GRIP: Scalable 3D Global Routing Using Integer Programming

Tai-Hsuan Wu, Azadeh Davoodi
Department of Electrical and Computer Engineering

Jeffrey T. Linderoth
Department of Industrial and Systems Engineering

University of Wisconsin - Madison WI 53706
{twu3,adavoodi,linderoth}@wisc.edu

ABSTRACT

We propose GRIP, a scalable global routing technique via Integer Programming (IP). GRIP optimizes wirelength and via cost without going through a layer assignment phase. GRIP selects the route for each net from a set of candidate routes that are generated based on an estimate of congestion generated by a linear programming *pricing* phase. To achieve scalability, the original IP is decomposed into smaller ones corresponding to balanced rectangular subregions on the chip. We introduce the concept of a *floating terminal* for a net, which allows flexibility to route long nets going through multiple subregions. We also use the IP to plan the routing of long nets, detouring them from congested subregions. For ISPD 2007 benchmarks, we obtain 3.9% and 11.3% average improvement in wirelength and via cost for the 2D and 3D versions respectively, compared to the best results reported in the open literature.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids

General Terms

Algorithms, Design

Keywords

Global Routing, Integer Programming

1. INTRODUCTION

Design of Integrated Circuits in nanometer regime is subject to many obstacles such as manufacturability, variability, yield-loss and timing failures. With increasing design sizes and shrinking device geometries, the severity of many of these issues is impacted by the routing of interconnects. Global routing (GR) is the primary step of routing during which the net regions will be planned. It has increasingly gained significance in recent years due to its larger role on the above-mentioned issues. It is crucial that the GR generates a high-quality routing solution in a manageable runtime that scales well with the design size.

DAC 2009, July 26 - 31, 2009, San Francisco, California, USA.

Among the two-categories of *concurrent* [8], [20], [21], [15], [4], [3], [5] and *sequential* [19], [22], [6], [18], [17] global routers, the latter has been more successful in terms of the tradeoff between solution quality and execution runtime. Sequential approaches have much smaller runtime but rely on an ordering of the nets and applying rip-up and re-route.

Much attention has been given to sequential approaches because the concurrent ones are inherently more time consuming. The most recent concurrent approach is the IP-based BoxRouter [8]. BoxRouter is extremely fast, but it only considers L-shaped routes in the IP. Recently [15] proposes the use of a few more basic patterns for routing each net in a progressive congestion-driven IP formulation. Similarly, [15] also has the downside of only considering a limited number of pre-determined patterns in the IP formulation. This in turn requires applying complicated pre- and post-processing steps to generate a final solution [8]. Also recently [21] proposes a hierarchical IP formulation for GR. However, as we discuss, the major downside of any hierarchical GR is failure to effectively account for the impact of short nets. In this paper, we make the following contributions to overcome some of these challenges:

1. We propose an IP formulation that simultaneously minimizes wirelength and via cost, thereby skipping the traditional layer assignment phase. The IP works with 3D Steiner routes and heavily penalizes overflow.
2. Promising routes for each net are generated by a linear programming *pricing* phase that takes into account a measure of current congestion at each iteration. The IP decides among these many promising routes for each net while considering capacity constraints.
3. To achieve scalability, we decompose the chip area into rectangular subregions to achieve *balanced* and smaller-sized IPs and then effectively integrates their solutions. The execution runtime depends on the number of subregions, some of which can be processed in parallel.
4. We introduce the concept of *floating terminals* for a net. Floating terminals allow flexibility in routing long nets through subregions while remaining compatible with our pricing phase for candidate route generation. We also discuss a pricing procedure for planning the regions through which long nets will travel during the decomposition of the IP into subregions.

Compared to [3], our candidate routes are generated by varying a base Steiner route, considering congestion as weights in a grid-graph as well as the other candidate routes generated so far at each iteration of column generation.

In our simulation results, we achieve an average 11.3% improvement in total wirelength and via cost of 3D ISPD 2007 benchmarks, compared to the best result reported for each benchmark. This is due to the concurrent nature of our approach, the pricing phase for candidate route generation, and directly working with the 3D model of the problem.

The organization of the paper is as follow. In Section 2, we discuss the IP formulation and customized column generation procedure. In Section 3, we discuss IP decomposition, subregion extraction, long net planning, and subregion solution integration. Simulation results are in Section 4.

2. PRICE AND BRANCH FOR GR

The algorithm proposed for global routing is based on the (approximate) solution of a large-scale integer program (IP). The solution procedure begins with a column generation (pricing) phase, followed by branch-and-bound.

2.1 An Integer Program for the GR Problem

In a mathematical description of the global routing problem, we are given a grid-graph $G = (V, E)$ describing the network topology, a set of (multi-terminal) nets given by $\mathcal{N} = \{T_1, T_2, \ldots, T_N\}$, (with $T_i \subset V$), and edge capacities u_e and weights c_e $\forall e \in E$. Denote by $\mathcal{T}(T_i)$ the collection of *all* Steiner trees (routes) connecting the terminals in T_i, and let the parameter $a_{te} = 1$ if Steiner tree t contains edge $e \in E$, $a_{te} = 0$ otherwise. Define the binary decision variable x_{it} that is equal to 1 if and only if net T_i is routed with route $t \in \mathcal{T}(T_i)$. An integer program for the global routing problem can be written as

$$\min_{x,s} \sum_{i=1}^{N} \sum_{t \in \mathcal{T}(T_i)} c_{it} x_{it} + \sum_{i=1}^{N} M s_i \qquad \text{(ILP-GR)}$$

$$\begin{cases} \sum_{t \in \mathcal{T}(T_i)} x_{it} + s_i = 1 & \forall i = 1, \ldots, N \\ \sum_{i=1}^{N} \sum_{t \in \mathcal{T}(T_i)} a_{te} x_{it} \le u_e & \forall e \in E \\ x_{it} = \{0, 1\} & \forall i = 1, \ldots, N, \forall t \in \mathcal{T}(T_i), \\ s_i \ge 0 & \forall i = 1, \ldots, N. \end{cases}$$

The parameter c_{it} is the cost of route t for net T_i which is computed as the total length of the 3D route, $c_{it} = \sum_{e \ni t} c_e$, where the notation $e \ni t$ denotes that edge $e \in E$ is contained in route $t \in \mathcal{T}(T_i)$. The first set of equations in the model enforces the routing of each net. The decision variable s_i will be positive if net T_i cannot be routed, and the objective function trades off the total routing length with the number of nets that are routed. Typically M is chosen sufficiently large to ensure that all nets are routed. The second set of equations in the model ensure that the given edge capacities are not exceeded. The formulation (ILP-GR) has a number of appealing properties.

1. The exact properties of the route, such as topology and metal layer can be incorporated into the "cost" of a route. The objective is to minimize this cost. The formulation can thus handle the 3D GR problem to include both wirelength and via cost as the cost of a route. It then avoids a traditional layer-assignment phase which can be a source of sub-optimality.
2. The formulation does not require that the nets be *a priori* broken into two-terminal segments. Breaking nets before doing routing can be a significant source of sub-optimality in the resulting final routing [19]. We note that the final version of our proposed algorithm has some "net-breaking" to define subproblems for scalability. (See Section 3).
3. The slack variables s_i and the corresponding objective penalty factor M push the optimization to generate a *no-overflow* routing solution. The model is quite flexible, as with minor modifications, the integer program can be set to minimize the total overflow.

A significant disadvantage of the formulation (ILP-GR) is its size. First, for a given net T_i, the number of decision variables for this net is equal to $|\mathcal{T}(T_i)|$—the number of possible Steiner trees connecting the terminals in T_i. Second, the number of nets N may also be very large. Nevertheless, we use (ILP-GR) as the basis of our GR algorithm. In the subsequent discussion, we outline the manner in which we deal with the issues posed by large formulation size.

2.2 Column Generation

The first step in an IP-based approach to global routing is to solve the linear-programming (LP) relaxation of (ILP-GR), a relaxation obtained by replacing the binary requirement on the variable $x_{it} \in \{0, 1\}$ with a nonnegativity restriction $0 \le x_{it} \le 1$. The linear program is solved by a *column-generation* (CG) procedure [11, 12].

To describe the column generation procedure it is helpful to consider the dual (LPD-GR) of the linear programming relaxation of (ILP-GR):

$$\max_{\lambda \le M, \pi \le 0} \sum_{i \in N} \lambda_i + \sum_{e \in E} \pi_e u_e \qquad \text{(LPD-GR)}$$

$$\text{s.t.} \quad \lambda_i + \sum_{e \ni t} \pi_e \le c_{it} \ \forall i = 1, \ldots, N, \forall t \in \mathcal{T}(T_i). \qquad (1)$$

In a column generation procedure, only a small subset of all possible routes is explicitly included in the LP relaxation of (ILP-GR). Let $\mathcal{S}(T_i) \subset \mathcal{T}(T_i)$ be the set of routes considered for net T_i. The restricted master problem for (ILP-GR) is

$$\min_{x \ge 0, s \ge 0} \sum_{i=1}^{N} \sum_{t \in \mathcal{S}(T_i)} c_{it} x_{it} + \sum_{i=1}^{N} M s_i \qquad \text{(RMLP-GR)}$$

$$\begin{cases} \sum_{t \in \mathcal{S}(T_i)} x_{it} + s_i = 1 & \forall i = 1, \ldots, N \\ \sum_{i=1}^{N} \sum_{t \in \mathcal{S}(T_i)} a_{te} x_{it} \le u_e & \forall e \in E. \end{cases}$$

Solving (RMLP-GR) yields a (primal) solution $(\hat{x}, \hat{s})$ as well as values $\hat{\lambda} \le M$ and $\hat{\pi} \le 0$ for the dual variables in (LPD-GR). By linear programming duality, if the solution $(\hat{\lambda}, \hat{\pi})$ satisfies all the dual constraints (1), then $(\hat{x}, \hat{s})$ is an optimal solution to the LP relaxation of (ILP-GR). If not, then the violated dual constraint suggests a column (variable) that may be added to (RMLP-GR) to reduce its objective value.

To determine if the dual solution $(\hat{\lambda}, \hat{\pi})$ is feasible, we must determine if there exists a route $t \in \mathcal{T}(T_i)$ with $\hat{\lambda}_i + \sum_{e \ni t} \hat{\pi}_e > c_{it}$. This is itself an optimization problem, known as the *pricing problem*, that can be decomposed into independent problems for each individual net $i = 1, \ldots, N$. Specifically, given net T_i, for each edge $e \in E$ define the binary decision variables t_e, taking value 1 if and only if edge e is used in a route for net T_i. The pricing problem for net T_i is then

$$\min_{t} \{ \sum_{e \in E} (c_e - \hat{\pi}_e) t_e \mid t \in \mathcal{T}(T_i) \}. \qquad \text{(PP}(T_i)\text{)}$$

Let t^* be an optimal solution to (PP(T_i)). If $\sum_{e \in E} \hat{\pi}_e t^*_e + \sum_{e \in E} c_e t^*_e < \hat{\lambda}_i$, then t^* identifies a violated constraint (1) in (LPD-GR), and the current solution to (RMLP-GR) can be improved. The CG procedure is summarized as follows:

0. For each $i = 1, \ldots, N$, initialize $\mathcal{S}(T_i)$ with at least one route. (In our implementation, we use the route generated for net T_i by the package Flute [9]).
1. Solve (RMLP-GR), yielding primal solution $(\hat{x}, \hat{s})$ and dual values $(\hat{\lambda}, \hat{\pi})$.
2. For each i=1,…, N, solve (PP(T_i)), yielding a route t^*. If $\hat{\lambda}_i + \sum_{e \in E} \hat{\pi}_e t^*_e > \sum_{e \in E} c_e t_e$, then $\mathcal{S}_i = \mathcal{S}_i \cup \{t^*\}$.
3. If improving routes for some net T_i were found, return to step 1. Otherwise, stop—the solution $(\hat{x}, \hat{s})$ is an optimal solution to the LP relaxation of (ILP-GR).

In order to speed solution time, we typically stop the procedure once the solution value has "tailed off." Specifically, if the objective value of (RMLP-GR) has made little or no improvement in the last 10 iterations, the CG procedure is terminated.

2.3 Solving the Pricing Problem

In the pricing phase (step 2) of the CG procedure, small-weight Steiner trees with respect to the weights $\hat{w}_e = c_e - \hat{\pi}_e$ must be identified. Finding a minimum-weight Steiner tree is in general NP-Hard [14], so our approach for finding columns that reduce the optimal value of (RMLP-GR) is based on local search. Given a dual solution $(\hat{\lambda}, \hat{\pi})$, the *reduced cost* of route t of net T_i is $\bar{c}_{it} = c_{it} - \hat{\lambda}_i - \sum_{e \ni t} \hat{\pi}_e$. Note that the pricing problem (PP(T_i)) can be viewed as a procedure for identifying a Steiner tree t for net T_i whose reduced cost $\bar{c}_{it} < 0$. By the complementary slackness conditions of linear programming, for any optimal solution $(\hat{x}, \hat{s})$ to (RMLP-GR) and corresponding dual solution $(\hat{\lambda}, \hat{\pi})$, the reduced cost $\bar{c}_{it} = 0$ if $\hat{x}_{it} > 0$.

Our local improvement procedure for solving (PP(T_i)) uses this fact as well as the following simple observation. Given a route $t \in \mathcal{S}(T_i)$, let $V(t)$ be the set of vertices in t. If the variable $\hat{x}_{it} > 0$, and if there exists a path P' from some terminal $u \in T_i$ to a vertex $v \in V(t)$ such that the weight of P' (with respect to weights $\hat{w}$) is less than the weight of the path P from u to v using edges in t, the reduced cost of tree $t' = t \cup P' \setminus P$ is negative. Thus, adding the variable corresponding to route t' to (RMLP-GR) may reduce its objective value. Figure 1 demonstrates how new routes can be constructed by finding short u-v paths from $u \in T_i$ to a vertex v on the base Steiner tree. An interesting feature of this pricing algorithm is that the new routes can use different Steiner points than the original routes.

To approximately solve (PP(T_i)) for a net T_i, our procedure starts with the tree $t \in \mathcal{S}(T_i)$ with largest value of $\hat{x}_{ti}$. Using edge weights $\hat{w}_e = c_e - \hat{\pi}_e$, a single-source shortest path problem from some terminal $u \in T_i$ to each vertex $v \in V(t)$ is solved. If the uv path length is smaller than the existing path length, a new route has been identified.

Dijkstra's single-source shortest path algorithm [13] generates an entire tree of shortest path weights, thus possibly identifying *many* routes that would reduce the objective value of (RMLP-GR). In our implementation, we add a pre-specified maximum number of routes selected uniformly from the set of all identified negative cost routes per net.

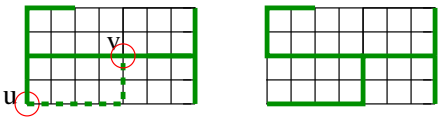

Figure 1: Improving routes via a shortest path algorithm on a weighted grid-graph

An important component of our pricing algorithm for a given net T_i is the selection of the starting terminals from which Dijkstra's algorithm to identify improving routes is run. In our implementation, Dijkstra's algorithm is run using the most congested terminals as starting points.

We identify these congested terminals as follows. For each terminal we compute the weight of the path P' that connects it to the base Steiner tree. The most congested terminals are those for which the corresponding P' has maximum weight.

2.4 Selecting Nets to Price

For large instances of (ILP-GR), the CG procedure can be significantly accelerated by only solving the pricing problem (PP(T_i)) for a subset of all the nets. To select the nets $T_i \in \mathcal{N}$ for which (PP(T_i)) is solved, our procedure takes advantage of information provided by the solution of the restricted master problem. Specifically, if $\hat{s}_i > 0$, then the net T_i is not completely routed using the existing routes in $\mathcal{S}_i$, so net T_i is priced by step (2) of the CG procedure.

Nets for pricing are also selected based on measures of congestion in the current LP solution to (RMLP-GR). Congestion may be identified in one of two ways. First, congested edges are those edges e that have the most negative value of $\hat{\pi}_e$. The intuition behind this choice is that $\hat{\pi}_e$ provides the rate of change in the objective function of (RMLP-GR) per unit additional capacity on edge e. A second way to identify a congested edge is to let $r_i \in \arg\max_{t \in \mathcal{S}(T_i)} \hat{x}_{ti}$ be the route for net T_i with the largest solution value in (RMLP-GR). The value $\eta_e = \sum_{i=1}^{N} a_{r_i e}$ is the number of units of capacity on edge e that would be used if the routes r_i were used for each net $T_i \in \mathcal{N}$. If the value $(\eta_e - u_e)$ is large, then edge e is highly-congested. In our algorithm, a bounding box around a congested edge e (identified by either of the two measures) is created, and all nets T_i that contain a terminal inside the bounding box are also candidates to be priced by (PP(T_i)) in step 2 of the CG procedure.

2.5 Branch and Bound

Once the CG procedure for the solution of the LP relaxation of (ILP-GR) is complete, either because no improving routes were found in the pricing phase, or because an iteration limit was reached, a promising candidate subset of routes $\mathcal{S}(T_i) \subset \mathcal{T}(T_i)$ has been identified for each net T_i. Using only these route variables, the integer program (ILP-GR) is formulated and solved by the commercial integer programming solver CPLEX (v9) [10]. The solution returned by CPLEX is a feasible solution to the problem.

The proposed approach, based on the direct solution of (ILP-GR), has significant promise to improve the solution quality of existing GRs. For example, using this approach, we solved the 2D IBM01 circuit of the ISPD1998 suite [1] and were able to improve the wirelength by approximately 5% compared to the best solution found by FGR [19], without any overflows. However, the runtime to achieve this high-quality solution was prohibitively long—a few hours. Thus, in the following section, we discuss mechanisms for decomposing the full global routing (ILP-GR) into smaller instances in order to accelerate the overall runtime.

3. DECOMPOSITION FOR SCALABILITY

Many existing global routing algorithms define reasonably-sized subproblems and create a full global routing out of solutions to these subproblems. For example, to achieve a good runtime, BoxRouter [8] starts by solving an IP over a small rectangular box on the chip and then progressively increases the size of the box to generate new IPs, fixing the solution to the previous IP. Fixing the solution of previous IP when increasing the box size may lead to a degradation in solution quality. SideWinder [15] solves an IP over the entire chip by gradually introducing more base patterns for the nets in the congested areas at each iteration. However, Sidewinder only works with three simple-shaped patterns which are defined *a priori*. The work [21] proposes a hierarchical IP approach that first solves a small IP to plan the routing of the longest nets. However, the impact of the shorter nets is neglected.

As demonstrated in Section 2, our proposed algorithm for global routing has potential to find high-quality solutions, but also requires a mechanism to accelerate the procedure. In this section, we first discuss a decomposition of the integer program (ILP-GR) into smaller ones that correspond to non-overlapping rectangular "subregions" on the chip. We introduce the concept of "floating-terminals" to define the IP of each subregion, providing significant flexibility for routing nets that might enter or exit that subregion. We then discuss effective integration of the subregion solutions to generate a valid and high quality final solution. Finally, we discuss a technique to plan long nets that pass multiple subregions.

3.1 Subregion Extraction / IP Decomposition

The goal of our decomposition procedure is to define non-overlapping rectangular subregions on the chip. Each subregion defines the boundaries of a smaller-sized GR problem which we solve using the IP-based procedure outlined in Section 2. The objective of the subregion definition is to define *balanced* subregions, resulting in "equally-difficult" optimization problems that take approximately the same time to solve. We first tried a coarse, uniform grid to define the subregions. However, we noticed that the IPs corresponding to the congested subregions were taking significantly longer time to be solved by our procedure (e.g., hours for congested subregion and minutes for the less congested ones).

To decrease the gap in solution times, our procedure attempts to create subregions having the same average edge utilization (AEU). To define the AEU, we first assume that all nets are routed using the Steiner route generated by [9] in the 2D-projected problem. For each edge, we define a utilization factor as the ratio of the number of routes that cross the edge to the edge capacity. Many edges might have a utilization factor higher than one, indicating an overflow. For a subregion, the AEU is the the average utilization factor of all edges contained in the subregion.

The subregions are defined using a partitioning-based strategy, depicted graphically in Figure 3. We recursively apply bi-partitioning to subregions to obtain two new smaller-sized subregions, at each step ensuring that the generated subregions have similar AEU. During one bi-partitioning step, to decide between a vertical or horizontal partitioning, we choose the one that results in the smaller aspect ratio of the generated subregions. The partitioning of a subregion is stopped when any of its sides reaches 32 units of the routing grid, a size empirically set to generate an IP that can be typically solved by the procedure outlined in Section 2 in an acceptable runtime.

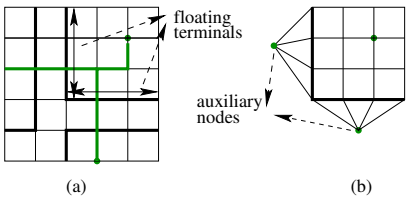

Figure 2: Modifying grid-graph of a subregion to handle floating terminals within our IP procedure.

Each subregion defines a new grid-graph $G'(V', E')$ and set of nets $\mathcal{N}' \subset \mathcal{N}$. The set $\mathcal{N}'$ is composed of two types of nets, nets that have all terminals inside the subregion ($T_i \subseteq V'$), and nets that have at least one terminal outside the subregion ($T_i \not\subseteq V'$). Figure 2(a) shows the latter type of these nets. The net in the figure belongs to three different subregions. The common boundaries of these subregions are shown in bold. Considering the top-right subregion, we can think of having a net with one fixed and two "floating" terminals. Each floating terminal represents a portion of a subregion boundary through which the net will connect to another subregion.

To specify such nets in our IP, we represent each floating terminal using an auxiliary node. The auxiliary node is added to the set of nodes V' in the grid-graph. Edges connecting the nodes that are on the subregion boundary to their corresponding auxiliary node are added to the set E'. The added edges have infinite capacity and zero cost in the definition of the integer program (ILP-GR). Figure 2(b) illustrates the addition of auxiliary nodes and edges. After applying this simple construction, the integer program (ILP-GR) is well-defined, and can be solved by the procedure outlined in Section 2.

The example of Figure 2 is for 2D routing, but in the general 3D case, each boundary of a subregion is a plane and graph G' extends to the third dimension. The nodes on this vertical boundary plane are connected to their corresponding auxiliary node.

3.2 Handling Long Nets

In our subregion extraction procedure, the regions through which net T_i are routed, and hence the locations of floating terminals for T_i are taken from a given Steiner topology (e.g., the route generated by Flute for T_i). Even though the subregion IP has significant flexibility in implementing the routes that connect floating terminals, the entire procedure relies on knowing the assignment of each net to one or more subregions. For long nets that are assigned to more than two subregions, the subregion assignment issue becomes particularly important. (This is in spite of the fact that the number of subregions is typically much smaller than the number of bins of the routing grid). Figure 3 illustrates this point. The two long nets are routed using their Steiner routes, both of which pass from subregion A. If A is congested, it is better to detour these nets from A.

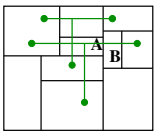

Figure 3: Assigning long nets to subregions using an initial Steiner route can cause unresolvable overflow.

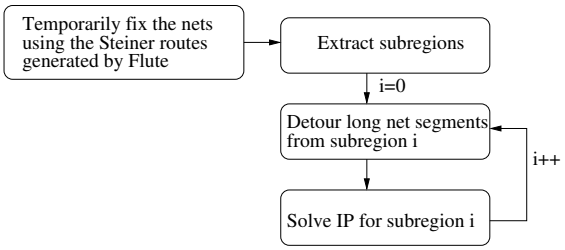

Figure 4: Dynamic planning of long nets

Many procedures for assigning the subregions of long nets were investigated. First, similar to [21], we tried a standard hierarchical IP formulation in terms of long nets. This approach helped considerably with the removal of overflow, but its failure to accurately consider the impact of short nets falling completely inside a subregion led to poor quality solutions in terms of wirelength. Instead, we use the procedure depicted in Figure 4, which is explained below:

1. Before extracting the subregions, all nets are broken into two-terminal segments using the Steiner trees generated by Flute.

2. Next, the subregion extraction begins. The routes used to compute the AEU during subregion extraction are those generated by step 1.

3. Once the subregions are created, we begin solving the IPs for each subregion in a congestion-based order that is discussed in the next subsection. Before solving a subregion IP, we use a procedure to detour as many long nets that pass from the subregion as possible. Once a subregion is solved, its solution remains fixed. Before solving the IP for the next subregion, we apply the same procedure to detour as many passing nets as possible to the remaining subregions.

To detour long net segments outside a subregion, as required by step 3, we apply a shortest path algorithm on the grid-graph. We set the edge weights inside the target subregion to a large number to avoid getting re-routed inside that subregion. Also, outside the target subregion, the edges that have overflow will also have a large weight. The weights of the grid-graph gets updated every time a long net is detoured.

Our procedure provides two significant benefits. First, it detours the long nets dynamically, every time a new subregion is processed. Second, it considers an estimate of the current congestions based on continually updating the edge weights to detour long nets.

3.3 Subregion Integration

So far we explained how the subregions are extracted and the long nets are dynamically planned. We then finalize the routing solution using a two-phase approach. In the first phase, we fix the locations of the floating terminals and generate a routing solution for all the routes that completely fall within a subregion. For the routes that cover more than one subregion, we fix a "backbone" inside each of its subregions. In the second phase, we connect the backbones of these longer nets in adjacent subregions. This two-phase approach is entirely based on integer programming and solved using formulation (ILP-GR) as we elaborate next.

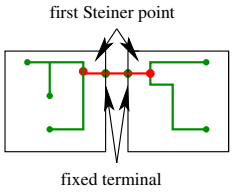

Figure 5: We remove the segment connecting the fixed boundary terminal (previously floating) to the first Steiner point in the route backbone. We reroute these connecting segments given the fixed boundary terminals using our IP procedure.

In phase 1, we visit and process the subregions in the following order. We first compute the total edge overflow (TEO) based on the initial Flute Steiner-routes in each subregion. Subregions are then processed in decreasing order of their TEOs. Every time a subregion is solved, the floating terminals for a net T_i will get fixed at a boundary of the region. Specifically, the net T_i is partially routed, and subsequent subregion IPs must respect this partial routing. If two consecutive subregions (in terms of TEO) are not physically adjacent, we process them in parallel.

Phase 1 fixes the locations of the floating terminals on the subregion boundaries and also generates an initial routing solution. For this routing solution, we fix all the (short) nets that completely fall within a subregion. For the (long) nets covering more than one subregion, we fix a "backbone" inside each of its subregions as follows. For a long net we visit each of its subregions. Inside each subregion, we remove the "branch" that connects the fixed terminal on the subregion boundary to the route backbone. Figure 5 illustrates. Considering the route that goes within two subregions, for each subregion, we remove the segment that connects the identified fixed terminal on the boundary to the backbone of the route. Specifically, the removed segment is one that connects the boundary terminal to the first Steiner point of the route in the subregion.

In phase 2, once these "connecting segments" are removed, routing resources will be freed and we reconnect these segments using the formulation (ILP-GR) while fixing the routes of short nets and backbones of long nets from phase 1.

In summary, our procedure uses the IP formulation of Section 2 as a basic component throughout the routing process. Subregion extraction aims to generate equally-difficult optimization problems. Subregions are initially solved while using the flexibility of floating terminals on their boundaries as they are visited in the order of their TEO. After the initial phase, all the short nets and the backbone of long nets within each subregion are routed. In addition, the locations of the floating terminals on the subregion boundaries will get fixed to ones that ensure obtaining a feasible solution. The final phase effectively connects the backbones of the long nets in different subregions. In our simulation results, we observed significant improvement in solution quality from phase 2 for connecting the subregions using integer programming.

The result is a scalable, effective global router, called GRIP (Global Routing via Integer Programming). GRIP is a *robust* tool and does not rely on any design-dependent tuning. Everything is based on integer programming. The defined parameters AEU (for defining subregions) and TEO (for ordering them) both depend on our congestion estimate which is only based on the Flute-Steiner routes.

Table 1: Results for ISPD 2007 benchmarks.

Benchmark	Best Approach			GRIP		
	tool	TOF	WL	TOF	WL	%Impr
adaptec1.2D	FGR	0	54.7	0	52.8	3.5
adaptec2.2D	FGR	0	52.4	0	50.1	4.4
adaptec3.2D	FGR	0	131.5	0	125.9	4.3
adaptec4.2D	FGR	0	125	0	122.1	2.3
adaptec5.2D	FGR	0	153.2	0	144.5	5.7
newblue1.2D	FGR	400	46.3	0	44.9	3.1
newblue2.2D	FastRoute	0	76.4	0	73.2	4.1
newblue3.2D	NTHU-R	31454	110.8	35573	107.9	N/A
adaptec1.3D	FGR	0	88.6	0	78.9	10.94
adaptec2.3D	FGR	0	90.1	0	80.7	10.41
adaptec3.3D	FGR	0	200.6	0	182.2	9.17
adaptec4.3D	FGR	0	183.0	0	167.7	8.36
adaptec5.3D	NTHU-R	0	260.2	0	227.8	12.45
newblue1.3D	NTHU-R	0	91.0	0	78.1	14.14
newblue2.3D	FGR	0	132.5	0	114.7	13.46
newblue3.3D	NTUgr	31454	167.0	33158	162.7	N/A

4. SIMULATION RESULTS

We implemented GRIP using C++. For solving individual LPs and IPs we used CPLEX (v9) [10]. We demonstrate the performance of GRIP on the ISPD 2007 benchmarks [2]. Table 2 reports the total number of routed nets and the grid size for each benchmark. Each benchmark has a 3D as well as a projected-2D version, and the grid size is the same in both versions. The 2D and 3D benchmarks have two and six metal layers, respectively.

We compare the summation of wirelength and via cost (denoted by WL) in Table 1[1]. The comparison is made against the best reported solution for each benchmark, found by either FGR 1.1 [19], NTHU-Route 2.0 [6], FastRoute 3.0 [22] or NTUgr [7]. For the 2D and 3D benchmarks, we obtained an average improvement of 3.9% and 11.3%. The improvement in the 3D benchmarks were more significant than in the 2D case, because GRIP considers explicit 3D Steiner routes, skipping the layer assignment phase.

The total overflow (denoted by TOF) is also given in Table 1. The GRIP solutions had zero overflow for all the benchmarks except newblue3 which is known to be unroutable. For newblue3 we report NTUgr as the best tool only because it generates the smallest overflow (and not the smallest WL).

Table 2 reports the running time (wall clock time) of both 2D (column 5) and 3D (column 6) benchmarks. The runtime unit is minutes. The number of subregions created by the subregion extraction procedure described in Section 3.1 for each benchmark is given in column 4. In reporting the runtimes we process independent subregions in parallel, which is why wall time is the appropriate measure. GRIP was run on a heterogenous grid of CPUs, shared by many users, and controlled by the Condor grid computing toolkit [16]. When solving the IP formulation, the majority of runtime was spent on the linear program (for column generation) rather than solving the integer program using branch-and-bound. This helped us to effectively identify candidate routes; the number of candidate routes reached up to a hundred for some nets, while for some nets only a few routes were generated in the linear program. Overall, our runtimes are scalable and adjustable, since they depend on the number of subregions we chose to create. Continuing work is aimed at further exploiting parallelism to obtain similar high-quality solutions in a smaller run time.

[1]Benchmark solutions are available for download at: http://wiscad.ece.wisc.edu/gr/

Table 2: Our grid size, subregion count and runtime

benchmark	#nets	grid	#subregions	2D	3D
adaptec1	176715	324x324	91	290	440
adaptec2	207972	424x424	169	229	366
adaptec3	368494	774x779	562	262	387
adaptec4	401060	774x779	558	240	398
adaptec5	548073	465x468	199	410	688
newblue1	270713	399x399	137	319	513
newblue2	373790	557x463	242	230	296
newblue3	442005	973x1256	1162	701	1389

5. CONCLUSIONS

We presented GRIP, a tool for global routing using integer programming. We introduced a novel IP formulation to select candidate routes for each net based on a continually-updated congestion metric, while directly working with the 3D model of the routing problem. To achieve reasonable runtime, we discussed subregion extraction and IP decomposition as well as a method for planning long nets and integrating the subregion solutions.

6. REFERENCES

[1] ISPD 1998 global routing benchmark suite, 1998.

[2] ISPD global routing contest and benchmark suite, 2007.

[3] C. Albrecht. Global routing by new approximation algorithms for multicommodityflow. *IEEE TCAD*, 20(5):622–632, 2001.

[4] L. Behjat and A. Chiang. Fast integer linear programming based models for VLSI global routing. In *ISCAS (6)*, pages 6238–6243, 2005.

[5] M. Burstein and R. Pelavin. Hierarchical wire routing. *IEEE TCAD*, 2(4):223–234, 1983.

[6] Y.-J. Chang, Y.-T. Lee, and T.-C. Wang. Nthu - route 2.0: A fast and stable global router. In *ICCAD*, pages 338–343, 2008.

[7] H.-Y. Chen, C.-H. Hsu, and Y.-W. Chang. High-performance global routing with fast overflow reduction. In *ASPDAC*, pages 582–587, 2009.

[8] M. Cho, K. Lu, K. Yuan, and D. Z. Pan. Boxrouter 2.0: architecture and implementation of a hybrid and robust global router. In *ICCAD*, pages 503–508, 2007.

[9] C. C. N. Chu and Y.-C. Wong. Flute: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design. *IEEE TCAD*, 27(1):70–83, 2008.

[10] CPLEX Optimization, Inc., Incline Village, NV. *Using the CPLEX Callable Library, Version 9*, 2005.

[11] G. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

[12] J. Desrosiers and M. E. Lübbecke. A primer in column generation. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, chapter 1. Springer, 2005.

[13] E. W. Dijkstra. A note on two problems in connetion with graphs. *Numerische Mathematik*, 1:269–271.

[14] M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM Journal of Applied Math*, 32:826–834, 1977.

[15] J. Hu, J. A. Roy, and I. L. Markov. Sidewinder: a scalable ILP-based router. In *SLIP*, pages 73–80, 2008.

[16] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor—A hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 104–111, 1998.

[17] M. D. Moffitt. Maizerouter: Engineering an effective global router. In *ASPDAC*, pages 226–231, 2008.

[18] M. M. Ozdal and M. D. F. Wong. Archer: a history-driven global routing algorithm. In *ICCAD*, pages 488–495, 2007.

[19] J. A. Roy and I. L. Markov. High-performance routing at the nanometer scale. In *ICCAD*, pages 496–502, 2007.

[20] T. Terlaky, A. Vannelli, and H. Zhang. On routing in VLSI design and communication networks. *Discrete Applied Mathematics*, 156(11):2178–2194, 2008.

[21] Z. Yang, S. Areibi, and A. Vannelli. An ILP based hierarchical global routing approach for VLSI ASIC design. *Optimization Letters*, pages 281–297, 2007.

[22] Y. Zhang, Y. Xu, and C. Chu. Fastroute3: a fast and high quality global router based on virtual capacity. In *ICCAD*, pages 344–349, 2008.