01 Jan 2006

Speeding Up VLSI Layout Verification Using Fuzzy Attributed Graphs Approach

Nian Zhang

Donald C. Wunsch
Missouri University of Science and Technology, dwunsch@mst.edu

Recommended Citation

N. Zhang and D. C. Wunsch, "Speeding Up VLSI Layout Verification Using Fuzzy Attributed Graphs Approach," *IEEE Transactions on Fuzzy Systems*, Institute of Electrical and Electronics Engineers (IEEE), Jan 2006.
The definitive version is available at https://doi.org/10.1109/TFUZZ.2006.877358

Speeding up VLSI Layout Verification Using Fuzzy Attributed Graphs Approach

Nian Zhang, *Member, IEEE*, and Donald C. Wunsch II, *Fellow, IEEE*

***Abstract*—Technical and economic factors have caused the field of physical design automation to receive increasing attention and commercialization. The steady down-scaling of complementary metal oxide semiconductor (CMOS) device dimensions has been the main stimulus to the growth of microelectronics and computer-aided very large scale integration (VLSI) design. The more an Integrated Circuit (IC) is scaled, the higher its packing density becomes. For example, in 2006 Intel's 65-nm process technology for high performance microprocessor has a reduced gate length of 35 nanometers. In their 70-Mbit SRAM chip, there are up to 0.5 billion transistors in a 110 mm^2 chip size with 3.4 GHz clock speed. New technology generations come out every two years and provide an approximate 0.7 times transistor size reduction as predicted by Moore's Law. For the ultimate scaled MOSFET beyond 2015 or so, the transistor gate length is projected to be 10 nm and below. The continually increasing size of chips, measured in either area or number of transistors, and the wasted investment involving fabricating and testing faulty circuits, make layout analysis an important part of physical design automation. Layout-versus-schematic (LVS) is one of three kinds of layout analysis tools. Subcircuit extraction is the key problem to be solved in LVS. In LVS, two factors are important. One is run time, the other is identification correctness. This has created a need for computational intelligence. Fuzzy attributed graph is not only widely used in the fields of image understanding and pattern recognition, it is also useful to the fuzzy graph matching problem. Since the subcircuit extraction problem is a special case of a general-interest problem known as subgraph isomorphism, fuzzy attributed graphs are first effectively applied to the subgraph isomorphism problem. Then we provide an efficient fuzzy attributed graph algorithm based on the solution to subgraph isomorphism for the subcircuit extraction problem. Similarity measurement makes a significant contribution to evaluate the equivalence of two circuit graphs. To evaluate its performance, we compare fuzzy attributed graph approach with the commercial software called SubGemini, and two of the fastest approaches called DECIDE and SubHDP. We are able to achieve up to 12 times faster performance than alternatives, without loss of accuracy.**

***Index Terms*—Fuzzy attributed graph, fuzzy logic, very large scale integration (VLSI).**

I. Introduction

In THE physical design automation of very large scale integration (VLSI) systems, we are concerned with layout analysis—recovering properties that the chip would have if it were manufactured as specified by a given layout. There are three kinds of layout analysis tools: first, design-rule checkers, which detect violations of rules that govern the technology in which the chip is to be made; second, netlist extractors and comparators, which tell the designer what circuit is expressed by a layout, for comparison to another representation of the circuit; and third, parameter extractors, which provide information about electrical characteristics that can be used for full-scale simulation [1]. This paper focuses on designing and implementing netlist extractors and comparators.

In these, we must perform LVS to ensure layout correctness. After the mask layout design of the circuit is completed, the LVS design will extract the circuit netlist description from the mask layout, and compare it with the original circuit description to see whether they are equivalent [2]. If they are equivalent, the design process will move to the post-layout simulation; otherwise the designer must go back to the mask layout to correct the errors, such as unintended connections between transistors, missing connections or missing devices [3]. LVS can be done in two steps: first, known as circuit extraction, converts the layout into a machine-readable network description; second, the extracted circuit has to be compared to a description of the original schematic [4]. In this paper, we designed and tested the algorithm for the second step.

One primary difficulty associated with an LVS process is caused by a dissimilarity in the labeling of devices and nets in the extracted schematic relative to the original schematic [5]. Designers are frequently confronted with different netlists representing the same design. For example, one netlist might be generated from a schematic representation of a circuit, while the other is produced by an extraction program from a physical layout of that circuit. Inevitably, the two netlists employ different names for the nets and devices of the circuit and list the objects in different orders. The most efficient way to overcome these difficulties is to identify a related collection of interconnected primitive devices as an instance of a pattern circuit, which is usually called the *subcircuit extraction problem*. If the two netlists represent different circuits, the program will pinpoint the differences.

We can obtain either a hierarchical or a flattened netlist from a schematic. However, when doing LVS, the circuit extracted from the layout comes back as a flat netlist, requiring a flat-versus-flat comparison [6]. In the layout versus schematic, two factors are important. The first factor is the run time associated with circuit setup and identification. The second factor is identification correctness. An excellent algorithm should have a very good tradeoff between speed and accuracy.

Many specialized LVS algorithms have been devised, starting as early as 1983 [4]. Early algorithms rely on the specific characteristics of the technology or circuits being transformed and are not easily applied to different technologies or circuit types,

Manuscript received March 18, 2004; revised April 3, 2006. This work was supported by the National Science Foundation and the Mary K. Finley endowment.

N. Zhang is with the Department of Electrical and Computer Engineering, South Dakota School of Mines and Technology, Rapid City, SD 57701 USA (e-mail: nian.zhang@sdsmt.edu).

D. C. Wunsch II is with the Department of Electrical and Computer Engineering, University of Missouri-Rolla, Rolla, MO 65409 USA (e-mail: dwunsch@ece.umr.edu).

Digital Object Identifier 10.1109/TFUZZ.2006.877358

such as analog circuits [7]. Moreover, these techniques rely on assumptions about the subcircuits being extracted and do not generalize to allow arbitrary subcircuits to be found [8].

In 1993, graph theory was applied to the subcircuit extraction problem [9]. By treating the subcircuit extraction as the subgraph isomorphism problem that assumes nothing about the underlying circuits, we obtain a truly technology-independent solution. Technology-independence means the same algorithm can be used in many different contexts, including digital and analog circuits, metal-oxide-semiconductor (MOS) and bipolar technologies, and for circuits using varying levels of abstraction.

Ohlrich *et al.* were the first researchers to solve the subcircuit extraction problem based on a solution to subgraph isomorphism. Their algorithm has been implemented in commercial software called SubGemini. SubGemini works in two phases. In Phase I, it identifies all possible locations of the subcircuit in the main circuit. It does this by applying a partitioning algorithm to both the subcircuit and the main circuit, in order to choose a key vertex (K) in the subcircuit, and identify all possible vertices in the main circuit that might match the key vertex. This set of vertices is called the candidate vector (CV). Phase I acts as a filter that tries to reduce the number of instances that need to be checked; later, Phase II will check each instance in order to determine if it is part of a subcircuit. Because this algorithm replies on the assumption that the outputs of a subgraph are not connected to its inputs, thus this algorithm is not applicable to the shorted circuit.

More approaches were proposed [10]–[16]. Among these approaches, DECIDE [13] and SubHDP [14] approaches reported the fastest speed against the others. The DECIDE algorithm created by Chang *et al.* in 2001, adopts a recursive scheme to achieve the identification operation. In 2002, Wunsch and Zhang reported a neural networks-based heuristic dynamic programming (HDP) algorithm, called SubHDP, for subcircuit extraction. HDP is a type of approximate dynamic programming, discussed in great detail in the literature. See [17]–[20] for a small, but representative sample. The SubHDP approach took advantage of the high accuracy and speed of trained neural nets.

Recently, we proposed a successful fuzzy attributed graph approach on subcircuit extraction [21], which demonstrated the power of fuzzy logic for netlist comparison. In this paper, we thoroughly present an improved fuzzy attributed graph approach on the subcircuit extraction problem and show its superior performance to all published approaches we are aware of.

The rest of this paper is organized as follows: Section II provides the notation and definitions for fuzzy attributed graph. Section III described the subgraph isomorphism and the subcircuit extraction problem. Section IV first introduces the similarity of a fuzzy attributed graph pair, then describes the circuit setup approach, followed by the subcircuit identification process. Section V provides the experimental results. Section VI gives the time complexity analysis and conclusions.

II. Fuzzy Attributed Graph

Attributed graph was introduced by Tsai and Fu for pattern analysis [22]. It gives a straightforward representation of structural patterns. The vertices of the graph represent pattern primitives describing the pattern, while the arcs are the relations between these primitives. However, the pattern often possesses properties that are fuzzy in nature and it has been extended to include fuzzy information into the attributes.

Each vertex may take attributes from the set $Z = \{z_i \mid i = 1, 2, \ldots, I\}$. For each attribute z_i, it will take values from $S_i = \{s_{ij} \mid j = 1, 2, \ldots, J_i\}$. The set of all possible attribute-value pairs of the vertices is $Lv = \{(z_i, s_{ij}) \mid j = 1, \ldots, J_i; i = 1, \ldots I\}$. A valid pattern primitive is just a subset of Lv in which each attribute appears only once, and $\prod$ represents the set of all those valid pattern primitives. Thus, each vertex will be represented by an element of $\prod$.

Similarly, each arc may take attributes from the set $F = \{f_i \mid i = 1, \ldots, I'\}$ in which each f_i may take values from $T_i = \{t_{ij} \mid j = 1, \ldots, J_i'\}$. $La = \{(f_i, t_{ij}) \mid j = 1, \ldots, J_i'; i = 1, \ldots, I'\}$ denotes the set of all possible relational attribute value pairs. A valid relation is just a subset of La in which each attribute appears only once. The set of all those valid relations is denoted as ϕ.

Let N be a finite nonempty set of vertices and $E \subseteq N \times N$ a set of distinct ordered pairs of distinct elements in N.

Definition 1: An *attributed graph* over $L = (Lv, La)$, with an underlying graph structure $H = (N, E)$ is defined to be an ordered pair (V, A), where $V = (N, \sigma)$ is called an attributed vertex set and $A = (E, \partial)$ is called an attributed arc set. The mapping $\sigma : N \to \prod$ and $\partial : E \to \phi$ are called vertex interpreter and arc interpreter, respectively.

The vertex and arc interpreter is just a mapping that maps the vertices or arcs to their corresponding attribute sets. When using attributed graph to represent a CMOS circuit, a natural way is to represent each node as a vertex and the relation between nodes as arcs. Each node has an attribute of Node_type, and each relation has an attribute of Edge_relation

$$\begin{aligned}\text{Node\_ type} &= \{N\text{- type},\ P\text{- type},\ \text{terminal}\}\\ \text{Edge\_ relation} &= \{N\text{- type-terminal}\\ &\qquad P\text{- type- terminal},\ \text{terminal- terminal}\\ &\qquad N\text{- type-}N\text{- type},\ N\text{- type-}P\text{- type}\\ &\qquad P\text{- type-}P\text{- type}\}.\end{aligned}$$

With the definition of attributed graph given previously

$$\begin{aligned}Z &= \{z_1 = \text{ Node\_ type}\}\\ S_1 &= \{S_{11} = N\text{- type},\ S_{12} = P\text{- type},\ S_{13} = \text{ terminal}\}\\ L_v &= \{(\text{ Node\_ type},\ N\text{- type}),\ (\text{ Node\_ type},\ P\text{- type})\\ &\qquad (\text{ Node\_ type},\ \text{terminal})\}.\end{aligned}$$

Similarly

$$\begin{aligned}F &= \{f_1 = \text{ Edge\_ relation}\}\\ T_1 &= \{t_{11} = N\text{- type- terminal}\ t_{12} = P\text{- type- terminal}\\ &\qquad t_{13} = \text{ terminal- terminal}\ t_{14} = N\text{- type-}N\text{- type}\\ &\qquad t_{15} = N\text{- type-}P\text{- type}\ t_{16} = P\text{- type-}P\text{- type}\}\\ La &= \{(\text{ Edge\_ relation},\ N\text{- type- terminal})\\ &\qquad (\text{ Edge\_ relation},\ P\text{- type- terminal})\end{aligned}$$

$$
\begin{aligned}
&(\text{ Edge\_ relation, terminal- terminal})\\
&(\text{ Edge\_ relation, } N\text{- type-}N\text{- type})\\
&(\text{ Edge\_ relation, } N\text{- type-}P\text{- type})\\
&(\text{ Edge\_ relation, } P\text{- type-}P\text{- type})\}.
\end{aligned}
$$

The following equations are valid pattern primitives and relations:

$$
\begin{aligned}
\pi_1 &= \{(\text{ Node\_ type, } N\text{- type})\} \text{ and}\\
\theta_1 &= \{\text{ (Edge\_ relation, } N\text{- type- terminal})\}.
\end{aligned}
$$

The matching of vertices and arcs can be determined by equality of attributes.

Since every crisp set can be represented as a particular case of a fuzzy set by setting the membership to 0 or 1 for all elements, we can extend the definition to include fuzzy attributes.

In a fuzzy attributed graph, each vertex may take attributes from the set $Z = \{z_i \mid i = 1, 2, \ldots, I\}$ [23]. For each attribute z_i, it will take values from $S_i = \{s_{ij} \mid j = 1, 2, \ldots, J_i\}$. The set of all possible fuzzy attribute-value pairs is $Lv = \{(z_i, A_{S_i}) \mid i = 1, \ldots I\}$, where A_{S_i} is a fuzzy set on the attribute-value set S_i. A valid pattern primitive is just a subset of Lv in which each attribute appears only once, and $\prod$ represents the set of all those valid pattern primitives. Thus, each vertex will be represented by an element of $\prod$.

Similarly, each arc may take attributes from the set $F = \{f_i \mid i = 1, \ldots, I'\}$ in which each f_i may take values from $T_i = \{t_{ij} \mid j = 1, \ldots, J_i'\}$. And $La = \{(f_i, B_{T_i}) \mid i = 1, \ldots, I'\}$ denotes the set of all possible relational attribute value pairs, where B_{T_i} is a fuzzy set on the relational attribute value set T_i. A valid relation is just a subset of La in which each attribute appears only once. The set of all those valid relations is denoted as ϕ.

Definition 2: A *fuzzy attributed graph* over $L = (Lv, La)$, with an underlying graph structure $H = (N, E)$ is defined to be an ordered pair (V, A), where $V = (N, \sigma)$ is called a fuzzy vertex set and $A = (E, \partial)$ is called a fuzzy arc set. The mapping $\sigma : N \to \prod$ and $\partial : E \to \phi$ are called fuzzy vertex interpreter and fuzzy arc interpreter, respectively [24].

With the aforementioned definition and the assumption that the accuracy of the circuit extraction is 90%, we may have an N-type node and its possible relationship to another node represented as

$$
\begin{aligned}
\pi_1 = \Bigg\{\Bigg(&\text{ Node\_ type, } \Bigg\{\frac{0.9}{N\text{- type}}, \frac{0.1}{P\text{- type}}\\
&\frac{0.1}{\text{terminal}}\Bigg\}\Bigg)\Bigg\} \text{ and}\\
\theta_1 = \Bigg\{\Bigg(&\text{ Edge\_ relation, } \Bigg\{\frac{0.9}{N\text{- type- terminal}}\\
&\frac{0.1}{P\text{- type- terminal}}, \frac{0.1}{\text{terminal- terminal}}\\
&\frac{0.1}{N\text{- type-}N\text{- type}}, \frac{0.1}{N\text{- type-}P\text{- type}}\\
&\frac{0.1}{P\text{- type-}P\text{- type}}\Bigg\}\Bigg)\Bigg\}.
\end{aligned}
$$

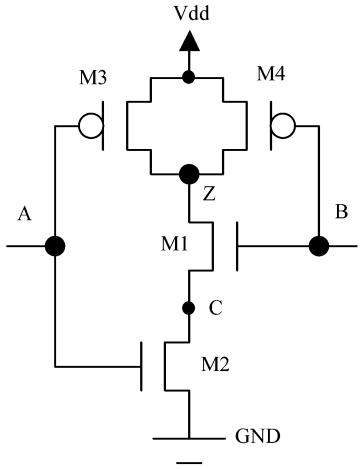

Fig. 1. Pattern circuit.

Definition 3: In pairing two fuzzy attributed graphs, nodes that are paired are named *core nodes*; nodes that are not paired but have branches directly connected to core nodes are named *goal nodes*, and the others are named *free nodes*.

III. Circuit Graph Representation

A. Subgraph Isomorphism

The problem of subcircuit extraction can be transformed to the subgraph isomorphism problem. A graph G consists of a finite nonempty set $V = V(G)$ of p points together with a prescribed set X of q unordered pairs of distinct points of V [25]. Each pair $x = \{u, v\}$ of points in X is a line of G, and x is said to join u and v. A subgraph of G is a graph having all of its points and lines in G. The subgraph isomorphism problem is defined as: *Given a graph S and another larger graph T, to find all the subgraphs of T which are identified with S.*

B. Subcircuit Extraction Problem

Similarly, the *subcircuit extraction problem* is to determine whether one given pattern circuit has any isomorphic circuits in the input circuit. A 2-input NAND gate serving as the pattern circuit is shown in Fig. 1. In Fig. 2, a netlist is shown as the input circuit (i.e., main circuit), in which we will find the instance of the pattern circuit. Our goal is to verify that the netlist composed of M8, M9, M10, and M11 in the input circuit is isomorphic to the pattern circuit.

A circuit graph contains two types of nodes: device and terminal. It is a *bipartite* (or *2-chromatic*) graph, in which device vertices connect to only terminal vertices, and terminal vertices connect only to device vertices. A device is represented by a square, while a terminal is represented by a circle. Therefore, we can convert the 2-input NAND circuit in Fig. 1 to a graph in Fig. 3(a). If we give each node a value, Fig. 3(a) can be transformed to Fig. 3(b). The integer values can be obtained from the circuit file, which we will introduce later.

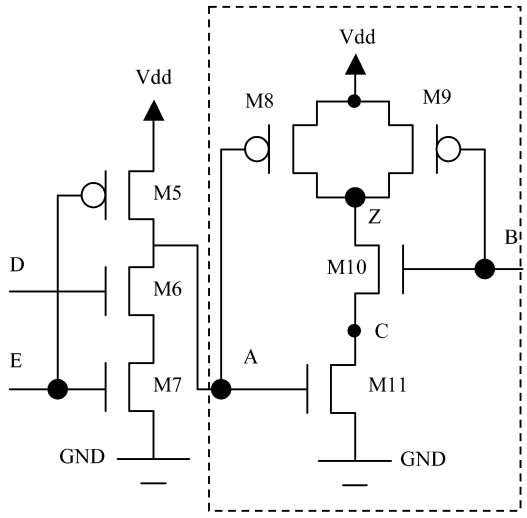

Fig. 2. Main circuit.

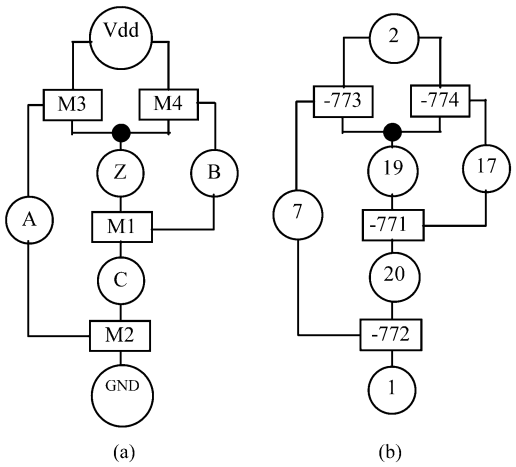

Fig. 3. (a) Graph representation of a 2-input NAND gate. A device is represented by a square, and a terminal is represented by a circle. (b) Coded graph representation of a 2-input NAND gate. A device has a negative integer value, and a terminal vertex has a positive integer value. These values are obtained from the circuit file.

IV. Fuzzy Attributed Graph Algorith on Subcircuit Extraction Problem

A. Similarity of a Fuzzy Attributed Graph Pair

We apply the definition of the similarity [26] to our subcircuit extraction problem. Assume we have a fuzzy attributed graph pair: one is the pattern circuit graph, G1; the other is a candidate subcircuit graph, G2. The ith vertex in G1 and G2 is denoted as n_x^i and n_y^i, respectively. Any node n_x^i and n_y^i can be represented by a vector of a_x^i and a_y^i, respectively. The edge between n_x^i and n_x^j in G1 can be represented by a vector of $e_x^{(i,j)}$, and the edge between n_y^i and n_y^j in G2 can be represented by a vector of $e_y^{(i,j)}$. Similarity between G1 and G2 is calculated with all the a_x^i, a_y^i, $e_x^{(i,j)}$ and $e_y^{(i,j)}$. The vertex in G1 and G2 can be expressed as $a_x^i = \left(\alpha_x^i[1], \alpha_x^i[2], \alpha_x^i[3]\right)^T$ and $a_y^i = \left(\alpha_y^i[1], \alpha_y^i[2], \alpha_y^i[3]\right)^T$, respectively, where $0 \le \alpha_x^i[s]$, $\alpha_y^i[s] \le 1$, ($s = 1, 2, 3$) are memberships of a component belonging to an *N-type device*, a *P-type device* and a *terminal*, respectively, and

$$e_x^{(i,j)} = \Big(\beta_x^{(i,j)}[1], \beta_x^{(i,j)}[2], \beta_x^{(i,j)}[3], \beta_x^{(i,j)}[4] \quad \beta_x^{(i,j)}[5], \beta_x^{(i,j)}[6]\Big)^T$$

with $0 \le \beta_x^{(i,j)}[t] \le 1$, ($t = 1, 2, 3, 4, 5, 6$) being memberships of an edge between N-type device and terminal, between P-type device and terminal, between terminal and terminal, between N-type device and N-type device, between N-type device and P-type device, and between P-type device and P-type, respectively.

If n_x^i is a core node in G1 and its pair node in G2 is $n_y^{r_i}$, a square distance between the two core nodes can be given by

$$\mathrm{DAC}^i = \frac{\sum_{s=1}^{3} \left(\alpha_x^i[s] - \alpha_y^{r_i}[s]\right)^2}{3} \tag{1}$$

where DAC^i is a unified distance ($0 \le \mathrm{DAC}^i \le 1$). For goal nodes and free nodes in G1 and G2, since there is no pair node, the corresponding square distances can be defined as distances to a zero vector as follows:

$$\mathrm{DAO}_x^j = \frac{\sum_{s=1}^{3} \left(\alpha_x^j[s]\right)^2}{3} \tag{2}$$

$$\mathrm{DAO}_y^j = \frac{\sum_{s=1}^{3} \left(\alpha_y^j[s]\right)^2}{3}. \tag{3}$$

A synthesis square distance caused by all nodes then can be

$$\mathrm{DE} = \frac{\sum_{i=1}^{I} \mathrm{DAC}^i + \sum_{j_1=1}^{J_1} \mathrm{DAO}_x^{j_1} + \sum_{j_2=1}^{J_2} \mathrm{DAO}_y^{j_2}}{I + J_1 + J_2} \tag{4}$$

where I is the number of core nodes, J1 the number of goal nodes and free nodes in G1, and J2 the number of goal nodes and free nodes in G2.

Similarly, if branches $e_x^{(i,j)}$ and $e_y^{(r_i,r_j)}$ are paired branches (i.e., nodes n_x^i and n_x^j are paired with $n_y^{r_i}$ and $n_y^{r_j}$, respectively), a square distance between two paired branches whose both end nodes are core nodes can be given as:

$$\mathrm{DEC}^{(i,j)} = \frac{\sum_{t=1}^{6} \left(\beta_x^{(i,j)}[t] - \beta_y^{(r_i,r_j)}[t]\right)^2}{6}. \tag{5}$$

If $\beta_x^{(i,j)}$ and $\beta_y^{(r_i,r_j)}$ is the kth branch pair, then $\mathrm{DEC}^{(i,j)}$ can also be represented as DEC^k. Thus, (5) can also be written as

$$\mathrm{DEC}^k = \frac{\sum_{t=1}^{6} \left(\beta_x^{(i,j)}[t] - \beta_y^{(r_i,r_j)}[t]\right)^2}{6}.$$

For other branches in G1 and G2, the corresponding square distances can be given by

$$\mathrm{DEO}_x^{l_1} = \frac{\sum_{t=1}^{6} \left(\beta_x^{l_1}[t]\right)^2}{6} \tag{6}$$

$$\mathrm{DEO}_y^{l_2} = \frac{\sum_{t=1}^{6} \left(\beta_y^{l_2}[t]\right)^2}{6}. \tag{7}$$

A synthesis square distance caused by all edges can also be given as:

$$\mathrm{DE} = \frac{\sum_{k=1}^{K} \mathrm{DEC}^k + \sum_{l_1=1}^{L_1} \mathrm{DEO}_x^{l_1} + \sum_{l_2=1}^{L_2} \mathrm{DEO}_y^{l_2}}{K + L_1 + L_2} \tag{8}$$

where K is the number of paired branches whose both end nodes are core nodes, L1 the number of other branches in G1, and L2 the number of other branches in G2.

It is clear that both DA and DE are unified. So a similarity measure between a fuzzy attributed graph pair can be defined directly with

$$\mathrm{SIM}(\mathrm{G1}, \mathrm{G2}) = 1 - \sqrt{\frac{w_a \cdot \mathrm{DE} + w_e \cdot \mathrm{DE}}{w_a + w_e}} \tag{9}$$

where w_a and w_e are properly selected parameters, which are both set to 0.5 in our algorithm.

To determine the subgraph isomorphism between the two fuzzy attributed graphs, we should measure how much one fuzzy attributed graph is a part of another. We denote the measurement as COM, shown in (10). Value "1" stands for a complete part.

$$\mathrm{COM} = \max\{\mathrm{SIM}(\mathrm{CORE}_X(\mathrm{CN}), \mathrm{G1}) \quad \mathrm{SIM}(\mathrm{CORE}_Y(\mathrm{CN}), \mathrm{G2})\} \tag{10}$$

where CN is the number of core nodes, and $\mathrm{CORE}_X(\mathrm{CN})$ and $\mathrm{CORE}_Y(\mathrm{CN})$ are two fuzzy attributed graphs, consisting of only the core nodes in G1 and G2, respectively. This notation provides us a good way to determine the similarity between the two graphs based on the percentage of the core nodes in each graph. That is, if all nodes of one of the two graphs become core nodes, then this graph is a subgraph of the larger graph; if all the nodes of the two graphs are core nodes, then the two graphs are equivalent.

B. Fuzzy Attributed Graph Algorithm on Subcircuit Extraction Problem

Given two fuzzy attributed graphs G1 and G2. G1 is the pattern circuit graph, and G2 is the candidate subcircuit graph. G1 has M nodes, which is represented as $n_x^i (i = 1 \ldots M)$, and G2 has N nodes, which is represented as $n_y^j (j = 1 \ldots N)$. Assume Vdd is the start node for both graphs. Our subcircuit extraction algorithm is partitioned into two parts: Circuit setup and subcircuit identification.

TABLE I
A 2-INPUT NAND GATE REPRESENTATION IN THE CIRCUIT FILE

Mxx	d	g	s	b	type
M1	19	17	20	1	N
M2	20	7	1	1	N
M3	19	7	2	2	P
M4	19	17	2	2	P

C. Circuit Setup

The objective is to analyze the circuit file and choose the candidate subcircuits. First, we read in the circuit file, and partition the circuit. The circuit files have the following format:

$$\mathrm{Mxx}\ d\ g\ s\ b\ \ \mathrm{type}$$

where Mxx is the name of the MOSFET device, d is the node connected to the Mxx's drain, g is the node connected to the Mxx's gate, s is the node connected to the Mxx's source, b is the node connected to the Mxx's bulk, and type denotes the device's type (i.e., N-type or P-type). For example, Table I is part of a circuit file, and it is actually a two-input NAND gate. "2" denotes the power (i.e., Vdd), and "1" denotes the ground (i.e., GND).

To make the subcircuit identification process easier, we represent the Mxx (i.e., device) as a negative integer to distinguish it from the terminal that is a positive integer. For example, since $M = 77$ in the ASCII character set, then expressed as a radix-10 integer, M1 becomes $-(77 * 10 + 1) = -771$. In the same manner, we convert all the transistors' names into negative integers.

Next, we partition the circuit. The circuit can be partitioned according to the appearance of "2" in the *source* column. We read each line from top to bottom. If a line has a "2" in the source columns, and the next line has a different value in the source column, the program will partition these two lines into different netlists.

Second, we choose those netlists containing four devices as candidate netlists. This is because a two-input NAND gate consists of four devices.

We now provide an example to make the whole procedure clear. Given a raw circuit file shown in Fig. 4(a), we choose the first five columns, and replace Mxx with a negative integer, and thus obtain a circuit as shown in Fig. 4(b).

The raw circuit is partitioned into six netlists, as shown in Fig. 5.

If the amount of the transistors in a netlist is four, the netlist will be considered a candidate netlist. Thus, there are three candidate netlists: The first, the third and the sixth netlists. Each netlist is equivalent to a circuit graph, as shown in Fig. 6. Although we can see that Fig. 6(a) and (b) are two-input NAND gates, and Fig. 6(c) is a two-input NOR gate, the computer does not know. We need to identify these NAND gates in the identification procedure later. Also, these three example graphs show that it is possible that different kinds of netlists can exist in the candidate netlists.

In addition, it is necessary to create a Hash table. The ith column stores all the information of a node whose value is i, in-

M1	19	17	20	1	N	-771	19	17	20	1
M2	20	7	1	1	N	-772	20	7	1	1
M3	19	7	2	2	P	-773	19	7	2	2
M4	19	17	2	2	P	-774	19	17	2	2
M5	9	19	1	1	N	-775	9	19	1	1
M6	9	19	2	2	P	-776	9	19	2	2
M7	21	5	22	1	N	-777	21	5	22	1
M8	22	6	1	1	N	-778	22	6	1	1
M9	21	6	2	2	P	-779	21	6	2	2
M10	21	5	2	2	P	-780	21	5	2	2
M11	17	21	1	1	N	-781	17	21	1	1
M12	17	21	2	2	P	-782	17	21	2	2
M13	10	23	1	1	N	-783	10	23	1	1
M14	10	23	2	2	P	-784	10	23	2	2
M15	23	7	1	1	N	-785	23	7	1	1
M16	23	18	1	1	N	-786	23	18	1	1
M17	23	7	24	2	P	-787	23	7	24	2
M18	24	18	2	2	P	-788	24	18	2	2

(a) (b)

Fig. 4. (a) Raw circuit file. (b) Circuit after data processing.

-771	19	17	20	1
-772	20	7	1	1
-773	19	7	2	2
-774	19	17	2	2
-775	9	19	1	1
-776	9	19	2	2
-777	21	5	22	1
-778	22	6	1	1
-779	21	6	2	2
-780	21	5	2	2
-781	17	21	1	1
-782	17	21	2	2
-783	10	23	1	1
-784	10	23	2	2
-785	23	7	1	1
-786	23	18	1	1
-787	23	7	24	2
-788	24	18	2	2

Fig. 5. Circuit has been partitioned into six netlists.

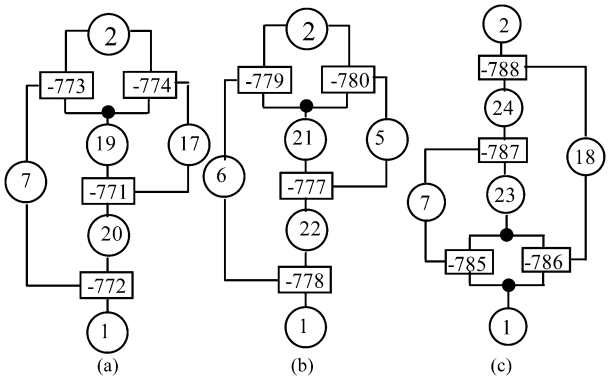

Fig. 6. Circuit graph for a netlist. (a) Circuit graph for the netlist consisting of devices -771, -772, -773, and -774. (b) Circuit graph for the netlist consisting of devices -777, -778, -779, and -780. (c) Circuit graph for the netlist consisting of devices -785, -786, -787, and -788.

cluding node type, its neighbors, and whether it has been visited before. It plays an important role in the identification procedure later, because of the following.

4
0
0
19
17
20

(a)

0
8
1
-771
-772
0

(b)

Fig. 7. (a) Column 771 in the Hash table. It stores all the attributes of device -771 in the circuit file. (b) Column 20 in the Hash table. It stores all the attributes of terminal 20 in the circuit file.

1) Since the Hash table tells us what a node's neighbors are, it is possible to find whether there is an edge between the core nodes and the element in the goal node set.
2) Since the Hash table tells us how many neighbors a node has, it is possible to find out how many edges there are from the core node set.
3) It helps to find whether an element has been explored.

The Hash table is established as follows: The ith column stores all the information of a vertex with value i. It has the format of $[$ type weight flag neighbor $1 \ldots$ neighbor $n]^T$. We assume the *type* value for a P-type device, N-type device, and a terminal is 3, 4, and 0, respectively. The weight for a terminal is defined as: Number of P-type neighbors $*$ 3 $+$ number of N-type neighbors $*$ 4; and a device's weight is always 0. If a vertex has been visited, then the flag is set to "1"; otherwise, it is "0". The length of the vector is extended to the length of Hash table with zeros. For example, according to Table I, an N-type device -771 has all of its information in column 771, as shown in Fig. 7(a). Assume it has not been visited. The first row is 4 because this node is an N-type device, whose type value is 4; the second row is 0 because a device's weight is always 0; the third row is 0 because it has not been visited; row 4 through row six are its three neighbors. Terminal 20 has all of its information in column 20, as shown in Fig. 7(b). Assume it has been visited. The first row is 0 because this node is a terminal, whose type value is 0; terminal 20 is connected to two N-type devices (-771 and -772), respectively. Thus, the second row is calculated as follows: $0 \times 3 + 2 \times 4 = 8$; the third row is 1 because this node has been visited; the fourth row and the fifth row are its two neighbors; we add zero to the last row because this vector is shorter than the length of the Hash table.

D. Subcircuit Identification

The objective is to identify the instances of the pattern circuit from the candidate subcircuits. We compare the fuzzy attributed graphs for pattern circuit (G1) and each candidate subcircuit (G2). Apparently, Vdd in G1 and Vdd in G2 is a core node pair, thus, we first put Vdd of G1 into the core node set, CNA, and put Vdd of G2 into the other core node set, CNB. Set the flag attribute of Vdd to "1." The following pseudocode, shown in Fig. 8 is to implement the subcircuit identification for each candidate subcircuit.

The maximum SIM of the above process is the similarity between G1 and G2. The corresponding core node pairs represent the optimal matching result.

TABLE II
EXPERIMENTAL RESULTS IN EXPERIMENT I

Testing files		Setup time (second)	Identification time (second)	Total time (second)	# of S found in G
Size of main circuit (G)	# of S in G				
400	4	0.039	0.0018	0.0408	4
1,000	4	0.0975	0.0018	0.0993	4
2,000	4	0.235	0.0018	0.2368	4
3,000	4	0.341	0.0018	0.3428	4
5,000	4	0.7218	0.0018	0.7236	4
10,000	4	1.407	0.0018	1.4088	4
20,000	4	2.874	0.0018	2.8758	4
50,000	4	7.135	0.0018	7.1368	4
100,000	4	14.66	0.0018	14.6618	4
500,000	4	76.27	0.0018	76.2718	4
1,000,000	4	147.8	0.0018	147.8018	4
5,000,000	4	735.4	0.0018	735.4018	4

Initialize the core node sets in G1 and G2 with a core node pair, Vdd in G1 and Vdd in G2.
Do
Find out the goal nodes connected to the current core nodes in G1 and G2.
For each goal node, *s* in G1
For each goal node, *t* in G2
If *s* and *t* satisfy the following conditions:
1. core nodes with branches to *s* and *t* are all paired
2. the number of goal nodes with branches to *s* and *t* are equal

then s and t is a new core node pair.
Compute the similarity of G1 and G2 according to (9).
While the similarity measurement is not equal to 1 and at least a new core node pair has been found after this iteration.
If the similarity is 1 and all the nodes of G1 and G2 are core nodes, then display that "This candidate subcircuit is an instance of the pattern circuit."

Fig. 8. Pseudocode for subcircuit identification using fuzzy attributed graph approach.

For example, given the G1 (pattern circuit) and G2 (candidate circuit) shown later, the only difference between G1 and G2 is the missing edge between device -825 and the node 32 in G2. The similarity between them is found to be 0.9.

Let us take another example. The similarity between the pattern circuit [Fig. 6(a)] and the candidate circuit [Fig. 6(c)] is only 0.7.

This makes sense because the candidate circuit in the first example is a 2-input NAND gate with a missing connection. Since the similarity between the two graphs is not 1, it indicates that the candidate circuit is not an instance of pattern circuit. However, since the similarity between G1 and G2 (i.e., 0.9) is above the threshold value (i.e., 0.8), it is possible that the candidate circuit is an instance of a 2-input NAND gate with some drawing errors. On the other hand, the similarity between Fig. 6(a) and (c) (i.e., 0.7) falls below the threshold value, thus the algorithm would say the candidate circuit is not a two-input NAND gate at all. This is true because the candidate circuit in the second example is actually a two-input NOR gate.

V. EXPERIMENTAL RESULTS

We implemented the proposed approach in Matlab on Sun Ultra Sparc II 440 MHz, 1-G RAM platform. Three experiments were carried out to investigate the algorithm performance.

A. Experiment I

The objective is to see how well the algorithm performs with increasing size of VLSI circuits. The number of true NAND gates remains the same. Experimental results are shown in Table II. G denotes the main circuit and S denotes the instance of the pattern circuit in the main circuit. Identification time is the time to find out all the subcircuits. Total time includes setup time and subcircuit identification time. Run time is measured in seconds. From Table II, it shows that the setup time increased proportionally to the size of the input file; however, the identification time did not change. Also, our algorithm can find all the NAND gates.

B. Experiment II

The objective is to see whether the number of true NAND gates in the circuit file has impact on the algorithm. The input circuits remain to be 5,000-transistor, but we increase the number of true NAND gates in it. Experimental results are shown in Table III. It shows that the identification time increased proportionally to the number of NAND gates, but the setup time didn't change. Also, it can find all the NAND gates.

C. Experiment III

The objective is to compare our fuzzy attributed graph algorithm with the commercial software called SubGemini and two of the fastest approaches called DECIDE and SubHDP, as shown in Table IV. DECIDE and SubHDP used the same platform as that used for this approach. However, SubGemini uses SUN 4/490 25 MHz, so we normalize the run time with a scalar factor, ratio of the two CPU rates. The four approaches use the same testing circuits.

In Table IV, when the main circuit has no more than 5,000 transistors, the DECIDE and SubHDP algorithms are faster than the fuzzy attributed graph approach. However, when the main circuit has much more than 5,000 transistors, the fuzzy attributed graph algorithm is faster.

We then plot the results in Table IV into Figs. 11 and 12. In Fig. 11, the circuits are no more than 100,000 transistors. The x-axis denotes the number of transistors, and the y-axis denotes the total run time. We can see that the fuzzy attributed graph approach has kept a fast run time all the way. The gap between the

TABLE III
EXPERIMENTAL RESULTS IN EXPERIMENT II

Testing files		Setup time (second)	Identification time (second)	Total time (second)	# of S found in G
Size of main circuit (G)	# of S in G				
5,000	4	0.7218	0.0018	0.7236	4
5,000	16	0.7218	0.0071	0.7289	16
5,000	24	0.7218	0.0102	0.732	24
5,000	64	0.7218	0.026	0.7478	64
5,000	132	0.7218	0.0524	0.7742	132
5,000	200	0.7218	0.0862	0.808	200
5,000	332	0.7218	0.153	0.8748	332

TABLE IV
COMPARISONS OF THE FOUR ALGORITHMS

Testing files		Setup time for Fuzzy (second)	Identification time for Fuzzy (second)	Total time (second)				# of S found in G
Size of main circuit (G)	# of S in G			Sub-Gemini	SubHDP	DECIDE	Fuzzy	
400	24	0.047	0.015	0.55	0.064	0.026	0.062	24
1,000	64	0.0962	0.031	1.38	0.081	0.1022	0.1272	64
2,000	132	0.2524	0.057	3.4	0.113	0.1249	0.3094	132
3,000	200	0.3462	0.089	4.98	0.166	0.148	0.4352	200
5,000	332	0.7218	0.153	8.3	0.353	0.2772	0.8748	332
10,000	664	1.347	0.298	21.85	1.104	10.9745	1.645	664
20,000	1332	2.74	0.589	43.7	3.546	20.574	3.329	1332
50,000	3332	7.2226	1.49	87.4	20.488	80.44	8.7126	3332
100,000	6664	15.682	3.01	174.3	85.442	230.471	18.692	6664
500,000	33332	76.9866	15.7781	892.7	473.56	1157.4	92.7647	33332
1,000,000	66664	156.82	30.5362	1758	860.4	2320.7	187.3562	66664
5,000,000	333332	789.74	151.2355	8735	4170.1	11520.5	940.9755	333332

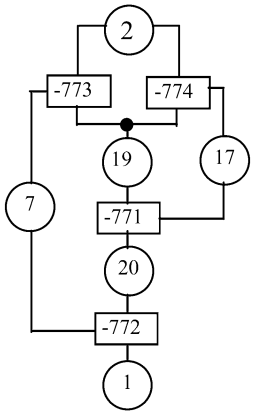

Fig. 9. Pattern circuit (G1).

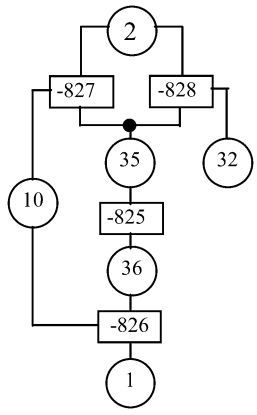

Fig. 10. Candidate circuit (G2).

fuzzy approach and the other three approaches becomes larger and larger with the increase of the circuit size. In Fig. 12, the circuits are larger than 100,000 transistors. The fuzzy approach still keeps the lowest run time.

VI. DISCUSSIONS AND CONCLUSION

A. Time Complex Analysis

In order to explain why the fuzzy attributed graph has such an excellent performance, we analyze the time complexity for the three experiments. Assume we have an M-transistor input circuit with n instances of the pattern circuit in it. Each circuit graph can be represented as $G = (V, E)$, where V is the number of vertices, and E denotes the number of edges.

In the circuit setup part, finding a "2" in the *source* column takes $O(1)$ time [27]. Since we have at least M "2's" in number, so the circuit partition takes $O(M)$. In the subcircuit identification part, assume the matching time between a pair of vertices is $O(1)$. Since there are V vertices in a pattern circuit graph, the worst case matching time is $O(V)$. Since we have at least n candidate subcircuits, so the worst case identification time is $O(nV)$.

Finally, we add the circuit setup time and subcircuit identification time together. The total run time is $O(M + nV)$.

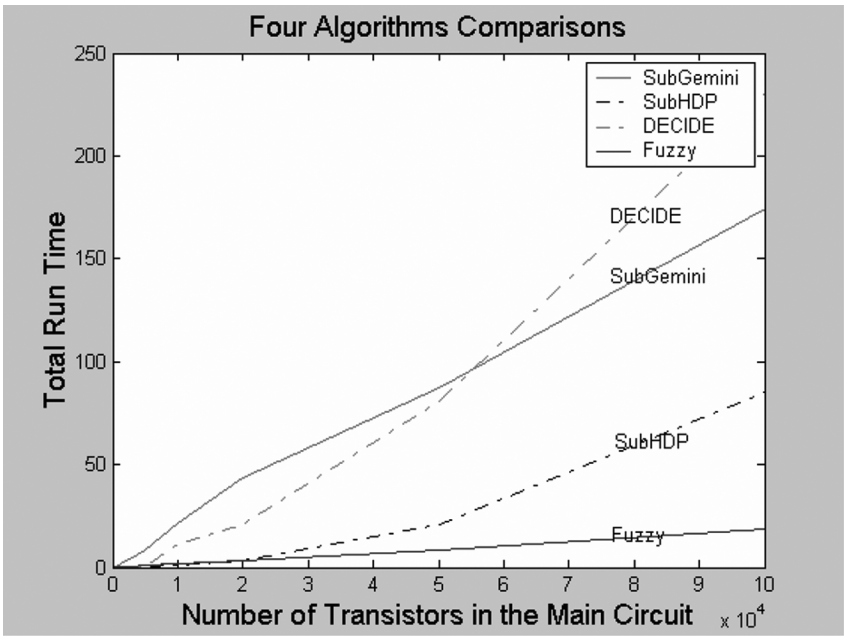

Fig. 11. Comparisons among four algorithms: SubGemini, SubHDP, DECIDE, and fuzzy attributed graph when circuit is no more than 100,000 transistors. The x-axis denotes the number of transistors in the main circuit. The y-axis is the total run time.

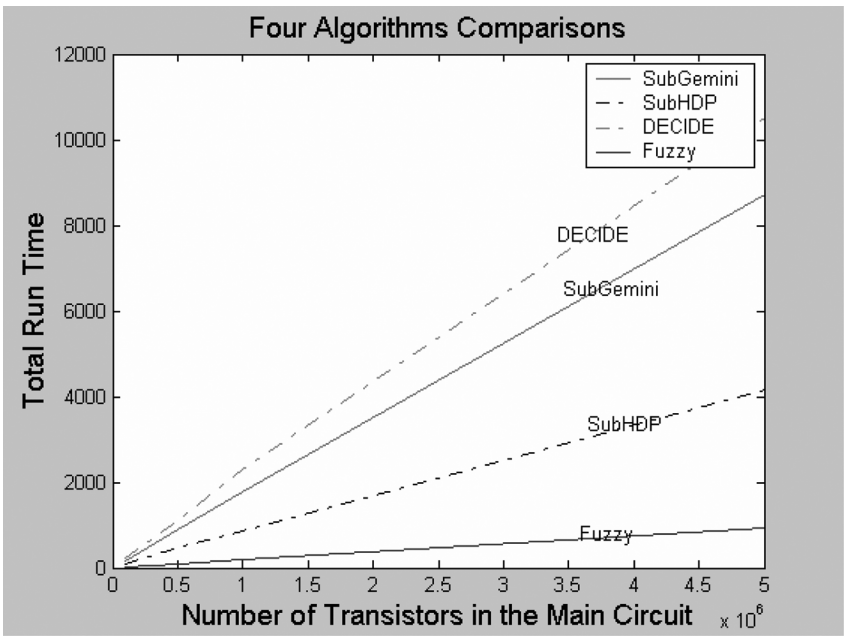

Fig. 12. Comparisons among four algorithms: SubGemini, SubHDP, DECIDE, and fuzzy attributed graph when circuit is larger than 100,000 transistors. The x-axis denotes the number of transistors in the main circuit. The y-axis is the total run time.

The aforementioned analysis can explain the experimental results. In Experiment I, M was increasing, but n and V remained unchanged. Therefore, the circuit setup time increases proportionally to the size of the input file, but the identification time did not change.

In Experiment II, n was increasing, but M and V remained unchanged. Therefore, the setup time did not change, but the subcircuit identification time increased proportional to the number of NAND gates.

In Experiment III, M, n, and V were all increasing, so both the setup time and subcircuit identification time increased. The results match well with the complexity computation, $O(M + nV)$.

B. Conclusions

The experimental results show that the fuzzy attributed graph is an efficient approach to implement the subcircuit extraction problem. By comparing the original circuit and the netlist extracted from the layout, the algorithm can not only find out all the instances of the pattern circuit, but also tell how much degree a candidate circuit is an instance of the pattern circuit. Importantly, it is the fastest algorithm over the existing commercial software and algorithms. Its advantages become more notable with the increase of the circuit size. This is because the fuzzy attributed graph algorithm employs several effective approaches to reduce the setup time and the identification time. First, we

pick only the netlists that consist of the same number of transistors as that in the pattern circuit. It helps greatly, to take out a number of false netlists. Second, in the subcircuit identification procedure, we terminate the program if no new core node pair was found after one iteration. Therefore, we can diagnose a false netlist at a very early time. Note that this algorithm used NAND gate as the pattern circuit; however, it can be other primitive gates, for example, AND gate, OR gate, buffer, and NOR gate. This algorithm should also work for complex gates, such as exclusive-OR (XOR), exclusive-NOR (XNOR), AND-OR-INVERTER (AOI), OR-AND-INVERTER (OAI), 2-to-1 multiplexer, and RAM cell. This makes our fuzzy attributed graph approach widely applicable to different circuits.

Our algorithm is the first fuzzy logic approach to solve the subcircuit extraction problem. It has a valuable contribution to the VLSI physical design automation, because it has superior speed over other software on flat layout versus schematic. In addition, the successful implementation of fuzzy attributed graph on the subcircuit extraction problem verifies the ability of fuzzy logic on practical very large scale integrated circuits applications. The complexity analysis agrees with experimental results.

REFERENCES

[1] S.-M. Kang and Y. Leblebici, *CMOS Digital Integrated Circuits Analysis and Design*, 3rd ed. New York: McGraw-Hill, 2002.

[2] M. J. S. Smith, *Application-Specific Integrated Circuits*. Reading, MA: Addison-Wesley, 1997.

[3] N. A. Sherwani, *Algorithms for VLSI Physical Design Automation*. Boston, MA: Kluwer, 1999.

[4] N. Zhang, D. C. Wunsch, II, and F. Harary, "The subcircuit extraction problem," *IEEE Potentials*, vol. 22, no. 3, Aug./Sep. 2003.

[5] S. M. Sait and H. Youssef, *VLSI Physical Design Automation: Theory and Practice*. New York: McGraw-Hill, 1995.

[6] V. D. Lehner, J. M. Cohn, and U. A. Finkler, "Pattern-matching for transistor level netlists," U.S. Pat. 473,881, 2002.

[7] T. Watanabe, M. Endo, and N. Miyahara, "A new automatic logic interconnection verification system for VLSI design," *IEEE Trans. Computer-Aided Design Integr Circuits Syst.*, vol. CAD-2, no. 1, pp. 70–82, Apr. 1983.

[8] M. Boehner, "LOGEX—An automatic logic extractor from transistor to gate level for CMOS technology," in *Proc. 25th Design Automation Conf.*, Anaheim, CA, Jun. 12–15, 1988, pp. 517–522.

[9] M. Ohlrich, C. Ebeling, E. Ginting, and L. Sather, "SubGemini: Identifying subcircuit using a fast subgraph isomorphism algorithm," in *Proc. 30th ACM/IEEE Design Automation Conf.*, Dallas, TX, Jun. 18, 1993, pp. 31–37.

[10] F. Luellau, T. Hoepken, and E. Barke, "A technology independent block extraction algorithm," in *Proc. 21st Design Automation Conf.*, Albuquerque, NM, Jun. 14–18, 1993, pp. 610–615.

[11] N. Vijaykrishnan and N. Ranganathan, "SUBGEN: A genetic approach for subcircuit extraction," in *Proc. 9th Int. Conf. VLSI Design*, Bangalore, India, Jan. 1996, pp. 343–345.

[12] Z. Ling and D. Y. Y. Yun, "An efficient subcircuit algorithm by resource management," in *Proc. 2nd Int. Conf. ASIC*, Shanghai, China, 1996, pp. 9–14.

[13] W.-H. Chang, S.-D. Tzeng, and C.-Y. Lee, "A novel extraction algorithm by recursive identification scheme," in *Proc. IEEE Int. Symp. Circuits and Systems*, Sydney, Australia, 2001, vol. 5, pp. 491–494.

[14] N. Zhang and D. C. Wunsch, II, "A novel subcircuit extraction algorithm using heuristic dynamic programming (HDP)," in *Proc. 2002 Int. Conf. VLSI*, Las Vegas, NV, Jun. 24–27, 2002, pp. 38–44.

[15] N. Zhang, F. Harary, and D. C. Wunsch, II, "CMOS IC topology design verification by heuristic dynamic programming," in *Proc. ANNIE '02*, St. Louis, MO, Nov. 10–13, 2002, pp. 33–38.

[16] N. Zhang and D. C. Wunsch, II, "Comparison of decision tree approach and neural networks based heuristic dynamic programming approach for subcircuit extraction problem," in *Proc. Intelligent Computing: Theory and Applications Conf. SPIE's 17th Annu. AeroSense Symp.*, Orlando, FL, Apr. 21–25, 2003, vol. 5103, pp. 143–149.

[17] D. V. Prokhorov and D. C. Wunsch, II, "Adaptive critic designs," *IEEE Trans. Neural Netw.*, vol. 8, no. 5, pp. 997–1007, Sep. 1997.

[18] P. Werbos, "Approximate dynamic programming for real-time control and neural modeling," in *Handbook of Intelligence Control, Neural, Fuzzy and Adaptive Approaches*. New York: Van Nostrand Reinhold, 1992, pp. 493–525.

[19] J. J. Murray, C. J. Cox, G. G. Lendaris, and R. Saeks, "Adaptive dynamic programming," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 32, no. 3, pp. 140–153, May 2002.

[20] D. Han and S. N. Balakrishnan, "State-Constrained agile missile control with adaptive-critic-based neural networks," *IEEE Trans. Control Syst. Technol.*, vol. 10, no. 7, pp. 481–489, Jul. 2002.

[21] N. Zhang and D. C. Wunsch, II, "A fuzzy attributed graph approach to subcircuit extraction problem," in *Proc. IEEE Int. Conf. Fuzzy Systems*, St. Louis, MO, May 25–28, 2003, pp. 1063–1067.

[22] W.-H. Tsai and K.-S. Fu, "Error-correcting isomorphism of attributed relational graphs for pattern analysis," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-9, no. 12, pp. 757–768, Dec. 1979.

[23] M. T. M. Gary and J. C. H. Poon, "A fuzzy-attributed graph approach to handwritten character recognition," in *Proc. 2nd IEEE Int. Conf. Fuzzy Systems*, 1993, vol. 1, pp. 570–575.

[24] K.-P. Chan, "Learning templates from fuzzy examples in structural pattern recognition," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 26, no. 1, pp. 118–123, Feb. 1996.

[25] F. Haray, *Graph Theory*. New Delhi, India: Narosa, 1998.

[26] W.-J. Liu and M. Sugeno, "A similarity measure of fuzzy attributed graphs and its application to object recognition," in *Proc. 5th IEEE Int. Conf. Fuzzy Systems*, 1996, vol. 2, pp. 767–772.

[27] T. H. Cormen, *Introduction to Algorithms*, 2nd ed. New York: McGraw-Hill, 2001.

Nian Zhang (S'00–M'05) received the B.E. degree in electrical engineering from Wuhan University of Technology, Wuhan, China, in 1996, the M.S. degree in automatic control engineering from Huazhong University of Science and Technology, Wuhan, China, in 1999, and the Ph.D. degree in computer engineering from University of Missouri-Rolla, in 2004.

She has been an Assistant Professor in the Department of Electrical and Computer Engineering at the South Dakota School of Mines and Technology, Rapid City, since August 2004. Her research interests include fuzzy logic, neural signal processing, pattern recognition, autonomous robot navigation, and VLSI design of neural networks.

Dr. Zhang is the recipient of Governor's 2010 Individual Research Seed Grant and 2003 Best Student Paper Award at the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2003). She is a Member of IEEE Computational Intelligence Society. She served as program committee member of the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'03 and FUZZ-IEEE'05) and 2006 IEEE Mountain Workshop on Adaptive and Learning Systems (SMCals 2006), and reviewer for International Joint Conference on Neural Networks (IJCNN'02–IJCNN'06).

Donald C. Wunsch II (S'86–M'90–SM'94–F'05) received the B.S. degree in applied mathematics from the University of New Mexico, Albuquerque, in 1984, the M.S. degree in applied mathematics and the the Ph.D. degree in electrical engineering, both from the University of Washington, Seattle, in 1987 and 1991, respectively, and the M.B.A. degree from Washington University, St. Louis, MO, in 2006.

He is the Mary K. Finley Missouri Distinguished Professor of Computer Engineering at the University of Missouri-Rolla, where he has been since 1999. His prior positions were Associate Professor and Director of the Applied Computational Intelligence Laboratory at Texas Tech University, College Station, Senior Principal Scientist at Boeing, Seattle, WA, Consultant for Rockwell International, Kirkland AFB, NM, and Technician for International Laser Systems, Kirkland AFB, NM. He has well over 200 publications, and has attracted over $5 million in research funding. He has produced eleven Ph.D.'s—six in electrical engineering, four in computer engineering, and one in computer science.

Dr. Wunsch has received the Halliburton Award for Excellence in Teaching and Research and the National Science Foundation CAREER Award, among many other awards. He served as a Voting Member of the IEEE Neural Networks Council, Technical Program Co-Chair for IJCNN 02, General Chair for IJCNN 03, International Neural Networks Society Board of Governors Member, and 2005 President of the International Neural Networks Society.