A GENERALIZED CONTROL STRATEGY FOR RULE-BASED SYSTEMS

by

Gregory S. Whitcomb

Memorandum No. UCB/ERL M88/46

6 July 1988

A GENERALIZED CONTROL STRATEGY FOR RULE-BASED SYSTEMS

by

Gregory S. Whitcomb

Memorandum No. UCB/ERL M88/46

6 July 1988

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A GENERALIZED CONTROL STRATEGY FOR RULE-BASED SYSTEMS

by

Gregory S. Whitcomb

Memorandum No. UCB/ERL M88/46

6 July 1988

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Contents

List of Figures

List of Tables

Abstract

Many difficult problems do not lend themselves to recipe-like solutions characteristic of the traditional algorithmic approach. An effective alternative for solving these problems is the rule-based approach. Rule-based systems often utilize a state-space problem formulation in which rules are used to move between states in search of the goal, or solution, states. Directing this search is the responsibility of the control strategy. In this report the use of probabilistic hill-climbing techniques as the basis for a powerful, generalized control strategy is described. Probabilistic hill-climbing overcomes many of the inherent weaknesses of other control strategies which complicate the rule-based solution to difficult problems. One such problem is logic synthesis—the process of converting a functional description of a digital circuit into an optimal implementation. A rule-based system for logic synthesis has been developed to study and assess the quality of the PHC control strategy. The quality of the solutions generated by the system indicate the effectiveness of this technique.

Acknowledgments

I greatly acknowledge my research advisor, Professor Richard Newton, for his encouragement, advice, and ideas during the course of this project. I would also like to thank Richard Rudell for providing answers to numerous questions about MIS and for many interesting discussions on logic synthesis.

Also acknowledged is Professor Alberto Sangiovanni-Vincentelli for the helpful comments and suggestions he provided concerning the project and report. Srinivas Devadas suffered through an early draft of the report and I greatly appreciate the feedback he provided.

This project was funded by the Defense Advanced Research Projects Agency under contract N00039-87-C-0182. Their support is acknowledged.

I must also thank the many users of our VAX machines EROS and IC who had to tolerate the loss of uncountable CPU hours during the development of OPAL.

Finally, I thank my wife, Terri, for her never-ending source of encouragement and support.

Chapter 1

Introduction

1.1 Overview

Many difficult problems do not lend themselves to recipe-like solutions characteristic of the traditional algorithmic approach. Combinatorial complexity, numerous, conflicting constraints, and limited understanding of the problems make them intractable. An alternative to the traditional algorithmic approach is the rule-based approach[WH78,DK76]. Rule-based systems are data-driven—that is, the order of steps in solving the problem are determined. in part, by the state of the problem and are not fixed as in a recipe. Because knowledge about the problem is encapsulated in the *rules*, rule-based systems can easily incorporate new knowledge by introducing new rules. One other important characteristic of rule-based systems is that they separate rule knowledge from the *control structure*—the mechanism that determines how rules are selected. Because of this separation, many rule-based systems can effectively utilize general-purpose control strategies. This report describes the use of probabilistic hill-climbing (PHC) techniques in rule-based system control strategies. PHC techniques have demonstrated success in obtaining near optimal solutions for combinatorial optimization problems. Their effectiveness is demonstrated by applying them to a rule-based system for logic synthesis. Logic synthesis involves the transformation of a functional description of a digital circuit into an optimal implementation consisting of components and their interconnections. This topic has received much

attention[McC56,HCO74,BHH*82,DBG*84,GBdH86] and programs for logic synthesis [BRSW87,dC85] are available. Since these programs present different approaches to the problem, they can serve as a basis for determining the quality of the PHC rule-based approach.

1.2 Rule-Based Systems

Rule-based systems contain three distinguishing components: a set of if-then, or *antecedent-consequent*, pairs known as rules; a separate *database*, or *working-memory*, containing problem information; and a *control strategy, or inference engine*, defining the order and selection of rules to be applied to the database. Presented in Figure 1.1 is the structure common to all rule-based systems. The most common type of rule-based system used in problem-solving is the production system. These systems can be characterized by a *recognize-act* cycle [WH78]. The recognize phase determines which rules can be applied to the database by matching rule antecedents with database elements. If more than one rule is applicable, the control strategy must determine which rule to "fire"—a process known as conflict resolution. The act phase involves executing the action indicated by the selected rule's consequent. This sequence is repeated until the solution is obtained. Only rule-based systems with these characteristics will be considered in this report.

Problems to be solved by rule-based systems are often formulated as state-space searches. A state-space is the set of all possible configurations of the problem being solved. The problem as presented to the system defines the initial state; one or more goal states define the solution. The control strategy, then, directs a search on a graph whose nodes are states and edges are rules. Since a wide variety of problems can be formulated in this manner, rule-based systems define a powerful problem-solving paradigm.

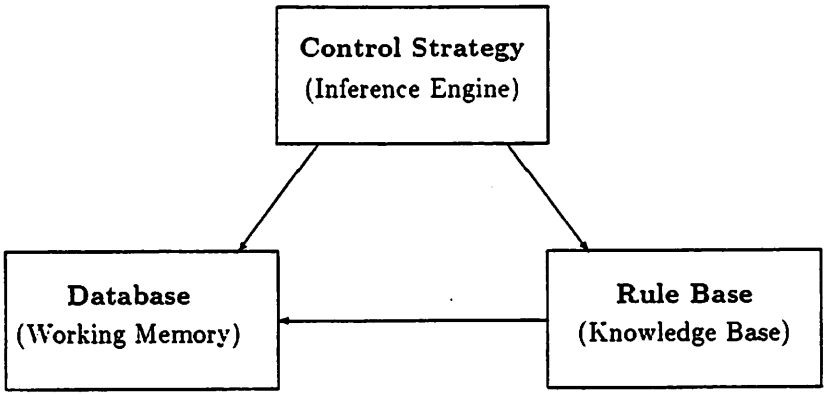

Figure 1.1: Structure of Rule-based Systems

1.3 Controlling the State-Space Search

Most interesting problems have a state-space which grows exponentially with the size of the problem being solved. The selection of the proper problem representation can dramatically reduce the extent of the state-space [Ama68]. However, bounding the problem-solving process in both space and time necessitates a limited search. This task is accomplished through the use of knowledge about the problem. Such knowledge can be expressed in the rule base or it can exist in an intelligent control strategy.

The expert system school of thought has been that "in the knowledge is the power" and "the interesting action arises from the knowledge base, not the inference engine." [Feg79]. This approach emphasizes the use of high-level *macromoves* which can eliminate unproductive paths and provide a more rapid traversal of the state-space than can simpler rules [BAE*83]. Unfortunately, little information is available concerning how various rule-based expert systems control their search with knowledge [Hay85]. Thus, it is unclear as to how effective rule knowledge by itself can limit the state-space search. In addition, the number of macromove rules necessary for a particular problem is often quite large. With so many rules, determining their sufficiency for solving all problem instances becomes very difficult if not impossible.

Also, the efficiency of rule matching and the complexity of conflict resolution become a major concern.

The other possibility is to incorporate knowledge into the control strategy. Approaches to this task include rule orders [MF78], metarules [DK76], state transitions, and feedback [Zuc78]. These methods tend to be inflexible for the general case of the problem—working well on some examples but not on others. Also, the addition of new knowledge to these control strategies often requires a deep insight into both the problem being solved and the operation of the control mechanism. In addition to these methods, knowledge can exist within *evaluation functions*. These functions provide a numeric indication of the desirability, or cost, of a state. Rule desirability is thus based on the state it generates given the current state. Using this information, useless searching can be reduced. In situations where rule desirability is a dynamic property of the state-space, evaluation functions are essential.

If evaluation functions were perfect, the correct path to the goal state would be obvious and no search would be required. However, this situation is most often not the case and the evaluation function must be combined with an appropriate search procedure. The appropriateness of a particular search procedure is defined by characteristics of the desired solution and the state-space graph. For some problems, a minimal path to the goal state is desired. Branch-and-bound procedures are useful for this purpose. For other problems, only obtaining the solution state is important. In this case, general-purpose procedures such as steepest-descent and best-first search can be used.

Steepest-descent[1] is based on the assumption that the evaluation function provides a good indication of how close a particular state is to the goal. It is often called a greedy algorithm because rules are ordered according to the desirability of the states they generate and the best choice is always selected. The other possible states are not stored for later consideration, and thus memory utilization is minimal. Unfortunately,

[1]Steepest-descent here refers to the case where the purpose of the search is to minimize the evaluation function. The term *hill-climbing* is sometimes used when such methods are used on a function where the maximum is required. This use of "hill-climbing" should not be confused with the hill-climbing of PHC methods—here, "hill-climbing" refers to a move away from the local minimum, for a function where the minimum solution is the goal.

there is no guarantee that the search will converge to the best solution, the global optimum. It is too easy for the algorithm to choose a path which leads to a locally optimal state for which no rule can be applied to improve the situation. One technique for overcoming this problem requires that the algorithm *backtrack* to a state which has a more promising path. Unfortunately, the amount of backtracking necessary cannot be determined in advance. Another technique is to explore a local space consisting of the states reachable by sequences of rules. Rules that appear undesirable at one level might provide for significant improvements at future levels. Once again however, the number of levels required cannot be determined in advance. In addition, this "look-ahead" technique suffers from exponential growth—often restricting sequences to only two or three rules.

The best-first search strategy overcomes locally optimal states by remembering the unselected states along the search path. When selecting the next configuration state, the algorithm selects the best choice from previously unselected states as well as those derivable from the current state. In so doing, the algorithm can always locate the globally optimal solution. Unfortunately, if many nearly equivalent paths exist, the memory requirement increases exponentially. This situation then requires that only a fraction of possible next states be maintained—thus, optimality is traded off against practicality. Another difficulty with best-first and other search procedures is coping with redundant, or reconvergent, paths in the state-space graph. If it is too costly to detect this situation, the search may waste a lot of time covering old territory. An additional concern is the efficiency of expanding a node completely—that is, determining the set of states which can be obtained by a rule instantiation on the current state. Procedures which require that a node be expanded completely before making a rule selection decision can waste a lot of time generating unfruitful states.

Problems which suffer from these characteristics have been restricted to concentrating the knowledge in the rule base and using a simplified control strategy with little search. Design and optimization problems often fall into this category. For example, WEAVER [Joo85] is a rule-based expert system which performs detailed routing of VLSI circuits. Its routing knowledge is concentrated in the rule base—

utilizing a simple control strategy of rule ordering and greedy search. According to Joobbani, the reason for adopting such a control strategy was that expert human designers do not utilize the simple backup-to-a-previous-state-and-continue approach characteristic of current search techniques. Instead, designers often work *forward* from their current state, taking short cuts to another state *without* considering the previous states and how they arrived at them. This report describes a control strategy which searches the state-space in this same manner but is able to overcome many of the difficulties associated with current search techniques. This control strategy utilizes the combinatorial optimization technique of probabilistic hill-climbing.

1.4 Probabilistic Hill-Climbing

Probabilistic hill-climbing (PHC) unlike steepest-descent, is able to escape locally optimal states by occasionally accepting moves which appear to move away from the goal [RS85]. Because PHC techniques have traditionally been applied to optimization and design problems, the evaluation function is called the cost function. Thus, moving away from the goal increases the cost of the system. The difference between various PHC techniques is the probability of moving to, or accepting, a state with a higher cost. Simulated annealing is the best-known PHC technique in the electronic CAD area and utilizes the Boltzmann probability distribution as its acceptance criterion [KGv83]. This distribution is used in statistical mechanics to model the atomic behavior of a material as it undergoes the annealing process. In the annealing process, displacements of atoms which increase the energy state are very likely to occur at high temperatures. As the material is cooled, these displacements are less likely and the material takes on atomic configurations with lower energy states. The minimum energy state (the desired result) is called a crystal. However, if the material is not cooled properly, the result is a higher energy state called a glass. In simulated annealing, changes in energy state of materials are analogous to changes in the cost function of an optimization process. Moves which increase the cost are likely at the beginning of the process and less likely as the process proceeds. The global optimum is the crystalline state and local optimums are glasses. With certain assumptions on

the rules governing the generation of moves and on the time spent at each temperature, simulated annealing has been proven to locate the globally optimal solution with probability 1 [RS85]. Probabilistic hill-climbing, in a manner similar to expert designers, continually moves forward from the problem state and does not consider previous actions. In addition, since PHC does not perform explicit backtracking or look-ahead, memory demand is proportional to problem size.

In problems such as integrated circuit layout, simulated annealing has achieved results superior to other techniques [KGv83,VK83,SS84,SS86,AJMS84,DN86]. and, in some cases, using comparable computing resources (time, memory) has achieved better results than algorithmic techniques [DN86]. Most applications of PHC techniques share a similar state-space formulation. First, a constructive procedure is used to generate an initial configuration. New configurations are derived by applying simple moves on randomly selected elements of the configuration. For partitioning and placement, these elements are circuit modules and the possible moves are interchange and relocation. These moves can be considered to be simple rules with null antecedents. By expanding the formulation to the more general case of rule-based systems, simulated annealing and other PHC techniques can be used to solve a much broader class of problems.

Probabilistic hill-climbing possesses features which overcome many of the deficiencies of current general-purpose search techniques. PHC techniques explore the state-space in much the same manner as backtracking and look-ahead approaches. However, the extent of backtracking and look-ahead is controlled by the process and problem states and is not specified in advance. PHC and best-first search share the characteristic of being able to locate the globally optimal solution given sufficient time. However, unlike best-first and standard hill-climbing approaches, PHC search is not hindered by reconvergent paths. The similarities between current general-purpose search and PHC techniques indicate their potential usefulness as control strategies in rule-based systems. Furthermore, problems which were difficult to solve using standard hill-climbing or best-first search may benefit from the use of a PHC control strategy. An example of such a problem is logic synthesis.

1.5 The Problem of Logic Synthesis

An important and difficult part of the design of combinational digital circuits involves the transformation of a functional description to a circuit implementation consisting of components and their interconnections. Constraints are placed on this process to ensure an implementation which is both cost and performance effective. This process is called logic synthesis. A special case of logic synthesis involves the minimization of two-level combinational logic and the subsequent mapping to PLAs [McC56,HCO74,BHH*82]. However, even this subset of the logic synthesis problem contains subproblems known to be NP-complete. Because of the complexity of the general case of multi-level logic synthesis, no single heuristic technique is clearly superior. Often the synthesis process is separated into several steps to make the problem more tractable. Rule-based systems have been applied to individual steps of the process [DBG*84,GBdH86]. However, the complexity of the state-space demands a powerful control strategy. The availability of results from these systems in addition to those of recently developed algorithmic approaches provide a means of determining the quality of a new approach. Thus, logic synthesis serves as a useful problem for studying and assessing the effectiveness of the PHC control strategy.

1.6 Organization of the Report

This report is organized as follows. Provided in Chapter 2 is a detailed description of probabilistic hill-climbing. In Chapter 3 the framework for a PHC control strategy is presented; necessary considerations for rule-based system components are described. Detailed aspects of PHC rule-based systems are illustrated through a rule-based system for logic synthesis, OPAL. The logic synthesis problem is described in Chapter 4. A system description of OPAL follows in Chapter 5. Provided in Chapter 6 is a comparison between results obtained from using probabilistic hill-climbing as the control strategy and results of traditional descent strategies with fixed degrees of look-ahead. Results indicate that fixed look-ahead strategies do not provide consistent results over a set of problem instances or evaluation functions. However, the

probabilistic hill-climbing control strategy not only performs consistently, but also obtains better results in almost all comparisons. Comparisons between OPAL and the MIS Logic Synthesis System[BRSW87] point out the strengths and weaknesses of both rule-based and algorithmic-based problem-solving. Conclusions and directions for future study are presented in Chapter 7.

Chapter 2

Probabilistic Hill-Climbing

Probabilistic hill-climbing techniques are a generalization of the simulated annealing technique proposed by Kirkpatrick et al. [KGv83] as a technique for solving combinatorial optimization problems [RS85]. Simulated annealing is based on the Metropolis procedure [MRRT53] which simulates the thermal motion of atoms of a substance at a given temperature. The motion of the atoms is determined as follows: random displacements of atoms which cause a decrease in the energy of the system are always accepted; displacements which increase the energy are accepted probabilistically. This probability is $f(\Delta E, T) = exp(\frac{-\Delta E}{k_B T})$ where ΔE is the change in energy, k_B is Boltzmann's constant, and T is the temperature of the system. As the temperature decreases, the probability of an increase in energy decreases. In simulated annealing and other probabilistic hill-climbing techniques, ΔE is equivalent to the change in cost, ΔC. The temperature, T, becomes a controlling parameter without any specific meaning. The basic PHC technique does not specify that the acceptance function, $f(\Delta E, T)$, be an exponential probability distribution. The parameter T is assumed to be ≥ 0 and is updated according to some monotonically decreasing function.

2.1 The Structure of PHC Techniques

The basic procedure and acceptance function structures characteristic of all probabilistic hill-climbing techniques are presented in Figures 2.1 and 2.2.

```
PHC(j0, T0)
    {
    T = T0;
    X = j0;
    while ("stopping criterion" is not satisfied)
        {
        while ("inner loop criterion" is not satisfied)
            {
            j = generate(X);
            if (accept(C(j), C(X), T))
                X = j;
            }
        T = update(T);
        }
    }
```

Figure 2.1: Basic PHC Structure

The basic PHC procedure is provided with the initial configuration state, j_0, and the initial value of the controlling parameter, T_0. Two nested loops control the state-space search. Each iteration of the outer loop is associated with a progressively smaller value of T provided by the *update*() function. The "stopping criterion" is usually based on the extent of cost improvement made by recent iterations. The inner loop performs a process of randomly generating successor states and deciding to accept or reject them. An "inner loop criterion" determines when the process terminates. This criterion attempts to identify when a condition of equilibrium is established[RS85, HRS86]. Equilibrium is established when the probability distribution of accessible states reaches steady-state.

The *accept*() function computes the difference in cost, ΔC_{ij}, between a current state i and a successor state j. The function $f()$ returns the probability of accepting the new state. For $\Delta C_{ij} \leq 0$, this probability is 1. The probability of accepting an increase in cost, $\Delta C_{ij} > 0$, is dependent on ΔC_{ij} and the process parameter T. Accepting the new state is then determined by generating a random number between 0 and 1 and comparing it to the value returned by $f()$.

```
accept(C(j), C(i), T)
    {
    ΔC_ij = C(j) − C(i);
    y = f(ΔC_ij, T);
    r = random(0. 1);
    if (r ≤ y)
        return(TRUE);
    else
        return(FALSE);
    }
```

Figure 2.2: Acceptance Function

The process parameter's initial value, T_0, is selected such that the probability of accepting an increase in cost is large. Thus, in the first few iterations of the outer loop, most generated states are accepted. This phase of the process is referred to as the "melt" phase. As the value of T decreases, increases in cost are less frequent. In the limit as T approaches 0, probabilistic hill-climbing eventually becomes greedy like steepest-descent—only accepting cost-improving states.

Chapter 3

Framework for a PHC Control Strategy

Described in this chapter is the framework for a rule-based system which utilizes a probabilistic hill-climbing control strategy. Although the components of rule-based systems in general are inherently modular, practical considerations for dependencies between the components must be made. In the process of describing each component, considerations pertaining to the use of a PHC control strategy are presented. Rather than constraining the use of the approach, these considerations provide for exploiting its characteristics.

3.1 The Control Strategy

The function of the control strategy in a rule-based system is to direct a search on a space $\mathcal{S}$ utilizing a set of rules $\mathcal{R}$. Each state, s_i, in $\mathcal{S}$ is defined uniquely by a database configuration, d_i. Elements within d_i exist which "trigger", or match, rule antecedents in $\mathcal{R}$. The elements necessary to match a particular rule are collectively called a *location*. For some problems, a rule will only trigger at one location. However, the general case provides for multiple triggering by a single rule. The set of all rule-location pairs which trigger in d_i is called the *conflict set*, C_i. The "firing" of a rule r_j, or the application of that rule, at a location l in d_i defines a new database

configuration and state:

$$d_{i+1} = r_j(l, d_i)$$

Firing a rule consists of performing the actions specified by the rule's consequent. In deciding which rules to fire, the control strategy utilizes an evaluation function, $\mathcal{E}$, to ascertain the quality of a particular state in $\mathcal{S}$. This function assigns a numeric value to a state according to estimated distance to a goal, value as a solution, or other criteria.

A PHC control strategy procedure is presented in Figure 3.1. Parameters to the procedure are the starting state, specified by d_0 and the initial process parameter value, T_0.

The function *generate_conflict_set()* creates the set of all applicable rule-location pairs given the database and rule set. An additional inner loop (contained in the standard PHC "inner loop") iterates until a rule is accepted. The *select()* function randomly selects from the set C a rule to be applied. To ensure that selected rule-location pairs are not chosen again, they are removed from C. The expression $\mathcal{E}(r(l, d))$ returns the evaluation of the state generated by the invocation of the rule r. Thus, the value of the rule is determined by the state it generates. If the rule is accepted, the database receives the new state description. Note that it is possible that C will become empty before a rule is accepted. The "inner loop criterion" must account for this possibility.

For problems where the conflict set is large, generating the complete set is inefficient—for as soon as a rule is accepted, the remaining conflict pairs are discarded. For this reason, the procedure of Figure 3.1 is a naive one. A more efficient variation of this procedure is given in Figure 3.2. This variation is suitable for rule-based systems containing rules which match at only one or a small number of locations in the database. The function *select_location()* randomly selects from matching locations or returns $\emptyset$ if matching locations for the rule do not exist. Note that iterations of the "inner loop" may occur without a rule being fired. At most only one location of each rule is tested, so it is possible that other acceptable locations exist. If a rule is not fired, the database remains unchanged and the process of selecting a rule and

```
Naive_PHC_control(d_0, T_0)
    {
    T = T_0;
    d = d_0;
    while ("stopping criterion" is not satisfied)
        {
        while ("inner loop criterion" is not satisfied)
            {
            C = generate_conflict_set(d, R);
            fired = FALSE;
            while (fired is FALSE and C ≠ ∅)
                {
                {r, l} =select(C);
                C = C − {r, l};
                if (accept(E(r(l, d)), E(d), T));
                    {
                    d = r(l, d);
                    fired = TRUE;
                    }
                }
            }
        T = update(T);
        }
    }
```

Figure 3.1: PHC Control Strategy Procedure: Naive Approach

location starts over. Thus, other acceptable locations for each rule will be considered.

Other variations of the PHC control strategy can easily be realized according to the characteristics of the rules and database in the system. For instance, in a problem such as logic synthesis, each rule can match at a large number of locations in the database. In this case, it is appropriate to first randomly select a location and then test each rule at that location.

3.2 The Evaluation Function

The evaluation function, $\mathcal{E}$, indicates the "quality" of a state in the state-space. This "quality" may be an estimation of how close the state is to a goal, or it may be the quality of the state as a solution (i.e., as a goal state itself). As is the case with all heuristic search methods, the accuracy of the evaluation function has a significant impact on the effectiveness of the search. Of special concern to PHC rule-based systems is how the evaluation function can improve the accessibility of states and the efficiency of the search.

A state s_j is said to be *accessible* to a state s_i if the probability P_{ij}^n of a transition from s_i to s_j in some $n > 0$ steps is non-zero and therefore the probability P_{ij} of eventually reaching state s_j from s_i is non-zero. Accessibility of states is an important theoretical requirement of PHC techniques [RS85] which impacts both the rule base and the evaluation function. Certainly, if a goal state is not accessible from the initial state, then the state-space search is wasted effort. Just as important is the accessibility of a goal state, s_g, from a locally optimal state, s_l. The probability P_{lg} is directly related to the number of cost increasing rules which need to be accepted to escape the local optimum. A useful technique to increase P_{lg} is to allow *illegal* states—states which contain inconsistent assertions or violate problem constraints. These states can reduce the necessary number of cost increasing rules by providing "short-cuts" to better locally optimal and correct states. Because these states are usually undesirable, the evaluation function must be able to identify them and impose a penalty on their value. This technique has been successfully applied to various simulated annealing applications[DN86,SS86].

The evaluation function is often the most time-consuming task of PHC rule-based systems. The PHC strategy relies on moving quickly through a large state-space. The number of calls made to the evaluation function is usually several times the number of accepted rules. In addition, since states are not remembered, re-evaluations are often necessary. Making efficient state evaluations is therefore essential. For some problems, it may be possible to determine the value of a new state using the value of the current state and the rule being considered. Because the rule defines a change from

the current state's value, the time complexity of the evaluation for these problems is independent of database size. This method also alleviates the cost of applying a rule and then having to reverse it if not accepted. In more complex systems, the effect of rules on the database cannot be determined *a priori*. However, incremental evaluations are still possible if the effect of the rule can be sufficiently isolated.

3.3 The Rule Base

The proper selection of rules for the rule base is often the critical difference between systems that work well and those that fail. As distinct from conventional rule-based systems, issues such as state accessibility and rule granularity deserve special attention in PHC-based systems.

As noted in the previous section, accessibility of states is an important requirement of the PHC control strategy. The PHC strategy is to perform a forward-moving traversal through the state-space. Therefore, the goal state must be reachable from any intermediate state. Since a goal state is not known beforehand, this requirement implies that any state be accessible from any other state. That is, for $T > 0$, there exists some $m, n > 0$ such that $P_{ij}^{n} > 0$ and $P_{ji}^{m} > 0$. Thus, both "forward-" and "backward-" moving rules are necessary. If a forward-moving rule generates the state s_j from the state s_i, then a backward-moving rule must exist which will allow the generation of s_i from s_j. Some rules are naturally symmetric and can move in either direction. In other cases, the generation of s_i can be performed by a sequence of rules which move through intermediate states.

Another consideration concerns the "granularity" of the rule set. The question is whether the rules should perform high-level operations which make major transformations in the database or should they enact primitive operations. Neither of these extremes is necessarily the correct approach. One criterion for determining the appropriate granularity is the rule set's *completeness*. A set of rules is complete if they can always generate a goal state for any instance of the problem domain. High-level rules are often oriented toward special-case configurations which implies that a large number be present in $\mathcal{R}$ to solve problems in the general-case. Often the number of

rules alone makes it impossible to say anything about completeness. A primitive rule set is often small enough that their completeness can usually be determined. However, high-level rules can traverse the problem space more rapidly than primitive rules.

Another constraint on the granularity of a rule set is the ability to ascertain the quality of rules independent of other rules. Since the decision to accept a rule is according to how it alone affects the database, useful rules whose eventual benefit is not seen immediately may not be accepted. Of course, this problem cannot be completely avoided. If it could, a greedy search would be sufficient for solving the problem. However, the rule set should be closely matched with the evaluation function for optimal results.

```
PHC_control(d_0.T_0)
    {
    T = T_0;
    d = d_0;
    while ("stopping criterion" is not satisfied)
        {
        while ("inner loop criterion" is not satisfied)
            {
            R_u = R;
            fired = FALSE;
            while (fired is FALSE and R_u ≠ ∅);
                {
                r = select(R_u);
                R_u = R_u − r;
                l = select_location(d, r);
                if (l ≠ ∅)
                    if (accept(E(r(l, d)), E(d), T));
                        {
                        d = r(l, d);
                        fired = TRUE;
                        }
                }
            }
        T = update(T);
        }
    }
```

Figure 3.2: PHC Control Strategy Procedure #2

Chapter 4

Logic Synthesis

4.1 Introduction

Because of the sheer size of today's digital electronic systems, VLSI designers have become more dependent on automatic synthesis tools. The first widely used synthesis tools performed the layout tasks of placement, routing, and compaction. Increasing circuit complexity has lead to the demand for higher levels of automatic synthesis. One such task which has received much attention is logic synthesis [BRSW87,Sas86,Kv81, DBG*84,GBdH86]. Logic synthesis is the process of transforming a functional description of a combinational circuit into an optimal implementation consisting of circuit components and their interconnection. Early work on logic synthesis has concerned itself with the special case of two-level logic minimization [McC56,HCO74,BHH*82]. Two-level logic is especially attractive since it can be easily implemented using PLAs. However, multi-level logic representations are often more area efficient and can provide a higher level of performance. Unfortunately, area and timing constraints often conflict, making the optimal synthesis of multi-level logic a very difficult problem.

The functional description of a circuit supplied to a logic synthesis tool is usually a set of logic equations or truth tables extracted from a register-transfer level description. Area and timing requirements are often supplied as well. Because these constraints conflict, the most that can be expected from the synthesis tool is for it to make appropriate tradeoffs (e.g. "generate the smallest circuit with a critical path

delay less than 10ns"). Unfortunately, the techniques used in many logic synthesis tools can consider only one of these constraints at a time.

The synthesis process is also constrained by the desired technology. Standard cell and gate array designs limit suitable components to those provided in a cell library. Characteristics of these cells cause one technology's implementation to be very different from another. For instance, the use of NAND gates in CMOS designs is preferable to NOR gates.

These issues can overwhelm a designer attempting to synthesize manually anything but the simplest circuit. Optimization is often only attempted on the most critical portions of a design, resulting in needlessly larger and slower chips. Clearly, the demand for shorter design times and higher performance requires the utilization of automatic logic synthesis.

4.2 Approaches to Multi-Level Logic Synthesis

Algorithmic solutions to multi-level logic synthesis[BRSW87,Sas86] have not yet attained the acceptance that they have had with two-level logic. Algorithmic approaches perform most—if not all—of their optimization in a technology-independent manner. A common circuit representation used by these tools is the boolean network. After technology-independent optimization is performed on the network, it is mapped into a particular technology by assigning appropriate gates from a library[DGR*87,Keu87].

Part of the difficulty experienced by algorithmic approaches is determining a suitable technology-independent abstraction which represents an accurate picture of the design goal (e.g., minimum area, delay, etc). In the case of two-level logic, PLAs are often the realization of the circuit and can serve as the basis for the abstraction. Minimizing the number of product terms is a good approximation to minimizing the size of the PLA; minimizing the number of literals present in the product terms minimizes the number of connections in the AND-plane. In addition, fewer product terms and connections imply a faster circuit. In contrast, multi-level logic is implemented in many different ways (e.g. static, dynamic logic; standard cell, gate matrix). The result being that a set of equations minimal with respect to literal count does not

always provide for a minimal area implementation. Since little can be done outside of transistor sizing and layout issues to minimize PLA delays, timing was never a major consideration in two-level logic synthesis. However, logic delays vary considerably with multi-level network implementations. The most common abstraction for network delay used in algorithmic approaches today is the number of logic levels. Unfortunately, the variety of gates used to implement the network and the impact of fanout delay make the logic level approximation far from accurate in most cases. Current algorithmic approaches do not provide solutions that are both optimal in area and timing.

Rule-based systems, unlike algorithmic approaches, do not require a technology-independent abstraction but can optimize the target implementation directly [DBG*84, GBdH86]. Rules consist of transformations from subnetworks of library components to other equivalent subnetworks. The generality of the rules allow complex modules, such as full adders, to be used effectively during the optimization. In addition, rule-based logic synthesis can incorporate timing optimization easily due to the availability of correct delay information and the use of a flexible control strategy.

However, problems exist in current rule-based logic synthesis systems which hinder their effectiveness. Because of the local nature of rules, global optimizations, such as the identification of common subexpressions, are often missed. If the domain of rules is limited to library modules, optimization can also be hindered by the limited fanin of available gates. The use of a technology-specific rule library also has an impact on the flexibility of the system. Because typical libraries contain hundreds of rules, providing for a different technology can be a formidable task. An additional problem with many rule-based systems is that the desire for generality often results in an inefficient implementation. Thus, in the same amount of computer time that a rule-based system requires, an algorithmic, problem-specific approach may be able to perform a much larger search of the solution space. Finally, the choice of a control strategy can have a significant impact on solution quality.

Some rule-based systems such as LSS[DBG*84] use greedy, steepest-descent strategies. However, de Geus and Cohen[dC85] note that this strategy is not sufficient for timing optimization because the benefit of a timing rule is often not immediately obvi-

ous. Look-ahead techniques are suggested as a means to overcome this problem. They also note that area reduction can suffer from the same "short-sightedness" when less desirable rules lead to more powerful ones. As noted previously, exponential growth of the look-ahead tree requires that the number of levels be restricted to two or three rules.

The lack of a successful algorithmic approach to the multivariate objectives of logic designers and the potential of rule-based systems make the logic synthesis problem an appropriate test case for the probabilistic hill-climbing control strategy. A rule-based logic synthesis tool, OPAL, has been developed which attempts to overcome many of the problems associated with current rule-based approaches. In particular, it utilizes the PHC control strategy for performing technology-independent and -dependent area/timing optimization. The application of simulated annealing to logic synthesis has been reported in the literature[LD86,Gon86]. However, in both cases, simulated annealing was only applied to subproblems of the synthesis process. In addition, the logic transformations performed by these approaches are not complete. These limitations restrict the state-space search and effectiveness of the simulated annealing technique. Finally, these efforts did not formulate the problem using the rule-based paradigm nor did they identify simulated annealing as a useful control strategy for rule-based problem solving in general. For these reasons, OPAL represents a new approach to the logic synthesis problem.

Chapter 5

OPAL: A Rule-Based System for Logic Synthesis

Illustrated in Figure 5.1 are the relationships between the major components of the OPAL Logic Synthesis System. A similar structure would exist for rule-based systems for solving different problems and where the PHC control strategy was applied. Because the system is inherently data-driven, the network (database) is located centrally. The PHC control mechanism calls upon the evaluation function for rule selection. A technology library is utilized by the evaluation function to determine a mapping and associated area and delay.

5.1 Network Representation

The state-space for logic synthesis and optimization is the set of all networks which are composed of combinational modules and their interconnection and are functionally equivalent to a given logic description. Although it is possible to construct combinational networks which contain cycles[1], such configurations are not considered. Due to the limitations of such networks, little is lost by ignoring them.

As stated previously, a Boolean network can serve as the network representation.

[1] however, their Boolean functions do not contain feedback

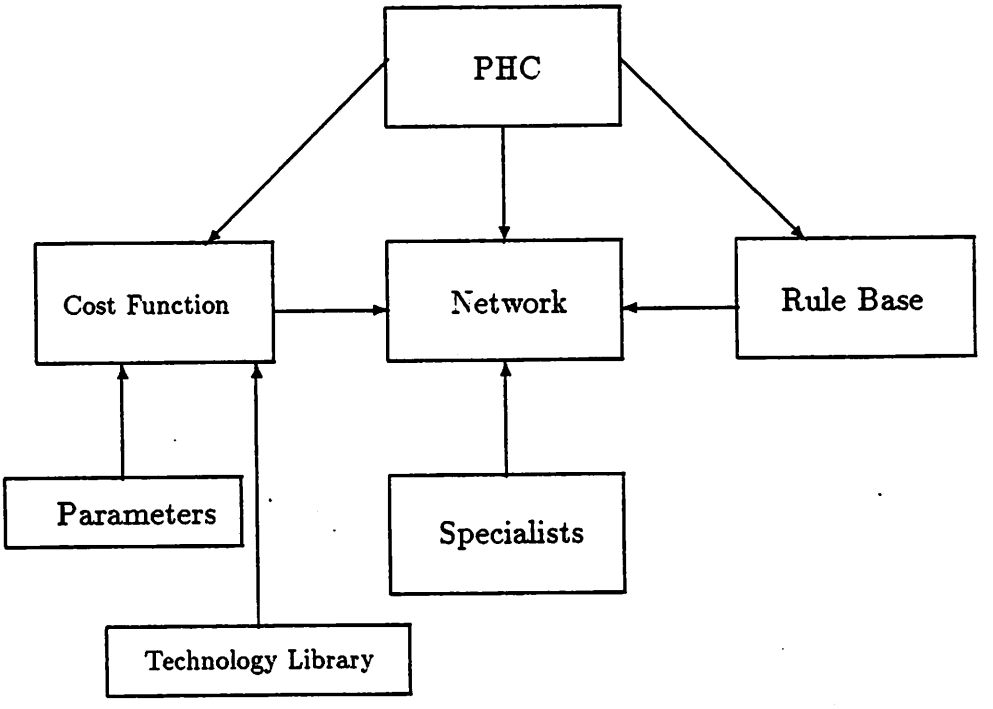

Figure 5.1: PHC Rule-Based System for Logic Synthesis

A Boolean network is a directed, acyclic graph where each node in the graph is a Boolean function represented by a disjunction of product terms. Associated with each node n_i is a variable name N_i. For each node n_i with a product term containing a reference to the variable N_j, there exists an arc from n_j to n_i. Implicit in this representation is the eventual mapping of the network into the target technology. The advantage of Boolean networks is the ease they provide in performing Boolean minimization and common subexpression extraction.

Since the focus of the rules is on literals, each variable is internally represented by a subgraph of product terms and literals. The actual network representation for the function $a \oplus b \oplus c$ is shown in Figure 5.2. In addition to nodes that specify variables, term and literal nodes have been added.

The algebraic representation for the graph of Figure 5.2 is:

$$\begin{aligned} V_1 &= a\overline{v_2} + \overline{a}v_2 \\ V_2 &= b\overline{c} + \overline{b}c \end{aligned}$$

Because of the simplicity of the algebraic form, henceforth it will be used to describe both networks and rules.

The advantages of using a Boolean network representation are realized in OPAL without the limitations of a technology-independent evaluation function. Each time the network is evaluated, a mapping to the module library is performed. Because many one-to-one mappings exist from the Boolean network to a realizable circuit, an optimal mapping requires a difficult search. This search would have a large impact on run-time and is thus undesirable.

To overcome this potential problem, the optimality of the mapping is relaxed by defining an easily-obtained, direct mapping which is almost one-to-one. Thus, changes to the Boolean network correspond directly to changes in the implementation. This technique has the advantage of providing immediate feedback for rule selection but imposes some constraints on the node functions. These constraints come in the form of penalties rather than restrictions on illegal configurations of the network ("illegal states" in the state-space). For instance, a product term with 10 literals

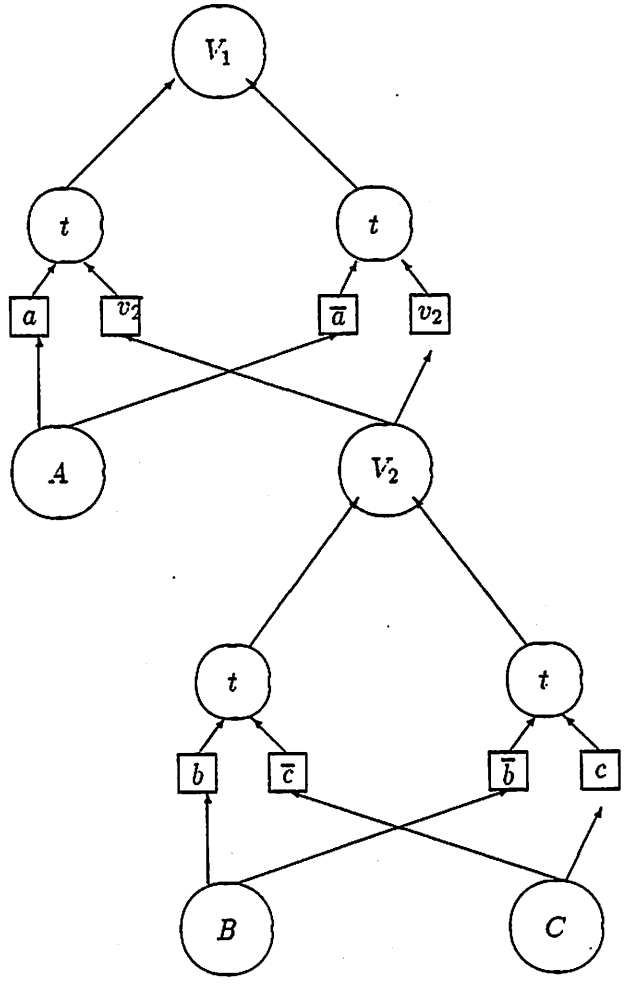

Figure 5.2: Network Representation of $a \oplus b \oplus c$

may not be directly implementable because a 10-input AND or NAND gate may not be available. Mapping to these gates is allowed but a penalty is imposed to discourage their use. Legal solutions are ensured by increasing the penalty during the optimization. However, optimization is not hindered by the module library as is the case with technology-dependent representations. The direct mapping approach provides a great deal of accuracy but without a loss of efficiency because remapping can occur incrementally.

A definition which will be of use later is the *fanout* of a variable. In the above network for $a \oplus b \oplus c$, V_2 is said to fanout to V_1 and V_1 is a fanout variable of V_2, or simply V_1 *references* V_2. In addition, V_2 is said to have both a *positive* and *negative* reference because of the literals $\overline{v_2}$ and v_2 respectively. It is often useful to refer to the reference count of a variable. V_2 has a negative reference count of 1, a positive reference count of 1, and a total reference count of 2. This definition of fanout and reference pertains to the Boolean network. The mapped network also has a definition of fanout. If V_1 is mapped to an exclusive-OR gate, the gate that represents V_2 would have a positive fanout count of 1 and a negative fanout count of 0 since v_2 is an input to the gate and $\overline{v_2}$ is not. The difference between the Boolean network meaning of fanout and the mapped network meaning is an important one. To avoid confusion, reference counts will always correspond to the Boolean network and fanout counts to the mapped network.

5.2 Rule Base

As stated in Section 3.3, the criteria that must be met by the rule base are that it provide state accessibility and completeness. Since all legal transformations of Boolean networks must be derivable in terms of the Boolean algebra identities, they certainly meet the completeness requirement. Unfortunately, some of these transformation (e.g. $A \Rightarrow A1$, meaning the literal(s) A replaced by $A \cdot 1$) require more direction than can be achieved using a generalized control strategy. This problem is circumvented by replacing such rules with frequently used theorems (e.g. $A + AB \Rightarrow A$). Obviously, such a strategy trades off completeness for efficiency. However, by minimizing the

	Forward Rule	Inverse Rule
1	$A0 \Rightarrow 0$	
2	$A+0 \Rightarrow A$	
3	$A1 \Rightarrow A$	
4	$A+1 \Rightarrow 1$	
5	$A+A \Rightarrow A$	$A \Rightarrow A+A$
6	$A\overline{A} \Rightarrow 0$	
7	$A+\overline{A} \Rightarrow 1$	$X \Rightarrow XA+X\overline{A}$
8	$\overline{\overline{A}} \Rightarrow A$	$A \Rightarrow \overline{\overline{A}}$
9	$A(BC) \Rightarrow ABC$	$ABC \Rightarrow A(BC)$
10	$A+(B+C) \Rightarrow A+B+C$	$A+B+C \Rightarrow A+(B+C)$
11	$A(B+C) \Rightarrow AB+AC$	$AB+AC \Rightarrow A(B+C)$
12	$A+AB \Rightarrow A$	
13	$A+\overline{A}B \Rightarrow A+B$	
14	$\overline{A+B} \Rightarrow \overline{A}\,\overline{B}$	$\overline{A}\,\overline{B} \Rightarrow \overline{A+B}$
15	$\overline{AB} \Rightarrow \overline{A}+\overline{B}$	$\overline{A}+\overline{B} \Rightarrow \overline{AB}$
16		$A+BC \Rightarrow (A+B)(A+C)$
17		$A(B+C) \Rightarrow A(AB+C)$

Table 5.1: Rule Set

introduction of "higher-level" rules, confidence in the degree of rule completeness can be maintained. The complete list of rules used in OPAL is presented in Table 5.2. By providing both a forward and inverse rule for the identities, state accessibility is maintained. In several of the theorems, the forward or inverse rule is left out because it can be easily obtained from simpler rules.

These rule specifications are symbolic and not literal. The variables (A, B, C, X) which appear in the rules are defined by their context and apply to the most general situation. For example, the variable A in the rule $A1 \Rightarrow A$ would "match" one or more literals of a product term. In addition, this particular term can be part of a disjunction of several terms. The rule $A+(B+C) \Rightarrow A+B+C$ should be interpreted as matching $Y = A+x; X = B+C$. That is, a single literal term x is defined by one or more product terms (indicated by $B+C$). A represents all the product terms of the variable Y except the one containing x. Rules such as $ABC \Rightarrow A(BC)$ select

a variable number of literals to represent BC and A is simply the remainder for the product term. In general this selection occurs randomly and is made by the rule.[2]

Given the flexibility inherent in the rule specification and the "fixed" nature of the rules, it was decided to implement them as test/fire compiled functions. Since the rules are mostly problem and technology-independent, the benefit provided by interpreted rules—the ability to change the rule base without recompilation—is not apparent. Network locations are identified by literal nodes. A rule test function quickly determines if the rule is applicable at that location. If the rule test succeeds, the fire function is executed so that the evaluation function can accurately determine the effect of the rule on the network. All the rules are specified in terms of primitive operations on the Boolean network which create and remove variable, term, and literal nodes. If a rule is not accepted, it is necessary to "undo" the modifications to the network. This task is accomplished efficiently by maintaining a transaction stack of the primitive operations performed by the rule. Each primitive operation is then reversed individually. This technique is rule-independent and thus eliminates the need to specify individual "undo" functions for each rule.

The rules in Figure 5.2 only provide for local transformations. In logic synthesis, an essential global operation is to identify and share common subexpressions in the network. In so doing, area is significantly reduced. Unfortunately, current rule-based logic synthesis systems either do not provide this optimization or do so in a limited manner. Algorithmic techniques which explicitly seek out the optimal set of subexpressions for extraction are not easily incorporated into the rule-based paradigm. The strategy taken with OPAL is to identify common subexpressions *as they are generated* by the rules. This operation is performed through the use of a *specialist.* Specialists are normally passive mechanisms which are independent of the control mechanism and the rule base. Their purpose is to provide global direction to the optimization by modifying or extending the effect of rules as they are fired. The acceptance of the global operation is determined by the acceptance of the extended rule. Thus, the control strategy ultimately determines which global operations are

[2]additional heuristics are incorporated which weight favorable selections

Before	After rule $\overline{AB} \Rightarrow \overline{A} + \overline{B}$ on p
$m = \overline{a} + \overline{b}$	$m = \overline{a} + \overline{b}$
$n = ab$	$p = a + mc$
$p = a + \overline{n}c$	

Figure 5.3: Merge Operation

Before	After rule $\overline{AB} \Rightarrow \overline{A} + \overline{B}$ on p
$m = \overline{a} + \overline{b}$	$m = \overline{a} + \overline{b}$
$n = ab$	$p = a + mc$
$o = mc + a$	$q = \overline{p} + r$
$p = a + \overline{n}c$	
$q = \overline{o} + r$	

Figure 5.4: Effect of Merging on Transitive Fanout

to be made. This technique has proven to be very effective in performing common subexpression extraction.

After a rule is fired, new and modified variables may define Boolean functions which exist elsewhere in the network. The common subexpression specialist maintains a hash table to identify equivalent variable nodes quickly. Two variable nodes are equivalent if they contain equivalent product terms (order independent). Equivalent product terms must contain the same literals (order independent). Thus two functions with the same functionality may not be considered equivalent. Because equivalent subexpressions are variable nodes in the Boolean network, only the combination, or merging, of variables is required. Figure 5.3 is an example of the merge operation. The merge operation is defined recursively since the merging of two intermediate variables may result in equivalence among the transitive fanout variables. An example of this situation is shown in Figure 5.4.

5.3 The PHC Control Mechanism

5.3.1 Conflict Resolution

Typically, rule-based systems test each rule with the database and select from those that match, a single rule which is to be fired. This selection process is *conflict resolution*, and is essential to state-space search. Potentially, the number of rule tests that must be made for each state of the database is $|\mathcal{R}||\mathcal{L}|$ where $\mathcal{R}$ is the set of rules and $\mathcal{L}$ is the set of database test locations. The actual number of tests is substantially reduced if $\mathcal{R}$ is decomposed into subsets of rules containing common antecedent components—a conflict resolution technique known as *context limiting*. For the logic synthesis problem, $\mathcal{L}$ usually contains all of the nodes in the Boolean network. Thus $|\mathcal{R}||\mathcal{L}|$ is prohibitively high. Consequently, the number of rule-location pairs in the conflict set, $|\mathcal{C}|$, is also quite large. [dC85] indicates that in Socrates, a circuit with 100 gates and a rule base of 50 rules typically generates a conflict set size of 50. Because high-level rules such as those used in Socrates are less likely to match as often as the fine-grain rules used by OPAL, the conflict set size for OPAL can be expected to be even larger. Clearly, avoiding generation of the entire conflict set is desirable.

Fortunately, conflict resolution in probabilistic hill-climbing is inherently simple. The rule to fire is simply any of the members of $\mathcal{C}$ which meet the acceptance criteria—no ordering of "acceptable" rules is necessary. Improved location selection is achieved by placing a user-specified amount of emphasis on locations near the previously accepted location and locations on the critical path. Methods for obtaining the rule to fire iterate through rule-location pairs until one is accepted. An appropriate method is one which performs efficiently but does not hinder the search by biasing the selection. Four selection methods are implemented in OPAL. Their procedures are contained in Figures 5.5–5.8. Because they all have a common PHC "outer" loop (the "stopping criterion" loop of Figure 2.1), only the inner loops are given. Methods #1 and #4 have been experimentally determined to be superior than the other two methods in solution quality and run-time efficiency.

A very useful conflict resolution strategy in rule-based systems is size ordering. Rules are classified according to their number of antecedent conditions and the like-

```
for ("a certain number of iterations")
    {
    l = select_location(d);
    fired = FALSE;
    R' = R;
    r_count = 0;
    while (fired is not FALSE and r_count < "maximum tries")
        {
        r_count++;
        r = select_rule(R');
        R' = R' - r;
        if (triggers(r, l, d) and accept(E(r(l, d)), E(d), T));
            {
            d = r(l, d);
            fired = TRUE;
            }
        }
    }
```

Figure 5.5: Rule-Location Selection Method #1

```
for ("a certain number of iterations")
    {
    l = select_location(d);
    r = select_rule(R);
    if (triggers(r, l, d) and accept(E(r(l, d)), E(d), T));
        d = r(l, d);
    }
```

Figure 5.6: Rule-Location Selection Method #2

```
for ("a certain number of iterations")
    {
    l = select_location(d);
    fired = FALSE;
    R' = R;
    while (fired is not FALSE and R' ≠ ∅)
        {
        r = select_rule(R');
        R' = R' − r;
        if (triggers(r, l, d) and accept(E(r(l, d)), E(d), T));
            {
            d = r(l, d);
            fired = TRUE;
            }
        }
    }
```

Figure 5.7: Rule-Location Selection Method #3

```
for ("a certain number of iterations")
    {
    r = select_rule(R');
    fired = FALSE;
    loc_count = 0;
    while (fired is not FALSE and loc_count < "maximum tries")
        {
        l = select_location(d);
        loc_count++;
        if (triggers(r, l, d) and accept(E(r(l, d)), E(d), T));
            {
            d = r(l, d);
            fired = TRUE;
            }
        }
    }
```

Figure 5.8: Rule-Location Selection Method #4

lihood of matching. This strategy encourages the acceptance of highly constrained rules over those which match more easily—often improving the search efficiency and solution quality. However, introduction of size ordering into PHC systems hinders the control strategy by reducing state accessibility. The benefits of size ordering can be achieved by *rule weighting*. According to the match frequency of a rule j, a weight w_j representing selection frequency is specified. These weights define a selection probability function:

$$p_i = \frac{w_i}{\sum_{j=1}^{|\mathcal{R}|} w_j}$$

where p_i is the probability of *select_rule*() returning rule i. Table 5.3.1 indicates typical match frequencies and possible rule weights for OPAL's rule base (Absence of the forward or reverse rule is indicated in the table by "-"). For OPAL, selection of rule weights were experimentally determined. It was found that extremely low and high weights are undesirable, and that precision in weight selection does not effect efficiency and solution quality. In addition to approximating size ordering, rule weights provide a method for discouraging rules which detract from the optimization.

5.3.2 Process Control

Probabilistic hill-climbing techniques are distinguished by a choice of stopping and inner loop criterions and an acceptance function. The control parameter T is defined by an initial starting point T_0 and an update function. The process of selecting these PHC parameters is known as "tuning" and can be considered both an advantage and disadvantage. Tuning provides the flexibility and customization which make PHC techniques useful. However, tuning is inherently an iterative and experimental process, often requiring a substantial amount of time.

The initial "temperature" T_0 affects the extent of the melt. Too high a value of T_0 wastes CPU time; too low a value may result in solutions that are far from the global optimum. Determination of T_0 requires consideration of the quality of the initial network configuration. A locally optimal network requires a higher temperature than required by a poor starting configuration. A variable which is independent of the

	Forward Rule		Inverse Rule	
	% Match	Weight	% Match	Weight
$A0 \Leftrightarrow 0$	<1	0.1	-	-
$A + 0 \Leftrightarrow A$	<1	0.1	-	-
$A1 \Leftrightarrow A$	<1	0.1	-	-
$A + 1 \Leftrightarrow 1$	<1	0.1	-	-
$A + A \Leftrightarrow A$	<1	0.5	100	0.1
$A\overline{A} \Leftrightarrow 0$	<1	0.1	61.1	0.1
$A + \overline{A} \Leftrightarrow 1$	<1	0.5	48.9	0.1
$\overline{\overline{A}} \Leftrightarrow A$	3	1.0	38.9	0.5
$A(BC) \Leftrightarrow ABC$	9.7	1.0	61.1	1.0
$A + (B + C) \Leftrightarrow A + B + C$	<1	1.0	26.2	0.1
$A(B + C) \Leftrightarrow AB + AC$	11	1.0	<1	1.0
$A + AB \Leftrightarrow A$	<1	1.0	-	-
$A + \overline{A}B \Leftrightarrow A + B$	<1	1.0	34.3	0.01
$\overline{A + B} \Leftrightarrow \overline{A}\,\overline{B}$	13.9	1.0	60.9	1.0
$\overline{AB} \Leftrightarrow \overline{A} + \overline{B}$	31.3	1.0	25.6	1.0
$A + BC \Leftrightarrow (A + B)(A + C)$	-	-	4.8	0.01
$A(B + C) \Leftrightarrow A(AB + C)$	-	-	8.2	0.01

Table 5.2: Rule Match Frequencies and Typical Weights

initial network is P_a, the desired probability of accepting any cost increasing rule at T_0. The value of T_0 is then defined as:

$$T_0 = -\lambda / \log P_a$$

where λ is the average cost increase for the network. Determination of λ is accomplished by choosing rule-location pairs randomly and evaluating their effect on the network. The sample size— number of matching rule-location pairs which increase the network cost—is equal to $|\mathcal{L}||\mathcal{R}|/4$. This sample size is conservative but has been experimentally found to perform reliably.

The control parameter T is updated after each iteration of the outer PHC loop. The general form of the update function is:

$$T_{n+1} = \alpha(T_n) \cdot T_n$$

Where $0 \leq \alpha(T_n) < 1$. OPAL divides the search process into three phases: melt, main search, and greedy search. $\alpha(T_n)$ is defined as:

$$\alpha(T_n) = \begin{cases} \alpha_1 & \text{until } C_{ave}(T_n) < (C_0 + B_c)/2, \text{ then} \\ \alpha_2 & \text{until } C_{max}(T_n) < C_{min}(T_n) + \lambda\delta_2, \text{ then} \\ \alpha_3 & \end{cases}$$

and subject to the condition that returning α_{i+1} inhibits α_i from being returned at a later n. $C_{min}(T_n)$, $C_{ave}(T_n)$, and $C_{max}(T_n)$ are, the minimum, average, and maximum evaluated network costs for outer loop iteration n, respectively. C_0 is the initial network cost and B_c is the cost upper bound (which will be explained later in this section). δ_2 is a fractional multiplier for the average increase in cost, λ, which specifies a window on the cost variation. The transition from α_2 to α_3 occurs when the cost variation drops to such a small amount that further significant cost reduction is unlikely. At this point, it is desirable to perform a greedy search which will locate a minimal solution quickly and exit. Typical α values are: $\alpha_1 = 0.97$, $\alpha_2 = 0.99$, and $\alpha_3 = 0.50$.

Usually, no more than a few outer loop iterations of the greedy search phase are required before the cost ceases to change. At this point, T is so low that only cost

improving rules are being accepted. The termination criterion is then specified by the number of constant cost iterations desired. To prevent the search from terminating when cost improvements can still be made, a local minimum check is also part of the termination criterion.

The PHC inner loop criterion requires that the process reach equilibrium at each value of T. Equilibrium is defined as the establishment of the steady-state probability distribution of the accessible states[RS85,HRS86]. Since the set of accessible states is not known beforehand, only an approximation is possible. The number of iterations of the inner loop is according to the number of rule tests performed. Minimum and maximum rule test counts are defined as:

$$N_{min} = K_{ss}|\mathcal{R}||\mathcal{L}_0|(1 - \exp\frac{-T_0}{2T})$$

and

$$N_{max} = \delta_{ss} N_{min}$$

where K_{ss} (typically 8) and δss (typically 1.3) are experimentally determined constants and $\mathcal{L}_0$ is the set of initial network locations. As T decreases, N_{min} increases to the asymptotic value of $K_{ss}|\mathcal{R}||\mathcal{L}_0|$. The maximum value, N_{max} is specified to ensure the termination of the inner loop, although equilibrium is usually attained before N_{max} iterations have occurred. Equilibrium is considered to have been reached when at least N_{min} iterations have occurred and the following condition is met:

$$|C - C_{ave}(T_n)| \leq \delta_1 \sigma(T_n)$$

C is the cost of the network, δ_1 (typically 0.6) is an experimentally determined constant and $\sigma(T_n)$ is the standard deviation of network cost at temperature T_n [HRS86]. The idea of this restriction is to make sure the inner loop terminates with a cost representative of the given temperature. It has been experimentally found to provide much more robustness to the process than the iteration count alone.

The acceptance criterion is another parameter of PHC whose purpose is to control the acceptance of rules. OPAL uses the Boltzmann function which characterizes the simulated annealing technique. In addition, an upper bound on the cost of the

network is imposed to restrict the growth of the network during the melt phase. A large majority of rules in the conflict set incur a cost increase which, if unrestricted at high temperatures, would result in needlessly large networks during the melt. The purpose of bounding is to reduce the required optimization from the melted state but not eliminate desirable paths to the globally optimal state. The upper bound used in OPAL is:

$$B_c = C_0 + K_b\lambda$$

where C_0 is the initial network cost and $K_b > 0$ is a constant which can be specified at run-time. λ was defined earlier as the average increase in cost. λ captures the circuit dependent information which provides the robustness in the bound calculation.

One possible method for enforcing the bound is to simply prevent cost increases over B_c. This method is undesirable because the cost approaches B_c soon after the melt is initiated. Henceforth, the network tends to "sit" at the bound until the temperature drops to a point where cost decreasing rules are accepted more often than increasing rules. To prevent this problem, the ΔC is increasingly weighted (penalized) as the network approaches the bound. This weight, γ is defined as:

$$\gamma = \frac{B_c - C_{min}}{B_c - C_{new}}$$

where C_{min} and C_{new} are the minimum network cost for all configurations up to the current state and the network cost incurred by the rule being considered. The acceptance probability is then defined as:

$$Cf(\Delta C, T, \gamma) = \exp(\frac{-\Delta C}{T})/\gamma$$

5.4 Network Evaluation

The evaluation function used in OPAL measures the cost of a particular network as a solution rather than indicating a distance from the optimal solution. The fact that each configuration in the state-space may represent a goal state and that the cost of the optimum solution is unknown implies that this solution cannot be identified

during the search. However, experimental evidence suggests that there are often many configurations similar in cost to the global optimum. If these configurations are considered acceptable solutions and are thus goal states, the accessibility of a goal state is naturally greater than if only the global optimum is acceptable. Because of the large state-space of equivalent network configurations, this consideration is essential to a PHC rule-based solution to logic synthesis.

The cost of a Boolean network is defined in terms of the area and/or delay of its implementation. Boolean networks possess a simple, technology-independent cost estimate: area is the number of literals and delay is the number of logic levels. The disadvantages of these estimates were mentioned in Section 4.2. However, it may be advantageous to perform an efficient technology-independent optimization first. The result of such an optimization can then be re-optimized using detailed area and timing information. This approach is possible with OPAL because of the flexibility of the evaluation function. Area and timing optimization can occur at many levels of technology-dependence. The type of area/delay estimate can be specified according to a tradeoff between accuracy and run-time efficiency.

Two models of technology-dependent optimization are used. The first maps the Boolean network into a set of simple gates (AND, NAND, OR, NOR, INV). The second method allows for the complex gate forms AND-OR, OR-AND, and exclusive-OR in addition to the simple gates. The definition of available gates and their area and delays are specified in a technology library.

For timing optimization, the cost of the network becomes a function of the critical path delays through the network. Given a mapping, approximate input to output propagation delays based on gate and fanout delays can be readily determined. The fanout delay model is an RC model based on the equivalent resistance of the driving gate and the total capacitance of the load gates. For an improvement in efficiency, a second model exists which assumes all gates have equivalent input capacitance. Both rise and fall delays are calculated for improved accuracy. External influences such as the arrival times and drive of the primary inputs and the primary output load can be specified.

The tradeoff between area and delay is reflected in the evaluation function by

assigning different weights to the different components.

5.4.1 Technology Mapping

In order to perform technology-dependent optimization, the Boolean network must be mapped into a network containing components available in the given technology. By using a direct map, this process is very efficient. If the mapping is one-to-one, transformations of the Boolean network are always reflected as transformations in the mapped network. Thus, the value of the rule is always defined. In mappings which are not one-to-one, a transformation may not reflect a change in the mapped circuit and cannot be effectively evaluated. The mapping performed by OPAL is *nearly* one-to-one. The reason it is not one-to-one concerns NAND trees. A *tree* is defined as a directed, acyclic graph under the constraint that all nodes have a fanout (or reference) count of one. In the case of simple gate mappings, the following two Boolean networks, represented algebraically, can both be mapped into a two-level tree of NAND gates:

$$
\begin{aligned}
X &= ab + cde + fg \\
&\textit{and} \\
X &= \overline{y} \\
Y &= \overline{m}\,\overline{n}\,\overline{o} \\
M &= ab \\
N &= cde \\
O &= fg
\end{aligned}
$$

However, since no single rule can transform either of these networks into the other, evaluation of a rule is always possible. The structure of the first Boolean network is more desirable than the second because more powerful transformations can be applied to it given the rule set in OPAL. In inverting logic styles, a NAND tree is much more area and time efficient than the more obvious AND-OR tree. By mapping and-or expressions into NAND trees, the manipulation of the Boolean network is not

hindered. The extent of using NAND gates over AND/OR gates is controlled by a parameter to the program.

The specifics of how a particular part of the Boolean network is mapped to a set of gates is largely dependent on the optimization model used. However, fundamental commonalities exist between all models. In all cases, only variable and product term nodes of the Boolean network can have a gate counterpart—literal nodes can only be associated with inputs to a gate. In addition, a product term (product term node) cannot be "shared" by multiple gates. That is, the conjunction of literals in a product term must be contained in a single gate mapping. However, a variable (variable node) can be shared between gates. For instance, the disjunction of product terms which define a variable can map to a gate as well as each of the product terms of that disjunction. Furthermore, any subnetwork of the Boolean network can be mapped to a single gate. Additionally, a legal mapping is defined to be the mapping obtained by starting from the primary outputs and successively mapping transitive fanin nodes. The *root* of a gate is defined as the variable or term node whose fanout corresponds to the output of the gate. Thus, the primary outputs (variable nodes) are always roots. Only one root is allowed per gate. This restriction prevents the mapping of multiple output gates.

The conditions mentioned above do not require a direct mapping. Further constraints provided by the optimization model are necessary to fully define the mapping. However, these conditions do provide a basis for the definition of incremental mapping.

Incremental Mapping

The time involved in mapping the entire network is proportional to the number of variable and term nodes in the Boolean network. Even this linear time complexity cannot be considered acceptable because the Boolean network must be mapped for each rule instantiation. It is obvious that each rule affects only a limited number of nodes in the Boolean network and therefore in the mapping. Remapping the entire network is duplicating work already performed. Fortunately, the set of affected nodes can be isolated and remapped without requiring the mapping of the remainder of the network nodes. The mapping process can thus be performed in constant time.

For simplicity, remapping occurs on a variable node basis. That is, if any variable's immediate fanin term nodes and their literals are affected by a rule, that variable and the immediate fanin term nodes are remapped. The following rules define when a variable node is remapped:

1. When a variable is created.
2. When a product term and/or literal of a product term for the variable is modified (created or removed).
3. If a variable's reference count becomes 1 and the technology mapping model allows complex gates, the fanout variable is remapped (if complex gates exist which can span more than two variable nodes, additional fanout variables are remapped).
4. If the variable's reference count becomes greater than 1 and if the variable node is internal to a complex gate tree, that variable and all variable node members of that complex gate are remapped.
5. If a variable's positive or negative fanout counts becomes zero or non-zero, that variable is remapped. Thus, remapping a node may effect the remapping of fanin nodes.

Variables to be remapped are recorded in a stack to approximate a desired ordering where transitive fanout variable nodes are mapped before predecessor nodes. This ordering of nodes is required to meet the conditions of the direct map. The algorithm for remapping the network is given in Figure 5.9.

mapping_model is the specific technology mapping model desired (e.g. simple gates). Maintaining the ordering constraint may result in a fanout variable being moved to the top of the stack. Rather than deleting the previous stack entry, the variable is duplicated in the stack. For each variable to be remapped, it is therefore necessary to determine if it was already processed. The for-loop ensures that the ordering constraint is met for the top-of-stack variable, V. V is marked as a root node by *makeroot* to prevent fanout nodes from being placed on the stack again. V is finally mapped when all fanout nodes have been mapped.

```
inc_map(mapping_model)
{    while (stack_empty(unmapped))
        {
        V = top(unmapped);
        if (mapped(V))
            pop(unmapped);
        else
            {
            foreach (fanout variable V_f of V)
                {
                if (mapped(V_f))
                    push(unmapped, V_f); /* maintain ordering */
                else if (V is not root node)
                    {
                    total_area -= area(V_f);
                    push(unmapped,V_f);
                    }
                }
            makeroot(V);
            if (top(unmapped) is V) /* i.e., all fanout nodes mapped */
                {
                pop(unmapped());
                map(mapping_model, V);
                }
            }
        }
}
```

Figure 5.9: Incremental Mapping Algorithm

5.4.2 Technology Mapping Models

OPAL has three mapping models. These include the simple gate and complex gate models as mentioned previously. In addition, there exists an AND-OR model which is used for technology-independent timing optimization. Technology-independent area optimization does not require mapping since the total number of literals is maintained in the database. However, mapping is required to perform network leveling which is described later.

The AND-OR Model

The AND-OR model is not really a technology-dependent mapping model but is presented here since it contains the same concepts as the more complex models. The three basic gates of this model are generic-AND, generic-OR, and generic-buffer. This model can be considered a dual-rail model because inverting and non-inverting outputs are present on the gates. In all models, a gate configuration which provides an inverting and non-inverting output must exist for all gates. This constraint is usually met by adding an inverter to the output of a gate available in the library. Each disjunction in the Boolean network is mapped to a generic-OR gate and each conjunction to a generic-AND. The generic-buffer gate is used when a variable contains a single product term which in turn contains a single literal. The area of the gates is irrelevant since the literal count is used as the technology-independent area estimate.

The Simple Gate Model

The simple gate model maps the Boolean network using only the available AND, NAND, OR, NOR, and inverter gates in the technology library. Because only these simple gates are considered, the mapping is very efficient. Inaccuracies are only introduced when complex gates would have provided a more area or delay efficient implementation.

Given just these few gates, there are many possible configurations that perform the same basic function. For instance, a NAND gate can be combined with an inverter to derive an AND gate. Fortunately, there is often a "best" configuration given the

particular fanout requirement of the gate. For instance, consider a CMOS standard cell technology. If a variable contains only positive fanout, the AND gate might be the desired configuration (possibly less area than a two-cell NAND-INV). If both negative and positive fanout exist, the NAND-INV configuration is better since it provides both the inverting and non-inverting outputs in less area than would the AND-INV configuration. These decisions are made once (by the library developer) and are stored in the technology library. The assumption of a "best" configuration fails when a tradeoff exists between drive capability and area. In such cases, the appropriate configuration can only be decided dynamically. The selection of a gate in the library is made according to the generic function of the gate (e.g. AND, OR), the fanin of the gate, and the positive and negative fanout counts.

Given a variable node to be remapped, there are three cases to be considered: the variable contains multiple product terms, a single product term, or, if it defines an input, no product terms. For multiple product terms, the variable node is the root of a sum-of-products tree and two possible mappings exist: a two-level NAND tree or an AND-OR tree. The decision to accept the NAND tree is based on the parameter *nand_nand_cutoff* specified in the user's configuration file. For each multiple product term variable, a sum of the number of literals less one of each term is calculated. If this sum is greater than *nand_nand_cutoff*, the NAND tree mapping is used. Low sums correspond to trees with only one literal in most product terms. These Boolean network configurations can be easily transformed by a rule into different configurations which are mapped identically. Thus the one-to-one correspondence in mapping is violated. When the sum is less than or equal to the cutoff, the AND-OR tree is used.

In the case of the AND-OR trees, the positive and negative fanout counts of the variable and term nodes are the positive and negative reference counts, respectively.[3] For NAND trees, the assignment is the opposite (e.g. positive fanout count set to negative reference). Thus, single literal terms are essentially inverted. Gate configurations are easily obtained from the library utilizing the fanout counts and the fanin requirements of the nodes.

[3]of course, term nodes always have a positive reference count of 1 and negative reference count of 0 due to the restrictions imposed by the Boolean network model

Variables defined with a single term containing a single literal are mapped to a buffer gate. According to the fanout counts, the gate will either be an inverting buffer or a non-inverting buffer.

The Complex Gate Model

Complex gates which match a tree structure in the Boolean network can be completely defined in the library. Currently, defining complex gates that are not of this structure requires compiled code in addition to the library specification. Typical libraries contain only a small number of gates (such as the exclusive-OR) which do not meet this criteria and thus optimization is not significantly weakened. To take advantage of gates not recognized by OPAL requires a postprocess technology mapping, and thus optimality may be lost.

Mapping a variable node proceeds by checking the library for a complex gate which matches the Boolean network tree structure rooted at the variable. The number of term nodes connected to the root node is used to reduce the number of gates which need to be checked. Priority among complex gates which may match the same network configuration is according to the definition order in the library. If no complex gates match the variable node, a simple gate mapping is determined using the simple gate model.

5.4.3 Delay Evaluation

In many situations, the logic network will be connected between register banks that are clocked according to the maximum path delay of the network. For critical networks, the configuration with the smallest maximum output delay represents the goal configuration state. For gates which are off the critical path, area is important and therefore always a part of timing optimization. OPAL uses a weighted sum of critical path delay and area as the evaluation function for timing optimization. In other situations, some outputs are considered more critical than others and must meet required arrival times. In these cases OPAL will minimize the sum of the differences between output arrival times and required arrival times for outputs which do not meet their

constraints.

The calculation of the output arrival times is performed in a quick two-pass process of network leveling and delay summing. During leveling, the gates of the mapped network are placed in a list. The order of placement is defined by the relation that a gate occurs in the list after the positions of all its transitive fanout gates. The network leveling process serves the dual purpose of connecting together the gates of the mapped network. Gate mapping only assigns gates to Boolean network nodes; connection among the gates is implied but not explicitly provided. Maintaining connectivity relationships between input/output pins of mapped gates during mapping is too expensive since it is only required for timing optimization. A standard critical path computation which performs leveling concurrently with delay computation must proceed from the Boolean network inputs to the outputs and would require substantial graph searching to extract gate connectivity. The network leveling pass proceeds from circuit outputs to inputs which is more suited to the structure of the Boolean network and mapping information. In addition, since only leveling is required for obtaining logic levels of the network, technology-independent optimization can be performed efficiently by omitting the delay summing pass.

The delay summing pass proceeds through the leveling list from the end of the list (the circuit inputs) to the beginning (the circuit outputs). The rise (fall) arrival times, a_i^r (a_i^f) for each gate output i are the sum of the maximum fanin arrival times a_j, the intrinsic delay of the gate I_i, and the fanout loading delays (loads L_i, drives D_i). The equations for computing a_i^r and a_i^f for positive unate, negative unate, and non-unate gate outputs are presented in Equations 5.2, 5.3, and 5.4 respectively.

$$\begin{aligned} a_i^r &= \max_{j \in FI(i)} (a_j^r) + I_i^r + \sum_{j \in FO(i)} L_j \cdot D_i^r \\ a_i^f &= \max_{j \in FI(i)} (a_j^f) + I_i^f + \sum_{j \in FO(i)} L_j \cdot D_i^f \end{aligned} \tag{5.1}$$

$$a_i^r = \max_{j \in FI(i)} (a_j^f) + I_i^r + \sum_{j \in FO(i)} L_j \cdot D_i^r \tag{5.2}$$

$$
\begin{aligned}
a_i^f &= \max_{j \in FI(i)} (a_j^r) + I_i^f + \sum_{j \in FO(i)} L_j \cdot D_i^f \\
a_i^r &= \max_{j \in FI(i)} (\max(a_j^r, a_j^f)) + I_i^r + \sum_{j \in FO(i)} L_j \cdot D_i^r \\
a_i^f &= \max_{j \in FI(i)} (\max(a_j^r, a_j^f)) + I_i^f + \sum_{j \in FO(i)} L_j \cdot D_i^f
\end{aligned}
\tag{5.3}
$$

The technology library contains capacitive loading and drive information for calculating fanout delays. The efficiency of fanout delay computation is improved if an average input capacitance can be assumed for all gates (e.g., $\forall$ outputs i, $L_i = 1.0$). The delay model used provides sufficient accuracy for obtaining optimal circuits in most cases. However, the evaluation function can easily support more accurate delay models if required.

5.5 The Technology Library

The technology library characterizes the various logic gates available for a particular technology. This library usually corresponds to gates which have been predefined for use with Standard Cell and Gate Array layouts. Because of the restrictions placed on technology mapping during cost evaluation, not all available gates can be utilized during logic synthesis. To take advantage of special gates, a postprocessing technology mapping step should be performed.

The technology library defines four classes of gates: AND, OR, BUFFER, and COMPLEX. For the first three classes, gate entries are classified according to fanin count. The COMPLEX gate class classifies entries according to the corresponding Boolean network tree. In all classes, a further classification according to fanout count exists:

1. positive fanout count > 0, negative fanout count = 0
2. positive fanout count = 0, negative fanout count > 0
3. positive fanout count > 0, negative fanout count > 0

For instance, a two-input AND gate configuration should be specified in case 1. In case 2, a two-input NAND gate is specified and in case 3, a two-input AND/NAND (both inverting and non-inverting outputs). If a configuration can be formed from an available gate and possibly an inverter, the gate configuration is said to be *legal*. Gate configurations which are *illegal* are usually specified with an area and delay which would be realistic if the configurations were available. Illegal configurations are commonplace early on in the synthesis process and are eventually eliminated as the penalties they incur are not tolerated by the acceptance function. As stated earlier, illegal configuration states improve the search by increasing the accessibility of the state-space. The specification of the library allows for a default entry which can be extrapolated to form gate configurations of fanin not specified explicitly. The default gate configuration can be used to specify illegal configurations beyond the "fanin limit" implied by the library.

The specification of a gate configuration contains the following information: area, input capacitance, rise/fall delays to negative and positive fanout, rise/fall drives of positive and negative fanouts, positive and negative dependence relationships, logic levels to positive and negative outputs, and the number of inverters required in the configuration. The only constraint on the units for input capacitance and drive are that they form time when multiplied together. The output delays are intrinsic delays except when one output is inverted to obtain the other. In this situation, the fanout load of the inverter is accounted for in the output delay. The non-inverting, or positive output, is said to *depend* on the inverting, or negative output, if it is derived from the negative output with an inverter. Likewise, the negative output depends on the positive output if it is formed with an inverter attached to the positive output. Since the fanout delay of the true output will affect the delay of the dependent output, dependency information is necessary.

Appendix A contains a typical library specification.

5.6 Synthesis Process Configuration

The user of OPAL can control and customize the synthesis process using parameters specified in a configuration file. Five parameter classes are defined: PHC process, network evaluation, rule base, network I/O, and reporting. Many of the PHC process and network evaluation parameters have already been described. In addition to these parameters, rule weighting can also be specified. OPAL can read and write networks specified as equations or in the LIF.3 logic interchange format[MCN87a]. Statistics can be generated at each value of T and also at the end of the synthesis run. The complete list of parameters is provided in Appendix B.

Chapter 6

System Performance

Presented in this chapter are performance comparisons between the Probabilistic Hill-Climbing Control Strategy and alternative strategies. Although these comparisons and evaluations are made with respect to performance in rule-based logic synthesis, the conclusions and insights obtained can be applied more generally. In addition to control strategy comparisons, the issue of rule-based versus algorithmic-based problem-solving is addressed. Results obtained with the rule-based system OPAL are compared to the MIS Logic Synthesis System.[BRSW87,DGR*87]

6.1 Comparison Between PHC and Alternative Strategies

The alternative control strategy model used in comparisons is the steepest-descent strategy with varying degrees of look-ahead. The traditional steepest-descent strategy is "greedy"—only the best rule is selected. Consequently, this approach obtains a locally optimal solution that is very sensitive to the initial problem configuration. Look-ahead provides a means of escaping local minima by accepting rules which are not individually acceptable. However, when associated with other rules in a sequence, a better overall configuration is obtained. This model was selected since it is used in most rule-based logic synthesis systems[GBdH86,DBG*84]. In addition, a spectrum of strategies can be constructed by adjusting the breadth and depth of the look-ahead

search. Each strategy will be defined by its associated breadth and depth pair: (b, d).

Five different steepest-descent/look-ahead strategies are used in comparisons: (1,1), (1,5), (n,1), (70,3), and (25,4). (1,1) represents a strategy which randomly tests rule-location pairs in the network and accepts only those which improve the cost. (1,5) tries sequences of 5 rules and accepts those which improve the cost. (n,1) accepts the best of n rule-location pairs. When n is the number of unique rule-location pairs for the current configuration, this strategy represents the standard "greedy" approach. (70,3) searches 3 rules deep and 70 rule-location pairs at each depth level—accepting the best three rule sequence. (25,4) performs more look-ahead but at the expense of fewer rule-location pairs per depth level.

Since only the control strategies are being compared, other rule-based system components such as the rule set and evaluation function are left unchanged. The only exception to this is that the weights applied to individual rules for the steepest-descent/look-ahead strategies differ from those of the PHC strategy and were adjusted experimentally for optimal performance.

The steepest-descent/look-ahead strategy models represent "pure" strategies since the breadth/depth parameters are static. Of course, it is possible to optimize performance for a particular problem by adjusting the breadth and depth during the process. For example, in logic optimization and synthesis, it may be desireable to use a minimal amount of look-ahead during early stages of the process before local minima are reached. Later stages may increase the look-ahead to help escape local minimas. However, the optimal breadth and depth to use on a given problem configuration *is* indeed the problem to be solved. *What this report intends to convey is that the Probabilistic Hill-Climbing control strategy provides a robust solution to this problem.*

The measure of performance between the various strategies is solution quality given comparable run-times. However, because each strategy continues until improvements in cost are no longer possible, obtaining comparable run-times is a difficult task. This problem is addressed by requiring the run-times for the steepest-descent/look-ahead strategies to be identical to the PHC strategy run-time. In some cases, the solution converges rapidly and is not helped much by the extra time provided. In

Name	Description	Format	Inputs	Outputs	Literals
5xp1-hdl	$5 \cdot x + 1$	multi-level	7	10	117
5xp1	$5 \cdot x + 1$	2-level	7	10	296
9sym-hdl	'1' if 3,4,5, or 6 inputs= 1	multi-level	9	1	169
9sym	'1' if 3,4,5, or 6 inputs= 1	2-level	9	1	522
9symml	'1' if 3,4,5, or 6 inputs= 1	multi-level	9	1	277
alupla	4 bit ALU w/muxes	multi-level	25	5	173
bw	N/A	2-level	5	28	324
f2	N/A	multi-level	4	4	36
f51m-hdl	$(5 \cdot x + 1) mod 256$	multi-level	8	8	116
f51m	$(5 \cdot x + 1) mod 256$	multi-level	8	8	155
misex1	control	multi-level	8	7	125
misex2	control	multi-level	25	18	166
rd53-hdl	counts 1's	multi-level	5	3	55
rd53	counts 1's	2-level	5	3	144
rd73-hdl	counts 1's	multi-level	7	3	91
wdcnt	counter	multi-level	7	8	114
z4ml-hdl	2-bit adder w/cin,cout	multi-level	7	4	68
z4ml	2-bit adder w/cin,cout	multi-level	7	4	58

Table 6.1: Benchmark Circuits

other cases, a better solution may have been obtained if the run continued. To indicate such discrepancy, the solution at convergence (and required run-time) for each strategy is also presented.

A summary description of the benchmark circuits is contained in Table 6.1. The circuits whose names end in "-hdl" were initially specified using a hardware description language rather than a sum-of-products (2-level) specification. Many of these examples are part of the Microelectronics Center of North Carolina (MCNC) benchmark set from the 1987 International Workshop on Logic Synthesis[MCN87b].

6.1.1 Technology-Independent Optimization

The least computationally expensive and least accurate optimization is literal count optimization. Results obtained for literal count optimization are presented in Tables 6.2 and 6.3. For each circuit benchmark, the initial area (literal count) is specified. For

each control strategy, optimized solution areas and required run-times are presented. Run-times here and in all comparisons with the exception of timing optimization are VAX 8650 minutes. Timing optimization run-times are in VAX 8800 minutes.[1] For the PHC runs, a best and average of 3 runs is presented. The purpose of averaging multiple runs is to provide an expected value for the solution quality. Multiple runs are not useful for the steepest-descent/look-ahead strategies since the effect of the random number sequence is less pronounced. *All PHC runs utilize the same set of process parameters but with different random number seeds.* The CPU_1 time corresponds to the best run. For each alternative strategy, two areas are specified. $area_1$ is the comparable run-time result. $area_2$ and CPU_2 represent the result when the run-time constraint is removed. Runs without the time constraint terminate after a fixed number of rule-location pairs have been unsuccessful in reducing the cost of the final configuration. This rule-location limit is based on the number of rules in the network, the initial size of the network, and an experimentally determined multiplier.

Area and CPU time totals for each control strategy are presented in Table 6.4.

For the steepest-descent/look-ahead strategies it is observed that better results are often obtained for the time limited runs over those of the unconstrainted runs. This behavior occurs when the time limit is greater than the unconstrained run-time, in which case the termination criterion is ignored. Since the termination criteria is based on the rate of rule-location pair acceptance rather than the exact, and computationally prohibitive, determination of a local minimum, it is very likely that the unconstrained runs will complete before an acceptable rule-location pair is found.

Comparing results among the steepest-descent/look-ahead strategies indicates that in many cases increasing the breadth/depth of the search decreases solution quality. The advantages gained by the ability to escape local minima do not overcome the disadvantage of accepting fewer rules. An examination of the rule sequences accepted by the extended look-ahead models indicates that most sequences consist only of cost-decreasing (improving) rules. The scarcity of local minima often renders the look-ahead useless. A practical implementation of the steepest-descent/look-ahead

[1] the VAX 8800 has been experimentally determined to be about 10% faster than the VAX 8650

Circuit		PHC				(1,1)			$(n,1)$		
name	*area*	*best*	CPU_1	*ave*	CPU_2	$area_1$	$area_2$	CPU_2	$area_1$	$area_2$	CPU_2
5xp1-hdl	117	81	4.8	83	4.8	88	88	0.4	104	104	1.2
5xp1	296	172	15.8	181	15.3	190	190	1.4	193	193	8.6
9sym-hdl	169	90	9.9	98	9.4	139	139	0.5	132	132	1.8
9sym	522	345	36.5	350	38.2	343	347	3.3	423	325	61.8
9symml	277	271	14.6	273	14.2	266	266	1.0	266	266	3.0
alupla	173	145	8.8	148	9.7	152	152	1.3	152	152	2.7
bw	324	251	18.6	254	18.3	235	235	0.9	224	224	6.7
f2	36	26	7.1	29	9.9	30	30	0.1	30	30	2.5
f51m-hdl	116	78	4.7	81	4.5	87	87	0.3	104	104	1.2
f51m	155	143	6.8	150	6.4	154	154	0.4	154	154	1.5
misex1	125	73	4.2	75	4.2	80	80	0.3	79	79	1.7
misex2	166	142	9.4	143	9.2	155	155	0.7	154	154	2.3
rd53-hdl	55	44	2.4	48	2.2	47	47	0.1	47	47	0.4
rd53	144	104	6.4	110	6.2	86	95	0.4	72	73	2.9
rd73-hdl	91	60	3.9	74	3.8	84	84	0.2	83	83	0.7
wdcnt	114	38	2.3	40	2.3	46	46	0.3	46	46	0.6
z4ml-hdl	68	52	3.1	55	2.7	55	55	0.2	54	54	0.5
z4ml	58	43	2.0	46	2.1	46	46	0.1	54	54	0.4

Table 6.2: Literal Count Optimization Results

strategy is suggested by these results: minimal look-ahead while many cost-decreasing rules exist; increased look-ahead as cost-increasing rules appear more frequently.

The Probabilistic Hill-climbing strategy provides the best individual and collective results of the strategies compared. This observation is not surprising given the conclusions concerning look-ahead strategies. Since cost-decreasing rules are always accepted, the PHC strategy contains characteristics of the $(1,1)$ model. However, the occasional acceptance of cost-increasing rules provides local minimal escapes and hence the better results. However, for several of the circuits, $(1,1)$ optimization provided a better solution than the PHC optimization. The acceptance of cost-increasing rules occasionally results in a move to a configuration subspace with a local minima of higher cost. Such behavior can be controlled by increasing the allowed run-time for the optimization or adjusting the process parameters.

Circuit		(1,5)			(70,3)			(25,4)		
name	*area*	$area_1$	$area_2$	CPU_2	$area_1$	$area_2$	CPU_2	$area_1$	$area_2$	CPU_2
5xp1-hdl	117	88	103	1.0	106	106	4.3	106	105	7.5
5xp1	296	213	215	9.2	208	182	50.7	223	216	38.1
9sym-hdl	169	135	146	2.2	148	149	7.3	153	148	7.7
9sym	522	377	385	15.1	373	368	62.1	373	366	59.4
9symml	277	266	269	3.9	270	270	15.0	270	270	19.7
alupla	173	152	152	1.9	154	154	7.4	154	155	10.7
bw	324	237	240	6.7	244	244	22.3	251	230	23.8
f2	36	28	30	0.1	34	28	2.5	33	30	1.4
f51m-hdl	116	92	103	1.3	106	106	4.3	108	105	4.8
f51m	155	154	155	9.2	154	154	7.8	155	154	5.6
misex1	125	77	78	1.2	76	76	5.0	74	80	6.8
misex2	166	166	153	1.6	122	119	15.0	139	126	15.2
rd53-hdl	55	47	51	0.3	51	53	1.6	52	51	1.6
rd53	144	87	88	1.5	96	95	11.4	110	76	18.7
rd73-hdl	91	84	88	0.3	83	83	3.9	86	87	2.0
wdcnt	114	42	44	0.5	48	44	5.0	46	46	2.2
z4ml-hdl	68	55	64	0.3	63	63	3.0	64	64	1.6
z4ml	58	53	57	0.2	57	57	1.0	57	58	0.8

Table 6.3: Literal Count Optimization Results (cont.)

	CPU limited		no CPU limit	
Strategy	area	CPU	area	CPU
PHC (best)	2159	155.1	2159	155.1
PHC (ave)	-	-	2240	154.3
(1,1)	2283	155.5	2296	11.7
(n,1)	2371	157.3	2274	98.2
(1,5)	2353	145.3	2421	48.2
(70,3)	2393	156.6	2351	229.8
(25,4)	2454	155.6	2367	227.7

Table 6.4: Totals for Literal Count Optimization

Gate	Area	Input Load	Delay		Output Drive	
			rise	fall	rise	fall
INV1	1.0	1.0	0.9	0.9	0.3	0.3
AND2	3.0	1.0	1.9	1.9	0.3	0.3
NAND2	2.0	1.0	1.0	1.0	0.1	0.1
NAND3	3.0	1.0	1.1	1.1	0.3	0.3
NAND4	4.0	1.0	1.4	1.4	0.4	0.4
NAND5	5.0	-	-	-	-	-
OR2	3.0	1.0	2.4	2.4	0.3	0.3
NOR2	2.0	1.0	1.4	1.4	0.5	0.5
NOR3	3.0	1.0	2.4	2.4	0.7	0.7
NOR4	4.0	1.0	3.8	3.8	1.0	1.0
XR	5.0	2.0	1.9	1.9	0.5	0.5
XN	5.0	2.0	2.1	2.1	0.5	0.5
AOI21	3.0	1.0	1.6	1.6	0.4	0.4
AOI22	4.0	1.0	2.0	2.0	0.4	0.4
OAI21	3.0	1.0	1.6	1.6	0.4	0.4
OAI22	4.0	1.0	2.0	2.0	0.4	0.4
AOI32	5.0	-	-	-	-	-
AOI222	6.0	-	-	-	-	-

Table 6.5: CMOS Standard Cell Gate Library

6.1.2 Technology-Dependent Optimization

The CMOS standard cell gate library described in Table 6.5 is used in technology-dependent optimizations. The gates containing delay information represent the MCNC benchmark set library and are used for timing optimization comparisons. The units for area and delay are transistor pairs and nanoseconds respectively. Results for area optimization are presented in Tables 6.6 and 6.7 and totals in Table 6.8.

The benefits of extended breadth/depth search are more apparent with mapped area optimization than with literal count optimization. Examination of the accepted rule sequences indicate considerably more cost-increasing rules were accepted than with literal count optimization. The effects of using rules with a fine granularity are more pronounced. Unlike the look-ahead models, the PHC control strategy demonstrates insensitivity to these effects. The superior results of the this strategy suggest

Circuit	PHC				(1,1)			(n,1)		
	best	CPU_1	*ave*	CPU_2	$area_1$	$area_2$	CPU_2	$area_1$	$area_2$	CPU_2
5xp1-hdl	104	9.7	1050	11.4	146	147	0.7	142	142	3.7
5xp1	242	46.3	247	56.9	300	325	3.1	293	258	19.5
9sym-hdl	125	22.0	155	21.3	209	209	1.3	206	201	3.6
9sym	460	129.5	478	120.9	541	596	6.9	423	447	115.0
9symml	344	51.9	348	53.7	346	356	3.2	346	358	16.5
alupla	190	23.8	196	23.2	204	208	2.1	213	220	3.3
bw	295	51.0	300	54.2	291	308	2.6	285	286	15.2
f2	35	2.0	38	1.8	42	42	0.2	48	50	0.4
f51m-hdl	100	9.6	104	11.0	143	144	1.0	148	148	3.0
f51m	181	13.5	197	16.2	204	208	1.2	210	210	2.8
misex1	89	9.9	90	9.5	102	103	0.7	103	107	1.8
misex2	169	26.3	174	29.8	200	217	1.5	203	216	6.6
rd53-hdl	60	4.5	63	4.3	69	71	0.2	68	68	0.7
rd53	133	17.7	137	19.3	154	159	1.2	103	109	5.5
rd73-hdl	103	9.2	112	7.9	116	115	0.4	117	117	1.1
wdcnt	44	5.2	49	5.2	64	64	0.4	61	63	1.1
z4ml-hdl	68	5.0	72	5.3	85	86	0.3	85	85	1.1
z4ml	48	4.2	62	4.1	71	71	0.3	75	75	0.6

Table 6.6: Area Optimization Results

that it was often more capable of avoiding local minima than the extended look-ahead models. In some sense, a greater *effective look-ahead* was achieved with the PHC control strategy.

Results for timing optimization are presented in Tables 6.9 and 6.10. Area and critical output delays are presented for PHC area optimization, PHC timing optimization, and (1,1) and (70,3) timing optimization. The drives for all circuit inputs and load of the circuit outputs were set to that of the INV1 gate. In all but the area optimization runs, the required arrival time optimization model was used.

The column *req'd* indicates desired output arrival times. These numbers were arbitrarily chosen with the criteria that they provide sufficient difficulty to the optimization. The only circuit where the required arrival time was met was alupla. In comparing timing optimization results, the tradeoff between area and delay must be considered. Runs with comparable output delays should be compared according to the areas of their solutions. The (1,1) model is clearly not as effective in timing

Circuit name	(1,5) $area_1$	(1,5) $area_2$	(1,5) CPU_2	(70,3) $area_1$	(70,3) $area_2$	(70,3) CPU_2	(25,4) $area_1$	(25,4) $area_2$	(25,4) CPU_2
5xp1-hdl	133	145	1.2	141	147	6.1	142	145	6.7
5xp1	280	305	10.2	304	301	57.0	325	348	25.8
9sym-hdl	201	206	2.2	207	207	11.7	206	204	18.4
9sym	535	570	18.1	495	542	88.4	541	556	75.8
9symml	351	364	3.2	346	366	15.1	347	365	13.9
alupla	205	210	2.6	205	220	10.4	220	226	6.2
bw	282	287	9.0	280	297	37.7	282	308	24.3
f2	36	42	0.2	41	35	7.5	39	38	1.8
f51m-hdl	142	146	1.3	146	152	5.2	139	146	8.8
f51m	202	207	2.0	209	212	10.7	219	213	10.6
misex1	102	103	1.6	102	102	9.2	102	102	6.0
misex2	199	205	3.4	180	184	46.3	183	201	18.3
rd53-hdl	65	68	0.3	68	69	1.5	68	68	2.0
rd53	139	154	1.5	148	146	13.1	151	156	15.6
rd73-hdl	116	119	0.4	116	118	3.8	115	119	2.9
wdcnt	63	67	0.6	68	70	6.3	67	71	2.3
z4ml-hdl	86	87	0.5	86	88	3.6	84	87	1.8
z4ml	68	72	0.6	72	75	4.2	72	78	2.5

Table 6.7: Area Optimization Results (cont.)

Strategy	CPU limited area	CPU limited CPU	no CPU limit area	no CPU limit CPU
PHC (best)	2790	441.3	2790	441.3
PHC (ave)	-	-	2920	456.0
(1,1)	3287	441.8	3429	27.28
(n,1)	3129	442.7	3160	201.7
(1,5)	3205	441.6	3357	58.9
(70,3)	3214	444.0	3331	337.9
(25,4)	3302	441.9	3431	243.6

Table 6.8: Totals for Area Optimization

circuit	Area Optimized area	delay	req'd	PHC area	delay	CPU_1
9sym-hdl	125	42.8	22	251	29.1	53.1
alupla	190	30.3	20	241	20.0	45.3
f51m-hdl	100	37.3	10	176	14.4	18.6
rd53-hdl	60	22.3	13	80	15.7	8.1
z4ml-hdl	68	15.1	8	89	10.9	3.3

Table 6.9: Timing Optimization Results - PHC

	(1,1)					(70,3)				
circuit	$area_1$	$delay_1$	$area_2$	$delay_2$	CPU_2	$area_1$	$delay_1$	$area_2$	$delay_2$	CPU_2
9sym-hdl	232	34.5	232	34.5	12.3	248	32.1	248	32.1	66.2
alupla	238	22.9	240	22.9	9.7	254	21.3	256	20.8	95.1
f51m-hdl	184	16.6	184	16.6	10.9	193	16.4	184	15.7	90.2
rd53-hdl	80	16.7	80	16.7	0.7	96	15.3	96	15.5	18.5
z4ml-hdl	87	11.1	87	11.1	0.8	96	9.9	92	9.2	32.0

Table 6.10: Timing Optimization Results - Look-ahead

optimization as it is in literal count optimization. This observation is again explained by the reduced effectiveness of single rules in improving the configuration. As could be expected, the (70,3) model was able to overcome local minima and obtain consistently good results. However, given comparable run-times, the PHC control strategy performed the best overall.

6.2 Comparisons Between Rule-Based and Algorithmic Logic Synthesis

Rule-based problem-solving can be expensive. Their use is justified when they provide better results than can be obtained with available algorithms. In recent years, the quality of algorithms for multi-level logic synthesis has improved enormously. The MIS logic synthesis system [BRSW87,DGR*87] incorporates many of these new algorithms. Presented here is a comparison between the OPAL rule-based system utilizing the PHC control strategy and the MIS system. Since algorithms for tim-

Circuit		PHC		MIS	
name	area	area	CPU	area	CPU
5xp1-hdl	117	81	4.8	83	0.1
5xp1	296	172	15.8	155	0.2
9sym-hdl	169	90	9.9	140	0.4
9sym	522	345	36.5	261	0.9
9symml	277	271	14.6	253	0.9
alupla	173	145	8.8	179	1.5
bw	324	251	18.6	224	0.7
f2	36	26	7.1	32	0.0
f51m-hdl	116	78	4.7	78	0.1
f51m	155	143	6.8	160	0.3
misex1	125	73	4.2	72	0.1
misex2	166	142	9.4	113	0.1
rd53-hdl	55	44	2.4	54	0.1
rd53	144	104	6.4	62	0.1
rd73-hdl	91	60	3.9	80	0.2
wdcnt	114	38	2.3	41	0.0
z4ml-hdl	68	52	3.1	59	0.1
z4ml	58	43	2.0	42	0.1

Table 6.11: Literal Count Optimization Results for OPAL and MIS

ing optimization are still in their infancy, only literal count and mapped area results can be compared. Literal count optimization results are compared in Table 6.11 and mapped area optimization results in Table 6.12.

Although OPAL was often able to provide a better solution, it came at the cost of one to two orders of magnitude increase in run-time. Furthermore, the quality of results from algorithmic approaches *can be* considerably better—at the cost of increased CPU time—if more extensive Boolean operations are performed. Rule-based systems for area optimization may no longer be cost effective given the quality of algorithms currently available. However, due to the present lack of algorithms for timing optimization, the need for rule-based approaches to this problem is still present. Furthermore, it is unclear how successful algorithmic approaches will be when multiple constraints are introduced in the problem. Algorithmic approaches tend to separate the problem into subproblems which are solved individually. Often, the effectiveness of

Circuit name	PHC		MIS	
	area	CPU	area	CPU
5xp1-hdl	104	9.7	101	0.1
5xp1	242	46.3	181	0.3
9sym-hdl	125	22.0	182	0.5
9sym	460	129.5	328	1.0
9symml	344	51.9	310	1.0
alupla	190	23.8	219	0.7
bw	295	51.0	271	0.7
f2	35	2.0	36	0.0
f51m-hdl	100	9.6	96	0.1
f51m	181	13.5	191	0.4
misex1	89	9.9	90	0.1
misex2	169	26.3	152	0.2
rd53-hdl	60	4.5	69	0.1
rd53	133	17.7	77	0.1
rd73-hdl	103	9.2	104	0.2
wdcnt	44	5.2	54	0.1
z4ml-hdl	68	5.0	75	0.1
z4ml	48	4.2	53	0.1

Table 6.12: Mapped Area Optimization Results for OPAL and MIS

this technique decreases as the number of separations increases. Current separations introduce the following major subproblems: two-level logic minimization, decomposition, resynthesis for timing, decomposition into a canonical 2-input nand gate structure, mapping into a technology, and gate-sizing [BRSW87,BMS2,Keu87,DGR*87]. With the exception of gate-sizing, *all* of these subproblems are considered simultaneously by OPAL. Because optimizations which span these subproblems are not usually possible, consideration of the entire problem can potentially obtain better solutions. One algorithmic approach to spanning across problem divisions involves iterating each step until some convergence criteria is achieved. As this technique suggests, algorithmic methods which attempt to overcome non-optimal problem divisions possess a structure and efficiency approaching that of rule-based systems.

Chapter 7

Conclusions

A particularly interesting class of rule-based problem-solving involves performing searches of a complex state-space. Various methods, such as hill-climbing and best-first search, exist for controlling the state-space search and were presented. Problems with these methods have been described and used as justification for the appropriateness of the Probabilistic hill-climbing (PHC) control strategy. A model of PHC-controlled rule-based systems was described and the impact of the control strategy on aspects of the system was addressed. A rule-based system for logic synthesis, OPAL, has been developed which utilizes the simulated annealing PHC control strategy.

Comparisons with different models of the steepest-descent/look-ahead control strategy indicate the robustness of the PHC strategy. Overall, the PHC control strategy provides better solutions than can be achieved by fixed breadth/depth look-ahead models. Because the degree of necessary look-ahead varies across problem instances and for different evaluation functions, static look-ahead approaches are not very robust. The PHC control strategy is able to dynamically *adapt* its search to suit the problem being solved. Thus, PHC control strategies are able to achieve a more effective search of the state-space and obtain better solutions.

Finally, the issue of Rule-based versus algorithmic approaches was addressed by considering the logic synthesis problem. Although solution quality of the two approaches are comparable, the required execution time for the rule-based approach was identified as prohibitively expensive. However, some problems, especially those

with many constraining factors, can not be solved optimally when separated into subproblems which are solved individually. When optimal results are required, the rule-based paradigm is a useful approach. Rule-based systems for logic synthesis are useful when a) there exists a lack of good algorithms for a given subproblem (such as timing optimization), and b) many conflicting constraints are imposed on the synthesis process which must be dealt with simultaneously.

Important work left for future study include improvements in timing optimization. In particular, the currently existing rule base in OPAL appears to be best suited for area optimization. The value of individual rules can not be easily determined independently of other rules. Rules suited for timing optimization should be investigated and incorporated into the existing rule base. Also, the sharing of common subexpressions is not always desirable when delays are being minimized. Currently OPAL *always* shares common subexpressions. Methods for identifying when this operation is not desirable should be developed and incorporated.

Future work on PHC-based rule-based systems should investigate automatic methods for obtaining optimal sets of parameter values for a given evaluation function (e.g., area optimization, timing and area optimization, etc). In addition, automatic techniques for obtaining rule weights should be investigated. Rule weighting can improve both the quality of solutions as well as the efficiency of the search. One possible approach involves having the program discover for itself the appropriate set of weights by obtaining feedback from solutions to selected instances of the problem to be solved. Typical information which would be useful for this purpose are the frequency of rule matches, the accept/reject ratio at different stages of the process, and the sequences of rules which are often effective.

Appendix A

Technology Library Example

```
#
# enhanced MCNC benchmark library
# CMOS standard-cell
#

AND 5 5 1 0.6 0.6

   0
   # exists     legal unate  load    delay              drive          dep lvl invs
   #       name          area      pos      neg      pos      neg      p n p n
   # - ------------ - - --- --- ------- ------- ------- ------- - - - - -
     Y  AND_ILLEGAL N P 6.0 1.0 3.2 3.2  0.0 0.0  0.3 0.3  0.0 0.0  0 0 2 0 1
     Y NAND_ILLEGAL N P 5.0 1.0 0.0 0.0  1.8 1.8  0.0 0.0  0.5 0.5  0 0 0 1 0
     Y NAND_ILLEGAL N P 6.0 1.0 3.2 3.2  2.2 2.2  0.3 0.3  0.5 0.5  0 1 2 1 1
   2
     Y AND2         Y P 3.0 1.0 1.9 1.9  0.0 0.0  0.3 0.3  0.0 0.0  0 0 2 0 0
     Y NAND2        Y P 2.0 1.0 0.0 0.0  1.0 1.0  0.0 0.0  0.2 0.2  0 0 0 1 0
     Y NAND2        Y P 3.0 1.0 2.1 2.1  1.2 1.2  0.3 0.3  0.2 0.2  0 1 2 1 1
   3
     Y NAND3        Y P 4.0 1.0 2.3 2.3  0.0 0.0  0.3 0.3  0.0 0.0  0 0 2 0 1
     Y NAND3        Y P 3.0 1.0 0.0 0.0  1.1 1.1  0.0 0.0  0.3 0.3  0 0 0 1 0
     Y NAND3        Y P 4.0 1.0 2.3 2.3  1.4 1.4  0.3 0.3  0.3 0.3  0 1 2 1 1
```

```
4
  Y NAND4        Y P 5.0 1.0 2.7 2.7  0.0 0.0  0.3 0.3  0.0 0.0  0 0 2 0 1
  Y NAND4        Y P 4.0 1.0 0.0 0.0  1.4 1.4  0.0 0.0  0.4 0.4  0 0 0 1 0
  Y NAND4        Y P 5.0 1.0 2.7 2.7  1.8 1.8  0.3 0.3  0.4 0.4  0 1 2 1 1
5
  Y NAND5        Y P 6.0 1.0 3.2 3.2  0.0 0.0  0.3 0.3  0.0 0.0  0 0 2 0 1
  Y NAND5        Y P 5.0 1.0 0.0 0.0  1.8 1.8  0.0 0.0  0.5 0.5  0 0 0 1 0
  Y NAND5        Y P 6.0 1.0 3.2 3.2  2.2 2.2  0.3 0.3  0.5 0.5  0 1 2 1 1


OR 4 4 1 2.0 2.0

  0
  # exists     legal unate  load    delay            drive        dep lvl invs
  #       name          area      pos      neg      pos      neg    p n p n
  # - ------------ - - --- --- ------- -------  ------- -------  - - - - -
    Y OR_ILLEGAL   N P 5.0 1.0 5.7 5.7  0.0 0.0  0.3 0.3  0.0 0.0  0 0 2 0 1
    Y NOR_ILLEGAL  N P 4.0 1.0 0.0 0.0  3.8 3.8  0.0 0.0  1.0 1.0  0 0 0 1 0
    Y NOR_ILLEGAL  N P 5.0 1.0 5.7 5.7  3.8 3.8  0.3 0.3  1.0 1.0  0 1 2 1 1
  2
    Y OR2          Y P 3.0 1.0 2.4 2.4  0.0 0.0  0.3 0.3  0.0 0.0  0 0 2 0 0
    Y NOR2         Y P 2.0 1.0 0.0 0.0  1.4 1.4  0.0 0.0  0.5 0.5  0 0 0 1 0
    Y NOR2         Y P 3.0 1.0 2.4 2.4  1.9 1.9  0.3 0.3  0.5 0.5  0 1 2 1 1
  3
    Y NOR3         Y P 4.0 1.0 4.0 4.0  0.0 0.0  0.3 0.3  0.0 0.0  0 0 2 0 1
    Y NOR3         Y P 3.0 1.0 0.0 0.0  2.4 2.4  0.0 0.0  0.7 0.7  0 0 0 1 0
    Y NOR3         Y P 4.0 1.0 4.0 4.0  3.1 3.1  0.3 0.3  0.7 0.7  0 1 2 1 1
  4
    Y NOR4         Y P 5.0 1.0 5.7 5.7  0.0 0.0  0.3 0.3  0.0 0.0  0 0 2 0 1
    Y NOR4         Y P 4.0 1.0 0.0 0.0  3.8 3.8  0.0 0.0  1.0 1.0  0 0 0 1 0
    Y NOR4         Y P 5.0 1.0 5.7 5.7  4.8 4.8  0.3 0.3  1.0 1.0  0 1 2 1 1
BUFFER 2 1 0 0 0
# 0 corresponds to primary outputs, 1 for internal buffers
  0
```

```
# exists     legal unate  load    delay            drive       dep lvl invs
#       name          area      pos      neg      pos      neg    p n p n
# - ------------ - - --- --- ------- ------- ------- ------- - - - - -
  Y OUTPUT       Y P 0.0 0.0 0.0 0.0  0.0 0.0  0.0 0.0  0.0 0.0  0 0 0 0 0
  Y OUTPUT-INV   Y P 1.0 1.0 0.0 0.0  0.9 0.9  0.0 0.0  0.3 0.3  0 0 0 1 0
  Y OUTPUT-INV   Y P 1.0 1.0 0.0 0.0  0.9 0.9  0.0 0.0  0.3 0.3  0 1 0 1 0
1
  Y INV1-INV1    Y P 2.0 1.0 2.1 2.1  0.0 0.0  0.3 0.3  0.0 0.0  0 0 2 0 1
  Y INV1         Y P 1.0 1.0 0.0 0.0  0.9 0.9  0.0 0.0  0.3 0.3  0 0 0 1 0
  Y INV1-INV1    Y P 2.0 1.0 2.1 2.1  1.2 1.2  0.3 0.3  0.3 0.3  0 1 2 1 1

# format for complex gate definitions:
#
#     <graph>        ::= <termlist> <gatecode>
#     <termlist>     ::= <termcnt> [<term_info> ...]
#     <term_info>    ::= 0 # literal cnt doesn't matter
#                    ::= -<literal_cnt> # indicates graph termination
#                    ::= <literal_cnt> <literal_info> ...
#     <literal_cnt> ::= natural_int
#     <literal_info>::= <sign> <termlist>
#     <sign>         ::= U # undefined, reqd for > 1 lit
#                    ::= P # positive
#                    ::= N # negative
#     <gatecode>     ::= 0 # no special function
#                    ::= int # corresponds to special gate#
#
# note: term_info list is as long as termcnt specifies
#       literal_info list is as long as literal_cnt specifies.
#       undefined sign required for terms with > 1 literal (otherwise ambiguous)
# gate codes:
#    1 - xor/xnor
#


COMPLEX 10 3 0 0 0
```

```
2 -2 -2 1
# exists     legal unate  load    delay            drive       dep lvl invs
#       name         area      pos      neg      pos      neg    p n p n
# - ------------ - - --- --- -------  -------  -------  -------  - - - - -
  Y XR           Y U 5.0 2.0 1.9 1.9  0.0 0.0  0.5 0.5  0.0 0.0  0 0 1 0 0
  Y XN           Y U 5.0 2.0 0.0 0.0  2.1 2.1  0.0 0.0  0.5 0.5  0 0 0 1 0
  Y XR           Y U 6.0 2.0 1.9 1.9  3.3 3.3  0.5 0.5  0.3 0.3  1 0 1 2 1

2 -2 -2 0
# exists     legal unate  load    delay            drive       dep lvl invs
#       name         area      pos      neg      pos      neg    p n p n
# - ------------ - - --- --- -------  -------  -------  -------  - - - - -
  N
  Y AOI22        Y P 4.0 1.0 0.0 0.0  2.0 2.0  0.0 0.0  0.4 0.4  0 0 0 1 0
  N

2 -2 -1 0
# exists     legal unate  load    delay            drive       dep lvl invs
#       name         area      pos      neg      pos      neg    p n p n
# - ------------ - - --- --- -------  -------  -------  -------  - - - - -
  N
  Y AOI21        Y P 3.0 1.0 0.0 0.0  1.6 1.6  0.0 0.0  0.4 0.4  0 0 0 1 0
  N

2 -1 -2 0
# exists     legal unate  load    delay            drive       dep lvl invs
#       name         area      pos      neg      pos      neg    p n p n
# - ------------ - - --- --- -------  -------  -------  -------  - - - - -
  N
  Y AOI21        Y P 3.0 1.0 0.0 0.0  1.6 1.6  0.0 0.0  0.4 0.4  0 0 0 1 0
  N

1 2 U 2 -1 -1 U 2 -1 -1 0
# exists     legal unate  load    delay            drive       dep lvl invs
#       name         area      pos      neg      pos      neg    p n p n
```

```
# - ------------ - - --- --- ------- ------- ------- ------- - - - - -
  N
  Y OAI22        Y P 4.0 1.0 0.0 0.0  2.0 2.0  0.0 0.0  0.4 0.4  0 0 0 1 0
  N

1 2 U 2 -1 -1 U 0 0
# exists     legal unate  load    delay            drive       dep lvl invs
#       name         area      pos      neg      pos      neg    p n p n
# - ------------ - - --- --- ------- ------- ------- ------- - - - - -
  N
  Y OAI21        Y P 3.0 1.0 0.0 0.0  1.6 1.6  0.0 0.0  0.4 0.4  0 0 0 1 0
  N

1 2 U 0 U 2 -1 -1 0
# exists     legal unate  load    delay            drive       dep lvl invs
#       name         area      pos      neg      pos      neg    p n p n
# - ------------ - - --- --- ------- ------- ------- ------- - - - - -
  N
  Y OAI21        Y P 3.0 1.0 0.0 0.0  1.6 1.6  0.0 0.0  0.4 0.4  0 0 0 1 0
  N

2 -2 -3 0
# exists     legal unate  load    delay            drive       dep lvl invs
#       name         area      pos      neg      pos      neg    p n p n
# - ------------ - - --- --- ------- ------- ------- ------- - - - - -
  N
  Y AOI32        Y P 5.0 1.0 0.0 0.0  2.4 2.4  0.0 0.0  0.4 0.4  0 0 0 1 0
  N

2 -3 -2 0
# exists     legal unate  load    delay            drive       dep lvl invs
#       name         area      pos      neg      pos      neg    p n p n
# - ------------ - - --- --- ------- ------- ------- ------- - - - - -
  N
  Y AOI32        Y P 5.0 1.0 0.0 0.0  2.4 2.4  0.0 0.0  0.4 0.4  0 0 0 1 0
```

```
  N

3 -2 -2 -2 0
# exists     legal unate  load    delay              drive        dep lvl invs
#       name          area      pos      neg      pos      neg    p n p n
# - ------------ - - --- --- ------- ------- ------- ------- - - - - -
  N
  Y AOI222       Y P 6.0 1.0 0.0 0.0  2.8 2.8  0.0 0.0  0.4 0.4  0 0 0 1 0
  N
```

Appendix B

Configuration Parameters

alpha1 temperature update during melt phase.

alpha2 temperature update during main search phase.

alpha3 temperature update during greedy search phase.

delta multiplier for standard deviation of network cost to specify equilibrium bound.

p_accept desired probability of accepting a cost-increasing rule at $T = T_0$.

illegal_gate_penalty upper bound multiplier for determining penalty for an illegal gate.

delay_penalty penalty for delay beyond required arrival times.

rule_tries the number of rules to attempt per location (rule-location selection method #1).

step_size_factor inner loop iteration count factor.

literal_threshold used to determine minimum inner count.

constant_iterations required number of outer loop iterations with a constant cost.

upper_bound_multiplier used to obtain the cost upper bound.

nand_nand_cutoff specifies when an and-or tree should be implemented as a NAND tree.

logic_level_factor multiplier for logic levels.

delay_factor multiplier for delay.

area_factor multiplier for area.

rule_weight_precision precision of specified rule weights.

literal_weight literal multiplier.

single_load load of one input if using common load fanout model.

breadth search breadth for steepest-descent/look-ahead control strategy.

depth search depth for steepest-descent/look-ahead control strategy.

any_loc_prob probability of generating a random network location.

local_loc_prob probability of generating a location near the previously accepted rule.

critical_loc_prob probability of generating a location on the critical path.

verbosity extent of result and diagnostic reporting.

inputs_avail default input availability (negative, positive, or both phases).

bound_mode cost bounding mode.

simple_fanout selects efficient, common load fanout model.

do_ruleuse if rule statistics should be generated.

preoptimize if input network should be stripped of extra buffers.

output_mode netlist output format.

input_mode netlist input format.

random_seed seed for random number generator.

area_optimization area optimization model.

delay_optimization timing optimization model.

do_resources if resource usage report should be generated.

loop_mode method of generating inner loop iteration limit.

selection_method selects the rule-location selection method.

rule_weight assigns a weight to a particular rule.

cpu_limit maximum number of CPU seconds permitted.

literal_cnt_type type of literal count (sum-of-products or factored form).

process_circuit if optimization should be performed.

control_strategy type of control strategy (PHC or steepest-descent/look-ahead).

timing_opt_model timing optimization model (required arrival, critical path delay).

checkpoint_rate number of outer loop iterations between checkpoint writes.

library_file name of the technology library file.

input_file name of the input file.

Bibliography

[AJMS84] C. R. Aragon. D. S. Johnson, L. A. McGeoch, and C. Schevon. Simulated annealing performance studies. In *Proceedings of the Workshop on Statistical Physics in Engineering and Biology*, April 1984.

[Ama68] Saul Amarel. On representations of problems of reasoning about actions. In Donald Michie, editor, *Machine Intelligence 3*, pages 131–171, Edinburgh University Press, 1968.

[BAE*83] Ronald J. Brachman, Saul Amarel, Carl Engelman, Robert S. Engelmore, Edward A. Feigenbaum, and David E. Wilkins. What are expert systems? In Frederick Hayes-Roth, Donald A. Waterman, and Douglas B. Lenat, editors, *Building Expert Systems*, chapter 1, pages 31–57, Addison-Wesley, 1983.

[BHH*82] R. Brayton, G. D. Hachtel, L. Hemachandra, A. R. Newton, and A. L. Sangiovanni-Vincentelli. A comparison of logic minimization strategies using ESPRESSO. an apl program package for partitioned logic minimizatoin. In *Proceedings of the International Symposium on Circuits and Systems*, pages 43–49, Rome, May 1982.

[BM82] R. Brayton and C. McMullen. The decomposition and factorization of boolean expressions. In *Proceedings of the 1982 IEEE International Symposium on Circuits and Systems*, pages 42–54, 1982.

[BRSW87] Robert K. Brayton, Richard Rudell, Alberto Sangiovanni-Vincentelli, and Albert R. Wang. MIS: a multiple-level logic optimization system. *IEEE Transactions on Computer-Aided Design*, 6(6):1062–1081, November 1987.

[DBG*84] J. A. Darringer, D. Brand, J. Gerbi, W. Joyner, Jr., and L. Trevillyan. Lss: a system for production logic synthesis. *IBM Journal of Research and Development*, 28(5):537–545, 1984.

[dC85] Aart de Geus and William Cohen. A rule-based system for optimizing combinational logic. *IEEE Design and Test*, (8):22–32, 1985.

[DGR*87] Ewald Detjens, Gary Gannot, Richard Rudell, Alberto Sangiovanni-Vincentelli, and Albert Wang. Technology mapping in mis. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 116–119, 1987.

[DK76] Randall Davis and J. King. An overview of production systems. In E. W. Elcock and Donald Michie, editors, *Machine Intelligence 8*, pages 300–332, Wiley, 1976.

[DN86] Srinivas Devadas and A. Richard Newton. Genie: a generalized array optimizer for VLSI synthesis. In *Proceedings of the 23rd Design Automation Conference*, pages 631–637, 1986.

[Feg79] Edward A. Fegenbaum. Themes and case studies of knowledge engineering. In Donald Michie, editor, *Expert Systems in the Micro-electronic Age*, pages 3–25, Edinburgh University Press, 1979.

[GBdH86] David Gregory, Karen Bartlett, Aart de Geus, and Gary Hachtel. Socrates: a system for automatically synthesizing and optimizing combinational logic. In *Proceedings of the 23rd Design Automation Conference*, pages 79–85, 1986.

[Gon86] G. Gonsalves. Logic synthesis using simulated annealing. In *Proceedings of the International Conference on Computer Design*, pages 561–564, 1986.

[Hay85] S. A. Hayward. Is a decision tree an expert system? In M. A. Bramer, editor, *Research and Development in Expert Systems*, pages 185–192, Cambridge University Press, 1985.

[HCO74] S. J. Hong, R. G. Cain, and D. L. Ostapko. Mini: a heuristic approach for logic minimization. *IBM Journal of Research and Development*, 18:443–458, September 1974.

[HRS86] M. D. Huang, F. Romeo, and A. Sangiovanni-Vincentelli. An efficient general cooling schedule for simulated anneailing. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 381–384, IEEE Computer Society Press, 1986.

[Joo85] Rostam Joobbani. *WEAVER: An Application of Knowledge-based Expert Systems to Detailed Routing of VLSI Circuits.* PhD thesis, Carnegie-Mellon University, June 1985.

[Keu87] K. Keutzer. DAGON: technology binding and local optimization by DAG matching. In *Proceedings of the 24th ACM/IEEE Design Automation Conference*, pages 341–347, 1987.

[KGv83] S. Kirkpatrick, C. Gelatt, Jr., and M. vecchi. Optimization by simulated annealing. *Science*, 220:671–680, May 1983.

[Kv81] S. Kang and W.M. vanCleemput. Automatic PLA synthesis from a DDL-P description. In *Proceedings of the 18th Design Automation Conference*, pages 391–397, 1981.

[LD86] J. Lam and J. Delosme. Logic minimization using simulated annealing. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 348–351, 1986.

[McC56] E. J. McCluskey. Minimization of boolean functions. *Bell System Technical Journal*, 35:1417–1444, April 1956.

[MCN87a] MCNC. International workshop on logic synthesis benchmark library format. Microelectronics Center of North Carolina, May 1987. Available from MCNC.

[MCN87b] MCNC. International workshop on logic synthesis benchmark set. Microelectronics Center of North Carolina, May 1987. Distributed to attendees by Tom Krakow, MCNC.

[MF78] J. McDermott and C. Forgy. Production system conflict resolution strategies. In D.A. Waterman and Frederick Hayes-Roth, editors, *Pattern-Directed Inference Systems*, pages 177–199, Academic Press, 1978.

[MRRT53] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, and A.H. Teller. Equation of state calculations by fast computing machines. *Chem. Phys.*, 21:1087, 1953.

[RS85] Fabio Romeo and Alberto Sangiovanni-Vincentelli. Probabilistic hill-climbing algorithms: properties and applications. In H. Fuchs, editor, *Proceedings of the 1985 Chapel Hill Conference on Very Large Scale Integration*, Computer Sciences Press, Chapel Hill, NC, 1985.

[Sas86] T. Sasao. MACDAS: multi-level AND-OR circuit synthesis using two-variable function generators. In *Proceedings of the 23rd Design Automation Conference*, pages 86–93, 1986.

[SS84] Carl Sechen and Alberto Sangiovanni-Vincentelli. The TimberWolf placement and routing package. In *Proceedings of the Custom Integrated Circuit Conference*, May 1984.

[SS86] Carl Sechen and Alberto Sangiovanni-Vinceltelli. TimberWolf3.2: a new standard cell placement and global routing package. In *Proceedings of the 23rd Design Automation Conference*, pages 432–439, 1986.

[VK83] M. P. Vecchi and S. Kirkpatrick. Global wiring by simulated annealing. *IEEE Transactions on Computer-Aided Design*, 2(4), October 1983.

[WH78] D. A. Waterman and Fredeick Hayes-Roth. An overview of pattern-directed inference systems. In D.A. Waterman and Frederick Hayes-Roth, editors, *Pattern-Directed Inference Systems*, pages 3–22, Academic Press, 1978.

[Zuc78] Steven W. Zucker. Production systems with feedback. In D.A. Waterman and Frederick Hayes-Roth, editors, *Pattern-Directed Inference Systems*, pages 539–555, Academic Press, 1978.