by a factor of $(1+\varepsilon)$ each time and in the iteration before the last there is at least one tree of weight less than 1). Thus, since $\bar{\alpha} = \delta/\Gamma$ in the first iteration, the number of increases of $\bar{\alpha}$ is no larger than $\log_{1+\varepsilon}(1+\varepsilon)\Gamma/\delta$ and the final value of i is $\lfloor \log_{1+\varepsilon}(1+\varepsilon)\Gamma/\delta \rfloor$.

The two main loops of the algorithm (on i and on k) are both repeated a fixed number of times, $\lfloor \log_{1+\varepsilon}(1+\varepsilon)\Gamma/\delta \rfloor$ and K, respectively. However, this does not immediately determine the

[4]The relaxation of the GMTMCF ILP is referred to as *separable packing LP* in [8].

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, weights $c(X)$, $X\in\mathcal{V}$, and $s(v)$, $v\in V$
Output: GMTMCF LP solution f_T, $T\in\mathcal{T}$

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

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

runtime of the algorithm because of the variable number of iterations in the inner while loop. The following lemma gives an upper bound on the runtime.

Lemma 1: Overall, the algorithm in Fig. 2 requires $O(K\log_{1+\varepsilon}(1+\varepsilon)\Gamma/\delta)$ 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 $K\lfloor\log_{1+\varepsilon}(1+\varepsilon)\Gamma/\delta\rfloor$. Indeed, in each iteration and for each net N_k, there is exactly one minimum-weight feasible Steiner tree computation revealing that $\min_{T\in\mathcal{T}_k} weight(T)\geq(1+\varepsilon)\bar{\alpha}$, 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 $K\log_{1+\varepsilon}(1+\varepsilon)\Gamma/\delta$. 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 $1+\varepsilon$ in each update, including the last one. This implies that the final value of $w(\{s_k\})$ is at most $(1+\varepsilon)\Gamma$. Recalling that $w(\{s_k\})$ is initially set to δ, this gives that the number of updates to $w(\{s_k\})$ is at most $\log_{1+\varepsilon}(1+\varepsilon)\Gamma/\delta$. The lemma follows by summing this upper bound over all nets. □

Lemma 2: The algorithm in Fig. 2 computes a feasible solution to the GMTMCF LP.

Proof: We need to show that the values f_T returned by the algorithm satisfy the inequality

$$\sum_{T\in\mathcal{T}}\pi_T(X)f_T\leq c(X)\cdot\log_{1+\varepsilon}\frac{(1+\varepsilon)\Gamma}{\delta}$$

for every $X\in\mathcal{V}$. Consider an arbitrary set $X\in\mathcal{V}$. Every time the variable f_T is incremented by one unit, the weight of X is also increased by a factor of $(1+x_i\varepsilon/c(X))$, where $x_i=\pi_T(X)$. Using that $1+y\varepsilon\geq(1+\varepsilon)^y$ for $0\leq y\leq 1$, we get that every sequence of updates with $\Sigma_i x_i=c(X)$ increases the weight of X by a factor of at least

$$\prod_i\left(1+\frac{x_i\varepsilon}{c(X)}\right)\geq\prod_i(1+\varepsilon)^{x_i/c(X)}=(1+\varepsilon)^{\Sigma_i x_i/c(X)}=1+\varepsilon.$$

Let $M=\sum_{T\in\mathcal{T}}\pi_T(X)f_T$. Since the initial weight of X is δ, from the previous inequality we get that the final weight of X is at least $\delta(1+\varepsilon)^{M/c(X)}$.

Now, the last update of $w(X)$ is done when incrementing f_T for a tree $T\in\mathcal{T}_k$ of weight less than one. Thus, the weight of X is at most $g(T)\cdot weight(T)<\Gamma$ before last update and at most $(1+\varepsilon)\Gamma$ after. Combining this upper bound on the final weight of X with the lower bound above gives that $\delta(1+\varepsilon)^{M/c(X)}\leq(1+\varepsilon)\Gamma$, i.e.,

$$M\leq c(X)\log_{1+\varepsilon}\frac{(1+\varepsilon)\Gamma}{\delta}.$$

□

Finally, we show that, for an appropriate value of the parameter δ, the feasible solution found by the algorithm is close to optimum.

Theorem 1: For every $\varepsilon<0.15$, the algorithm in Fig. 2 computes a feasible solution to the GMTMCF LP within a factor of $1/(1+4\varepsilon)$ of optimum by choosing $\delta=(1+\varepsilon)\Gamma((1+\varepsilon)L\Gamma)^{-1/\varepsilon}$; the runtime of the algorithm for this value of δ is $O(1/\varepsilon^2K(\log L+\log\Gamma)T_{\text{tree}})$. 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 [10] and Fleischer [9]. We show that the solution computed by the algorithm is within a factor of $1/(1+4\varepsilon)$ of the optimum objective value, β, of the dual LP. The claimed approximation guarantee follows, since, by LP duality theory, β is an upper bound on the optimum objective value of the GMTMCF LP.

Let $\alpha(w)$ be the weight of a minimum weight tree from $\mathcal{T}$ with respect to weight function $w:\mathcal{V}\rightarrow R_+$ 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 GMTMCF LP at the end of this iteration.

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

$$\varepsilon\sum_{X\in\mathcal{V}}\pi_T(X)w(X)=\varepsilon\,weight(T)g(T).$$

If this update takes place in the ith iteration, then $\text{weight}(T) \leq (1+\varepsilon)\alpha(i-1)$. Adding this over all f_Ts incremented in ith iteration gives

$$D(i) - D(i-1) \leq \varepsilon(1+\varepsilon)\alpha(i-1)\,(h_i - h_{i-1})$$

which implies that

$$D(i) - D(0) \leq \varepsilon(1+\varepsilon)\sum_{j=1}^{i}\alpha(j-1)\,(h_j - h_{j-1})\,.$$

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, $\alpha(i) \leq \alpha(w_i - w_0) + L\delta$. Hence, if $\alpha(i) - L\delta > 0$, then

$$\begin{aligned}\beta \leq & \frac{D(w_i - w_0)}{\alpha(w_i - w_0)} \leq \frac{D(i) - D(0)}{\alpha(i) - L\delta} \\ \leq & \frac{\varepsilon(1+\varepsilon)\sum_{j=1}^{i}\alpha(j-1)\,(h_j - h_{j-1})}{\alpha(i) - L\delta}.\end{aligned}$$

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

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

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

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

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

$$\alpha(i) \leq L\delta e^{(\varepsilon(1+\varepsilon)/\beta)h_i}.$$

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

$$1 \leq L\delta e^{(\varepsilon(1+\varepsilon)/\beta)h_t}$$

and thus

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

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

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

For $\delta = (1+\varepsilon)\Gamma((1+\varepsilon)L\Gamma)^{-1/\varepsilon}$

$$\begin{aligned}\frac{\ln\frac{(1+\varepsilon)\Gamma}{\delta}}{\ln(L\delta)^{-1}} = & \frac{\ln\left((1+\varepsilon)L\Gamma\right)^{1/\varepsilon}}{\ln\left((1+\varepsilon)L\Gamma\right)^{-1+1/\varepsilon}} \\ = & \frac{\frac{1}{\varepsilon}\ln\left((1+\varepsilon)L\Gamma\right)}{\frac{1-\varepsilon}{\varepsilon}\ln\left((1+\varepsilon)L\Gamma\right)} \\ = & \frac{1}{1-\varepsilon}\end{aligned}$$

and thus

$$\gamma \leq \frac{\varepsilon(1+\varepsilon)}{(1-\varepsilon)\ln(1+\varepsilon)} \leq \frac{\varepsilon(1+\varepsilon)}{(1-\varepsilon)\left(\varepsilon - \frac{\varepsilon^2}{2}\right)} \leq \frac{(1+\varepsilon)}{(1-\varepsilon)^2}.$$

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

IV. 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 $\text{weight}(T) = 1/g(T)\sum_{X\in\mathcal{V}}\pi_T(X)w(X)$. Define a weight function w' on the vertices of the routing graph $G = (V, E)$ by setting $w'(v) = 1/g(T)\sum_{v\in X\in\mathcal{V}} w(X)$ and let $w'(T) = \sum_{v\in V(T)} w'(v)$ be the total vertex weight w.r.t. w' of T. Then $\text{weight}(T) = w'(T)$ 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 D_k, defined as follows. Let $L_k = \max\{l_k^1, \ldots, l_k^{q_k}\}$ and $V' = V(G) \setminus (S \cup S')$. Then

$$V(D_k) = \{s_k\} \cup \{v_j \mid v \in V', 1 \leq j \leq L_k\} \cup \left\{t_k^1, \ldots, t_k^{q_k}\right\}$$

and $E(D_k) = E_1 \cup E_2 \cup E_3$, where

$$\begin{aligned}E_1 = & \{(s_k, v_1) \mid v \in V', (s_k, v) \in E(G)\} \\ E_2 = & \{(u_j, v_{j+1}) \mid u, v \in V', 1 \leq j < L_k, (u, v) \in E(G)\} \\ E_3 = & \left\{\left(u_j, t_k^h\right) \mid u \in V', 1 \leq h \leq q_k,\right. \\ & \left. 1 \leq j \leq l_k^h, j \equiv a_k^h (\text{mod } 2), \left(v, t_k^h\right) \in E(G)\right\}.\end{aligned}$$

Fig. 3 shows the routing graph and the diagraph D_k corresponding to a net with source $s = s_k$ and a single sink $t = t_k^1$, for which $l_k^1 = 2$ and $a_k^1 = 2$. In this example there are only

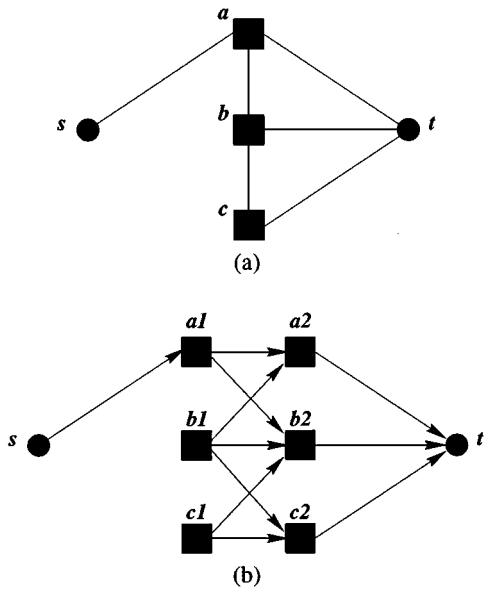

Fig. 3. (a) Routing graph with three buffer blocks and (b) the correponding diagraph D_k.

Input: Fractional GMTMCF solution f_T, $T \in \mathcal{T}$
Output: Routed trees, $T_k \in \mathcal{T}_k$

For each $k = 1,\ldots,K$, select net N_k with probability $f_k = \sum_{T \in \mathcal{T}_k} f_T$
Route each selected net N_k by picking a tree T_k from $\mathcal{T}_k$, where tree $T \in \mathcal{T}_k$ is picked with probability f_T / f_k

Fig. 4. Randomized GMTMCF rounding algorithm.

Input: Net- and edge-cumulated GMTMCF 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. 5. Random walk-based GMTMCF rounding algorithm.

two feasible buffered routings (i.e., directed s_k-t_k paths in D_k): one is using two buffers in block a, the other one uses one buffer in block a and one buffer in block b.

For a given directed graph $H = (X, U)$ with costs on arcs, a specified root $r \in X$, and a set of terminals $Y \subset X$, the directed rooted Steiner tree problem asks to find the minimum cost arborescence rooted at r and spanning all the vertices in Y (in other words r should have a directed path to every vertex in Y). It is easy to see that finding a feasible Steiner tree $T \in \mathcal{T}_k$ with minimum $w'(T)$ reduces to finding a minimum cost DRST in D_k after assigning to each arc entering vertex v_i, $v \in V'$, $1 \leq j \leq L_k$, a cost of $w'(v)$ and to each arc entering sink t_k^h, $1 \leq h \leq q_k$, a cost of $w'(t_k^h)$.

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 quasipolynomial 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 $M = 2, 3, 4$). Theorem 1 immediately gives the following.

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

Since most of the nets in real designs have small size, Corollary 1 justifies the following practical strategy for finding approximate solutions to the GMTMCF LP: decompose nets with more than four pins into nets with two to four pins, then apply the approximation algorithm in Fig. 2. Another heuristic approach to speedup the computation, used by Albrecht [1] for edge-capacitated MTMCF approximation, is to compute (exactly or approximately) a DRST once, then use in each of the following iterations minimum directed spanning trees (with respect to the updated edge lengths) in the directed acyclic graph induced by $\{s_k, t_k^1, \ldots, t_k^{q_k}, p_1, \ldots, p_s\}$ in the metric closure of D_k, where $p_1, \ldots, p_s$ are the Steiner points of the original DRST. To find a minimum spanning directed tree in directed acyclic graphs, one can use a very simple procedure: for each vertex choose a shortest incoming arc, then, after running this procedure, recursively delete all leaves that are not sinks of the net N_k.

We have implemented both heuristics that use approximate DRSTs instead of optimum DRSTs and heuristics based on two-, three-, respectively four-pin decompositions; results of experiments comparing these approaches are reported in Section VII.

V. Rounding the Fractional GMTMCF

In the previous two sections we presented an algorithm for computing near-optimal solutions to the GMTMCF LP. In this section we give two algorithms based on the randomized rounding technique of Raghavan and Thomson [19] (see also [15]) for converting near-optimal fractional GMTMCF solutions to near-optimal integer GMTMCF solutions, i.e., to near optimal buffered routings.

The first algorithm is given in Fig. 4. Since the algorithm routes net N_k with probability $f_k = \sum_{T \in \mathcal{T}_k} f_T$, it follows that, on the average, the total importance of the nets routed by the 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 $1/(1+4\varepsilon)$ of the optimum GMTMCF LP solution, which in turn is an upper bound on the optimum GMTMCF ILP solution.

A potential drawback of the first rounding algorithm is that it requires the explicit representation of trees $T \in \mathcal{T}$ with $f(T) \neq 0$. Although the approximate GMTMCF algorithm produces a polynomial number of trees with nonzero f_T, storing all such trees is infeasible for large problem instances. Our second rounding algorithm (Fig. 5) takes as input the net and edge *cumulated* GMTMCF 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 $O(K|E|)$ space. Note that the GMTMCF algorithm in Fig. 2 can be easily modified to compute these cumulated GMTMCF values instead of f_Ts.

Our second rounding algorithm routes net $N_k = (s_k; t_k^1, \ldots, t_k^{q_k})$ with the same probability as the first rounding algorithm and thus, as argued above, the total importance of the routed nets is within a factor of $1/(1 + 4\varepsilon)$ of optimum. 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 s_k or a vertex already connected to the source. The random walks are performed in the directed acyclic graphs D_k, with probabilities given by the normalized $f_k(e)$ values.

Ensuring that no set capacity is exceeded can be accomplished in two ways. One approach is to solve the GMTMCF LP with set capacities scaled down by a small factor; this guarantees that the rounded solution meets the *original* capacities with very high probability (see [15] for an application of this approach to VLSI global routing). A simpler approach, extending the so-called *greedy-deletion algorithm* [6] to multiterminal nets, is to repeatedly drop routed nets passing through overused sets until feasibility is achieved. We implement a modified version of the greedy-deletion algorithm in which, instead of dropping an entire tree, we drop only the sinks using paths through over-used sets.

VI. Implemented Algorithms

In this section we describe the implemented algorithms for the GRBB problem.

Greedy Routing Algorithms: We have implemented four greedy algorithms for the GRBB problem, all of them based on the generic greedy routing algorithm given in Fig. 6. All four greedy algorithms route nets sequentially. For a given net, the algorithms start 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 two-, three-, respectively, four-pin nets and then apply the algorithm in Fig. 6 to this decomposition.[5] The fourth algorithm—which we refer to as MTG—is simply the algorithm in Fig. 6 applied to the original (undecomposed) nets.

GMTMCF Rounding Algorithms: We have implemented four GMTMCF rounding algorithms, all of them based on the generic schema given in Fig. 7. The first three algorithms—referred to as G2TMCF, G3TMCF, and G4TMCF, respectively—start by decomposing larger multiterminal nets into two-, three-, respectively, four-pin nets and then apply the generic GMTMCF routing algorithm to this decomposition. Since the optimum DRST can be efficiently computed for nets of these sizes, the three algorithms do not need to resort to the DRST approximations suggested at the end of Section IV. The fourth algorithm—henceforth referred to as GMTMCF—applies the flow rounding schema in Fig. 7 to the undecomposed nets, using shortest path trees as approximate DRSTs in the GMTMCF approximation step.

[5]We remark that 2TG is essentially the algorithm suggested in [5], except that in [5] shortest paths are computed in the forward direction, from sources toward sinks and not from sinks toward sources as in Fig. 6. It has been experimentally observed [7] that backward shortest paths give slightly better results than forward shortest paths.

Input: Graph G with K nets $N_1, \ldots, N_K$, vertex capacities $c(v)$
Output: Fully or partially routed feasible Steiner trees $\mathcal{T}_k \in \mathcal{T}_k$

For each $k = 1, \ldots, K$, do
  $\mathcal{T}_k \leftarrow \{s_k\}$
  For each sink t_k^i in N_k do
    With a backward BFS search, find a shortest path P from t_k^i to $\mathcal{T}_k$ in G using only vertices v with $c(v) > 0$; if no such path exists let $P = \emptyset$
    $\mathcal{T}_k \leftarrow \mathcal{T}_k \cup P$
    For each vertex v in P, $c(v) \leftarrow c(v) - 1$
  End for
End for

Fig. 6. The generic greedy routing algorithm.

Input: Graph G with K nets $N_1, \ldots, N_K$, vertex capacities $c(v)$
Output: Fully or partially routed feasible Steiner trees $\mathcal{T}_k \in \mathcal{T}_k$

Find an approximate GMTMCF using the algorithm in Fig. 2
Round the approximate GMTMCF using the algorithm in Fig. 5
Use greedy deletion to find a feasible integer solution
Use the MTG algorithm in Fig. 6 on the unrouted nets to find a maximal routing

Fig. 7. The generic GMTMCF-based routing algorithm.

TABLE I
Instance Parameters

ID	#Nets	#Sinks	Pins/net	L	U	BB Cap.
h1	2396	2958	2.23	2000	4000	200
h2	2438	3077	2.26	1000	4000	200
h3	2448	3099	2.27	500	4000	200
i1	4764	6038	2.27	2000	4000	400
i2	4925	6296	2.28	1000	4000	400
i3	4938	6321	2.28	500	4000	400

VII. Implementation Experience

All experiments were conducted on an SGI Origin 2000 with 16 195 MHz MIPS R10000 processors (only one of which is actually used by the sequential implementations included in our comparison) and 4 G-Bytes of internal memory, running under IRIX 6.4 IP27. Timing was performed using low-level Unix interval timers, under similar load conditions for all experiments. All algorithms were coded in C and compiled using `gcc` version egcs-2.91.66 with `-O4` optimization.

The six test cases used in our experiments were extracted from the next-generation (as of January 2000) microprocessor chip at SGI. We used an optimized floorplan of the circuit blocks and also optimized the location of the source/sink pin locations based on coarse timing budgets. We used $U = 4000$ μm and varied L between 500 and 2000 μm. The upper bounds on the number of buffers on paths from sources s_k to sinks t_k^i were computed with the formula $l_k^i = d(s_k, t_k^i)/1000$. In all test cases considered, the number of nets was large (up to 5000) and the number of buffer blocks small (50), with relatively large capacity (200–400 buffers per block); such values are typical for this application. Table I summarizes the parameters for the six test cases.

TABLE II
PERCENT OF SINKS CONNECTED (BOLDFACE) AND CPU TIME FOR ALGORITHMS BASED ON TWO-PIN DECOMPOSITION

ID	2TG	G2TMCF				
		$\varepsilon = 0.64$	$\varepsilon = 0.32$	$\varepsilon = 0.16$	$\varepsilon = 0.08$	$\varepsilon = 0.04$
h1	**88.3**	**93.7**	**95.4**	**95.4**	**95.7**	**95.5**
	0.33	1.63	5.38	20.29	87.71	357.09
h2	**88.7**	**93.9**	**95.8**	**96.8**	**96.6**	**96.5**
	0.37	2.35	5.99	22.78	88.44	349.63
h3	**88.4**	**93.5**	**95.4**	**96.5**	**96.4**	**95.5**
	0.37	1.80	6.13	24.87	97.75	392.02
i1	**90.6**	**94.8**	**95.8**	**96.5**	**96.6**	**96.8**
	0.65	3.26	10.71	39.61	164.85	622.87
i2	**91.6**	**96.2**	**97.1**	**97.4**	**97.5**	**97.6**
	0.70	3.54	12.34	45.00	171.36	671.72
i3	**91.5**	**96.2**	**96.9**	**97.3**	**97.3**	**97.5**
	0.73	3.57	11.84	47.17	172.01	770.51

TABLE III
PERCENT OF SINKS CONNECTED (BOLDFACE) AND CPU TIME FOR ALGORITHMS BASED ON THREE-PIN DECOMPOSITION

ID	3TG	G3TMCF				
		$\varepsilon = 0.64$	$\varepsilon = 0.32$	$\varepsilon = 0.16$	$\varepsilon = 0.08$	$\varepsilon = 0.04$
h1	**90.2**	**96.2**	**97.1**	**97.3**	**97.7**	**97.8**
	0.31	9.16	35.33	127.17	498.34	2090.61
h2	**90.1**	**96.4**	**98.3**	**98.6**	**98.9**	**98.5**
	0.34	11.10	41.56	154.76	626.47	2355.66
h3	**89.8**	**96.4**	**97.7**	**98.3**	**98.1**	**98.0**
	0.45	12.56	39.65	156.93	639.53	2364.51
i1	**91.9**	**95.7**	**96.8**	**97.3**	**97.5**	**97.6**
	0.63	15.99	54.94	208.06	814.21	3362.20
i2	**92.7**	**97.0**	**98.0**	**98.4**	**98.5**	**98.6**
	0.66	20.74	69.33	248.32	964.22	3834.26
i3	**92.5**	**96.8**	**97.8**	**98.3**	**98.4**	**98.4**
	0.72	19.07	66.23	251.17	992.41	4164.50

TABLE IV
PERCENT OF SINKS CONNECTED (BOLDFACE) AND CPU TIME FOR ALGORITHMS BASED ON FOUR-PIN DECOMPOSITION

ID	4TG	G4TMCF				
		$\varepsilon = 0.64$	$\varepsilon = 0.32$	$\varepsilon = 0.16$	$\varepsilon = 0.08$	$\varepsilon = 0.04$
h1	**90.8**	**97.9**	**98.3**	**98.9**	**98.9**	**99.2**
	0.31	56.16	305.70	1187.99	4881.40	19083.50
h2	**90.5**	**98.2**	**99.0**	**99.6**	**99.8**	**99.8**
	0.34	75.98	364.02	1629.60	6520.44	24779.28
h3	**90.1**	**97.7**	**98.8**	**99.5**	**99.6**	**99.3**
	0.33	75.26	392.30	1619.30	6378.52	25136.72
i1	**92.6**	**97.1**	**98.1**	**98.6**	**98.8**	**98.8**
	0.60	153.46	782.65	3354.75	13910.05	56530.93
i2	**93.0**	**98.3**	**98.7**	**99.4**	**99.6**	**99.4**
	0.71	191.14	1038.84	4550.27	17888.36	71636.67
i3	**92.8**	**98.2**	**98.6**	**99.3**	**99.3**	**99.3**
	0.69	195.40	1062.71	4507.83	18438.52	73712.45

TABLE V
PERCENT OF SINKS CONNECTED (BOLDFACE) AND CPU TIME FOR ALGORITHMS OPERATING ON UNDECOMPOSED NETS

ID	MTG	GMTMCF				
		$\varepsilon = 0.64$	$\varepsilon = 0.32$	$\varepsilon = 0.16$	$\varepsilon = 0.08$	$\varepsilon = 0.04$
h1	**92.2**	**96.7**	**97.4**	**97.5**	**97.6**	**97.4**
	0.30	2.33	11.21	47.11	223.96	946.78
h2	**92.3**	**97.6**	**98.9**	**99.2**	**99.3**	**99.3**
	0.33	2.87	13.84	53.31	226.13	868.25
h3	**92.1**	**97.3**	**98.2**	**98.5**	**98.8**	**98.7**
	0.33	2.86	12.66	53.74	219.20	876.63
i1	**93.5**	**96.3**	**97.4**	**97.5**	**97.7**	**97.7**
	0.59	4.98	24.33	102.57	420.24	1865.81
i2	**93.6**	**97.7**	**98.1**	**98.2**	**98.3**	**98.4**
	0.64	5.38	26.39	111.32	452.82	1827.98
i3	**93.3**	**97.7**	**98.1**	**98.1**	**98.2**	**98.2**
	0.70	5.43	26.54	121.55	454.84	1833.17

Tables II–V give the number of routed sinks and the runtime on the six test cases for each implemented algorithm. The results clearly demonstrate the high quality of solutions obtained by flow rounding methods. When applied to identical decompositions, flow-based methods yield improvements of 5%–9% in the number of connected sinks over the corresponding greedy algorithm. In fact, significant improvement over the best of the greedy methods is possible even with a very small increase in runtime, proof that even very coarse MCF/MTMCF approximations give helpful hints to the randomized rounding procedure.

Furthermore, the experimental results show that even a limited use of multiterminal nets (decomposition into nets of size 3 or 4) gives improvements over the already very high-quality solutions found by the flow-rounding algorithm based on two-pin decompositions. More importantly, these improvements are observed even when the same time budget is given to the compared algorithms. To facilitate such a comparison, Fig. 8 plots the solution quality versus the CPU time (in seconds, excluding I/O and memory allocation) of each algorithm when run on test case i1. The GMTMCF algorithm proves to be the best among all flow 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 G4TMCF, which dominates all other algorithms when high time budgets are allowed.

Table VI gives the amount of routing resources (buffers and wire area) used by each algorithm on test case i1. As expected, the amount of routing resources is higher for the algorithms with higher completion rates. In fact, even when normalizing by the number of connected sinks, the resource usage is slightly higher for these algorithms. This is at least partly explained by the fact that higher completion rates can only be achieved by routing a larger percentage of "difficult" nets, which may otherwise be ignored. Note for example the increase in wirelength and average number of buffers per routed sink for the MTG algorithm when the completion rate is boosted by increasing buffer block capacities (columns MTG and MTG_∞ in Table VI).

VIII. CONCLUSION AND FUTURE DIRECTIONS

In this paper, we addressed the problem of how to perform buffering of global *multiterminal* nets given an existing buffer block plan. We gave a provably good algorithm based on a novel approach to GMTMCF approximation inspired by recent results due to Garg and Könemann [10] and Fleischer [9]. Our GMTMCF algorithm outperforms existing algorithms for the problem [5] and has been validated on top-level layouts extracted from a recent high-end microprocessor design.

Ongoing work is aimed at increasing the space of methodologies to which our new techniques apply. As presented here, our work is clearly targeted to very early global wireplanning activity. In other words, the application domain is presynthesis chip planning: prescribed repeater intervals are driven only by coarse estimates of Miller coupling factors, repeater sizing, and source impedance or sink capacitance. The presented formulation also does not address timing criticalities or budgets except via net weighting; this is fortunately fairly common for initial wireplanning that breaks the "chicken-egg" problem of budgeting between-block and within-block paths in presynthesis

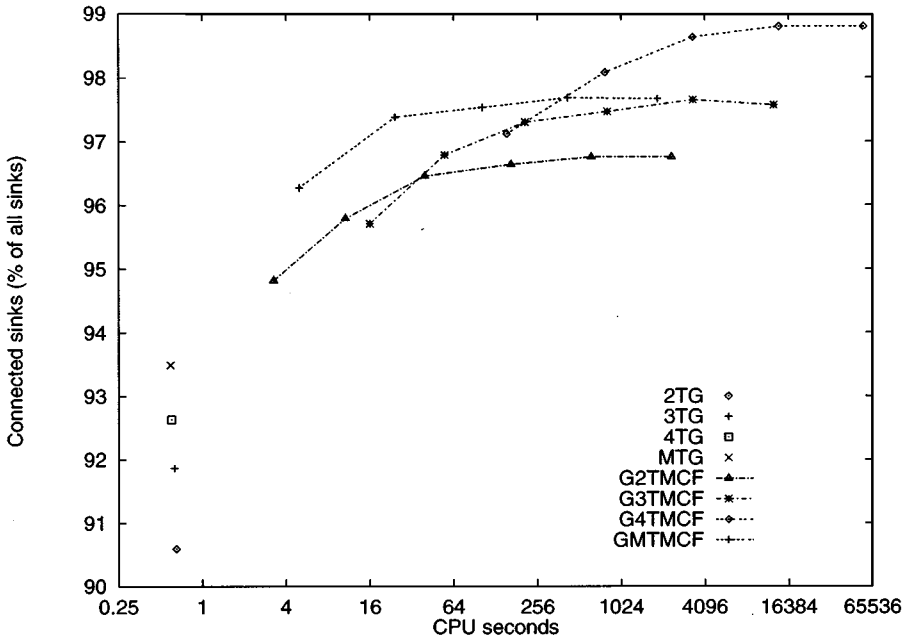

Fig. 8. Percent of sinks connected versus CPU time on test case i1.

TABLE VI
USAGE OF ROUTING RESOURCES ON TEST CASE I1 (6038 SINKS TOTAL). THE MTG_∞ COLUMN GIVES RESOURCE USAGE FOR THE MTG ALGORITHM WHEN RUN WITH INFINITY CAPACITY FOR EACH BUFFER BLOCK

	Greed					G2TMCF		G3TMCF		G4TMCF		GMTMCF	
	2TG	3TG	4TG	MTG	MTG_∞	$\varepsilon = 0.64$	$\varepsilon = 0.04$	$\varepsilon = 0.64$	$\varepsilon = 0.04$	$\varepsilon = 0.64$	$\varepsilon = 0.04$	$\varepsilon = 0.64$	$\varepsilon = 0.04$
#Conn. Sinks	5469	5547	5592	5645	6038	5725	5842	5779	5896	5864	5965	5813	5897
%Conn. Sinks	90.6	91.9	92.6	93.5	100	94.8	96.8	95.7	97.6	97.1	98.8	96.3	97.7
WL (meters)	42.26	41.99	42.21	42.22	47.89	45.18	47.80	44.48	47.66	44.48	47.90	45.33	47.51
WL/Sink (μm)	7727	7570	7548	7479	7931	7891	8182	7697	8083	7585	8031	7798	8057
#Buffers	9240	9053	9076	9037	10330	9860	10676	9591	10610	9546	10730	9860	10647
#Buffers/Sink	1.69	1.63	1.62	1.60	1.71	1.72	1.83	1.66	1.80	1.63	1.80	1.70	1.81

RTL planning with aggressive global wire optimization.[6] We are presently extending our approach to achieve better handling of timing criticality and budgets by improved use of net ordering and weighting during rounding and post-processing of the solution to eliminate unneeded repeaters.

Further, we seek practical algorithms for handling routing congestion, i.e., simultaneously enforcing buffer block *and* channel capacities. By inserting "virtual" nodes corresponding to channels, the problem becomes a generalized type of integer GMTMCF in a vertex capacitated graph. However, computing minimum-weight feasible Steiner trees in this graph now entails finding minimum-weight *length-restricted* paths between buffer blocks. Although the latter problem is NP-hard, it can be approximated arbitrary close [11], [18] and it is still possible to apply our GMTMCF schema.

Finally, we note that actual applications would likely iterate the GRBB solution with incremental modification of buffer block locations, pin placements, and both channel and buffer block capacities. Adapting the GMTMCF approach and its runtime/quality profile for use in an iterative environment is challenging and is the subject of ongoing collaboration with industry.

[6]In other words, maximal repeater insertion allows maximum timing budgets for within-block timing paths and this permits blocks to go through synthesis, place and route with more aggressive area targets. A strategy of uniform buffering of as many global nets as possible also helps control signal integrity and delay uncertainty problems.

REFERENCES

[1] C. Albrecht, "Provably good global routing by a new approximation algorithm for multicommodity flow," in *Proc. ISPD*, 2000, pp. 19–25.
[2] R. C. Carden and C.-K. Cheng, "A global router using an efficient approximate multicommodity multiterminal flow algorithm," in *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*, vol. 33, pp. 73–91, 1999.
[4] J. Cong, L. He, C.-K. Koh, and P. H. Madden, "Performance optimization of VLSI interconnect layout," *Integration*, vol. 21, pp. 1–94, 1996.
[5] J. Cong, T. Kong, and D. Z. Pan, "Buffer block planning for interconnect-driven floorplanning," in *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," in *Proc. ICCAD*, 2000, pp. 104–109.
[7] ——, "Provably good global buffering by multiterminal multicommodity flow approximation," in *Proc. ASP-DAC*, 2001, pp. 120–125.
[8] ——, "Practical approximation algorithms for separable packing linear programs," in *Proc. 7th Int. Workshop Algorithms and Data Structures*, vol. 2125, Lecture Notes in Computer Science, 2001, pp. 120–125.
[9] L. K. Fleischer, "Approximating fractional multicommodity flow independent of the number of commodities," in *Proc. 40th Ann. Symp. Foundations of Computer Science*, 1999, pp. 24–31.
[10] N. Garg and J. Könemann, "Faster and simpler algorithms for multicommodity flow and other fractional packing problems," in *Proc. 39th Ann. Symp. Foundations of Computer Science*, 1998, pp. 300–309.

[11] R. Hassin, "Approximation schemes for the restricted shortest path problem," *Math. Oper. Res.*, vol. 17, pp. 36–42, 1992.
[12] J. Huang, X.-L. Hong, C.-K. Cheng, and E. S. Kuh, "An efficient timing-driven global routing algorithm," in *Proc. DAC*, 1993, pp. 596–600.
[13] A. B. Kahng, S. Muddu, E. Sarto, and R. Sharma, "Interconnect tuning strategies for high-performance IC's," in *Proc. DATE*, 1998.
[14] J. Lillis, C. K. Cheng, and T. T. Y. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model," in *Proc. ICCAD*, 1995, pp. 138–143.
[15] R. Motwani, J. Naor, and P. Raghavan, "Randomized approximation algorithms in combinatorial optimization," in *Approximation Algorithms for NP-Hard Problems*, D. Hochbaum, Ed. Boston, MA: PWS, 1997, pp. 144–191.
[16] A. P.-C. Ng, P. Raghavan, and C. D. Thomson, "Experimental results for a linear program global router," *Computers Artificial Intelligence*, vol. 6, pp. 229–242, 1987.
[17] T. Okamoto and J. Cong, "Buffered Steiner tree construction with wire sizing for interconnect layout optimization," in *Proc. ICCAD*, 1996, pp. 44–49.
[18] C. A. Phillips, "The network inhibition problem," in *Proc. 25th Ann. ACM Symp. Theory of Computing*, 1993, pp. 776–785.
[19] P. Raghavan and C. D. Thomson, "Randomized rounding," *Combinatorica*, vol. 7, pp. 365–374, 1987.
[20] ——, "Multiterminal global routing: A deterministic approximation scheme," *Algorithmica*, vol. 6, pp. 73–82, 1991.
[21] E. Shragowitz and S. Keel, "A global router based on a multicommodity flow model," *Integration*, vol. 5, pp. 3–16, 1987.
[22] X. Tang and D. F. Wong, "Planning buffer locations by network flows," in *Proc. ISPD*, 2000, pp. 180–185.

Feodor F. Dragan received the M.S. degree in applied mathematics from Moldova State University, in 1985, and the Ph.D. degree in theoretical computer science from the Belorussian Academy of Sciences, in 1990.

He was an Assistant and then an Associate Professor at the Mathematics and Computer Science Department of Moldova State University from 1988 to 1999. From 1994 to 1999, he was on leave of absence and worked in Germany as a Research Associate on a Volkswagen Foundation (VW) project and on a German Research Community (DFG) project. During 1999 to 2000, he was a Research Associate at the Computer Science Department of University of California, Los Angeles. Since August 2000 he has been an Assistant Professor of Computer Science at Kent State University, Ohio. He has authored more than 60 refereed scientific publications. His research interests include design and analysis of algorithms, algorithmic graph and hypergraph theory, computational geometry, VLSI CAD, and combinatorial optimization.

Dr. Dragan was also awarded a DAAD Research Fellowship (Germany) from 1994 to 1995.

Andrew B. Kahng received the A.B. degree (in applied mathematics/physics) from Harvard College and the M.S. and Ph.D. degrees (computer science) from the University of California at San Diego.

He was with the UCLA computer science department from 1989 to 2000, most recently as Professor and Vice-Chair. Since January 2001 he has been a Professor of CSE and ECE at the University of California San Diego. He has published over 160 papers in the VLSI CAD literature, centering on physical layout and performance analysis. His research interests include VLSI physical layout design and performance analysis, combinatorial and graph algorithms and stochastic global optimization.

Dr. Kahng also received the National Science Foundation Young Investigator award and a Design Automation Conference Best Paper award. He was the founding General Chair of the ACM/IEEE International Symposium on Physical Design, Co-Founder of the ACM Symposium on System-Level Interconnect Prediction and since 1997 has defined the physical design roadmap for the SIA International Technology Roadmap for Semiconductors. He is currently Chair of the U.S. Design Technical Working Group for the 2001 ITRS and technical program chair of EDP-2001 (the Electronic Design Processes symposium of the IEEE DATC). He is also on the steering committees of ISPD-2001 and SLIP-2001.

Ion I. Măndoiu received the M.S. degree in Computer Science from Bucharest University in 1992 and the Ph.D. degree in Computer Science from Georgia Institute of Technology in 2000.

He was a Research Assistant at the Computer Science Department of Bucharest University between 1992 and 1995, a Research Assistant and Instructor at the College of Computing, Georgia Institute of Technology, Atlanta, between 1995 and 2001, and a Postgraduate Researcher at the Computer Science Department of the University of California, Los Angeles, between 2000 and 2001. Since August 2001 he has been a Visiting Graduate Researcher in the VLSI CAD Group lead by Prof. Kahng at the University of California San Diego. He has authored more than 20 refereed scientific publications. His research interests include the design, analysis, and implementation of exact and approximation algorithms, VLSI physical layout design, combinatorial optimization, coding theory, and algorithmic information theory.

Sudhakar Muddu received the B.S. degree in electronics and communications engineering from Indian Institute of Technology, Madras, India and the M.S./Ph.D. degrees in computer science from the University of California at Los Angeles.

He is President and Founder of Sanera Systems, Inc. He has more than ten years of experience in the areas of VLSI Design and Network architecture/protocols. Before founding Sanera he worked at SGI as a Lead Designer in the high-performance processor playing a key role in the design of MIPS R100000, R12000, R14000 microprocessors and ORIGIN series systems. Prior to SGI, he also worked at AT&T Bell Laboratories, Holmdel, as an architect on ATLANTA protocol-independent networking chipsets and switching technologies and at IBM T. J. Watson Research Center, Yorktown Heights, in the next-generation VLSI group and at Intel Corporation, Santa Clara, significatly contributing in the Pentium and Pentium Pro Architecture and Performance Modeling groups. He holds five patents in addition to the ten patents pending in the areas of network architecture and VLSI design/architecture. He has published over 30 papers in the above areas.

Dr. Muddu has served on the technical program committees for the IEEE International Conference on Computer Design (1999–2000), IEEE International ASIC/SoC Conference (1998–2000), IEEE Symposium on IC/Package Design Integration (1998), IEEE Multi-Chip Module Conference (1997), and XI Symposium on Integrated Circuit Design (1998–2000). He was also nominated as a session chair for the IEEE Circuits and Systems Conference in 1998.

Alexander Zelikovsky received the Ph.D. degree in computer science from the Institute of Mathematics from the Belorussian Academy of Sciences in Minsk, Belarusi, in 1989.

He worked at the Institute of Mathematics in Kishinev as a Senior Research Scholar from 1989 to 1995. Between 1992 and 1995 he visited Bonn University and the Institut fur Informatik in Saarbrueken, Germany. He was a Research Scientist at Virginia University (1995–1997) and a Postdoctoral Scholar at UCLA (1997–1998). Since January 1999 he has been an Assistant Professor in the Computer Science Department of Georgia State University. He is the author of more than 50 refereed publications. His research interests include VLSI physical layout design and performance analysis, design for manufacturing, discrete and approximation algorithms, combinatorial optimization, and computational geometry.