Multiterminal Pathfinding in Practical VLSI Systems with Deep Neural Networks

DMITRY UTYAMISHEV and INNA PARTIN-VAISBAND, University of Illinois at Chicago, USA

A multiterminal obstacle-avoiding pathfinding approach is proposed. The approach is inspired by deep image learning. The key idea is based on training a conditional generative adversarial network (cGAN) to interpret a pathfinding task as a graphical bitmap and consequently map a pathfinding task onto a pathfinding solution represented by another bitmap. To enable the proposed cGAN pathfinding, a methodology for generating synthetic dataset is also proposed. The cGAN model is implemented in Python/Keras, trained on synthetically generated data, evaluated on practical VLSI benchmarks, and compared with state-of-the-art. Due to effective parallelization on GPU hardware, the proposed approach yields a state-of-the-art-like wirelength and a better runtime and throughput for moderately complex pathfinding tasks. However, the runtime and throughput with the proposed approach remain constant with an increasing task complexity, promising orders of magnitude improvement over state-of-the-art in complex pathfinding tasks. The cGAN pathfinder can be exploited in numerous high throughput applications, such as, navigation, tracking, and routing in complex VLSI systems. The last is of particular interest to this work.

CCS Concepts: • **Hardware → Physical design (EDA)**; • **Computing methodologies → Neural networks;**

Additional Key Words and Phrases: Computer-aided design, CAD, electronic design automation, EDA, pathfinding, global routing, machine learning, deep learning, deep neural networks, DeconvNets, convolutional neural networks, ConvNets, generative adversarial networks, GANs, variational autoencoders, VAEs

ACM Reference format:
Dmitry Utyamishev and Inna Partin-Vaisband. 2023. Multiterminal Pathfinding in Practical VLSI Systems with Deep Neural Networks. *ACM Trans. Des. Autom. Electron. Syst.* 28, 4, Article 51 (May 2023), 19 pages.
https://doi.org/10.1145/3564930

1 INTRODUCTION

Multiterminal pathfinding in presence of obstacles, or **obstacle-avoiding rectilinear Steiner minimum tree construction (OARSMT)** is a task of finding an optimal (e.g., shortest) path between two or more placed terminals. While being computationally hard, this is one of the fundamental navigation problems, required in a wide range of applications, from mobile robot path planning, navigation, wayfinding, and tracking to routing wires in **printed circuit boards (PCBs)** and **integrated circuits (ICs)**. In particular, multiterminal pathfinding in presence of obstacles plays an important role in **electronic design automation (EDA)** algorithms, such as global routing and placement. Furthermore, recent trends in **machine learning (ML)**-based EDA require an

Authors' address: D. Utyamishev and I. Partin-Vaisband, University of Illinois at Chicago, 851 S Morgan St, Chicago, IL, 60607; emails: {dutyam2, vaisband}@uic.edu.

1084-4309/2023/05-ART51 $15.00
https://doi.org/10.1145/3564930

ML-inspired revision of the underlying EDA algorithms. For example, the reinforcement learning solution used in state-of-the-art Google placement methodology [20] relies on solving instances of pathfinding problems to train the reward generation model for each design layout [5]. Thus, efficient pathfinding is critical for today's and modern EDA tools.

Multiterminal pathfinding is an NP-hard problem [11]. Thus, due to the excessively large search space, modern multiterminal pathfindng tasks cannot be optimally solved in reasonable time. To mitigate the computational complexity of pathfinding, existing solvers use approximation heuristics, producing sub-optimal solutions in reasonable time. The traditional deterministic pathfinding approaches typically exhibit the following design flow: A multiterminal pathfinding task is decomposed into multiple two-terminal shortest path tasks and these simpler tasks are solved individually.

Modern pathfinding and routing approaches vary primarily by algorithms for pathfinding decomposition and two-terminal shortest path search as well as by algorithm convergence criteria. The primary limitation of these approaches is the unpredictability of pathfinding convergence and performance. Several important ML approaches have recently been proposed to alleviate the unpredictability issue. Guided by the ML insight, these approaches, however, still rely on poorly parallelizable computational methods, such as the shortest path search. While **neural networks (NNs)** typically utilize highly parallelizable topologies (i.e., a single instruction can be simultaneously performed with multiple data points on multiple processing units), existing pathfinding algorithms are serial in their nature and their runtime cannot be efficiently shortened through parallelization (i.e., instructions are highly inter-dependent and should be executed in order). With the poorly scalable traditional methods and increasing complexity of modern pathfinding applications (e.g., number of terminals and obstacles in modern ICs), runtime and throughput have become a major concern.

While a continuous incremental improvement in OARSMT runtime has been demonstrated over the past decade with state-of-the-art approaches, the industry is still in search after a fundamentally more effective OARSMT solutions [25]. Such a paradigm shift can be achieved by moving away from executing OARSMT algorithms on inherently sequential CPUs with $O(n \cdot \log(n))$ runtime complexity in the expected case and $O(n^2)$ in the worst case (as it is done today) toward parallelizing OARSMT execution on modern GPU platforms. While such a move can ultimately result in $O(1)$ OARSMT runtime complexity, it requires fundamentally different, parallelizable OARSMT approaches.

In this work, an end-to-end ML pathfinding is proposed. With this approach, ML models are trained on routed multiterminal training samples and exploited to solve unseen test cases. Specifically, the traditional multiterminal obstacle-avoiding pathfinding task is mapped onto a modern image manipulation task and solved with a generative NN. It is shown that a properly designed deep NN, trained on robust reference data, can efficiently learn and detect routing patterns in inference and determine and execute preferred pathfinding heuristics. As a result, all unseen pathfinding test cases are routed with state-of-the-art wirelength, and those complex test cases are routed within a fraction of runtime as compared with existing pathfinders. The significant increase in performance is possible due to the following factors:

(1) *Reducing the multiterminal pathfinding task to the image-to-image manipulation task:* Pathfinding inputs and outputs are mapped onto 2D bitmaps.
(2) *Accumulating pathfinding information over various tasks, continuously increasing pathfinding performance in new tasks with similar obstacle configuration:* This approach is in particular effective in typical IC routing use cases, in which thousands of nets are routed within the same placement configuration of standard cells, or autonomous vehicle driving navigation

problems, where thousands of autonomous vehicles drive on different routes in the same city or region.

(3) *Efficiently generating a synthetic robust dataset of solved pathfinding task instances:* The proposed methodology for generating pathfinding data enables training of a truly deep NN, overcoming a major concern of limited existing multiterminal shortest path data.

(4) *Parallelizing the pathfinding process for systematic execution on parallel processing hardware accelerators:* While pathfinding parallelization is limited with traditional approaches, the parallel nature of ML branchless computation allows to seamlessly and efficiently process pathfinding on GPU, TPU, NPU, or other parallel processing hardware in a simultaneous manner and with no overhead.

As a result, the proposed method opens new directions for parallelization of multiterminal pathfinding and scaling the capabilities of its applications, such as of the computer-aided IC global routing. The rest of the article is organized as follows: The background on hardware acceleration-based parallelization and ML approaches for image manipulation is provided in Section 2. The proposed ML system and design considerations are described in Section 3. Evaluation methods and experimental results are presented in Section 4. The article is concluded in Section 5.

2 BACKGROUND

In this research, multiterminal obstacle-avoiding task is considered as a rectangular array of labeled tiles. In an input array, each tile is marked as an obstacle, terminal, or empty. During the pathfinding process, some of the empty tiles are marked as path tiles. The primary objective of a pathfinder is determining the tile-to-tile path, connecting all (two or more) terminal tiles. Two terminal tiles are considered connected if there is a set of adjacent (via a shared edge) non-obstacle tiles within the path that includes the two terminal tiles.

The expected output set of path tiles comprises a minimum number of path tiles (i.e., minimum length) while connecting all terminals within the array of tiles and does not intersect with the obstacle tiles set. This task is NP-hard but can be approximately decomposed and solved in polynomial time with reasonable path length overhead. Existing multiterminal pathfinding solutions are described in Section 2.1. Machine learning methods exploited in this article are explained in Section 2.2.

2.1 Traditional Multiterminal Pathfinding

Traditional multiterminal pathfinding approaches are based on **minimum rectilinear Steiner tree (MRST)** approximation. With MRST, a multiterminal pathfinding task can be split into multiple terminal-to-terminal pathfinding tasks, using additional auxiliary nodes (i.e., Steiner split-point nodes), and a path between two tiles can be determined with two-point pathfinding algorithm in polynomial time.

Decomposition of a multiterminal pathfinding task into multiple terminal-to-terminal pathfinding tasks is, however, also NP-hard. Thus, approximating methods such as **minimal spanning tree (MST)** are often utilized instead of MRST, yielding suboptimal, yet computationally preferred solutions with a typical computational complexity of $O(n \cdot \log(n)^2)$), where n is the overall number of tiles [4, 9]. A primary advantage of the MST method is that the total length of the generated path is within certain bounds of the optimal length. Another method for mitigating the MRST complexity is lookup tables [1, 6]. These methods typically exhibit polynomial time complexity, trading off the optimal length for a shorter execution runtime.

Once the original multiterminal pathfinding task is successfully split into multiple two-terminal sub-tasks, best-first search algorithms are commonly utilized for determining paths between the

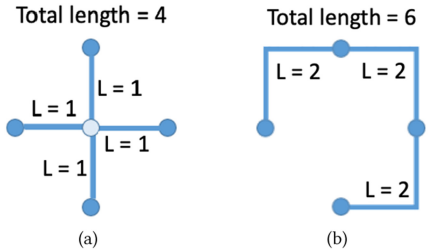


Fig. 1. Pathfinding of a four-terminal input with (a) the optimal terminal-to-split path (as determined by the MRST approach), resulting in the shortest total path length, and (b) terminal-to-terminal path (as determined by the MST approximation), yielding suboptimal (i.e., longer than in (a)) total path length.

original terminals and split-point nodes, as well as paths between different split-points of the Steiner tree. Methods such as pattern routing, negotiated-congestion routing, and **integer linear programming (ILP)** are utilized in modern global routers to speed those easy-to-route nets up and resolve difficult-to-route regions. An optimal MRST and sub-optimal MST routing is illustrated in Figure 1.

Albeit the progress in IC routing, at the core of the existing pathfinding solutions remains the reliance on graph traversing and similar algorithms that yield non-regular and, thus, poorly parallelizable computational methods. While neural networks typically utilize highly parallelizable topologies (i.e., a single instruction can simultaneously be performed with multiple data points on multiple processing units), existing routers are serial in their nature and their runtime cannot be efficiently shortened through parallelization (i.e., instructions are highly interdependent and should be executed in order). Such pathfinding with poorly scalable traditional methods becomes even more challenging with the increasing number of terminals and obstacles in modern VLSI systems.

2.2 Existing ML-based Multiterminal Pathfinding

While ML-driven pathfinding is of particular interest, most of the existing pathfinding approaches utilize ML to predict convergence, wirelength, and other path characteristics with a certain initialization state [2, 3, 13, 17, 18, 22, 24, 28–30]. Alternatively, several important attempts to route paths with an NN exhibit limited scalability and training restrictions. For example, in our preliminary work [26], a generative deep learning model utilized for IC pathfinding in our preliminary work exhibits limited maximum resolution (up to 64×64 pathfinding tiles) and cannot be used in practical applications or replace traditional routers. Performance of the **variational autoencoder (VAE)** architecture used in Reference [26] is known to significantly decrease with an increasing input resolution. Furthermore, the performance of this supervised method is a strong function of the robustness and quality of the training set, yielding another primary concern. To produce a robust prediction model, supervised ML methods require significant amount of training data (i.e., pathfinding task-solution pairs). While actual IC physical design data is proprietary and not available in required amount, the straight-forward method of generating random inputs and solving them with traditional pathfinding algorithms is not feasible due to computational complexity of pathfinding. The resolution and training set limitations are typical for existing ML routing approaches, based on routing problem representation as an image translation problem.

Another VAE-based model [31] also exploits the image representation of a two-dimensional IC to route paths in analog ICs. In this work, the labeled data for the supervised training comprises a limited set of existing routed analog ICs, as hand-solved by the human experts. To obtain

reasonable performance with such a limited training set, ICs have been aggressively downsampled down to 64×64 tiles (similar to Reference [26]). While the model performs well in analog domain, the limitations of the training set and ML architecture make these approaches impractical for pathfinding in modern digital ICs and other applications. The variational autoencoder architecture in Reference [28] is known for its training limitations for high-resolution bitmaps. While it is reasonable to use 64×64 tiles for global routing in analog circuits, global routing in digital circuits usually requires a much higher number of tiles.

Additional recent works explore ML pathfinding solutions within a simplified design space (e.g., small input space, no obstacles, two-pin instances) [10, 15]. The importance of a large robust training set to enable deep learning is emphasized in Reference [15], and a methodology for generating a set of multiterminal obstacle-avoiding pathfinding tasks is proposed in the same paper. However, how to route numerous generated tasks with a state-of-the-art like path length is still an open question. Therefore, a novel method to generate synthetic training data for supervised ML pathfinding is required. To address this question, a methodology is proposed in this article for efficiently generating a large amount of practical training data.

A breakthrough in ML physical design has recently been achieved by researchers from Google who developed a deep learning approach for IC floorplanning [20]. In this work, the whole IC gate graph is embedded into a low-resolution latent space and upsampled with a stack of convolutional layers to a two-dimensional image representation of the placed IC. Graph-embedding models are used to represent a preferred placement graph [14]. Based on the reported results, months of traditional floorplanning are reduced to a few hours with Reference [20]. While the floorplanning solution cannot be seamlessly adapted for global routing and multiterminal pathfinding, this work is an important step toward a physical design end-to-end learning in modern ICs. The goal of this article is to initiate a similar paradigm shift in multiterminal pathfinding.

2.3 Pathfinding as Bitmap Translation

Modern ML image processing solutions are reduced to convolution operations (e.g., within a convolutional kernel or convolutional layers of deep neural networks) that are decomposed into a large number of small independent matrix multiplications. Hardware accelerators can, therefore, be efficiently utilized in this type of computation with large number of cores and parallel access to local and shared memory. Thus, mapping an array of IC tiles onto a 2D image transforms an inherently sequential task to a naturally parallelizable one, enabling efficient utilization of hardware acceleration platforms. Identifying an appropriate class of imaging problems and effectively representing pathfinding tasks within that class is, therefore, a primary objective. Yet another objective is to design a large training dataset of robust pathfinding tasks (2D arrays with varying number and location of terminals and obstacles) and corresponding solutions (arrays with state-of-the-art-like paths), as required for typical ML imaging training.

Imaging translation is a class of problems that focuses on learning the per-pixel mapping from an input bitmap to an output bitmap, hence translating one possible data representation into another. This approach is useful in various domains and applications, such as style transfer, inpainting, and object transfiguration and typically exploited for transforming and repairing photos. Similarly, we propose to reconsider a pathfinding task as a problem of image reconstruction in which the path tiles are the missing image parts that are reconstructed using image translation. Existing image translation solutions are typically based on generative NNs and thus, highly parallelizable. A generative NN is designed in this article to demonstrate pathfinding with image translation, yielding a fundamentally novel, highly scalable, and parallelizable solution for the multiterminal pathfinding task.

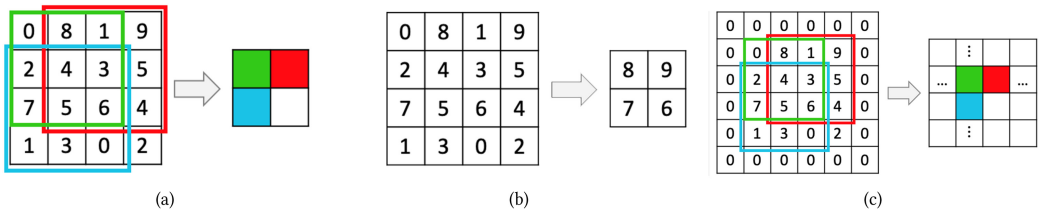


Fig. 2. Illustration of ConvNet hyperparameters on a 4×4 bitmap with a 3×3 kernel, (a) a convolutional 3×3 kernel with stride of one reduces the bitmap dimension from 4×4 to 2×2, (b) dimensionality is reduced with max-pooling, and (c) bitmap dimension is maintained from (a) to (b) with intermediate zero padding.

2.4 Generative Machine Learning

ML is a set of computational techniques that can be exploited for searching complex patterns in large volume of data and predicting the output based on provided input and ML parameters. Supervised learning is one type of ML paradigm that utilizes training data for determining ML model parameters. Each training sample comprises an input and the corresponding true label. During the training, ML model is iteratively updated to minimize the error between its output and the true label.

VAEs have been demonstrated as a powerful solution for image-to-image processing problems, such as image colorization, stylization, or inpainting. A VAE is a **deep NN (DNN)** that combines a recognition and a generative models. The recognition model (commonly designed as a **convolutional neural network (ConvNet)**) encodes DNN input into a vector of latent state probabilistic distributions of learned attributes, while the generative model (commonly designed as a **deconvolutional neural network (DeconvNet)**) decodes the randomly sampled latent state distributions into the DNN output.

Convolutional neural networks have been proven as a preferred ML architecture for efficient detection of complex local patterns in 2D maps. A ConvNet is a stack of convolutional and pooling layers. Each convolutional layer is defined by a convolutional kernel that slides over the inputs of the layer, generating a local map based on the local layer features. Two important hyperparameters of a convolutional layer are stride and padding. While stride controls the sliding of the kernel over the input volume, padding maintains the dimensionality of the data. The objective of a pooling layer is to reduce the dimensionality of data, abstracting the information about complex features as this information propagates forward through a ConvNet. The inner (i.e., with the lowest dimension) latent space represents attributes of a given input as a probability distribution. When decoding from the latent space, latent attributes are sampled from corresponding distributions to generate a vector, which is further processed with deconvolutional layers. The concepts of kernel size, stride, padding, and pooling are exemplified based on a 4×4 bitmap with a 3×3 kernel, stride of one, and pooling that prioritizes maximum values, as shown in Figure 2.

Deconvolutional neural networks are commonly used to decode a low-dimensional data space into a dimensionally higher space. The DeconvNet topology is similar to ConvNet, except for the upsampling DeconvNet layers that replace the pooling ConvNet layers.

Conditional generative adversarial networks (cGANs) are an advanced ML training approach. With this approach, generator (VAE is commonly used as a cGAN generator) and discriminator submodels are utilized and conditioned by a certain input (e.g., a generated path is conditioned by certain placed terminals and obstacles). A discriminator convolutional model is trained to classify an output bitmap as a true label or ML-generated bitmap. Simultaneously, the generator is trained to produce output bitmaps that cannot be recognized by the discriminator as ML-generated. As a result, the error between the generated and expected (i.e., true label) output

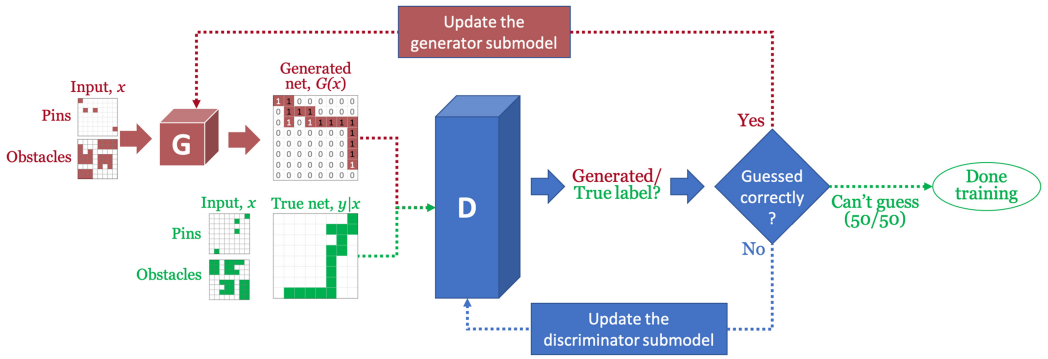


Fig. 3. Training a cGAN to generate robust multiterminal paths.

bitmaps is reduced over successive training iterations. The adversarial nature of the architecture allows the generator submodel to simultaneously learn the mapping between (1) the input (e.g., pathfinding array with marked obstacles and terminals) and generated output (e.g., the corresponding routed path) bitmaps, and (2) the true label (e.g., a path from the training dataset) and generated output. Note that a cGAN can generate routed paths based on previously unseen inputs and thus can be trained to route unseen ICs. The training process of cGAN ML model is illustrated in Figure 3.

Performance of ML system is a strong function of a training set and training time. Model convergence time increases with the increasing number of training data samples. Alternatively, as the number of training samples is reduced, or the diversity of training data becomes limited, the risk of model overfitting is increased, yielding high performance with a training set but low performance with unseen input data. Typical imaging training sets comprise up to a few thousand data samples. In the next chapters, the formulation of a multiterminal pathfinding as an imaging task is proposed and ML system design considerations are described. Design solutions that facilitate generation of a robust training set and convergence of the model in reasonable time without overfitting are also proposed.

3 PROPOSED PATHFINDING SYSTEM

The proposed workflow of ML-based pathfinder comprises three key phases. A training set of 2D bitmaps (i.e., routed and unrouted bitmap pairs) is generated during the first phase (see Section 3.1). During the second phase, a cGAN model is trained on the training set with physics-aware loss function (see Section 3.2). While the generation of training set and the training are time-consuming tasks (e.g., can take hundreds of hours on NVIDIA GTX1080 platform), these tasks are not necessarily performed from scratch. Existing training sets from other ML pathfinding and routing systems can be reused and enhanced. Finally, transfer learning and learning with partial layout information (e.g., information about certain standard cells, as shown in Section 4) can be utilized to fine-tune pre-trained models, enhancing the overall pathfinding performance. The process of training set generation and training itself is, therefore, expected to improve over generations of VLSI systems.

During the third (inference) phase, pathfinding input data is parsed and mapped into a pathfinding solution with a properly trained generative ML model. A typical concern with the proposed approach is the connectivity of a generated path. While in imaging problems, a missing or incorrect pixel has little effect on image perception, a routing path with a missing pixel exhibits an open circuit and is thus invalid. To maintain path connectivity, those paths that are generated with disconnected clusters (based on experimental results, less than 10) are postprocessed.

Each of the framework phases is explained in detail in the following subsections. The methodology to efficiently generate a robust training set is described in Section 3.1. The architectures of the cGAN pathfinder and the proposed physics aware loss function are explained in Section 3.2. The postprocessing algorithm for merging clustered cGAN paths is presented in Section 3.3.

3.1 Training Set Generation

To approach a pathfinding challenge as a supervised ML task, the ML model needs to be trained on a set of pathfinding reference samples (i.e., bitmaps with terminals, obstacles, and corresponding paths). A deep learning model requires a significant amount of training samples. Effective and fast generation of pathfinding tasks and the corresponding, state-of-the-art-like output path solutions is, therefore, a primary concern. While a straightforward generation of random pathfinding tasks is feasible, generating tens of thousands of complex path samples required to train a cGAN model with one specific resolution has not been practical with existing tools. The pathfinding reference samples are synthetically generated in this article.

The key idea is merging several low-resolution, reference samples into more complex samples. Note that pathfinding optimality in the high-resolution training samples is not required for efficient ML training. Alternatively, the training samples should capture a broad variety of pathfinding patterns. First, a large set of pathfinding tasks is generated within small (i.e., 8×8 to 128×128) rectangular bitmaps. Each small bitmap comprises at least one terminal or a path segment placed on the bitmap perimeter (i.e., edge terminal or edge path segment). These small bitmaps are optimally solved with exhaustive pathfinding methods. A valid merge of solved paths into a longer, more complex path within a larger bitmap is accomplished by joining two bitmaps via two edge segments. For that purpose, smaller bitmaps can be rotated and flipped as needed. The merging process continues until the resulted bitmap reaches the target sample size. Note that the training set generation algorithm guarantees the optimal path width (one tile) during the tile merging operation and thus, the cGAN pathfinder is trained not to generate fat lines. Examples of valid and non-valid merges are shown in Figure 4.

The flow diagram of the training set generation is shown in Figure 5. At each iteration, a bitmap is randomly selected from a pool of bitmaps and matched with another random bitmap from the pool for a valid merging. Note that bitmaps generated in this manner tend to exhibit statistically significant difference in path density at the edge tiles, increasing the risk of model overfitting during training. To mitigate overfitting, each path resulted from a valid merging is shifted in a random direction, as shown in Figure 4. Input data comprises a $2 \times N \times M$ array, where $N, M \leq 1{,}024$. The bounding box of each input sample is randomly shifted within the $1{,}024 \times 1{,}024$ processing space and the remaining space is marked as an obstacle. The grouped and shifted bitmap is added to the pool and the process continues to the next iteration. A model trained on the resulted training set is expected to capture broad pathfinding rules in presence of obstacles. To capture system-specific obstacle constraints, another fine-tuned dataset is generated. Samples in this set are generated in the following manner: Tiles from a typical layout are randomly sampled and combined into a small bitmap (e.g., 128×128). The bitmap is utilized to generate thousands of pathfinding tasks with numerous randomly placed terminals, which are routed with conventional methods. Based on experimental results, including the fine-tuned data within the training set significantly increases the saturation speed of the trained model.

The proposed high-resolution map generation method is greedy and yields suboptimal wirelength. Please note that this approach is only used for generating training pathfinding samples. Based on our observation, such suboptimality in training data does not have a critical effect on the actual wirelength of the paths generated in inference. The primary goal for the cGAN pathfinder model is to learn the route patterns (e.g., directional L/Z/U-shapes [12]) rather than optimum

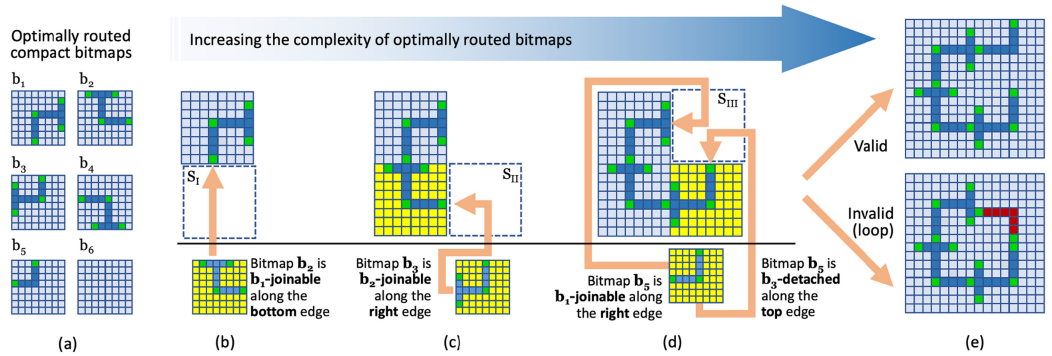


Fig. 4. Illustration of the proposed flow for generating a training sample of a robust routed path. (a) A set of low-resolution optimally routed bitmaps. (b) Bitmap b1 is randomly selected and placed. Given the single bitmap, b1, adjacent to the unprocessed space SI, the bitmap, b2, is randomly selected from the list of b1-joinable bitmaps and placed in SI along the bottom edge of b1. (c) Given the single bitmap, b2, adjacent to the unprocessed space SII, the bitmap, b3, is randomly selected from the list of b2-joinable bitmaps and placed in SII along the right edge of b2. (d) Given the two bitmaps, b1 and b3, adjacent to the unprocessed space SIII, the bitmap, b5, is randomly selected from the intersection of the b1-joinable and b3-detached bitmaps and placed in SIII along the right edge of b1 and top edge of b3. (e) As a result, a robust routed valid bitmap with 4x resolution is generated (top). Alternatively, selecting the last bitmap from the intersection of the b1- and b3-joinable bitmaps results in an invalid circular path (bottom). Thus, bitmaps are always selected from the intersection of a joinable and detached lists, as shown in the flowchart in Figure 5.

routes. The proposed method yields high-quality training data, as has been demonstrated based on the cGAN ability to converge to an effective pathfinding model.

3.2 Neural Network Architecture

The typical conditional adversarial loss is defined as

$$L_{cGAN} = \mathbb{E}_{x,y}[\log D(y|x)] + \mathbb{E}_{x,z}[\log(1 - D(G(x,z)|x))], \tag{1}$$

where x is the input bitmap, y is the expected (routed) output bitmap (i.e., the true label), z is the random noise, generator $G : x, z \rightarrow \hat{y}$ aims at minimizing the loss, and the adversarial discriminator $D : x, y \rightarrow \{$"true,""generated"$\}$ aims to maximize the loss.

While in traditional cGANs, random noise is utilized to generate different stochastic outputs, in a typical pathfinder, the preferred output is not random but determined based on physical IC characteristics. In this article, the cGAN is designed without the random noise but enhanced with physics-aware path generation reconstruction loss function, L_r. The trained generator G^* is, therefore, determined by

$$G^* = \arg \min_G \max_D L_{cGAN}(G, D) + \lambda L_r(G). \tag{2}$$

To understand how the $L_r(G)$ is determined, consider the following definitions for formulating the pathfinding task as a supervised ML task: Let X and Y be the sets of, respectively, unrouted bitmaps with placed terminals and obstacles and corresponding single-path routed bitmaps. A pathfinding task is to find the preferred pathfinding path of tiles, $y_x \in Y$, connecting a certain number of placed terminals under certain obstacle constraints, as defined by $x \in X$. For an unrouted $n \times n$ bitmap $x \in X$, the corresponding single-path routed bitmap $y_x \in Y$ is an $n \times n$ bitmap of tiles $(i, j), 1 \leq i, j \leq n$, where each tile is associated with a binary score, $y(i, j) = 0$ or $y(i, j) = 1$ if the tile (i, j) is, respectively, excluded from or included within the preferred output path. These definitions are illustrated in Figure 6.

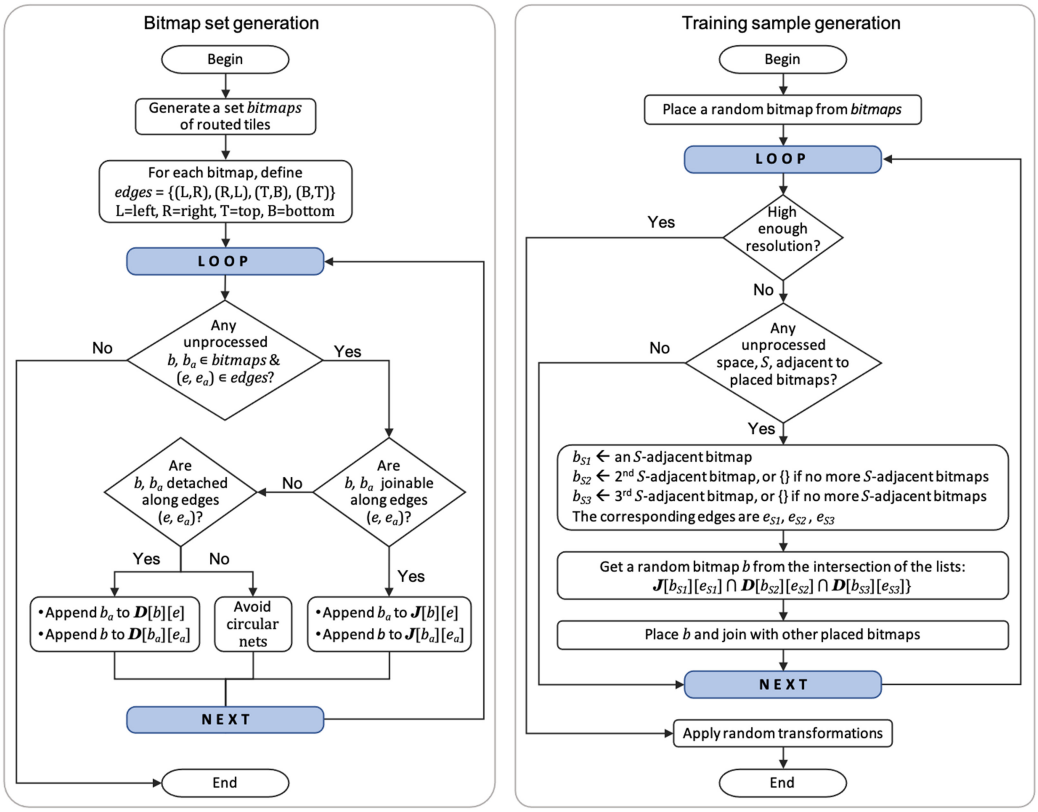


Fig. 5. Flow diagram of the training set generation algorithm. On the left is the flow for generating small bitmaps. These bitmaps are optimally routed with optimal algorithm. On the right is the flow for generating complex training dataset by joining the routed small bitmaps.

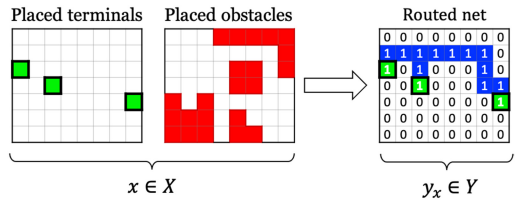


Fig. 6. Illustration of a single pathfinding task with input bitmap $x \in X$ and the corresponding solved path $y_x \in Y$.

Note that the overall objective is to maximize the total number of routed paths, while minimizing the total length of the pathfinding solution. In the ML domain, the goal is to train an ML system $\hat{Y} = \lfloor f(X, G^*) + 0.5 \rfloor$ that for each $x \in X$ provides the conditional probability of each tile, $\hat{y}_x(i, j)$, to be either included within (i.e., $f_{(i,j)} \geq 0.5$) or excluded from (i.e., $f_{(i,j)} < 0.5$) the preferred pathfinding solution,

$$f_{(i,j)}(X, G^*) = P_{G^*}(\hat{y}_{x(i,j)} = 1|x), \tag{3}$$

where G^* is the generator model trained based on the conditional probability distribution of the input features, x_i, and output observations, y_i (i.e., true labels), as defined by (2). The training

dataset $\{(x_k, y_k)\}_{k=1}^{N}$ comprises N synthetic pathfinding tasks in the bitmap representation and N corresponding reference single-path routed bitmaps (i.e., the true labels).

Mean square error (MSE) loss function is typically used with autoencoders for evaluating sum of squared distances between the predicted values and true labels [7, 27]. For the pathfinding task, MSE counts the number of tiles that marked differently ("0" vs. "1") with the true label and generated solution. Note that for an $n \times n$ optimally routed bitmap, the number of empty ($f_{(i,j)} < 0.5$) and routed ($f_{(i,j)} >= 0.5$) tiles scales as, respectively, $O(n^2)$ and $O(n)$ with n). This unbalanced nature of the pathfinding dataset fosters prioritization of the "all zeros" solution (i.e., an empty layout), which validity further increases with the increasing bitmap size n. Thus, MSE loss function is impractical for ML pathfinding.

To account for specifics of path minimization task, a custom loss function is proposed. The custom loss function is designed to penalize the model if the number of tiles, $\hat{n}_t$, included by the model within an output path is different from the number of tiles in a reference output path, n_t. The penalties for $\hat{n}_t$ exceeding and falling short of n_t differ. A path with redundant tiles is not optimal in terms of the path length, but is legal if it connects all the input terminals. Alternatively, if $\hat{n}_t < n_t$ and the reference path is optimal, then some components in the model solution are disconnected and the path is, therefore, incorrect. In particular, the $\hat{n}_t < n_t$ penalization pertain to the "all-zeros" local minimum. Given a predicted routed bitmap, $\hat{y}$, and a reference bitmap, y, the proposed loss function accounts for $|\hat{n}_t - n_t| \neq 0$ with penalty rate of $k_{sub-opt}$ and for incomplete paths with additional penalty rate of k_{err}, yielding

$$L_r = \text{MSE}\,(\hat{y}, y) \cdot (1 + k_{\text{sub-opt}} \cdot \text{step} \cdot \text{distance}), \tag{4}$$

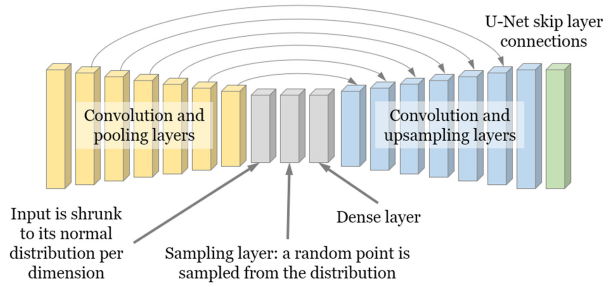
where

$$\text{distance} = \sum_{i,j} H(\hat{y}_{i,j}) - \sum_{i,j} H(y_{i,j}), \tag{5}$$

$$\text{step} = k_{\text{err}} \cdot \text{sign}(\text{distance} - 1) + 1. \tag{6}$$

Here, $H(\cdot)$ is the Heaviside step function. In this article, the proposed loss function is used with $k_{\text{sub-opt}} = 10^{-3}$ and $k_{\text{err}} = 10^{2}$.

The proposed generator is designed as a multi-stage NN. All tiles with the individual obstacle and terminal indicators are fed as ML features into the input channels of the generator. The input dimension of the network is therefore $2n^2$, as determined by the total number of features of the $n \times n$ tile bitmap. To mitigate the high input dimensionality of the system, a ConvNet-based VAE is used. A typical VAE architecture (see Figure 7) is utilized, comprising seven encoding layers, three latent dense layers, seven decoding layers, and a single refining layer. In dense layers, 25% of inputs are dropped out (i.e., set to zero) at each update during the training to prevent overfitting. The encoder converts the $2n^2$-dimensional input data into an intermediate low-dimensional (i.e., 256×2) data space, using a stack of convolutional layers. The decoder then deconvolves the abstracted data into the n^2-dimensional routed output space, using a stack of deconvolutional layers. Each of the output values indicates the probability of the corresponding tile to be included within the output path. The final decision to include a tile within the pathfinding output is made based on the decision threshold of 0.5. If a tile output value exceeds this threshold, then the tile is considered to be part of the output path. Unlike other generative models that require more complex training approaches, the proposed NN configuration is a linear stack of layers and naturally supports error backpropagation throughout the overall network. As a result, efficient training of the VAE generator within the intermediate low-dimensional dense layers is possible. Owing to the stochastic nature of the ML model, new (unseen) pathfinding solutions can be generated based on the training data points (i.e., existing or synthetically generated routed bitmaps) by sampling the



(a)

Layer id	Layer type	Number of filters/neurons	Kernel size	Stride	Padding	Activation function
1	Convolutional	32	3x3	1	Same	ReLu
2	Pool		2x2			
3	Convolutional	64	3x3	1	Same	ReLu
4	Pool		2x2			
5	Convolutional	128	3x3	1	Same	ReLu
6	Pool		2x2			
7	Convolutional	256			Same	ReLu
8	Dense	512* 25% dropout				ReLu
9	Sampling					ReLu
10	Dense	256* 25% dropout				ReLu
11	Convolutional	256	3x3	1	Same	ReLu
12	Upsampling		2x2			
13	Convolutional	128	3x3	1	Same	ReLu
14	Upsampling		2x2			
15	Convolutional	64	3x3	1	Same	ReLu
16	Upsampling		2x2			
17	Convolutional	32		1	Same	Sigmoid

(b)

Fig. 7. Architecture of the proposed generator network, (a) block-level schematics comprising convolutional, dense, and deconvolutional layers, as well as an example of input features and output, and (b) NN parameters for each layer of the $1{,}024 \times 1{,}024$ pathfinder.

intermediate probabilistic space. For optimization reasons, the generator NN is augmented with skip connections between corresponding convolutional and deconvolutional layers, similar to the U-Net [23] NN architecture.

3.3 Postprocessing Algorithm

While several missing or corrupted pixels typically go unnoticed in ML-generated images, missing path pixels yield an invalid path. A postprocessing algorithm is proposed to merge the few disconnected cGAN output clusters, as needed. As part of the algorithm, an invalid path is processed with the median filter for image noise reduction and the path tiles clusters are connected with the cluster merging algorithm (see Figure 8).

With the proposed algorithm, pairs of closest endpoints (from two different clusters) are identified for all disconnect endpoints based on Manhattan distance. To merge two clusters, the identified closest endpoints are connected with maze-routing algorithm. While the proposed greedy algorithm is generally suboptimal, it has been shown to exhibit optimal results when used to connect only few clusters in cGAN-generated paths.

An example of the postprocessed output is shown in Figure 9. In this case, the cGAN routed path exhibits three disjoint clusters that are joined with the proposed postprocessing algorithm. Note that with a completely random input (i.e., randomly generated number and location of pins and obstacles), most of the unprocessed cGAN routed paths yield between 2 and 10 disjoint clusters. This behavior is primarily caused by overfitting issues and can be solved with either the proposed postprocessing or fine-tuning of training set. When the cGAN model struggles to converge to a

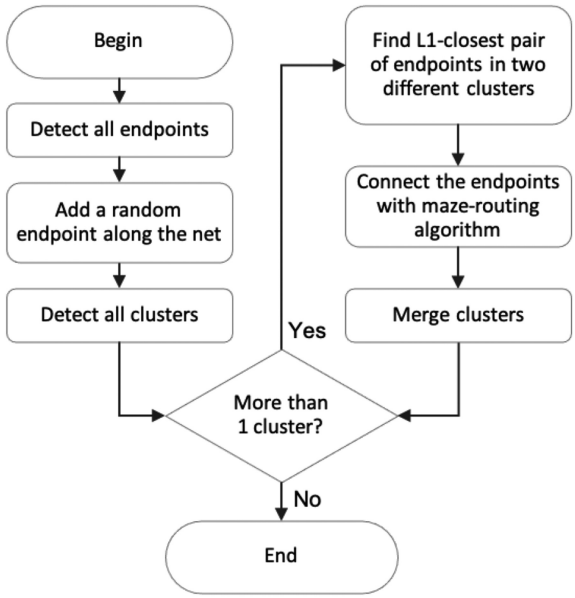


Fig. 8. Flow diagram of the proposed postprocessing method.

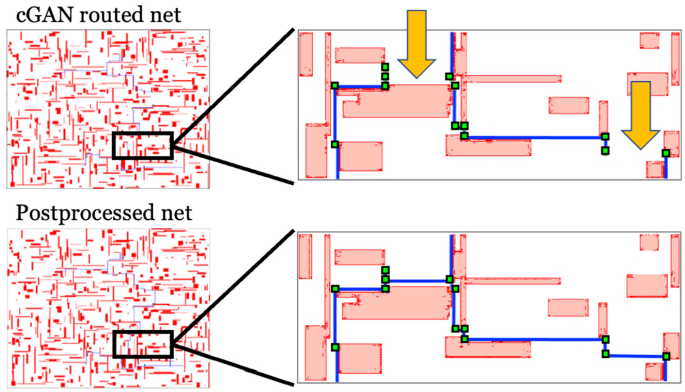


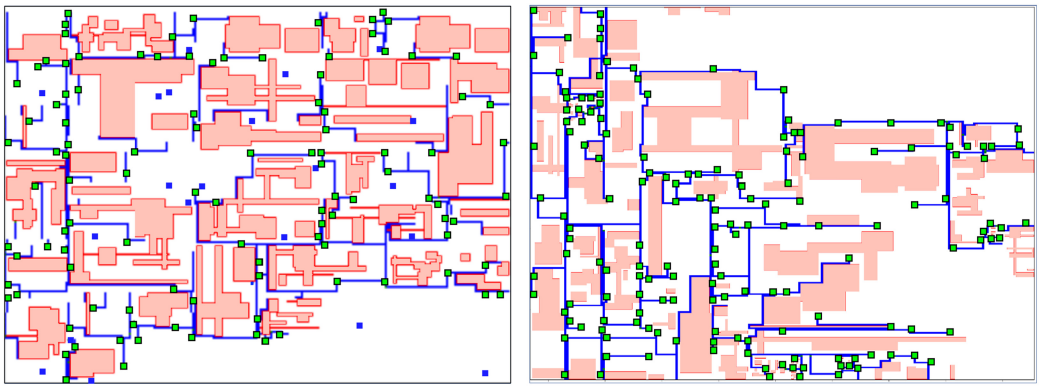
Fig. 9. Raw cGAN output of RT01 and a refined solution. Three disjoint clusters are connected during the postprocessing stage.

correct output, a grid of dots with tile-like size is produced, as shown in Figure 10. This is, however, not a fundamental limitation of the proposed approach, but a constraint of the utilized synthetic training set. With a more heterogeneous training set, which contains larger diversity of generated tile sizes, postprocessing may not be required. Such training set generation methods should be considered in the future.

4 EVALUATION AND EXPERIMENTAL RESULTS

The proposed cGAN pathfinder has been tested with the following pathfinding inputs:

(1) A set of unseen pathfinding tasks synthetically generated with the proposed algorithm (see Section 3.1.).



(a) (b)

Fig. 10. Illustration of the model output. Terminals (enlarged for visibility), obstacles, and routed paths are shown in, respectively, green, red, and blue shades. (a) An incorrect output with grid-like path is generated due to the tiled nature of the training set, and (b) A correct output with no scattered path pieces.

(2) A set of pathfinding tasks generated by randomly placing rectangular obstacles and terminals within a bitmap. The number and size of the obstacles as well as the number of terminals are all randomly determined.
(3) Multiterminal pathfinding benchmarks RT 01-05 [16].

For evaluation purposes, the input data is represented as a $1{,}024 \times 1{,}024 \times 2$ array, in which the first and second channel are the per-tile obstacle and terminal indicators, respectively. Bitmaps smaller than $1{,}024 \times 1{,}024$ are upsized to $1{,}024 \times 1{,}024$ and/or filled with obstacle indicators along the bitmap edge.

4.1 Evaluation Metrics

The cGAN pathfinding algorithm is evaluated with respect to four primary metrics: correctness of paths (i.e., a correct path must connect all the terminals in a continuous manner), wirelength (as determined by the number of path tiles), runtime, and throughput. To evaluate the correctness of the ML pathfinding, best-find search is utilized to find all connected tiles within the routed path (i.e., those tiles with $f_{(i,j)} \geq 0.5$). Each search starts at one of the terminals and progressively constructs a set of visited tiles. At each iteration, the tiles adjacent to the already traversed tiles are added to the set. The search stops when all the tiles have been traversed.

4.2 Experimental Results

The cGAN pathfinder is trained on a synthetically generated pathfinding dataset and tested on both the RT multiterminal pathfinding benchmarks [16] and synthetically and random generated test cases. The tested tasks are not part of the training set and have never been seen by the cGAN model. The cGAN pathfinder is able to generate a path with similar to state-of-the-art length for inputs of different complexity. An example of the cGAN pathfinder output at various stages of training is shown in Figure 11. Performance comparison between the cGAN and state-of-the-art deterministic pathfinding algorithms (ML-OARSMT [16] and FOARS [1]) is listed in the table below for the RT benchmarks. Note that the size and complexity of these benchmarks are typical for commercial multiterminal pathfinding cases in applications such as IC design global and detailed routing, where a typical number of net terminals ranges between 10 and 1,000 [8, 19].

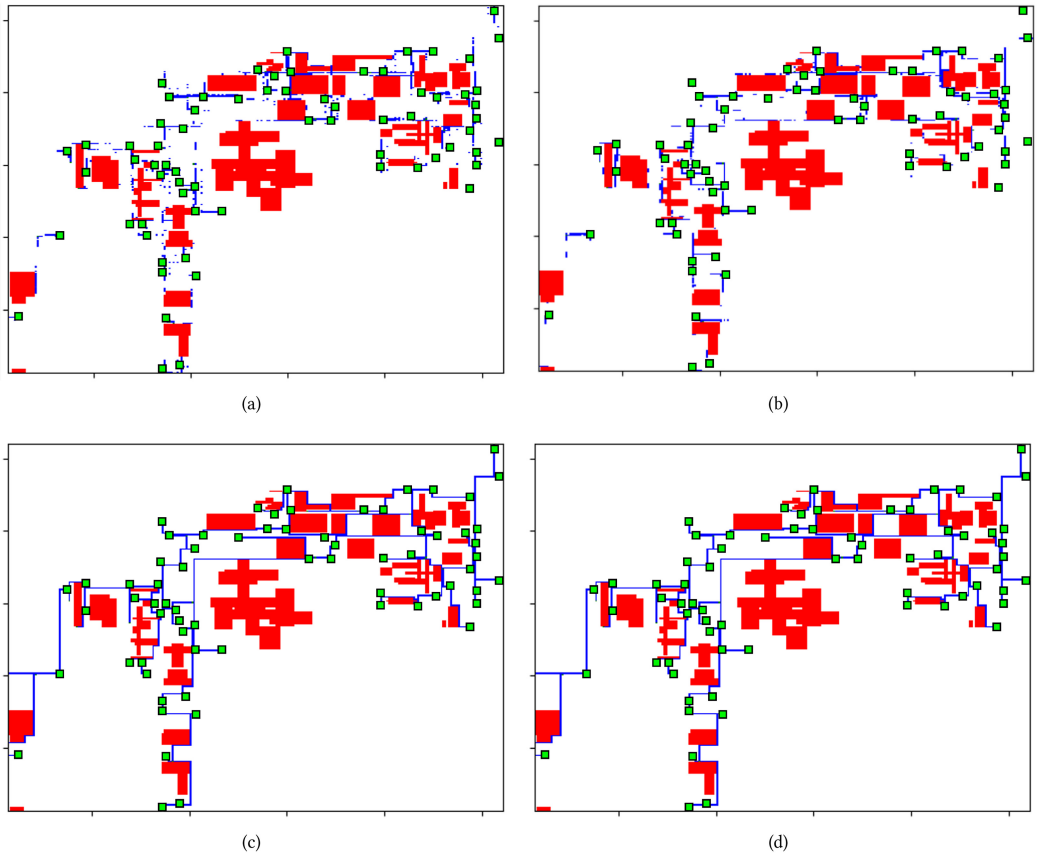


Fig. 11. ML output at different stages of model training without postprocessing. Terminals (enlarged for visibility), obstacles, and routed paths are shown in, respectively, green, red, and blue shades. Models at early training stages ((a) and (b)) produce invalid noisy outputs. Alternatively, a converged model produces a valid output with the same training set (c), similar to the optimally routed path (d).

Both deterministic algorithms are based on the look-up table-accelerated decomposition of multiterminal pathfinding. FOARS provides an enhanced method of obstacle-aware decomposition, yielding state-of-the-art performance. While ML-OARSMT yields less competitive performance, it utilizes an open source fundamental algorithm (FLUTE) that lies at the core of other state-of-the-art pathfinders and is re-executed for a fair, hardware-specific comparison. Alternatively, FOARS cannot be conveniently re-evaluated. Thus, performance metric from the original paper [1] are considered for FOARS comparison with the cGAN pathfinder. Note, however, that the frequency of the CPUs used for pathfinding evaluation in the original and this article is comparable.

With the modern hardware accelerators, such as GPU, TPU, or NPU, batching individual ML inference requests can significantly impact ML runtime and throughput performance [21]. While the optimal batching parameters vary for different models, systems, and environments, the throughput-to-latency ratio can usually be efficiently controlled by batching within hardware constraints (e.g., batch data should fit into hardware accelerator memory). In practical applications such as IC design, millions of pathfinding tasks are solved during each pathfinding iteration. Thus, pathfinding throughput (as determined by the number of solved pathfinding tasks per unit of time) is a critical metric and should be considered along with the traditional pathfinding runtime

Table 1. Pathfinding Performance Comparison between the Proposed cGAN Pathfinder (with Postprocessing) and Traditional Pathfinders (ML-OARSMT [16] and FOARS [1])

Benchmark	$RT01$	$RT02'$	$RT03$	$RT04$	$RT05'$
Terminals	10	50	100	100	200
Obstacles	500	500	500	1,000	2,000
Wirelength (tiles)					
[16]	2,267	5,871	8,363	10,306	5,468
[1]	2,119	n/a	8,282	10,330	n/a
cGAN	2,272	5,873	8,412	10,287	5,476
Runtime (seconds)					
[16]	4.3	5.4	7.0	40.0	174.5
[1]	<0.01	0.02	0.03	0.06	0.15
cGAN	0.5	0.5	0.5	0.6	0.5
Throughput (Pathfinding tasks per seconds)					
[16]	0.23	0.19	0.14	0.025	0.0057
[1]	>100	50.0	33.3	16.6	6.7
cGAN	5.5	5.5	5.5	3.6	5.5

metric. To maximize throughput performance, a batch size of 16 samples is preferred, yielding a 5× higher throughput as compared with non-batched single pathfinding inference. To account for both the parallel hardware accelerator ML processing and sequential CPU postprocessing with the proposed approach, pathfinding throughput is determined as

$$TP_{cGAN} = \frac{BatchSize}{RT_{ML} + BatchSize \times RT_{Postprocess}}, \tag{7}$$

where RT_{ML} is the runtime to route $BatchSize$ paths with ML cGAN model, and $RT_{Postprocess}$ is the postprocessing runtime per a single path. Alternatively, the throughput of the existing CPU-based sequential approaches is determined as one over a single pathfinding runtime.

Note that the proposed formulation of pathfinding as image translation enables parallelization of pathfinding with non-branching computations, propagating the input through the directed acyclic graph of cGAN generator submodel layers. Thus, the runtime of the cGAN pathfinder is not a function of the number and configuration of terminals and obstacles. Intuitively, ML processing runtime is constant and defined by the ML model, underlying framework implementation, and hardware. Based on the experimental results, the ML processing runtime of the trained ML model executed on NVIDIA GTX1080 GPU is ≈0.4 second. Note that the postprocessing runtime varies between 0.1 and 0.2 second for all the tested data and is a function of the ML model prediction quality. Thus, the postprocessing runtime can be reduced with additional training. Alternatively, the runtime with traditional approaches increases quadratically [1, 16] with the increasing number of obstacles and terminals.

The constant runtime of the ML model is experimentally verified on a synthetic test set of 60 1,024 × 1,024 unseen pathfinding tasks generated based on the proposed methodology (see Section 3.1). The length of the synthetically generated paths is considered as the reference length in these experiments. The number of terminals among the test set paths ranges between 10 and 1,063. The total area occupied by obstacles ranges between 6.7% and 35.8%. All the 60 test samples

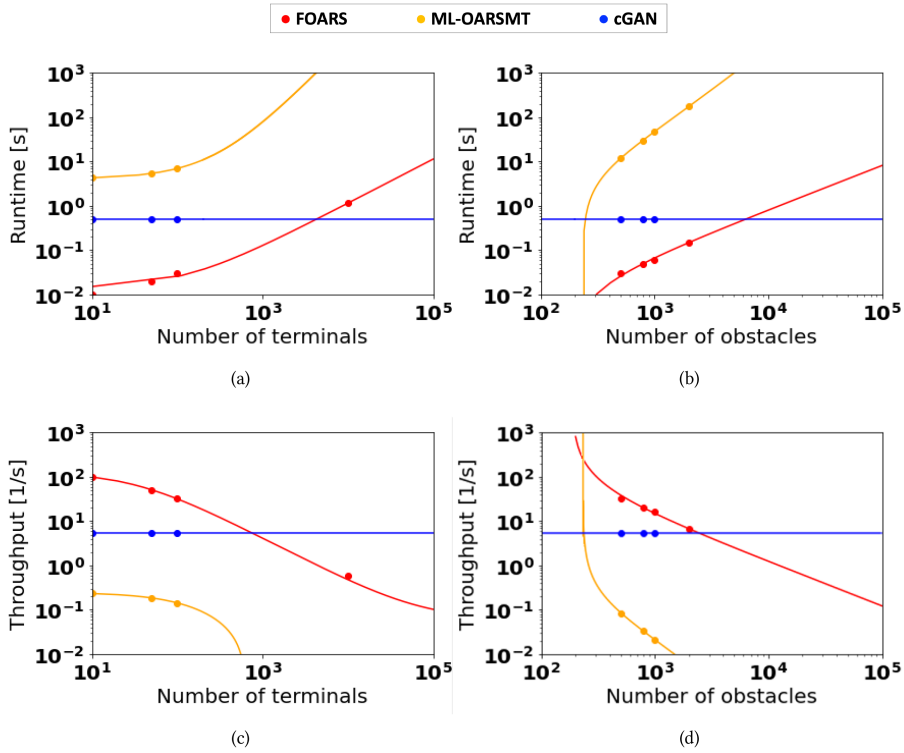


Fig. 12. Projection of multiterminal pathfinding performance with traditional state-of-the-art and proposed cGAN ML algorithms. Point markers represent existing data points. Curves represent the best fit functions: polynomial for runtime (to capture the quadratic scaling [1, 16]) and hyperbolic for throughput to capture the one over runtime behavior). Runtime as a function of terminal and obstacle count is shown in, respectively, (a) and (b). Throughput as a function of terminals and obstacles count is shown in, respectively, (c) and (d).

are routed with the cGAN pathfinder. The wirelength with cGAN is similar to the reference length, and the runtime for all the paths varies between $0.4 + 0.1 = 0.5$ and $0.4 + 0.2 = 0.6$ second.

The cGAN pathfinder is further evaluated with standard RT benchmarks. The experimental results are listed in Table 1. As expected, the speedup with cGAN pathfinder over sequential state-of-the-art continuously increases with the increasing number of terminals and obstacles. Similarly, the throughput gain with cGAN also increases in more complex pathfinding problems. The projection of these trends is shown in Figure 12 based on extrapolated results from Table 1 and Reference [1]. The data is extrapolated as follows: (i) The reported worst-case complexity of traditional pathfinding algorithms is $O(n^2)$, where n is the number of terminals or obstacle corners of the input. (ii) The throughput of these algorithms is approximated as a reciprocal function of runtime. The cGAN pathfinder outperforms the FOARS algorithm in terms of runtime and throughput (for $\approx n > 10^3$–10^4 – a realistic number of terminals and obstacles in modern and future pathfinding tasks). The cGAN pathfinder outperforms the ML-OARSMT algorithm by over an order of magnitude even in small pathfinding systems. As compared with the proposed method, the traditional pathfinders are less practical in tasks with high number of terminals and obstacles.

5 CONCLUSION

This work shows that a multiterminal obstacle avoiding pathfinding can be efficiently solved with a generative cGAN model executed on effective hardware accelerators, such as GPU, TPU,

or NPU hardware. Based on the experimental results, the proposed cGAN model correctly determines paths in unseen benchmarks, yielding a state-of-the-art-like path length and in those larger systems over an order of magnitude speedup and throughput gain. The proposed approach exploits the grid-like structure, which is most common in routing and navigation systems, to map the input pathfinding tasks and output paths to two-dimensional bitmaps and reduce the multiterminal obstacle-avoiding pathfinding to an image-to-image mapping. The proposed framework is enhanced with field-aware information and methodology for designing robust routed training dataset. Executing the pathfinding on parallel hardware accelerators allows to simultaneously and efficiently process high number of pathfinding tasks without additional overheads. The proposed cGAN pathfinding architecture and the methodology for designing synthetically obtained training samples enables a fundamentally novel approach for obstacle-avoiding multiterminal pathfinding in modern computing systems. This approach is expected to overcome some of the existing CPU bottlenecks by utilizing GPU or other parallel processing hardware. In particular, cGAN pathfinder is effective in industrial IC physical design tasks such as global routing and placement, as well as in autonomous vehicle navigation and planning.

REFERENCES

[1] Gaurav Ajwani, Chris Chu, and Wai-Kei Mak. 2011. FOARS: FLUTE-based obstacle-avoiding rectilinear Steiner tree construction. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst.* 30, 2 (2011), 194–204. DOI: https://doi.org/10.1109/TCAD.2010.2096571

[2] Wei-Ting J. Chan, Yang Du, Andrew B. Kahng, Siddhartha Nath, and Kambiz Samadi. 2016. BEOL stack-aware routability prediction from placement using data mining techniques. In *IEEE 34th International Conference on Computer Design (ICCD'16)*. IEEE, 41–48.

[3] Wei-Ting J. Chan, Pei-Hsin Ho, Andrew B. Kahng, and Prashant Saxena. 2017. Routability optimization for industrial designs at sub-14nm process nodes using machine learning. In *ACM International Symposium on Physical Design*. 15–21.

[4] Huang-Yu Chen, Chin-Hsiung Hsu, and Yao-Wen Chang. 2009. High-performance global routing with fast overflow reduction. In *Asia and South Pacific Design Automation Conference*. IEEE, 582–587.

[5] Ruoyu Cheng and Junchi Yan. 2021. On joint learning for solving placement and routing in chip design. *Adv. Neural Inf. Process. Syst.* 34 (2021), 16508–16519.

[6] Chris Chu. 2004. FLUTE: Fast lookup table based wirelength estimation technique. In *IEEE/ACM International Conference on Computer-aided Design*. IEEE Computer Society, 696–701.

[7] Aditya Deshpande, Jiajun Lu, Mao-Chuang Yeh, Min Jin Chong, and David Forsyth. 2017. Learning diverse image colorization. In *IEEE Conference on Computer Vision and Pattern Recognition*. 6837–6845.

[8] Sergei Dolgov, Alexander Volkov, Lutong Wang, and Bangqi Xu. 2019. 2019 CAD contest: LEF/DEF based global routing. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'19)*. 1–4. DOI: https://doi.org/10.1109/ICCAD45719.2019.8942107

[9] Jin Hu, Jarrod A. Roy, and Igor L. Markov. 2010. Completing high-quality global routes. In *19th International Symposium on Physical Design*. ACM, 35–41.

[10] Sambhav R. Jain and Kye Okabe. 2017. Training a fully convolutional neural network to route integrated circuits. *arXiv preprint arXiv:1706.08948* (2017).

[11] Daniel Juhl, David M. Warme, Pawel Winter, and Martin Zachariasen. 2018. The GeoSteiner software package for computing Steiner trees in the plane: An updated computational study. *Math. Program. Computat.* 10, 4 (2018), 487–532.

[12] Andrew B. Kahng, Jens Lienig, Igor L. Markov, and Jin Hu. 2011. *VLSI Physical Design: From Graph Partitioning to Timing Closure*. Springer Science & Business Media.

[13] Ilgweon Kang, Dongwon Park, Changho Han, and Chung-Kuan Cheng. 2018. Fast and precise routability analysis with conditional design rules. In *20th System Level Interconnect Prediction Workshop*. 1–8.

[14] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. 2019. PyTorch-BigGraph: A large-scale graph embedding system. *arXiv preprint arXiv:1903.12287* (2019).

[15] Haiguang Liao, Wentai Zhang, Xuliang Dong, Barnabas Poczos, Kenji Shimada, and Levent Burak Kara. 2020. A deep reinforcement learning approach for global routing. *J. Mechan. Des.* 142, 6 (2020).

[16] Chung-Wei Lin, Shih-Lun Huang, Kai-Chi Hsu, Meng-Xiang Lee, and Yao-Wen Chang. 2008. Multilayer obstacle-avoiding rectilinear Steiner tree construction based on spanning graphs. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst.* 27, 11 (2008), 2007–2016.

[17] Siting Liu, Qi Sun, Peiyu Liao, Yibo Lin, and Bei Yu. 2021. Global placement with deep learning-enabled explicit routability optimization. In *IEEE/ACM Design, Automation and Test in Europe Conference (DATE'21)*.

[18] Dani Maarouf, Abeer Alhyari, Ziad Abuowaimer, Timothy Martin, Andrew Gunter, Gary Grewal, Shawki Areibi, and Anthony Vannelli. 2018. Machine-learning based congestion estimation for modern FPGAs. In *28th International Conference on Field Programmable Logic and Applications (FPL'18)*. IEEE, 427–4277.

[19] Stefanus Mantik, Gracieli Posser, Wing-Kai Chow, Yixiao Ding, and Wen-Hao Liu. 2018. ISPD 2018 initial detailed routing contest and benchmarks. In *International Symposium on Physical Design*. 140–143.

[20] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Dean Jeff. 2021. A graph placement methodology for fast chip design. *Nature* 27 (2021), 207–212.

[21] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. 2017. TensorFlow-serving: Flexible, high-performance ML serving. *arXiv preprint arXiv:1712.06139* (2017).

[22] Chak-Wa Pui, Gengjie Chen, Yuzhe Ma, Evangeline F. Y. Young, and Bei Yu. 2017. Clock-aware ultrascale FPGA placement with machine learning routability prediction. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'17)*. IEEE, 929–936.

[23] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-assisted Intervention*. Springer, 234–241.

[24] Aysa Fakheri Tabrizi, Logan Rakai, Nima Karimpour Darav, Ismail Bustany, Laleh Behjat, Shuchang Xu, and Andrew Kennings. 2018. A machine learning framework to identify detailed routing short violations from a placed netlist. In *55th ACM/ESDA/IEEE Design Automation Conference (DAC'18)*. IEEE, 1–6.

[25] Hao Tang, Genggeng Liu, Xiaohua Chen, and Naixue Xiong. 2020. A survey on Steiner tree construction and global routing for VLSI design. *IEEE Access* 8 (2020), 68593–68622.

[26] Dmitry Utyamishev and Inna Partin-Vaisband. 2020. Late breaking results: A neural network that routes ICs. In *57th ACM/IEEE Design Automation Conference (DAC'20)*. IEEE, 1–2.

[27] Ruxin Wang and Dacheng Tao. 2016. Non-local auto-encoder with collaborative stabilization for image restoration. *IEEE Trans. Image Process.* 25, 5 (2016), 2117–2129.

[28] Zhiyao Xie, Yu-Hung Huang, Guan-Qi Fang, Haoxing Ren, Shao-Yun Fang, Yiran Chen, and Jiang Hu. 2018. RouteNet: Routability prediction for mixed-size designs using convolutional neural network. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'18)*. IEEE, 1–8.

[29] Cunxi Yu and Zhiru Zhang. 2019. Painting on placement: Forecasting routing congestion using conditional generative adversarial nets. In *56th Annual Design Automation Conference*. 1–6.

[30] Zhonghua Zhou, Ziran Zhu, Jianli Chen, Yuzhe Ma, Bei Yu, Tsung-Yi Ho, Guy Lemieux, and Andre Ivanov. 2019. Congestion-aware global routing using deep convolutional generative adversarial networks. In *ACM/IEEE 1st Workshop on Machine Learning for CAD (MLCAD'19)*. IEEE, 1–6.

[31] Keren Zhu, Mingjie Liu, Yibo Lin, Biying Xu, Shaolan Li, Xiyuan Tang, Nan Sun, and David Z. Pan. 2019. GeniusRoute: A new analog routing paradigm using generative neural network guidance. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'19)*. IEEE, 1–8.

Received 15 April 2022; revised 20 September 2022; accepted 23 September 2022