VLSI Design
1995, Vol. 3, No. 1, pp. 1–12
Reprints available directly from the publisher
Photocopying permitted by license only



# Cell Generator-Based Technology Mapping by Constructive Tree-Matching and Dynamic Covering

MARTIN LEFEBVRE and CLIFF LIEM*
Department of Electronics
Carleton University
1125 Colonel By Drive
Ottawa, Canada K1S 5B6

*(Received July 5, 1993, Revised September 25, 1993)*

Technology mapping is the final step of logic synthesis which consists of mapping an optimized technology independent logic network representation into a circuit realization in a given technology. An important component of the technology mapping problem is the identification of feasible library cells for the realization of the logic operators in the logic tree. There are two main classes of such matching algorithms. **Library-based matching algorithms** [1–4] require that all available physical components be represented explicitly in a pattern library. Sections of the logic network are then matched against this pattern list for the identification of suitable components. In contrast, **cell generator-based matching techniques** [6–8] accept feasibility constraints on the complexity and quantity of physical components according to limits imposed by the target technology or the capabilities of the cell generator. Hence, individual patterns are not stored in a library and are instead generated as needed. In this paper, we present a new cell generator-based constructive matching algorithm. Because the algorithm builds matched patterns incrementally, very large cell families can be accommodated using time and space resources that are proportional to the size of the largest feasible cell pattern and not the size of the library of patterns as would be the case for library-based approaches. Also, whereas existing cell generator-based matching techniques combine the tasks of matching (identification) and covering (selection), constructive matching provides more flexibility by not restricting the covering phase. Empirical results demonstrate the increased quality of the technology-mapped circuits when larger cells are available.

**Key Words:** *Technology mapping; Logic synthesis; Circuit optimization; Physical design; Cell generators*

## I. INTRODUCTION

### 1.1 Overview of Technology Mapping

Seminal work in logic synthesis [1–3] has provided a four-step methodology for technology mapping. Starting from an optimized logic network represented as a directed acyclic graph (DAG), the mapping process can be summarized as follows:

- partitioning of the initial DAG into a forest of trees, in order to avoid the computationally intractable graph covering problem,
- decomposition of each tree into an interconnection of two-input AND/OR and inverter functions,
- matching, or the identification of all possible physical realizations (cells) of each node in the decomposed logic trees, and
- covering, or the selection of a subset of the cells identified during the matching phase that implement the logic network while optimizing some objective cost function.

This technology mapping methodology is generally referred to as **tree mapping** due to the initial partitioning of the DAG into a forest of trees. An efficient method for tree matching [1, 2] consists of using pattern representations for the library of cells,

*Cliff Liem is now with Bell-Northern Research, P.O. Box 3511 Station C, Ottawa, Canada, K1Y 4H7.

then using string matching techniques to find subtrees corresponding to cells. An alternative matching procedure based on Boolean techniques was proposed in a system called Ceres, a technology mapper developed at Stanford University [3, 4]. Boolean representations can handle multiple occurrences of variables better than tree-based representations, and allow the use of *don't care* information which is useful for optimization.

### 1.2 Need for New Mapping Techniques for Very Large Cell Families

Both string matching and Boolean matching require that all the feasible cells be represented explicitly in a pattern library. This is not a serious limitation for typical standard cell families comprised of a few hundred cells. However, where the size of the target cell library is only bounded by technology parameters such as the number of p-type and n-type transistors allowed to be connected in series in a given cell, the number of corresponding patterns may be impractically large for storage in an explicit library. As an example, a cell library comprised of all series-parallel static CMOS logic cells with a maximum of six transistors in series would contain 154,793,519 distinct cells. An enumeration of all non-isomorphic CMOS complex gates as a function of the number of allowable p-type and n-type transistors in series is shown in Table 1.

An important question is whether such large cell families find practical applications in VLSI design. Clearly, for circuit applications where performance is not critical, significant area savings can result from merging discrete logic functions into series-parallel complex gate structures in order to reduce the number of transistors used. Furthermore, a recent study of the abilities of sub-micron CMOS technologies [5] gives a clear indication that the delay associated with series-connected transistors exhibits a linear dependence on the number of transistors in series for up to 6 or 7 transistors. Therefore, deep sub-micron processing technologies invite an increasing use of larger complex cell families even where circuit delay is critical. This is in contrast with the limit of 3 or 4 transistors in series usually imposed by current technologies.

TABLE I
Enumeration of non-isomorphic CMOS complex gates based on the limit of series transistors in the nMOS(s) and pMOS(p) regions [2].

| p \ s | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| 2 | 2 | 7 | 18 | 42 | 90 | 186 |
| 3 | 3 | 18 | 87 | 396 | 1677 | 6877 |
| 4 | 4 | 42 | 396 | 3503 | 28435 | 222913 |
| 5 | 5 | 90 | 1677 | 28435 | 425803 | 6084393 |
| 6 | 6 | 186 | 6877 | 222913 | 6084393 | 154793519 |

(From: E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, A. Wang, "Technology Mapping in MIS", *Proceedings of the International Conference on Computer-Aided Design*, 1987, pp. 116–119. © 1987 IEEE)

In this paper, we present a new tree-based matching algorithm for cell generator-based technology mapping. The approach, called **constructive matching**, identifies series-parallel gate implementations for sub-trees in a logic network. Since the algorithm does not require an explicit representation of all feasible cells, it can handle very large libraries.

### 1.3 Organization of the Paper

The remainder of this paper is organized as follows. Section II presents related work in the field of complex-cell-based technology mapping. Section III describes our constructive matching algorithm with a detailed comparison to both library-based and cell generator-based approaches. Section IV provides an overview of our prototype technology mapping system based on constructive matching. In association with the new proposed matching module, we have implemented standard partitioning, decomposition, and covering modules to complete the technology mapper. Finally, Section V contains experimental results followed by conclusions in Section VI.

## II. RELATED WORK

### 2.1 Algorithm for the Formation of the Largest Possible Complex Cells

The first approach to cell-generator based technology mapping was proposed by researchers at the Eindhoven University of Technology [6]. The mapper finds cell matches constrained by a size restriction. Any Boolean network is associated with an ordered pair $(a, b)$, which is defined as follows:

- if the expression is a sum, $a$ is the largest cut set and $b$ is the longest path of the respective graph.
- if the expression is a product, $a$ is the longest path and $b$ is the largest cut set of the respective graph.

The matching problem is approached with a one-level view of the Boolean expression to be mapped which is equivalent to a breadth-first traversal of the logic tree until the size restriction is encountered. Given an expression:

$$expr = (op\ subexpr_1 \ldots subexpr_n),$$

where $op$ is either * or +

and a size restriction $(a, b)$ (restricts the allowable size of an expression), a complex cell mapping can be calculated. The parameter $a$ restricts the number of subexpressions that may be taken for mapping to one cell. If the number of subexpressions exceeds $a$, then a substitute variable is introduced to take the place of a number of subexpressions, i.e.:

$$expr = (op\ subexpr_1 \ldots subexpr_n)$$

becomes the set of expressions:

$$expr = (op\ subst\ subexpr_1 \ldots subexpr_{n-i})$$

$$subst = (op\ subexpr_{n-i+1} \ldots subexpr_n)$$

and the substitute variable is put at the top of the list so that the recursive algorithm takes the lowest level subexpressions at the bottom of the list. Of course, this will not always guarantee the lowest level subexpressions; therefore, the list must be sorted at times. Using a heuristic rule, a subexpression of size $(x_1, y_1)$ is considered *smaller* than a subexpression of size $(x_2, y_2)$, if $(x_1^2 + y_1^2) < (x_2^2 + y_2^2)$.

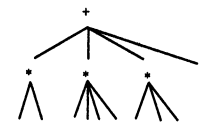
This approach aims at finding the largest possible cells which can realize a given function. While this may be useful for area optimization, there is no evidence to suggest that such a greedy algorithm will result in circuits with optimal performance. For example, the realization of a high fanout node with a large complex gate may lead to very poor performance.

## 2.2 Depth Reduction Algorithm

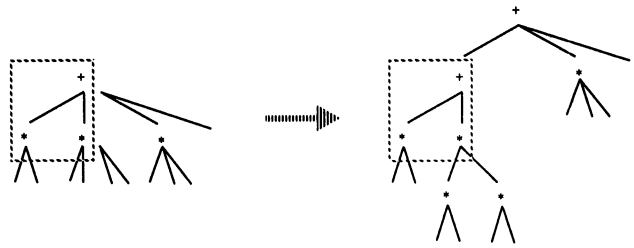
L'Institut National Polytechnique de Grenoble [7, 8] has introduced a technology mapper which also works on a library of complex cells. A matching phase identifies and selects pattern trees corresponding to cells in the library based on their relative costs. Pattern costs are defined in such a way that depth-reducing patterns are given priority. One of the distinguishing features of this approach is that it operates on the initial factorization of the circuit and does not perform binary decomposition of the network before matching. In this way, the depth of the factorized tree is already limited and therefore, the depth-decreasing pattern matching strategy can be very effective. One consequence of this choice of a non-binary tree representation is that some matches may not consume all of the inputs of all the nodes covered. The selection of such matches results in necessary expansions of the logic tree which in turn increase the depth of the tree. An example is shown in Figure 1.

The second phase of the technology mapping is the conversion of patterns into actual cells in the library. For an auto-dual library (i.e. AOIxxx (And-Or-Invert) present implies OAIxxx (Or-And-Invert) present), the organization is in layers:

- For layer $i$: the pattern is transformed into the dual form with a virtual inverter included.
- For layer $i + 1$: the inverters of the previous layers are absorbed and a direct mapping is performed. See Figure 2 for an example.



(a) Initial tree



(b) Mapping requiring two expansions

(c) Mapping requiring no expansion

FIGURE 1 Expansion requirements of selected mappings [7]. (Reproduced with permission of the authors).

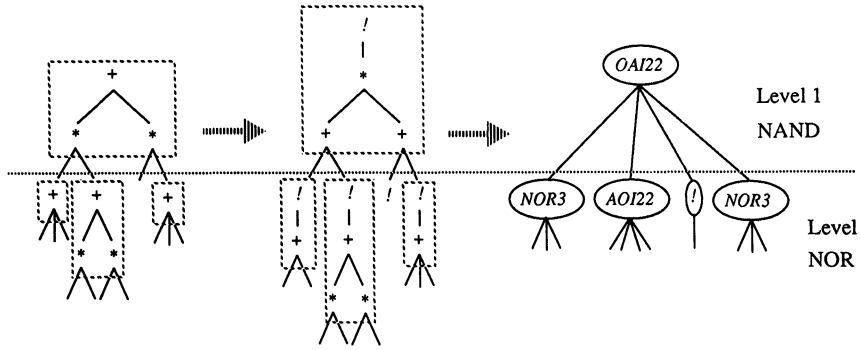



FIGURE 2 Second phase-conversion into cells [7]. (Reproduced with permission of the authors).

A similar approach, with modifications, is used if the library is not auto-dual.

While this approach is computationally efficient and yields results with good area figures, depth-reduction may be an insufficient criterion for other types of optimization problems such as delay minimization or area minimization under delay constraints. Furthermore, the key benefits of optimal tree covering using dynamic programming are lost.

## III. CONSTRUCTIVE MATCHING

Constructive matching is a **tree-based** matching algorithm capable of identifying any logic function that can be represented as a tree. In practice, however, the size of matched patterns must be bound by technology considerations and/or by feasibility of the corresponding logic cells. This section describes the constructive matching algorithm in detail, as well as the representation scheme for size limitations.

### 3.1 Cell Library Representation

The target *library* of cells is characterized by four simple parameters. These are the maximum number of n-type transistors in series $s$, the maximum number of p-type transistors in series $p$, the maximum number of levels of logic realized by a single gate $l$, and the maximum number of gate inputs $n$. The values of $s$ and $p$ are determined by technological factors. The logic level $l$ is defined as the depth of the logic tree (without redundant operators) minus one. (The leaves of the tree are usually not included in the calculation of the logic level; therefore, the depth of a tree (without redundant operators) is one larger than the logic level of the function it represents). The value of $l$ is not restricted by the technology, but may be restricted by the capabilities of the cell compiler. The number of inputs per cell $n$ may be restricted due to layout considerations: An exceedingly large number of inputs to a cell may introduce routing congestion during layout. Note that a restriction on $l$ and $n$ is implicitly placed by defining $s$ and $p$. For example, for $(s, p) = (4, 4)$ the maximum $l$ is 6, and the maximum $n$ is 16.

### 3.2 Constructive Matching Algorithm

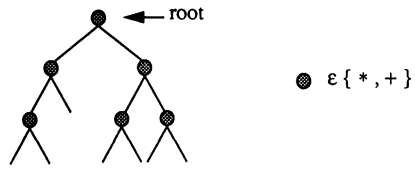
Definition—Given a tree ($T$) to be matched, a *candidate_pattern* is defined as a subtree of $T$ with the same root as $T$. A *candidate_pattern* is comprised of *seed_nodes* and *incremental_nodes*. The *incremental_nodes* are all situated at the leaf level of the *candidate_pattern*. Refer to Figure 3 for an illustration of the terms used.

Definition—A *matched_pattern* is defined as a *candidate_pattern* which corresponds to a physical CMOS complex gate which does not exceed the $(s, p, l, n)$ constraints.

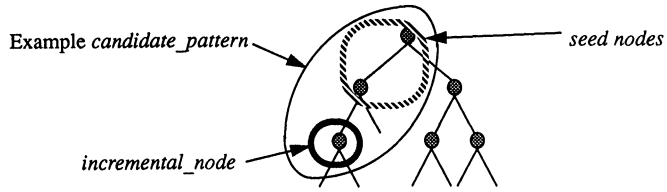
Definition—A *dual_matched_pattern* is defined as a *candidate_pattern* whose dual corresponds to a physical CMOS complex gate which does not exceed the $(s, p, l, n)$ constraints. (Note: the dual of a tree $T$ is another tree $T^d$ whose topology is identical to $T$ but in which all the AND (OR) operators have been replaced by OR (AND) operators).

Definition—Given a *matched_pattern* $P$, a *descendant_node* of $P$ is defined as a node which is an immediate child of an *incremental_node* in $P$ but which is not itself a node in $P$.

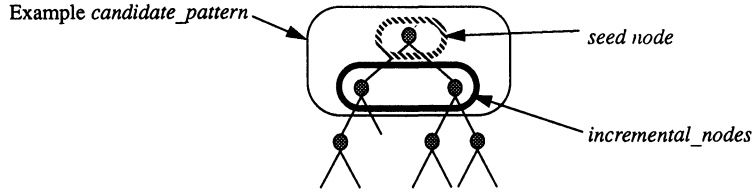
Given the above definitions, constructive matching is a recursive algorithm which is initially invoked with a candidate pattern comprised of only the root of the tree. It is assumed that the tree to be matched as well as the $(s, p, l, n)$ parameters are defined as




(a) Initial tree to be mapped




(b) Example *candidate_pattern* with two *seed nodes* and one *incremental node*




(c) Example *candidate_pattern* with one *seed_node* and two *incremental_nodes*

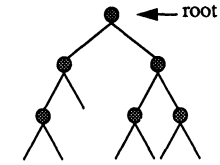
FIGURE 3 Example *candidate_patterns* comprised of *seed_nodes* and *incremental_nodes*. A *candidate_pattern* becomes a *matched_pattern* if it corresponds to a physical CMOS complex gate which does not exceed the $(s, p, l, n)$ parameters.

global variables. In the simplified description that follows, constructive matching prints a list of matched patterns as they are found. In the actual implementation of the algorithm, *matched_patterns* are stored in a match tree. Each node in this tree represents a *matched_pattern*. Each node stores the cost of a given match as well as the cost of its dual and only contains incremental information about how this match is constructed from its parent node.

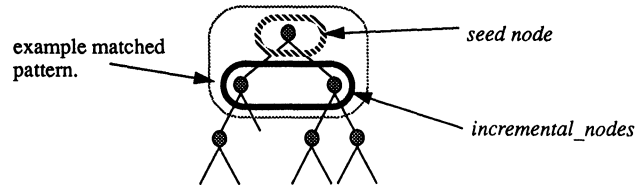
```
Algorithm constructive_matching (candidate_pattern);
    if candidate_pattern is a matched_pattern then
        print candidate_pattern;
        expand(candidate_pattern);
        if candidate_pattern is a dual_matched_pattern then
        print dual(candidate_pattern);
        end if
    else if candidate_pattern is a dual_
        matched_pattern then
        print dual (candidate_pattern);
        expand(candidate_pattern);
    end if
end {constructive_matching}
Function expand(pattern);
    new_candidate_list = generate all possible candidate patterns resulting from pattern;
    for each new_pattern in new_candidate_list do
        invoke constructive matching (new_pattern);
    end for each
end {expand}
```

The procedure used to generate new *candidate_patterns* from a *matched_pattern* is illustrated in Figure 4. It is important to note that in order to avoid repetition between matches found, only the *incremental_nodes* in a *matched_pattern* are used for expansion. Given a *matched_pattern*, with *b descendant_nodes*, there are $(2^b - 1)$ combinations of the *descendant_nodes* which when merged with the *matched_pattern* itself can form new *candidate_patterns*. The original *matched_pattern* comprises the seed nodes of the new *candidate_patterns* whereas the *descendent_nodes* become the *incremental_nodes* in each *candidate_pattern*.

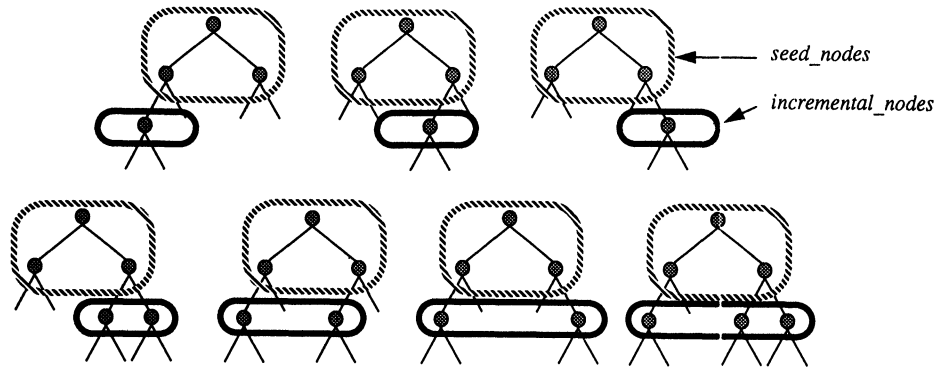
The behavior of the constructive matching algorithm is illustrated in Figure 5. The example tree is




(a) Initial tree to be mapped




(b) Example *matched_pattern* with b=3 *descendant_nodes*




(c) $2^b$ -1 = 7 new *candidate_patterns* when b=3.

FIGURE 4 Expansion of a *matched_pattern* into new *candidate_patterns*.

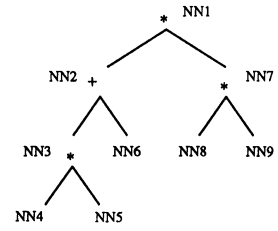
matched without any restrictions on the size of the cell family. The initial tree is shown in Figure 5(a), whereas Figure 5(b) shows the matches found and their organization in a match tree.

### 3.3 Time and Space Complexity

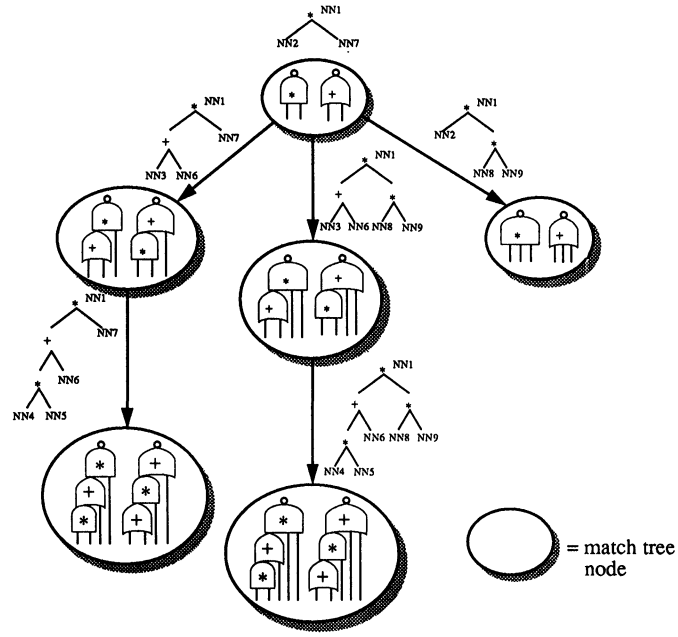
The time complexity of the unbounded constructive matching algorithm, i.e. when the size of the cell library in unlimited, is $O(n2^{n/2})$ in the worst case of a perfectly balanced tree comprised of $n$ nodes. The number of matches at the root of the tree is dominated by the number of matches found at level $(\log(n) - 1)$, just above the leaf level. At that level, the worst case matched_pattern has $n/2$ distinct descendant nodes, leading to $2^{n/2}$ possible new *candidate_matches*.

In practice however, the number of matches found at a given node is bounded by the $(s, p, l, n)$ parameters. For given values of $(s, p, l, n)$, there is a possibly very large but constant maximum number of matches at each node. This makes the complexity of the constrained constructive matching algorithm $O(n)$ with respect to the size of the tree, with a possibly very large constant factor.

The space required by the algorithm is a function of the number of matches found which may not exceed the size of the largest feasible cell pattern at each node. Because the matches found can be stored using a match tree, the storage requirement is a product of the number of nodes in the tree times the size of the largest possible pattern in the target cell family. This is much better than the space requirement of library-based approaches for which the storage requirement is a product of the size of the tree multiplied by the size of the library.




(a) Logic Tree




(b) Matches found stored in match tree

FIGURE 5 Match Tree produced by Constructive Matching Algorithm (From: C. Liem, M. Lefebvre, "A Constructive Matching Algorithm for Cell Generator-Based Technology Mapping", *Proc. 1992 IEEE Symposium on Circuits and Systems*, 1992, pp. 2965–2968. © 1992 IEEE).

## IV. CONSTRUCTIVE MATCHING BASED TECHNOLOGY MAPPING

This Section presents an overview of our prototype technology mapper based on constructive matching.

### 4.1 Tree Partitioning

Beginning with a Boolean Network which has been performance-optimized through logic minimization techniques using MisII [11], the logic is parsed into a forest of trees, each comprised of primitive AND, OR, and NOT logic functions. The DAG is partitioned at multiple fanout points, then NOT functions are *pushed* to the leaf level by repeated applications of DeMorgan's Theorem. Having the inversions only at the leaves allows for easy matching of large complex cells. Redundant operators (i.e. the same operators directly joined by an edge) are then removed.

### 4.2 Balanced Binary Decomposition

Each tree of primitive functions is then decomposed into a binary tree. If the arrival times of all the input signals are the same, we favor a balanced decomposition technique [14, 15, 16] to produce trees which have the fewest logic levels possible. If the input arrival times are skewed, then an unbalanced decomposition is used which attempts to place the latest arriving signals as close to the root as possible.

### 4.3 Constructive Matching

At this point, the constructive matching algorithm is applied to each node of each tree in order to identify feasible cells from the target cell family. Each *matched_pattern* is assigned an area cost and a delay cost. For area, the cost of a cell is simply calculated as the number of inputs to the cell plus one. This is to reflect the number of grids the layout of the cell would use. The delay through a cell is estimated considering the number of series-connected transistors, determined by the values of $s$ and $p$. The value of $s$ determines the maximum possible fall transition delay, while the value of $p$ determines the maximum possible rise transition delay. Quantitative delay information is determined by SPICE simulations. Delay models are comprised of an intrinsic delay and a fanout coefficient for each combination of the $s$ and $p$ parameters.

### 4.4 Covering

The covering algorithm uses a dynamic programming procedure. It begins at the leaves of a tree and traverses toward the root node, choosing the best implementation at each node. The best match at a node is determined by the current cost metric. Delay is calculated as the greatest arrival time at the inputs (leaves) of the gate plus the delay associated with the gate itself. Area is calculated as the total area at the leaves of the gate plus the area of the gate itself. At each node (including the root node), two solutions are retained corresponding to the best possible implementation of each phase of the output signal.

### 4.5 Tree-boundary phase assignment

The covering algorithm produces locally optimal solutions, in linear time, for each tree of the logic network. This may produce sub-optimal solutions at the graph level. In particular, until the phases of signals across tree boundaries are known, it is difficult to estimate the loading on the output node of each tree. Without an accurate loading estimate, optimal covering of each tree cannot be assured. In order to partially overcome this limitation, we have implemented a tree-boundary phase assignment heuristic. For each phase of the output signal, a number of different solutions are generated corresponding to a range of anticipated loading conditions. Then, a user-specified number of combinations of tree covers are compared for relative cost.

## V. SUBJECTIVE EVALUATION

### 5.1 Comparison to Other Approaches

Constructive matching can be distinguished from both **library-based** and **cell generator-based** approaches to technology mapping.

When compared with **library-based** approaches, it enables the use of much larger cell libraries, a feature which has a favorable impact on the area of technology mapped networks and which may yield performance enhancements with deep sub-micron CMOS technologies. Furthermore, while large libraries may be represented efficiently as sets of data strings, as is the case in existing **library-based** systems [1], there is much repeated effort involved in matching against a collection of very large patterns that may differ only slightly. This repetitive matching of common sub-patterns is avoided by our approach since new matches are built incrementally on existing ones.

On the other hand, the key benefit of constructive matching over its **generator-based** competitors is the ability to perform locally optimal network covering using dynamic programming, a feature that is lost with existing **cell generator-based** technology mappers.

### 5.2 Limitations of the Approach

The limitations of constructive matching are those limitations that are inherent to all **tree-based** mapping techniques using dynamic covering. For instance, some logic elements that cannot be described as a tree must be excluded from the mapping process. Exclusive OR gates and multiplexers are the two most important classes of such elements. It is possible to envisage an hybrid matching technique which would combine constructive matching for the identification of possibly very large series-parallel logic structures with a more general **graph-based** matching technique which would concentrate on relatively small patterns such as multiplexors and exclusive OR gates. Such a matching approach would, however, mandate the use of a **graph-based** covering algorithm.

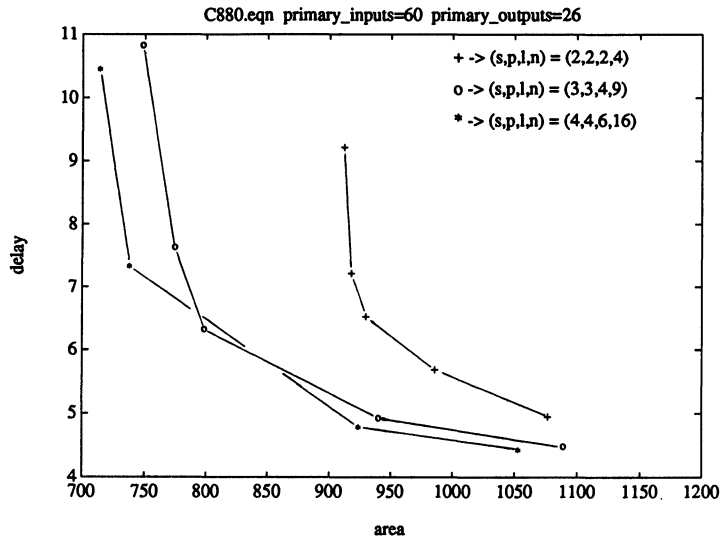



FIGURE 6 Design space graph for the C880 benchmark circuit. Each point on the graph corresponds to one feasible realization of the C880 benchmark circuit in a two-dimensional delay/area trade-off space. This graph is the result of repeated synthesis and mapping for the C880 circuit using various optimization scripts in MisII and objective functions during mapping. Three symbols are used to represent possible circuits corresponding to each of the three target libraries.

## VI. EXPERIMENTAL RESULTS

### 6.1 Experimental Procedure

Experimental results are based on logic networks from the 1991 MCNC (Microelectronics Center of North Carolina) International Workshop on Logic Synthesis benchmark set. These multi-level logic benchmarks were prepared by various logic minimization scripts performed by MisII before technology mapping [11]. A 1.2 $\mu$m CMOS technology was used to determine the delay characteristics. Three cell families are used for experimentation:

- Family 1: $(s, p, l, n) = (2, 2, 2, 4)$, comprised of 7 cells,
- Family 2: $(s, p, l, n) = (3, 3, 4, 9)$, comprised of 87 cells, and
- Family 3: $(s, p, l, n) = (4, 4, 6, 16)$, comprised of 3503 cells.

### 6.2 Design Space Exploration

The impact of the size of the cell library on the quality of technology mapped circuits can be visualized by considering the design space graph of a typical benchmark circuit. Each point on the graph corresponds to one feasible realization of the circuit under consideration in a two-dimensional space representing the delay/area trade-off space for that circuit. An example design space graph for the C880 benchmark circuit is shown in Figure 6. This graph is the result of repeated synthesis and mapping for the C880 circuit using various optimization scripts in MisII and objective functions during mapping. Three symbols are used to represent possible circuits corresponding to each of the three target libraries. It is clear from this design space graph that the size of the family has a significant impact on the quality of technology-mapped circuits, particularly in terms of area. Indeed, the contour of feasible solutions for the (4, 4, 6, 16) cell family is situated to the left (area savings) and slightly below (small delay savings) the contour of feasible solutions of the other two cell families.

### 6.3 Area Optimization

The results generated for all the benchmark circuits under area minimization are summarized in Table 2. Table 2 contains the final area and delay attributes of each benchmark circuit as well as the amount of CPU time required for technology mapping. Both the technology independent optimization phase and technology mapping are aimed at area minimization. The improvement provided by Family 3 over Family 2 is as high as 20% for some circuits and is 5.2% on average. Note that Family 2 is the largest "complete" family that can be accommodated by existing **library-based** approaches to technology mapping. A

TABLE II
Area optimization for different sized cell families

| Network | Family 1 (2, 2, 2, 4) area | Family 1 (2, 2, 2, 4) delay(ns) | Family 2 (3, 3, 4, 9) area | Family 2 (3, 3, 4, 9) delay(ns) | Family 3 (4, 4, 6, 16) area | Family 3 (4, 4, 6, 16) delay(ns) | Family 1 to 2 Δ area | Family 1 to 3 Δ area |
|---|---|---|---|---|---|---|---|---|
| apex6 | 1701 | 5.093 | 1459 | 3.677 | 1391 | 6.139 | −14.2% | −18.2% |
| C1355 | 1024 | 6.959 | 998 | 7.222 | 1024 | 7.178 | −2.5% | 0% |
| C432 | 578 | 10.503 | 502 | 16.952 | 466 | 17.786 | −13.2% | −19.4% |
| C499 | 1024 | 6.959 | 998 | 7.222 | 1024 | 7.178 | −2.5% | 0% |
| C880 | 912 | 9.208 | 748 | 10.828 | 714 | 10.456 | −18.0% | −21.7% |
| cm151a | 46 | 1.660 | 42 | 1.411 | 34 | 1.831 | −8.7% | −26.1% |
| con1 | 43 | 1.010 | 37 | 1.100 | 31 | 1.179 | −14.0% | −27.9% |
| cordic | 169 | 1.825 | 141 | 1.881 | 131 | 2.341 | −16.6% | −22.5% |
| count | 305 | 6.320 | 247 | 6.381 | 247 | 6.444 | −19.0% | −19.0% |
| dalu | 2514 | 14.445 | 2180 | 17.346 | 2042 | 17.243 | −13.3% | −18.8% |
| duke2 | 1083 | 4.303 | 831 | 5.645 | 762 | 6.292 | −23.3% | −29.6% |
| example2 | 771 | 4.487 | 688 | 4.772 | 671 | 4.810 | −10.8% | −13.0% |
| f51m | 284 | 8.915 | 215 | 10.966 | 206 | 13.238 | −24.3% | −27.5% |
| i4 | 466 | 1.804 | 330 | 1.460 | 298 | 1.770 | −29.2% | −36.1% |
| i5 | 396 | 3.489 | 396 | 3.489 | 396 | 3.489 | 0% | 0% |
| inc | 436 | 3.287 | 331 | 3.800 | 311 | 5.017 | −24.1% | −28.6% |
| mux | 103 | 1.639 | 96 | 2.026 | 74 | 1.879 | −6.8% | −28.2% |
| pcler8 | 199 | 4.151 | 173 | 4.893 | 171 | 4.742 | −13.1% | −14.1% |
| squar5 | 187 | 2.348 | 136 | 3.118 | 132 | 2.831 | −27.3% | −29.4% |
| x1 | 733 | 2.531 | 553 | 2.894 | 515 | 2.894 | −24.6% | −29.7% |
| z4ml | 96 | 2.649 | 82 | 2.789 | 78 | 3.014 | −14.6% | −18.8% |
| | | | | | | | −15.2% | −20.4% |

(From: C. Liem, M. Lefebvre, "A Constructive Matching Algorithm for Cell Generator-Based Technology Mapping", *Proc. 1992 IEEE Symposium on Circuits and Systems*, 1992, pp. 2965–2968. © 1992 IEEE)

TABLE III
Delay optimization for different sized cell families

| Network | Family 1 (2, 2, 2, 4) area | Family 1 (2, 2, 2, 4) delay(ns) | Family 1 (2, 2, 2, 4) cpu(s) | Family 2 (3, 3, 4, 9) area | Family 2 (3, 3, 4, 9) delay(ns) | Family 2 (3, 3, 4, 9) cpu(s) | Family 3 (4, 4, 6, 16) area | Family 3 (4, 4, 6, 16) delay(ns) | Family 3 (4, 4, 6, 16) cpu(s) | Family 1 to 2 Δ delay | Family 1 to 3 Δ delay |
|---|---|---|---|---|---|---|---|---|---|---|---|
| apex6 | 1974 | 3.476 | 170.8 | 1790 | 3.119 | 185.6 | 1768 | 3.119 | 684.9 | −10.3% | −10.3% |
| C1355 | 1148 | 5.053 | 108.6 | 1148 | 5.028 | 128.0 | 1164 | 4.988 | 220.1 | −0.5% | −1.3% |
| C432 | 948 | 9.227 | 102.5 | 1002 | 9.478 | 53.9 | 866 | 8.884 | 399.9 | +2.7% | −3.7% |
| C499 | 1124 | 4.816 | 114.0 | 1124 | 4.816 | 121.3 | 1148 | 4.777 | 127.6 | 0% | −0.8% |
| C880 | 1077 | 4.935 | 106.4 | 1089 | 4.482 | 108.2 | 1053 | 4.430 | 173.1 | −9.2% | −10.2% |
| cm151a | 64 | 1.570 | 13.8 | 71 | 1.400 | 75.4 | 71 | 1.400 | 76.8 | −10.8% | −10.8% |
| con1 | 45 | 0.954 | 58.5 | 43 | 0.904 | 52.1 | 43 | 0.904 | 56.0 | −5.2% | −5.2% |
| cordic | 190 | 1.540 | 54.7 | 190 | 1.505 | 54.2 | 180 | 1.489 | 95.3 | −2.3% | −2.3% |
| count | 607 | 3.056 | 75.2 | 509 | 2.722 | 70.5 | 499 | 2.790 | 87.7 | −10.9% | −8.7% |
| dalu | 3184 | 10.809 | 297.3 | 2852 | 10.367 | 291.2 | 2794 | 10.308 | 440.7 | −4.1% | −4.6% |
| duke2 | 1450 | 4.146 | 23.9 | 1484 | 3.926 | 411.5 | 1434 | 3.905 | 481.5 | −5.3% | −5.8% |
| example2 | 996 | 3.208 | 94.9 | 952 | 3.199 | 114.8 | 952 | 3.208 | 106.9 | −0.3% | 0% |
| f51m | 351 | 2.488 | 12.9 | 343 | 2.420 | 76.2 | 311 | 2.249 | 107.5 | −2.7% | −9.6% |
| i4 | 766 | 1.300 | 19.3 | 510 | 1.077 | 93.3 | 426 | 1.040 | 120.0 | −17.2% | −20.0% |
| i5 | 1580 | 2.461 | 23.7 | 1121 | 2.326 | 96.9 | 1109 | 2.326 | 104.1 | −5.5% | −5.5% |
| inc | 456 | 3.093 | 7.2 | 502 | 2.935 | 59.6 | 490 | 2.935 | 66.6 | −5.1% | −5.1% |
| mux | 113 | 1.625 | 66.7 | 109 | 1.586 | 56.3 | 108 | 1.496 | 69.4 | −2.4% | −7.9% |
| pcler8 | 390 | 2.496 | 63.6 | 362 | 2.395 | 76.2 | 340 | 2.395 | 75.9 | −4.1% | −4.1% |
| squar5 | 219 | 2.339 | 57.4 | 211 | 2.237 | 63.4 | 205 | 2.237 | 58.8 | −4.4% | −4.4% |
| x1 | 801 | 2.196 | 25.8 | 809 | 2.108 | 99.1 | 787 | 2.167 | 117.7 | −4.0% | −1.3% |
| z4ml | 175 | 1.638 | 58.1 | 159 | 1.567 | 53.7 | 139 | 1.457 | 77.8 | −4.3% | −11.1% |
| | | | | | | | | | | −5.0% | −6.3% |

(From: C. Liem, M. Lefebvre, "A Constructive Matching Algorithm for Cell Generator-Based Technology Mapping," *Proc. 1992 IEEE Symposium on Circuits and Systems*, 1992, pp. 2965–2968. © 1992 IEEE)

complete family of cells is a family that contains all feasible cells with a set maximum number of transistors in series between and output node and a supply rail.

### 6.4 Delay Optimization

The results generated for all the benchmark circuits with delay optimization as the objective function are summarized in Table 3. Table 3 contains the final area and delay attributes of each circuit as well as cpu time information. This time the objective function is delay minimization. Results show a modest improvement in delay for the largest cell family (Family 3; 3503 cells) over the 87-cell family (Family 2) which contains all the cells that would typically be found in a standard cell library. This modest improvement is expected for a low performance technology such as the one used. The savings can be attributed to the ability of cells with a high level to reduce the depth of the logic network.

## VII. CONCLUSIONS

Constructive matching bridges the gap between library-based and cell generator-based technology mapping algorithms. It enables the use of very large cell families that are not feasible with library-based approaches while retaining the ability to perform optimal network covering using dynamic programming, a feature that is lost with existing cell generator-based technology mappers. The time and space complexities of the bounded constructive matching algorithm are $O(n^*K)$ where $n$ is the size of the logic network and $K$ is the size of the largest feasible cell pattern in the target family. The empirical results demonstrate the increased quality of the technology-mapped circuits when larger cells are available. It is expected that progress in fabrication technology (sub-micron technologies) and advances in logic minimization capability will further these trends.

#### Acknowledgments

This work has been funded in part by: the Natural Sciences and Engineering Research Council of Canada, Bell-Northern Research Limited, Ottawa, and the Canadian Microelectronics Corporation. The authors would like to thank Frederic Mailhot for a number of helpful discussions and Kurt Keutzer for general comments on the constructive matching algorithm. The authors are also grateful to John Chinneck for a critical review of the manuscript before submission. Helpful comments made by the reviewers are also acknowledged.

#### References

[1] K. Keutzer, "DAGON: Technology Binding and Local Optimization by DAG Matching", *Proceedings of the 24th ACM/IEEE Design Automation Conference*, June 1986, pp. 79–85.

[2] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, A. Wang, "Technology Mapping in MIS", *Proceedings of the International Conference on Computer-Aided Design*, 1987, pp. 116–119.

[3] G. DeMicheli, D. Ku, F. Mailhot, T. Truong, "The Olympus Synthesis System", *IEEE Design & Test of Computers*, vol. 7, pp. 37–53, October 1990.

[4] F. Mailhot and G. DeMicheli, "Technology Mapping Using Boolean Matching and Don't Care Sets", *Proceedings of the European Design Automation Conference*, 1990, pp. 212–216.

[5] T. Sakurai and A.R. Newton, "Delay Analysis of Series-Connected MOSFET Circuits", *IEEE Journal of Solid-State Circuits*, vol. 26, pp. 122–131, Feb. 1991.

[6] M.R.C.M. Berkelaar, J.A.G. Jess, "Technology Mapping for Standard-Cell Generators", *Proceedings of the International Conference on Computer-Aided Design*, 1988, pp. 470–473.

[7] K. Sakouti, G. Saucier, "A fast and effective technology mapper on an autodual library of standard cells", *Proceedings on the IFIP Working Conference on Logic and Architecture Synthesis*, May 1990, pp. 137–150.

[8] M. Crastes, K. Sakouti, G. Saucier, "A Technology Mapping Method Based On Perfect And Semi-Perfect Matchings", *Proceedings of the 28th ACM/IEEE Design Automation Conference*, June 1991, pp. 93–98.

[9] R. Ramirez Ortiz, M. Lefebvre, "Technology Mapping Algorithms for NORA Dynamic Logic Circuits", in *Proc. of The European Conference on Design Automation with The European Event in ASIC Design*, 1993, pp. 310–314.

[10] C. Liem, M. Lefebvre, "A Constructive Matching Algorithm for Cell Generator-Based Technology Mapping", *Proc. 1992 IEEE Symposium on Circuits and Systems*, 1992, pp. 2965–2968.

[11] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, A. Wang, "MIS: A Multiple-Level Logic Optimization System", *IEEE Transactions on Computer-Aided Design*, Vol. CAD-6, pp. 1062–1081, Nov. 1987.

[12] R. Rudell, *Logic Synthesis for VLSI Design*, Ph. D. Dissertation, U.C. Berkeley, Memorandum No. UCB/ERL M89/49, 26 April 1989, 223 p.

[13] G. Hachtel, R. Jacoby, C. Morrison, "TECHMAP: Technology Mapping with Delay and Area Optimization", *Proceedings of the International Workshop on Logic and Architecture Synthesis for Silicon Compilers*, Grenoble, May 1988.

[14] K. Keutzer, M. Vancura, "Timing Optimization in a Logic Synthesis System", *Proceedings of the International Workshop on Logic and Architecture Synthesis for Silicon Compilers*, Grenoble, May 1988.

[15] P. Paulin, "Logic Decomposition Algorithms for the Timing Optimization of Multi-Level Logic", *Proceedings of the International Conference on Computer Design*, Cambridge, MA, October 1989.

[16] K. Bartlett, R. Brayton, G. Hatchel, R. Jacoby, C. Morrison, R. Rudell, A. Sangiovanni-Vincentelli, A. Wang, "Multilevel Logic Minimization Using Implicit Don't Cares", *IEEE Transactions on Computer-Aided Design*, Vol. 7, pp. 723–740, June 1988.

[17] H. Touati, C. Moon, R. Brayton, "Performance-Oriented Technology Mapping", *Proceedings of the sixth MIT VLSI Conference*, 1990, pp. 79–97.

[18] W.N. Li, A. Lim, P. Agrawal, S. Sahni, "On the Circuit Implementation Problem", in *Proc. of 29th ACM/IEEE Design Automation Conference*, 1992, pp. 478–483.

[19] K. Chaudhary, M. Pedram, "A Near Optimal Algorithm for Technology Mapping Minimizing Area under Delay Constraints", in *Proc. of 29th ACM/IEEE Design Automation Conference*, 1992, pp. 492–498.

## Biographies

**MARTIN LEFEBVRE** received the B.A.Sc. in electrical engineering from the University of Ottawa in 1982, and the M.Eng. and Ph.D. degrees in Electronics from Carleton University, Ottawa, in 1986 and 1989, respectively. Between 1982 and 1989 he was a member of scientific staff at Bell-Northern Research, Ottawa, first with the digital switching division and later with the advanced technology research group. Since July 1989, he has been with the Department of Electronics, Carleton University, as an assistant professor. His research interests lie in the area of optimization-based VLSI systems design, the current emphasis of which is on low power/high speed interfaces for mixed voltage systems.

**CLIFFORD LIEM** is a member of the Embedded Systems Technology group at Bell-Northern Research Inc. in Ottawa, Canada. His interests include microcode synthesis, hardware/firmware codesign, architecture and logic synthesis. He received his B.Sc. in theoretical physics from St. Francis Xavier University in 1989 and his M.Eng. in electrical engineering from Carleton University in 1991.