A Additional details of ReSPack

A.1 Steiner tree generation

We use two different tree-building algorithms, Algorithm 2 and Algorithm 3 in ReSPack generator. The former is applied to create medium and large RSTPP instances, and the latter is implemented for extra large RSTPP instances. `SteinerTreeHeuristic` is a Steiner Tree heuristic algorithm which indicates both 2-approximation [27] and router that is similar to Prim's algorithm [49] unless otherwise specified.

Algorithm 2: `BuildSteinerTree: 2-approximation method`

Data: Graph G, candidate nodes $W \subset V$ for terminals, number of terminals k, a random sampler $\sim$, a bound for distances between each terminals $[r_{min}, r_{max}]$ given a distance metric d over the nodes in the graph G, and a function $\texttt{SteinerTreeHeuristic}(G, T)$ that outputs an approximate Steiner tree solution S via 2-approximation method given a graph G and a set of terminals T.

Result: A set of terminals T, and a feasible Steiner tree S spanning T.

1 $v \sim W$;
2 *initialize* $T \leftarrow \{v\}$; `/* initial terminal */`
3 **while** $i \leq k-1$ **do**
4 $\quad v \sim W \setminus T$;
5 $\quad$ **if** $r_{min} \leq d(v, T) \leq r_{max}$ **then**
6 $\quad\quad T \leftarrow T \cup \{v\}$;
7 $\quad\quad i \leftarrow i + 1$;
8 $\quad$ **end**
9 **end**
10 $S \leftarrow \texttt{SteinerTreeHeuristic}(G, T)$

Algorithm 3: `BuildSteinerTree: incremental tree expansion`

Data: Graph G, candidate nodes $W \subset V$ for terminals, number of terminals k, maximum number of iterations N, a random sampler $\sim$, and a bound for distances between each terminals $[r_{min}, r_{max}]$ given a distance metric d over the nodes in the graph G.

Result: A set of terminals T, and a feasible Steiner tree S spanning T

1 $v \sim W$;
2 *initialize* $T \leftarrow \{v\}$ and $S \leftarrow \{v\}$; `/* initial terminal and tree */`
3 **while** $i \leq k-1$ **do**
4 $\quad v \sim W \setminus T$;
5 $\quad$ **if** $d(v, T) \leq r_{max} \wedge d(v, S) \geq r_{min}$ **then**
6 $\quad\quad P \leftarrow \{\texttt{ShortestPath}(u, v) : u \in S\}$;
7 $\quad\quad$ **if** $P \neq \emptyset$ **then**
8 $\quad\quad\quad S \leftarrow S \cup \text{argmin}_{\pi \in P} |\pi|$;
9 $\quad\quad\quad T \leftarrow T \cup \{v\}$;
10 $\quad\quad\quad i \leftarrow i + 1$;
11 $\quad\quad$ **end**
12 $\quad$ **end**
13 **end**

The function `SteinerTreeHeuristic` runs in $O(|V|(|E| + |V|) \log |V|)$, and hence Algorithm 2 has time complexity $O(|V|(|E| + |V|) \log |V|)$, assuming $O(1)$ for each random sampling operation. This can be significantly reduced to $O(k \cdot |V||E|)$ in Algorithm 3 where its computation bottleneck $O(|V||E|)$ arises from `ShortestPath`. If line 6 is replaced with solving a path existence problem (instead of finding a shortest path) the time complexity of Algorithm 3 can be reduced up to $O(k \cdot |V|^2)$. Note that finding `LargestConnectedComponent` in Algorithm 1 runs in $O(|V| + |E|)$, and hence the overall time complexity of Algorithm 1 is at best $O(k \cdot K(|V| + |E|)|V|^2)$. Table 3 reports clock time in the generation process for each size of instance.

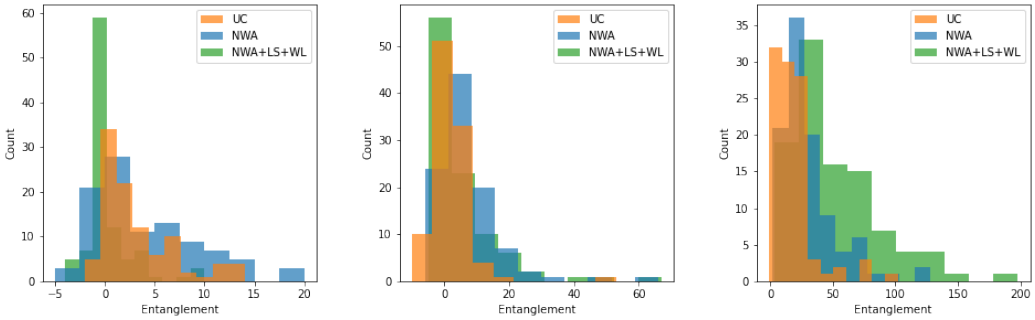
Table 3: An average generation time per sample for ReSPack datasets.

Type	Grid Size	Time / Sample
Medium	8×8	0.04 ± 0.01 s
	16×16	0.12 ± 0.01 s
	32×32	0.80 ± 0.07 s
Large	64×64	0.11 ± 0.01 m
	128×128	1.17 ± 0.12 m
	256×256	14.12 ± 1.07 m
Extra Large	512×512	2.30 ± 0.48 h
	1024×1024	32.35 ± 9.79 h

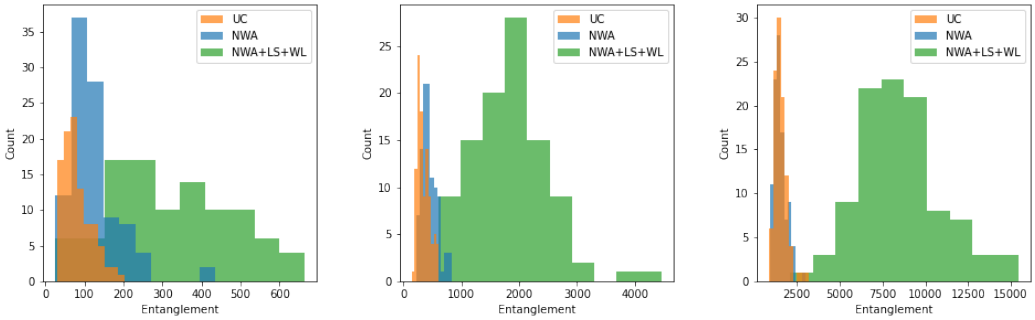
Large-scale instance generation. Due to the heavily time-consuming process for large-scale dataset generation, we adapted the evolutionary generation process for the extra large benchmark datasets to reduce computational time. It starts from an initially given instance and keeps rebuilding a small fraction of the existing Steiner trees, iteratively. In each step, the modified instances are added as new instances for RSTPP. To obtain diverse instances, it utilizes multiple initial instances and runs them in parallel using multiprocessors. The extra large-scale datasets in the benchmark are generated from 10 initial instances.

A.2 Measurement results for Steiner tree entanglement

Although an RSTPP instance consists of a tremendous number of Steiner trees, if nets are distributed sparsely on a grid, then it can be solved easily as intersections between nets are negligible. Thus, to quantify the entanglements between the trees in the RSTPP instance, we define a measure of entanglement as the difference between the total cost of the feasible solution and the sum of total costs of individual Steiner tree problems solved by the 2-approximation algorithm [27]. The problem with the large value of the entanglement where the trees are tightly entangled can be considered difficult compared to the small value because the solver should carefully consider congestion between the nets. In Figure 4, we report the distribution of entanglement for each dataset and assess the effects on the entanglement by adding constraints. For medium datasets, the constraints rarely affect the entanglement. On the other hand, large datasets are prone to increasing the entanglement as constraints are considered. Extra large datasets are excluded from the results because their computational time is too long to get the solution by the heuristic algorithms.



(a) Medium dataset: 8×8 (b) Medium dataset: 16×16 (c) Medium dataset: 32×32



(d) Large dataset: 64×64 (e) Large dataset: 128×128 (f) Large dataset: 256×256

Figure 4: Histogram of entanglement measure for each dataset.

A.3 Instance format

An instance in our benchmark consists of a pair of problem and solution, and they are given as txt format file. There are 4 properties in the problem file: INFO, GRAPH EDGES, TERMINAL NODES, CONSTRAINTS.

INFO gives a general description about the instance such as size, number of trees(nets), number of terminals for each tree. 'gird size' consists of [layer, width, height]. GRAPH EDGES gives all edges in a grid graph of the instance. Each edge is represented as a pair of nodes, and each node is represented as the xyz coordinate (z-axis is a layer). TERMINAL NODES gives positions of terminal nodes. CONSTRAINTS gives additional information about constrains of each tree. 'margin' represents line spacing constraint and 'radius' represents wire length constraint.

Below is an example of 2-layer 8×8 instance.

Problem (2-layer 8×8)

[INFO BEGIN]
grid size: [2, 8, 8]
number of trees: 2
number of terminals in each tree: [2, 4]
[INFO END]

[GRAPH EDGES BEGIN]
(0, 0, 0) - (1, 0, 0)
(0, 0, 0) - (0, 1, 0)
(0, 0, 0) - (0, 0, 1)
(1, 0, 0) - (2, 0, 0)
...
(7, 6, 0) - (7, 6, 1)
(7, 6, 1) - (7, 7, 1)
(7, 7, 0) - (7, 7, 1)
[GRAPH EDGES END]

[TERMINAL NODES BEGIN]
[TERMINAL NODES of TREE 1 BEGIN]
(1, 2, 1)
(6, 3, 0)
[TERMINAL NODES of TREE 1 END]
[TERMINAL NODES of TREE 2 BEGIN]
(1, 0, 1)
(0, 3, 0)
(1, 6, 0)
(6, 7, 0)
[TERMINAL NODES of TREE 2 END]
[TERMINAL NODES END]

[CONSTRAINTS BEGIN]
[STEINER TREE 1 CONSTRAINTS BEGIN]
margin: 1
radius: 7
[STEINER TREE 1 CONSTRAINTS END]
[STEINER TREE 2 CONSTRAINTS BEGIN]
margin: 1
radius: 25
[STEINER TREE 2 CONSTRAINTS END]
[CONSTRAINTS END]
[EOF]

Solution (2-layer 8×8)

[STEINER TREE 1 BEGIN]
(1, 2, 1) - (1, 3, 1)
(1, 3, 1) - (1, 4, 1)
(1, 4, 1) - (1, 5, 1)
(1, 5, 1) - (1, 6, 1)
(1, 6, 1) - (2, 6, 1)
(2, 6, 1) - (3, 6, 1)
(3, 6, 1) - (3, 7, 1)
[STEINER TREE 1 END]

[STEINER TREE 2 BEGIN]
(1, 5, 0) - (1, 6, 0)
...
(7, 4, 0) - (7, 5, 0)
[STEINER TREE 2 END]

[STEINER TREE 3 BEGIN]
(0, 0, 0) - (0, 1, 0)
...
(0, 4, 0) - (0, 5, 0)
[STEINER TREE 3 END]

[STEINER TREE 4 BEGIN]
(0, 1, 1) - (1, 1, 1)
...
(6, 3, 0) - (6, 3, 1)
[STEINER TREE 4 END]
[EOF]

A.4 Instance example

We provide examples of ReSPack instances in the below. Due to visibility, we only visualize instances with a size of 128×128 or less.

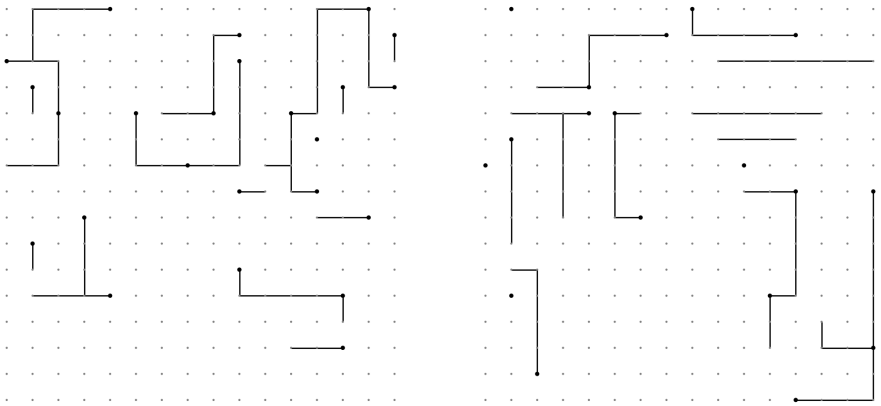


Figure 5: An instance in UC with 2-layer 16×16

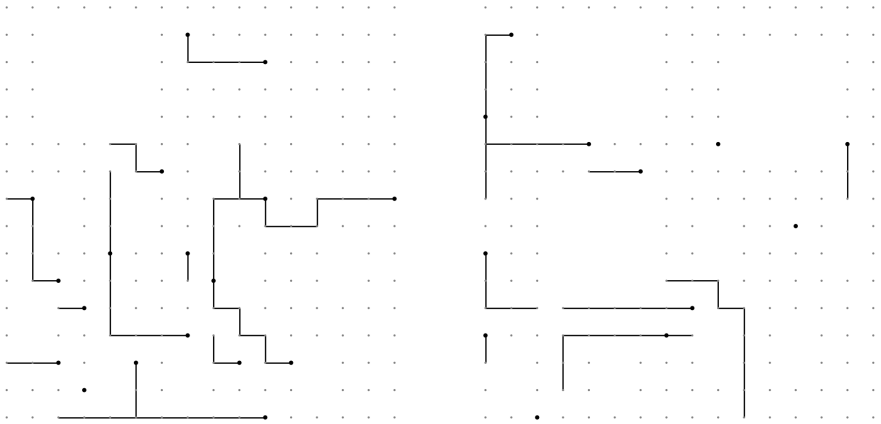


Figure 6: An instance in NWA with 2-layer 16×16

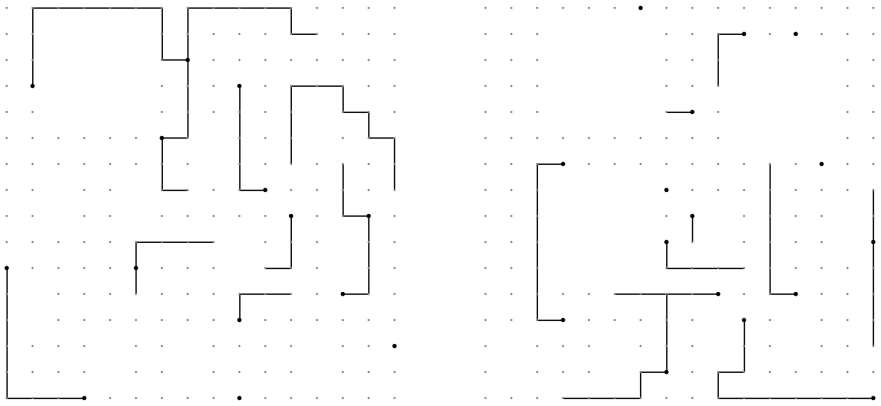


Figure 7: An instance in NWA+LS+WL with 2-layer 16×16

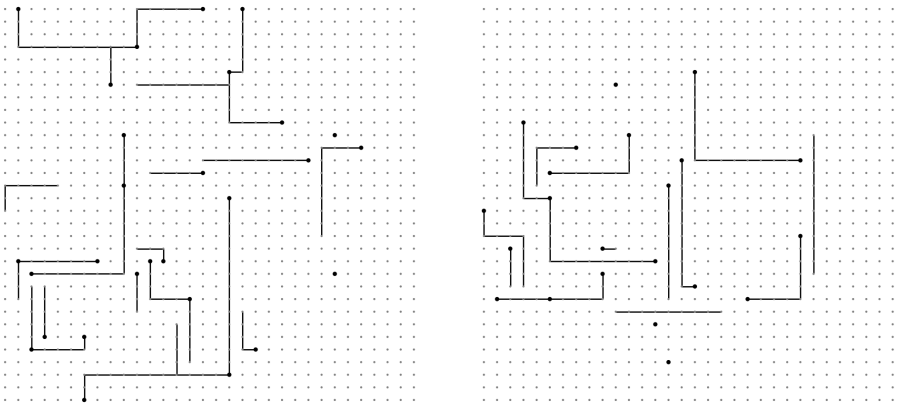


Figure 8: An instance in UC with 2-layer 32×32

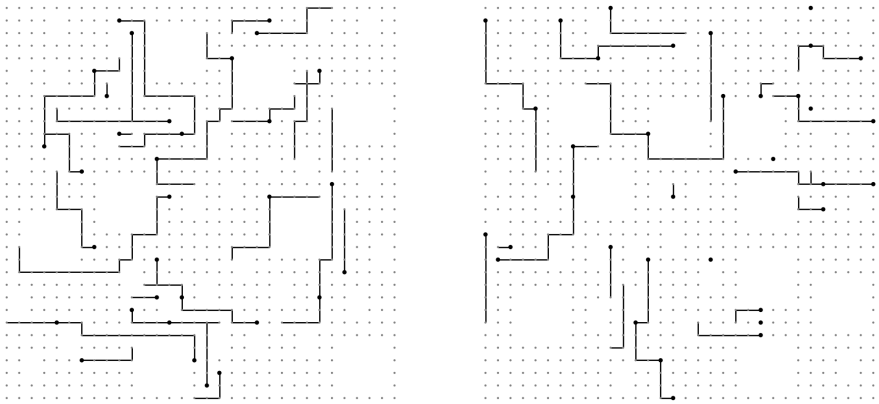


Figure 9: An instance in NWA with 2-layer 32×32

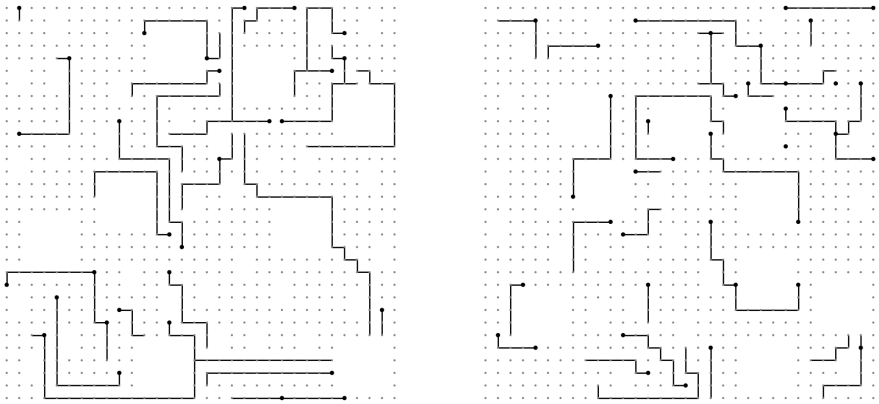


Figure 10: An instance in NWA+LS+WL with 2-layer 32×32

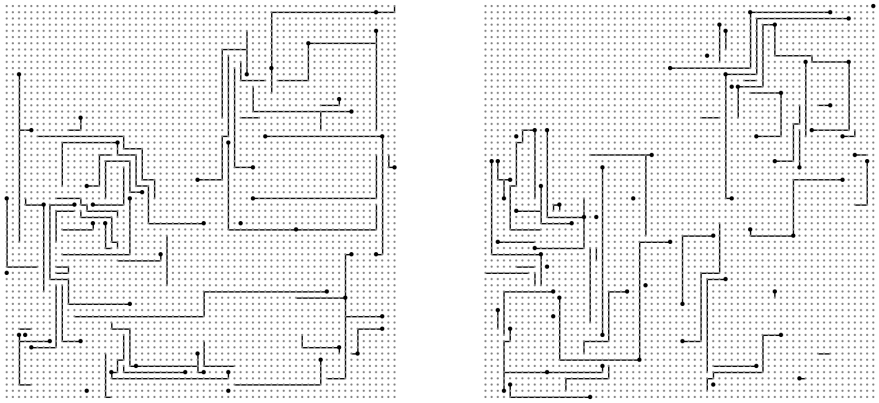


Figure 11: An instance in UC with 2-layer 64×64

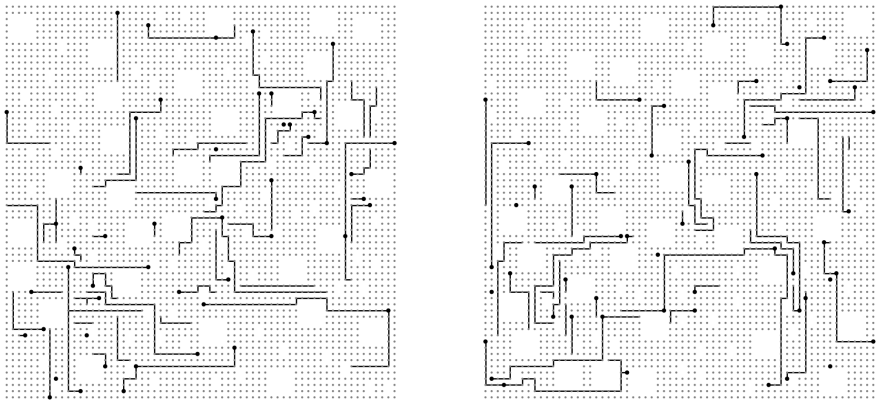


Figure 12: An instance in NWA with 2-layer 64×64

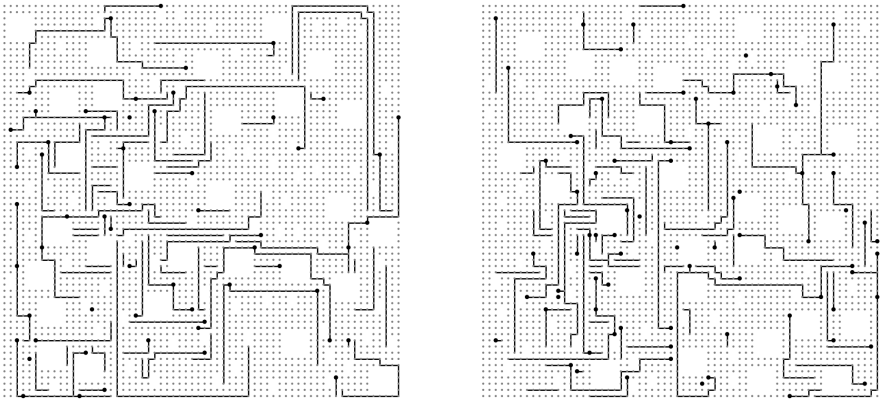


Figure 13: An instance in NWA+LS+WL with 2-layer 64×64

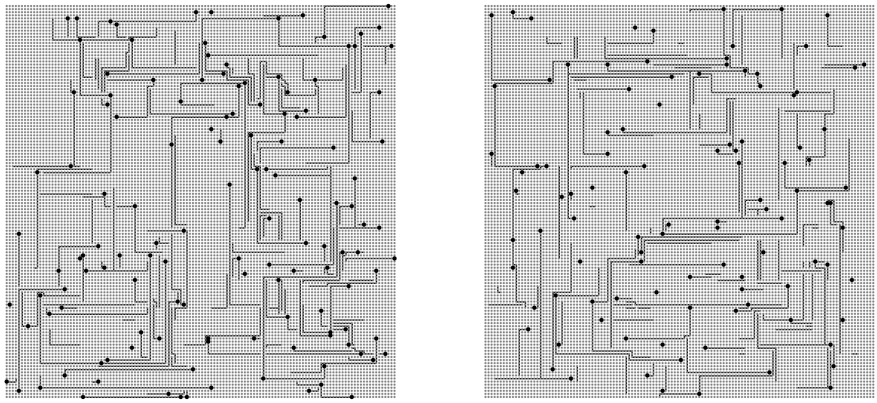


Figure 14: An instance in UC with 2-layer 128×128

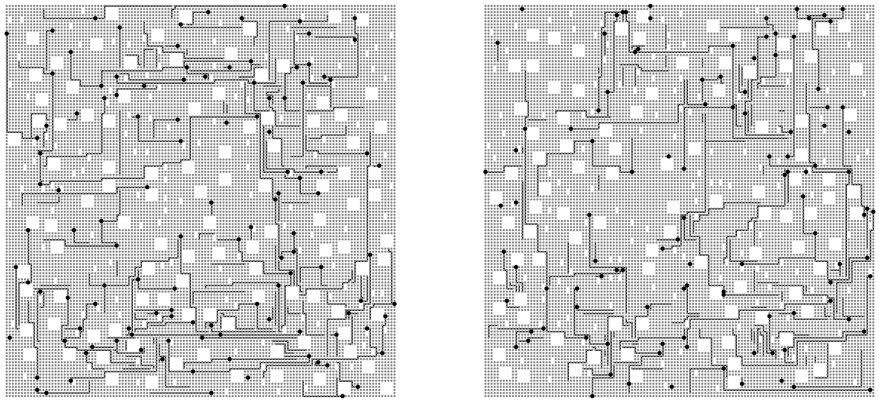


Figure 15: An instance in NWA with 2-layer 128×128

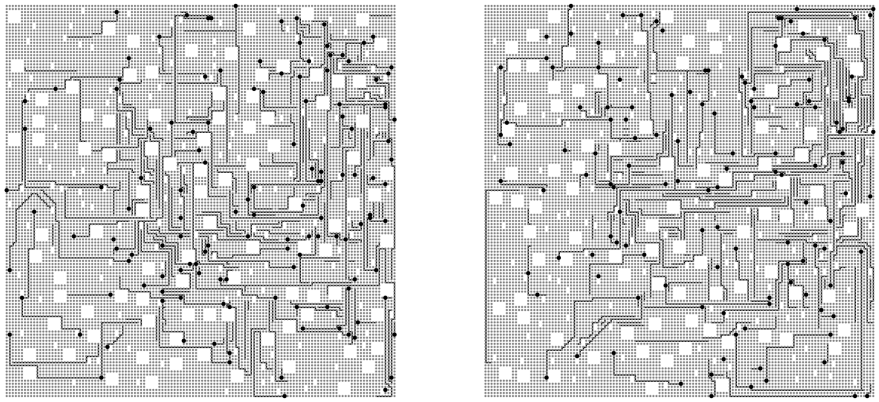


Figure 16: An instance in NWA+LS+WL with 2-layer 128×128

B Experimental details

B.1 Baseline algorithms

SCIP. We compute Steiner tree packing based on the multicommodity flow formulation [2] as follows:

$$\min \sum_{a\in\mathcal{A}} \sum_{(i,j)\in E} c_{ij}^a \bar{x}_{ij}^a,$$

$$\sum_{(i,j)\in\delta_j^-} y_{ij}^t - \sum_{(j,k)\in\delta_j^+} y_{jk}^t = \begin{cases} 1, & \text{if } j = t \\ -1, & \text{if } j = r_{\sigma(t)} \\ 0, & \text{otherwise} \end{cases} \quad \text{for all } j \in V, t \in T \setminus R,$$

$$0 \le y_{ij}^t \le \bar{x}_{ij}^{\sigma(t)} \quad \text{for all } (i,j) \in E, t \in T \setminus R,$$

$$\sum_{a\in\mathcal{A}} (\bar{x}_{ij}^a + \bar{x}_{ji}^a) \le 1 \quad \text{for all } (i,j) \in E,$$

$$\bar{x}_{ij}^a \in \{0,1\} \quad \text{for all } a \in \mathcal{A}, (i,j) \in E,$$

$$\sum_{a\in\mathcal{A}} \sum_{(i,j)\in\delta_j^-} \bar{x}_{ij}^a \le \begin{cases} 0, & \text{if } j \in R \\ 1, & \text{otherwise} \end{cases} \quad \text{for all } j \in V,$$

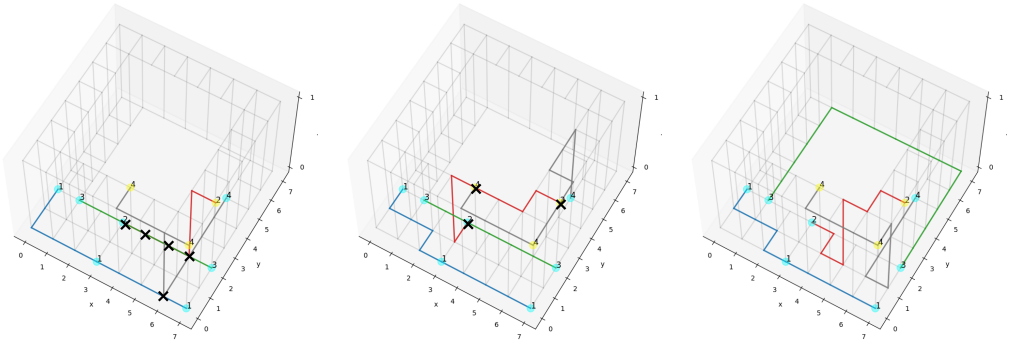
where graph $G = (V, E, c)$, the union of all terminals $T = \bigcup_{a\in\mathcal{A}} T_a$ for $\mathcal{A} = \{1, \ldots, K\}$, a set of roots $R = \{r_a | r_a \in T_a, a \in \mathcal{A}\}$, edge cost c_{ij}^a for a-th nets and edge (i,j) (in our experiments, $c_{ij}^a = 1$), outgoing edges $\delta_i^+ = \{(i,j) \in E | j \in V\}$, incoming edges $\delta_i^- = \{(j,i) \in E | j \in V\}$, $\sigma(t) = a$ for $t \in T_a, a \in \mathcal{A}$. Since we assume undirected graph in this paper, outgoing and incoming edges are equivalent, binary variables $\bar{x}_{ij}^a = 1$ if and only if edge (i,j) is contained in a-th Steiner tree.

Sequential. It is a straightforward method of routing nets sequentially, avoiding previously routed area, after determining the order of nets to be routed. The detailed algorithm is demonstrated in Algorithm 4. This method is highly dependent on net ordering, because pre-routed regions block available nodes for subsequent nets. If all terminals cannot be connected without congestion, it reorders the nets and repeats the above process. Early work by Abel [3] concluded that there is no single net ordering technique that consistently performs better than any other ordering method, so we randomly change an order in our experiments (Line 4 in Algorithm 4).

Algorithm 4: `Sequential`

Data: Graph G, candidate Steiner tree S_i for i-th nets, iterations $iteration$, cost of node n c_n, history of node n h_n, and a function `NetOrdering`(K) that outputs a list of net orders.
Result: Steiner trees S which is a set of S_i spanning T_i for $i = 1, \cdots, K$.

```
1  initialize iteration ← 0
2  while not exit condition do
3      iteration ← iteration + 1;
4      N ← NetOrdering(K) ;          /* change a net order in every iteration */
5      for i in N do
6          S_i ← SteinerTreeHeuristic(G, T_i, c);
7          if S_i = ∅ or S_i violate constraints then
8              S ← ∅
9              break;
10         end
11         for n in V(S_i) do
12             c_n ← ∞ ;                  /* can not pass previously routed path */
13         end
14     end
15 end
```



(a) PathFinder (1st iter.) (b) PathFinder (3rd iter.) (c) PathFinder (9th iter.)

Figure 17: Routing results of PathFinder. Each net differs in color, black cross denotes congested nodes, and dots indicate terminal points on 1st (blue) and 2nd layer (yellow).

PathFinder. Another sequential routing algorithm, PathFinder [39], is a negotiation congestion algorithm that routes each net and then rips up and reroutes each net several times in sequence. During the sequential iterations, cost updates of routing nodes can migrate routes from congested areas to sparsely populated areas. Figure 17 shows how PathFinder finds routes sequentially. First, as shown in Figure 17(a), PathFinder connects terminals of each net ignoring other nets. If congestion occurs, node cost is updated as follows:

$$c_n = (b_n + h_n) \cdot p_n, \tag{1}$$

where c_n is a cost of node n, b_n is the base cost of using node n, h_n is related to the history of congestion on node n, and p_n is related to the number of nets presently using n. In our experiment, we convert node cost into edge cost by averaging costs of connected nodes (*i.e.*$C(e_{ij}) = \frac{c_i + c_j}{2}$). We add $0.8 \cdot$ *#iterations* to p_n when node n is used, and define a tunable parameter $\delta(k)$ as 0.1. Figure 17(b) and 17(c) show that node cost allows router to find a path through other nodes and PathFinder finally finds a feasible solution. The detailed algorithm is demonstrated in Algorithm 5.

Algorithm 5: `PathFinder`

Data: Graph G, Steiner tree S_i for i-th nets, iterations $iteration$, cost of node n c_n, history of node n h_n, and a function `NetOrdering`(K) that outputs a list of net ordering.
Result: Steiner trees S which is a set of S_i spanning T_i for $i = 1, \cdots, K$.

initialize $iteration \leftarrow 0$ and $S_i \leftarrow \emptyset$
$N \leftarrow$ `NetOrdering`(K) ; `/* route in a pre-defined order */`
while *not exit condition* **do**
  $iteration \leftarrow iteration + 1$;
  for i *in* N **do**
    Rip up S_i;
    $S_i \leftarrow$ `SteinerTreeHeuristic`(G, T_i, c);
    for n *in* $V(S_i)$ **do**
      update c_n
    end
    if *congestion occurs in* $V(S_i)$ **then**
      for n *in* $V(S_i)$ **do**
        $h_n \leftarrow h_n + \delta(k)$
      end
    end
  end
end
if *congestion occurs or violate constraints* **then**
  $S \leftarrow \emptyset$
end

RankingCost-MT. The original RankingCost [21] assumes that every net has only two terminals and sequentially routes each two-terminal net searching a path between them based on A* algorithm. The net ordering is determined by descending order of trainable ranking parameter, for example, ranking parameter $[0.1, 0.8, 0.3]$ for three nets implies the corresponding order of nets is $[3, 1, 2]$ (Line 10 in Algorithm 6). For trainable A* router, it utilizes an additional term parameterized by trainable cost map in priority function as follows:

$$s_j(v) = g_j(v) + h_j(v) + \sum_{i=j+1}^{K} C(i, v), \tag{2}$$

where $s_j(v)$ is the priority function for a node v in j-th ordered net, $g_j(v)$ is total length from start node, $h_j(v)$ is future heuristic cost from v to end node (e.g., Euclidean distance), and $C(i, v)$ is additional trainable cost map for i-th net. In order to consider multiple terminals per net, we decompose the multiple terminals into multiple two-terminal pairs by computing a minimum spanning tree (MST) of the complete graph of the terminals as in [27]. Then, when we route one net, we exclude terminals in other nets from search space. Following OpenAI-ES [47], each worker run this procedure in parallel taking its own perturbed parameters from global parameters, and then the global parameters are updated by the collection of normalized rewards from arbitrary reward function that evaluates the given solution. We named this extension of RankingCost as RankingCost-MT and the detailed algorithm is demonstrated in Algorithm 6. For RankingCost-MT, we used the default setting of original RankingCost except for several hyperparameters due to modification for dealing with multiple terminals; learning rate 0.01, population size 30.

Algorithm 6: `RankingCost-MT`

Data: Graph G, sets of terminals $\mathcal{N} = \{T_1, \cdots, T_K\}$, a random sampler $\sim$, net ranking parameter $\theta_r \in \mathbb{R}^K$, cost map parameter $\theta_c \in \mathbb{R}^{K \times |V|}$, learning rate α, noise standard deviation σ, population size β, function $A * (G, (v_1, v_2), \theta_c)$ that outputs a path between v_1, v_2 on G given cost map θ_c, and reward function f.

Result: Steiner trees S which is a set of S_i spanning T_i for $i = 1, \cdots, K$.

initialize $iteration \leftarrow 0$;
while *not exit condition* **do**
  $iteration \leftarrow iteration + 1$;
  for $p = 1, \ldots, \beta$ **do**
    $\epsilon_p \sim N(0, I)$;
    $\hat{\theta}_r \leftarrow \theta_r + \sigma\epsilon_p$;
    $\hat{\theta}_c \leftarrow \theta_c + \sigma\epsilon_p$;
    $G' \leftarrow G$;
    for $j = 1, \ldots, K$ **do**
      $t \leftarrow$ `rank(j)` induced by $\hat{\theta}_r$;
      $\mathcal{P} \leftarrow$ all edges of MST induced by complete graph of T_t;
      $S_t \leftarrow \emptyset$;
      for $l = 1, \ldots, |T_t|$ **do**
        $G'' \leftarrow$ subgraph of G' which excludes $\bigcup_{t' \neq t} T_{t'}$;
        $\tau \leftarrow A^*(G'', P_l, \hat{\theta}_c)$;
        $S_t \leftarrow S_t \cup \tau$;
        if $\tau = \emptyset$ *or violate constraints* **then**
          $S_t \leftarrow \emptyset$;
          $break$;
        end
      end
      $G' \leftarrow$ subgraph of G' which excludes $V(S_t)$;
      if $S_t = \emptyset$ **then**
        $break$;
      end
    end
    $r_p \leftarrow f(S)$;
  end
  $\bar{r} \leftarrow (r - r_{mean})/r_{std}$; `/* parameter update with normalized reward */`
  $\theta_r \leftarrow \theta_r + \alpha \frac{1}{\beta\sigma} \sum_{j=1}^{\beta} \bar{r}_j \epsilon_j$;
  $\theta_c \leftarrow \theta_c + \alpha \frac{1}{\beta\sigma} \sum_{j=1}^{\beta} \bar{r}_j \epsilon_j$;
end
$S \leftarrow$ solution given the best θ_r, θ_c;

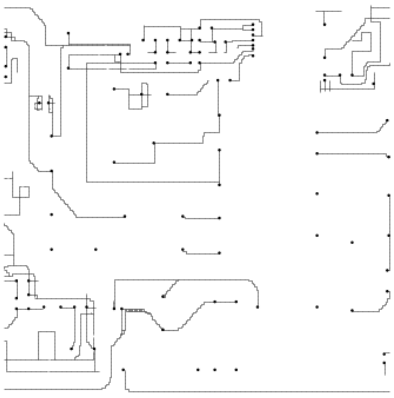
Table 4: The result on ReSPack of large size. Note that underlined metric, SR, indicates that higher is better, otherwise, Gap and Time, lower is better. Gap and Time are measured on solved instances. We denote 'FAIL' to reaching exit condition on every instance and 'N/A' to no experiment.

		2-layer 64×64			2-layer 128×128			2-layer 256×256		
		SR(%)	Gap(%)	Time	SR(%)	Gap(%)	Time	SR(%)	Gap(%)	Time
UC	SCIP	—FAIL—			—FAIL—			—FAIL—		
	Sequential-1	$6.8_{\pm 1.33}$	$+0.4_{\pm 0.59}$	47s	—FAIL—			—FAIL—		
	Sequential-2	$7.6_{\pm 2.42}$	$+0.4_{\pm 0.46}$	3m	—FAIL—			—FAIL—		
	PathFinder-1	$100.0_{\pm 0.00}$	$-0.3_{\pm 0.12}$	10s	$100.0_{\pm 0.00}$	$-0.1_{\pm 0.07}$	2m	$100.0_{\pm 0.00}$	$-0.8_{\pm 0.08}$	21m
	PathFinder-2	$100.0_{\pm 0.00}$	$-1.4_{\pm 0.04}$	35s	$100.0_{\pm 0.00}$	$-1.6_{\pm 0.03}$	6m	$100.0_{\pm 0.00}$	$-2.6_{\pm 0.02}$	53m
	RankingCost-MT	$100.0_{\pm 0.00}$	$+2.0_{\pm 0.06}$	1m	$100.0_{\pm 0.00}$	$+2.1_{\pm 0.05}$	8m	$100.0_{\pm 0.00}$	$+0.6_{\pm 0.00}$	1h
NWA	SCIP	—FAIL—			—FAIL—			—FAIL—		
	Sequential-1	$0.6_{\pm 0.80}$	$+2.1_{\pm 0.76}$	29s	—FAIL—			—FAIL—		
	Sequential-2	$1.8_{\pm 1.33}$	$-0.2_{\pm 1.06}$	2m	—FAIL—			—FAIL—		
	PathFinder-1	$100.0_{\pm 0.00}$	$-2.1_{\pm 0.07}$	10s	$100.0_{\pm 0.00}$	$-1.1_{\pm 0.06}$	2m	$100.0_{\pm 0.00}$	$-0.4_{\pm 0.07}$	26m
	PathFinder-2	$100.0_{\pm 0.00}$	$-3.7_{\pm 0.03}$	28s	$100.0_{\pm 0.00}$	$-3.1_{\pm 0.02}$	6m	$100.0_{\pm 0.00}$	$-2.2_{\pm 0.01}$	58m
	RankingCost-MT	$100.0_{\pm 0.00}$	$+1.9_{\pm 0.13}$	1m	$100.0_{\pm 0.00}$	$+2.0_{\pm 0.07}$	8m	$100.0_{\pm 0.00}$	$+1.3_{\pm 0.00}$	1h
NWA+LS+WL	SCIP	N/A			N/A			N/A		
	Sequential-1	—FAIL—			—FAIL—			—FAIL—		
	Sequential-2	—FAIL—			—FAIL—			—FAIL—		
	PathFinder-1	—FAIL—			—FAIL—			—FAIL—		
	PathFinder-2	—FAIL—			—FAIL—			—FAIL—		
	RankingCost-MT	—FAIL—			—FAIL—			—FAIL—		

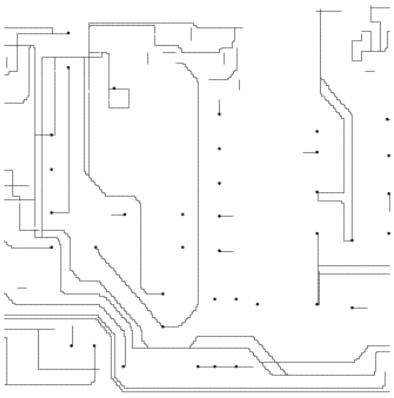
B.2 Experimental results on ReSPack of large size

C Real-world PCB examples

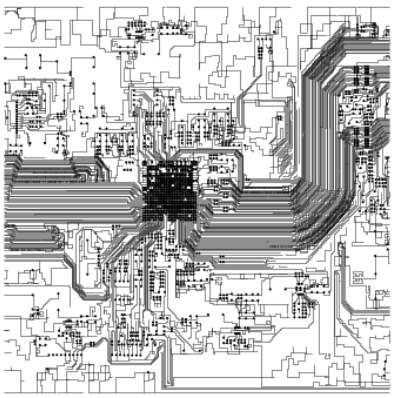
Figure 18 shows examples of RSTPP instances converted from real PCBs. We collected the real PCB data designed for commercial products by PCB artwork designers, and the original PCB data was converted into the RSTPP format of ReSPack defined above. A total of 208 collected PCB data contains wiring results by human experts as solutions, of which 167 belong to Large and 41 belong to Extra Large. Therefore, we believe that ReSPack can cover real-world wire routing problems well enough. Unfortunately, making all of these data publicly available is difficult, but we have attached some of the converted instances.



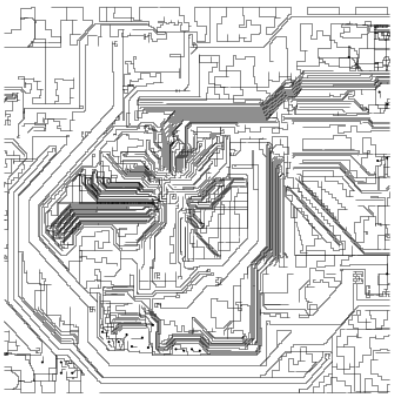
(a) Large instance (Layer-1)



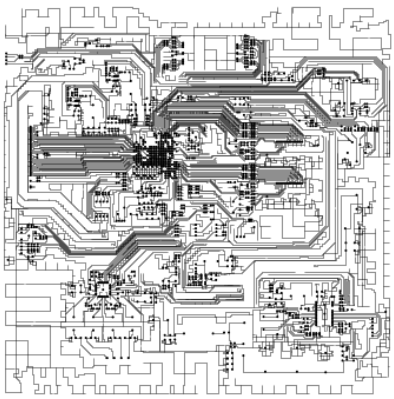
(b) Large instance (Layer-2)



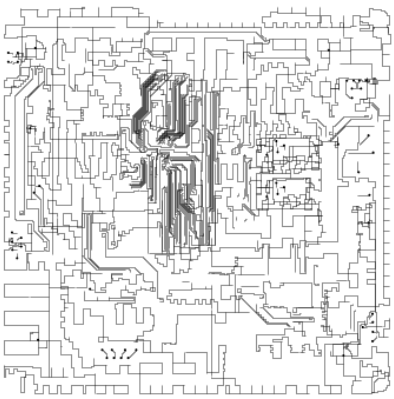
(c) Extra Large instance #1 (Layer-1)



(d) Extra Large instance #1 (Layer-2)



(e) Extra Large instance #2 (Layer-1)



(f) Extra Large instance #2 (Layer-2)

Figure 18: RSTPP instances converted from real PCBs. The two figures in the first row represent the first and second layers of a real PCB included in Large scale. The figures in the second and third rows represent the first and second layers of two real PCBs included in Extra Large scale. For confidentiality, all images are deliberately cropped to conceal some information.

D Motivation

This dataset is created to tackle RSTPP, which is one of popular CO problems and relevant to real-world wire routing problem. Especially, we provide RSTPP with various scales and constraints so that we make the dataset more realistic and challenging. We expect the dataset encourages ML community to find effective approaches for constrained RSTPP solver. One of promising direction is to build the RL environment using this dataset and problem generator, and train an agent with better performance in cost and time or generalization. A feasible solution provided by the generator can be a baseline for the evaluation or a starting point to imitate, and so on. Still, it is not the only way and we hope that more creative approaches to conquer this problem will be suggested.

E Related Work

We briefly review heuristic and machine learning based approaches for wire routing, and public benchmarks in those studies.

E.1 Routing algorithms

The routing problems are usually solved in two-stage where global routing is followed by detailed routing. During global routing stage, the wire segments are temporarily assigned to coarse circuit blocks called G-cells, which is used as guidance for detailed routing. Detailed routing routes on a more fine-grained grid. During detailed routing stage, actual path of the wires are determined while considering complicated design rules, such as minimum wire length and spacing, and global routing segments as guidance.

There are concurrent and sequential approaches in global routing. The global routing approaches can be divided into two types concurrent and sequential. Concurrent approaches [10, 11, 53, 54] route numerous nets simultaneously, and thus they are computationally expensive to be applied on large circuits. Sequential approaches [30, 12, 24, 41, 9, 43, 13, 7, 35], which are known to be effective in practice and computationally cheaper than the concurrent approaches, route nets one by one in a specific order. The main drawback of these approaches is that the quality of the solution highly relies on the order of the nets. Recently, learning based approaches has also been proposed. [33] models the routing problem as a sequential decision making problem, and solve it through deep reinforcement learning algorithm. On the other hand, [50] models the routing problem as an image-to-image processing problem where the routing in done in a single step manner with a deep generative model.

For detailed routing, [46] proposes the first maze routing based approach by applying the breadth-first search method, and [16] proposes the Minimum Detour algorithm by applying A^* heuristic search. [19] and [20] apply the line-search algorithms and improve Lee's and A^* algorithm respectively in terms of time and space efficiency. Recently, [23] proposes TritonRoute, which divides each layer into parallel panels and routes each pannel through integer linear programming. [8] proposes Dr. CU that adopts minimum-area-captured path search algorithm, and shows the state-of-the-art result. There also has been learning based approaches in detail routing, which mainly focuses on finding a best net order. [32] and [34] utilize attention-based REINFORCE method and an asynchronous actor-critic framework respectively to optimize routing ordering.

As [48] pointed out, global routing and detailed routing do not harmonize well especially for complex routing problems, and therefore there has been another line of work called area routing [22], where routes are done in a one-stage instead of two-stage. [18] also models the problem as a sequential decision making problem, and presents Monte-Carlo tree search with deep neural network guided rollout for wire routing. [21] proposes a method that combines search-based methods and learning based methods, where A^* algorithm is utilized to route each net, and the net ordering and the cost maps for A^* algorithm are learned through evolution strategy. Those works show the promise of area routing, but are restricted to small-scaled grid graphs with only 2 pins per net, and they test their algorithms on the dataset generated by themselves due to lack of an open benchmark.

E.2 Datasets

There are some public datasets for a single Steiner tree problem (STP) in CO community. SteinLib [25] is a dataset library which covers diverse STP such as randomly generated instances, collected samples from application areas such as VLSI design and telecommunication. Instances are represented in graph with nonnegative costs on edges. Vienna [31] is a collection of large, sparse STP graphs arise from real-world fiber optic telecommunication networks. Its terminals and edges represent the position of building and the construction cost, respectively. However, there is no public dataset for Seinter tree packing problems.

International Symposium on Physical Design (ISPD) has held annual contest with various circuit-design-related topics such as a placement, and it announced benchmarks of global routing in both ISPD 2007[4] and ISPD 2008[5], and initial detailed routing in both ISPD 2018[6] and ISPD 2019[7]. These routing benchmarks are less than 20 real-world samples for each task which are represented in multi-layered routing plane. The domain characteristics are well reflected in data, which is represented in the preferred routing direction for each layer (i.e., vertical, horizontal), physical design rules (e.g., line spacing) for detailed routing, etc [40, 38, 36]. It considers various design rules like ReSPack, but it has limitations that benchmark datasets are available only for two-stage based routing methods and have fewer samples than ReSPack, so that it may not be suitable for ML research where a lot of data is required

F Open Problems and Discussion

Utilizing sub-optimal solutions. ReSPack provides a sub-optimal solution for each instance, which can be utilized to find the optimal solution more efficiently. We foresee future researches that can bootstrap upon the sub-optimal solution such as via local search [1] or reducing the search space [29].

Scalable algorithm. Latest advancements in real-world inspired large-scale models for TSP, VRP, etc. are actively supported by thriving large-scaled benchmark datasets [15]. We hope that ReSPack serves as a starting point to the research in large-scale circuit routing problems.

Embedding heuristic solver into a neural network. Some recent studies show the potential that the traditional CO solver can be a composable building block of neural network architectures [42]. We believe we can apply the same intuition to improve the performance.

Constrained CO for real-world problem. Real-world wire routing problems have lots of constraints so that it needs to add more design rules in wire routing into consideration. Unfortunately, different types of circuits have different requirements (or design rules). It becomes more challenging to efficiently generate the problem instances with guaranteed feasible solution as we incorporate more design rules; hence, not scalable. Instead, a more scalable approach is to implement the design rule as a *soft* constraint, or penalty, rather than a hard constraint. RL framework provides an interface to model such soft constraint as a form of negative reward. Indeed, the deep RL has emerged as a promising way to build a scalable solution to tackle CO problems [5, 26]. However, while RL has been applied to many popular CO problems, application to Steiner tree (packing) problem has been limited. Our dataset and benchmark provides a basic building blocks (*e.g.*, feasibility checker and imperfect solution) to build the RL environment, but we leave building the environment with complex design rules as a future work.

[4] http://www.sigda.org/ispd2007/contest.html
[5] http://www.ispd.cc/contests/ispd08rc.html
[6] http://www.ispd.cc/contests/18/index.htm
[7] http://www.ispd.cc/contests/19/index.htm