SubHunter: A High-Performance and Scalable Sub-Circuit Recognition Method with Prüfer-Encoding

Hong-Yan Su, Chih-Hao Hsu and Yih-Lang Li
Institute of Computer Science and Engineering, National Chiao Tung University, Taiwan

***Abstract*—Sub-circuit recognition (SR) is a problem of recognizing sub-circuits within a given circuit and is a fundamental component in simulation, verification and testing of computer-aided design. The SR problem can be formulated as subgraph isomorphism problem. Performance of previous works is not scalable as the complexities of modern designs increase. In this paper we propose a novel Prüfer-encoding based SR algorithm that performs scalable and high–performance sub-circuit matching. Several techniques including tree structure partition, tree cutting and circuit graph encoding are proposed herein to decompose the SR problem into several small sub-sequence matching problems. A pre-filtering strategy is applied before matching to remove the sub-circuits that are not likely to be matched. A fast branch and bound approach is developed to identify all the sub-circuits within the given circuit. Experimental results show that SubHunter can achieve better performance than SubGemini and detect all the sub-circuits as well. As the circuit size increases, we can also achieve near linear runtime growth that outperforms the exponential growth for SubGemini, showing the scalability of the proposed algorithm.**

Keywords*—Sub-circuit recognition, graph isomorphism, Prüfer encoding

I. Introduction

Sub-circuit instance Recognition (SR) in a large circuit is a fundamental component and widely adopted in the field of computer-aided design (CAD), including circuit simulation, verification and testing. For example a divide and conquer approach can be applied on a complex design by identifying sub-circuits and replacing them with the corresponding components [8]. Therefore an effective SR algorithm can develop the performance of these operations. Traditionally the SR problem can be formulated as a subgraph isomorphism problem which is a widely known as a NP-Complete problem. For modern large circuits, traditional SR algorithms tend to degrade dramatically.

Traditional SR algorithms can be roughly categorized into two types: search-oriented subgraph isomorphism methods [1]-[7] and mathematical approaches [8][9]. Both of them formulate the SR problem into a subgraph isomorphism problem by constructing graphs for given circuits. Therefore identifying sub-circuits in the main circuit is equivalent to find isomorphic subgraphs within a graph. Several strategies have been proposed to overcome performance barrier and vertex labeling is a widely adopted strategy among them. Given a main graph and a sub-graph planned to be identified in the main graph, search-oriented algorithms employ breadth-first-search (BFS) or depth-first search (DFS) algorithm to traverse the main graph, and identify identical subgraphs during traversal. These approaches tend to investigate all the subgraphs and then require diminishing searching space for performance improvement. Vertex labeling is one of the strategies used to prevent unnecessary traversal. However, as the sizes of sub-circuits and main circuit increase, the runtime for these approaches also increases dramatically. In contrast to search-oriented methods, mathematical approaches formulate the SR problem into an optimization problem. A novel vertex labeling is proposed to give discriminative labels on the vertices. The SR problem can then be solved through minimizing the proposed objective function. Therefore the vertex labeling techniques of mathematical approaches determine its accuracy and runtime.

In this work we propose a high-performance and scalable SR algorithm based on Prüfer encoding method, named as SubHunter. The algorithm first encodes each labeled tree graph as a sequence of numbers and then the SR problem is converted into a sub-sequence matching problem. Since a graph representing a circuit is not necessarily a tree, a tree structure partition process is introduced herein to break a cyclic graph into a forest. A new vertex labeling technique is proposed with respect to the partition process. In order to further reduce the searching complexity, we propose several pruning strategies to avoid choosing unnecessary candidate sequences for matching. Compared to the state-of-the-art work in search-oriented methods, SubGemini [3], experimental results show that SubHunter can achieve dramatic runtime improvement and much better scalability.

The remainder of this paper is organized as follows. Section II introduces preliminary of this work. The proposed Prüfer encoding based circuit encoding method is described in section III. In section IV we illustrate how the proposed encoding method is applied to the SR problem. Then section V presents the experimental results and section VI draws the conclusions.

II. Preliminary

A. Previous Works

Search-oriented based algorithms are commonly adopted to solve the SR problem. The works in [4]-[7] only process the circuits consisting of NAND gates in their experiments, making their applications inclined to be limited. SubGemini firstly represents a circuit as an undirected bipartite graph with two sets of devices and nets (wires). Each vertex in the graph is assigned a label according to their function or degrees. After constructing the circuit graph model, SubGemini solves the SR problem in two phases. In phase I SubGemini locates a key vertex in the sub-graph and a candidate vector of vertices in the main graph. In phase II, SubGemini exploring each vertex in the candidate vector using relabeling procedure. It identifies a sub-circuit when every explored vertex label is identical to its associated vertex in sub-graph.

978-3-9815370-4-8/DATE15/©2015 EDAA

B. Prüfer Encoding

Prüfer encoding is a well-known encoding method in graph theory that provides a bijection mapping between a tree and its associated encoded Prüfer sequences. For an n-vertex labeled tree with distinct labels from 1 to n, the associated Prüfer code sequence is a sequence of numbers of length n-2. Prüfer encoding iteratively removes from the tree a 1-valent vertex with the lowest numerical label and its connected edge, reports the label of the vertex adjoining the removed vertex, and finally terminates when only two vertices remain.

The Prüfer encoding procedure for a tree takes only linear time and is easy to implement. Two isomorphic trees with the same labels are naturally encoded as the same Prüfer code sequence. These properties provide the capability of developing efficient algorithm to identify the isomorphism of two trees by checking the identity of their corresponding Prüfer code sequences.

III. Encoding Circuit Code Sequence

In this section we will present the procedures of constructing a circuit graph for a given circuit, partitioning the circuit graph to a forest and producing the corresponding circuit code sequences using the proposed Prüfer encoding algorithm.

A. Circuit Graph Construction

The circuit graph model adopted in this work to construct a circuit graph is similar to that used in SubGemini [3].

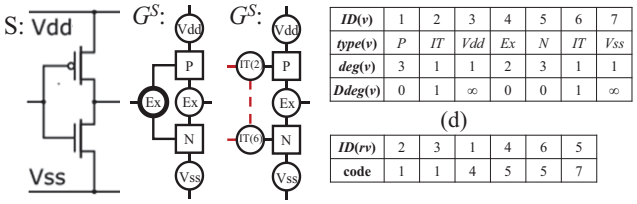
Def. Circuit graph $G = (V, E)$ is an undirected bipartite graph with $V = \{V_D \cup V_W\}$ where each vertex in V_D represents a device and that in V_W for a net (wire). Two vertices are connected by an edge $e = (v_1, v_2) \in E$ if $v_1 \in V_D$, $v_2 \in V_W$ and the corresponding device and wire are connected in the original circuit.

Fig. 1 shows an example of circuit graph where Fig. 1(a) is the original circuits and Fig. 1(b) presents the constructed circuit graphs where device vertices are represented as squares while net vertices are circles. We called the constructed circuit graphs as sub-graph $G^S = (V^S, E^S)$ and main-graph $G^M = (V^M, E^M)$ with respect to sub-circuit and main-circuit.

Each vertex v has a type, $type(v)$, that is a device or net. If $v \in V_D$, $type(v)$ is distinguished by the device functions. For example, if v represents a transistor, $type(v) = P$ or N. If $v \in V_W$, $type(v) \in \{In, Ex, Vdd, Vss\}$, in which four symbols represent internal, external, power, and ground respectively. A vertex type of *In* means that all device vertices that connect with the vertex are in the graph while that of *Ex* means the vertex connections link with other graph.A vertex v has another labels such as $ID(v)$ and $deg(v)$ to represent a unique stochastic number from 1 to $|V^S| + |V^M|$ and the degree of v respectively.

B. Tree Structure Partition

The constructed circuit graph may be cyclic, invalidating the process of Prüfer encoding. Randomly breaking cycle may generate different tree topologies; in other words, the same cycles in G^S and G^M may be turned into two non-isomorphic acyclic graphs. To avoid this, consistency has to be assured after applying any operation on the graphs. Notably, V_W can be classified into three subsets V_{W1}, V_{W2}, and V_{W3}. For a vertex $v \in V_W$, let N_v be the vertex set which contains every vertex $v' \in V_D$ that connects with v.



(a) (b) (c)

ID(v)	1	2	3	4	5	6	7
type(v)	P	IT	Vdd	Ex	N	IT	Vss
deg(v)	3	1	1	2	3	1	1
Ddeg(v)	0	1	∞	0	0	1	∞

(d)

ID(rv)	2	3	1	4	6	5
code	1	1	4	5	5	7

(e)

Figure 1. (a) Example of circuit; (b) graph representation of (a); (c) graph after tree structure partition; (d) labels of every vertex in (c); (e) associated circuit sequence of (c).

1. A vertex $v \in V_{W1}$ if v is the input of the circuit or power / ground net.
2. $v \in V_{W2}$ if v is an output of the circuit
3. $v \in V_{W3}$ for all the other vertices in V_W

According to this feature, we propose a cycle removal procedure that guarantees the consistency between G^S and G^M.

I. Each vertex $v_1 \in V_{W1}$ is separated to $|N_{v1}|$ vertices, each of which v_1^t is assigned with a new vertex type of *IT*, called *input terminal*, $ID(v_1^t)$ and $deg(v_1^t)$ that is a unique stochastic number from $|V^S| + |V^M| + 1$ to $|V^S| + |V^M| + |N_{v1}|$ and 1. Then each vertex pair (v_1^{ti}, v_1^{tj}), for $1 \le i, j \le |N_{v1}|$, is implicitly connected to represent that they are the same vertex [dashed lines are used to imply the implicit connection in Fig. 1(c)].

II. Each vertex $v_3 \in V_{W3}$ is firstly separated to two vertices v_{3I} and v_{3O} where $v_{3I} \in V_{W1}$ and $v_{3O} \in V_{W2}$. The vertex pair (v_{3I}, v_{3O}) is also implicitly connected. Then v_{3I} is split into several input terminal vertices like the methodology in step I.

III. If the graph is still cyclic, the net vertices in a cycle must be in V_{W2}. For each cycle c, each net vertex $v_2 \in V_{W2}$ in c is separated to $|N_{v2}|$ vertices v_2^t with the $type(v_2^t) = type(v_2)$ using the same concept as step I.

Generally some trees in G^M are much larger than those in G^S, which results in a very lengthy Prüfer encoded code sequence as well as large increase in the runtime of sequence matching. We propose a procedure, called tree cutting, to deal with big trees. Firstly, the largest degree d_s from V^S is identified and then a threshold value is obtained by multiplying d_s by a user-defined variable k. Then a vertex $v \in V^M$ will be split into d_m vertices using the same method as that in Step I if the degree of v (d_m) is bigger than $d_s \times k$. In this paper we choose k to be 15 through experiments. After conducting the above procedure, a new circuit graph $\hat{G}$ can be defined as follows.

Def. A circuit graph after partitioning $\hat{G} = (\hat{V}, \hat{E})$ consists of a set of trees T_r and each implicit connection describes the connection between two trees in the input circuit. Each vertex v has a five-tuple attribute $label(v) = (ID(v), type(v), deg(v), Ideg(v), TID(v))$ where $Ideg(v)$ is the number of implicit connections of v. $TID(v)$ is a number from 1 to $|T_r|$ to represent that v is in the

$TID(v)^{th}$ tree in $\hat{G}$. Let $E_I \subset \hat{E}$ be the set of all implicit connections in $\hat{E}$. E_I can be classified into two categories E_{DIT} and E_{DIN}. E_{DIT} is a set of intra implicit connections, in which edge $e = (v_1, v_2) \in E_{DIT}$ for $TID(v_1) = TID(v_2)$. E_{DIN} is a set of inter implicit connections, in which edge $e = (v_1, v_2) \in E_{DIN}$ for $TID(v_1) \neq TID(v_2)$.

Notably, the *Ideg* of a net vertex with type *Vdd* or *Vss* is set to infinity for matching since the matching of *Vdd* and *Vss* is little different from the others. We discuss the details in section IV. According to the graph definition above, we can define the isomorphism of the two partitioned circuit graphs $\widehat{G_1}$ and $\widehat{G_2}$ as follows.

Def. $\widehat{G_1}$ and $\widehat{G_2}$ are said to be *isomorphic* if there is a bijection $f: V(\widehat{G_1}) \rightarrow V(\widehat{G_2})$ such that $\forall\ uv \in E(\widehat{G_1})$, $f(u)f(v) \in E(\widehat{G_2})$, $label(u) = label(f(u))$ and $label(v) = label(f(v))$.

Lemma 1*: Let $\widehat{G^S}$ and $\widehat{G^M}$ be two forests produced by partitioning G^S and G^M into trees. G^S is isomorphic to SG^M if and only if $\widehat{G^S}$ is isomorphic to $\widehat{SG^M}$ where SG^M is a subgraph of G^M and $\widehat{SG^M}$ is the outcome of applying cycle removing procedure on SG^M.*

Lemma 1 enables identifying the isomorphism on the partitioned graphs directly. Since trees in $\hat{G}$ are connected through only implicit connections, $\hat{G}$ will become a forest if we remove E_I from $\hat{E}$. This dramatically decreases the matching complexity since finding subtrees in a tree is much simpler than that in a general graph.

C. Circuit Sequence Generation with Prüfer Encoding

Prüfer encoding algorithm encodes an *n*-vertex tree as a sequence of numbers of length *n*-2. Since each iteration can be treated as encoding an edge of the tree to a number, the resultant Prüfer code preserves the connection properties of the original tree. On the other hand, in addition to the labels on a labeled tree, a circuit graph contains supplementary circuit information. We slightly modify Prüfer encoding to preserve more information, and call the corresponding sequence *circuit code sequence* (*cs*). Firstly we modify Prüfer encoding to be *n*-1 iterations other than *n*-2 iterations for recording all edge connections. Let v_r be the vertex to be removed in current iteration. During each iteration we report not only the adjoining vertex ID of v_r but also the ID of v_r itself. Fig. 1(d) displays the table storing circuit information in Fig. 1(c) with a number on each vertex denoting each vertex's *ID* and Fig. 1(e) presents the corresponding circuit code sequence in Fig. 1(c).

Each tree *i* is encoded as a code sequence, and thus a forest will become a set of code sequences after Prüfer encoding. Let CS^S and CS^M be the sets of code sequences of sub-circuit and main circuit respectively. The SR problem is then transformed to be the code sequence matching problem. The objective of SR problem is to find all the subsets $SCS^M \subseteq CS^M$ that are isomorphic to CS^S.

IV. Code Sequence Matching

In this section we will present how to find bijective mapping between CS^S and SCS^M and the bijective function between every pair of cs^S and scs^M where $cs^S \in CS^S$ and $cs^M \in SCS^M$. Since the size of CS^M is greatly larger than CS^S, we will firstly present a pre-filtering technique to find candidate cs^M in section IV.A. Then in section IV.B we will show how to find the function f for a given pair of cs^S and cs^M.

A. Candidate Finding

In the code sequence generated by the proposed method, each column precisely describes an edge connection of the original acyclic graph, and can be represented as a vertex pair (v_1, v_2) comprising a device vertex v_1 and a net vertex v_2. Resolving code sequence matching problem requires identifying a vertex bijective mapping between main circuit and sub-circuit. A naïve approach is to enumerate all possible mappings from cs^M onto cs^S. However most of the mappings are illegal due to the mismatching on vertex type. A pre-filtering technique is thus applied to avoid unnecessary exploration in illegal mappings.

For vertex mapping, we consider two ending vertices of an edge at a time. A vertex pair $(v_1{}^M, v_2{}^M) \in cs^M$ is a candidate of a vertex pair $(v_1{}^S, v_2{}^S) \in cs^S$ if it satisfies following rules:

1. $type(v_1{}^M) = type(v_1{}^S)$, $deg(v_1{}^M) = deg(v_1{}^S)$, and $Ideg(v_1{}^M) = Ideg(v_1{}^S)$.
2. The type of $v_2{}^S$ can be in one of three categories:

case i. $type(v_2{}^S) = In$, *Vdd* or *Vss*: $type(v_2{}^M) = type(v_2{}^S)$, $deg(v_2{}^M) = deg(v_2{}^S)$ and $Ideg(v_2{}^M) = Ideg(v_2{}^S)$;

case ii. $type(v_2{}^S) = IT$: $type(v_2{}^M) = IT$, $deg(v_2{}^M) = deg(v_2{}^S)$ and $Ideg(v_2{}^M) \geq Ideg(v_2{}^S)$;

case iii. $type(v_2{}^S) = Ex$: $type(v_2{}^M) = Ex$, *In*, *IT*, *Vdd*, or *Vss*, $deg(v_2{}^M) \geq deg(v_2{}^S)$, and $Ideg(v_2{}^M) \geq Ideg(v_2{}^S)$.

For device vertices, rule 1 conforms to the consistency of *type*, *deg*, and *Ideg* between $v_1{}^M$ and $v_1{}^S$. However, the type and degree of a net vertex may alter depending on the place it connects to, requiring a different mapping policy for net vertices. These mappings can be classified into three possible mappings based on the types of net verties in $\widehat{G^S}$ and are stated in rule 2. When the type of $v_2{}^S$ is *In*, *Vdd* or *Vss*, the *type*, *deg* and *Ideg* of $v_2{}^M$ should be the same as those of $v_2{}^S$ (case i). If the type of $v_2{}^S$ is *IT*, $v_2{}^S$ is a separated vertex during tree structure partition. Therefore $v_2{}^M$ should have the type of *IT* and identical degree to $v_2{}^S$. However the main graph may have more separated vertices than that in the subgraph. Hence the number of implicit connection edges of $v_2{}^M$ can be greater than $v_2{}^S$ in this situation (case ii). When the type of $v_2{}^S$ is *Ex*, it represents that this net connects with some devices outside the subgraph. Thus the type of $v_2{}^M$ can be any of all the possible types of which a net vertex could be in, which is mainly decided by the kind of net vertex that $v_2{}^M$ connects to. The degree and the number of implicit connection edges of $v_2{}^M$ then should not be less than that of $v_2{}^S$ (case iii). With the aid of these rules, a vast number of mappings can thus be expelled.

B. Bounded Sequence Matching

After applying the proposed pre-filtering technique, the number of remaining mappings is still substantial since devices are primitive components in a chip and thus the device number is considerable. We solve this problem using the proposed

bounded sequence matching combining the methods of the branch and bound and breadth first search (BFS) algorithms.

A vertex pair is referred to as two vertices connected with an edge herein. Firstly we create an empty bijective mapping table T^M, and the algorithm chooses a vertex pair candidate from the associated candidate list to match the currently processed vertex pair in cs^S, denoted as *cp*. When the mapping from a *cp* in cs^S to a cp' in cs^M is selected, we then record the corresponding *ID* mappings in T^M. The matching starts from a *cp*, chooses a candidate from its candidate list, records the mapping information, moves to next *cp*, and repeats the same process until all the possible mappings have been examined. Once an inconsistent mapping happens during the process, this mapping branch is terminated and the trials on another mapping candidates are conducted. At the same time the connections of intra implicit connection edges of vertices in *cp* can also be used to prune illegal mappings by considering the property of identical connections of intra implicit connection edges for two vertices with mapping relation in the bijective function. Finally if a legal T^M is successfully identified, a sub-sequence of cs^M is identified to match cs^S. In other words, a subtree of the main graph that is isomorphic to the sub-graph is identified.

The matching procedure above can find all legal bijective functions between given cs^S and cs^M. However G^S may have several code sequences. Each $cs^S \in CS^S$ should match with a $scs^M \in CS^M$. Therefor we check the equivalence of inter implicit connections between cs^S and corresponding scs^M and then we can identify the isomorphism between a subgraph SG^M and G^S.

V. Experimental Result

In this section we present a set of experiments to show that our algorithm is promising in high performance and scalability. The source code and binary of the work in [13] is not available after contacting with authors. Thus the experiments are conducted only considering SubGemini. We implement SubHunter in C++ on a quad-core 2.4GHz Xeon-based Linux server with 82GB memory. SubGemini is run on the same machine.

In the experiments, a random generated case and a two RAM arrays are used. The random generated case is generated through randomly duplicating four components, consisting of a buffer, a register and two random sub-circuits, with random connections for all components. Two random sub-circuits are generated through repeatedly and randomly choose a gate out of a set of gates, including *nor*, *nand*, *or*, *inv*, *buf*, *RAM_cell*, and *reg*, with random connections for all components. A RAM array is formed by a set of RAM cells and a RAM cell contains two inverters. We summarize the results in Table I, where *Size*, *#N* and *#NF* are the gate count of the case, number of sub-circuits in the main circuit and number of found sub-circuits respectively. All the runtime are expressed in seconds. As shows in Table I, SubHunter can identify all the sub-circuits in the main circuit while SubGemini fails in one case and misses some sub-circuits in another case. On the other hand, the runtime of SubHunter mainly depends on the main circuit size and then on sub-circuit size. It is approximately linear to main circuit size or sub-circuit size. In contrast, the runtime of SubGemini increases dramatically when main circuit size increases.

TABLE I. Comparisons between SubGemini and SubHunter using random generated case and RAM array.

Main Circuit		Sub-Circuit			SubGemini		SubHunter	
Name	Size	Name	Size	*#N*	*#NF*	Time	*#NF*	Time
Random Case	2,394,822	Buffer	4	104,224	104,104	9.85	104,224	7.7
		Register	8	52,112	-	-	52,112	8.4
		Random circuit 1	24	26,268	26,268	17.05	26,268	12.1
		Random circuit 2	32	25,726	25,726	22.18	25,726	13.6
RAM Array	600 K	Inverter	2	200 K	200 K	5.17	200 K	2.0
		RAM cell	6	100 K	100 K	290	100 K	2.4
	6 M	Inverter	2	2 M	2 M	51.44	2 M	22.0
		RAM cell	6	1 M	1 M	40,918	1 M	25.8

VI. Conclusion

The sub-circuit recognition problem is an important component for CAD. We propose a high-performance and scalable SR algorithm called SubHunter. SubHunter adopts the concept of Prüfer encoding and combine several new techeniques including tree structure partition, circuitg graph encoding and bounded sequence matching. It transforms traditional SR problem into sub-sequence matching problems. Experimental results show that SubHunter can find all sub-circuits much faster than SubGemini and has near linear runtime growth rate as circuit size increases, making SubHunter practical for real application in modern designs.

References

[1] J. Ullman, "An Algorithm for Subgraph Isomorphism," *In J. Assoc. Computing Machinery*, vol. 23, no. 1, pp. 31–42, 1976.

[2] B. Messmer and H. Bunke, "Efficient Subgraph Isomorphism Detection: A Decomposition Approach," *In, Proc. IEEE Trans., Knowledge Data Engineering*, vol. 12, no. 2, pp. 307–323, 2000.

[3] Miles Ohlrich, Carl Ebeling, Eka Gintingt, and Lisa Sather, "SubGemini: Identifying SubCircuits using a Fast Subgraph Isomorphism Algorithm," In *Proc. Design Automation Conference*, 1993.

[4] N.'Vijaykrishnan and N. Ranganathan, "SUBGEN: A Genetic Approach for Subcircuit Extraction," *In International Conference on VLSl Design*, 1996.

[5] Wei-Hsin Chang, Shuenn-Der Tzeng, and Chen-yi Lee, "A Novel Extraction Algorithm By Recursive Identification Scheme," *In IEEE International Symposium on Circuits and Systems*, 2001.

[6] N. Zhang and D. C. Wunsch, II, "A fuzzy attributed graph approach to subcircuit extraction problem," *In Proc. IEEE Int. Conf. Fuzzy Systems*, pp. 1063–1067, 2003.

[7] Nian Zhang and Donald C. Wunsch II, "A Novel Subcircuit Extraction Algorithm Using Heuristic Dynamic Programming," *In International Conference on VLSI*, 2002.

[8] Nikolay Rubanov, "SubIslands: The Probabilistic Match Assignment Algorithm for Subcircuit Recognition", *In, Proc. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 1, pp. 26-38, 2003.

[9] Nikolay Rubanov, "A High Performance Subcircuit Recognition Method Based on the Nonlinear Graph Optimization", *In, Proc. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 11, pp. 2353-2363, 2006.