Performing Technology Mapping and Optimization by DAG Covering: A Review of Traditional Approaches

Evriklis Kounalakis

Abstract

Technology mapping is the process of describing a circuit with a netlist that consists of only gates that belong to a specified technology library. Typically, the problem is formulated as a DAG covering problem and most of the mapping algorithms are applied on a forest of trees which originates from the original DAG. This work describes and compares three technology mapping approaches; DAGON, NOA and DOT. Each of them is described in detail and compared against the others. Based on this review, the advances in technology mapping can be derived.

1 Introduction

Technology mapping is the process of transforming a technology independent netlist into a technology dependent one. A technology independent netlist is a set of generic gates like *AND*, *OR*, *XOR* gates with the appropriate connections among them so that they describe an arbitrary circuit. The process of technology mapping transforms this set of technology independent gates into an equivalent netlist, which has the same functionality and uses only gates that belong to a given technology library. The library gates sets may have an arbitrary size and contain any set of gates. Typically, gate libraries contain gates such as 2-input *NAND* gates, inverters and any other suitable gate.

Keutzer in [5] proposed an algorithm which performs technology mapping using a tree-covering technique. Keutzer assumes a *subject graph*, which is the circuit to be mapped with the constraint that it is described only with *NAND* and *NOT* gates. This transformation, called decomposition, is easy to be performed on the technology independent netlist. Keutzer assumes also *pattern graphs* which are the decomposed technology dependent gates. Keutzer's approach is to decompose the subject graph, which is a DAG, into a forest of trees, perform technology mapping on every tree and finally, compose the partial results to form the general solution. The reason for decomposing the subject graph into a forest of trees is given in [6], which states that the problem of covering a DAG for minimum area is NP-hard.

Chaudhary and Pedram in [2] proposed a way to minimize the area of the technology mapped circuit taking into account the delay constraints of the design. This approach is claimed to be a near optimal algorithm (NOA) and provides an area-speed tradeoff for every node of the circuit trees. The options available for each node are the alternative technology mappings for this specific node, which infer an area and a delay overhead. Based on the delay constraints, the minimum area selection can be chosen

for each node of the tree, thus the minimum area implementation of the whole circuit can be derived without violating the delay constraints.

Motivated by the loss of optimality when mapping the forest of trees instead of the complete DAG that describes the circuit, Kukimoto *et al.* in [7] proposed a technology mapping algorithm that can be applied to the whole DAG of the circuit. This algorithm is a delay-optimal technology mapper (DOT) that guarantees better results compared to the algorithms that require the original DAG to be decomposed into a forest of trees.

This paper is a review of the approaches described in [5], [2], [7].

The rest of the paper is organized as follows. Section 2 provides a formalization of the problem of technology mapping. Section 3 describes the methodology followed by the algorithms reviewed. The experimental results of the algorithms are presented in section 4, which is followed by Section 5, which compares the algorithms presented. Section 6 describes the possible enhancements that can be applied to the algorithms and Section 6 provides the conclusions.

2 Problem Formulation

Given a circuit and a set of library gates, map the circuit so that it uses only library gates. The circuit is expressed as a Directed Acyclic Graph (DAG) and each library gate is described by a tree of NAND and NOT gates. The root of the tree is the output of the gate and the leafs of the tree are the inputs of the gate. Each internal vertex of the tree is one 2-input NAND or NOT gate and each edge of the tree is a connection between two of the generic gates (NAND and NOT gates). Thus, one library gate may include more than one generic gates. An example of a 4-input NAND gate expressed with the generic gates is given in Figure 1.

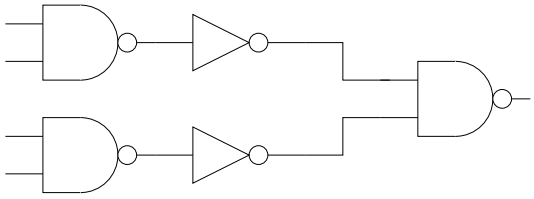


Figure 1. Mapped 4-input NAND

An example of a possible input circuit is given in Figure 2. During technology mapping, instances

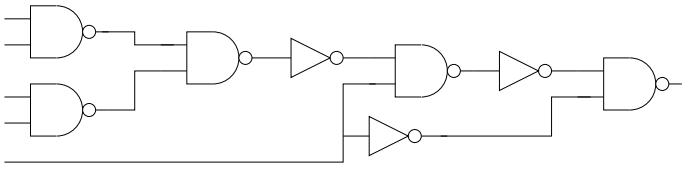


Figure 2. A Possible Input Circuit

of the library gates are matched against the DAG which describes the input circuit. The objective is to find the best matches, which minimize some design parameter, like area or delay. Keutzer and Richards

showed in [6] that graph covering for a circuit described as a DAG with the objective of minimizing area is NP-hard. Thus, the usual methodology involves decomposing the DAG into a forest of trees and apply technology mapping to the forest of trees instead of the complete DAG. This approach sacrifices some optimality, as the multiple fanout points are omitted from optimization, but can be solved in polynomial time.

3 Methodology

Due to the complexity of the task of mapping a circuit expressed as a DAG against a set of library gates, the common approach is to decompose the DAG into a forest of trees. To illustrate this notion, the circuit in Figure 2 would be decomposed into the circuits shown in Figure 3. Notice that the rightmost

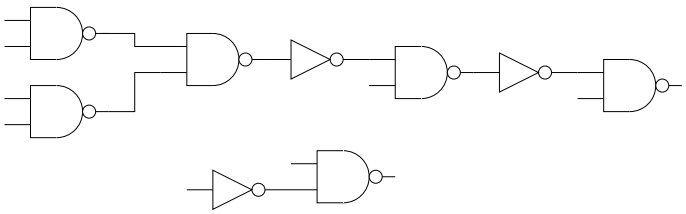


Figure 3. A Decomposed Input Circuit

NAND-gate is present in both trees of the decomposed DAG. The DAGON approach and the NOA use this decomposition of the initial DAG. DOT assumes a load independent scheme, which means that the load values of the gates are not taken into account. DOT also ignores any possible area optimizations. This approach reduces the complexity of DAG mapping to polynomial time, thus removing the need to decompose the DAG into a forest of trees.

3.1 DAGON

The DAGON approach consists of three distinct phases. Initially, the DAG is decomposed into a forest of trees. The decomposition is simply the creation of a new tree for every node that has fanout greater than one, as it is shown in Figure 3. The trees are then individually traversed and technology mapped using *twig* [11]. *Twig* performs tree matching by firstly identifying all the candidate matches using the Aho-Corasick [1] algorithm. *Twig* uses a technology library which contains all the available gates in canonical NAND/NOT form. An example of a complex *AND-OR-INVERT* gate in NAND/NOT form is given in Figure 4. The technology library may have alternative implementations for the same gate, or implementations of the same gate with different number of inputs. Figure 5 shows an AOI-gate with three inputs. Each gate of the technology library has a *cost* assigned to it, which defines the implementation cost of the particular gate. The *cost* variable is used for the cost calculation of the chosen implementation. The cost at each node n is the cost of the tree T (gate) that has its root at node n plus the cost of all the gates that belong to the subtrees of the T. An example of the cost calculation is given below. Consider the circuit of Figure 6. Assume that an inverter has a cost of 2, a NAND-gate has a cost of 3 and a AOI-gate (AND-OR-INVERT) has a cost of 7. There are two possible implementations of the circuit in Figure 6. Either use an inverter at the root of the tree plus some 2-input NAND-gates

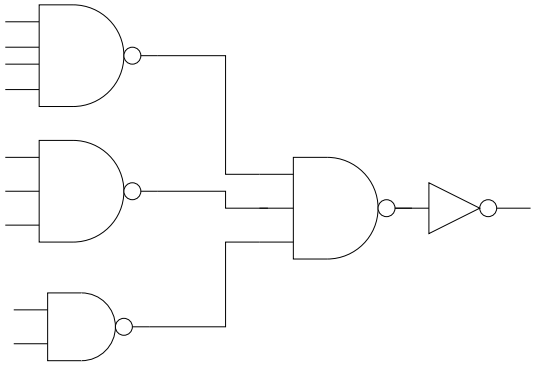


Figure 4. AND-OR-INVERT Gate in NAND/NOT Form

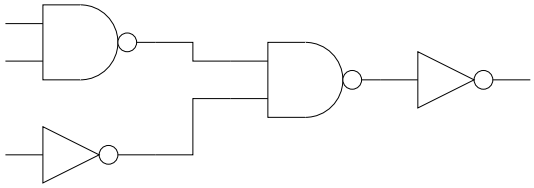


Figure 5. 3-input AND-OR-INVERT Gate in NAND/NOT Form

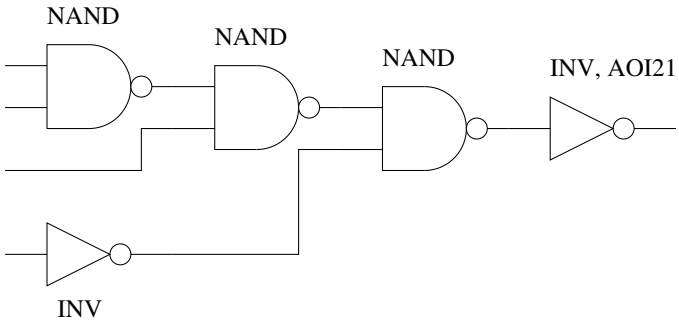


Figure 6. An Example of DAGON Minimal Cost Decision

and a second inverter or use a complex AOI gate which consists of all the gates in the tree, except for one NAND-gate. In the first case the cost of the implementation is the cost of the inverter plus the cost of the implementation of the tree originating below the root. In this case, the total cost of the tree is 13, as there are three NAND-gates of cost 3 each and two inverters of cost 2 each. In the second implementation, the cost is only the cost of the complex AOI-gate, which includes 2 NAND-gates and two inverters as is shown in Figure 5 and has a cost of 7, plus the cost of a NAND-gate. Thus, the cost of the latter implementation is 9, compared to the cost of the first implementation which is 13.

In order to determine all the candidate matches for a given tree, a brute-force tactic, which compares all the library gates against each node of the tree, would require $O(TREESIZExLIBRARYSIZE)$ time. *Twig* uses the Aho-Corasick [1] which requires $O(TREESIZE)$ time. When all the candidate matches

have been determined, each possible implementation is stored at each node along with the corresponding cost.

The next step is to find the minimal cost matches. This is accomplished by traversing the tree in depth-first fashion. Given a tree T with root r and subtrees $T_1 \ldots T_n$, the minimal cost implementation for T is found by first finding the minimal cost implementation for each one of $T_1 \ldots T_n$. Using a depth-first search, each subtree T_i with root r_i, which is matched against each applicable pattern p_i of the technology library, is assigned to the minimal cost pattern p_{min} and the cost for this subtree is fixed. Iteratively, the cost of a parent tree of T_i will be defined the same way and the total cost of the tree T will be the sum of the cost of all the subtrees T_i.

After the minimal cost match for every tree is determined, the results are concatenated in order to produce the complete solution for the initial DAG.

3.2 NOA

NOA focuses on technology mapping of NAND-decomposed trees under delay constraints with the objective of providing area-speed tradeoffs. NOA consists of two phases. During the first phase, a postorder traversal of the tree determines the area-speed tradeoff for all the nodes. At the end of this phase, an area-speed tradeoff is available for the tree. During the second phase, an implementation based on the area-speed tradeoff of the root is chosen and a preorder traversal determines the implementation of all the tree nodes. During the postorder traversal NOA calculates area-speed curves like the ones in Figure 7. Figure 7 shows the possible a, b implementations for *NODE A* and the possible c, d, e imple-

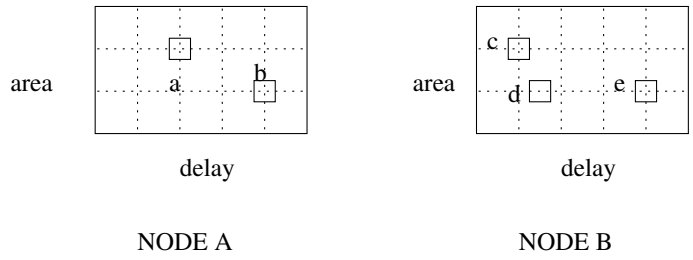


Figure 7. NOA Area-Speed Curve

mentations for *NODE B* along with the delay and area cost for each implementation. For example, the c implementation of *NODE B* has a delay cost of 1 unit and an area cost of 2 units. When a node is visited, its area-speed curve is calculated based on the area-speed curve of its fanin gates. The area-speed curves of the fanin nodes of a given node is determined previously, as the postorder traversal visits first the leafs of the tree and then moves upwards until it reaches the root. Next we give an example of how the area-speed curve is computed for a non-leaf node. Suppose the tree of Figure 8. Suppose that there is a library gate which can be matched at node D and that is consists of the nodes A, B, C and D. The area-speed curves are given in Figure 7. The area-speed curve for node D is created using the area-speed curves of nodes A and B in the following way. First, we select a point in the area-speed curve of node A, say point b. This point has a delay cost of 4 units and an area cost of 1 unit. Then, we select a point in the area-speed curve of node B which has a delay cost of at least 4 units. This is point e. Thus, a point on the area-speed curve of node D will have a delay cost of 4 plus the delay of the new gate and an area

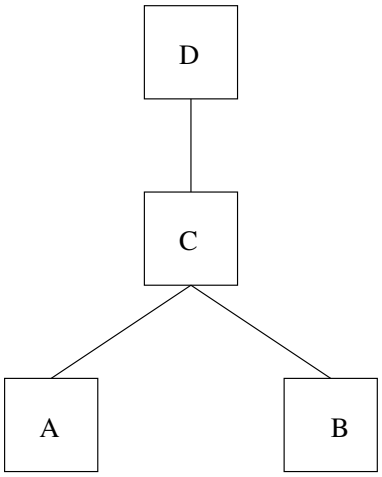
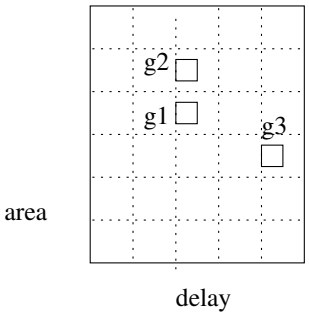


Figure 8. NOA Tree Example

cost of 2 plus the area of the new gate. In general, for a point g_{new} in the area-speed curve of gate g, which depends on points a and b holds that:

$$delay(g_{new}) = max(delay(a), delay(b)) + delay(g)$$
$$area(g_{new}) = area(a) + area(b) + area(g)$$

Using the same procedure for all the pairs of points in the area-speed curves A and B, we generate the area-speed curve for node D, which looks like the one in Figure 9. Point $g1$ is generated from merging



NODE D

Figure 9. NOA Generated Area-Speed Curve

points a and d, $g2$ comes from a and c and $g3$ is produced from b and e. Note that point $g2$ is redundant, as point $g1$ has the same area cost as point $g2$, but with less delay cost. Thus, point $g2$ can be omitted from the area-speed curve of node D. Following the same procedure iteratively, the area-speed curve for the root of the tree is constructed. After that, given the predefined delay constraints provided by the designer, the minimum area implementation which satisfies the delay constraint is chosen for the

root. Using a preorder traversal for the tree until it reaches the tree, the minimum area implementations are chosen for the complete tree. The minimum area implementation for the whole circuit is found by applying the same procedure to all of the trees of the decomposed DAG of the circuit.

3.3 DOT

DOT focuses on minimal-delay technology mapping without decomposing the DAG of the circuit into a forest of trees. The algorithm of DOT is based on the FPGA technology mapping procedure described by Cong *et al.* in [3]. This algorithm tries all the possible k-cuts of a specific node that can be implemented by a single lookup-table (LUT). In order to elaborate on this, assume a DAG in which the number of nodes that fanin into any node is no more than k. This is a k-bounded network. The FlowMap algorithm presented in [3] visits each node of the tree starting from the leafs and moving towards the root. For each node it visits, it investigates all cuts of its fanin that have size of at most k. Recall that the network is already k-bounded, thus no transformation is needed and no inputs are omitted here. For every cut that the algorithm assumes, it maps the gates that are included in the cut into an LUT. The mappings are compared against each other and the optimal cut is chosen. Thus, the optimal depth of each cut is determined. The algorithm proceeds until the root is reached. At each node, the optimal cuts for its fanin nodes is already determined, so the algorithm guarantees that there are no nodes left out from this optimization. The algorithm stops when the optimal cut for every node is determined. A brute-force approach of this method has a complexity of $O(n^k)$, where n is the number of the network nodes. In [3] it is shown that this problem can be solved as a network flow problem with complexity proportional to k. When the optimal depth for each node is determined, the network is traversed from the root towards the leafs and assigns an LUT to the nodes that are in the optimal cut of the node currently being visited. The complexity of this algorithm is $O(kmn)$, where m denotes the number of edges in the network.

The approach in [7] is the same as in [3], with the only difference that instead of LUTs there are only library gates available. The optimal fanin cone is determined for each node and the mapping proceeds by matching the library gates against the subgraph that consists of the nodes that are in the fanin cone of the node currently being investigated. The matching can be either exact or *extended*. In exact match, the pattern graph (library gate) must be identical with the subgraph that is being matched. In the case of *extended* match, if a gate fanouts to more than one gate, then duplication of this node in the subgraph is allowed, in order to find an exact match. An example of an *extended* match is given in Figure 10.

4 Results

In this section, the experimental results for all the approaches are presented. Table 1 shows the results for DAGON on the de Geus [4] benchmarks. Table 1 compares the DAGON implementations with the NAND/NOT implementations of the same circuit. The *grids* columns correspond to standard cell grids with the assumption that a two-input NAND gate uses three grids. The runtime is measured on a VAX 8650 machine. The gate count and the grid count show improvement if the DAGON approach is used.

NAO is implemented in a program called ADIEU and is compared against the MIS2.2 technology mapper [12]. Tables 2 and 3 show the normalized results for MIS2.2 and ADIEU. The normalization in Tables 2 and 3 is for the area mode of ADIEU. ADIEU produces 6% faster circuits than MIS2.2 on average, but with an overhead of 3% when using the area mode. With timing mode, the timing

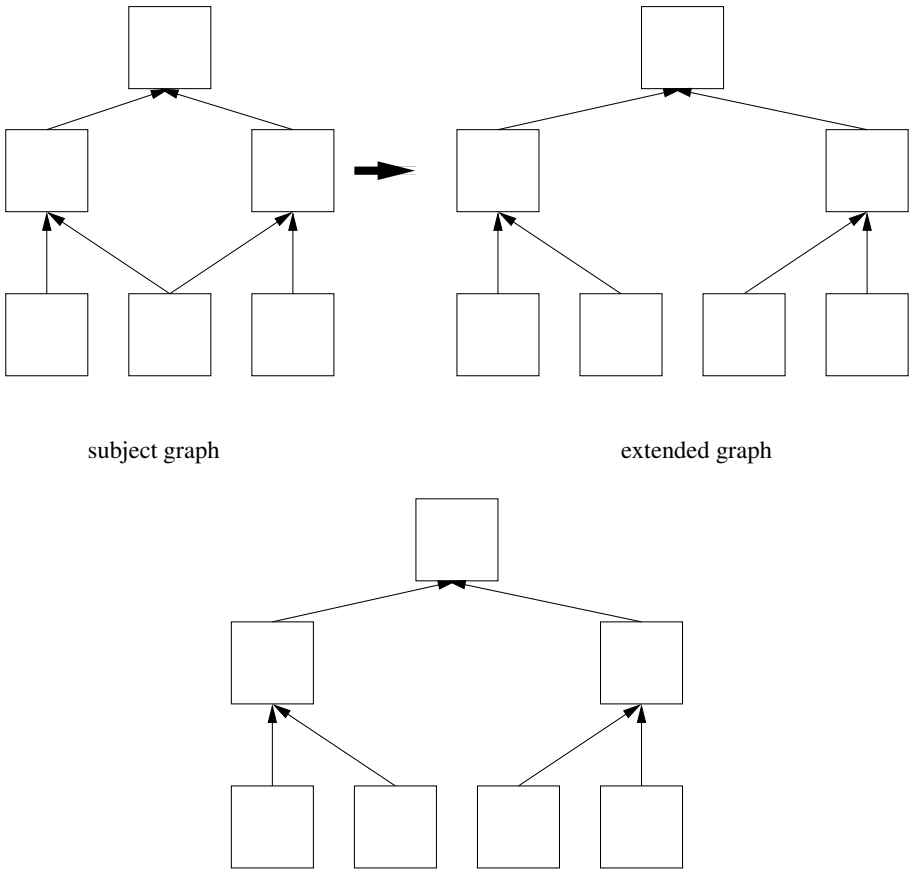


Figure 10. Extended Match Example

Circuit	NAND/NOT		DAGON		runtime
	gates	grids	gates	grids	seconds
rd53	36	109	19	79	0.1
9sym	66	202	44	163	0.2
vg2	90	253	65	208	0.2
rd73	90	275	47	196	0.3
sao1	111	331	64	245	0.4
sao2	111	370	61	273	0.4
bw	171	479	118	380	0.6
duke2	338	1013	251	851	1.2
gpio	400	1073	281	835	1.3
lmtc	908	2559	660	2078	3.1
pla4	1478	4338	1022	3465	5.6

Table 1. Results for DAGON

optimization is similar for both mappers, but ADIEU's circuit are 17% smaller.

Circuit	MIS2.2			
	Area mode		Timing mode	
	area	delay	area	delay
9symml	1.00	0.99	1.35	0.91
apex6	0.94	0.92	1.35	0.90
apex7	0.97	0.97	1.26	0.88
b9	0.96	0.93	1.06	0.78
des	0.98	2.19	1.43	1.24
rot	0.99	0.98	1.18	1.02
z4ml	0.93	1.01	1.24	0.87
C1908	0.93	1.16	1.27	1.13
C1355	0.97	0.92	1.18	0.86
C432	1.00	1.00	1.26	0.81
C880	0.98	0.99	1.17	0.83
C3540	0.99	1.00	1.27	0.96
C5315	0.98	0.89	1.28	0.84
C7552	0.97	1.03	1.31	0.74
average	0.97	1.06	1.26	0.91

Table 2. *MIS2.2* **Results for NAO**

Circuit	MIS2.2			
	Area mode		Timing mode	
	area	delay	area	delay
9symml	1.00	1.00	1.09	0.90
apex6	1.00	1.00	1.03	0.75
apex7	1.00	1.00	1.01	0.93
b9	1.00	1.00	1.00	0.89
des	1.00	1.00	1.06	0.92
rot	1.00	1.00	1.01	0.95
z4ml	1.00	1.00	1.10	0.87
C1908	1.00	1.00	1.04	1.01
C1355	1.00	1.00	1.10	0.93
C432	1.00	1.00	1.07	0.83
C880	1.00	1.00	1.07	0.90
C3540	1.00	1.00	1.04	0.91
C5315	1.00	1.00	1.05	0.85
C7552	1.00	1.00	1.03	0.90
average	1.00	1.00	1.05	0.91

Table 3. *ADIEU* **Results for NAO**

Tables 4 and 5 show the experimental results for DOT compared against the standard tree matching procedure for the `lib2.genlib` and `44-1.genlib` libraries respectively. The CPU time is measured on a DEC AlphaServer 8400 5/300 and is presented in seconds. The results show that there is a significant improvement in delay, but with a large area overhead.

Circuit	Delay		Area		CPU time	
	tree	DAG	tree	DAG	tree	DAG
C2670	11.54	9.43	1552	2008	2.3	2.6
C3540	17.20	14.00	2075	2926	3.1	3.7
C5315	16.55	13.04	3687	4275	5.4	6.0
C6288	56.99	41.95	4107	9291	4.9	5.9
C7552	14.23	11.06	4983	6452	6.8	8.4

Table 4. *lib2.genlib* **Results for DOT**

Circuit	Delay		Area		CPU time	
	tree	DAG	tree	DAG	tree	DAG
C2670	27	18	2998	4568	2.0	2.0
C3540	42	30	4007	6640	2.7	2.8
C5315	46	33	6817	8352	4.6	4.8
C6288	125	120	7782	7121	4.3	4.4
C7552	39	28	9552	11149	6.0	6.3

Table 5. *44-1.genlib* **Results for DOT**

5 Comparison

In this section, the three approaches are compared against each other. The comparison is focused on the complexity of each approach.

DAGON tries all library gates on each node it visits. Clearly, the complexity of DAGON is $O(DAGSIZExLIBRARYSIZE)$, where *DAGSIZE* is the size of the DAG that describes the circuit and *LIBRARYSIZE* is the number of gates that the technology library consists of.

The complexity analysis of NOA requires that we find how quickly we can generate the delay curves for each node. Each match g of a node n with a fanin cone of size k has a delay curve with at most $N = \sum_{i=0}^{k} N_i$ points. If the delay curve for each point is sorted, then the time required to generate each point is $O(klog(N_{max}))$, where N_{max} denotes the maximum N_i. After each point is generated a new sorting of cost $O(Nlog(N))$ is required. The total time required for the generation of a delay curve for a match g at a node n is $O(N^2 log(N) klog(N_m))$.

For DOT, one instantiation of `graph_match` has a complexity of $O(p)$ according to [10], where p denotes the size of the technology library. Thus, the complexity of DOT for the entire graph is $O(sp)$, where s is the number of nodes in the graph as `graph_match` is called once for each node.

6 Enhancements

This section describes the enhancements that have been applied to the basic algorithms described in a previous section in order to improve the quality of the results.

For DAGON, three enhancements have been explored and applied to the DAGON tool. The first improvement stems from the fact that the matching algorithm can do better if it takes into consideration details relevant to the complete DAG instead of using information relevant only locally to the tree it tries to match. Such details are the fanout of the nodes and the existence or not of the inverse of the signals present in the tree. The second improvement originates from the application of DAGON to sequential circuits. In these circuits, both the inverted and the non-inverted outputs of a flip-flop can be used in optimization gates. A final enhancement of DAGON is the search of redundant gates that may be present due to the optimization of two neighboring trees.

For DOT, there are two enhancements, one of which targets optimization of a sequential circuit and the second one targeting global optimization. Pan and Liu in [9] proposed a polynomial-time algorithm for the optimization of a sequential circuit. The basic steps are the retiming of the initial circuit, which is followed by technology mapping of the re-timed circuit. After technology mapping, another retiming on the mapped circuit is performed. The procedure is repeated iteratively and can compute the minimum cycle mapping in polynomial time. The second enhancement takes into consideration the fact that the initial subject graph is constructed without knowledge of the technology library. Lehman *et al.* in [8] include a number of decompositions into an extended subject graph and perform technology mapping on the extended graph.

7 Conclusions

This paper has provided a comparison of three technology mapping approaches. All of them address the problem by providing solutions for matching pattern graphs against trees that describe the technology independent circuit. One of them tries to find the best matches of the pattern graphs against the complete DAG of the circuit without decomposing it into a forest of trees. All of the approaches have been described in detail and their experimental results have been presented. The complexity analysis has shown that all algorithms require at least $O(sp)$ where s and p describe the size of the technology library and the size of the circuit respectively.

References

[1] A. V. Aho and M. J. Corasick. Efficient string matching: An aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[2] K. Chaudhary and M. Pedram. A near optimal algorithm for technology mapping minimizing area under delay constraints. In *Proceedings of 29th ACM/IEEE Design Automation Conference*, 1992.

[3] J. Cong and Y. Ding. An optimal technology mapping algorithm for delay-optimization in lookup-table based fpga designs. *IEEE Transactions on Computer-Aided Design*, 13(1):1–12, 1994.

[4] A. J. de Geus. Logic synthesis and optimization benchmarks for the 1986 design automation conference. In *Proceedings of the 23rd Design Automation Conference*, 1986.

[5] K. Keutzer. Dagon: Technology binding and local optimization by dag matching. In *Proceedings of 24th ACM/IEEE Design Automation Conference*, 1987.

[6] K. Keutzer and D. Richards. Computanional complexity of logic synthesis and optimization. In *Proceedings of International Workshop on Logic Synthesis*, 1989.

[7] Y. Kukimoto, R. K. Brayton, and P. Sawkar. Delay-optimal technology mapping by dag covering. In *Proceedings of 1998 DAC Conference*, 1998.

[8] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness. Logic decomposition during technology mapping. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, 1995.

[9] P. Pan and C. L. Liu. Optimal clock period fpga technology mapping for sequential circuits. In *Proceedings of 33rd ACM/IEEE Design Automation Conference*, 1996.

[10] R. Rudell. Logic synthesis for vlsi design. *Technical Report UCB/ERL M89/49 University of California, Berkeley*, 1989.

[11] S. Tjiang. Twig reference manual, 1986.

[12] H. J. Touati, C. W. Moon, R. K. Brayton, and A. Wang. Performance-oriented technology mapping. In *Proceedings of the 6th MIT Conference in Advanced Research in VLSI*, 1990.