Abstract—In this paper, we propose a new global router, BoxRouter, powered by the concept of box expansion, progressive integer linear programming (ILP), and adaptive maze routing. BoxRouter first uses a simple PreRouting strategy to predict and capture the most congested region with high fidelity, compared to the final routing. Based on progressive box expansion initiated from the most congested region, BoxRouting is performed with progressive ILP and adaptive maze routing. Our progressive ILP is shown to be much more efficient than traditional ILP in terms of speed and quality, and the adaptive maze routing based on multi-source multi-target with bridge model is effective in minimizing the congestion and wirelength. It is followed by an effective PostRouting step which reroutes without rip-up to enhance the routing solution further and obtain smooth trade-off between wirelength and routability. Our experimental results show that BoxRouter significantly outperforms the state-of-the-art published global routers, e.g., 91% better routability than [1] (with 14% less wirelength and 3.3x speedup), 79% better routability than [2] (with similar wirelength and 2x speedup), 4.2% less wirelength and 16x speedup than [3] (with similar routability). Additional enhancement in box expansion and PostRouting further improves the result with similar wirelength, but much better routability than the latest work in global routing [4], [5].

Index Terms—Global routing, physical design, congestion, routability, integer linear programming (ILP), rectilinear minimum Steiner tree

I. INTRODUCTION

Routing is a key stage for VLSI physical design. Aggressive technology scaling has led to much smaller/faster devices, but more resistive interconnects and larger coupling capacitance. Since routing directly determines interconnects (wirelength, routability/congestion, and so on) and the overall VLSI system performance [6]–[8], it plays a critical role in the deep submicron design closure. For nanometer interconnects, the manufacturability and variability issues such as antenna effect, copper chemical-mechanical polishing (CMP), subwavelength printability, and yield loss due to random defects are becoming growing concern and shown to be directly impacted by wire embedding [8]–[15]. Thus, routing plays a major role in terms of the manufacturing closure as well.

In general, routing consists of two steps, global routing and detailed routing. While detailed routing finalizes the exact DRC-compatible pin-to-pin connections, the global routing, as its name implies, is the routing stage that plans the approximate routing path of each net to reduce the complexity of routing task and guide the detailed router [16]. Thus, it has significant impact on the overall wirelength, routability, and timing [16], [17]. Furthermore, it is also the key stage for optimizing the wire density distribution to improve the overall manufacturability (e.g., less post-CMP topography variation, less copper erosion/dishing, and less optical interference for better printability [10]–[13]) and yield (e.g., smaller critical area [14], [15]).

The importance of global routing in VLSI design flow has led to many works in predicting and estimating routing congestion, and designing global routers. Probability based congestion prediction for global routing is studied in [18]–[21], and global router based congestion estimation is researched in [22], [23] for early wirelength estimation. Within the scope of over-the-cell global routing model [2], Burstein et al. [24] proposed a hierarchical approach to speed up integer programming formulation for global routing, and Kastner [25] proposed a pattern-based global routing. Raja et al. [2] presented Chi dispersion router based on linear cost function as well as predicted congestion map, and showed better results than [25]. The multicommodity flow-based global router by Albrecht [3] showed good results and was used in industry, but at the expense of computational effort. Fast global router [4], [26], [27] can feed more accurate interconnect information (such as wirelength and congestion) back to placement or other early physical synthesis engines for better design convergence and tighter integration.

In this paper, a new routability-driven global router, namely BoxRouter is proposed. BoxRouter first performs a very fast yet effective PreRouting to identify the most congested regions or boxes. Then, it progressively expands the routing box, and performs routing within each expanded box (BoxRouting), until the entire circuit is covered, i.e., all the wires are routed. Efficient progressive ILP is formulated with adaptive maze routing, and effective PostRouting follows BoxRouting for further enhancement. The major contributions of this paper include the following.

• We propose a new ILP formulation which is significantly faster and more scalable than the traditional formulation in [3], [28], which makes it practical to apply ILP to solve VLSI routing.
• We observe that a simple PreRouting step can capture the overall congestion, and improve runtime.
• We propose the key BoxRouting idea which efficiently
utilizes limited routing capacities based on box expansion initiated from the most congested region estimated by PreRouting. BoxRouting is efficient in terms of routability as the wires in the more congested region are routed before those in the less congested region.

- We propose an efficient progressive integer linear programming (ILP) for BoxRouting. In our progressive ILP, only wires between two successive boxes are routed with L-shape patterns. Thus even with ILP, our runtime is still much faster than existing global routers [2], [3], [25].
- We propose an adaptive maze routing based on multi-source multi-target with bridge model. Our adaptive maze routing uses different routing strategies inside and outside the box such that routability can be maximized with minimum wirelength increase.
- We propose an effective PostRouting step which reroutes wires from the most congested region without rip-up. It is more efficient than the conventional rip-up & route. It also provides smooth trade-off between wirelength and routability with only a simple parameter.

BoxRouter achieves much better results on the standard ISPD98 IBM benchmarks than [1]–[5], thus pushes the state-of-the-art considerably. Due to the fundamental importance of global routing in routability, timing, and manufacturability [29], we believe it shall have many applications/implications for nanometer designs. Preliminary work of BoxRouter is presented in [30].

The rest of the paper is organized as follows. In Section II, preliminaries are described. Previous works are surveyed in Section III. Comparison and evaluation of ILP formulations are presented in Section IV. In Section V, BoxRouter is proposed. Experimental results are discussed in Section VI. Section VII concludes this paper with future work.

II. PRELIMINARIES

A. Notations

Table I lists the notations used throughout this paper.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_i$</td>
<td>vertex / global routing cell $i$</td>
</tr>
<tr>
<td>$e_{ij}$</td>
<td>edge between $v_i$ and $v_j$</td>
</tr>
<tr>
<td>$m_{ij}$</td>
<td>maximum routing capacity of $e_{ij}$</td>
</tr>
<tr>
<td>$c_{ij}$</td>
<td>available routing capacity of $e_{ij}$</td>
</tr>
</tbody>
</table>

B. Global Routing Model

The global routing problem can be modeled as a grid graph $G(V, E)$, where each vertex $v_i$ represents a rectangular region of the chip, so called a global routing cell (G-cell), and an edge $e_{ij}$ represents the boundary between $v_i$ and $v_j$ with a given maximum routing capacity $m_{ij}$. All the pins are assumed to be at the center of the corresponding G-cell. Fig. 1 shows how the chip can be abstracted into a grid graph where $m_{AB} = 3$. A global routing is to find paths that connect the pins inside the G-cells through $G(V, E)$ for every net.

C. Global Routing Metrics

The key task of global router is to maximize the routability for successful detailed routing [27]. In addition, wirelength, runtime, and timing are other important metrics for global router.

- **Routability** is usually the most important metric for global routing. It can be estimated by the number of overflows which indicates that routing demand exceeds the available routing capacity [25], [27]. In Fig. 1, the number of overflow between $v_A$ and $v_B$ is one, as there are four routed nets, but $m_{AB} = 3$. Formal definition of overflow can be found in [25].
- **Wirelength** is an important metric for placement as well as routing. But, it is less important compared to routability, as most wires are routed with shortest distances, thus the total wirelength is in general not too far away from optimum for a reasonable global routing solution [27]. However, there can be huge difference in terms of routability between two different global routing solutions of similar wirelength.
- **Runtime** is also an important consideration, as global routing links placement and detailed routing. A fast global router can feed proper interconnection information to higher level design flow for better design convergence [26].
- Other objectives such as timing and manufacturability are significant objectives as well. Since the focus of this paper is on the core global routing techniques, they are not explicitly considered in this work. However, our framework can be extended to handle them in the future.

III. PREVIOUS WORKS

A. Congestion Prediction and Estimation

Fast and accurate congestion prediction and estimation are essential techniques for reducing congestion in multiple stages of physical synthesis. For example, during placement, the cells can be inflated or the white space can be allocated in the congested region to reduce the congestion and enhance the routability [31], [32].

Recent study in congestion prediction includes a number of probabilistic approaches. Lou et al. [18] decompose a net into multiple two pin wires, then compute the probabilistic congestion for each G-cell based on the chance of having the two pin wire routed in the G-cell. While all possible detour-free paths are assumed with the same probability in [18], Westra et al. [20] only consider the simple L/Z shape routing...
based on the observation that one or two-bend nets are dominant in the real designs. By empirically extracting the occurrence of L and Z shape routings from multiple real industrial designs, different probability weights are assigned to L and Z shapes routings. In [23], it is shown that fast global routing based congestion estimation can be more accurate than probabilistic congestion prediction, as probabilistic approach highly depends on tools or designs. However, global routing based congestion estimation is not exact neither. A recent paper [26] claims that congestion estimation can be different from global routing result, unless the same techniques and optimization parameters are applied in both congestion estimation and global routing.

B. Global Router

Typical global router decomposes every net into a set of two pin wires by building minimum spanning tree or Steiner tree, then routes them by maze routing algorithm, followed by rip-up and reroute technique for further improvement. In [1], [25], Kastner et al. propose a simple pattern based routing rather than maze routing for fast runtime without incurring significant routing quality degradation. Hadsell et al. [2] take advantage of predicted congestion map to guide global router, and show considerable routing quality improvement over [1]. Congestion-aware Steiner tree in [4], [5], [26] reduces the runtime by increasing the number of nets routed by simple and fast pattern routing, and thus less relying on expensive maze routing.

While the previous global routers [2], [25], [26] are mainly based on pattern routing, maze routing, or shortest path algorithm, Albrecht [3] formulates the global routing as multi-commodity problem which can be solved by an approximation algorithm for fractional flow with randomized rounding. First, it repeats building a Rectilinear Minimum Steiner Tree (RMST) using maze routing in net-by-net manner. After all nets are routed in RMST, a set of G-cells above the given congestion threshold are selected, and all the nets on any of those G-cells are routed again by building new RMST. Two key advantages of such approach are that congestion can be evenly distributed over the chip and a small set of nets which are penalized by extreme detour will be discouraged. Overall, this algorithm shows good congestion reduction, but at a cost of high computational overhead.

IV. PRACTICAL INTEGER LINEAR PROGRAMMING FOR GLOBAL ROUTING

Integer linear programming (ILP) technique has been believed unacceptably slow for global routing in VLSI design, despite that it finds the global optimum for a given instance of problem. In this section, we propose a new ILP formulation for global routing, which is inherently different from the one in [3], [28], and discuss pros/cons of each formulation. In this work, to avoid any confusion, we call the traditional ILP as T-ILP and our new ILP as N-ILP. Both T-ILP and N-ILP are routability-driven, but they adopt different formulations, which make big difference in performance and scalability.

Before the main discussion, we describe Fig. 2 for clear explanation in the following sections. Fig. 2 (a) shows two unrouted nets a and b which are further decomposed into wires (See Section V-A): net a has three wires \((w_{a1}, w_{a2}, w_{a3})\), and net b has one wire \((w_{b})\). For each wire, we can enumerate all the possible routing paths, but for simplicity we show only the paths in the minimum length and with minimum vias as in Fig. 2 (b). Each possible routing path is called a routing candidate of the given wire. In this example, we assume that the routing capacity is 2 for all the edges \((r_{12} = 2, r_{25} = 2, \text{and so forth})\), thus both Fig. 2 (c) and (d) are routable solutions.

A. T-ILP

T-ILP minimizes the maximum congestion over all the edges. Fig. 3 is a T-ILP formulation of Fig. 2 (b) where a variable \(C\) is set to be larger than any congestion on any edge (i.e., the upper bound). The routing result after solving Fig. 3...
is not Fig. 2 (d) but Fig. 2 (c), as Fig. 2 (d) has the maximum congestion 1.0 on $e_{10}$ while Fig. 2 (c) has the maximum congestion 0.5.

Let $E$ be the set of edges in the grid (indexed by $e$), and let $N$ be the set of all feasible routing candidates. Furthermore, let $L(e)$ be the set of routing candidates crossing edge $e$. Suppose $x_{ijk}$ is a binary variable set to 1 if the $k$-th routing candidate of wire $j$ of net $i$ is chosen. Then, Fig. 4 shows a general formulation of T-ILP. Note that the number of routing candidates must be kept small (L-shape or L/Z-shape path) due to practical limitations (e.g. memory). The advantages of T-ILP formulation include:

- As it minimizes the maximum congestion (min-max formulation), it essentially tries to achieve more uniform congestion distribution.

- The solution of T-ILP formulation always includes one routing candidate for each unrouted wire. Thus, it completes routing by itself, and does not need any additional step, unless there is any over-congested edge.

Meanwhile, the drawbacks of T-ILP formulation include:

- When $C$ in Fig. 4 is larger than any $m_e$ (the maximum routing capacity of the edge $e$), the number of over-congested edges will explode. It considers not the overall congestion but the maximum congestion. Therefore, as long as the congestion is smaller than $C$, it is possible to have many over-congested edges.

- All the over-congested edges should be taken care of to meet congestion constraint (otherwise, it is unrouteable by detailed router) by post-processing steps such as rip-up&reroute.

- The T-ILP cannot be efficiently solved with branch-and-bound or branch-and-cut algorithms. This will be explained in Section IV-C.

B. N-ILP

Our proposed N-ILP maximizes the weighted summation of the number of routed wires under the routing capacity constraint. Fig. 5 is a N-ILP formulation of Fig. 2 (b) where each routing candidate is weighted by its length in the objective. The result from Fig. 5 can be either Fig. 2 (c) or Fig. 2 (d), as N-ILP does not care about the maximum congestion, as long as there is no overflow. Fig. 6 shows the general formulation of N-ILP where $a_{ijk}$ is the weight of the routing candidate $x_{ijk}$ and the other notations are the same as in Fig. 4. Again, the number of routing candidates should be kept small (L-shape or L/Z-shape path). The advantages of N-ILP formulation include:

- As each candidate $x_{ijk}$ can have a different weight, other design objectives like timing can easily be incorporated.

- Due to the hard constraint on routing capacity, the solution from N-ILP does not cause any over-congestion on any edge.

- The N-ILP can be efficiently solved with branch-and-bound or branch-and-cut algorithms. This will be explained in Section IV-C.

However, the drawbacks of N-ILP formulation include:

- The N-ILP may produce a biased routing solution in terms of congestion uniformness. For example, if there are two valid solutions with different congestion distributions, it may choose any of both depending on the solver regardless of congestion uniformness (See Fig. 2).

- Different from T-ILP, it may not complete the routing. If the over-congested edge appears, it will give up routing some wires with smaller weight not to violate the hard routing capacity constraint. Thus, N-ILP requires an additional step for complete routing.

C. T-ILP vs. N-ILP

Based on the discussion in Section IV-A and IV-B, we compare both ILP formulations in two aspects: routability and runtime.

1) Routability: As mentioned earlier, both T-ILP and N-ILP maximize the routability, but in different manners: T-ILP minimizes the maximum congestion, but N-ILP maximizes the number of routed wires under the routing capacity constraint. This difference becomes highly distinct, depending on whether the design is under-congested or over-congested.

- For under-congested designs, it is easy for T-ILP and N-ILP to satisfy the routing constraint. Therefore, T-ILP may be superior to N-ILP, as it can make more uniform
congestion distribution which improves manufacturability and crosstalk noise.

- **For over-congested designs,** T-ILP may unnecessarily cause a lot of overflows, as it only cares about the maximum congestion. However, N-ILP itself does not cause any over-congested edges by leaving some wires unrouted. The overflows from T-ILP and the unrouted wires from N-ILP need to be picked up by the following maze routing.

Since, modern VLSI designs are highly congested in general, the advantage of T-ILP is quite trivial.

2) **Runtime:** For a given ILP solver, different ILP formulations may have different runtime complexity. An ILP problem is first solved as linear programming (LP), then branch based algorithm is applied to any fractional variable to find the integral optimal solution. We find that for the most widely used ILP solving algorithms, branch-and-bound or branch-and-cut [33], [34], the N-ILP formulation can be solved much more efficiently than the T-ILP formulation for the same routing problem.

For demonstration purpose, we prepare various routing problems in different problem sizes (in terms of the number of variables), then formulate them into both T-ILP and N-ILP. Fig. 7 shows the normalized runtime of each T-ILP and N-ILP formulation under typical computing environment (See Section VI) with GNU Linear Programming Kit (GLPK) 4.8 with all speedup options turned on. Note that we obtain very similar trend for various algorithms such as branch-and-bound and branch-and-cut with different cutting planes [34], [35]. It is clear that N-ILP is significantly faster than T-ILP, and such speedup becomes more significant for larger problem size, e.g., over 1100 times for some large cases. There are two theoretical explanations why N-ILP can be solved much faster than T-ILP.

- **Since N-ILP is similar to a binary knapsack formulation,** the solution after LP is a near feasible solution with almost all variables non-fractional [33], [36]. However, due to the min-max nature of the objective function, the variables in T-ILP have more incentive to remain fractional after LP as opposed to their counterparts in N-ILP. Consequently, the LP solution of T-ILP is much more fractional than that of N-ILP, resulting in more branches during branch-and-bound or branch-and-cut.
- **The branch-and-bound or branch-and-cut techniques terminate in shorter time,** if more nodes can be fathomed [33]. Unfortunately, the min-max nature of the objective function in T-ILP results in many near optimal solutions. Therefore, the corresponding nodes cannot be fathomed efficiently and the branch tree grows needlessly.

3) **Summary:** As discussed in Section IV-C.1 and IV-C.2, N-ILP is significantly faster than T-ILP, and the solution quality from N-ILP is similar to that from N-ILP for the over-congested design. Thus, N-ILP is expected to work better for the modern VLSI designs. Our proposed N-ILP is adopted in BoxRouter in Section V, in progressive manner with box expansion concept.

![Fig. 7. Runtimes of T-ILP and N-ILP are compared. It shows that N-ILP is much faster and more scalable for larger problems than T-ILP.](image)

V. **BoxRouter**

In this section, we present our new global router, BoxRouter, which is based on congestion-initiated box expansion. BoxRouter progressively expands a box which initially covers the most congested region only, but finally covers the whole circuit. After every expansion, a circuit is divided into two sections, inside the box and outside the box. BoxRouter uses different routing strategies for each section to maximize routability and minimize wirelength. Consider Fig. 8 (a), where two wires (a and b) are inside the box, while the other wires (c and d) are not inside the box. The routing capacity inside the box is more precious to a and b than c and d for two reasons:

- If a and b are not routed within the box, wirelength will increase due to detour.
- c and d may have another viable routing path outside box which does not waste the routing capacity inside the box.

Therefore, BoxRouter first routes as many wires inside the box as possible with N-ILP in Section IV-B, maximally utilizing the routing capacity inside the box. Then, for the wires which cannot be routed by N-ILP within the box (due to insufficient routing capacities), BoxRouter detours them by adaptive maze routing with the following two strategies:

- **Inside the box,** use the routing capacities as much as possible (greedily), as the wires inside the box have priority over those outside the box.
- **Outside the box,** use the routing capacities conservatively, as the wires outside the box may need them later for their viable routing paths.

![Fig. 8. The basic concept of BoxRouter](image)
Those two strategies keep the wire density of the circuit as in Fig. 8 (b), and make the wires detour the more congested region to maximize the routability with minimum wirelength overhead.

The overall flow of BoxRouter is in Fig. 9, which will be explained in detail in the rest of this section. Section V-A describes the preprocessing for BoxRouter. Section V-B illustrates PreRouting for congestion estimation and routing speedup. Section V-C explains BoxRouting, the main idea of BoxRouter which includes progressive ILP (PILP), adaptive maze routing (AMR), and box expansion. Finally, Section V-D shows how PostRouting improves wirelength and routability further while controlling the trade-off between them.

A. Steiner Tree and Net Decomposition

A net can be decomposed into two pin wires with Rectilinear Minimum Steiner Tree as shown in Fig. 10. In BoxRouter, Flute [37] and GeoSteiner [38], [39] are tested for Steiner tree construction, but Flute is finally adopted due to its small computational overhead. Note that different Steiner tree algorithms such as timing-driven or congestion-driven Steiner tree algorithms can be used in BoxRouter as well. A special wire which does not need a bend is called a flat wire [18]. For example, wire a-e, e-d, e-f and b-f in Fig. 10 (b) are flat wires, while wire f-c requires at least one bend to be routed. Each wire from a net becomes a single routing object. However, the net is finally routed, only if all the wires from a net are routed. Routing each wire from a single net separately may have downside of losing information on other wires, resulting in suboptimal routing. This issue is addressed in adaptive maze routing in Section V-C.2.

B. PreRouting and Initial Box

PreRouting simply routes as many flat wires as possible via the shortest path without creating any overflow as in Algorithm 1. As bulk of nets are destined to be routed in simple patterns (L-shape or Z-shape) [20], [23], [25], Pre-Routing can improve the runtime without degrading the final solution. More importantly, if enough number of wires can be routed by PreRouting, the global congestion can be captured with reasonable accuracy. According to our experiments for the tested benchmarks, about 60% of the final wirelength on average can be routed with tiny computational overhead by PreRouting. Fig. 11, shows two congestion maps, one after PreRouting and the other one after BoxRouting where more congested area is brighter. It shows that congestion hotspots in Fig. 11 (b) can be predicted from Fig. 11 (a) by PreRouting. A box which encompasses the four G-cells in the most congested area will be created as shown in Fig. 12 (a) as a starting point of BoxRouting. Note that if there are two most congested areas, then the one closer to the center of the circuit is selected.

C. BoxRouting

In this subsection, BoxRouting will be explained with Fig. 12. BoxRouting consists of three steps, progressive integer linear programming routing, adaptive maze routing, and box expansion as in Fig. 9. Those three steps are repeated until the
Fig. 12. BoxRouting example

(a) Initial box is created on the hotspot which is estimated by PreRouting.
(b) Box $i$ with wires which will be routed by BoxRouting is shown.
(c) Wires within Box $i$ will be routed by progressive ILP.
(d) Unrouted wire $b$ after progressive ILP is routed by adaptive maze routing.
(e) Box $i$ is expanded, and more wires are enclosed by Box $i+1$.
(f) BoxRouting is performed with Box $i+1$.

Fig. 13. Progressive ILP formulation of Fig. 12 (c)

\[
\begin{align*}
\text{max} : & \ x_{b1} + x_{b2} + x_{f1} + x_{f2} + x_{h1} + x_{h2} \\
\text{s.t.} : & \begin{cases} 
    x_{b1} + x_{b2} \leq 1 \\
    x_{f1} + x_{f2} \leq 1 \\
    x_{f2} = 0 \\
    x_{h1} + x_{h2} \leq 1 \\
    x_{b1} + x_{f1} + x_{h1} \leq c_{AB} \\
    x_{b1} + x_{h1} \leq c_{BD} \\
    x_{h2} + x_{h2} \leq c_{AC} \\
    x_{b2} + x_{h2} \leq c_{CD} 
\end{cases}
\end{align*}
\]

Fig. 14. General progressive ILP formulation

\[
\begin{align*}
\text{max} : & \sum_{i} \{x_{i1} + x_{i2}\} \\
\text{s.t.} : & \begin{cases} 
    x_{i1} + x_{i2} \leq 1 \\
    x_{i2} = 0 \\
    \sum_{e \in x_{i,j}} x_{ij} \leq c_e
\end{cases} \\
\forall i \in W_{box} & \forall i \in W_{box} & \forall i \in W_{box} & \forall i \in W_{box} \\
\forall i \in W_{box} & \forall i \in W_{box} & \forall i \in W_{box} & \forall i \in W_{box}
\end{align*}
\]

until BoxRouting, if PreRouting gives up routing them due to any overflow, or new Steiner points introduced by adaptive maze routing (AMR) (explained later in this section) convert a non-flat wire into a flat wire. For efficient routing as mentioned in the beginning of this section, only wires within the box will be routed by PILP and AMR.

In Fig. 12 (c), the wires within the box are shown with G-cells ($v_A, v_B, v_C$ and $v_D$), and the corresponding PILP formulation for maximum routability is shown in Fig. 13. To minimize the number of vias, two L-shape routing candidates ($x_{b1}, x_{b2}$ and $x_{h1}, x_{h2}$) are considered for each wire in our PILP formulation, but only one routing candidate ($x_{f1}$ and $x_{f2}=0$) is considered for flat wires. General PILP formulation is shown in Fig. 14, where $c_e$ is the available routing capacity on edge $e$ (See Table I), $W_{box}$ is a set of unrouted wires within the current box, and $W_{flat}$ is a set of flat wires.

Different from the hierarchical ILP [24], our PILP progressively routes a part of the circuit, which is covered by each expanding box. This box expansion limits the problem size

Fig. 12. BoxRouting example

Fig. 13. Progressive ILP formulation of Fig. 12 (c)
such that PILP which is NP-hard can be solved efficiently. Three advantages of our PILP can be summarized as follows:

- Basic formulation is the same as N-ILP of Section IV-B, inheriting its advantages in runtime and scalability.
- Even though the last box can cover the whole circuit, the PILP size remains tractable, as N-ILP is performed on the wires between two successive boxes like between Box $i$ and Box $i+1$ in Fig. 12 (e).
- As shown in Fig. 12 (e), the newer box always contains the older box. Consequently, the solution from the older PILP is reflected in the newer PILP formulation, providing smooth transition between two successive problems for high quality solution.

Due to the limited routing capacity of each edge, some wires may not be routed with the above PILP. $x_{i1} + x_{i2} \leq 1$ in Fig. 14 relaxes the routing constraint such that some wires may not be routed if the overflow occurs. For example, assuming $m_{BD} = m_{CD} = 2$ and $x_{i1} = 1$, the wire $b$ cannot be routed with ILP ($x_{i1} = x_{i2} = 0$), as two prerouted wires on $e_{CD}$, and one prerouted wire with the wire $h$ ($x_{h1} = 1$) on $e_{BD}$ consume all the routing capacities. For this case, the wire $b$ is routed by AMR as in Fig. 12 (d) with the routing cost from Algorithm 3.

2) Adaptive Maze Routing (AMR): Algorithm 3 returns a unit cost as long as $e_{XY}$ is inside box and still has available routing capacity (line 2, 3). Otherwise, it returns a cost inversely proportional to the available routing capacities (line 1). This cost function makes maze routing adaptively find the best routing path such that the shortest path inside the box for wirelength minimization, but the most idle path outside the box for routability maximization. Note that the resource outside the box should be used conservatively, as the wires outside the current box may need them later. If too big detour is required to avoid small overflows, AMR may return a path with overflows for the least overall cost.

For the maze router implementation, we propose a multi-source multi-target with bridge (MMB) maze routing model for higher efficiency as illustrated in Fig. 15. Consider the example in Fig. 15 (a) where the source G-cell $S$ and the target G-cell $T$ are to be routed and the congestion is represented as shaded region. To avoid congestion, a simple maze routing can easily find the routing path path2 instead of path1. However, as the goal is to make $S$ and $T$ electrically connected, we can achieve electrical connection as well as shorter wirelength by alternatively routing $x$ and $y$ shown as path3. The other example in Fig. 15 (b) shows the case where the routing between $b$ and $c$ is detoured due to congestion. In this case, even though path1 is the shortest path between $S$ and $T$ without any congestion issue, the path $S-x-y-T$ shown as path2 – path3 is the better routing path, because it shares and utilizes the existing routed path path3, resulting in the shorter total wirelength.

![Fig. 16. Multi-source multi-target with bridge maze routing model](image-url)

Aware of the above mentioned cases, the proposed multi-source multi-target with bridge (MMB) based maze routing in Fig. 16 is implemented for AMR. The basic idea behind MMB is to make the maze router honor the existing partial routed paths of the net for shorter wirelength and less congestion. In detail, the proposed model is based on three different groups of G-cells as in Fig. 16.

- **Source group**: a group of G-cells which are electrically connected to the source G-cell $S$.
- **Target group**: a group of G-cells which are electrically connected to the target G-cell $T$.
- **Bridge group**: multiple groups of G-cells on the partial routing paths which are connected to neither the source $S$ nor the target $T$.

Note that identifying each group of G-cells can be done with any graph traversal algorithm with trivial computational

### Algorithm 2 BoxRouting

**Input:** A list of wires $W$ in box $B$

1. Solve progressive ILP with $W$
2. for each $w$ in $W$ do
3.   if $w$ is unrouted then
4.     Perform adaptive maze routing for $w$
5.   end if
6. end for

### Algorithm 3 Adaptive Maze Routing Cost for BoxRouting

**Input:** G-cell $V_x, V_y$, Box $B$

1. Cost $C = m_{xy} - e_{xy}$
2. if $e_{xy}$ is inside $B$ and $e_{xy} > 0$ then
3.   $C = 1$
4. end if

**Output:** $C$

---

Fig. 15. Efficient multi-source multi-target maze routing examples are illustrated. More efficient alternative paths are found by considering multiple sources and targets.
overhead. There can be multiple bridge groups in case that many routed paths (from PreRouting or previous AMR) are not connected with each other.

Flooding of the maze routing is started from the all the G-cells in the source group, and is terminated when any G-cell not connected with each other.

There can be multiple bridge groups in case that congestion-driven Steiner tree algorithm is used in this work.

As AMR in BoxRouting uses conservative strategy outside the box as in Algorithm 3 (finding the most idle routing path outside the box), it may create unnecessary detour and overflow. Thus, PostRouting simply reroutes wires to remove unnecessary overhead with box expansion initiated from the most congested region, as done in BoxRouting. In detail, a wire in the more congested region will be rerouted first, and such rerouted wire can release the routing capacity, as it may find the better routing path. Then, the surrounding wires can be rerouted with the released routing capacity, potentially reducing wirelength and overflow again. This chain reaction propagates from the most congested region to less congested regions along the box expansion. Consider the example in Fig. 17 where two wires \( x \) and \( y \) are routed around the G-cell \( V_z \). Before the PostRouting (thus, during BoxRouting), the wire \( x \) detours \( V_z \) during AMR in Section V-C.2 due to high congestion in \( V_z \) in spite of the available routing capacity \( R \). However, if \( R \) is still available after BoxRouting is finished, then there is no reason to leave \( R \) available at a cost of longer

### Algorithm 4 Adaptive Maze Routing

**Input:** Source \( s \) and target \( t \) of net \( N \) with box \( B \)

1. Find source group \( G_s \) of \( s \)
2. Find target group \( G_t \) of \( t \)
3. Find all bridge groups \( G_{b1}, G_{b2}, \ldots \) of \( N \)
4. A priority queue \( Q = \emptyset \)
5. for each G-cell \( V_x \) in \( G_s \) do
6. Cost \( T_x \) of \( V_x = 0 \)
7. Enqueue \( V_x \) into \( Q \)
8. end for
9. Best target G-cell \( V_b = \emptyset \), \( T_b = \infty \)
10. while \( Q \) is not empty do
11. dequeue a G-cell \( V_x \) from \( Q \)
12. if \( T_x \geq T_b \) then
13. break
14. end if
15. for each adjacent G-cell \( V_y \) of \( V_x \) do
16. \( T_n = \) Algorithm 3 \((V_x, V_y, B)\)
17. \( T_y = T_x + T_n \)
18. if \( V_y \in G_t \) and \( T_y < T_b \) then
19. \( V_b = V_y \), \( T_b = T_y \)
20. else if \( V_y \in G_{b1} \) then
21. for each G-cell \( V_z \) in \( G_{bi} \) do
22. \( T_z = T_y \)
23. Enqueue \( V_z \) into \( Q \)
24. end for
25. else
26. Enqueue \( V_y \) into \( Q \)
27. end if
28. end for
29. end while
30. \( P = \) Backtrace the best path from \( V_b \) to any G-cell of \( G_s \)

**Output:** \( P \)
wirelength, $x$ can be rerouted through $R$ to minimize the wirelength without causing any overflow. After $x$ is rerouted, the wire $y$ which is detoured during BoxRouting due to $x$ can be rerouted as well using the routing capacity released after $x$ is rerouted, thus reducing wirelength again.

**Algorithm 5 Maze Routing Cost for PostRouting**

**Input:** G-cell $V_x, V_y$, Param $K$

1. Cost $C = K$
2. if $c_{xy} > 0$ then
3. $C = 1$
4. end if

**Output:** $C$

AMR of Section V-C.2 is used again for PostRouting, but with a different routing cost function in Algorithm 5, where a user-defined parameter $K$ is introduced. The parameter $K$ controls the trade-off between wirelength and routability (overflow), by setting the cost of each overflow as $K$. Thus, higher $K$ will discourage overflow at a cost of wirelength increase (more detours), but lower $K$ will suppress detour at a cost of overflows. The effectiveness of parameter $K$ is discussed in Section VI.

Our PostRouting is more efficient than the widely used Rip-up&Reroute (R&R), as PostRouting makes a wire voluntarily release a routing capacity (this happens, only when the solution improves) during its rerouting, while R&R deprives it from a wire in the congested region without guaranteeing any improvement. Although R&R and PostRouting target for less congestion, the approaches are different. While R&R rips up the already routed wires to secure routing capacity directly, PostRouting makes more routing capacity available indirectly by shortening the wirelength of each wire (wirelength is linearly proportional to the number of routing capacities in use). PostRouting is guaranteed to find the equal or better routing path for the given objective function, as the current routing path can always be found as the worst case routing path. Thus, the routing quality can be improved gradually by repeating PostRouting.

**VI. EXPERIMENTAL RESULTS**

We implement BoxRouter in C++. All the experiments are performed on a 2.8 GHz Pentium-4 Linux machine with 2G RAM. Flute [37] with high accuracy option is used for Rectilinear Minimum Steiner Tree, and GNU Linear Programming Kit (GLPK) 4.8 [34] is used as ILP solver. We use ISPD98 IBM benchmarks [1] for our experiments. Table II summarizes each ISPD98 IBM benchmark circuit and its corresponding grid graph model. The lower bound wirelength of each circuit is computed by the most accurate GeoSteiner 3.1 [37], [39].

Table III shows the routing completion percentage after PreRouting. On average, 61% of the lower bound wirelength can be routed after PreRouting which is enough to capture the overall congestion as well as the most congested region. Further, over 61% routing completion even before the main routing phase will improve the runtime.

Fig. 18 shows the overflow and runtime by the amount of box increment (See Section V-C) for one benchmark. It clearly shows that with larger box increment, the overflow decreases, but the runtime increases exponentially. While the wirelength varies only by 0.11%, the overflow decreases by 30%, but the runtime increases by 500%. It indicates that with larger box increment during box expansion of BoxRouting, the solution quality can be improved at a cost of runtime.

The effectiveness of parameter $K$ (See Section V-D) is shown in Fig. 19. It shows that with larger $K$, overflow decreases exponentially, but wirelength increases logarithmically. We constantly find that overflow saturates faster than
wirelength, and the best trade-off occurs between $K=10$ to $K=15$ for all the tested benchmarks. Fig. 20 also shows the runtime is independent of parameter $K$. The runtime variations ($stdev$) of ibm02 and ibm10 are only 0.7% and 0.5% respectively, while $K$ varies from 1 to 30.

Table IV shows the routing results by BoxRouter with $K=10$ and $K=15$, the best trade-off found in Fig. 19. It shows that BoxRouter has on average 3.4% and 3.7% wirelength overhead (regarding the lower bound wirelength) for $K=10$ and $K=15$ respectively, and provides high quality solutions for larger circuits with small overflows.

Table VI compares the congestion-initiated box expansion with the random-initiated box expansion when $K=15$. It shows that the box expansion initiated from the most congested region can improve the number of overflow by 33.1% on average, proving that it is more effective than randomly initiated one in terms of congestion.

For thorough comparison, we download two available global routers, Labyrinth 1.1 [1], [25] and Fengshui 5.1 (which has the newest implementation of the Chi dispersion router) [2], [41], and implement multicommodity flow-based global router [3] in C++ (the binary is not available from the author). Note that we use the same routine for Rectilinear Minimum Steiner Tree, congestion estimation, and maze routing for fair comparison in the multicommodity flow-based global router implementation. Although the results of Labyrinth and Fengshui are reported in [2], we reproduce the results due to the recent update in the benchmarks [1].

Table V shows the experimental results and comparison for Labyrinth and Fengshui, and Table VII for the multicommodity flow-based router. As there is a trade-off between

![Fig. 18. Overflow and runtime change by box increment for ibm04](image1)

![Fig. 19. Routability and wirelength trade-off by Parameter $K$](image2)

![Fig. 20. Runtime change by Parameter $K$](image3)

**TABLE IV**

<table>
<thead>
<tr>
<th>circuit</th>
<th>BoxRouter ($K=10$)</th>
<th>BoxRouter ($K=15$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>lw.len</td>
<td>wlen</td>
</tr>
<tr>
<td>ibm01</td>
<td>60142</td>
<td>65029</td>
</tr>
<tr>
<td>ibm02</td>
<td>165863</td>
<td>177921</td>
</tr>
<tr>
<td>ibm03</td>
<td>145678</td>
<td>149466</td>
</tr>
<tr>
<td>ibm04</td>
<td>162734</td>
<td>171044</td>
</tr>
<tr>
<td>ibm05</td>
<td>409709</td>
<td>409747</td>
</tr>
<tr>
<td>ibm06</td>
<td>275868</td>
<td>281715</td>
</tr>
<tr>
<td>ibm07</td>
<td>363537</td>
<td>374910</td>
</tr>
<tr>
<td>ibm08</td>
<td>402412</td>
<td>408997</td>
</tr>
<tr>
<td>ibm09</td>
<td>411260</td>
<td>417599</td>
</tr>
<tr>
<td>ibm10</td>
<td>574407</td>
<td>590738</td>
</tr>
<tr>
<td>average</td>
<td>3.4</td>
<td>3.7</td>
</tr>
</tbody>
</table>

*a* wirelength hereafter in this section  
*b* overflow hereafter in this section  
*c* wirelength overhead
wirelength and routability, we choose the parameter $K$ of BoxRouter of Table V with wirelength constraint such that wirelength from BoxRouter is as small as or smaller than those from Labyrinth and Fengshui for fair comparison. Regarding Table VII, we first carefully choose the parameters of the multicommodity flow-based router for each benchmark such that the best results are yielded within 25 phases (the maximum phase in [3]), then simulate ibm01, ibm02, ibm04 and ibm07 (circuits with non-zero overflow in Table V) again for BoxRouter without any constraint.

As shown in Table V, BoxRouter outperforms Labyrinth and Fengshui by wide margin. In terms of wirelength and overflow, BoxRouter can reduce the wirelength by 14.3%, the overflow by 91.7% compared with Labyrinth, and improve the overflow by 79% with similar wirelength (actually 0.8% better) compared with Fengshui. Also, BoxRouter is 3.3x and 2.0x faster than Labyrinth and Fengshui respectively. Multicommodity flow-based router and BoxRouter show very comparable overflow as shown in Table VII. However, BoxRouter is on average 15.7x, up to 29x faster, and produces 4.2% shorter wirelength on average than multicommodity flow-based router. It implies that BoxRouter can provide high quality global routing solution with significantly less design turn-around time.

Figure 21 shows pie chart for cputime break down averaged from all the benchmark circuits. PreRouting takes negligible amount of total cputime (1.4%), while routing over 60% of wires. BoxRouting which may be considered the slowest part of BoxRouter due to ILP takes about 25%, while PostRouting takes over 57%. This proves that the proposed PILP is significantly fast while providing high quality solution. On the other hand, PostRouting which is mainly the maze routing is the current bottleneck in runtime for BoxRouter.

So far, all the results from BoxRouter are with constant

### Table V

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Labyrinth 1.1</th>
<th>Fengshui 5.1</th>
<th>BoxRouter</th>
<th>Improv. on Labyrinth</th>
<th>Improv. on Fengshui</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm01</td>
<td>65098</td>
<td>65193</td>
<td>626</td>
<td>26.3</td>
<td></td>
</tr>
<tr>
<td>ibm02</td>
<td>178924</td>
<td>179086</td>
<td>0.0</td>
<td>40.0</td>
<td></td>
</tr>
<tr>
<td>ibm03</td>
<td>149895</td>
<td>149879</td>
<td>9</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>ibm04</td>
<td>171812</td>
<td>171756</td>
<td>342</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>ibm05</td>
<td>409744</td>
<td>409747</td>
<td>0</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>ibm06</td>
<td>282875</td>
<td>282802</td>
<td>0.3</td>
<td>44.4</td>
<td></td>
</tr>
<tr>
<td>ibm07</td>
<td>375584</td>
<td>376247</td>
<td>81</td>
<td>0.2</td>
<td>29.6</td>
</tr>
<tr>
<td>ibm08</td>
<td>409025</td>
<td>409584</td>
<td>31</td>
<td>-0.1</td>
<td>46.6</td>
</tr>
<tr>
<td>ibm09</td>
<td>418131</td>
<td>418023</td>
<td>4</td>
<td>0.0</td>
<td>42.9</td>
</tr>
<tr>
<td>ibm10</td>
<td>592784</td>
<td>591820</td>
<td>10</td>
<td>0.2</td>
<td>47.4</td>
</tr>
<tr>
<td>Average</td>
<td>65193</td>
<td>65193</td>
<td>626</td>
<td>26.3</td>
<td></td>
</tr>
</tbody>
</table>

### Table VI

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Random Init.</th>
<th>Congestion Init.</th>
<th>Imprv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm01</td>
<td>65098</td>
<td>65193</td>
<td>26.3</td>
</tr>
<tr>
<td>ibm02</td>
<td>178924</td>
<td>179086</td>
<td>0.0</td>
</tr>
<tr>
<td>ibm03</td>
<td>149895</td>
<td>149879</td>
<td>0.0</td>
</tr>
<tr>
<td>ibm04</td>
<td>171812</td>
<td>171756</td>
<td>342</td>
</tr>
<tr>
<td>ibm05</td>
<td>409744</td>
<td>409747</td>
<td>0.0</td>
</tr>
<tr>
<td>ibm06</td>
<td>282875</td>
<td>282802</td>
<td>0.3</td>
</tr>
<tr>
<td>ibm07</td>
<td>375584</td>
<td>376247</td>
<td>81</td>
</tr>
<tr>
<td>ibm08</td>
<td>409025</td>
<td>409584</td>
<td>31</td>
</tr>
<tr>
<td>ibm09</td>
<td>418131</td>
<td>418023</td>
<td>4</td>
</tr>
<tr>
<td>ibm10</td>
<td>592784</td>
<td>591820</td>
<td>10</td>
</tr>
<tr>
<td>Average</td>
<td>65193</td>
<td>65193</td>
<td>26.3</td>
</tr>
</tbody>
</table>

### Table VII

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Multicommodity</th>
<th>BoxRouting</th>
<th>Improv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm01</td>
<td>68981</td>
<td>676745</td>
<td>1.9</td>
</tr>
<tr>
<td>ibm02</td>
<td>190418</td>
<td>182268</td>
<td>4.3</td>
</tr>
<tr>
<td>ibm03</td>
<td>160755</td>
<td>152199</td>
<td>5.9</td>
</tr>
<tr>
<td>ibm04</td>
<td>176610</td>
<td>173778</td>
<td>1.6</td>
</tr>
<tr>
<td>ibm05</td>
<td>410954</td>
<td>409747</td>
<td>4.9</td>
</tr>
<tr>
<td>ibm06</td>
<td>296981</td>
<td>283235</td>
<td>4.9</td>
</tr>
<tr>
<td>ibm07</td>
<td>408510</td>
<td>394170</td>
<td>3.5</td>
</tr>
<tr>
<td>ibm08</td>
<td>429913</td>
<td>415025</td>
<td>3.5</td>
</tr>
<tr>
<td>ibm09</td>
<td>442514</td>
<td>418615</td>
<td>5.9</td>
</tr>
<tr>
<td>ibm10</td>
<td>634247</td>
<td>593186</td>
<td>6.5</td>
</tr>
<tr>
<td>Average</td>
<td>676745</td>
<td>676745</td>
<td>1.9</td>
</tr>
</tbody>
</table>

---

**Fig. 21.** Pie chart for average cputime break down
In order to cope with the increasing impact of interconnect on system performance, we present an efficient global router, BoxRouter, to maximize the routability with minimum wirelength. Experimental results show that BoxRouter outperforms the state-of-the-art publicly available global routers in terms of wirelength, routability, and runtime. As the BoxRouter is still in beta version, we believe that further improvement can be achieved with multiple box expansions, faster ILP solver and so on. Current implementation of BoxRouter is available at www.cerc.utexas.edu/utda. We plan to address timing, crosstalk, and manufacturability issues on the top of the BoxRouter framework.

VII. CONCLUSION

In order to cope with the increasing impact of interconnect on system performance, we present an efficient global router, BoxRouter, to maximize the routability with minimum wirelength. Experimental results show that BoxRouter outperforms the state-of-the-art publicly available global routers in terms of wirelength, routability, and runtime. As the BoxRouter is still in beta version, we believe that further improvement can be achieved with multiple box expansions, faster ILP solver and so on. Current implementation of BoxRouter is available at www.cerc.utexas.edu/utda. We plan to address timing, crosstalk, and manufacturability issues on the top of the BoxRouter framework.

VIII. ACKNOWLEDGMENT

The authors would like to thank Prof. Patrick Madden from SUNY Binghamton and Dr. Christoph Albrecht from Cadence Berkeley Lab for helpful discussions, Ilyas Mohamed Iyoob in Dept. Operation Research in Univ. of Texas at Austin for the comparison of ILP formulations, and Kun Yuan and Katrina Lu in UTDA Lab in Univ. of Texas at Austin for proofreading.

REFERENCES


[34] http://www.gnu.org/software/glpk/glpk.html/


[41] http://vlsicad.cs.binghamton.edu/