GRIP: Global Routing via Integer Programming

Tai-Hsuan Wu, *Student Member, IEEE*, Azadeh Davoodi, *Member, IEEE*, and Jeffrey T. Linderoth

***Abstract*—This work introduces GRIP, a global routing technique via integer programming. GRIP optimizes wirelength and via cost directly without going through a traditional layer assignment phase. Candidate routes spanning all the metal layers are generated using a linear programming pricing phase that formally accounts for the impact of existing candidate routes when generating new ones. To make an integer-programming-based approach applicable for today's large-scale global routing instances, the original problem is decomposed into smaller subproblems corresponding to rectangular subregions on the chip together with their net assignments. Route fragments of nets that fall in adjacent subproblems are connected in a flexible manner. In case of overflow, GRIP applies a second-phase optimization that explicitly minimizes overflow. By using integer programming in an effective manner, GRIP obtains high-quality solutions. Specifically, for the ISPD 2007 and 2008 benchmarks, GRIP obtains an average improvement in wirelength and via cost of 9.23% and 5.24%, respectively, when compared to the best result in the open literature.**

***Index Terms*—Global Routing, Integer Programming.**

I. Introduction

Design of Integrated Circuits in nanometer regime is subject to obstacles such as manufacturability, variability, yield-loss and timing failures. The increase in the size of modern designs and the shrinking geometries of devices continue to escalate these challenges. The severity of many of these design issues is impacted by the routing of the interconnects. Global routing (GR) is the primary step of routing during which the net regions are planned, so it has increasingly gained significance in recent years. Higher quality GR solutions can potentially alleviate the severity of nanometer design issues.

The release of large-sized ISPD 2007 and 2008 benchmarks [2], [3] resulted in remarkable progress in GR procedures. Specifically, among the two categories of concurrent and sequential procedures, the latter became very popular recently to handle large-sized problem instances [8], [31], [23], [22], [26], [9]. However, an inherent downside of the sequential procedures is their high dependence on properties such as the ordering of the nets or the definition of empirical cost functions. Alternatively, some of the concurrent techniques that are "optimization-centric" such as those based on integer programming have typically been able to generate good solutions for moderate-sized problem instances. [7], [32], [4], [25], [6], [29], [17]. Procedures based on a hybrid combination of sequential and concurrent approaches are also proposed [10] to explore the solution quality and runtime tradeoff.

An extended abstract of this paper was published by the 2009 Design Automation Conference (DAC'09) [30].

T.-H Wu and A. Davoodi are with the Department of Electrical and Computer Engineering, and J. Linderoth is with the Department of Industrial and Systems Engineering, University of Wisconsin, Madison, WI, 53706 USA (e-mail: {twu3, adavoodi, linderoth}@wisc.edu).

In this work, we propose GRIP, a GR procedure that heavily relies on integer programming techniques. Not only is GRIP able to generate solutions for large-sized instances, but the solutions found by GRIP demonstrate a considerable improvement in quality compared to the best solutions in the open literature. In addition, GRIP has minimal dependency on the nature of the benchmark instances and robustly generates the best solution in each case.

To effectively use integer programming, GRIP decomposes the large-sized problem into smaller-sized subproblems. Each subproblem corresponds to a rectangular subregion on the chip together with its net assignments. The smaller-sized subproblems are solved individually, and later the route fragments of the same net in adjacent subproblems are connected. A final phase can be run to reduce overflow. The above steps are based on solving an integer program (IP) that aims to select one route for each net from a set of promising candidate routes.

This work makes the following contributions:

- An integer program for the GR problem that minimizes the cost of the routed nets as its objective. The cost is the sum of wirelength and via costs of 3-D routes, thus avoiding a layer assignment phase.
- Generation of a promising set of candidate routes for each net using a linear-programming-based pricing procedure. Pricing is an iterative procedure that effectively considers the impact of currently-generated routes when generating new ones via a measure that correlates with congestion.
- A decomposition procedure to make integer programming applicable to large-scale instances. The routing problem is divided into a set of balanced subproblems in terms of the complexity required to solve them. Consequently the runtime of our procedure depends on the number of subproblems, some of which can be processed in parallel.
- A novel method called "floating terminals" for retaining connection flexibility when solving each subproblem.
- An integer-programming based technique for reconnecting route fragments from the decomposed subproblems.
- A final "clean-up" integer programming-based procedure for routing a set of designated nets to minimize overflow.

In simulation results, GRIP achieves an average 9.23% and 5.24% improvement in total cost (i.e., wirelength and via cost) for the ISPD 2007 [2] and ISPD 2008 [3] benchmarks respectively. These results are compared to the best solution reported for each case from four state-of-the-art academic global routers. The significant improvement is possible due to a combination of the concurrent nature of IP, effective pricing for candidate route generation, directly working with the 3-D model of the problem, effective decomposition into subproblems, and effective recombination of the solution fragments.

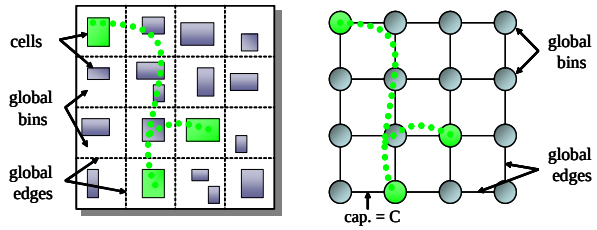

Fig. 1. The construction of grid graph for Global Routing problem.

The remainder of the paper is divided into eight sections. In Section II, we discuss preliminaries of the GR problem. In Section III, we discuss our IP model, and we give a procedure to approximately solve the IP in Section IV. Section V discusses problem decomposition and integration of solutions of subproblems. In Section VI, we give an IP to minimize overflow and describe a procedure that uses this IP. A brief comparison to other modern optimization-based approaches for global routing is given in Section VII. Computational results are reported in Section VIII, followed by conclusions.

II. Preliminaries

Global routing is a critical step of the design flow during which the routing path of each net is planned considering the placement information. This planning later guides the detailed routing stage that assigns wire segments to routing tracks.

A. Global Routing Formulation

The GR problem can be conceptualized on a grid-graph $G = (V, E)$ as depicted in Figure 1. After placement, a chip is partitioned into rectangular regions called global bins. Each bin is a vertex $v \in V$ in grid-graph. The boundary between two adjacent global bins is modeled as an edge $e \in E$. With each edge $e \in E$ is associated a capacity u_e reflecting the maximum available routing resource between two adjacent bins. Also given as problem input to GR is a set of (multi-terminal) nets $\mathcal{N}$. Each net T_i is defined by a set of vertices (terminals) in V ($T_i \subset V$). At the level of GR, we assume the terminals for the nets are located at the center of each global bin. The goal of the GR problem is to find a set of Steiner trees connecting the terminals of each net $T_i, \forall i \in \mathcal{N}$.

When evaluating a routing solution, two metrics are typically used—wirelength and overflow. The wirelength is the sum of the lengths of the Steiner trees that route the nets $\mathcal{N}$. Overflow is defined as the total amount by which routing demand exceeds capacity on the edges. Typically, overflow should also be minimized (zero is desirable), since it directly corresponds to the routability of a design. Global routing is often repeatedly used in the physical design flow, so runtime is an important practical consideration for a GR algorithm.

B. Multi-Layer Global Routing

In modern design, a chip usually contains multi-layer routing resources. For example, the ISPD'07 benchmarks have six routing layers—three horizontal layers and three vertical layers [2]. Adjacent layers are connected by vias. In the grid graph, vias are modeled as edges with unlimited capacity. By associating higher wirelength costs to these edges, the usage of vias can be reduced.

To handle the multi-layer GR problem, a two-step approach is usually adopted. The first step is to project all 3-D edges to 2-D edges, aggregating their capacities, and to solve a 2-D GR problem. Next, a procedure called layer assignment is employed to take the solution to the 2-D assignment problem and assign routes to the layers. Vias are used to connect these segments for the final 3-D solution. Minimizing the via cost (or via count) is considered during the layer assignment phase.

The above two-step approach is adopted by the majority of recent academic GR procedures [8], [23], [10], [26], [31]. However, neglecting the via cost when solving the 2-D routing problem may lead to significant degradation in solution quality. Even minor details in the strategies can significantly impact the solution quality [20]. Recently [31] proposes via-aware Steiner tree generation and "3-bend routing", in conjunction with layer assignment to more effectively consider the via cost throughout the GR problem. However, via cost is not directly considered, thereby the solution space is not fully explored.

Ideally, layer assignment could be avoided by solving the 3-D GR problem directly. Among the recent routing methods, FGR [28] is the only one that is based on 3-D maze routing, and as a result demonstrates better solutions in the reported benchmarks but with higher execution runtime.

III. An Integer Program for Global Routing

In a mathematical description of the GR 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 edge costs 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 GR problem can be written as

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

$$\sum_{t \in \mathcal{T}(T_i)} x_{it} + s_i = 1 \quad \forall i = 1, \ldots, N, \qquad (1)$$

$$\sum_{i=1}^{N} \sum_{t \in \mathcal{T}(T_i)} a_{te} x_{it} \leq u_e \quad \forall e \in E, \qquad (2)$$

$$x_{it} \in \{0, 1\} \quad \forall i = 1, \ldots, N, \forall t \in \mathcal{T}(T_i),$$
$$s_i \geq 0 \quad \forall i = 1, \ldots, N.$$

The parameter c_{it} is the cost of route t for net T_i which is computed as the total length of the 3-D 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 equations (1) in the model enforce 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. The equations (2) in the model ensure that the given edge capacities are not exceeded.

Typically M is chosen sufficiently large to ensure that all nets are routed. Specifically, M is chosen larger than the maximum wirelength that a route could have in an optimal solution—for example, the total number of grid edges. When M is chosen in this way, the formulation explicitly maximizes the number of routed nets, in addition to minimizing wirelength. Other choices of M are possible if wirelength is more important than routing all of the given nets. 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 its cost c_{it}. The formulation can thus directly handle the 3-D Global Routing problem, avoiding the traditional layer-assignment phase which can be a source of sub-optimality.
2) The cost of a route can correspond to any other metric such as the area-capacitance of the route over multiple metal layers.
3) 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 [28]. We note that the final version of GRIP has some "net-breaking" to define subproblems for scalability. (See Section V).
4) 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. (See Section VI).

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 and edges E may also be very large. Nevertheless, we use (ILP-GR) as the basis of GRIP. In the subsequent discussion, we outline the manner in which we deal with the issues posed by the large formulation size.

IV. Solution Procedure via Price-and-Branch

GRIP's procedure to obtain an approximate solution to the above large-scale integer program (IP) consists of two phases, as shown in Figure 2. First, a *pricing* procedure is used to generate a set of candidate routes. Second, *branch-and-bound* is applied to solve (ILP-GR) using only the set of generated candidate routes. This two-phase heuristic procedure is commonly known as price-and-branch [5], [19].

A. Overview of Candidate Route Generation

To generate a set of candidate routes for each net, GRIP solves a linear-programming (LP) relaxation of (ILP-GR), a relaxation obtained by replacing the binary requirements on the variables $x_{it} \in \{0,1\}$ with the weaker constraints $0 \leq x_{it} \leq 1$. The linear program is solved by a *column-generation* (CG) procedure [13], [14] during which a subset of all possible routes (GRIP's candidate routes) are identified.

In column generation, we start by replacing $\mathcal{T}(T_i)$ (the set of all possible routes of net i) in the relaxed version of (ILP-GR) by the set $\mathcal{S}(T_i) \subset \mathcal{T}(T_i)$, initially containing only one candidate route per net. We then expand $\mathcal{T}(S_i)$ over the iterations of CG, while guaranteeing the added routes decrease the objective expression.

To describe the CG procedure, we need to first consider the dual (LPD-GR) of linear programming relaxation of (ILP-GR):

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

$$\text{s.t.} \quad \lambda_i + \sum_{e \ni t} \pi_e \leq c_{it} \ \forall i = 1, \ldots, N, \forall t \in \mathcal{T}(T_i). \quad (3)$$

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) considering only $\mathcal{S}(T_i)$ is

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

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

Solving (RMLP-GR) yields a (primal) solution $(\hat{x}, \hat{s})$ as well as values $\hat{\lambda} \leq M$ and $\hat{\pi} \leq 0$ for the dual variables in (LPD-GR). By linear programming duality, if the values $(\hat{\lambda}, \hat{\pi})$ satisfy all the dual constraints (3), then $(\hat{x}, \hat{s})$ is an optimal solution to the LP relaxation of (ILP-GR). If not, then the violated dual constraint suggests that adding the associated column (as a new route variable) to (RMLP-GR) may reduce its objective value. The process can then repeat to further identify routes which can reduce the objective value. Solving the LP relaxation via column generation guarantees obtaining the optimal solution to the linear program, as if all the routes were explicitly considered.

To determine if the dual values $(\hat{\lambda}, \hat{\pi})$ (generated using $\mathcal{S}(T_i)$) are feasible in (LPD-GR) (which includes $\mathcal{T}(T_i)$), we must determine if there exists at least one 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*, and the pricing problem can be decomposed into independent problems for each individual net $i = 1, \ldots, N$.

Specifically, the pricing problem for net T_i is

$$\min_t \{c_{it} - \sum_{e \ni t} \hat{\pi}_e \mid t \in \mathcal{T}(T_i)\}. \qquad \text{(PP}(T_i)\text{)}$$

If the optimal solution of (PP(T_i)) is sufficiently small ($< \hat{\lambda}_i$), then the values $(\hat{\lambda}, \hat{\pi})$ are not dual feasible. Specifically, let t^* be an optimal solution to (PP(T_i)). If

$$c_{it^*} - \sum_{e \ni t} \hat{\pi}_e < \hat{\lambda}_i, \qquad (4)$$

then t^* identifies a violated constraint (3) in (LPD-GR). The current solution to (RMLP-GR) can thus be improved by updating $\mathcal{T}(S_i)$ to include t^* as a new column (candidate route).

Next we describe the steps of the iterative CG procedure:

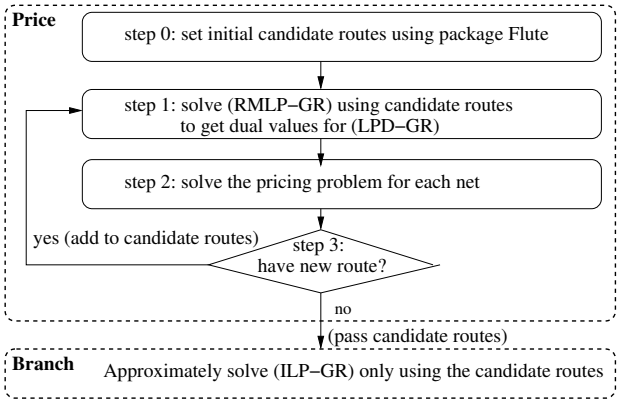

Fig. 2. Overview of Price-and-Branch for GR.

0. For each $i = 1, \ldots, N$, initialize $\mathcal{S}(T_i)$ with exactly one route. (For net T_i GRIP uses the route generated by the package Flute [11].)
1. Solve (RMLP-GR), yielding a primal solution $(\hat{x}, \hat{s})$ and dual values $(\hat{\lambda}, \hat{\pi})$.
2. For each i=1,. . . , N, use the dual solution $(\hat{\lambda}, \hat{\pi})$ to set up the pricing problem (PP(T_i)). Solve (PP(T_i)) to obtain a route t^*. If $c_{it^*} - \sum_{e \ni t} \hat{\pi}_e < \hat{\lambda}_i$, then $\mathcal{S}(T_i) = \mathcal{S}(T_i) \cup \{t^*\}$
3. If an improving route for some net T_i was found in step 2, return to step 1 to further identify new routes. Otherwise, stop—*the solution* $(\hat{x}, \hat{s})$ *is an optimal solution to the LP relaxation of (ILP-GR).*

Figure 2 illustrates these steps. First, one initial route for each net T_i is generated by Flute. When using Flute we randomly select one of attainable routes from the generated Steiner points for each net. These routes are very close to minimum Steiner trees for the nets. The total of their wirelength is likely to give a lower bound on the total wirelength in an optimal solution to the GR problem. On the other hand, these routes are initially 2-D routes that only use the lowest horizontal and vertical layers and would result in significant overflow if all used in combination.

After step 1, in a primal solution $(\hat{x}, \hat{s})$ with dual values $(\hat{\lambda}, \hat{\pi})$, nets $T_i \in \mathcal{N}$ that are not able to be routed completely with existing Steiner trees in the set $\mathcal{S}(T_i)$ will have $\hat{s}_i > 0$ and (by the complementary slackness condition of linear programming) $\hat{\lambda}_i = M$. Also by linear programming duality theory, the dual variable $\hat{\pi}_e$ is the rate of change of the optimal objective value of (RMLP-GR) per unit change in u_e, the capacity of edge e. It gives a (local) measure of how much the objective function (wirelength) would improve if one more unit of capacity was available for edge e. We use this information to systematically identify new columns which can reduce the objective value (routes that pass the condition in step 2) as we explain in the Section IV-B.

By observing the condition (4), the CG procedure will *naturally* seek to find routes for nets T_i with large $\hat{\lambda}_i$, routing them with edges that have $\hat{\pi}_e$ as close to zero as possible. Ideally the routes would use edges with $\hat{\pi}_e = 0$, which implies (again by complementary slackness) that the edge is not being used to capacity. In this way, one can imagine that the CG procedure helps to iteratively disperse the initial nets from lower layers to upper layers and from congested areas to less congested ones. As a result, it is unnecessary to utilize layer assignment to manipulate the initial 2-D routes.

To summarize, the strengths of the pricing procedure are the following:

- When generating new candidate routes at each iteration of the CG procedure, the impact of candidate routes of previous iterations are effectively taken into account. This is done by resolving (RMLP-GR), incorporating the impact of all existing routes to get a new fractional solution, and new dual values.
- Within each iteration, solving the pricing procedure effectively identifies new candidate routes since the objective of (RMLP-GR) is always improved. Moreover, based on the dual values, a measure that correlates with the current congestion is also incorporated in selecting the nets to price. (see Section IV-C).

The computational experience with the CG procedure in GRIP was that the objective value of (RMLP-GR) was quickly improved in the first iterations, but the rate of improvement decreased significantly in the later iterations. This "tailing off" phenomenon is very common to the CG procedure [14]. The improving routes at later iterations of the algorithm almost always come from nets outside highly congested areas. Further, the wirelengths of the improving routes are almost identical to trees currently available for routing. In these cases, adding the routes to (RMLP-GR) makes little or no improvement to the objective value. A significant portion of the runtime of the CG procedure can be spent on iterations that improve the objective value of (RMLP-GR) only marginally. Thus, in order to speed solution time, GRIP typically stops the procedure once the solution value has tailed off. Specifically, if the objective value of (RMLP-GR) has made little or no improvement (less than 10 wirelength units) in the last 20 iterations, the CG procedure is terminated.

Next, we discuss the details of step 2 of the CG procedure.

B. Solving the Pricing Problem for One Net

In the pricing phase (step 2) of the CG procedure, GRIP solves (PP(T_i)) for each net. We rewrite the objective expression of (PP(T_i)) as $c_{it} - \sum_{e \ni \mathcal{T}(T_i)} \hat{\pi}_e = \sum_{e \ni \mathcal{T}(T_i)} (c_e - \hat{\pi}_e)$, where c_e is the cost associated with edge e (e.g., $c_e = 1$ when considering wirelength and via count).

To minimize the above objective for net T_i, GRIP considers a weighted graph with edge weights $\hat{w}_e = c_e - \hat{\pi}_e$. Minimizing the objective of (PP(T_i)) requires finding the smallest-weight Steiner trees on this weighted graph. Finding a minimum-weight Steiner tree is in general NP-Hard [16], so GRIP adopts a (heuristic) approach for finding columns that reduce the objective value of (RMLP-GR) based on local search.

Within the pricing problem, condition (4) should be evaluated in step 3 of the CG procedure. Given a dual solution $(\hat{\lambda}, \hat{\pi})$, the *reduced cost* of route t of net T_i is $\bar{c}_{it} = c_{it} - \sum_{e \ni t} \hat{\pi}_e - \hat{\lambda}_i$. The pricing problem 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

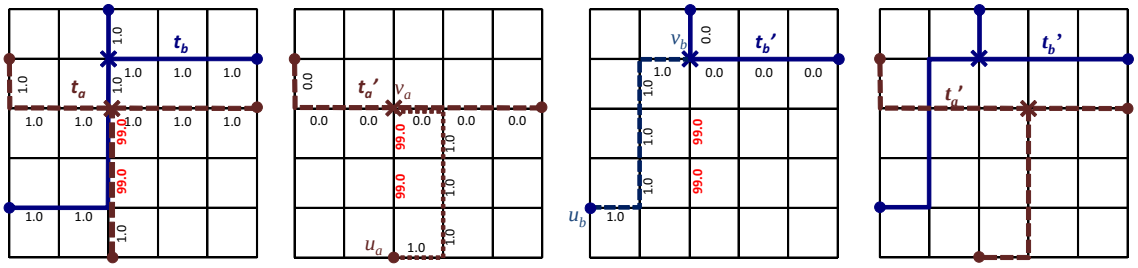

Fig. 4. Procedure to identify new candidate routes with reduced cost via rerouting segments of an existing route.

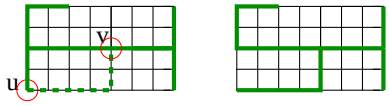

Fig. 3. Improving routes via shortest path algorithm on a weighted grid-graph

condition of linear programming, for any optimal solution $(\hat{x}, \hat{s})$ to (RMLP-GR) and corresponding dual values $(\hat{\lambda}, \hat{\pi})$, the reduced cost $\bar{c}_{it} = 0$ if $\hat{x}_{it} > 0$.

GRIP's 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 of the terminals and Steiner points in t. If the variable $\hat{x}_{it} > 0$, and if there exists a path P' that connects two vertices $(u, v) \in V(t)$ 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 3 shows inserting such a u-v path into a base Steiner tree.

To approximately solve (PP(T_i)) for a net T_i, GRIP starts with the tree $t \in \mathcal{S}(T_i)$ with largest value of $\hat{x}_{it}$. Using edge weights $\hat{w}_e = c_e - \hat{\pi}_e$, a single-source shortest path problem is solved for some u-v paths, where $(u, v) \in V(t_i)$. If the new u-v path has smaller length than the existing path, a route with negative reduced cost has been identified. To identify sources and sinks for the shortest path problems, the selected route t is decomposed into a set of two-terminal segments r_{jt} by breaking it at the Steiner points of t. The segments are considered in descending order of their weight $\sum_{e \ni r_{jt}} \hat{w}_e$. When considering segment r_{jt}, the remaining segments of t are considered as a "base Steiner tree", and an alternative route of the segment r_{jt} must be found to connect to this base. Zeroing the weights for all tree edges which are not on the segment ($\hat{w}_e = 0 \ \forall e \ni t \setminus r_{jt}$) and running Dijkstra's single-source shortest path algorithm connects the the segment r_{jt} to the base net in a minimum cost fashion [27].

Dijkstra's algorithm [15] generates an entire tree of shortest path weights, thus possibly identifying *many* routes for one net that would reduce the objective value of (RMLP-GR). At each iteration of column generation, GRIP identifies a subset of these routes by uniformly sampling from all identified nets, and adds them as new columns. Specifically, at most 40 routes per net per CG iteration will be added for the nets inside the congested area, and at most 16 routes per net per CG iteration are added for nets outside the congested area. (See Section IV-C for how we define congested area.)

Figure 4 demonstrates how new routes are constructed by rerouting a two-terminal u-v segment. In the figure, the cost and capacity of each edge is 1, and there are two initial routes t_a and t_b for nets T_a and T_b, respectively. After solving (RMLP-GR), GRIP sets the edge weights to $\hat{w}_e = c_e - \hat{\pi}_e$. These edge values are shown in 4(a). Note that the two edges with overflow have large negative dual values ($\hat{\pi}_e << 0$), resulting in large positive edge weights. Based on these edge weights, the cost of routes t_a and t_b are 205.

Assume that net T_a is selected for pricing first. As shown in Figure 4(a), tree t_a can be decomposed into three segments, each including a terminal. The total edge weight is maximum in the segment that includes the two edges with large weights, so GRIP starts by rerouting this segment, using the remaining ones for the base Steiner tree. To reroute the segment, it is removed from t_a, and the edge weights of the remaining edges on the base Steiner tree are set to zero, as shown in Figure 4(b). Thus, GRIP considers the base tree as a backbone when reconnecting u_a and v_a using Dijkstra's algorithm.

After reconnecting, an improved route t'_a, avoiding the highly-weighted edges, is identified with cost 10. In a similar fashion, GRIP considers net T_b, and reroutes the segment u_b-v_b as shown in Figure 4(c). The new segment t'_b for net T_b has a new cost of 9 units. The reduced costs for these routes are $\bar{c}_{at'_a} = 10 - \lambda_a$ and $\bar{c}_{bt'_b} = 9 - \lambda_b$, respectively. If these reduced costs are smaller than zero, (indicating the identified routes to violate the constraint (3) in (LPD-GR)), then these routes are added to (RMLP-GR) as new candidate routes.

An interesting feature of this pricing algorithm is that the new routes can use different Steiner points than the original ones. In this example, t_a and t_b are generated independently at the same major iteration of column generation, with costs coming from the same dual variables. GRIP will add both of these generated columns to the master LP (RMLP-GR), and later let IP decide how to utilize these routes in the most effective manner possible.

C. 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, GRIP takes advantage of information provided by the solution of (RMLP-GR). Specifically, if $\hat{s}_i > 0$, then net T_i is not completely routed using the existing routes in $\mathcal{S}(T_i)$, so it is necessary to find routes for net T_i using the pricing procedure. GRIP first prices all the nets in descending order of $\hat{s}_i$ (> 0).

To decide whether or not to price the remaining nets with $\hat{s}_i = 0$, GRIP considers measures of congestion in the current LP solution to (RMLP-GR). In the first measure, congested edges are the edges e that have the most negative value of $\hat{\pi}_e$. The intuition is that the dual value $\hat{\pi}_e$ gives a (local) measure of how much the objective function (wirelength) would improve if one more unit of capacity was available for edge e. In this sense, the edge weights have a positive correlation with congestion.

The second measure identifies a congested edge by letting $r_i \in \arg\max_{t \in \mathcal{S}(T_i)} \hat{x}_{ti}$ be a 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 $(\eta_e - u_e)$ is large, then e is highly congested.

GRIP defines a bounding box (of 3x3 units of grid edges) around an identified-congested edge e. All nets that contain a terminal inside the bounding box are repriced. GRIP first reprices nets that are identified by the first congestion measure, followed by repricing nets found by the second measure.

D. 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 tailing off was detected, 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 a black-box commercial integer programming package. The solution returned by the solver 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 small 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 [28], without any overflows. However, the runtime to achieve this high-quality solution for such a relatively-small instance was prohibitively long—a few hours. Thus, in the following section, we discuss mechanisms for decomposing the full global routing (ILP-GR) problem into smaller instances and procedures for combining the solutions in order to generate high-quality solutions to large-scale GR instances. The decomposition procedure considerably accelerates the overall runtime.

V. Decomposition for Scalability

Many existing global routing algorithms define reasonably-sized subproblems and create a full solution from the solutions to these subproblems. For example, to achieve a good runtime, BoxRouter [10] starts by solving an IP over a small rectangular box on the chip and progressively increases the size of the box to generate new IPs, fixing the solution to the previous IP. However, this solution fixing when increasing the box size may lead to a degradation in solution quality. The work [32] 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.

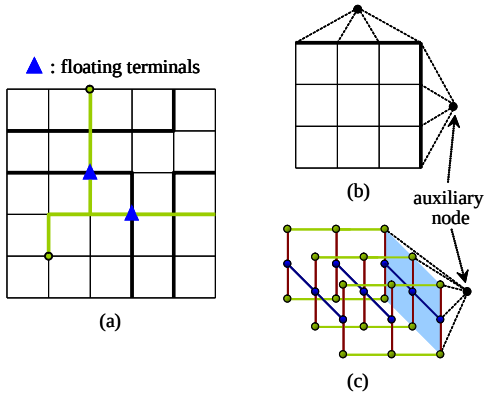

Fig. 5. Modifying grid-graph of a subregion to handle floating terminals.

As demonstrated in Section IV, the price-and-branch procedure has potential to find high-quality solutions, but needs to be accelerated. In this section, we discuss ideas for decomposing the integer program (ILP-GR) into smaller ones that correspond to non-overlapping rectangular areas on the chip, together with their net assignments. For example, if all the terminals of a net fall within a rectangle, then the net is assigned to that subproblem and is bound to be routed inside the rectangle. We first discuss how GRIP's IP-based procedure is applicable to solve one subproblem. We then discuss the procedures to define subproblems and integrate their solutions.

A. Solving One Subproblem

A subproblem is characterized by a rectangle on the chip referred to as a subregion, together with a set of nets that must be routed within that subregion. For some nets, all terminals will lie within the subregion, but for longer nets, additional (or all) of their terminals might be outside the subregion. Nets whose terminals do not all fall within the subregion are referred to as inter-region nets. Inter-region nets are partially routed by each subproblem, and subsequently their segments in different subregions are connected.

To be applicable in a decomposition-based procedure, GRIP must handle the case when a subproblem includes both within-region and inter-region nets. GRIP's procedure works as follows. Each subproblem 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: the within-region nets that have all terminals inside the subregion $(T_i \subseteq V')$, and the inter-region nets that have at least one terminal outside the subregion $(T_i \not\subseteq V')$. Figure 5(a) shows the latter type of these nets. The net in the figure belongs to three different subregions. The neighboring boundaries of these subregions are shown in bold. The routing problem for the bottom-left subregion views this net to have 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 route inter-region nets in a subproblem, GRIP represents each floating terminal using an auxiliary node that 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 5(b) illustrates the addition

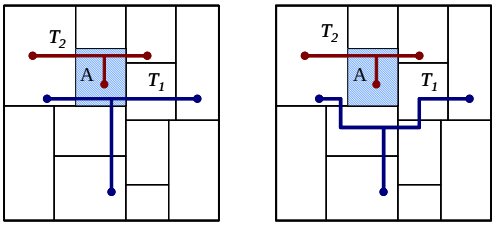

Fig. 6. Defining subproblems using initial Flute-based net planning (left), Improving net assignment to the subregions via detouring (right).

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 IV.

The example of Figure 5(b) is for 2-D routing, but in the general 3-D case, each boundary of a subregion is a plane and the graph G' extends to the third dimension, as shown in Figure 5(c). The nodes on this vertical boundary plane are connected to their corresponding auxiliary node.

B. Forming the Subproblems

The challenge of decomposing the problem into subproblems is best understood by means of our initial computational experience. Our first decomposition approach was to define a uniform grid of subregions consisting of the same area. Net assignment to the subregions was based on their routing by Flute. This natural but naive decomposition approach resulted in the IPs corresponding to the congested subregions taking significantly longer to be solved by our procedure (e.g., hours for congested subregion and minutes for the less congested ones). Thus, an important objective of the subproblem definition is to achieve *balance*, resulting in "equally-difficult" problems that take approximately the same time to solve.

GRIP's procedure for defining subproblems begins by routing all nets using the 2-D Steiner route generated by Flute. For each grid edge e of the 2-D problem, a utilization factor is defined as the ratio of the number of (Flute) routes that cross edge e to its (projected) capacity u_e. The utilization factor plays an important role in defining the subregion boundaries.

Next, GRIP applies a recursive bi-partitioning strategy, trying to balance an average edge utilization factor (AEU) for each region. At each step, one rectangular partition is divided into two new rectangles where the AEU is balanced between the two. The AEU for a partition is defined as the average of the utilization factors of the grid edges in the corresponding rectangle. Moreover, to decide between a vertical or horizontal partitioning, GRIP chooses the one that results in the smaller aspect ratio of the generated rectangles. The recursive bi-partitioning stops when any of the sides of the current partition 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 IV in an acceptable runtime. This partition will then be taken as a subregion. Figure 6(left) shows a chip that has been divided into subregions by the procedure.

Once the subregions are created, the net assignments suggested by Flute are further improved by considering the congestion of the subregion. Figure 6(left) illustrates this point. The two nets T_1 and T_2 are routed using their Steiner routes, both of which pass through subregion A. Net T_1 does not have any terminals inside subregion A. If A is congested, it is better to detour T_1 from A, as shown in Figure 6(right), reserving the routing resources for nets that *must* be routed into the subregion.

To improve the net assignments to the subregions, GRIP relies on the fact that subproblems are solved in a congestion-based ordering, described with more details in Section V-C. Before solving a subproblem, GRIP detours as many nets as possible that "pass" through the corresponding subregion (i.e., do not have a terminal in it). The remaining (undetoured) nets inside the subregion are the ones assigned to it and the corresponding subproblem is then solved. The procedure repeats before solving the subsequent subproblem.

To detour routes out of a subregion, a shortest path algorithm is used. For a net that does not have any terminals in the current subregion, we identify the segment (using its Flute route) that passes the subregion, and consequently the two terminals that are connected using this segment. The two terminals are reconnected via a new segment back to its tree backbone using the same shortest path procedure explained in the Section IV-B (see Figure 6). The weights on the grid graph for the shortest path problem are set as follows. Since the net should be detoured outside the subregion, weights of all grid-edges inside the subregion are set to infinity. For the remaining edges, if an edge is used to capacity by the existing (Flute and detoured Flute) routes, the weight is set to a large positive number (=100). The remaining edges have a weight of 1. The detouring procedure in GRIP has the benefit that it is dynamic, continually updating edge weights for rerouting, every time a new subregion is processed.

This strategy of creating the subproblems may result in congested regions getting divided, which may result in breaking many nets. An advantage of breaking nets in this way is that it allows more routing resources into each subregion, as the regions are typically composed of small, highly-congested areas and less-congested areas. In addition, before solving a subproblem, GRIP detours as many nets as possible into unprocessed subproblems (which are less congested) to further release routing resources. Moreover, the detrimental effects of net breaking are mitigated by the flexible, IP-based mechanism for attaching pseudo-terminals, described further in Section V-C.

C. Patching the Solutions of Subproblems

Thus far, we have explained how the subregions are formed and how the net assignments are made to define subproblems. GRIP solves the subproblems in a rather sequential order with limited parallel processing. After all the subproblems are solved, a final phase connects the route segments that pass neighboring subregions. Both of these phases are explained in this section.

GRIP first orders the subproblems based on the total edge overflow (TEO) inside their subregions. The TEO is total amount of overflow that would occur in the subregion if the assigned Steiner routes (the Flute and detoured Flute routes described in Section V-B) were used. Subregions are sorted in

a list in decreasing order of their TEOs and processed in that order. Every time a subregion is processed, the floating terminal(s) for a net T_i are fixed at a boundary(s) of that subregion, as shown in Figure 7. Thus, the net T_i is partially routed, and subsequent, neighboring, subproblems must respect this partial routing by assuming the imposed boundary-terminal is fixed. This is *not*, however, the final connection for the net. After all subregions are routed, a IP-based post-processing step (subsequently described in this section) is applied that unfixes pseudoterminal locations and reconnects route segments.

GRIP processes the subregions in parallel by making a number of passes through the subregion list, ordered by TEO. At each pass, it processes few of the unrouted subregions. Specifically, it processes an unrouted subregion in a pass if the subregion is not physically adjacent to any other subregions being processed on that pass. As previously mentioned, once the subregions in a pass are processed, their pseudoterminal locations are temporarily fixed. GRIP iterates over the subregion list until all subregions are routed. We subsequently refer to the number of passes through the subregion list as the *number of steps* of the GRIP algorithm.

After solving all the subproblems, they need to get connected to each other. Specifically if an inter-region net spans multiple subregions, its segments in different subregions should be connected to each other. This connection is via utilizing the grid edges between the boundaries of the subregions, as shown in Figure 5(a). Note, by reserving the edges between adjacent subregions to be used only during the connection phase, GRIP in effect uses a non-greedy strategy to fairly allocate the routing resources between them.

Before attempting to connect the subregions to each other, GRIP further releases some routing resources inside each subregion by unrouting branches of some route fragments which connect to the subregion boundaries. These branches will get rerouted during the connection phase. This strategy allows obtaining a higher quality solution when connecting route fragments in different subregions to each other. Specifically, for each inter-region net which spans multiple subregions, GRIP fixes a "backbone" for each of its route fragments inside each of its subregions. To create the backbone, GRIP removes the branch of the route fragment that connects the boundary terminal to the first Steiner point of that tree in the subregion. (See Figure 7). After identifying the backbones, an inter-region net will end up with a set of backbones that are not connected to the boundaries and fall inside a set of adjacent subregions which need to be connected to each other.

Once these connecting segments are removed, routing resources are freed. GRIP then uses the same IP-based procedure to connect the route segments in adjacent subproblems. GRIP connects these segments using the formulation (ILP-GR), first fixing all routes of within-region nets and backbones of the inter-region nets. In the IP, the nets to be routed are two terminal nets crossing the inter-region boundary, each terminal being a Steiner point of the backbone in the region. By setting the edge weight $w_e = 0$ for all edges in the backbone, the IP effectively connects the two sub-nets at *any* location on the backbones. When connecting two neighboring subregions, remaining (unfixed) capacity is allocated to the subproblem

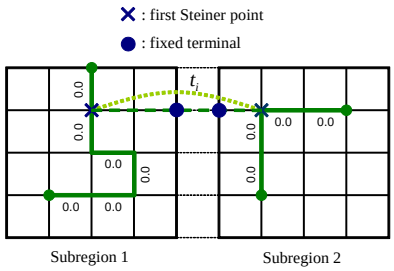

Fig. 7. Connecting route-segments in adjacent subproblems.

in a manner that ensures that neighboring subproblems in *all* quadrants will be able to be effectively connected. It is important to reiterate that when generating candidate routes in the reconnection phase, the set $\mathcal{S}(T_i)$ is augmented, so that routes may contain different pseudo-terminal locations in the boundaries compared to the initial locations when the subproblems were solved.

VI. Handling Overflow

After connecting the subproblem solutions, GRIP evaluates if any net is left unrouted. In case all the nets were routed, which we found to be the case in the majority of our tested benchmarks, GRIP terminates. If nets were left unrouted, then routing those nets using any of the generated candidate routes will introduce overflow (the corresponding slack variable in Equation (1) is 1). In this section, we discuss an IP and specifics of price-and-branch procedure to reduce overflow of a given solution. We then discuss how GRIP applies this procedure to selected areas on the chip.

A. Integer Program for Overflow Reduction

GRIP uses the following IP to minimize overflow:

$$\min_{o_e} \sum_{\forall e \in E} M_e o_e \qquad \text{(ILP-OV)}$$

$$\sum_{t \in \mathcal{T}(T_i)} x_{it} = 1 \quad \forall i = 1, \ldots, N, \tag{5}$$

$$\sum_{i=1}^{N} \sum_{t \in \mathcal{T}(T_i)} a_{te} x_{it} \leq u_e + o_e \quad \forall e \in E, \tag{6}$$

$$x_{it} \in \{0,1\} \quad \forall i = 1, \ldots, N, \forall t \in \mathcal{T}(T_i),$$
$$o_e \geq 0 \quad \forall e \in E.$$

Compared to (ILP-GR), the slack variable s_i is removed from the net constraint (1), but a new slack variable o_e is added to the edge capacity constraints (6). The slack variable o_e will be positive if edge e contains overflow, and the objective is to minimize the overflow over all edges. In (ILP-OV), M_e is a constant weight that can be set to a different value for each edge. For example, in the case $M_e = 1\ \forall e$, the objective is to minimize the total overflow.

GRIP sets M_e by considering the overflow produced by routing the nets which were left unrouted by the original IP-based procedure. To route an unrouted net, GRIP selects from the candidate routes for that net, the route that would lead to minimum additional overflow. If edge e contains overflow in

this complete solution, then M_e is set equal to the amount of overflow. If it does not contain overflow, M_e is set equal to 1. Assigning higher M_e will result in removing the overflow from these edges, but may transfer the overflow to the other edges. Consequently, the end result in terms of total units of overflow may not differ. In our computational experience, setting M_e as described above and setting $M_e = 1, \forall e \in E$ both result in roughly the same amount of final overflow reduction.

B. Solution Procedure via Price-and-Branch

Similar to IP-based procedure of Section IV, GRIP utilizes column generation to solve the LP relaxation of (ILP-OV). The dual of the LP relaxation of (ILP-OV) is

$$\max \sum_{i \in N} \lambda_i + \sum_{e \in E} \pi_e u_e \qquad \text{(LPD-OV)}$$

$$\begin{cases} \lambda_i + \sum_{e \ni t} \pi_e \leq 0 & \forall i = 1, \ldots, N, \forall t \in \mathcal{T}(T_i), \\ -M_e \leq \pi_e \leq 0 & \forall e \in E, \\ \lambda_i : \text{free.} \end{cases}$$

GRIP starts with a small subset of routes and solves the restricted master problem for (ILP-OV):

$$\min_{x \geq 0, o \geq 0} \sum_{\forall e \in E} M_e o_e \qquad \text{(RMLP-OV)}$$

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

At the first iteration, $\mathcal{S}(T_i)$ only contains one route per net—the route used to obtain the complete solution. GRIP solves (RMLP-OV) to obtain the dual values $\hat{\lambda}$ and $\hat{\pi}$. The pricing problem is solved to identify a new route t^* that violates the first constraint of (LPD-OV) (i.e. $\hat{\lambda}_i + \sum_{e \in E} \hat{\pi}_e t^*_e > 0$), indicating the objective of (RMLP-OV) may be improved if t^* is added to $\mathcal{S}(T_i)$.

When solving the pricing problem to identify a negative reduced cost route for each net, the edge weight $\hat{w}_e = -\hat{\pi}_e$ is used. Note that $\hat{\pi}_e \leq 0$, so $\hat{w}_e \geq 0$, and Dijkstra's single-source shortest path algorithm can be used to identify promising routes, exactly as in the procedure described in Section IV-B. Just as in the GRIP for solving (ILP-GR), the linear program solution process is terminated when tailing off is detected. The resulting routes are given to a branch-and-bound solver to find an integer solution to (ILP-OV).

C. Defining Subproblems for Overflow Reduction

After integrating the subproblem solutions, overflow was not observed in the majority of the tested benchmarks. Only for three benchmarks in the ISPD08 suite was overflow observed. Further, overflow was typically confined to a very few "hot spots". GRIP exploits this observation to define small-sized subregions with their net assignments on which to apply the IP-based procedure for overflow reduction, described in Section VI-A. The routes on the other portions of the chip remain intact.

GRIP defines the subregions on which to reduce overflow in a sequential order but solves the corresponding subproblems in parallel. To define the subregions, GRIP traverses the grid edges in descending order of their overflow values in a complete solution. GRIP defines a rectangular subregion of 40x40 grid edges centered at each overflow edge. If an overflow edge is already included in a previously-defined subregion, a new subregion will not be defined. Moreover, if defining the subregion for an overflown edge results in overlap with a previously defined subregion, the new subregion will be shifted until the overlap is removed.

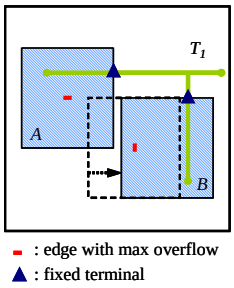

Fig. 8. Defining few subregions around the edges with overflow.

All the routes inside a subregion will be rerouted using the IP-based procedure for overflow reduction. If a net has terminals outside the subregion, fixed-terminal location(s) will be defined on the subregion boundary, based on its route generated from the previous steps. The fixed terminal locations are honored when solving the subproblem. Figure 8 illustrates this point. This process also ensures that the segments of rerouted nets in congested subregions will remain connected.

Please note that our overflow reduction procedure is particularly calibrated as a post-processing step when a global routing solution of low wirelength and overflow has already been generated. At this stage, we can define much larger subregions which we found necessary to effectively reduce overflow when solving (ILP-OV).

GRIP does not include any additional steps, such as post-processing, besides the ones mentioned thus far.

VII. Comparison to Optimization-Based Methods

A number of other authors have proposed optimization-based methods for global routing, and the purpose of this section is to attempt to place our work in context of these previous contributions.

An early description of applying a pricing procedure to solve the global routing problem is given by [18]. This work is perhaps the most similar to the GRIP algorithm, in that it relies on column generation, on defining subregions, and on pasting partial solutions together. In their work, there is no mention of solving the IP, only its LP relaxation, and computational results are not reported.

The works [4] and [29] both focus on developing fast algorithms for approximately solving the (full) LP relaxation of the global routing problem. In these approaches, the actual, primal, integer-valued, routing solution is done by a randomized rounding procedure. This is quite different from GRIP. GRIP is based on a price-and-branch approach for approximately solving the integer program. So both procedures of solving the LP relaxation (pricing), and obtaining an integer solution (branching) are different.

TABLE I
ISPD 2007, ISPD 2008 BENCHMARKS

Benchmark	# Nets	Grid	# Layers	V.cap	H.cap
adaptec1	176715	324x324	6	70	70
adaptec2	207972	424x424	6	80	80
adaptec3	368494	774x779	6	62	62
adaptec4	401060	774x779	6	62	62
adaptec5	548073	465x468	6	110	110
newblue1	270713	399x399	6	62	62
newblue2	373790	557x463	6	110	110
newblue4	636195	455x458	6	84	84
newblue5	1257555	637x640	6	88	88
newblue6	1286452	463x464	6	132	132
newblue7	2635625	488x490	8	212	212
bigblue1	282974	227x227	6	110	110
bigblue2	576816	468x471	6	52	52
bigblue3	1122340	555x557	8	148	148
bigblue4	2228903	403x405	8	202	202

The paper [25] is a similar approach to that of [4] and [29], but designs an algorithm that can specifically accounts for the effects of wire spacing during yield optimization.

The work [19] is a full branch-and-price procedure that mathematically shares many commonalities with GRIP. However, the work [19] is designed specifically for the switchbox routing problem, and the instance sizes are small enough so that region-based decomposition, done in GRIP, is not needed.

Similar to (ILP-GR), the paper [6] also suggests IP formulations for the global routing problem. To solve the formulations in [6], column generation is not employed, but rather a set of possible routes for each net is iteratively constructed during a congestion estimation phase. Generating routes *during* the LP solution process, as done in GRIP, has the significant advantage of exploiting the dual information to suggest good routes. The work [6] additionally considers a number of different objectives besides wirelength or overflow. The paper [32] builds on the work of [6], by describing different "heirarchical" approaches, where the routing problems are solved either "top-down" or "bottom-up." Computational results are given for chips with up to around 25,000 cells and nets.

BoxRouter [10] uses IP formulations as a fundamental component of their algorithm. These IP formulations are *not* Steiner-tree packing formulations like (ILP-GR). A fundamental idea behind BoxRouter is that of *progressive IP*, where the IP formulation is solved first for a subregion, and portions of this solution are fixed before proceeding to other regions. This is quite a different approach than GRIP's area-based decomposition and patching.

VIII. SIMULATION RESULTS

GRIP was implemented using C++. For solving individual LPs and IPs, MOSEK 5.0 [24] and CPLEX 6.5 [12] were used, respectively. In our simulations we set parameter M equal to 20,000 in formulation (ILP-GR) since we empirically knew that the wirelength of an individual route is smaller than this number. We report results on the ISPD 2007 [2] and 2008 benchmarks [3]. Table I reports the total number of routed nets and the grid size for each benchmark. Column 4 shows the number of metal layers. The last two columns report the projected edge capacities for vertical and horizontal layers.

Table II reports on the performance of GRIP on the benchmark instances[1]. For each benchmark, the total overflow is given in the column TOV, and the total wirelength (WL) is broken down into both edge and via costs. GRIP is compared to four recent academic global routers: FGR 1.1 [28], FastRoute 4.0 [31], NTHU-Route 2.0 [8], and BoxRouter 2.0 [10]. For each router, we report the percentage improvement in wirelength found by GRIP, as well as the total overflow.

Considering wirelength, GRIP consistently generates the best result for each benchmark, typically significantly improving the best known result. The improvement is larger in ISPD 2007 benchmarks since the via cost is 3 for these benchmarks, and therefore the benefits of avoiding layer assignment and directly generating 3-D routes are more significant. If the same GRIP solutions for the ISPD 2007 benchmarks (for via cost of 3) are evaluated assuming via cost is 1, still a wirelength improvement of on average 5.25% is obtained.

GRIP generates solutions with no overflow for the majority of the benchmarks, so the overflow reduction procedure of Section VI need not be applied. For four ISPD 2008 benchmarks, `newblue3`, `newblue4`, `newblue7`, and `bigblue4`, the overflow found by GRIP (before applying the overflow step) is reported in column 10. The corresponding overflow and wirelength numbers after applying the overflow reduction procedures is reported in the last two columns of Table II, as well as the number of subproblems solved for overflow reduction. GRIP generates the best known overflow for the `newblue4` and `newblue7`, and quite comparable overflow for `bigblue4`, while maintaining the wirelength improvement. For `newblue3` which is an artificially-generated benchmark, GRIP ends up obtaining significant wirelength improvement but the total overflow still remains higher than other methods even after the OV step os applied. For the three benchmarks (`newblue4`, `newblue7`, and `bigblue3`) the average degradation in wirelength compared to GRIP without its overflow step is about 0.11%.

As described in Section IV-B, when generating candidate routes, at most 40 routes per net, per iteration of column generation are selected. The total number of routes generated for a net may be higher than this number. Take `newblue7` as an example. For the first (most congested) subproblem, 60 iterations of column generation were required. In this subproblem, there was one net for which 2292 candidate routes were generated (about 38.2 per iteration). This net was long, spanned significantly over the congested subproblem, and had many terminals. For the same subproblem, there were some two-terminal nets for which only 2 candidate routes were generated. On average, 24.67 candidate routes per net were generated for the first subproblem in `newblue7`.

GRIP was run on a heterogenous grid of CPUs of 2GB memory, shared by many users, and controlled by the Condor grid computing toolkit [21]. Condor is a resource management software that allows for the creation of shared computational grids from the idle cycles of workstations. Table III reports the run time information, not including the overflow step. The number of subproblems created by the decomposition procedure for each benchmark is given in the second column.

As described in Section V-C, GRIP uses a congestion-based ordering to process and solve the subproblems in parallel. Column 5 in Table III gives the number of passes through

[1]Benchmark solutions can be downloaded at http://wiscad.ece.wisc.edu/gr/

TABLE II
RESULTS FOR ISPD 2007 AND ISPD 2008 BENCHMARKS. THE WIRELENGTH (WL) IS SCALED TO 10^5.

Benchmark	FGR1.1		FastRoute4.0		NTHU-Route2.0		BoxRouter2.0		GRIP (without OV step)				GRIP (with OV step)		
	TOV	WL(%)	TOV	WL(%)	TOV	WL(%)	TOV	WL(%)	TOV	WL	Edge	Via	TOV	WL	# sp
adaptec1 (07)	0	8.42%	0	10.99%	0	8.79%	0	11.99	0	81.0	36.5	44.5	–	–	–
adaptec2 (07)	0	8.33%	0	10.01%	0	9.33%	0	12.60	0	82.4	33.7	48.7	–	–	–
adaptec3 (07)	0	7.14%	0	9.39%	0	7.69%	0	10.61	0	185.4	97.5	87.9	–	–	–
adaptec4 (07)	0	3.94%	0	7.84%	0	7.36%	0	7.57%	0	172.3	91.5	80.8	–	–	–
adaptec5 (07)	0	8.11%	0	11.73%	0	8.18%	0	11.65	0	238.9	104.8	134.1	–	–	–
newblue1 (07)	526	10.99%	0	8.51%	0	7.76%	400	9.73%	0	83.9	24.9	59.0	–	–	–
newblue2 (07)	0	6.18%	0	10.49%	0	9.83%	0	9.83%	0	121.4	48.0	73.4	–	–	–
newblue3 (07)	39908	10.14%	31634	14.28%	31454	6.50%	38958	9.48%	52518	156.1	76.2	79.9	45960	157.6	38
Avg. WL Impr.		7.91%		10.40%		8.18%		10.43							
newblue4 (08)	262	4.00%	144	7.12%	138	4.65%	200	3.92%	196	124.2	83.2	41.0	136	124.4	2
newblue5 (08)	0	4.36%	0	5.88%	0	3.79%	0	4.35%	0	222.8	147.6	75.2	–	–	–
newblue6 (08)	0	5.44%	0	6.64%	0	3.61%	0	5.15%	0	170.5	102.4	68.1	–	–	–
newblue7 (08)	1458	4.12%	62	5.91%	68	4.97%	208	6.36%	110	335.5	189.5	146.0	54	335.8	1
bigblue1 (08)	0	6.33%	0	7.24%	0	4.02%	0	5.76%	0	53.7	37.2	16.5	–	–	–
bigblue2 (08)	0	5.94%	0	10.03%	0	5.07%	0	4.89%	0	86.0	48.3	37.7	–	–	–
bigblue3 (08)	0	4.40%	0	3.44%	0	3.43%	0	3.91%	0	126.2	78.7	47.5	–	–	–
bigblue4 (08)	414	4.72%	152	8.67%	162	4.48%	472	4.69%	232	220.5	121.9	98.6	180	220.7	1
Avg. WL Impr.		4.91%		6.87%		4.25%		4.88%							

TABLE III
RUNTIME INFORMATION (WITHOUT OVERFLOW STEP).

Benchmark	#Subp.	Runtime		#Steps	#Parallel Subp.	
		Wall	Total CPU		Ave.	Max.
adaptec1 (07)	100	388	2101	12	8.3	18
adaptec2 (07)	169	455	2704	16	10.6	23
adaptec3 (07)	576	478	6319	32	18.0	38
adaptec4 (07)	570	509	5221	30	19.0	51
adaptec5 (07)	225	584	3175	16	14.1	30
newblue1 (07)	144	483	2306	18	8.0	15
newblue2 (07)	238	467	4192	23	10.4	18
newblue3 (07)	1170	1430	14590	61	19.2	39
newblue4 (08)	174	529	2944	20	8.5	19
newblue5 (08)	311	821	4593	31	9.5	21
newblue6 (08)	140	448	2219	15	8.9	16
newblue7 (08)	325	985	4788	36	9.0	18
bigblue1 (08)	49	339	956	12	3.9	7
bigblue2 (08)	172	690	3411	21	8.0	20
bigblue3 (08)	208	731	2690	28	7.3	16
bigblue4 (08)	215	726	3096	27	7.6	21

the subproblem list before all subproblems were processed. The average and maximum number of subproblems processed independently on a pass are reported in columns 6 and 7. The wall clock times are given in columns 3. The runtime unit is minutes. These wall clock times are computed for the case when the computational grid was shared with other users. The actual wall clock time for solving the instances was larger, as jobs submit to the Condor-controlled shared grid often wait in the job queue while higher-priority jobs are run.

Compared to other routers, even IP-centric routers (specifically BoxRouter which is the fastest one), our walltime is *significantly* higher, so we skip any explicit runtime comparison. To reiterate, the focus of this work is to demonstrate the *significant improvement* in route quality that is possible with an intelligently-engineered IP-based scheme. Reducing the computational time, via additional parallelization of the algorithm, will be a focus of subsequent work.

Table III also reports the total CPU runtimes, computed as the summation of the runtimes spent by the CPUs to solve the subproblems in column 4. From the table, we can see that GRIP's significant improvement in solution quality comes at a considerable CPU time expense. However, comparing the total runtime to the wall clock time, it can be seen that even the small level of parallelism in GRIP can yield significant improvement, and reduce computational time to nearly acceptable levels. Continuing work is aimed at further exploiting parallelism to obtain similar high-quality solutions in even shorter wall clock times.

The Condor resource management policy ensured that each CPU ran at most one job at each time, so the CPUs were solely dedicated to solving the subproblems when utilized by us. Moreover, the CPUs in our computational grid were not vastly significantly different from each other in terms of speed or memory. Therefore, we expect that the total CPU runtimes listed in Table III provide a fair approximation of the sequential runtime if all the computations ran on one CPU.

For the benchmarks with overflow, an additional 30 minutes of walltime was used for solving the problem (RMLP-OV) to generate candidate routes, and a 5 hour limit was used for solving the IP using branch-and-bound. In the overflow case, the IP solver took significantly longer than when solving similar IPs whose primary objective was wirelengh. The runtime can potentially be improved by tuning the commercial solver, using a newer version of the CPLEX tool, or by developing a custom solver which we leave to future work. For each benchmark, the number of subregions defined around the congested areas for overflow reduction is listed in the last column in Table II. These subregions were all processed in parallel.

IX. CONCLUSIONS

GRIP is a new tool for global routing via integer programming (IP). It is based on solving an IP formulation by column generation followed by branch and bound to select candidate routes for each net. The method uses the information from the dual values of the relaxation of the integer program to create a dynamically-updated metric that correlates with congestion and suggests promising routes for each net. GRIP directly solves the 3D model of the routing problem. To achieve reasonable runtime for large instances, GRIP uses methods to decompose the problem into subproblems of manageable size and reconnects the subproblem solutions using IP. In case of overflow, it applies an overflow reduction procedure.

GRIP is the only IP-based procedure that achieves significantly lower wirelength and comparable overflow compared to the best solutions reported in the open literature. In future, we plan to develop techniques that can utilize the IP procedure and yet achieve a much higher degree of parallelism to improve the runtimes. We also plan to streamline the candidate route generation phase to consider constraints imposed on layer usage and route topology and to extend the formulation to

consider other objectives. We plan to explore means to utilize our framework in order to obtain quick estimate of congestion.

X. Acknowledgements

The comments of four anonymous reviewers were very helpful in clarifying the presentation and contribution of this work. The authors would like to thank Erling Andersen for providing free access to the Mosek LP solver for use in this research. The work of author Linderoth is supported by the U.S. Department of Energy Grant #DE-FG02-09ER25869. The work of author Davoodi is supported by the National Science Foundation Grant #CCF-0914981.

References

[1] ISPD 1998 global routing benchmark suite, 1998.
[2] ISPD global routing contest and benchmark suite, 2007.
[3] ISPD global routing contest and benchmark suite, 2008.
[4] Christoph Albrecht. Global routing by new approximation algorithms for multicommodity flow. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 20(5):622–632, 2001.
[5] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46:316–329, 1996.
[6] Laleh Behjat, Anthony Vannelli, and William Rosehart. Integer linear programming models for global routing. *INFORMS Journal on Computing*, 18(2):137–150, 2006.
[7] M. Burstein and R. Pelavin. Hierarchical wire routing. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2(4):223–234, 1983.
[8] Y.-J. Chang, Y.-T. Lee, and T.-C. Wang. NTHU - Route 2.0: A fast and stable global router. In *IEEE Intl. Conf. on Computer-Aided Design*, pages 338–343, 2008.
[9] Huang-Yu Chen, Chin-Hsiung Hsu, and Yao-Wen Chang. High-performance global routing with fast overflow reduction. In *IEEE Asia and South Pacific Design Automation Conf.*, pages 582–587, 2009.
[10] Minsik Cho, Katrina Lu, Kun Yuan, and David Z. Pan. Boxrouter 2.0: A hybrid and robust global router with layer assignment for routability. *ACM Trans. Design Autom. Electr. Syst.*, 14(2), 2009.
[11] Chris C. N. Chu and Yiu-Chung Wong. Flute: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 27(1):70–83, 2008.
[12] CPLEX Optimization, Inc., Incline Village, NV. *Using the CPLEX Callable Library, Version 9*, 2005.
[13] G. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.
[14] Jacques Desrosiers and Marco E. Lübbecke. A primer in column generation. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, chapter 1. Springer, 2005.
[15] E. W. Dijkstra. A note on two problems in connetion with graphs. *Numerische Mathematik*, 1:269–271.
[16] M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM Journal of Applied Math*, 32:826–834, 1977.
[17] Jin Hu, Jarrod A. Roy, and Igor L. Markov. Sidewinder: a scalable ILP-based router. In *ACM Intl. Workshop on System Level Interconnect Prediction*, pages 73–80, 2008.
[18] T. C. Hu and M. T. Shing. A decomposition algorithm for circuit routing. In *Mathematical Programming Essays in Honor of George B. Dantzig Part I*, volume 24, pages 87–103. Springer.
[19] David Grove Jogensen and Morten Meyling. A branch-and-price algorithm for switch-box routing. *Networks*, 40:13–26, 2002.
[20] T.-H. Lee and T.-C. Wang. Congestion-constrained layer assignment for via minimization in global routing. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 27(9):1643–1656, 2008.
[21] 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.
[22] Larry McMurchie and Carl Ebeling. Pathfinder: A negotiation-based performance-driven router for fpgas. In *FPGA*, pages 111–117, 1995.
[23] Michael D. Moffitt. Maizerouter: Engineering an effective global router. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 27(11):2017–2026, 2008.
[24] MOSEK ApS, Copenhagen, Denmark. *The MOSEK C API manual, Version 5.0*, 2008.
[25] Dirk Müller. Optimizing yield in global routing. In *IEEE Intl. Conf. on Computer-Aided Design*, pages 480–486, 2006.
[26] Muhammet Mustafa Ozdal and Martin D. F. Wong. Archer: A history-based global routing algorithm. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 28(4):528–540, 2009.
[27] Min Pan and Chris C. N. Chu. Fastroute 2.0: A high-quality and efficient global router. In *IEEE Asia and South Pacific Design Automation Conf.*, pages 250–255, 2007.
[28] Jarrod A. Roy and Igor L. Markov. High-performance routing at the nanometer scale. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 27(6):1066–1077, 2008.
[29] Tamás Terlaky, Anthony Vannelli, and Hu Zhang. On routing in VLSI design and communication networks. *Discrete Applied Mathematics*, 156(11):2178–2194, 2008.
[30] Tai-Hsuan Wu, Azadeh Davoodi, and Jeff Linderoth. GRIP: scalable 3-D global routing using integer programming. In *IEEE Design Automation Conf.*, pages 320–325, 2009.
[31] Yue Xu, Yanheng Zhang, and Chric Chu. FastRoute 4.0: global router with efficient via minimization. In *IEEE Asia and South Pacific Design Automation Conf.*, pages 576–581, 2009.
[32] Z. Yang, A. Vannelli, and S. Areibi. An ILP based hierarchical global routing approach for VLSI ASIC design. *Optimization Letters*, pages 281–297, 2007.