Late Breaking Results: Subgraph Matching Based Reference Placement for PCB Designs*

Miaodi Su[1,2], Yifeng Xiao[3], Shu Zhang[2], Haiyuan Su[2], Jiacen Xu[4], Huan He[4], Ziran Zhu[5], Jianli Chen[1], Yao-Wen Chang[6,7]
[1]State Key Lab of ASIC & System, Fudan University, Shanghai 200433, China
[2]College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350108, China
[3]Department of Electrical Engineering, University of Southern California, Los Angeles, CA, USA
[4]Hangzhou Huawei Enterprises Telecommunication Technologies Co., Ltd, Hangzhou 310000, China
[5]National ASIC System Engineering Center, Southeast University, Nanjing 210096, China
[6]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 10617, Taiwan
[7]Department of Electrical Engineering, National Taiwan University, Taipei 10617, Taiwan
200320022@fzu.edu.cn; zrzhu@seu.edu.cn; chenjianli@fudan.edu.cn

ABSTRACT

Reference placement is promising to handle the increasing complexity in PCB design. We model the netlist into a graph and use a subgraph matching algorithm to find the isomorphism of the placed template in component combination to reuse the placement. The state-of-the-art VF3 algorithm can achieve high matching accuracy while suffering from high computation time in large-scale instances. Thus, we propose the D2BS algorithm to guarantee matching quality and efficiency. We build and filter the candidate set (CS) according to designed features to construct the CS structure. In the CS optimization, a graph diversity tolerance strategy is adopted to achieve inexact matching. Then, hierarchical match is developed to search the template embeddings in the CS structure guided by branch backtracking and matched nodes snatching. Experimental results show that D2BS outperforms VF3 in accuracy and runtime, achieving 100% accuracy on PCB instances.

1 INTRODUCTION

Printed circuit board (PCB) placement is a crucial procedure in industrial chip design. However, it still highly relies on manual methods and thus consumes significant design time, especially for large-scale designs. There often exist the same modules that need the same placement, so reference placement has emerged as key research to improve PCB placement efficiency. Reference placement aims to take the placed module as a template and find out the components with the same pattern to be placed for the same placement. To handle this problem, we model the netlist information into a graph and apply the subgraph matching algorithm to achieve high-quality reference placement.

Efficiency has always been a challenging problem for subgraph matching, especially for large-scale graphs. The latest algorithms VF3 [2] and DAF [3] are proposed to address the case of the graphs with thousands of nodes and a high edge density. However, they still exhibit some intrinsic limitations. First, these methods all use the backtracking technique to explore the search paths node by node. So redundant computation is inevitable. Second, exact matching algorithms cannot handle the pattern differences between the query and data graphs well, leading to inflexible placement reuse. To handle these challenges, we propose an efficient subgraph matching algorithm called D2BS to facilitate PCB reference placement. Experimental results show that D2BS outperforms VF3 in matching accuracy and running time.

2 PRELIMINARY

2.1 Terminologies

- **Query graph (template)**: a combination of placed electric components and routed nets selected by users for reuse, represented as $Q = (V_q, E_q)$, where V_q is the node set and E_q is the edge set.
- **Data graph**: components and nets not yet placed and routed, represented as $D = (V_d, E_d)$.

*This work was supported by the Young Scientist Project of MOE Innovation Platform, the State Key Laboratory of ASIC System under Grant 2021MS008.

DAC '22, July 10–14, 2022, San Francisco, CA, USA

ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530670

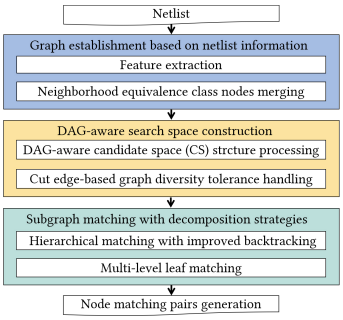

Figure 1: The overall framework of the proposed D2BS.

2.2 Problem Formulation

In this paper, the placed template is selected as a query graph, and the components and nets to be placed are regarded as a data graph. As a result, reference placement can be formulated as a subgraph matching problem.

Problem 1. *Given a data graph D and query graph Q, the reference placement problem aims to find all embedding graphs of Q in D.*

3 ALGORITHM

For reference placement, we propose a subgraph matching algorithm named D2BS. The overall framework of our algorithm is shown in Figure 1. We detail the three phases as follows.

3.1 Graph Construction

We convert the components and nets into component nodes and net nodes. Component nodes can only connect to net nodes and vice versa, so Q and D are both bipartite graphs. In our designs, the parameter properties of components and nets are recorded as features of nodes. For components, the features contain the component type, the type and number of neighbor components, whether to connect to power or ground. For nets, the features contain the net type, connected component types and pins number. To shorten the traversal time of the template, we reduce the number of nodes by merging the neighborhood equivalence class (NEC) nodes [4] to a hypernode. Q denotes the query graph after NEC merging in the following sections.

3.2 Search Space Construction

3.2.1 ***DAG-aware CS Structure Processing.*** Candidate set stores the data nodes that can be matched to the corresponding query node. With smaller candidate sets, the search space is reduced. Therefore, we propose a candidate set creation and screening method to construct and narrow the search space. The initial candidate set of a query node includes the data nodes with the same node types. Then, we develop the similarity score for each initial candidate according to features. As a result, the initial candidate set is screened by reserving the candidates with the highest similarity score.

In order to refine the search space, DAG (directed acyclic graph) and CS in [3] are adopted in our algorithm. However, undirected circles in DAG may confuse the matching process. Hence, we delete one of the edges that can be closed to guarantee no circles exist, so as to avoid matching errors. DAG is a connected graph, so Q should be divided according to the connectivity before

Algorithm 1 Subgraph hierarchical match

Input: BfsTree, $root$.
Output: matchPair.

```
1: matchPair ← ∅, layerIdx ← 1.
2: matchPair.append([root, v_r]) ← v_r is a candidate of root.
3: while layerIdx < len(BfsTree) do
4:   unvisitedB ← The nodes that are not visited in BfsTree[layerIdx].
5:   for u in unvisitedB do
6:     isBack, matchPair, layerIdx← NodeCheck(u, matchPair, layerIdx)
7:     if isBack == True then
8:       Break
9:     end if
10:  end for
11:  layerIdx ← layerIdx + 1
12: end while
13: return matchPair.
```

constructing DAG. Moreover, all edges in Q are used in CS, so finding the embeddings of Q in D is equivalent to finding the embeddings of Q in CS.

Theorem 1. *Q is converted into the DAG Q_d. Q_d is a tree, and node types of each layer are the same. Component and net nodes appear alternately in the layers of Q_d.*

Proof 1. *Since there is no circle in Q_d, each node has at most one parent node. Besides, DAG has a root, so Q_d is a tree. When constructing DAG, we traverse the bigraph Q with breadth-first search, so the node types in the same layer of the DAG tree are the same, and the node types in adjacent layers are different.* □

3.2.2 ***Cut Edge-based Graph Diversity Tolerance Handling.*** In the actual PCB instances, there may be slight differences between the template and the part of the data graph that should be matched. These differences are expected to be tolerated to reuse placement without being limited to the exact conditions. Therefore, we develop the graph diversity tolerance strategy, whose time complexity is $O(|E_q| \times |E_d|)$. *Suppose an edge $e = (u, v)$ in Q. In D, if there is no edge connecting two nodes in $C(u)$ and $C(v)$, then e is called the cut edge.* We find and delete all cut edges in Q during CS constructing to avoid excessive candidate filtering, so that the query pattern can be flexibly transformed into the same patterns as that in D for matching.

3.3 Subgraph Matching

As a pre-process of query graph, the leaf decomposition strategy [1] is adopted to accelerate the matching process.

3.3.1 ***Hierarchical Matching with Improved Backtracking.*** The divided connected subgraphs are denoted as $Q_i (i = 1, 2, ...)$. Noted that the graph is bipartite and no circle exists, we hierarchically search for embeddings of Q_i in CS according to the BFS tree. In this case, the parent of a node can only exist in former layer in the BFS tree, so we can achieve efficient layer-by-layer matching of Q_i, instead of node-by-node. The subgraph hierarchical matching is summarized in Algorithm 1. Whether node u can be matched is determined in *NodeCheck* function in line6. If u can find a match, the new matching pair is added to *matchPair*. Otherwise, the matching process is aborted.

In hierarchical match, it is possible that some query nodes can not match because no candidates meet the matching conditions, which results from the wrong matched pairs in embeddings. **We employ the branch backtracking to solve this problem.** The backtracking of other algorithms such as DAF [3] usually employs the recursive method, trying all possible paths until the matching is correct. While our method uses a loop instead of recursion. When the matching process is aborted, the matching pairs of the ancestor nodes are replaced in turn until the matching can proceed smoothly. We only select the path that may have errors for backtracking, which can save a lot of time.

If the correct match cannot be found even after backtracking to root, it is considered that some correct matched nodes are occupied. **In this case, *matched node snatching rule* should be adopted to solve the problem.** The core idea is to forcibly grab the occupied ideal match nodes. Only the nodes that meet the following conditions can forcibly grab matched nodes: (1) The candidate set is not empty; (2) At least one candidate has been matched by the embedded nodes. We expect the correct match to be found by branch backtracking and node snatching. Otherwise, unmatched node is skipped.

3.3.2 ***Multi-level Leaf Matching.*** Multi-level leaf nodes are separated from the DAG and matched individually after hierarchical matching, significantly reducing the time cost. Note that each node should be matched with the same number of candidates as its NEC nodes, so as to ensure that each multi-level leaf node in the restored real template is matched.

Table 1: The details of the cases and the comparison in matching accuracy and running time

Cases	Statistics				Accuracy		Runtime	
	#C_Q	#N_Q	#C_D	#N_D	VF3 [2](%)	D2BS(%)	VF3 [2](s)	D2BS(s)
case1	20	8	21	12	100	100	0.001	0.217
case2	52	31	79	65	100	100	0.001	2.222
case3	790	641	805	772	—	100	>3600	6.191
case4	546	182	1125	353	—	100	>3600	1.412
case5	237	102	418	187	100	100	48.221	2.419
case6	525	306	791	649	—	100	>3600	3.262
case7	481	153	481	173	—	100	>3600	2.150
case8	116	79	322	218	0	100	0	1.384
case9	341	169	406	285	0	100	0	2.234
case10	458	274	476	293	0	100	0	4.283

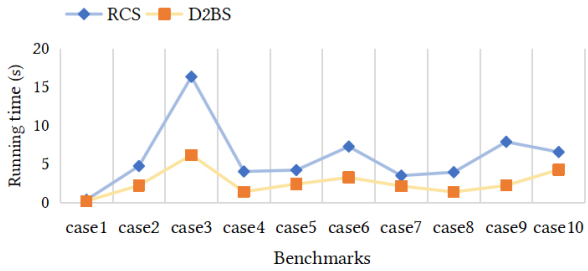

Figure 2: The comparison of time cost between RCS and D2BS.

4 EXPERIMENTAL RESULTS

In this paper, we chose ten netlist cases as the benchmarks, taken from industrial PCB designs. The statistics are shown in Table 1, where $\#C$ denotes the number of components, $\#N$ denotes the number of nets. These benchmarks consist of three types: small-scale cases (cases1–2), medium-or-large-scale cases(cases3–7), and the cases with structural differences (cases8–10).

In the first experiment, we compared the accuracy and runtime of our D2BS and the VF3 algorithm. Accuracy is the ratio of the matched query node number in total query nodes. As shown in Table 1, the D2BS algorithm achieves 100% matching accuracy for all benchmarks while VF3 does not. For case5, the runtime of VF3 is much longer than that of our D2BS. For other medium-or-large-scale cases, VF3 ran for hours and could not obtain any results, which is abnormal and unreasonable. For case8 to case10, the accuracy of the VF3 is zero because VF3 is an exact graph matching algorithm. When the structures of Q and D are different, VF3 has no matching results. In contrast, our D2BS algorithm applies the graph diversity tolerance strategy to expand the matching flexibility. Therefore, the matching results of case8 to case10 justify the superiority of the graph diversity tolerance strategy.

The second experiment compares the runtimes between RCS and D2BS, where RCS is our subgraph matching algorithm using recursions instead of hierarchical matching. Figure 2 shows the runtimes of the two algorithms. From the figure, the RCS algorithm can find correct matching results within 16 seconds on all benchmarks while the D2BS algorithm only takes less than 7 seconds, reducing the runtime by 53.87%. Therefore, this experiment proves the superiority of hierarchical match in backtracking.

REFERENCES

[1] F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang. Efficient subgraph matching by postponing cartesian products. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1199–1214, 2016.

[2] V. Carletti, P. Foggia, A. Saggese, and M. Vento. Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with vf3. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):804–818, 2017.

[3] M. Han, H. Kim, G. Gu, K. Park, and W.-S. Han. Efficient subgraph matching: Harmonizing dynamic programming, adaptive matching order, and failing set together. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1429–1446, 2019.

[4] W.-S. Han, J. Lee, and J.-H. Lee. Turboiso: towards ultrafast and robust subgraph isomorphism search in large graph databases. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 337–348, 2013.