Probabilistic Constructive Optimization Techniques

Jennifer L. Wong, *Student Member, IEEE*, Farinaz Koushanfar, *Student Member, IEEE*, Seapahn Megerian, and Miodrag Potkonjak, *Member, IEEE*

***Abstract*—We have developed a new optimization paradigm for solving computationally intractable combinatorial optimization and synthesis problems. The technique, named probabilistic constructive, combines the advantages of both constructive and probabilistic optimization mechanisms. Since it is a constructive approach, it has a relatively short runtime and is amenable for the inclusion of insights through heuristic rules. The probabilistic nature facilitates a flexible tradeoff between runtime and the quality of solution, suitability for the superimposition of a variety of control strategies, and simplicity of implementation. After presenting the generic technique, we apply it to a generic NP-complete problem (maximum independent set) and a synthesis and compilation problem (sequential code covering). Extensive experimentation indicates that the new approach provides very attractive tradeoffs between the quality of solution and runtime, often outperforming the best previously published approaches.**

***Index Terms*—Graph coloring, maximum independent set, optimization algorithm, sequence covering.**

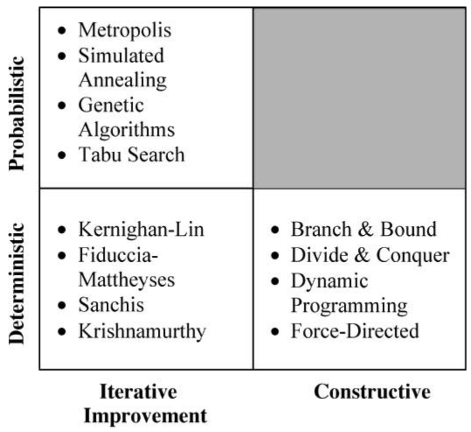


Fig. 1. Classification of optimization algorithms.

I. Introduction

A. Motivation

IN ORDER TO develop effective synthesis software, a number of components need to be in place. For example, one has to build proper abstractions of synthesis problems that capture the important features of the problem and eliminate the nonimportant ones, and build models that accurately characterize the design parameters such as delay, area, and early power prediction. Also, synthesis software must be modular and written in such a way that it can be easily reused and modified. Furthermore, there is a strong demand for convenient and intuitive user interfaces that simplify the designer's interaction with computer-aided design (CAD) tools during the design process. While the list of desired CAD software properties is long, at the heart of all synthesis software are optimization algorithms for solving computationally intractable problems. In the market place, the most important decision factor for purchasing a certain tool is its performance on standard benchmarks. Similarly, in the research world, one of the most important factors in judging new synthesis techniques is the experimental results with respect to the previously published papers on the same set of benchmarks. Therefore, it is not surprising that historically the CAD community has placed a strong emphasis on developing efficient algorithms [1].

Optimization algorithms used for synthesis have a great variety of features and, therefore, are difficult to be addressed in a fully systematic way. Fig. 1 shows the classification optimization algorithm according to two main criteria: 1) the way in which the solution is built and 2) the presence or absence of randomness during optimization. More specifically, all algorithms can be classified as either deterministic or probabilistic in one dimension, and as constructive or iterative improvement in the other dimension. The largest group of algorithms are constructive deterministic. For example, many CAD algorithms are based on the force-directed paradigm [2] or use dynamic programming [3]. In the last three decades, deterministic iterative improvement algorithms [4] were proposed for many problems and were able to produce excellent results. Simulated annealing [5] and other probabilistic iterative improvement approaches have attracted a great deal of attention for solving CAD problems. Techniques such as genetic algorithms, tabu search, and simulated evolution, due to their programming simplicity and flexibility, have been used for a variety of synthesis tasks. Their main disadvantage, however, is usually long runtime [6].

While numerous algorithms populate three of the quadrants in Fig. 1, the probabilistic constructive (PC) quadrant appears empty. There are some algorithms that can be interpreted in a way that is close in spirit to this quadrant (e.g., randomized deterministic algorithms [7]). Our goal is to explore techniques to develop algorithms which are simultaneously constructive and probabilistic, by leveraging on the noble properties of both constructive and probabilistic algorithms. The main advantage of constructive algorithms is their relatively short runtime and flexibility to incorporate a variety of insights as efficient heuristics.

Manuscript received July 19, 2002; revised February 25, 2003. This paper was recommended by Associate Editor R. Gupta.

J. L. Wong and M. Potkonjak are with the Computer Science Department, University of California, Los Angeles, CA 90095-1596 USA (e-mail: jwong@cs.ucla.edu).

F. Koushanfar is with the Electrical Engineering and Computer Science Departement, University of California, Berkeley, CA 94720 USA.

S. Megerian is with the Electrical and Computer Engineering Department, University of Wisconsin, Madison, WI 53706 USA.

Digital Object Identifier 10.1109/TCAD.2004.828136

On the other hand, the main advantage of probabilistic algorithms is their inherent flexibility that facilitates the tradeoff between quality of solution and runtime. They are also suitable for augmentation with a variety of control strategies such as multistart and delayed binding.

The new approach can be explained at the intuitive level in the following way. We start by searching for a small part of the problem that can be solved effectively, in such a way that the remainder of the problem is as suitable as possible for further optimization. For this search, we propose a probabilistic methodology, where parts of the solution are considered and the decision as to which part to select is made in a probabilistic manner, so that the likelihood of obtaining a high-quality solution is maximized. The quality of the solution is evaluated using an objective function. After the small part is solved, we eliminate it from further consideration and solve the remaining problem iteratively using the same approach.

We conclude this section by presenting informal specifications for the two demonstration optimization and synthesis problems. The maximum independent set (MIS) problem is an optimization problem defined on an undirected graph. The goal of the MIS problem is to select the largest number of vertices in the graph in such a way that there is no edge between any pair of the selected vertices. Sequence covering is defined on a sequence of symbols and a set of templates created using the same set of symbols. The templates are short sequences of symbols which are to be used to cover the long sequence. The problem is to cover as much of the long sequence as possible, using as many times as necessary the templates provided, without overlapping any of the templates. Further illustration of the application of the PC approach to two other optimization problems (the graph coloring and scheduling problems) can be found in [8].

II. Related Work

The related work in terms of its scope can be classified in two broad groups. The first one consists of generic algorithmic techniques, specifically, deterministic constructive algorithms, deterministic iterative improvement algorithms, and probabilistic iterative improvement algorithms. We restrict our scope to discrete optimization problems. The second part is related to state-of-the-art techniques for solving the MIS and sequence covering.

By far the most popular and widely used generic algorithmic paradigm is the deterministic constructive approach. Algorithms of this type have been applied on a vast variety of problems, starting from sorting and basic graph algorithms such as breadth first search and topological sort, to more complex graph algorithms, such as all-pairs shortest path and maximum flow [9]. This paradigm has also been applied to string matching, computational geometry problems and a number of theoretic algorithms in many other fields. Several generic algorithmic techniques of the constructive deterministic approach have found many applications. For example, greedy algorithms, dynamic programming, and branch-and-bound are used to solve many problems. There are a number of excellent textbooks on this topic including [7], [9], and [10].

In 1970, Kernighan and Lin introduced the first iterative improvement heuristic, which they applied to graph partitioning [4]. The algorithm uses pair swap moves to iteratively reassign elements to different partitions. It proceeds in a series of passes, during which each component is moved exactly once. A number of improvements on the basic strategies have been proposed over the years [11].

Randomized versions of the deterministic constructive algorithms have been popular for a long time [7]. Randomization often dramatically improves the average runtime of algorithms. A typical example is Quicksort and its randomized version [12].

Since 1953, a number of probabilistic iterative improvement algorithms have been proposed. Two of them have origins in statistical mechanics: the metropolis algorithm [13] and simulated annealing [5], [6]. Simulated annealing found a spectrum of applications in engineering, computer science, and image recognition [14]. In contrast to deterministic iterative improvement algorithms, simulated annealing allows hill-climbing moves. Moves are not accepted blindly, like in random-search algorithms, but according to criteria that takes the objective function and runtime into consideration. Consequently, a number of probabilistic iterative improvement algorithms that often explore with analogy to both the physical and biological world, have been proposed including genetic algorithms [15], neural networks, simulated evolution [16], and tabu search [17].

The new PC paradigm is different from all the above paradigms. In some sense, it is closest to randomized algorithms. Conceptually, the difference is that PC uses extensive probabilistic search to find an attractive way to solve an arbitrary small part of the problem and construct (as opposed to improve) the solution.

MIS is one of the most popular generic NP-complete problems [18], [19]. For example, it was one of the first problems proved to be NP-complete [19]. Most commonly, MISs are used as a set during graph coloring. As a matter of fact, it has been experimentally demonstrated that in many domains, finding an MIS is sufficient to make coloring popular benchmarks both fast and provably optimal [20]. Nevertheless, there are a number of intriguing MIS problem applications such as the hereditary subset problem, determining DNA sequence similarity [21], and efficient use of amorphous computers and wireless ad hoc networks. Recently, also several cryptographic and intellectual property protection techniques have been proposed which exploit the difficulty of finding the largest, intentionally placed MIS (or clique) in a random graph to build security mechanisms [22].

A number of optimization algorithms for MIS problems have also been proposed. The emphasis was mainly on parallel [23] and randomized algorithms. In addition, several algorithms for MIS discovery in special types of graphs have been proposed, in particular, ones where an optimal polynomial time solution can be found [19]. Furthermore, a great variety of constructive heuristic and iterative improvement approaches has been reported.

Finally, note that the closely related maximum clique problem also has a wide range of applications [24] and that numerous algorithms have been developed to locate the largest clique in the graph [25]. The MIS problem is equivalent to the maximum clique problem in the complemented graph [19]. An excellent survey on the maximum clique problem is [24]. It presents several exact and heuristic approaches (including sequential, greedy, simulated annealing, neural networks, genetic algorithms, tabu search, and continuous domain-based heuristics) and a number of applications such as coding theory, geometry of tiling, fault diagnosis, and vision and pattern recognition.

Sequence covering is a special case of template pattern matching. A work by Hoffman and McDonald provided major impetus research on this topic [26]. The most widely used template-matching technique in compilers is utilizing the dynamic programming approach [27]. In CAD, the most popular approach is also a dynamic programming-based implementation of the Dagon template matching at the logic synthesis level system by Keutzer [3]. In high-level synthesis, several approaches have been proposed, including [28] and [29]. Sequential code covering is also the main task in some approaches for early power estimation [30].

III. PC Optimization Approach

In this section, we describe a new generic method for solving intractable optimization problems using the PC approach. The main idea is to search, probabilistically, for a small part of the solution which can be solved well and which leaves the remaining problem amenable for further optimization. Essentially, during this step, we probabilistically search for the part of the problem that is under relatively strict constraints and try to solve this part in such a way that the reminder of the problem has the least amount of additional constraints imposed. The basic premise of the new paradigm is that a probabilistic search enables fast scanning of parts of the design space while it preserves the speed of deterministic algorithms.

A. Generic PC Optimization

The generic approach has the following nine components. Note that some of the components are specific for a particular problem, while others are invariant over different problems.

Candidate Part (CP). The candidate part is a relatively small portion of the problem that can be efficiently solved in a particular way. In the general case, we must make two choices regarding the CP: 1) which atomic components of the problem to consider and 2) how to solve that part of the problem. It is important that the CP is not too small, in order to avoid overly local and greedy solutions. It is also important that the CP is not too large, in order to avoid long search times.

Probabilistic Search (PS). One of the more important aspects of the algorithm is how to efficiently search the solution space using probabilistic constructs. There are two main alternatives: 1) iterative improvement and 2) constructive techniques. The first defines a move that probabilistically replaces a single component from the CP with a new component. The second method is to generate a new CP from scratch each time. From the implementation point of view, random number generation is a computationally intensive task in the PC algorithm. In our implementation, we use a stored list of randomly generated numbers that is traversed starting from randomly selected points. While this approach generates numbers that are not completely compliant with the standard tests for randomness [7], [9], the extensive experimentation implies that it can speed up the performance of the algorithm by an order of magnitude without sacrificing the quality of the solution.

Candidate List (CL). The candidate list contains the k best solutions for the CPs found using probabilistic search. The most important criteria related to the CLs are the ones that select which solutions should be included in the list. The simplest approach is to include only the k best solutions [with the k best objective function (OF) values]. A more sophisticated approach takes into account the overlap between the new proposed CPs and CPs in the candidate list. Another, interesting alternative is to mainly target the parts of problem that are most constrained. The intuition is that it is often better to solve the difficult parts of the problem early, in order to address this part of the problem while we still have significant freedom in how we can address the constraints of the problem.

Objective Function (OF). The objective function is a heuristic measure of likelihood that a particular solution to a particular part of the final solution is a promising choice. This idea is similar to the scoring strategies used in game playing [31], [32]. The main tradeoff is between the accuracy (ability to estimate) and the runtime. This tradeoff can be systematically exploited by considering the increasing levels of neighbors of the elements of the CP and by increasing the computational effort to form a more accurate picture.

Comprehensive Objective Function (COF). The OF is calculated for all proposed solutions and therefore it is important that it be fast. Once the number of candidates is reduced to only a few, it is essential to evaluate them as accurately as possible. Therefore, before the final selection of a particular candidate from the CL, we calculate the COF. Conceptually, the main difference between the OF and COF is that the former involves calculations of properties related only to properties of a small part of the solution, while the latter takes into account properties of the still remaining unsolved regions. Another important criterion that needs to be taken into consideration is the overlap between the selected CP and other CPs in the CL. Clearly, less overlap implies that more of the current candidates can be reused in the next stages of the algorithm. The same tradeoff, accuracy and runtime, that was stated for the OF also applies to the COF.

Stopping Criteria. The effectiveness of probabilistic search for a promising CP is positively correlated with the search time. Although, to some extent, only experimentation of a particular problem and particular instance of the problem can accurately indicate this. Nevertheless, two general guidance criteria can be stated: 1) longer search time is required in the beginning when the problem is still large and 2) the best indication of finding a new quality solution for a CP is that, for a long period of time, no new superior CP is observed.

Best Candidate Selection (BCS). The best candidate selection is the process of selecting the part of the CP which will become part of the final solution. The simplest strategy is to select the CP with the best COF. One can envision approaches where information from the previous runs of the algorithm or delayed binding are used.

```
Procedure GPC(P) {
While (Overall Control Strategy is not satisfied) {
    S=0;
        While (S(P) is not complete) {
            While (stopping criteria is not satisfied) {
                $CP_i$ = GenerateCPusingPS();
                $OF_i$ = CalculateOF($CP_i$);
                If($OF_i > OF_{min}$)
                    UpdateCL($CP_i$);
            }
            For(all $CP_j$ in CL)
                CalculateCOF($CP_j$);
            BCS = BestCandidateSelection(CL);
            S(P) = SolutionIntegration(S(P),BCS);
}}}
```

Fig. 2. Generic PC Algorithm (GPC).

Solution Integration. Divide and conquer is a popular algorithmic paradigm. Its application is often restricted due to the difficulty of integrating components. Therefore, one of the most important aspects of the PC approach is to develop mechanisms for integrating solutions to the CPs into the solution of the overall problem. This solution is the most demanding aspect of the PC approach, which requires the highest degree of creativity. Nevertheless, there exists a generic technique for this task. The technique is based on constraint manipulation, where the already solved parts, are presented as constraints to the remaining problem.

Overall Control Strategy. Since the new approach is probabilistic, each run of the algorithm, in principle, produces different solutions and has different runtimes. One can superimpose a variety of control strategies using the generic algorithm as the building block. For example, one can use multistarts or keep statistics about the difficulty of resolving some parts of the solution and use this as the decision criteria of when to terminate an unpromising start.

The new problem-solving paradigm can be explained in the following way. We attempt to find a small and readily solvable part of an overall problem and find a high quality solution to that part. The objective function is used to evaluate the quality of the proposed solution. Examining all parts of the problem is a procedure with exponential time complexity and therefore is not a plausible approach. This suggests the use of a randomized search algorithm. The search should avoid visiting the same parts of the problem more than once. The parts with a high solution quality are stored for future considerations. In particular, diverse solutions are very beneficial because they can be used consequently to form other parts of the solution. Furthermore, if possible, the CP should be flexible in order to allow the imposing of additional control or search strategies later on. The pseudocode of a generic approach for the PC procedure (GPC) is listed in Fig. 2.

First, the algorithm finds a CL of promising solvable CPs (CP_i). Each CP is found after applying the probabilistic search to the current instance of the problem P. During this probabilistic selection, the algorithm favors CPs that are more likely to be solved efficiently (have higher OF values), and adds only the best CPs to the CL. Next, the COF is calculated for each of the CPs in the CL. The BCS is selected from the CL according to the rule for BCS. This selected BCS, or CP_i, which evaluated best according to the best candidate selection rules, is then integrated as part of the solution and eliminated from the problem. The procedure then repeats on the remainder of the problem until a complete solution is found.

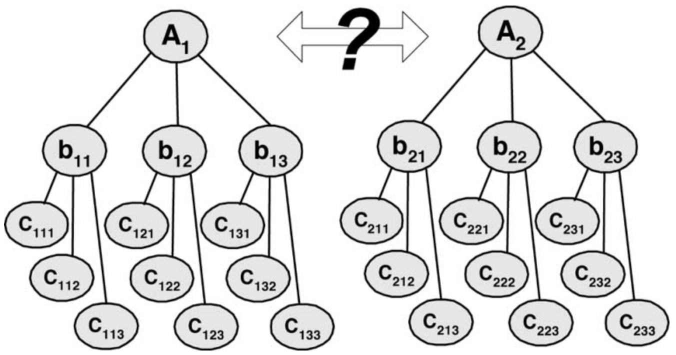


Fig. 3. Delayed binding example.

B. Delayed Binding

One of the main advantages of the PC approach is its flexibility. It is easy to superimpose a number of additional optimization mechanisms, such as multiple OFs, on the generic technique in order to explore the tradeoff between quality of the solution and runtime of the program. In this section, we explain how delayed binding can be used to enhance the performance of the generic algorithm.

The basic idea behind delayed binding is the postponement of the CP selection of a particular partial solution (BCS decision) until later search iterations. This mechanism is illustrated in Fig. 3. We assume that originally we make a pending commitment to two solutions A_1 and A_2. For each of them, we continue the search to find several consequent CPs. Specifically, for A_1, we find parts b_{11}, b_{12}, and b_{13}, and for A_2 we find parts b_{21}, b_{22}, and b_{23}. Repeating this again results in 18 different CPs as shown in Fig. 3.

Now, we evaluate each CP corresponding to each path in the trees. Specifically, we make the final decision for the A_1 or A_2 selection based on the OF values the best c_{ijk}. There are a number of strategies that can be adopted for this task; for example, one can adopt A_i, which has the best COF in its children. If C_{122} is the best, we would accept A_1 and then restart the same procedure by eliminating all branches that are not selected. Note that selecting the best potential child is not necessarily an optimal strategy for finding the optimal solution. One potential alternative is to consider the weighted sum of the best children. This mechanism is affected by several parameters such as the depth of the search to which a decision is delayed, and/or the number of branches at each level. The pseudocode for the delayed binding mechanism is shown in Fig. 4.

Each time we apply the GPC approach as described in the previous section, we obtain a different solution $(A_1, A_2, \ldots, A_{K_1})$. In order to make a decision to accept a particular A_i and proceed, we generate the first-level $(b_{i1}, b_{i2}, \ldots, b_{iK_2})$ and the second-level $(c_{ij1}, c_{ij2}, \ldots, c_{ijK_3})$ CPs, as they would occur as a consequence of selecting each solution A_i. We then assign the solution A_{best}, as the solution A_i that results in the CP with the best OF among all leaf CPs in the expansion tree. The

```
Procedure Delayed Binding()
While (Problem is not solved) {
    $A_1$=Select $K_1$ BCSs;
    For(i=1 to $K_1$)
        GPC(Problem with $A_i \in A_1$ removed);
    $A_{1i}$=Select K2 BCSs;
    For (j=1 to $K_2$)
        GPC(Problem with $A_i \in A_{1i}$ removed)
    $A_{1ij}$=Select $K_2$ BCSs;
    CalculateOF ($A_{1ij}$);
    Save $i$ value of best $A_{1ij}$;
    Select $A_{best}$ from $A_{1i}$ according to best $i$;
    S = SolutionIntegration(S, $A_{best}$);
    Remove $A_{best}$ from the Problem;
}
```

Fig. 4. Algorithm for delayed binding.

elimination and solution integration operations on this A_{best}, is the same as GPC.

IV. Application to MIS

In this section, we introduce the MIS problem and explain how the new PC paradigm can be applied to it. We conclude this section by illustrating the approach on a small example. Using the standard Garey–Johnson format [19], the MIS problem can be defined formally in the following way:

Problem: Maximum Independent Set (MIS)
Instance: Graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, positive integer $\mathrm{K} \leq |V|$.
Question: Does G contain an independent set V′, i.e., a subset $\mathrm{V}' \subseteq \mathrm{V}$ such that for all pairs of vertices $\mathrm{u}, \mathrm{v} \in \mathrm{V}'$ and the edge {u,v} not in E, with $|V'| \geq K$?

The PC algorithm can be applied to the MIS problem in at least two conceptually different ways. The first is to select nodes to include in the MIS. The other is to select nodes which are to be excluded from the MIS. The solution is then the set of nodes which remain unconnected in the final graph. For the first approach, where we select nodes to be included in the MIS, we define the PC components in the following way.

Candidate Part (CP). We select any subset of nodes which are not connected by any edges to be considered as the CP. Each CP is a possible subset of the nodes in the final solution, or MIS. The candidate part can be of size k, where k is a variable or constant value. In our experimental evaluations, we used $k = 4$ nodes. There are several simple and good heuristics for selecting k. For example, k can be a fraction of the number of nodes in the graph. Another intuitive heuristic is to select k as a linear or polynomial function of the number of edges in the graph.

Probabilistic Search. We search the solution space by excluding a single node from a CP of size k, and including another node that previously was not a member of the CP. The nodes to exclude $N_e(j)$ and include $N_i(j)$ in the CP are chosen according to the following equations calculated for each node j.

$$\begin{aligned} N_e(j) &= w_1 n(j) + w_2 n_u(j) + w_3 n_1(j) \\ \text{where} \quad n_1(j) &= \sum_{j=0}^{\#\_neib} n(j) \\ N_i(j) &= \frac{1}{N_e(j)}. \end{aligned}$$

We define $n(j)$ as the number of neighbors of node j, and $n_u(j)$ as the number of unique neighbors of node j, i.e., neighbors that no other node in the CP have edges to. The variable $n_1(j)$ is the total number of neighbors for all the neighbors of the current node j. In the remainder of the paper, we will denote the number of neighbors for a node as $\#\_neib$. Essentially, the intuition is to exclude nodes which have many neighbors, and in particular many unique neighbors, and to retain nodes whose neighbors have many neighbors. We used the following values: $w_1 = 1$, $w_2 = 5$, $w_3 = -0.01$. The reason for inclusion is exactly the opposite. We select probabilistically which node to exclude or include according to the value N_e or N_i for node j. Probability is assigned linearly proportional to a node's N_e and N_i values.

Candidate List (CL). We include k_1 CPs in the CL with the constraint that no node exists in more than one-fifth of the CPs in the CL. The intuition is that we do not want many CPs in the CL which cover the same node, because only one of them can be used. We also note that if the OFs of the CPs are relatively consistent in value, then we continue to add CPs to the CL to make it twice as long as usual. On the other hand, the values of the OF are distributed over long range, then we cease building the list, assuming that we have satisfied the minimum list size, $k_{\min}$. Our intuition is that if the values of the OF are relatively consistent, then most likely we should continue to search further to find a good overall selection.

Objective Function (OF). The objective function is the weighted sum of n_r, the number of nodes in the remainder of the graph that are still eligible to be included in the MIS, and e is the total number of edges in the current graph minus the number of incident edges. We give preference to the CPs that leave a large number of nodes eligible for selection in the next iteration. We also give preference to the CPs which eliminate many edges for the graph. The less edges in the graph the more likely we are to be able to select more nodes to eventually include in the MIS. Note that α_2 is negative in the following formula for the OF. Both α_1 and α_2 are set to 1:

$$\mathrm{OF}(CP_i) = \alpha_1 n_r + \alpha_2 e.$$

COF. For the COF, we combine the OF with an additional component. This component penalizes a CP for having a large number of neighbors. We denote the number of neighbors of node i in the CP by n_i. We denote the size of the CP by k. Therefore, the COF has the following form:

$$\mathrm{COF}(CP_i) = \mathrm{OF}(MIS_i) + \alpha_3 \sum_{i=1}^{k} n_i^2.$$

We penalize CPs with nodes that have uniform numbers of edges because they limit the number of possibly easy to include nodes for the next iterations. Note that in this case, α_3 is negative.

Stopping Criteria. We stop searching for new CPs for the CL after k_{nr} attempts to find a CP with an improved OF, where k_{nr} is the weighted number of remaining nodes in the graph. The idea is that if the recent searching efforts do not bring in any improvement, then most likely, it will not be found without significant additional search. We found that $k_{nr} = 5$ times the

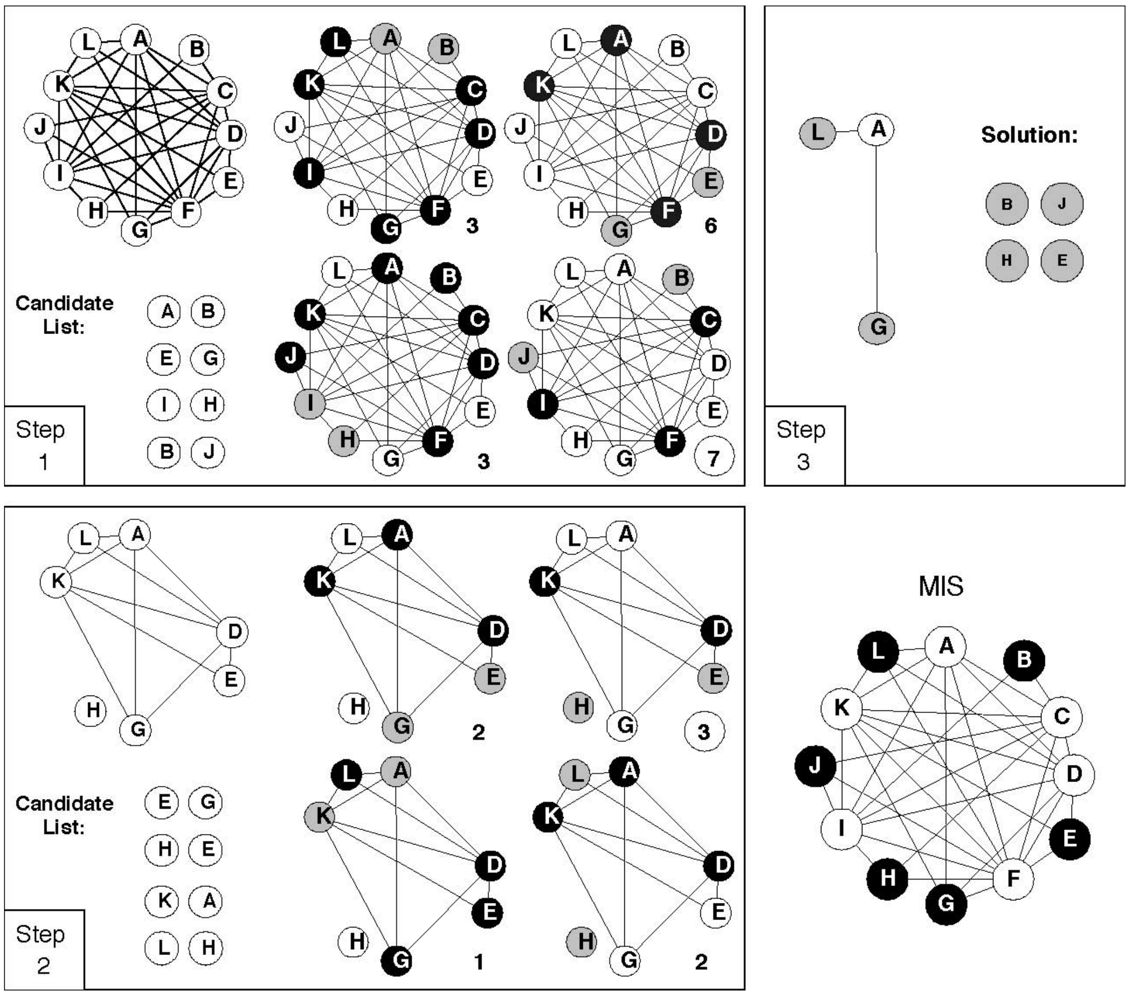


Fig. 5. Example of PC approach applied to the MIS problem.

number of nodes remaining in the graph performs well in practice.

Best Candidate Selection (BCS). We select the best CP by enhancing the COF with additional criteria—the number of occurrences of the CP in nodes in the CL. If the nodes in the BCS only appear in one CP in the CL, then by selecting the CP we preserve a large number of already found CPs and leave a large part of the solution space with high potential untouched. We denote the total number of appearances for node i in the CL as a_i

$$\mathrm{BCS}(CP_i) = w_1 \mathrm{COF}(CP_i) + \sum_{i=0}^{|CP|} \frac{1}{a_i}$$

where w_1 is the weight factor set to value 3.

Solution Integration. We integrate the BCS into the solution and leave the remaining problem to be solved by removing all nodes in the selected CP, as well as neighbors of the nodes and all incident edges to these nodes.

Overall Control Strategy. For the overall control strategy, we conduct $n/10$ multistarts given that n is the number of nodes in the original instance. This number was determined experimentally.

The second approach, where we select nodes to exclude from the MIS, uses many of the same component definitions as the first approach. The definitions of the candidate part, candidate list, COF, stopping criteria, best candidate selection, solution integration, and overall control strategy all remain the same. We define the remaining components in the following way.

Probabilistic Search. We again select any one of the four nodes to be excluded from the CP and replace it with another node. The node to be included and the node to be excluded are selected probabilistically using the following values:

$$N_e = \frac{1}{N_i}, \quad N_i = \sum_{j=1}^{\#\_\mathrm{neib}} \sum_{k=1}^{\#\_\mathrm{neib}(n_{ij})} \frac{1}{n_{ijk}}.$$

We define the neighbor of node n_i as n_{ij}, and the neighbor of n_{ij} as n_{ijk}. As previously denoted, #_neigh is the total number of neighbors of a node. We eliminate the nodes with many neighbors of neighbors, because these neighbors will greatly harm a potential solution by eliminating a significant number of nodes from consideration.

Objective Function (OF). In this case, we use the objective function that includes only the number of edges which remain in the resulting graph, e. We experimentally set α to 1

$$\mathrm{OF}(CP_i) = \alpha\, \mathrm{e}.$$

In order to better explain how the PC approach is applied to the MIS problem, consider the instance of the problem shown in Fig. 5. We have a graph G, with 12 vertices and 34 edges, shown in the top left of the figure. For the sack of simplicity and clarity, we select pairs of vertices for our candidate parts. For the sake of brevity, instead of doing an iterative probabilistic search, we use the constructive approach to create CPs.

For each pair of vertices (CP) which we consider to include in the CL, we evaluate the OF. The OF is equal to the number of nodes in the remainder of the graph after the selection of the CP as part of the MIS. For example, if we consider the pair, K and C, we see that the OF will evaluate to zero. By selecting these two vertices, we eliminate all the remaining vertices in the graph. In Fig. 5, we build the CL to include four different CPs with their OFs. Next, for each of the CPs in the CL, we evaluate their COFs. Again, for the sake of simplicity and clarity, we assume that the COF is equal to the number of vertices remaining in the graph. Furthermore, we define the BCS as the CP with the largest number of remaining vertices in the graph. The first iteration of the PC algorithm is shown in Step 1. In this case, we see that the last CP in the list, pair of vertices B and J, has the highest COF. Therefore, these vertices are selected as part of the solution. To conduct solution integration, we remove the selected vertices from the graph along with all their neighboring vertices and all incident edges to these vertices. The resulting smaller instance is shown in Step 2 of Fig. 5.

In the next iteration of the algorithm, we conduct exactly the same procedure. We first eliminate all CPs from the CL which are no longer valid, and replace them with new pairs of vertices. In this case, the only CP which is still valid consists of vertices E and G. Next, we re-evaluate the COF for each CP in the CL. As a result, we find that the CP consisting of vertices H and E has the highest COF and therefore is selected as the BCS. After solution integration, we have the resulting graph with three vertices as shown in Step 3.

Now the problem is reduced to the extent that the only feasible CP consists of vertices L and G. Also in this moment the CP algorithms is terminated. We combine all the selected CPs to build our final solution shown in the bottom right of the figure. The resulting MIS contains six nodes: B, E, G, H, J, and L. Exhaustive search indicates that this is the optimal solution.

V. Application to the Sequential Code-Covering Problem

We begin this section with the definition of sequential code covering and the proceed to illustrate how to define the PC approach on the problem. We conclude this section with an illustrative example.

In sequential code covering, the goal is to solve the following problem. Given a program at the assembly level and a set of functions (small programs) that are well characterized in terms of their power consumption, find an accurate estimation of the power consumption of the program by covering the program using the functions. In order to make the treatment of the problem more formal and, hence, provide a sound analysis, we abstract the sequential code covering problem into the sequence-covering problem.

Problem: Sequence Covering
Instance: Finite set of symbols $\mathrm{D} = \{d_1, d_2, \ldots, d_n\}$, set of templates $\mathrm{T} = \{t_1, t_2, \ldots, t_k\}$ such that each template t_i is formed by concatenating an arbitrary number of symbols from set D, sequence S formed using concatenation of symbols from set D and integer U.
Question: Can S be covered using multiple instances of templates T such that not two templates overlap and the number of uncovered symbols in S is less than U?

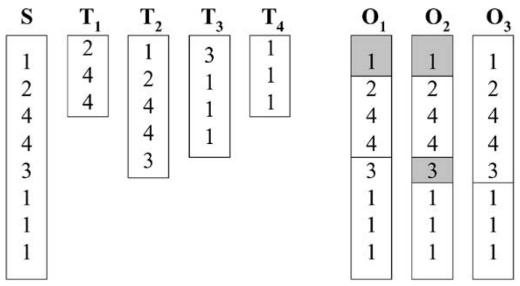


Fig. 6. Sequence-covering example.

We now summarize the relationship between the power-estimation problem and the sequence-covering problem. The sequence-covering problem can be mapped to the high-level power estimation problem and, therefore, optimization problems in programmable processors in the following way. The set T is the set of basic program instructions at the assembly level. The sequence S is a program written by using those instructions. Each of the k templates is well characterized in terms of its power consumption [30]. It has been experimentally verified that this procedure yields extremely accurate power prediction within 3% for the Toshiba R3900 microprocessor [30]. The goal is to define a way to cover the instruction sequence S with templates from the instruction set T, in such a way that the amount of uncovered instructions is minimized. No two templates can overlap when they cover the instruction sequence.

Another, probably even more widely applicable abstraction that leads to the sequence-covering problem is microcode compression in processors with complex instructions. The goal is to make the code as compact as possible and therefore as fast as possible by effectively using complex instructions [33]. Also note that the sequence covering problem is a special case of the technology mapping problem in logic synthesis [3] and the template matching problem in behavioral synthesis [34].

We further clarify the sequence-covering problem using the example shown in Fig. 6. Our set of symbols is $\mathrm{D} = \{1, 2, 3, 4\}$. We have four templates T_1–T_4. The example is simple; therefore, all the covering options can be easily recognized. If we set $U = 3$, which implies that we always have less than three instructions uncovered, O_1, O_2, and O_3 are the covering options. O_1 leaves one uncovered instruction, O_2 leaves two uncovered instructions, and O_3 covers all the instructions.

When the sequence-covering problem is addressed using the PC approach, during each pass through the sequence, the selection of new starting points is important because it impacts the decision of which templates will be used for immediate covering from that point on. In the example, if we start the covering procedure from the first character of the sequence, we will select template T_2. However, if we start from the second item in the sequence, we can never go back and use sequence T_2. For this reason, we defined our approach for sequence covering in such a way that the starting point is randomly selected and a numerous number of attempts ensure high probability to select a suitable starting point.

We define the components for the PC approach for sequence covering as follows:

Candidate Part (CP). We select any continuous subsequence of the sequence S which can be covered by any number of templates such that there are no more than $U_{\max}$ uncovered elements in the sequence. We found the value of $U_{\max} = 5$ performs well in practice.

Probabilistic Search. We search the solution space by analyzing subsets of the sequence S to be covered. For each subsequence, we calculate a PS value, which indicates the likelihood of adding the subset to the CL. We define O_i to represent the total number of occurrences of an element i in all CPs in the CL. If we define K_L as the length of the CP, we can define PS for each element j as follows:

$$PS(j) = \frac{1}{\sum_i^{K_L} O_i}.$$

The intuition behind the definition is as follows. If element j is difficult to be covered, any sequence that covers that element should receive proportionally higher preference.

Candidate List (CL). We continue to add CPs to the CL as long as each element of sequence S appears less than k times.

Objective Function (OF). The objective function takes into account the weighted sum of two components. The first components p_1 is the length of the subset that is covered by the CP and the second p_2 is the likelihood that the elements in the subset are covered by other CPs. A CP is more beneficial if it covers symbols in the sequence which are hard to cover and also covers a large number of symbols. Therefore, the objective function has the following form:

$$\mathrm{OF}(CP_i) = \alpha_1 p_1 + \alpha_2 p_2.$$

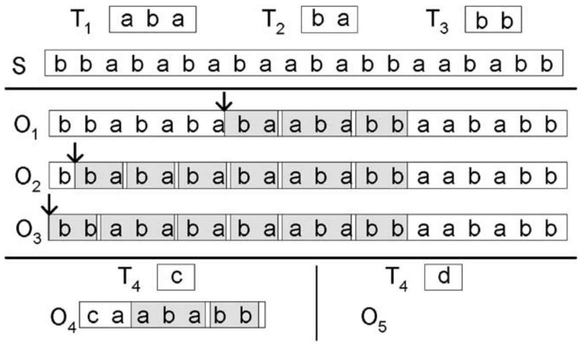
COF. In addition to the objective function, the COF has one more weighted component. For each of the CPs, we calculate the overlap between the CPs in the CL. We define the overlap between two CPs A and B, O_{AB}, as the number of identical symbols covered by both CPs. For each CP, we calculate the COF as a weighted sum of this overlap and the OF. We penalize the CP if it overlaps significantly with other CPs in the CL. The more overlap, the more we are constraining the remaining problem. Therefore, we have

$$\mathrm{COF}(CP_j) = \frac{w_3 O_{ij} OF(CP_j)}{|CP_j|}, \quad \text{where } w_3 < 0.$$

Stopping Criteria. We stop searching once each element has been covered by CPs in the CL k_{sc} times. By doing this, we give each element in the sequence a good chance of being covered. Experimentally, we found $k_{sc} = 16$ to work well in practice.

Best Candidate Selection (BCS). We select the CP with the best COF value. Note that the COF has a component that ensures that the overlap between high-quality CPs is taken into account.

Solution Integration. We replace the BCS with a new symbol d_i', which consists of a unique new symbol and we also add a new template to the library t_i'. This symbol can never be covered in the sequence by any other template other than t_i'. We assume that this template has length 0.



T1 a b a
T2 b a
T3 b b
S b b a b a b a b a a b a b b a a b a b b
O1 b b a b a b a b a a b a b b a a b a b b
O2 b b a b a b a b a a b a b b a a b a b b
O3 b b a b a b a b a a b a b b a a b a b b
T4 c
O4 c a a b a b b
T4 d
O5

Fig. 7. Sequence-covering example.

Overall Control Strategy. If after k_{ocs} multistarts there is no improvement in the number of total uncovered elements then we terminate search. In experimentation, we found $k_{\text{ocs}} = 5$ to provide the best results.

To clarify the key steps of the PC approach when applied to the sequence-covering problem, consider an instance of the problem shown in Fig. 7. The example contains two symbols, "a" and "b." There are three templates, one of length three (T_1—"aba,"), and two of length two (T_2—"ba," T_3—"bb"). The sequence, S, is composed of 20 symbols.

The goal is to cover the sequence using the templates T_1, T_2 and T_3 such that the largest number of symbols in the sequence is covered. In our definition of the CP for sequence covering we select subsequences of the sequence which can be covered with at most $U_{\max}$ elements uncovered. In this case, we set $U_{\max}$ to zero. We randomly select a starting point for building the CPs. In this case, we select positions 6, 1, and 0 in the sequence. We add each of the CPs to the CL and evaluate each of them using the COF. In this small example, for the sake of brevity, we simplify the COF to only consider the length of the sequence covered. The three resulting coverage are O_1, O_2, and O_3 as shown in Fig. 7. In this case, we select as our BCS, the sequence that covers the longest continuous subsequence. O_3 uses 6 templates to cover 14 symbols, O_2 uses 6 templates to cover 13 symbols, and O_1 uses 3 templates to cover 7 symbols. Therefore, we select CP O_3. Once we have selected our BCS, we do solution integration by creating a new symbol c and a new template T_4—"c." Then we replace the entire covered subsequence in O_3 with the symbol c. The resulting simplified sequence is O_4.

Next, we repeat the whole procedure in exactly the same way. First we select several new random starting points, and rebuild the CL. For the sake of brevity we omit the elaboration of the steps of the algorithms which are identical to the ones explained in the previous paragraph. The solution is shown in O_5 of Fig. 7. In this case, the best coverage is 19 out of the 20 symbols.

VI. Experimental Results

In this section, we present the experimental results conducted on real-life examples, as well as specially prepared examples for which an optimal solution is known. We applied the PC techniques to each of the four selected problems and compared the quality of our solutions to previously published results [20],

TABLE I
EXPERIMENTAL RESULTS FOR MAXIMUM INDEPENDENT SETS

Name	V	E in Clique	E in MIS	γ	CPU [20]	CPU
school1_nsh	358	16710	47193	14	0.92	0.22
keller4	171	9435	5100	11	4.87	0.9
sanr200_0.7	200	13868	6032	18	23.0	3.71
brock200_1	200	14834	5066	21	112.9	22.58
san200_0.7_2	200	13930	5970	18	1.66	0.31
P_hat300-2	300	21928	22922	25	4.21	0.94
hamming8-4	256	20864	11776	16	0.18	0.006
san200_0.9_1	200	17910	1990	70	5.61	1.02
MANN_a27	378	70551	702	126	98.4	12.3

[30], [35]. All experimentation of the PC approach was done on a 300-MHz Sun Ultra-10 Workstation (SpecInt 12.1). In the cases where we compare the new approach with previously published results for which we had the software available, we executed both programs on this machine. When we were not able to obtain the software from the best previously published result, we scaled our obtained runtimes to the runtimes on the originally used machine. For conversion, we exploit the ratio provided by SpecInt benchmarks on the Sun Ultra-10 workstation and the machine which the original results were obtained.

The PC approach and other heuristic techniques with a larger number of tunable parameters are intrinsically difficult for experimental evaluation. The source of difficulty is a well-known "curse of dimensionality:" there are an exponentially large number of potential combination of parameters. To address this problem, we used two directions: variety of example and the perturbation approach. The idea of the perturbation-based validation is to randomly perturbate each of the used parameters by a certain percentage. If the quality of the obtained solution is not significantly altered with a sizable change, it is a strong indicator that the obtained results are indeed due to the effectiveness of the employed optimization mechanisms and not consequence of parameter overtuning. In our experimentations, we altered the parameters by $\pm 25\%$ and did not notice a significant changes in the quality of the obtained solutions.

In the case of MIS, we ran testing on instances for the problem of finding the maximum clique. The maximum clique problem can be easily mapped to MIS by complementing the graph. Complemented graph G_c of graph G is a graph that has the same set of vertices as G. However, G_c has edges between two vertices if and only if G does not have edge between these two vertices. The MIS in a graph is the maximum clique in the complemented graph and vice versa. This decision was made due to the fact that we were not able to locate experimental results for MIS solvers, while a number of maximum clique programs are readily available.

The first column of Table I indicates the name of the maximum clique instance (the instances are from [36], [37]), while the next column states the number of vertices in the graph. The next two columns give the number of edges in the original graph and the number of edges in the complemented graph respectively. The fifth column represents the number of nodes in the MIS or maximum clique. The sixth column indicates the runtimes reported by Coudert on a 60 MHz SuperSparc (85.4 SpecInt). Finally, the seventh column displays the runtime for finding the MIS using the PC heuristic. These times are scaled using the SPEC conversion to the original machine, and therefore are good indicators of the real speed-up. The average speed-up is approximately 5.5 times. Both our PC approach and the Coudert approach find the optimal solution on all examples. The optimal solution is known from the implicit enumeration of the branch-and-bound approach.

TABLE II
EXPERIMENTAL RESULTS FOR SEQUENCE COVERING

# Temp.	Temp. Seq.	Seq. Len.	Opt. Sol.	Greedy	GPC	Delayed Decision
20	1000	4720	0.968	0.632	0.923	0.928
20	1000	4741	0.983	0.720	0.947	0.947
20	1000	4306	0.966	0.836	0.871	0.939
20	2000	8017	0.968	0.693	0.964	0.952
20	2000	8947	0.980	0.585	0.936	0.950
20	4000	19904	0.991	0.795	0.922	0.965
20	6000	25557	0.977	0.407	0.921	0.915
40	2000	7498	0.981	0.538	0.936	0.923
40	2000	9595	0.988	0.639	0.928	0.919
40	4000	14686	0.984	0.368	0.934	0.924
Ave	-	-	0.979	0.622	0.928	0.936

For the sequence covering problem, we created instances with a specified number of templates and the maximum number of templates in the sequence S. The sequences are created with the specified number of templates along with random components which are not templates. Therefore, the optimal solution is known *a priori* for all examples. This optimal solution provides an upper bound for the evaluation of the CP approach. The lower bound is provided by a greedy heuristic. The greedy heuristic functions in the following way. We begin by finding the first occurrence which can be covered with the longest template, and we cover the sequence with this template. From the end of this template, we continue through the sequence always trying to cover the sequence with the longest possible template until we detect a mismatch with respect to all the templates. For each subsequence which is covered by a template, we replace it with a character which is not used in any template, and the procedure is iteratively continued until no further matches can be found.

The first column of Table II lists the number of templates used in the instance, while the next column lists the number of templates in the sequence (the maximum of templates which can be matched). The length of the sequences is presented in the next column. The next four columns present the percentage of coverage for the optimal solution, by applying the greedy heuristic, by applying the generic PC (GPC) heuristic, and by applying the delayed binding approach, respectively. The last row represents the average percentage for each of the techniques. The results show that the delayed binding approach slightly outperforms the generic approach in this case. For all examples, the runtime was within a few seconds.

VII. CONCLUSION

We introduced a new PC algorithm paradigm. We searched for a small part of the problem that can be solved efficiently and in such a way that the remaining problem is as amenable as possible for further optimization. The approach proceeds in an iteration loop until the complete solution is constructed. The method combines the relatively short runtime of constructive algorithms and the flexibility of probabilistic algorithms. We discussed the main components of the new approach and how the generic approach can be augmented with additional optimization mechanisms. We applied the algorithm to both a generic NP-complete problem (MIS) and a design problem (sequential code covering).

REFERENCES

[1] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*. New York: McGraw-Hill, 1994.

[2] J. K. P. G. Paulin, "Force-directed scheduling for the behavioral synthesis of asics," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 661–679, June 1989.

[3] K. Keutzer, "Dagon: Technology binding and local optimization," in *Proc. ACM/IEEE Design Automation Conf.*, 1987, pp. 341–347.

[4] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–308, 1970.

[5] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[6] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, "Optimization by simulated annealing: An experimental evaluation, part II, graph coloring and number partitioning," *Oper. Res.*, vol. 39, no. 3, pp. 378–406, 1991.

[7] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 1995.

[8] J. L. Wong, F. Koushanfar, S. Megerian, and M. Potkonjak, "Probabilistic constructive optimization techniques," Univ. of California, Los Angeles, Tech. Rep. TR030 049, 2003.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. New York: McGraw-Hill, 1990.

[10] C. Brassard and P. Bratley, *Algorithmics: Theory and Practice*. Englewood Cliffs, NJ: Prentice Hall, 1988.

[11] F. Gavril, "Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph," *SIAM J. Comput.*, vol. 1, no. 2, pp. 180–187, 1972.

[12] C. A. R. Hoare, "Quicksort," *Comput. J.*, vol. 5, no. 1, pp. 10–15, 1962.

[13] N. Metropolis, A. Rosenbluth, R. Rosenbluth, A. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *J. Chem. Phys.*, vol. 21, pp. 1087–1092, 1953.

[14] E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. New York: Wiley, 1989.

[15] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[16] L. Fogel, A. Owens, and M. Walsh, *Artificial Intelligence Through Simulated Evolution*. New York: Wiley, 1966.

[17] F. Clover, "Tabu search part I," *J. Comput.*, vol. 1, no. 3, pp. 190–206, 1989.

[18] R. Tarjan and A. Trojanowski, "Finding a maximum independent set," *SIAM J. Comput.*, vol. 6, no. 3, pp. 537–546, 1977.

[19] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: W. H. Freeman, 1979.

[20] O. Coudert, "Exact coloring of real-life graphs is easy," in *Proc. ACM/IEEE Design Automation Conf.*, 1997, pp. 121–126.

[21] D. Joseph, J. Meidanis, and P. Tiwari, "Determining DNA sequence similarity using maximum independent set algorithms for interval graphs," in *Proc. Algorithm Theory—SWAT*, 1992, pp. 326–337.

[22] A. Juels and M. Peinado, "Hiding cliques for cryptographic security," *Designs, Codes Cryptog.*, vol. 20, no. 3, pp. 269–280, 2000.

[23] R. Karp and A. Wigderson, "A fast parallel algorithm for the maximal independent set problem," *J. ACM*, vol. 32, no. 4, pp. 762–773, 1985.

[24] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo, "The maximum clique problem," in *Handbook of Combinatorial Optimization*. Norwell, MA: Kluwer, 1999, vol. 4, pp. 1–74.

[25] L. Babel and G. Tinhofer, "A branch and bound algorithm for the maximum clique problem," *ZOR-Methods Models Oper. Res.*, vol. 34, no. 3, pp. 207–217, 1990.

[26] C. Hoffman and M. O'Donnell, "Pattern matching in trees," *J. ACM*, vol. 29, no. 1, pp. 68–95, 1982.

[27] A. Aho, M. Ganapathi, and S. Tjiang, "Code generation using tree matching and dynamic programming," *ACM Trans. Program. Lang. Syst.*, vol. 11, no. 4, pp. 491–516, 1989.

[28] F. Catthoor, G. Goossens, and H. D. Man, "Combined hardware selection and pipelining in high-performance data-path design," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 413–423, Apr. 1992.

[29] D. Rao and F. Kurdahi, "On clustering for maximal regularity extraction," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 1198–1208, Aug. 1993.

[30] G. Qu, N. Kawabe, K. Usami, and M. Potkonjak, "Function-level power estimation methodology for microprocessors," in *Proc. ACM/IEEE Design Automation Conf.*, 2000, pp. 810–813.

[31] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Reading, MA: Addison-Wesley, 1984.

[32] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice Hall, 1995.

[33] H. Massalin, "Superoptimizer—A look at the smallest program," in *Proc. Architec. Support Program. Lang. Oper. Syst.*, 1987, pp. 122–126.

[34] M. Corazao, M. Khalaf, L. Guerra, M. Potkonjak, and J. Rabaey, "Performance optimization using template mapping for datapath-intensive high-level synthesis," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 877–888, Aug. 1996.

[35] D. Kirovski, Y.-Y. Hwang, M. Potkonjak, and J. Cong, "Intellectual property protection by watermarking combinational logic synthesis solutions," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1998, pp. 194–198.

[36] J. Cong, M. Hossain, and N. Sherwani, "A probably good multilayer topological planar routing algorithm in IC layout designs," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 70–78, Jan. 1993.

[37] DIMACS.. DIMACS. [Online] Available: http://dimacs.rutgers.edu

Jennifer L. Wong (S'99) received the B.S. degree in in computer science and engineering and the M.S. degree in computer science from the University of California, Los Angeles, in 2000 and 2002, respectively, where she is currently pursuing the Ph.D. degree.

Her research interests include intellectual property protection, optimization for embedded systems, and mobility in ad-hoc sensor networks.

Farinaz Koushanfar (S'03) is currently pursuing the Ph.D. degree in electrical engineering and computer science at the University of California, Berkeley.

Her research interests include various aspects of optimization and modeling, including statistically-based optimization methods and techniques, and applications of the optimization to emerging distributed networks and embedded systems.

Ms. Koushanfar is the recipient of a NSF Graduate Research Fellowship in 2000, an ACM Mobicom Best Student Paper Award in 2001, and the Intel Labs Collaborative Research Fellowship in 2003.

Seapahn Megerian received the B.S. degree in computer science and engineering and the M.S. and Ph.D. degrees in computer science from the University of California, Los Angeles, in 1998, 1999, and 2003, respectively.

He is an Assistant Professor in the Electrical and Computer Engineering Department, University of Wisconsin, Madison. His main research areas are distributed embedded systems and wireless sensor networks. In addition, his research includes design automation of high-performance communication architectures, computational security, and intellectual property protection.

Miodrag Potkonjak (S'86–M'92) received the Ph.D. degree in electrical engineering and computer science from University of California, Berkeley, in 1991.

In 1991, he joined C&C Research Laboratories, NEC USA, Princeton, NJ. Since 1995, he has been with Computer Science Department at the University of California, Los Angeles (UCLA). His watermarking-based Intellectual Property Protection research formed a basis for the Virtual Socket Initiative Alliance standard. His research interests include system design, embedded systems, computational security, and intellectual property protection.

Dr. Potkonjak received the NSF CAREER Award, the OKAWA Foundation Award, the UCLA TRW SEAS Excellence in Teaching Award, and a number of best paper awards.