Hindawi Publishing Corporation
VLSI Design
Volume 2011, Article ID 892310, 9 pages
doi:10.1155/2011/892310

Research Article

Efficient Congestion Mitigation Using Congestion-Aware Steiner Trees and Network Coding Topologies

M. A. R. Chaudhry, Z. Asad, A. Sprintson, and J. Hu

Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843, USA

Correspondence should be addressed to M. A. R. Chaudhry, masadch@tamu.edu

Received 26 December 2010; Accepted 5 February 2011

Academic Editor: Zhuo Li

In the advent of smaller devices, a significant increase in the density of on-chip components has raised congestion and overflow as critical issues in VLSI physical design automation. In this paper, we present novel techniques for reducing congestion and minimizing overflows. Our methods are based on ripping up nets that go through the congested areas and replacing them with *congestion-aware* topologies. Our contributions can be summarized as follows. First, we present several efficient algorithms for finding *congestion-aware* Steiner trees that is, trees that avoid congested areas of the chip. Next, we show that the novel technique of *network coding* can lead to further improvements in routability, reduction of congestion, and overflow avoidance. Finally, we present an algorithm for identifying efficient congestion-aware network coding topologies. We evaluate the performance of the proposed algorithms through extensive simulations.

1. Introduction

In almost any VLSI design flow, global routing is an essential stage that determines the signal interconnections. Therefore, the capability of the global router may significantly affect the design turn-around time. Moreover, the results of the global routing stage impact many circuit characteristics, such as power, timing, area, and signal integrity. Global routing poses major challenges in terms of the efficient computation of quality routes. In fact, most of the global routing problems, even special cases, tend to be NP complete [1, 2].

In the advent of smaller devices, a significant increase in the density of on-chip components results in a larger number of nets that need to be routed, which, together with more stringent routing constraints, results in increasing congestion and overflow. In this paper, we propose novel techniques for congestion avoidance and overflow reduction. Our methods are designed for the rip-up-and-reroute phase of the global routing stage. At this stage, all the nets have already been routed using a standard prerouting technique however some of the nets need to be rerouted due to high congestion and overflow. Our methods are based on ripping up nets that go through congested areas and replacing them with congestion-aware topologies. The proposed techniques facilitate even distribution of the routing load along the available routing areas. We propose efficient algorithms for finding congestion-aware Steiner trees that favor uncongested routes. In addition, we use the novel technique of *network coding* for further reduction of congestion and overflow avoidance.

1.1. Congestion-Aware Steiner Trees. The major goal of congestion-aware Steiner tree routing is to find a tree that connects the required set of nodes (pins of a net) while avoiding congested areas with a minimum penalty in terms of the total wirelength. In addition, the running time of the routing algorithm should scale well with the growing number of nets. These requirements pose several challenges in terms of the algorithm design. The first challenge is to select a cost function that adequately captures the local congestion conditions at the edges of the routing graph. Next, the algorithm should find a minimum cost tree within acceptable running time. Since finding a Steiner tree is an NP-complete problem, the algorithm needs to use an approximation scheme or employ a heuristic approach. Finally, the proposed algorithm should ensure that the overall performance of the rip-up-and-reroute phase is satisfactory

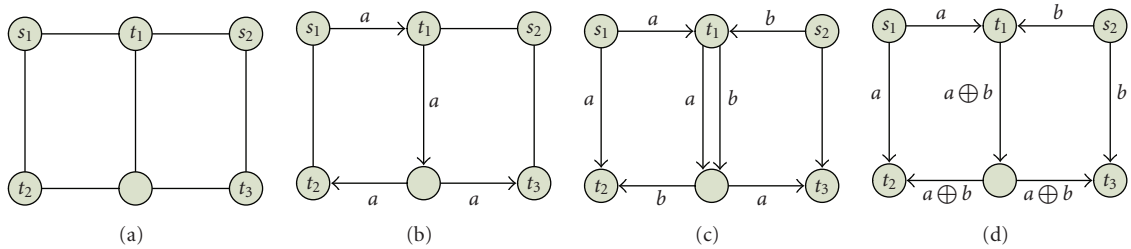

FIGURE 1: (a) The underlying routing graph with two source nodes s_1, and s_2 and three terminals t_1, t_2, t_3. (b) A rectilinear Steiner tree that connects source s_1 to all terminals. (c) Two rectilinear Steiner trees that connect sources s_1 and s_2 to all terminals. (d) A network coding solution.

in terms of congestion mitigation and overflow reduction. In this paper, we evaluate several cost functions which take into account various factors such as wire density, overflow, and congestion history. We propose several efficient algorithms for Steiner tree routing and compare their performance. Our algorithms are based on known approximations for the Steiner tree problem, heuristic methods, and artificial intelligence techniques.

1.2. Network Coding. The basic idea of the *network coding* technique [3] is to enable the intermediate nodes to generate new signals by combining the signals arriving over their incoming wires. This is in contrast to the standard approach, in which each node can only forward or duplicate the incoming signals.

For example, consider a routing instance depicted in Figure 1(a). In this example, we need to route two nets, one connecting source s_1 with terminals t_1, t_2, and t_3 and the other connecting source s_2 with the same set of terminals. The underlying routing graph is represented by a grid as shown in Figure 1(a). Suppose that due to congestion, each edge of this graph has a residual capacity of one unit, that is, each edge can accommodate only a single wire. It is easy to verify that using traditional Steiner tree routing, only one net can be routed without an overflow. For example, Figure 1(b) shows a possible routing of a net that connects s_1 with terminals t_1, t_2, and t_3. In contrast, Figure 1(c) shows that routing of both nets results in an overflow. In this example, two nets transmit different signals, a and b, over separate Steiner trees. Figure 1(d) shows that the network coding approach allows to route both nets without overflows. With this approach, the terminal t_1 creates a new signal, $a \oplus b$, which is delivered to terminals t_2 and t_3, while the signals a and b are delivered to terminals t_2, and t_3 directly. It is easy to verify that each terminal can decode the two original symbols a and b.

The network coding technique offers two distinct advantages. First, it has a potential of solving difficult cases that cannot be solved using traditional routing. For example, for the routing instance shown in Figure 1, the traditional routing results in an overflow of value 1, whereas with the network coding technique there are no overflows. Second, the network coding technique can decrease the total wirelength. For example, in the routing instance shown in Figure 1 the total wirelength for the traditional routing solution is 8, whereas for the network coding solution, the total wirelength is 7.

1.3. Previous Work. In the past decades, researchers have strived to improve the performance of global routing algorithms (see, e.g., [4–6] and references therein). To handle the complexity of large-scale global routing, multilevel routing techniques are proposed in [7, 8]. Recently proposed BoxRouter [9, 10] is based on progressive integer linear Programming (ILP) and rip-up-and-reroute techniques. A fast routing method is presented in [11]. Reference [12] proposes an approach based on the Lagrangian multiplier technique. An effective edge shifting technique is presented in [13]. Most of these previous works adopt the rip-up-and-reroute strategy. However, they usually reroute one path (i.e., a 2-pin connection) at a time. In contrast, our method reroutes entire multipin nets. We also propose to use network coding techniques to further reduce congestion and eliminate overflows.

1.4. Contribution. The paper makes the following contributions. First, we propose several algorithms for finding efficient congestion-aware Steiner trees. Second, we show that the novel technique of network coding can lead to further improvements in routability, reduction of congestion, and overflow minimization. Finally, we provide an algorithm for identifying efficient congestion-aware network coding topologies. We evaluate the performance of the proposed algorithms through extensive simulations.

2. Model

In this paper, we adopt the most commonly used global routing model. The routing topology is represented by a grid graph $G(V, E)$, where each node $v \in V$ corresponds to a global routing cell (GCell) [10, 12] and each routing edge $e \in E$ corresponds to a boundary between two adjacent GCells. A set $S = \{n_1, n_2, \ldots, n_{|S|}\}$ of nets are to be routed on this graph. Each net $n_i \in S$ connects a source node s_i with terminal nodes $T_i = \{t_1, t_2, \ldots, t_{|T_i|}\}$. If there is a wire connection between two adjacent GCells, the wire must cross their boundary and utilize the corresponding routing edge. Each routing edge $e \in E$ has a certain routing capacity $c(e)$ which determines the number of wires that can pass through

this edge. We denote by $\eta(e)$ the number of wires that are currently using edge e.

2.1. Global Routing Metric. The goal of a global router is to minimize congestion. Some of the important metrics for a global router are defined as follows.

(i) *Overflow.* For each edge $e \in E$, the overflow $\text{ov}(e)$ of e is defined as

$$\text{ov}(e) = \begin{cases} \eta(e) - c(e), & \text{if } \eta(e) > c(e), \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

The maximum overflow $\text{ov}_{\max}$ is defined as

$$\text{ov}_{\max} = \max_{e \in E} \text{ov}(e). \quad (2)$$

Total overflow ov_{tot} is defined as

$$\text{ov}_{\text{tot}} = \sum_{e \in E} \text{ov}(e). \quad (3)$$

(ii) *Wirelength.* Total wirelength wlen is defined as

$$\text{wlen} = \sum_{e \in E} \eta(e). \quad (4)$$

(iii) *Density.* The density $d(e)$ of edge $e \in E$ is defined as

$$d(e) = \frac{\eta(e)}{c(e)}. \quad (5)$$

2.2. Cost Functions. Our algorithms associate each edge $e \in E$ in the graph with a cost function $\rho(e)$ which captures its congestion and overflow. The cost of the tree is defined as the sum of the costs of all of its edges. Our goal is to identify trees that go through congested areas and replace them by Steiner trees or network coding topologies that go through areas with low congestion.

In this work, we consider several cost functions, described below.

Polynomial Cost Function. We propose a cost assignment function, where the cost of an overflowed edge is a polynomial function of the sum of its density and overflow. Formally, our proposed cost function is defined as follows:

$$\rho(e) = (d(e) + \text{ov}(e))^{\alpha}, \quad (6)$$

where α is a constant which determines the relative penalty for the congested edges.

Exponential Cost Function. We use the cost assignment function proposed by [12]. With this cost assignment, the cost of an edge is an exponential function of its density

$$\rho(e) = \begin{cases} \text{Exp}(\beta \cdot (d(e) - 1)) & \text{if } d(e) > 1, \\ d(e) & \text{otherwise,} \end{cases} \quad (7)$$

where β is a constant which determines the penalty for overflowed edges.

History-Based Cost Function. This cost function assigns cost to an edge based on its congestion history [10, 12]. Specifically, each edge is associated with a parameter h_e that specifies the number of times the edge has been overflowed during the previous iterations of the algorithm. That is, each time the edge with an overflow is used, the parameter h_e is incremented by one. Then, the modified cost $\rho'(e)$ of the edge is defined as follows:

$$\rho'(e) = 1 + h_e \cdot \rho(e). \quad (8)$$

Here, $\rho(e)$ is either the polynomial cost function (6) or exponential cost function (7). If the density of the edge is less than or equal to one, the parameter h_e is initially set to zero.

Since we focus on the rerouting phase, we assume that for each net $n_i \in S$, there exists a Steiner tree ϕ_i which connects all nodes in n_i. Given a set of trees $\{\phi \mid n_i \in S\}$, we can determine the values of $\text{ov}(e), \eta(e)$ for each edge $e \in E$ and identify the set of *congested nets* $S' \subseteq S$. A net $n_i \in S'$ is referred to as *congested* if its Steiner tree ϕ_i has at least one edge with overflow.

We propose a two-phase solution for rerouting congested nets using congestion-aware topologies. In the first phase we iteratively rip up each net $n_i \in S'$ and reroute it using a congestion-aware Steiner tree with the goal of minimizing the maximum overflow $\text{ov}_{\max}$ and the total overflow ov_{tot}. In second phase, we deal with the nets that remain congested at the end of the first phase and rip-up-and-reroute pairs of congested nets using congestion-aware network coding topologies to further reduce congestion and minimize the number of overflows. Note that the nets considered in phase two correspond to difficult cases, where congestion avoidance was not possible even after several attempts of ripping up and rerouting *individual* nets. Therefore in the second phase, we consider the *pairs* of congested nets for further improvement. The example given in Figure 1 shows the advantage of using network coding topologies for routing pairs of nets over using standard routing techniques that handle each net separately.

3. Congestion-Aware Steiner Trees

In this section, we present several techniques for finding congestion-aware Steiner trees. Our goal is to find Steiner trees that minimize congestion with a minimum penalty in terms of the overall wirelength. We would like to achieve better tradeoffs between congestion mitigation and total wirelength. These tradeoffs are useful for practical applications as in some cases, congestion mitigation is preferable to wirelength reduction, whereas in other cases, the wirelength reduction is of higher priority.

3.1. Previous Work on Steiner Tree Routing. The Steiner tree problem is a well-studied NP-complete problem [14]. There is a wealth of heuristic and approximate solutions that have been developed for this problem. The best-known approximation algorithm has an approximation ratio of 1.55 (i.e., the cost of the tree returned by the algorithm is less than 1.55 times the optimum) [15]. The best known

approximations require significant computation time, so we focus on computationally feasible and easy to implement approximation and heuristic solution for constructing Steiner trees.

3.2. Algorithms for Finding Congestion-Aware Trees. As mentioned above, our goal is to rip up and reroute nets that use congested edges of $G(V, E)$. For each net $n_i \in S$ which has been ripped up, we need to find an alternative Steiner tree that uses uncongested routes. In this section, we describe five algorithms for finding congestion-aware Steiner trees. The first three algorithms use combinatorial techniques (see, e.g., [1, 16, 17]) while, the last two are based on the intelligent search techniques [18]. The performance of the algorithms is evaluated in Section 5.

Algorithm stTree1. This algorithm approximates a minimum cost Steiner tree by using a *shortest path tree*. A shortest path tree is a union of the shortest paths between source s_i and a set of terminals T_i. A shortest path tree can be identified by a single invocation of Dijkstra's algorithm. However, the cost of the tree may be significantly higher than the optimum.

Algorithm stTree2. This algorithm constructs the tree in an iterative manner. We iteratively process the terminals in T_i in the increasing order of their distance form s_i. More specifically, we first find a shortest path P_1 between source s_i and terminal t_1. Then, we assign a zero cost to all edges that belong to P_1 and find a shortest path P_2 between s and t_2 with respect to modified costs. The idea behind this algorithm is to encourage sharing of the edges between different paths. That is, if an edge e belongs to P_1, it can be used in P_2 with no additional cost. In general, when finding a shortest path to terminal t, all edges that belong to paths of previously processed terminals are assigned a zero cost. This algorithm requires $|T| - 1$ iterations of Dijkstra's algorithm, but it typically returns a lower-cost tree than Algorithm *stTree1*.

Algorithm stTree3. This is a standard approximation algorithm with the approximation ratio of 2 (i.e., the cost of the tree returned by the algorithm is at most two times higher than the optimal cost). Specifically, with this algorithm, we find a shortest path between each pair of nodes in the set $\overline{T} = \{T \cup \{s_i\}\}$. Then, we construct a complete graph G' such that each node in G' corresponds to a node in $\overline{T}$. The weight of an edge $e \in G'$ is equal to the minimum length of the path between two corresponding nodes in $\overline{T}$. The algorithm then finds a minimum spanning tree ϕ in G'. Next, each edge in ϕ is substituted by the corresponding shortest path in G, which results (after removing redundant edges) in a Steiner tree in G that connects source s_i with terminals in T_i.

Algorithm stTree4. Algorithm $stTree4$ is an intelligent search-based algorithm. Our approach is inspired by Algorithm A^*. Algorithm A^* is a shortest path algorithm that uses a heuristic function $\lambda(v)$ to determine the order of visiting nodes of the graph in order to improve its running time. Specifically, for each node v, we define $\lambda(v)$ to be the maximum distance between node v and a terminal $t \in T_i$ which has not yet been visited. The distance between v and $t \in T_i$ is defined as the minimum number of hops that separate v and t in G. The Algorithm $stTree4$ follows the same steps as Algorithm $stTree1$, but it uses Algorithm A^* with heuristic function $\lambda(v)$ to find shortest paths.

Algorithm stTree5. Algorithm $stTree5$ is also based on Algorithm A^*. It follows the same steps as Algorithm $stTree2$, but it uses Algorithm A^* with the same heuristic function $\lambda(v)$ as in Algorithm $stTree4$.

4. Network Coding Techniques

In this section, we use the network coding techniques in order to achieve further improvement in terms of minimizing congestion and reducing the number of overflows. The network coding technique enables, under certain conditions, to share edges that belong to different nets. For example, in the graph depicted in Figure 1(c), there are two minimum Steiner trees one transmitting signal a from source s_1 and the second transmitting signal b from source s_2. These two trees clash at the middle edge (emanating from t_1), resulting in an overflow. This conflict can be resolved by coding at node t_1, which effectively allows two trees to share certain edges. Similarly, our algorithm will identify pairs of nets that share terminals and then apply network coding techniques to reduce overflow.

4.1. Previous Work on Network Coding. The problem of routing of multiple nets with shared terminals is related to the problem of establishing efficient multicast connections in communication networks. The network coding technique was proposed in a seminal paper by Ahlswede et al. [3]. It was shown in [3, 19] that the capacity of the multicast networks, that is, the number of packets that can be sent simultaneously from the source node s to all terminals, is equal to the minimum size of a cut that separates s from each terminal. Li et al. [19] proved that *linear network codes* are sufficient for achieving the capacity of the network. In a subsequent work, Koetter and Médard [20] developed an algebraic framework for network coding. This framework was used by Ho et al. [21] to show that linear network codes can be efficiently constructed through a randomized algorithm. Jaggi et al. [22] proposed a deterministic polynomial-time algorithm for finding feasible network codes in multicast networks. An initial study of applicability of network coding for improving the routability of VLSI designs appears in [23]. In [24], Gulati and Khatri used network coding for improving the routability of FPGAs, focusing on finding nets that are suitable for network coding. To the best of our knowledge, this is the first work that proposes efficient algorithms for finding the congestion-aware network coding topologies for VLSI circuits.

4.2. Network Coding Algorithm. We proceed to present the algorithm we use for constructing congestion-aware coding networks that reduce congestion and overflow.

Algorithm NC $(G, s_i, s_j, T_{ij}, T'_i, T'_j)$:
(1) Find a Steiner tree ϕ_i connects s_i to terminals in T'_i
(2) Find a Steiner tree ϕ_j connects s_j to terminals in T'_j
(3) For each link $e \in T'_i \cup T'_j$ update $\eta(e)$
(4) Find a Steiner tree $\hat{\phi}$ that connects s_i to terminals in T_{ij}
(5) Assign zero cost to all edges in $\hat{\phi}$
(6) For each terminal $t \in T_{ij}$ let $d(t)$ be the shortest distance between s_j and t
(7) For all terminals $t \in T_{ij}$ in the increasing order of $d(t)$ do:
(8) Let $G'(V', E')$ be a graph formed from $G(V, E)$ by reversing all edges in $P_{i,t}$, where $P_{i,t}$ is a path from s_i to t in $\hat{\phi}$
(9) Find shortest path $P_{j,t}$ from source s_j to terminal t in $G'(V', E')$
(10) Assign zero cost to all edges of $P_{j,t}$
(11) **For** each edge $e(v, u) \in P_{j,t}$ **do**
(12) If there exists an edge $e'(u, v) \in \hat{\phi}$, remove $e'(u, v)$ from $\hat{\phi}$
(13) Otherwise, add $e(v, u)$ to $\hat{\phi}$
(14) Return $\hat{\phi}$, ϕ_i, and ϕ_j

Algorithm 1: Algorithm *NC*.

The algorithm includes the following steps. First, we identify the subset S' of S that includes nets that go through edges with overflow. Second, we identify pairs of nets in S' that share at least three common terminals. Next, we check, for each such pair of nets (n_i, n_j), whether we can replace the Steiner trees for n_i and n_j by a more efficient routing topology with respect to congestion and overflow.

More specifically, let (n_i, n_j) be a pair of nets in S' that share at least three terminals. Let s_i and s_j be the source nodes of these nets. We denote the set of terminals shared by n_i and n_j by T_{ij}. We also denote by T'_i the set of terminals in T_i that do not belong to T_{ij} that is, $T'_i = T_i \setminus T_{ij}$. Similarly, we denote by T'_j the set of terminals in T_j that do not belong to T_{ij} that is, $T'_j = T_j \setminus T_{ij}$. Next, we find two congestion-aware Steiner trees ϕ_i and ϕ_j that connect s_i to T'_i and s_j to T'_j. These trees can be identified by one of the algorithms presented in Section 3. The parameter $\eta(e)$ for each $e \in G$ is updated after finding ϕ_i and ϕ_j.

Finally, we find a congestion-aware network coding topology $\hat{\phi}$ that connects s_i and s_j to the common set of terminals T_{ij} in an iterative manner. First, we let $\hat{\phi}$ be a Steiner tree with source s_i and terminals T_{ij}. All edges of $\hat{\phi}$ are always assigned zero cost. We then sort the terminals T_{ij} in the increasing order of their distance (in the original graph) from s_j and process them in that order. For each terminal $t \in T_{ij}$, we reverse all the edges in the path $P_{i,t}$ between source s_i and terminal t and find a shortest path $P_{j,t}$ between source s_j and terminal t. Then, for each link $e(v, u) \in P_{j,t}$ we perform the following procedure. If there exists a link $e'(u, v)$ in $\hat{\phi}$, we remove $e'(u, v)$ from $\hat{\phi}$ otherwise, we add $e(v, u)$ to $\hat{\phi}$. A sample execution of this procedure is shown in Figure 2. It is easy to verify that the algorithm produces a feasible network coding topology that is, a topology that ensures that for each terminal $t \in T_{ij}$ there are two-edge disjoint paths that connect s_i and s_j with t. The formal description of algorithm for identifying the network coding topology, referred to as Algorithm *NC*, is given in Algorithm 1.

After the execution of the algorithm, we determine whether the total cost of $\hat{\phi}$, ϕ_i, and ϕ_j is less than the total cost of the original Steiner trees for nets n_i and n_j. If there is a reduction in terms of cost, the two original Steiner trees are replaced by $\hat{\phi}$, ϕ_i, and ϕ_j.

Our experimental results, presented in Section 5, show that the number of coding opportunities is relatively small. However, by applying the network coding technique on a limited number of nets, we can achieve a significant reduction in the number of overflows. Also, since the network coding technique is applied to a limited number of nets, the overhead in terms of the number of additional required gates is relatively small.

5. Performance Evaluation

We have evaluated the performance of our algorithms using the ISPD98 routing benchmarks [25]. All the experiments are performed on a 3.2 GHz Intel Xeon dual-core machine. In all experiments, we first run the Steiner tree tool *Flute* [26] in order to determine the initial routing of all nets in the benchmark. Next, we perform an iterative procedure, referred to as Phase 1, which processes each net with overflows and checks whether an alternative Steiner tree of lower cost and with smaller number of overflows exists, and if yes, it rips up the existing tree and replaces it with an alternative one. This phase uses one of the algorithms described in Section 3. Phase 1 terminates when four subsequent iterations yield the same cost and the number of overflows, indicating that further reduction in the number overflows is unlikely.

Next, we check whether the application of the network coding technique can further reduce the number of overflows. This phase is referred to as the Phase 2. We first identify pairs of nets that have overflowed edges and share at least there terminals. We then apply Algorithm *NC*, presented in Section 4, to find an alternative network coding topology and perform rip-up and reroute if such a topology is beneficial in terms of reducing congestion and eliminating overflows.

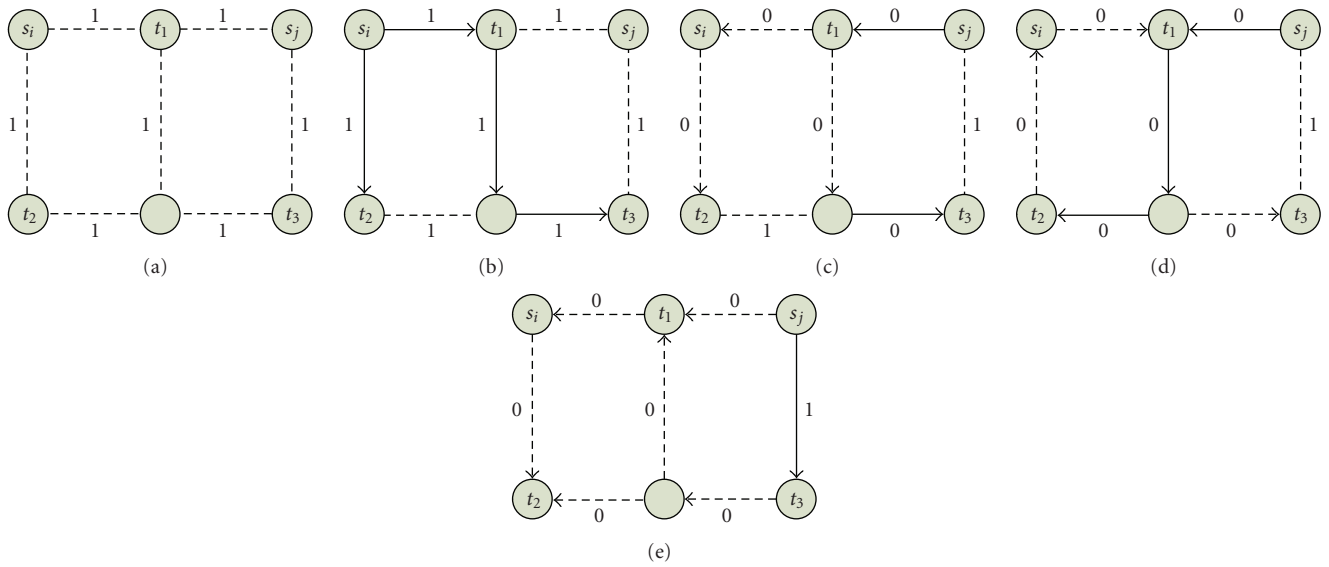

Figure 2: Steps for finding network coding topology using Algorithm *NC*. (a) A graph G with two nets n_i and n_j that connect nodes s_i and s_j to terminals $T_1 = T_2 = T_{12} = \{t_1, t_2, t_3\}$. (b) A Steiner tree $\hat{\phi}$ connecting s_j to T_2. (c) Modified graph in which the costs of all edges found in $\hat{\phi}$ are set equal to zero and the edges connecting s_i to t_1 are reversed. A shortest path from s_j to t_1 is shown. (d) Modified graph in which the edges connecting s_i to t_2 are reversed. A shortest path from s_j to t_2 is shown. (e) Modified graph in which the edges connecting s_i to t_3 are reversed. A shortest path from s_j to t_3 is shown.

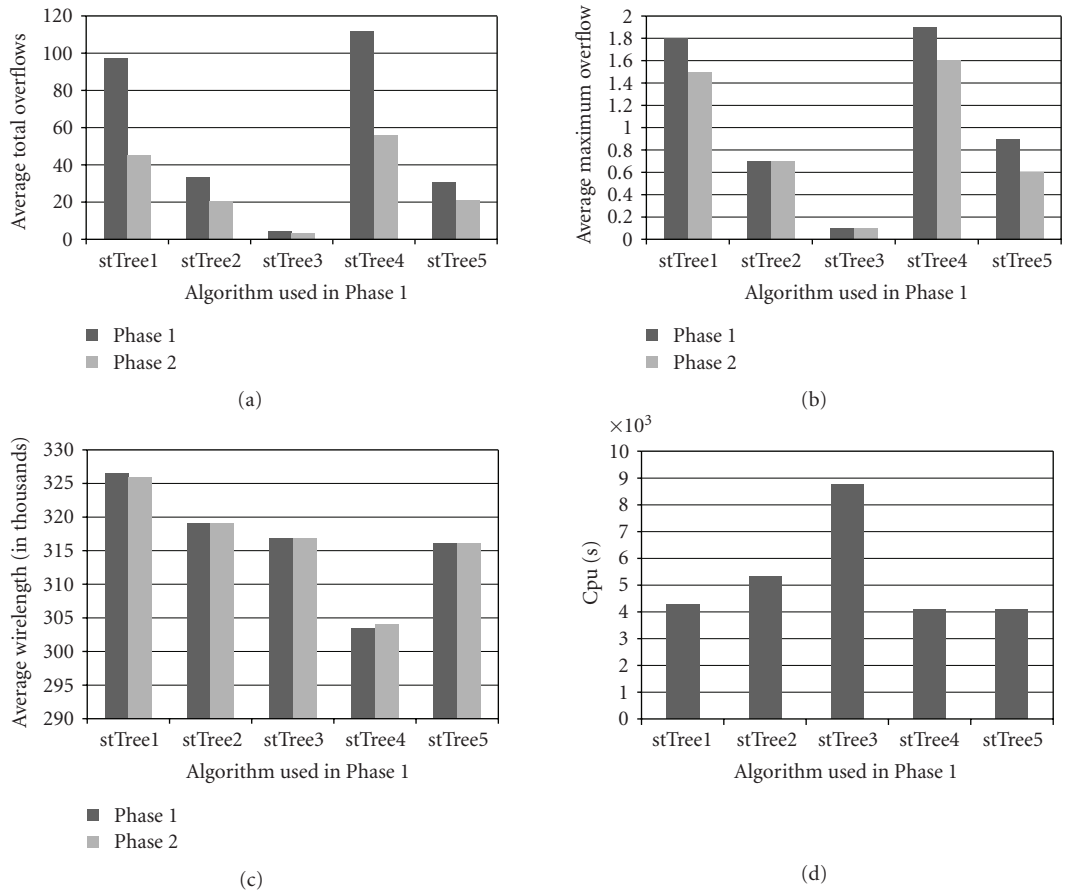

Figure 3: Evaluation of five different algorithms for Phase 1 and Algorithm *NC* for Phase 2 on ISPD98 benchmark files. In all experiments, the polynomial cost function is used. (a) Comparison of the average total overflow. (b) Comparison of the average maximum overflow. (c) Comparison of the average total wirelength. (d) Comparison of the average running time. Results present average over 10 ISPD98 benchmark files.

Table 1: Performance evaluation using Algorithm *stTree*3 for Phase 1 and Algorithm *NC* for Phase 2 on ISPD98 benchmarks with polynomial cost functions.

	Phase 1			Phase 2		
	ov_{tot}	ov_{max}	wlen	ov_{tot}	ov_{max}	wlen
ibm1	14	1	81058	6	1	80727
ibm2	0	0	216299	0	0	216299
ibm3	0	0	188391	0	0	188391
ibm4	125	2	196106	93	2	195949
ibm5	0	0	415681	0	0	415681
ibm6	0	0	336105	0	0	336105
ibm7	0	0	466381	0	0	466381
ibm8	0	0	477404	0	0	477404
ibm9	0	0	508661	0	0	508661
ibm10	0	0	711201	0	0	711201

Table 2: Performance evaluation of different cost functions (using Algorithm *stTree*3 for Phase 1 and Algorithm *NC* for Phase 2) on ISPD98 benchmarks.

	Polynomial cost		Exponential cost		History-based cost	
	ov_{tot}	ov_{max}	ov_{tot}	ov_{max}	ov_{tot}	ov_{max}
ibm1	0	0	117	1	0	0
ibm2	0	0	79	2	0	0
ibm3	0	0	0	0	0	0
ibm4	31	1	834	7	439	4
ibm5	0	0	0	0	0	0
ibm6	0	0	54	2	0	0
ibm7	0	0	82	1	0	0
ibm8	0	0	37	1	0	0
ibm9	0	0	15	1	0	0
ibm10	0	0	95	3	0	0

Table 3: Performance evaluation using Algorithm *stTree*3 for Phase 1 and Algorithm *NC* for Phase 2 on modified ISPD98 benchmarks (decreased both horizontal and vertical capacity by 1 unit) with polynomial cost function.

	Flute			Phase 1			Phase 2			Number of XOR gates
	ov_{tot}	ov_{max}	wlen	ov_{tot}	ov_{max}	wlen	ov_{tot}	ov_{max}	wlen	
ibm1	3257	14	60209	362	4	78161	184	3	77231	5
ibm2	5678	22	166193	18	1	217673	4	1	217737	12
ibm3	2308	16	145777	1	1	183395	0	0	183391	2
ibm4	4833	15	162844	550	6	193832	440	6	193131	52
ibm5	5	4	410038	0	0	476251	0	0	473265	70
ibm6	5492	27	276012	3	1	334165	1	1	334163	0
ibm7	4665	15	363678	1	1	473743	0	0	473743	2
ibm8	5400	13	403502	3	1	476880	0	0	476864	0
ibm9	8545	14	411524	16	1	496563	9	1	496409	24
ibm10	7103	20	574743	4	1	713569	1	1	713553	18

Table 4: Improvement over MaizeRouter for modified (congested) IBM benchmarks using Algorithm *stTree*3 for Phase 1 and Algorithm *NC* for Phase 2, using polynomial cost function.

	Ver. Cap	Hor. Cap	MaizeRouter		Phase 1 and Phase 2	
			ov_{tot}	ov_{max}	ov_{tot}	ov_{max}
ibm1	11	13	43	1	34	1
ibm5	21	31	45326	25	45151	25
ibm8	17	28	649	9	641	9
ibm9	10	24	4006	9	3989	9

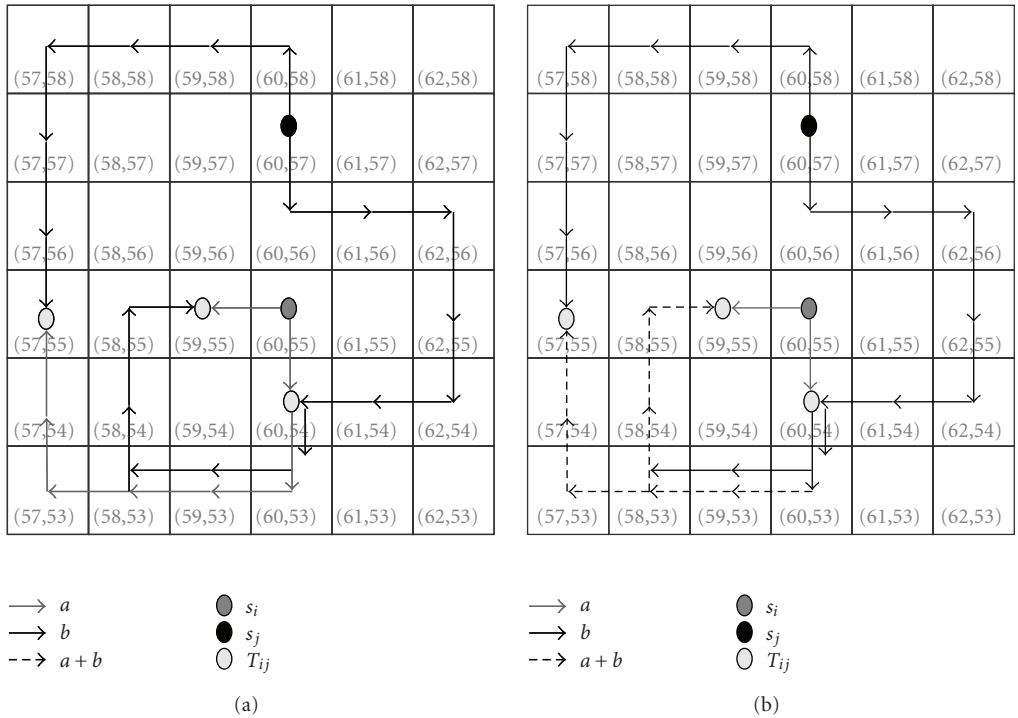

Figure 4: A network coding topology for benchmark ibm1 for pair of nets $n_i = 66$ and $n_j = 1531$ computed using Algorithm *NC*. There are two nets n_i and n_j with sources $s_i = (60, 55)$, and $s_j = (60, 57)$ and set of common terminals $T_{ij} = (59, 55), (60, 54)$, and$(57, 55)$. Part(a) shows routing layout without network coding. Part(b) shows routing layout with network coding. Example shows that network coding has helped to reduce congestion on edges $(60, 54)$, $(60, 53)$, $(59, 53)$, and $(58, 53)$.

The experimental results are shown in Figures 3(a)–3(c). The figures present average performance over all ten benchmarks. The cost function for this set of experiments was set according to (6) with $\alpha \geq 10$. We have observed that larger values of α yield fewer overflows but result in larger running times and increased wirelength. We also note that for Phase 1, Algorithm $stTree3$ shows the best performance in terms of reducing the total number of overflows as well as reducing the maximum overflow. In fact, Algorithm $stTree3$ eliminates all overflows in all benchmarks, except for ibm4 as given in Table 1. We also note that Algorithms $stTree4$ and $stTree5$ yield Steiner trees with a smaller total wirelength. This is due to the fact the intelligent search algorithms favor paths that have small hop count.

We observe that the network coding technique results in a considerable reduction of the total number of overflows as well as reduces the maximum overflow. Furthermore, for each pair of nets for which we perform network coding, the number of required gates is small. Moreover, in all cases that we have encountered, the network coding operations can be performed over finite field $GF(2)$, that is, each encoding node can be implemented with a single XOR gate. Such gates incur minimum overhead, because they can serve as buffers for long wires. An example of how network coding can help in reducing the wirelength of two nets in the ibm1 benchmark is shown in Figure 4.

Figure 3(d) compares the running times of the different Algorithms for Phase 1 and Algorithm *NC* for Phase 2. As expected, Algorithm $stTree4$ is one of the fastest algorithms, whereas Algorithm $stTree1$ and $stTree5$ have running times comparable to Algorithm $stTree2$. Moreover, Algorithms $stTree4$ and $stTree5$ are faster than their counterparts (Algorithms $stTree1$ and $stTree2$, resp.). This is due to the fact that intelligent search methods speed up the search by preferring nodes closer to the destination.

In the second set of experiments, we evaluated the performance of three cost functions mentioned in Section 2.2 on the ISPD98 benchmarks using Algorithm $stTree3$ for Phase 1 and Algorithm *NC* for Phase 2. For cost function given by (6), we used $\alpha \geq 10$, whereas for cost function given by (7), we used $\beta \geq 50$, and for cost function given by (8), $\rho(e)$ was a polynomial cost function with $\alpha \geq 10$. The results are shown in Table 2. Polynomial cost function showed the best performance in terms of overflows.

In another set of experiments, we worked on slightly modified ISPD98 benchmark files. The modification included reducing the vertical and horizontal capacity by one unit. For Phase 1, we used Algorithm $stTree3$ and then applied Algorithm *NC* in Phase 2. Polynomial cost function given by (6) was used to check the performance on these more congested cases. The results are given in Table 3.

We have also conducted the same experiment on several selected benchmarks on the output of MaizeRouter [13]. In these experiments, we iteratively reduced the vertical and horizontal capacity of the ISPD98 benchmarks until we got overflows while running them through MaizeRouter. Then,

we used the output of MaizeRouter as input to Phase 1 using Algorithm $stTree3$, and after that, we have applied Algorithm *NC* in Phase 2. The cost function used was polynomial with $\alpha = 10$. The results are shown in Table 4. The results demonstrate that Algorithms *stTree3* combined with Algorithm *NC* perform well and can contribute to further reduction of the number of overflows.

6. Conclusions

In this paper, we presented several efficient techniques for rip-up-and-reroute stage of the global routing process. Our techniques are based on ripping up nets that go through highly congested areas and rerouting them using efficient Steiner tree algorithms. We have considered several efficient Steiner tree algorithms as well as several cost functions that take into account congestion and overflow. We have also studied the application of the novel technique of network coding and showed that it can efficiently handle difficult routing cases, facilitating reduction in the number of overflows.

Acknowledgment

A preliminary version of this paper appeared in the proceedings of the 2009 IEEE Computer Society Annual Symposium on VLSI (ISVLSI) [27]. This work was supported in part by NSF grant CNS-0954153.

References

[1] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley & Sons, New York, NY, USA, 1990.

[2] N. A. Sherwani, *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers, Norwell, Mass, USA, 2nd edition, 1995.

[3] R. Ahlswede, N. Cai, S. Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[4] J. Hu and S. S. Sapatnekar, "A survey on multi-net global routing for integrated circuits," *Integration, the VLSI Journal*, vol. 31, no. 1, pp. 1–49, 2001.

[5] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh, "Predictable routing," in *Proceedings of IEEE/ACM International Conference on Computer Aided Design (ICCAD '00)*, pp. 110–113, 2000.

[6] C. Albrecht, "Provably good global routing by a new approximation algorithm for multicommodity flow," in *Proceedings of the International Symposium on Physical Design (ISPD '00)*, pp. 19–25, April 2000.

[7] J. Cong, J. Fang, and Y. Zhang, "Multilevel approach to full-chip gridless routing," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 396–403, November 2001.

[8] Y. W. Chang and S. P. Lin, "MR: A new framework for multilevel full-chip routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 5, pp. 793–800, 2004.

[9] M. Cho and D. Z. Pan, "BoxRouter: a new global router based on box expansion and progressive ILP," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 373–378.

[10] M. Cho, K. Lu, K. Yuan, and D. Z. Pan, "BoxRouter 2.0: architecture and implementation of a hybrid and robust global router," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD '07)*, pp. 503–508, November 2007.

[11] M. Pan and C. Chu, "FastRoute 2.0: a high-quality and efficient global router," in *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC '07)*, pp. 250–255, January 2007.

[12] J. A. Roy and I. L. Markov, "High-performance routing at the nanometer scale," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD '07)*, pp. 496–502, November 2007.

[13] M. D. Moffitt, "MaizeRouter: engineering an effective Global Router," in *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC '08)*, pp. 226–231, March 2008.

[14] M. R. Garey and D. S. Johnson, *Computers and Intractability*, Freeman, San Francisco, Calif, USA, 1979.

[15] G. Robins and A. Zelikovsky, "Improved Steiner tree approximation in graphs," in *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 770–779, January 2000.

[16] S. Voß, "Steiner's problem in graphs: heuristic methods," *Discrete Applied Mathematics*, vol. 40, no. 1, pp. 45–72, 1992.

[17] M. P. D. Aragão and R. F. Werneck, "On the implementation of MST-based heuristics for the steiner problem in graphs," in *Proceedings of the 4th International Workshop on Algorithm Engineering and Experiments*, pp. 1–15, 2002.

[18] R. Dechter and J. Pearl, "Generalized best-first search strategies and the optimality of A*," *Journal of the ACM*, vol. 32, no. 3, pp. 505–536, 1985.

[19] S. Y. R. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Transactions on Information Theory*, vol. 49, no. 2, pp. 371–381, 2003.

[20] R. Koetter and M. Médard, "An algebraic approach to network coding," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 782–795, 2003.

[21] T. Ho, R. Koetter, M. Médard, D. R. Karger, and M. Effros, "The benefits of coding over routing in a randomized setting," in *Proceedings of IEEE International Symposium on Information Theory (ISIT '03)*, p. 442, July 2003.

[22] S. Jaggi, P. Sanders, P. A. Chou et al., "Polynomial time algorithms for multicast network code construction," *IEEE Transactions on Information Theory*, vol. 51, no. 6, pp. 1973–1982, 2005.

[23] N. Jayakumar, S. P. Khatri, K. Gulati, and A. Sprintson, "Network coding for routability improvement in VLSI," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD '06)*, pp. 820–823, November 2006.

[24] K. Gulati and S. R. Khatri, "Improving FPGA routability using network coding," in *Proceedings of the 18th ACM Great Lakes Symposium on VLSI (GLSVLSI '08)*, pp. 147–150, March 2008.

[25] C. J. Alpert, "ISPD98 circuit benchmark suite," in *Proceedings of the International Symposium on Physical Design (ISPD '98)*, pp. 80–85, April 1998.

[26] C. Chu, "FLUTE: fast lookup table based wirelength estimation technique," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD '04)*, pp. 696–701, November 2004.

[27] M. A. R. Chaudhry, Z. Asad, A. Sprintson, and J. Hu, "Efficient rerouting algorithms for congestion mitigation," in *Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI '09)*, pp. 43–48, May 2009.