DGR: Differentiable Global Router

Wei Li[*][†] Rongjian Liang[§] Anthony Agnesina[§] Haoyu Yang[§] Chia-Tung Ho[§] Anand Rajaram[§]
Haoxing Ren[§]
Carnegie Mellon University[*] NVIDIA[§]

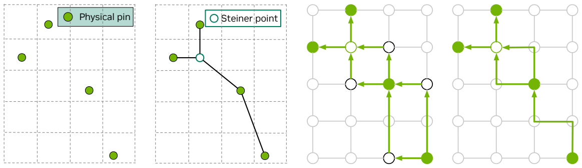
Abstract

Modern VLSI design flows necessitate fast and high-quality global routers. In this paper, we introduce DGR, a differentiable global router capable of concurrent optimization for hundreds of thousands of nets [1]. Our innovation lies in the development of a routing Directed Acyclic Graph (DAG) forest to represent the 2D pattern routing space for all nets, enabling coordinated selection of Steiner trees and 2-pin routing paths from a global perspective. For efficient search within the DAG forest, we relax the discrete search space to be continuous and develop a differentiable solver accelerated by deep learning toolkits on GPUs. Experimental results demonstrate that DGR substantially mitigates routing overflow while concurrently reducing total wirelengths from 0.95% to 4.08% and via numbers from 1.28% to 2.54% in congested testcases compared to state-of-the-art academic global routers. Additionally, DGR exhibits favorable scalability in both runtime and memory with respect to the number of nets.

1 Introduction

The continuously shrinking technology node has led to a notable increase in the density and scale of VLSI circuits, posing substantial challenges for routing algorithms. To manage this complexity, VLSI routing has been divided into two sub-problems: global routing, which focuses on generating coarse-grain routing guides, and detailed routing, where routing tracks are assigned for all connections guided by these routing guides while ensuring compliance with design rule constraints. The outcome of global routing plays a pivotal role in shaping the efficiency of detailed routing and the overall quality of post-route circuits. Additionally, the efficiency of global routing is crucial, as it contributes to streamlining timing- and congestion-driven floor-planing and placement by providing accurate interconnect information [1]. Thus, modern VLSI design workflows require fast and high-quality global routing algorithms.

A Directed Acyclic Graph (DAG)-based global router called CUGR2 is proposed in [2] and achieves state-of-the-art results in the academic community. Figure 1 depicts the construction of a routing DAG, which serves as the data structure for representing available pattern routing paths for a net. Initially, a rectilinear Steiner minimum tree (RSMT) [3] connects all pins, breaking the multi-pin net into 2-pin sub-nets (Figure 1b). Subsequently, routing paths conforming to available patterns are established for each 2-pin sub-net. The final routing DAG (depicted in Figure 1c) comprises vertices representing pins, Steiner points, or turning points, and edges representing interconnect wire segments within routing paths. Like other sequential algorithms, CUGR2 optimizes one net at a time, employing a dynamic programming-based algorithm to determine the best pattern routing paths and layer assignments. However, sequential algorithms do not guarantee optimal solution among all nets because of its sequential heuristic. Moreover, its sequential heuristic falls short in addressing routing congestion from a global perspective, possibly leading to unnecessary iterations of rip-up and reroutes. Combinatorial optimization techniques [4, 5] could concurrently optimize multiple nets. But they are often too slow for modern VLSI circuits.



(a) Original Net (b) Generated routing tree (c) A routing DAG by CUGR2 (d) A routing DAG by DGR

Figure 1: An example from a multi-pin net (a) to routing DAGs (c or d). (b) is the rectilinear Steiner minimum tree (RSMT) by FLUTE. Edges in a CUGR2 DAG represent interconnect wire segments, whereas edges in a DGR DAG symbolize potential 2D routing paths for 2-pin sub-nets.

In this work, we propose a novel Differentiable Global Router, named **DGR**, to enable concurrent optimization of hundreds of thousands of nets. It involves the creation of a **routing DAG forest** (depicted in Figure 2 (a)) to concisely represent the 2D pattern routing space for all nets. Here, a single net may be associated with multiple routing DAGs, with each DAG signifying a distinct routing tree topology candidate for that net (Figure 1(d)). Within each routing DAG, individual edges symbolize potential 2D routing path candidates for 2-pin sub-nets. With a DAG forest, the 2D global routing problem is formulated as selecting a routing DAG for each net, followed by choosing one 2-pin path for each 2-pin sub-net within the DAG, such that the total wire length, via count and overflow are minimized. To enable scalable and effective search within the DAG forest, we relax the discrete search space to be continuous and develop an end-to-end differentiable solver (as shown in Figure 2) accelerated by deep learning toolkits on GPUs. Additionally, we introduce a Gumbel-Softmax technique with temperature annealing and top-p selection to bridge the gap between the continuous search space and discrete routing solutions. We summarize our contributions as follows:

- We propose a GPU-accelerated differentiable global routing framework, named DGR, designed for concurrent optimization for hundreds of thousands of nets. To our best knowledge, it is the first GPU-accelerated concurrent global router.
- We propose establishing a routing DAG forest to represent the search space encompassing routing tree topologies and 2-pin routing paths for all nets in a given layout. This facilitates the coordinated selection of Steiner trees and routing paths for all nets from a global perspective.
- We propose a differentiable algorithm for scalable and efficient search within the DAG forest. Gumbel-Softmax technique with temperature annealing and top-p selection are introduced to further enhance the solution quality.

[1] The source code is released on: https://github.com/NVlabs/Differentiable-Global-Router

† Work performed while at NVIDIA.

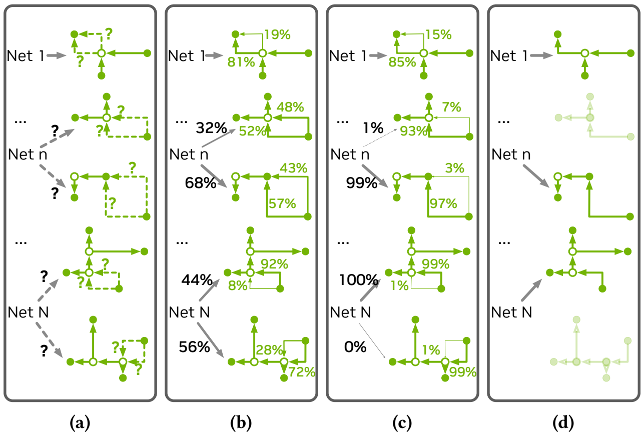


Figure 2: Differentiable search in a DAG forest. (a) Optimal selection of routing tree topologies and 2-pin routing paths is initially unknown. (b) Continuous relaxation of the search space by placing a probability on each routing tree topology candidate and each 2-pin path candidate. (c) Optimization of the probabilities in a differentiable manner. (d) Inducing the final routing trees from the optimized probabilities.

- Experimental results demonstrate DGR easily scales to solve a million-level benchmark, and outperforms the state-of-the-art academic global routers.

2 Prior Works

Global routing approaches can be categorized into two major groups: concurrent [5–7] and sequential [8–13], based on whether they handle one or multiple nets simultaneously. Sequential methods often utilize a “rip-up and reroute" framework [1]. They commence with an initial routing solution and then iteratively improve it, addressing one net at a time. In each iteration, a net that traverses a congested area is removed (rip-up) and subsequently rerouted to bypass the congested regions. The sequencing of net routing plays a vital role in determining the ultimate routing quality. And sequential approaches can stagnate on local minima due to their refinement heuristics. On the other hand, concurrent approaches try to handle multiple nets simultaneously. They are generally based on combinatorial optimization techniques, such as multi-commodity flow [4] and integer linear programming [5–7]. While concurrent techniques have demonstrated their ability to yield superior routing outcomes, particularly in challenging routing scenarios, they often exhibit noticeably slower computational speeds in contrast to sequential methodologies. Recently, GPU-accelerated global routing [10–12] has garnered considerable attention, achieving approximately a tenfold increase in speed while maintaining routing quality. However, the majority of these approaches rely on “parallelizing" traditional sequential algorithms in GPUs. While these methods significantly enhance runtime, the quality of the routing result is still limited by the traditional sequential-based algorithms. Our work falls within the realm of concurrent approaches, yet it not only tries to deliver superior performance but also good scalability comparable to sequential methods by effectively harnessing the massive parallelism offered by GPUs in a distinct novel way.

Another categorization of global routing is based on the dimensionality of routing: 2D [5–8] and 3D [2, 9–12]. In 2D global routing, routing is confined to the horizontal and vertical directions, necessitating a subsequent assignment of routing layers for each wire. Conversely, 3D global routing spans all three spatial dimensions, determining wire layers within the routing process. In theory, 3D routing offers the potential for superior solutions by enabling more efficient utilization of 3D routing resources. However, in practice, 2D approaches often achieve comparable quality results with significantly shorter runtime. Thus, our work adopts the 2D approach.

3 Formulation of Pattern Routing Based on DAG Forests

3.1 Routing DAG Forest

A routing DAG forest is a mathematical structure to systematically describe the 2D pattern routing space for all the nets. A single net may be associated with multiple routing DAGs, with each DAG signifying a distinct routing topology for that net. In each routing DAG, vertices denote pins and Steiner points, and edges represent potential 2D routing paths for two-pin sub-nets within the specified routing topology. These paths can take various forms, including L-/Z-/C-shape pattern routing, monotonic routing, or even maze routing. The realization of a 2D pattern routing solution entails the selection of a routing DAG for each net, followed by finding one routing path for each 2-pin sub-net within the DAG, as shown in Figure 2 (d). In contrast to CUGR2 [2], which addresses one net at a time and focuses on a single Steiner tree topology in each instance, our routing DAG forest allows multiple DAGs for each net and facilitates the coordination of DAG and DAG edge selection across all nets in a global view.

The construction of the DAG forest has a direct impact on the runtime and quality of DGR outcome. Section 4.2 will illustrate how we construct the DAG forest in this work. As a future direction, we plan to explore the adaptive expansion of the forest by introducing new DAGs and DAG edges for nets in congested areas when necessary.

3.2 Formulation of Pattern Routing

This work focuses on 2D pattern routing. The dynamic programming-based layer assignment and maze routing-based refinement presented in [2] are applied to our pattern routing outcome to generate the final 3D routing solution. Given a routing DAG forest constructed by techniques in Section 4.2, the objective of 2D pattern routing is to select the best routing DAGs (routing trees) and DAG edges (2-pin paths) for all the nets such that the total wire length, number of vias, and routing overflow are minimized. Before delving into the problem formulation, we need to define some terminology:

Let $\mathcal{N}$ be the set of input nets. The first task is to construct a routing DAG forest $\mathcal{F} = \{\mathcal{T}, \mathcal{S}, \mathcal{P}\}$, where $\mathcal{T}$ is the routing tree candidate pool, $\mathcal{S}$ is the set of 2-pin sub-nets in all routing trees, and $\mathcal{P}$ represents 2-pin path candidate pool. Let $i \in \mathcal{P}$ be a specific 2-pin path candidate, then $subnet(i) \in \mathcal{S}$ denotes the corresponding 2-pin sub-net of i, $tree(i) \in \mathcal{T}$ denotes the corresponding routing tree of i, and $x_i \in \{0, 1\}$ is the binary indicator about whether i is selected, i.e., $x_i = 1$ if and only if i is selected after global routing. Let $j \in \mathcal{T}$ be a specific routing tree candidate, then $net(j) \in \mathcal{N}$ denotes the corresponding net of j, and $y_j \in \{0, 1\}$ is the binary indicator about whether j is selected, i.e., $y_j = 1$ if and only if j is selected after global routing.

Let $\mathcal{E}$ be the set of all g-cell edges, and $e \in \mathcal{E}$ be a specific g-cell edge. The capacity of e, denoted as cap_e, can be formulated as:

$$cap_e = \mathtt{track}_e - \beta_v \mathtt{pin\_density}_v - \mathtt{local\_net}_e \qquad (1)$$

Where:

- $\texttt{track}_e$ is the number of available tracks in e.
- $\texttt{pin\_density}_v$ is the number of pins in the g-cell v which is connected to e in the g-cell graph.
- β_v is a weight from CUGR2, based on the minimal edge length and defined in the LEF file and physical length of edges connecting v.
- $\texttt{local\_net}_v$ denotes the number of local nets at v, i.e., nets only occupying v.

In the capacity formula, the second part ($\beta_v \texttt{pin\_density}_v$) and the third part ($\texttt{local\_net}_e$) are used to estimate the influence of pin connections and local nets, which are essential for final detailed routing quality.

The demand of e, d_e, is given by:

$$d_e = \sum_{i \in \mathcal{P}_e} y_{tree(i)} x_i + \beta_v \left(\sum_{k \in \mathcal{P}_v} y_{tree(k)} x_k \right) \tag{2}$$

Where:

- $\mathcal{P}_e$ is the set of 2-pin path candidates passing through e.
- $\mathcal{P}_v$ is the set of 2-pin path candidates with a turning point at v.
- β_v is a weight following the same definition as above.

The demand calculation comprises two parts: the first part models the influence of the wires that go through the edge, and the second part represents the influence of vias.

Given the above definitions, the DAG forest-based 2D pattern routing problem can be mathematically formulated as follows:

$$\begin{aligned} \min_{i \in \mathcal{P}, j \in \mathcal{T}} \quad & a_1 \times \texttt{WL\_cost} + a_2 \times \texttt{via\_cost} \\ & + a_3 \times \texttt{overflow\_cost} && (3) \\ \text{s.t.} \quad & \texttt{WL\_cost} = \sum_{i \in \mathcal{P}} y_{tree(i)} x_i \texttt{WL}_i, && (4) \\ & \texttt{via\_cost} = \sqrt{L} \sum_{i \in \mathcal{P}} y_{tree(i)} x_i \texttt{TP}_i, && (5) \\ & \texttt{overflow\_cost} = \sum_{e \in \mathcal{E}} f(cap_e - d_e), && (6) \\ & \sum_{i: subnet(i) = s} x_i = 1, \forall s \in \mathcal{S}, && (7) \\ & \sum_{j: net(j) = n} y_j = 1, \forall n \in \mathcal{N}, && (8) \end{aligned}$$

where a_1, a_2, and a_3 are the weights for wire length cost, via cost, and overflow cost, respectively. $\texttt{WL}_i$ is the wire length of the 2-pin path candidate i, and $\texttt{TP}_i$ is the number of turning points of i. L is the number of routable layers. $f(*)$ represents some non-linear function (activation function) to be applied to the resource (capacity - demand) of g-cell edges, e.g., the ReLU function used in [14] and logistic function used in [2, 9]. Equation (7) ensures the selection of a single path for each 2-pin sub-net. Similarly, Equation (8) guarantees the selection of a single routing tree for each net.

4 Differentiable Global Router

4.1 DGR Workflow

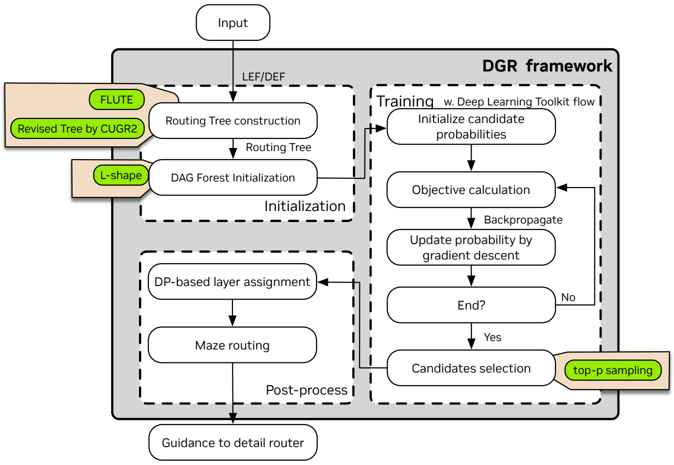

Input
DGR framework
LEF/DEF
FLUTE
Revised Tree by CUGR2
Routing Tree construction
Routing Tree
L-shape
DAG Forest Initialization
Initialization
Training
w. Deep Learning Toolkit flow
Initialize candidate probabilities
Objective calculation
Backpropagate
Update probability by gradient descent
End?
No
Yes
Candidates selection
top-p sampling
DP-based layer assignment
Maze routing
Post-process
Guidance to detail router

Figure 3: The workflow of DGR.

Figure 3 depicts the DGR workflow. Initially, we construct a DAG forest, encompassing routing tree candidates and 2-pin path candidates for each 2-pin sub-net within the trees. Every candidate is associated with a “selection" probability and the expectation of the costs defined in Equation (3) to Equation (6) are derived based on these probabilities. Subsequently, the selection probabilities are updated through back-propagation to minimize the costs. This iterative process continues until the iteration limit is reached. The 2D pattern routing solution is then obtained by selecting routing tree and 2-pin path candidates according to the optimized probabilities. This solution further undergoes dynamic programming-based layer assignment and maze routing-based refinement, as detailed in [2], to ultimately generate the final 3D global routing solution.

4.2 Routing DAG Forest Construction

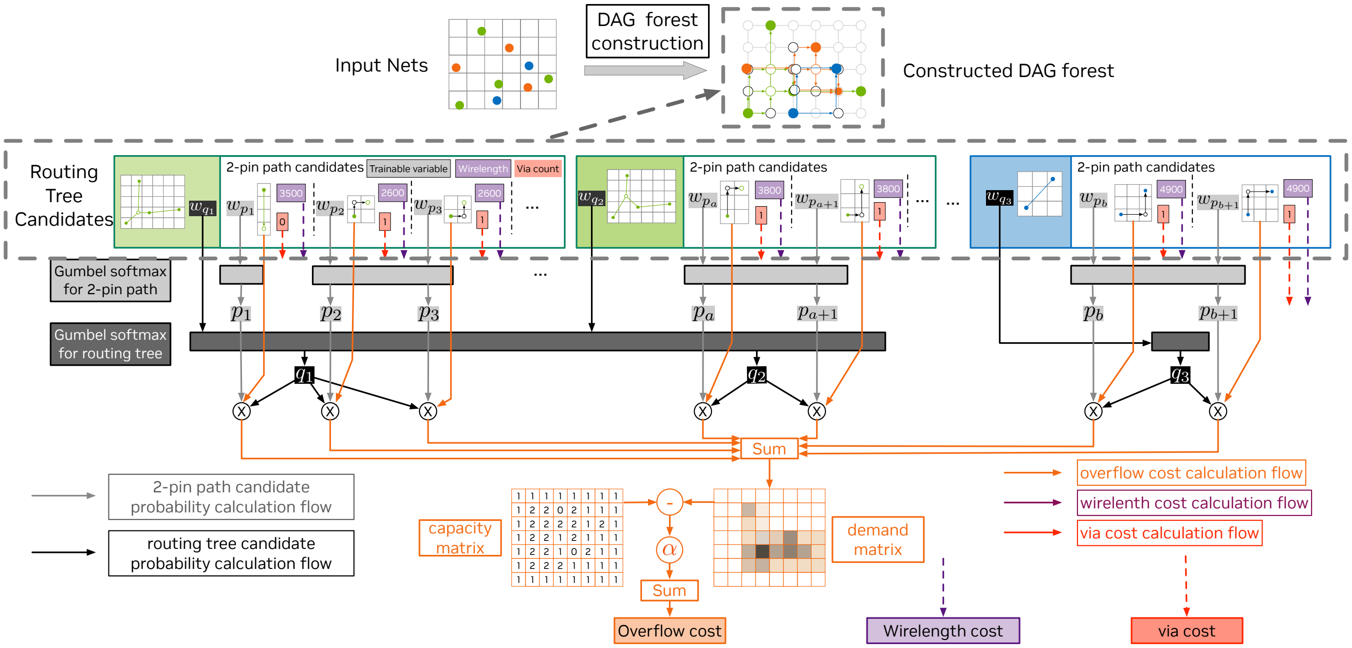
The routing DAG forest comprises routing tree candidates for each net, with each of these tree candidates spawning a collection of 2-pin path candidates. An example of the constructed DAG forest is provided in Figure 4. Initially, multiple routing tree candidates are formulated for each net using FLUTE and its fine-tuned version by CUGR2, which moves Steiner points based on congestion. It's worth noting that this is not restricted to just these two techniques; alternative routing tree generation algorithms, such as SALT [15] and TreeNet [16], can seamlessly integrate their resulting trees as additional candidates. Subsequently, every routing tree is used to segment the multi-pin net into 2-pin segments based on its tree topology. Then, all L-shape pattern paths are enumerated for each 2-pin sub-net and incorporated into the pool as 2-pin path candidates. In the final step, each candidate will be associated with a probability, which is initialized randomly.

4.3 Continuous Relaxation and Cost Calculation

We relax the categorical choice of candidates (x_i, y_j) to selection probabilities. Formally, we define $p_i \in [0, 1]$ as the probability of selecting 2-pin path candidate i, and $q_j \in [0, 1]$ as the probability of selecting the routing tree topology candidate j. Then, the expectation of the costs in Section 3.2 can be calculated as:

$$\texttt{overflow\_cost} = \sum_{e \in \mathcal{E}} f(cap_e - d_e) \tag{9}$$

$$\text{i.e.,} \quad d_e = \sum_{i \in \mathcal{P}_e} q_{tree(i)} p_i + \beta_v \left(\sum_{k \in \mathcal{P}_v} q_{tree(k)} p_k \right) \tag{10}$$



1	1	1	1	1	1	1	1
1	2	2	0	2	1	1	1
1	2	2	2	2	1	2	1
1	2	2	1	2	1	1	1
1	2	2	1	0	2	1	1
1	2	2	2	1	1	1	1
1	1	1	1	1	1	1	1

Figure 4: Workflow for cost calculation. For clarity and to avoid over-complication, certain elements such as the specific flows for via cost (in red) and wirelength cost (in purple) are not depicted.

$$\texttt{wirelength\_cost} = \sum_{i \in \mathcal{P}} q_{tree(i)} p_i \texttt{WL}_i \tag{11}$$

$$\texttt{via\_cost} = \sqrt{L} \sum_{i \in \mathcal{P}} q_{tree(i)} p_i \texttt{TP}_i \tag{12}$$

The total cost is a weighted sum of overflow cost, wirelength cost, and via cost. In our experiments, we adopt the metric weights from the ICCAD'19 contest: $\texttt{cost} = 500 \times \texttt{overflow\_cost} + 4 \times \texttt{via\_cost} + 0.5 \times \texttt{wirelength\_cost}$. Figure 4 illustrates the workflow to compute the cost.

4.4 Differentiable Optimization

We cannot, unfortunately, directly optimize the costs defined in Equation (9) to Equation (12) with respect to the probabilities due to the following constraints:

$$\sum_{subnet(i)=s} p_i = 1, \forall s \in \mathcal{S}; \forall p_i \in [0, 1]$$

$$\sum_{net(j)=n} q_j = 1, \forall n \in \mathcal{N}; \forall q_j \in [0, 1]$$

To transform the constrained optimization problem into an unconstrained one, we introduce an auxiliary layer to map unconstrained trainable variables ($w = \{w_i \in \mathbb{R}\}$) to probabilities, as depicted in Figure 4. A `softmax` layer seems to be an intuitive fit:

$$p_i = \frac{\exp(w_i)}{\sum_{k:tree(k)=tree(i)} \exp(w_k)} \quad q_j = \frac{\exp(w_j)}{\sum_{k:net(k)=net(j)} \exp(w_k)}$$

However, `softmax` deterministically samples a probability distribution. This deterministic nature can inadvertently lead to local optima, especially when the probabilities have a bad initialization. To circumvent this, we employ the `gumbel_softmax` function [17]:

$$p_i = \frac{\exp((w_i + g_i)/t)}{\sum_{k:tree(k)=tree(i)} \exp((w_k + g_k)/t)}$$

$$q_j = \frac{\exp((w_j + g_j)/t)}{\sum_{k:net(k)=net(j)} \exp((w_k + g_k)/t)}$$

This function, a stochastic variant of `softmax`, introduces Gumbel noise (g_i), i.e., the sample from the Gumbel distribution, to the logits prior to applying the softmax operation. Furthermore, to ensure that the final sampling of the routing tree candidate is discrete, we progressively reduce the temperature (t) of the `gumbel_softmax` throughout the iterations, called temperature annealing. It ensures that the final probabilities associated with routing tree candidates closely approximate either 0 or 1.

We have implemented our differentiable solver within the DGR framework using the deep learning toolkit *PyTorch*, as visualized in Figure 3. Leveraging *PyTorch*, we benefit from its robust and efficient support for matrix operations, automatic gradient derivation, and optimization. This toolkit seamlessly integrates GPU acceleration, further enhancing computational efficiency and enabling rapid experimentation in our research efforts.

4.5 Deriving Discrete Selection

When the maximum iteration count is reached, a 2D routing solution is derived by selecting candidates based on their associated probabilities.

Among the routing tree candidates, the one with the highest probability is selected. This probability tends to approach 1 as a result of our temperature annealing technique.

For 2-pin path candidates, we employ top-p sampling [18]. Initially, candidates are ranked by their probabilities. Subsequently, top candidates are selected until their cumulative probability surpasses a predefined threshold.

4.6 Post-Processing

Our 2D pattern routing solution will undergo dynamic programming-based layer assignment introduced in [2] to yield preliminary 3D routing results. Subsequently, maze routing is applied to nets in

Table 1: Comparison with ILP on synthetic data. DGR best/worst is the best/worst result among five runs with different random seeds, respectively. DGR* is the best result after the hyper-parameter search, i.e., additional 100 runs that randomly sample learning rate from 10^{-4} to 1 and temperature scaling factor in [0.8,0.85,0.9,0.95].

Synthetic data parameters				Runtime (s)		Overflow			
Grid Graph	cap_e	Net #	box size	ILP	DGR	ILP	DGR*	DGR best	DGR worst
20×20	1	20	4	0.08	5.09	30	30	30	30
50×50	1	50	10	36.12	5.16	173	173	173	173
50×50	1	100	10	102.46	4.95	1028	1028	1028	1028
50×50	2	100	10	38.79	4.89	0	0	0	0
50×50	1	1000	10	4621.12	5.59	35445	35445	35445	35445
50×50	10	1000	10	2493.21	78.85	7407	7407	7407	7407
50×50	10	10000	10	N/A	81.24	N/A	364536	364536	364621
100×100	2	1000	20	5763.16	5.83	48846	48846	48849	48891
100×100	2	10000	20	N/A	75.90	N/A	785664	785664	785678
1000×1000	1	100000	200	N/A	924.23	N/A	80237614	80238208	80249124
			Ratio	>118.28	1.00	1.00	1.00	$>1-10^{-6}$	$>1-10^{-5}$

ILP is implemented via CVXPY [20]. The green indicates that the result is as optimal as ILP method. N/A means ILP is running out of the time (> 8 hours).

congested areas to further minimize overflow. The final output is a comprehensive guide for detailed routing.

5 Experiments

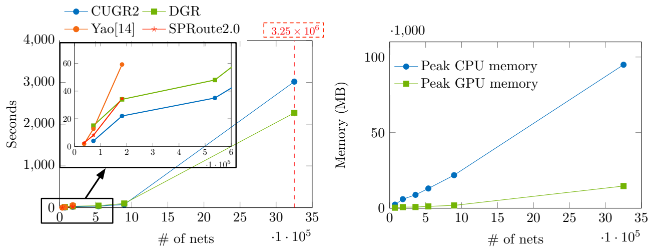
We implement DGR using PyTorch, a popular deep learning toolkit. Experiments are conducted on a 64-bit Linux workstation with Intel Xeon Silver 2.20 GHz CPUs and 256 GB memory. One NVIDIA GeForce RTX 2080Ti graphics card is used. We use the benchmarks from ISPD'18 and ISPD'19 contests [19]. The activation function f is sigmoid by default. We use Adam to optimize the weights w, with an initial learning rate 0.3. w is initialized randomly, and the random seed is fixed if not mentioned. The iteration number is 1000, and the initial temperature is 1. For every 100 iterations, we scale down the temperature by a factor of 0.9.

5.1 Comparison with ILP: Proof of Concept

To comprehend the performance gap between the results of our differentiable method and the optimal solutions, we compare DGR with an Integer Linear Programming (ILP)-based approach. Synthetic data is utilized for this experiment since the ISPD'18 and ISPD'19 benchmarks are too large for ILP. Statistics of synthetic testcases are shown in Table 1. ILP and DGR are applied to select L-shaped paths for every 2-pin net in order to minimize routing overflow. The overflow is calculated using ReLU, i.e., overflow $= \sum_{e \in \mathcal{E}} \mathrm{ReLU}(d_e - cap_e)$. The results are shown in Table 1. Remarkably, when equipped with the appropriate hyper-parameters (DGR*), DGR can deliver optimal solutions found by ILP. Even without hyper-parameter tuning, the performance gap between ILP and DGR remains relatively small.

5.2 Comparison with Leading Global Routers

We compare the global routing results of DGR with those of CUGR2 on the most congested ISPD'19 testcases that utilize only 5 routing layers. It is noteworthy that different global routers employ various overflow metrics, significantly influencing wirelength and via count. For fair comparison, we adopt the exact overflow metric as presented in CUGR2 [2] when evaluating the results of DGR against those of CUGR2. As illustrated in Table 2, compared with CUGR2, DGR shows a superior routing quality on all testcases : the number of G-cell edges with overflow after global routing is reduced by 23.9%, and the total wirelength and number of vias are also reduced by 0.95% and 1.28% on average.



(a) Runtime vs. # nets **(b) DGR memory vs. # nets**

Figure 5: Runtime and memory overhead vs. # nets.

[1] The runtime of Yao and SPRoute 2.0 are from [14] and [8], respectively.

[2] Yao[14], SPRoute 2.0, and CUGR2 run in a single-CPU-threaded mode, while DGR employs a single CPU thread and a NVIDIA 2080Ti GPU. For the largest case, the GPU used in DGR is NVIDIA A100.

[3] The DGR runtime excludes the time required to construct the DAG forest, a process that can be efficiently accelerated by net-level parallelism.

Moreover, we compare DGR with SPRoute2.0 [8] and Yao [14] on the same ISPD'18 benchmark set as presented in [14]. Both SPRoute2.0 and Yao [14] are leading sequential global routers, showcasing competitive results on the ISPD'18 benchmarks. As shown in Table 3, The average wirelength is reduced by 2.2% and 4.08% compared with the Yao [14] and SPRoute2.0, respectively. For the vias, even though DGR has more vias than Yao [14] and SPRoute2.0 when the benchmark is small (ispd18_test1 - ispd18_test4), the average number of vias is still reduced by 1.76% and 2.54% since DGR has less via number when the design becomes larger (ispd18_test5 - ispd18_test10).

5.3 Overflow cost function and Scalability Study

In global routing, how to model overflow cost is essential for result quality. Here, we represent f in Equation (9) using various functions, namely, ReLU, sigmoid, LeakyReLU, exp, and CELU, respectively. The results are shown in Figure 6. We can see that the selection of f influences the result, especially overflow, significantly, and sigmoid is the best choice, which outperforms CUGR2 (the red X mark) in most cases.

Moreover, we explore the runtime and memory scalability of DGR. As shown in Figure 5a, DGR has slightly more runtime overhead than CUGR2 when the number of nets is less than one million, when the design complexity continues increasing, DGR becomes more efficient than CUGR2. The reason is that DGR can generate better initial routing because of its concurrent global optimization nature, and better initial routing can avoid unnecessary rip-up and reroute. When the design becomes larger, the benefit can mitigate the runtime cost of DGR training. The memory result is given in Figure 5b, which shows that both CPU and GPU memory overhead is almost linear with the number of nets.

6 Conclusion

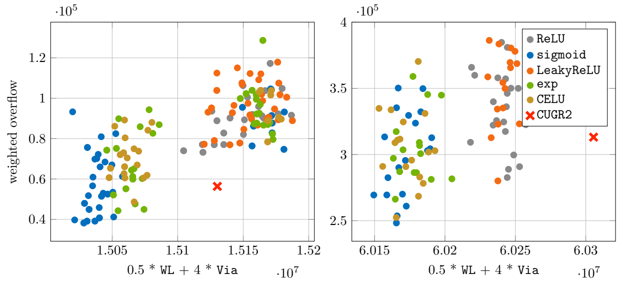
In this work, we propose a differentiable global router, which enables concurrent optimization hundreds of thousands of nets, and can be accelerated by deep learning toolkits on GPUs. Experimental results show that our method outperforms the state-of-the-art academic global routers. We are going to extend DGR to conduct concurrent 3D global routing.

Table 2: Performance comparison with CUGR2 [2] on the most congested testcases (with 5 routing layers) from ISPD'19 benchmarks, using the same overflow metric as [2].

	Benchmarks				# G-cell edges w/ overflow		Total Wirelength		# Vias	
	Cell #	Net #	G-cell Grid	Layer #	CUGR2	DGR	CUGR2	DGR	CUGR2	DGR
ispd18_5m	71954	72394	619 x 613	5	9	**7**	26748345	**26569365**	878760	**868049**
ispd18_8m	192003	179863	905 x 883	5	0	0	62171869	**61697029**	2182804	**2152936**
ispd18_10m	290386	182000	606 x 522	5	1	1	74439931	**74114344**	2311291	**2283099**
ispd19_7m	359746	358720	1053 x 1011	5	0	0	105264205	**104697033**	3830903	**3802312**
ispd19_8m	539611	537577	1202 x 1138	5	14	**5**	176164609	**173050359**	6276063	**6189202**
ispd19_9m	899341	895253	1337 x 1433	5	33	33	262680776	**260689064**	10438277	**10294296**
				Ratio	1.2391	**1.0000**	1.0095	**1.0000**	1.0128	**1.0000**

Table 3: Performance comparison with Yao [14] and SPRoute 2.0 [8] on ISPD'18 benchmarks using the exactly same testcases and overflow metric as presented in [14]. The results of Yao [14] and SPRoute 2.0 are from [14].

	Benchmarks				# G-cell edges w/ overflow			Total wirelength			# Vias		
	Cell #	Net #	G-cell Grid	Layer #	SPRoute 2.0	Yao [14]	DGR	SPRoute 2.0	Yao [14]	DGR	SPRoute 2.0	Yao [14]	DGR
ispd18_1	8879	3153	65 x 67	9	0	0	0	417181	415116	**409081**	30460	**29677**	31683
ispd18_2	35913	36834	216 x 201	9	0	0	0	7709980	7699590	**7603188**	338191	**324846**	343110
ispd18_3	35977	36700	329 x 247	9	0	0	0	8644190	8626810	**8531769**	336061	**326618**	347547
ispd18_4	72094	72410	592 x 403	9	0	0	0	26246000	26121900	**25635347**	680322	**677125**	757225
ispd18_5	71954	72394	619 x 613	9	0	0	0	27307600	27183700	**26648030**	803900	802244	**791171**
ispd18_6	107919	107701	571 x 354	9	0	0	0	35132200	34980000	**34249130**	1227804	1219063	**1182956**
ispd18_7	179881	179863	905 x 883	9	0	0	0	64964200	64224500	**62971368**	1979427	1969051	**1950290**
ispd18_8	192003	179863	905 x 883	9	0	0	0	64964200	64224500	**63165843**	2109296	2082828	**1954286**
ispd18_9	192911	178858	606 x 522	9	0	0	0	53981500	53674000	**52504359**	1995119	1987919	**1937057**
ispd18_10	290386	182000	606 x 522	9	0	0	0	71709900	67400100	**65204874**	2182178	2175317	**2098285**
				Ratio	1.0000	1.0000	1.0000	1.0408	1.0220	**1.0000**	1.0254	1.0176	**1.0000**



(a) ispd18_5m

(b) ispd19_7m

Figure 6: The impact of activation function on the global routing results. Each dot is a single run with different activation functions and different hyper-parametersss. The y-axis (weighted overflow) is calculated as $10 \times n_1 + 1000 \times n_2 + 10000 \times$ peak overflow, where n_1 is the number of nets with overflow after layer assignment, n_2 is the number of G-cell edges with overflow after global routing, peak overflow is the maximum overflow among all G-cell edges.

Acknowledgments

We would like to express our sincere gratitude to Prof. Hu Jiang from TAMU for his guidance and assistance throughout this work.

References

[1] M. Pan and C. Chu, "Fastroute: A step to integrate global routing into placement," in *Proceedings of the 2006 ICCAD*, 2006, pp. 464–471.

[2] J. Liu and E. F. Young, "EDGE: Efficient DAG-based Global Routing Engine," in *2023 60th DAC*, 2023.

[3] C. Chu and Y.-C. Wong, "FLUTE: Fast lookup table based rectilinear steiner minimal tree algorithm for VLSI design," vol. 27, no. 1, pp. 70–83, 2008.

[4] C. Albrecht, "Global routing by new approximation algorithms for multicommodity flow," *TCAD*, vol. 20, no. 5, pp. 622–632, 2001.

[5] T.-H. Wu, A. Davoodi, and J. T. Linderoth, "GRIP: Scalable 3D global routing using integer programming," in *Proceedings of DAC*, 2009, pp. 320–325.

[6] M. Cho, K. Lu, K. Yuan, and D. Z. Pan, "BoxRouter 2.0: Architecture and implementation of a hybrid and robust global router," in *2007 ICCAD*, 2007, pp. 503–508.

[7] J. Hu, J. A. Roy, and I. L. Markov, "Sidewinder: a scalable ILP-based router," in *Proceedings of SLIP*, 2008, pp. 73–80.

[8] J. He, U. Agarwal, Y. Yang, R. Manohar, and K. Pingali, "SPRoute 2.0: A detailed-routability-driven deterministic parallel global router with soft capacity," in *2022 27th ASP-DAC*, 2022, pp. 586–591.

[9] J. Liu, C.-W. Pui, F. Wang, and E. F. Young, "Cugr: Detailed-routability-driven 3d global routing with probabilistic resource model," in *2020 57th DAC*, 2020, pp. 1–6.

[10] Z. Guo, F. Gu, and Y. Lin, "GPU-Accelerated Rectilinear Steiner Tree Generation," in *Proceedings of the 41st ICCAD*, 2022, pp. 1–9.

[11] S. Liu, Y. Pu, P. Liao, H. Wu, R. Zhang, Z. Chen, W. Lv, Y. Lin, and B. Yu, "Fastgr: Global routing on cpu-gpu with heterogeneous task graph scheduler," *TCAD*, 2022.

[12] S. Lin and M. D. Wong, "Superfast Full-Scale CPU-Accelerated Global Routing," in *Proceedings of the 41st ICCAD*, 2022, pp. 1–8.

[13] M. Khasawneh and P. H. Madden, "Hydraroute: A novel approach to circuit routing," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI (GLSVLSI)*, 2019, pp. 177–182.

[14] Y. Pengju, Z. Ping, and Z. Wenxing, "Pathfinding Model and Lagrangian-Based Global Routing," in *2023 60th DAC*, 2023.

[15] G. Chen and E. F. Young, "Salt: provably good routing topology by a novel steiner shallow-light tree algorithm," *TCAD*, vol. 39, no. 6, pp. 1217–1230, 2019.

[16] W. Li, Y. Qu, G. Chen, Y. Ma, and B. Yu, "TreeNet: Deep point cloud embedding for routing tree construction," in *Proceedings of the 26th ASP-DAC*, 2021, pp. 164–169.

[17] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[18] A. Fan, M. Lewis, and Y. Dauphin, "Hierarchical neural story generation," *arXiv preprint arXiv:1805.04833*, 2018.

[19] S. Dolgov, A. Volkov, L. Wang, and B. Xu, "2019 cad contest: Lef/def based global routing," in *2019 ICCAD*, 2019, pp. 1–4.

[20] S. Diamond and S. Boyd, "Cvxpy: A python-embedded modeling language for convex optimization," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 2909–2913, 2016.