Probabilistic Constructive Optimization Techniques

Jennifer L. Wong, Farinaz Koushanfar, Seapahn Megerian, Miodrag Potkonjak

Abstract— **We have developed a new optimization paradigm for solving computationally intractable combinatorial optimization and synthesis problems. The technique, named Probabilistic Constructive (PC), 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 trade-off between runtime and the quality of solution, suitability for the super imposition of a variety of control strategies, and simplicity of implementation. After presenting the generic technique, we apply it to two generic NP-complete problems (maximum independent set) and two synthesis and compilation problems (sequential code covering). Extensive experimentation indicates that the new approach provides very attractive trade-offs between the quality of the solution and runtime, often outperforming the best previously published approaches.**

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 non-important 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 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 [17].

Optimization algorithms used for synthesis have a great variety of features and, therefore, are difficult to be addressed in a fully systematic way. Nevertheless, it is interesting and enlightening to classify optimization algorithms. Figure 1 shows the classification according to two main criteria: (i) the way in which the solution is built and (ii) 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 [49] or use dynamic programming [36]. In the last three decades, deterministic iterative improvement algorithms [35] were proposed for many problems and were able to produce excellent results. In particular, deterministic iterative improvement algorithms are widely and frequently used for partitioning [4]. Simulated annealing [38] 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 [30].

While numerous algorithms populate three of the quadrants in Figure 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 [47]). 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. On the other hand, the main advantage of probabilistic algorithms is their inherent flexibility that facilitates the trade-off between quality of solution and runtime. They are also suitable for augmentation with a variety of control strategies such as multi-start 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 of 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 subsection be presenting informal specifications for the four demonstration optimization and synthesis problems.

The maximum independent set (MIS) problem and graph coloring problems are optimization problems 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. For the graph coloring problem, the goal is to use

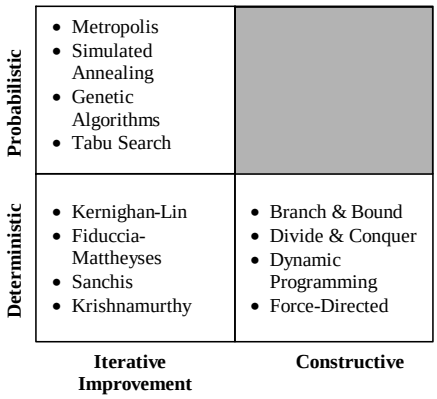

Fig. 1. Classification of Optimization Algorithms.

the minimum number of colors to color all the nodes. The constraint in the problem is that no two vertices with an edge between them can be colored the same color.

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 the templates provided as many times as necessary, without overlapping any of the templates.

Finally, the scheduling problem can be defined in many different ways even if we restrict our attention only on behavioral synthesis. Often it is defined on a directed graph where each vertex represents an operation and edges indicate execution dependencies between the operations. Scheduling is usually formulated to optimize one of two different cases. The objective of the first case is to minimize the amount of functional units (hardware that executes operations) used when scheduling the graph in a given number of clock cycles while keeping all the dependencies satisfied. The second case is the dual problem formulation of the first case. The goal is to minimize the number of clock cycles used to schedule all the operations with dependencies given the amount of operation hardware. We apply the PC approach to the first formulation in Section IV-B.

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 specific generic NP-complete (MIS and Graph Coloring) and CAD problems (Sequence Covering and Scheduling) discussed in this paper.

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 [15]. 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 [15], [9], [47].

In 1970, Kernighan and Lin introduced the first iterative improvement heuristic, which they applied to Graph Partitioning [35]. 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 [23]. The most notable are the ones proposed by Fiduccia and Mattheyses [19], Sanchis [57], and Krishnamurthy, and an excellent survey of this work is given in [4]. The iterative improvement paradigm has been applied to many other optimization problems, including the Travelling Salesman problem [42].

Randomized versions of the deterministic constructive algorithms have been popular for a long time [47]. Randomization often dramatically improves the average runtime of algorithms. A typical example is Quicksort and it's randomized version [27]. There are two types of randomized algorithms, Las Vegas and Monte Carlo. Las Vegas algorithms always generate the correct solution, but their runtime varies depending on the distribution of inputs. In contrast, Mote Carlo algorithms may sometimes produce an incorrect solution, but execute in a predictable amount of time. The probability of a Monte Carlo algorithm producing an incorrect solution can be made arbitrarily small by repetitively running the algorithm, each with independent random choices.

Since 1953, a number of probabilistic iterative improvement algorithms have been proposed. Two of them have origins in statistical mechanics: the Metropolis algorithm [45] and Simulated Annealing [38], [30]. Simulated Annealing found a spectrum of applications in engineering, computer science and image recognition [1]. 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 [29], [26], Neural Networks, Simulated Evolution [20], and Tabu Search [24], [25].

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.

A. Maximum Independent Set

Maximum Independent Set is one of the most popular generic NP-complete problems [59], [22]. For example, it was one of the first problems proved to be NP-complete [22]. Most commonly MIS are used as a set during Graph Coloring. As a matter of fact, it has been experimentally demonstrated that in many domains, finding a MIS is sufficient to make coloring popular benchmarks both fast and provably optimal [16]. Nevertheless, there are a number of intriguing maximum independent set problem applications such as the hereditary subset problem, determining DNA sequence similarity [31], 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 [3], [32].

A number of optimization algorithms for MIS problems have also been proposed. The emphasis was mainly on parallel [33] 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 [22]. 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 [8] and that numerous algorithms have been developed to locate the largest clique in the graph [5]. The MIS problem is equivalent to the maximum clique problem in the complemented graph [22]. An excellent survey on the maximum clique problem is [8]. 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.

B. Sequence Covering

Sequence covering is a special case of template pattern matching. A work by Hoffman and McDonald provided major impetus research on this topic [28]. The most widely used template matching technique in compilers is utilizing the dynamic programming approach [2]. 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 [36]. In high-level synthesis, several approaches have been proposed, including [48], [55]. Sequential code covering is also the main task in some approaches for early power estimation [52].

III. Probabilistic Constructive 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. For example, when we are searching for a Graph Coloring solution, we can color a few nodes in a particular way and remove them from further consideration. 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 the deterministic algorithms.

A. Generic Probabilistic Constructive Optimization Approach

The generic approach has the following nine components. Note that some 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: (i) which atomic components of the problem to consider, and (ii) 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. For example, in Graph Coloring, coloring a single node at a time is a CP decision that is most often too local. However, it is difficult to find a promising coloring solution if we decide to color too many nodes simultaneously.

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: iterative improvement and constructive techniques. The first defines a move that probabilistically replaces a single component from the CP with the 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 [15], [47], the extensive experimentation implies that it can speed up the performance of the algorithm by an order of magnitude without sacrificing the quality of solution. Another important component of the probabilistic search is the way in which probabilities are assigned to each component of the CP so that the component is either included or excluded.

For this task, we use an additionally simplified objective function (OF).

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 k best OFs). A more sophisticated approach takes into account the overlap between the new proposed solution and solutions 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 [50], [56]. The main trade-off is between accuracy (ability to estimate) and the runtime. This trade-off can be systematically exploited by considering increasing levels of neighbors of the elements of the CP and by increasing the computational effort to form a more accurate picture. For example, in Graph Coloring, one can consider all nodes that are neighbors to a node in the CP, and then all the neighbors from that set and so on. At the same time, one could just count the number of edges in the considered set, or their uniformity, or number of pairs that have a larger number of neighbors and so on.

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 trade-off, 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: (i) longer search time is required in the beginning when the problem is still large, (ii) 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.

```
Procedure GPC(P) {
While (Overall Control Strategy is not satisfied) {
   S=;
      While (S(P) is not complete) {
         While (stopping criteria is not satisfied) {
            CP_i = GenerateCP(); //using PS
            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).

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 a multitude of alternatives where information from the previous runs of the algorithm or delayed binding are used.

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. In a sense, this 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. A small example may better explain this paradigm. Consider the graph coloring problem. If we decide to color two nodes n_1 and n_2 with the same color as a CP, all that is needed is to replace n_1 and n_2 with a single node n' in the remainder of the problem. Note that n' should have edges to all the nodes that were connected to n_1 and n_2.

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 multi-starts 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

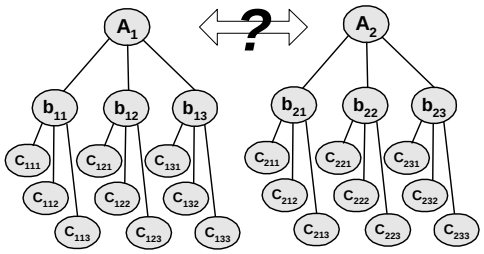

Fig. 3. Delayed Binding Example.

```
Procedure Delayed Binding()
While (Problem is not solved) {
    A1=Select Kd BCSs;
    For(i=1 to K1)
        GPC(Problem with Ai ∈ A1 removed);
    A1i=Select K2 BCSs;
    For (j=1 to K2)
        GPC(Problem with Ai A1i removed)
    A1ij=Select K2 BCSs;
    CalculateOF (A1ij);
    Save I value of best A1ij;
    Select Abest from A1i according to best i;
    S = SolutionIntegration(S, Abest);
    Remove Abest from the Problem;
}
```

Fig. 4. Algorithm for Delayed Binding.

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 pseudo-code of a generic approach for the PC procedure (GPC) is listed in Figure 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 comprehensive objective function, 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.

B. Delayed Binding

One of the main advantages of the PC approach is its flexibility. In particular, it is easy to superimpose a number of additional optimization mechanisms, such as multiple OFs, on the generic technique in order to explore the trade-off between quality of the solution and runtime of the program. In this subsection, 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 ultimate selection of a particular partial solution (BCS decision) until later search iterations. This mechanism is illustrated in Figure 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 Figure 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 re-start 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, the number of branches at each level. The delayed binding mechanism can be summarized using the pseudo-code in Figure 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 elimination and solution integration operations on this A_{best}, is the same as GPC.

C. Other Augmentation Mechanisms

In addition to the delayed binding, we can also have a number of additional mechanisms that can be used to augment the generic strategy such as backtracking and hierarchical application of the generic technique. Backtracking is a technique where we replace one or more of the selected best candidates that are already included in the solution by some other candidates which where previously excluded from the further consideration. The hierarchical PC approach enables that the PC approach itself is used instead of the probabilistic search step. Also, the whole approach can be employed within a large optimization loop and the information from previous runs about difficulty to solve a part of the problem can be considered in later attempts. Finally, and maybe most importantly, let us mention that probabilistic search can be augmented using some of the generic maximally constrained, minimally constraining heuristics to focus efforts on the most difficult and sensitive parts [6], [51], [50].

D. Application to Maximum Independent Set

Using the standard Garey-Johnson format [22], the MIS problem can be defined formally in the following way:

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

The PC algorithm can be applied to the maximum independent set (MIS) problem in at least two conceptually different ways. The first is to select nodes to include in the MIS. The other way 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 node to exclude, N_e, and include, N_i, in the CP are chosen according to the following equations calculated for each node j.

$$N_e(j) = w_1 n(j) + w_2 n_u(j) + w_3 n_1(j)$$
$$\text{where } n_1 = \sum_{i=0}^{\#\_neib} n(i)$$

$$N_i(j) = \frac{1}{N_e(j)}$$

We define $n(j)$ as the number of neighbors of the node, 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. 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: $w_1 = 1$, $w_2 = 5$, $w_3 = -.01$. 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 the nodes N_e and N_i values.

Candidate List (CL). We include k_1 CPs to the CL with the constraint that no node exists in more than 1/5 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.

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

Comprehensive Objective Function (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.

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

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 nr is the 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 $k_{nr} = 5nr$ 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.

$$\text{BCS}(CP_i) = w_1 \text{COF}(CP_i) + \sum_{i=0}^{|CP|} \frac{1}{a_i}, \text{ where } w_1 \text{ is weight factor set to value 3}$$

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 $n/10$ multi-starts 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, comprehensive objective function, stopping criteria, best candidate selection, solution integration and overall control strategy all stay 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 replaced with another node. The nodes to be included and excluded are selected probabilistically using the following values.

$$N_e = \frac{1}{N_i},\ N_i = \sum_{j=1}^{\#\_neib} \sum_{k=1}^{\#\_neib(j)} \frac{1}{n_{ijk}}$$

We define the neighbor of node n_i as n_{ij}, and the neighbor of n_{ij} as n_{ijk}. Therefore, 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.

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

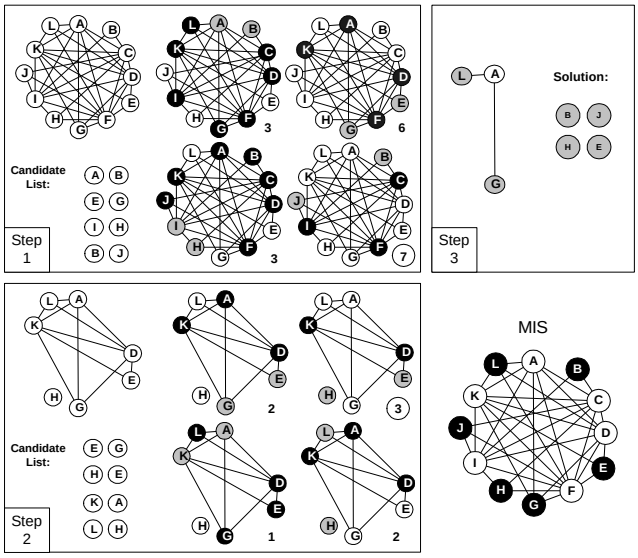

Fig. 5. Example of PC approach applied to the MIS problem.

In order to better explain how the PC approach is applied to the maximum independent set problem consider the instance of the problem shown in Figure 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 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 figure 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 one. 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 two of Figure 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 reevaluate 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 3 vertices as shown in Step three.

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 6 nodes: B, E, G, H, J, and L. Exhaustive search indicates that this the optimal solution.

E. The Graph Coloring Problem

In this section, we explain how the new PC paradigm can be applied to the Graph Coloring problem. A Graph Coloring solution is an assignment of colors to each vertex in a graph such that no two vertices that are connected with an edge have the same color. Using the standard Garey-Johnson format [22], the problem can be defined formally in the following way:

Problem: Graph K-Colorability
Instance: Graph G(V,E), positive integer K $\leq |V|$
Question: Is G colorable, i.e., does there exist a function f: V $\rightarrow$ 1,2,3,. . .,K such that f(u) $\neq$ f(v) whenever u,v $\in$ E?

We use the GPC algorithm for Graph Coloring at two levels of abstraction: i) to identify good set of nodes that can be colored with a single color and ii) for the overall assignment of all colors. First, we use it to find a maximal independent set with the largest number of incident edges (mIS). Note the difference between a mIS and a MIS. A mIS is a maximal independent set in the graph that encompasses the nodes that are suitable to be colored with a single color, while a MIS is the maximum independent

set. The importance of the mIS is a consequence of the fact that all nodes that belong to a mIS, can be colored with the same color and that they will eliminate the largest number of constraints for coloring the remainder of the graph. All components of the mIS used for graph coloring are identical to the components of the MIS CP algorithm from the previous subsection. The only difference is that the OF and COF are now defined in the following way. OF is equal to the sum of edges incident to the nodes in the CP. The COF, in addition to this component, also considers the sum of the squares of the number of neighbors for each node in the CP. The intuition is that if the remainder of the graph has nodes with high degrees it is more difficult to be colored. In the COF both components are scaled to a value between 0 and 1 by normalizing them respectively against the number of edges and the overall sum of squares of the number of neighbors for all nodes in the initial graph.

Once when we have a mean to select mIS, we can approach the second part of the problem, that is to start to select a few mISs that will form an efficient graph coloring solution. The components of the PC algorithm for this part of the problem are defined in the following way.

Candidate Part (CP). We select k mISs as a single CP. Experimentally, we found that $k = 5$ performs well.

Probabilistic Search. The first phase generates a large number of different mISs. We probabilistically (with probabilities directly proportional to their relative perceived quality) select one mIS at a time according to two criteria. The first criterion is the number of outgoing edges from the nodes in the mIS. The second is the amount of overlap the mIS has with other mISs in the CL. If the mIS overlaps with many of the other CPs in the CL, then by selecting the CP which contains this mIS, we eliminate many of the future possibilities for finding large mISs with many incident edges.

Candidate List (CL). The size of the candidate part depends on the size of the instance. In this case we allow the user to specify the size of the list. For all our experimentation, we used as the default value a list of length 25.

Objective Function (OF). The objective function is the sum of two weighted components:

$$\text{OF}(\ mIS_i) = \alpha_1 P_1 + \alpha_2 P_2$$

The first component directly comes from the following criteria. The main problem in coloring graphs is the number of edges in the graph. This component can be summarized using the following formula.

$$P_1(mIS_i) = \sum_{j=1}^{|mIS_i|} d(v_j)$$

where d is the degree of the node v_i in mIS_i. This property has already been used for evaluating the quality of nodes colored with the same color in two exceptionally well-performing algorithms by Leighton [41] and by Kirovski [39]. The second component is a variation of the first component only it emphasizes on the uniform density of edges in the graph, the reason being that it is much more difficult to color a graph with non-uniform edge density [39].

$$P_2(mIS_i) = \sum_{j=1}^{|mIS_i|} d^2(v_j)$$

Comprehensive Objective Function (COF). The comprehensive objective function (COF) aims to ensure that as many as possible mutually exclusive mIS CPs from the CL are used simultaneously. In order to accomplish this, we introduce a mIS conflict graph illustrated using a small example in Figure 6. The conflict graph has as nodes CPs (mIS) from CL. An edge between two nodes indicates that at least one node is shared between the two CPs (mIS). The weight on the edge is proportional to the number of nodes in the original graph that are common to both mISs. The COF for Graph Coloring has two components: P_3 and P_4. The first component can be expressed as:

$$P_3 = \sum_{j=1}^{\#\_neib\_mIS} w_{ij}$$
$$\text{where } w_{ij} = \sum_{m=1}^{|V_k|} n_m \ , \ V_k = (mIS_i \cap mIS_j),$$
$$\text{and } n_m = \#\text{ of neighboring vertices for } v_m \in V_K.$$

Essentially, P_3 is proportional to the number of neighbors of nodes that are in mIS_i and mIS_j. The second component is formally defined in the following way:

$$P_4 = \sum_{j=1}^{|S_i|} n_j, \text{ where } S_i = (V \backslash U) \cap mIS_i,$$
$$\text{U} = \bigcup_{i=1}^{\#\_of\_mIS} mIS_i,$$
$$n_j = \#\text{ of neighboring vertices for } vij \in S_i$$

The intuition is the following. We find all nodes that are not elements of any mIS in the candidate list. If some mIS has more of this type of nodes, than it is considered more important for immediate inclusion in the final solution because it colors a node which is otherwise difficult to be colored simultaneously with the other nodes.

Stopping Criteria. We stop searching for new CPs for the CL after k_{sc} attempts to find a CP with an improved OF. We found that $k_{sc} = 100$ achieves strong results in practice.

Best Candidate Selection (BCS). We select the best CP simply by finding the CP with the best COF evaluation. In this case, the COF examines all needed aspects of the CP.

Solution Integration. To integrate the BCS into the solution and find the remaining problem to be solved, we merge all the nodes in the CP into a single new node, and reassign all incident edges to this node. The final solution will consist of a fully connected graph, or a clique. Each of the nodes in this graph will be assigned a different color. All nodes encompassed in these final nodes will be colored with the same color.

Overall Control Strategy. For the overall control strategy we do k_{ocs} multi-starts. Experimentally we found $k_{ocs} = 3$ to work well. Note that one can envision numerous more sophisticated strategies that take into account how close were previous runs in terms to eliminate at least one color with additional effort.

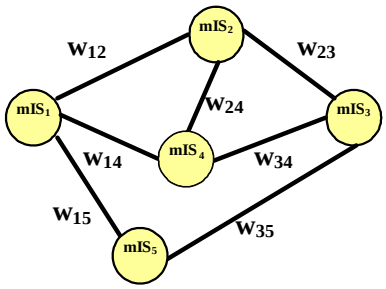

Fig. 6. mIS Conflict Graph.

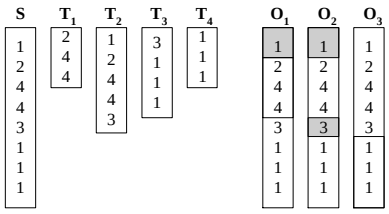

Fig. 7. Sequence Covering Example.

IV. Applications to CAD Problems

In this section, we apply the generic PC approach to two CAD problems: sequence covering and scheduling. We start by formulating the sequence covering problem and establishing its relation to the power modelling task. In addition, we also discuss the relationship between sequential code compression and the sequence covering problem. Next, we employ the PC heuristics to solve the formulated problem. Furthermore, we illustrate the new approach on a small, but illustrative example. Finally, we apply the CP approach to the scheduling problem.

A. The Sequential Code Covering Problem

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 D = $\{d_1, d_2, \ldots, d_n\}$, set of templates T = $\{t_1, t_2, \ldots, t_k\}$ s.t. 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 s.t. not two templates overlap and the number of uncovered symbols in S is less than U?

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 [52]. It has been experimentally verified that this procedure yields extremely accurate power prediction within 3% for Toshiba R3900 microprocessor [52]. 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 [43]. Also note that the sequence covering problem is a special case of the technology mapping problem in logic synthesis [36] and the template matching problem in behavioral synthesis [14].

We further clarify the sequence covering problem using the example shown in Figure 7. Our set of symbols is D = {1,2,3,4}. We have four templates T_1, T_2, T_3, and 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.

We now briefly illustrate the key trade-offs during the sequence covering problem. The best coverage is achieved by option O_3 where the number of uncovered instructions is the least. The first interesting observation is that some of the templates might contain all the elements of another template in the sequence. In the example, T_2 contains T_1 and T_3 contains T_4. As we have seen from the best solution for this example, it is not always the case that the template that is covering more symbols of the sequences in the local area leads toward the best final solution. For example, T_3 has more local coverage in its immediate neighborhood, but using this template would prevent us to use T_2 and thus finding the best solution for this particular example. Therefore, the best selection for the covering template is not only dependent on the coverage performance in the immediate local neighborhood, but also on the complementary function of other templates for covering the most possible number of symbols in the sequence.

When the sequence covering problem is address 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 subsequence of the sequence 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 two values O_i and PS which indicates the likelihood of adding the subset to the CL. O_i represents 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 following. 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 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.

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

Comprehensive Objective Function (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 the overlap and 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

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

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.

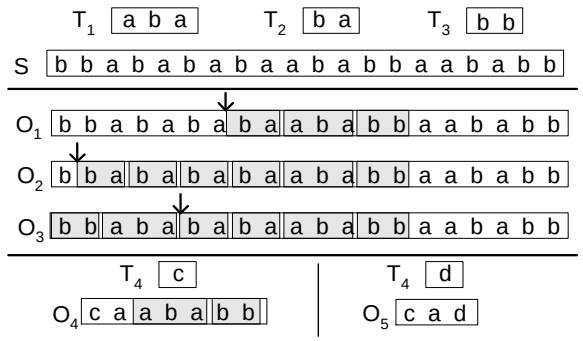

Fig. 8. Sequence Covering Example.

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.

Overall Control Strategy. If after k_{ocs} multi-starts there is no improvement in the number of total uncovered elements then we terminate search. In experimentation we found $k_{ocs} = 5$ 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 Figure 8. 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, 2 and 1 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 Figure 8. 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 rebuilding 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 Figure 8. In this case, the best coverage is 19 out of the 20 symbols.

B. The Scheduling Problem

Scheduling is a mandatory task in both behavioral and software compilation [17]. We assume a synchronous data flow model of computation [40]. The computation is represented as a directed graph, where nodes indicate operations and edges represent data or control dependencies between operations. Each operation may require one or more clock cycles for its execution. The total execution time is given. Note that this abstraction is not only well suited for optimization, but more importantly it is fully adequate for numerous computation intensive application in fields such as digital signal processing and communication [40]. Scheduling is a synthesis task that assigns operations to clock cycles. Assignment is the process of assigning each operation to one of the execution units. The goal is to schedule and assign all nodes in the computation within the given time, so that all data and control dependencies are satisfied and the specified amount of hardware is used. In the simplest case only functional units are specified. But in a more realistic case the registers and interconnect are also specified. The scheduling problem in its simplifies form can be formally stated in the following way.

Problem: Scheduling
Instance: Directed Graph G(V,E), positive integers T and R, $\forall v_i \in V$, and mapping $f(v_i) : v_j \rightarrow 1, \ldots, n$.
Question: Are there $\forall v_i \in V$, $g(v_i) : v_j \rightarrow 1, \ldots, T$ and $h(v_i) : v_i \rightarrow 1, \ldots, R$ s.t. $\forall e_{ij} \in E$, $g(v_i) + f(v_i) \leq g(v_j)$ and for all (i, j), $i \neq j$, if $g(v_i) + f(v_i) \geq g(v_j)$ or $g(v_j) + f(v_j) \leq g(v_i)$ then $h(v_i) \neq h(v_j)$?

Function f(v_i) indicates the duration of operation v_i. Function g(v_i) indicates in which clock cycle the operation is scheduled and function h(v_i) indicates on which execution unit each operation is assigned.

As a preliminary step to the PC approach for scheduling, we compute ASAP (as soon as possible) and ALAP (as late as possible) times for each operation using breadth first search and dynamic programming [49]. After that, we construct the distribution graph [49] for each type of operation that calculates parallelism available in each control step. Distribution graphs contain information about the expected number of operations to be executed in a particular control step assuming that each operation has an equal probability to be scheduled in any clock cycle within its ASAP-ALAP timing interval. The distribution graph is weighted by the cost (area) of an execution unit for a particular type of operation. The distribution graph also contains information about the required interconnects and expected register requirements.

We will follow the following notations: S(v_i) denotes the slack of operation(v_i) and it is defined as : S(v_i) = ALAP(v_i) - ASAP(v_i). The scheduling difficulty (inverse slack) is defined in the following way:

$$\mathrm{I}(v_i) = \frac{1}{S(v_i)+1}$$

We define the components of the PC algorithm for scheduling as follows.

Candidate Part (CP). We select a set of operations which can be scheduled in the same control step as our candidate part. We select the maximal possible number of operations. We selected to define the CPs with respect to the control step because it is conceptually simple and facilitates both efficient implementation and effective exploitation of parallelism. Note that there are numerous other ways in which the CP can be defined. For example, one way is to select all operations which will be assigned to the same resource. Another is to select k operations on arbitrary k resources. We selected the first option for the reason that it preformed best in practice and it is intuitively appealing.

Probabilistic Search. For each control step we search as long as no new CP can be added to the CL is found in $5k$ attempts. We define k as the maximum number of operations that can be assigned to the control step. We select one operation at any time to remove and replace from the CP probabilistically. We measure the percentage of unscheduled operations in their transitive fan-in (TRI) and transitive fan-out (TRO) of each operation, and we give preference to operations that are expensive, have little slack or flexibility, and have many TRI and TRO operations. If the operation has all of these qualities, we want to schedule the operation as early as possible in the scheduling process. This way it will give us a better picture of the difficulty to schedule and assign the remaining operations.

Candidate List(CL). For each control step we keep k_i candidates, where k_i is the number of operations which have ASAP and ALAP times which allow the operation to be scheduled in control step i.

Objective Function(OF). We define the objective function as the weighted sum of two components. The first component P_1, aims at giving benefit to CPs that assigns expensive operations with little slack. The second component analyzes how well the CP preserves chances for scheduling the remaining operations. Note that after we schedule several operations, in principle, all operations in their TRI may have altered their ALAP times and all operations in their TRO may alter their ASAP times. Therefore, the slack of TRI operations may be reduced and the slack of TRO operations maybe increased. If the CP has a large impact on other operations, then by scheduling the operation in the control step may reduce the chances for the scheduling the remaining operations. Also note that if the operations in the TRI and TRO are expensive and have small amounts of slack, the CP has very little benefit. Therefore the OF has the following form:

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

where the component p_1 is the sum of the scheduling difficulties of all operations in the CP and p_2 is the sum of the change in the scheduling difficult of operations that in TRI and TRO of each operation in the CP. In both components each operation is weighted by a factor that is proportional

to the cost of execution units of which the operation may be assigned. We set α_1 to 1 and α_2 to .5.

Comprehensive Objective Function (COF). We define the COF as an enhanced version of the OF in the sense that we include the interconnect and register utilization when calculating p_1 and p_2. We essentially use criteria and mechanisms for treating interconnect and registers from [49], [**?**]. For example, interconnect is treated in absolutely the same way as operations after each transfer is explicitly defined.

Stopping Criteria. We stop searching for new CPs for the CL after having no success for $5k_{sc}$ attempts. We define k_{sc} as the number of operations which can be possibly scheduled in the i^{th} control step.

Best Candidate Selection (BCS). For the BCS we consider two aspects, the quality of the CP according to it's COF and the amount of damage each CP does to other CPs in the CL. The intuition behind the second component is that we do not want to eliminate other good CPs for other controls steps in the CL by selecting a CP. The damage is done exactly in the same way as the objective function for all operations which belong to a pair of considered CPs.

Solution Integration. We integrate our solutions by scheduling the operations in our BCS and recalculating the ASAP and ALAP values for all remaining operations.

Overall Control Strategy. As long as a solution is not found we continue to restart. If we can find a schedule for the operations, it is possible that less hardware is needed to complete the task and we provide this information to the designer as feedback. To avoid infinite loops, we count the number of unassigned operations at the end of each restart. If we do not find better solution in k attempts (k is specified by the user), we terminate the search.

V. 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 [16], [39], [**?**], [52]. 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 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.

A. Maximum Independent Set

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 vise 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 1 indicates the name of the maximum clique instance (the instances are from [13], [**?**], 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.

Name	V	E in Clique	E in MIS	γ	CPU [16]	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

TABLE I
Experimental Results for Maximum Independent Sets.

Name	V	E	χ	PC # of Colors	lmXRLF # of Colors
c-fat200-1	200	1534	12	13	12
DSJC125.1	125	736	5	6	6
Ex3a	44	176	10	10	10
Exam1	200	17124	126	126	126
Exam3	300	36801	162	162	162
flat300_20_0	300	21375	20	40	20
flat1000_50_0	1000	245000	50	50	50
flat1000_60_0	1000	245830	60	61	61
flat1000_76_0	1000	246708	76	78	85
le450_15d	450	16750	15	24	21
le450_25c	450	17343	25	28	28
le450_5d	450	9757	5	8	5
MANN_a9	45	918	18	18	18
queen7_7	49	476	7	10	9
queen9_9	81	2112	10	12	10
queen13_13	169	6656	13	17	18
R125.5	250	3838	36	38	37
sqelq2.I2	182	3254	26	26	26
school1_nsh	358	16710	14	24	14

TABLE II
Experimental Results for Graph Coloring.

B. Graph Coloring

Table II provides the experimental results for the application of the PC heuristics to the graph coloring problem. Testing was performed on instances from [39]. The first column identifies the name of the instance and the second and third show the number of vertices and edges, respectively. The chromatic number for each instance is listed in the fourth column. Finally, the results of the PC heuristic are shown.

It is important to note that the Table II has two sharply different types of examples: real-life easy to solve instances and intentionally constructed difficult to solve benchmarks. Flat300_20_0 is one of the well known Mycielski's graphs which are difficult to color due to the fact that their clique number is 2, while the required number of colors sharply increases as a function of the problem size. Lei450_xd are Leighton's graphs that are constructed by first creating a clique of user specified size. Each node in the clique is assigned to a separate group, that will be colored with one color. Then all other nodes are randomly assigned to one of the groups. Finally, a relatively larger number of edges is added between nodes in the different groups. Coloring this graph will the minimum number of colors is considered so difficult that this class of problems is considered as a good candidate for cryptographical one-way function, i.e. essentially indicates that the difficulty of this problem is as high as any other known problem [32].

To better quantify the effectiveness of the PC heuristic for graph coloring we apply the approach to a set of random R(V, p), p = 0.5 graphs. It is well known that these graphs are exceptionally difficult for coloring. However, there exists a long tradition in the algorithmic community to conduct comprehensive experimentation using these benchmarks. The lmRLF algorithm [39] reports superior results over all previously published. In Table III, the first row presents the size of the random coloring instance. The second row is the chromatic number for each of these instances. We present the average number of colors used to color the graph and the average runtime in seconds for a set of 10 graph instance for each random graph, for both the lmRLF and the PC heuristic. On average, the PC heuristic colors the random graphs 0.5 colors less and the runtime was significantly lower than the lmRLF algorithm's runtime. As Table III indicates we significantly reduce the required number of colors and approach the theoretically expected minima indicated in the second row. In addition, the runtime is reduced by a factor of more than five times.

C. Sequence Covering

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 know 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. Fore 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 IV 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 third column represents the trial number. The length of the sequences is presented in the column four. 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.

D. Scheduling

We use the Hyper benchmark suite to test our PC heuristic [53]. There are two major reasons for this decision. The first is that Hyper is one of very few tools which are public domain and is able to run a variety of real-life designs. Second, and more importantly, the tool provides sharp estimation techniques which report accurate lower bounds on what is achievable on a given instance. Therefore, we have a means to accurately estimate the effectiveness of the PC heuristic for Scheduling.

We consider both the area and bus reduction for each of

	R(125,0.5)		R(250,0.5)		R(500,0.5)		R(1000,0.5)	
	16		27		46		80	
	Colors	Time	Colors	Time	Colors	Time	Colors	Time
lmXRLF	18.2	46.2	29.9	172	49.7	4612	85.1	18083
ProbConstr	17.9	9.12	29.4	37.1	48.9	843.3	84.5	3004.3

TABLE III
Experimental Results of Random Graph Coloring Examples.

the design in our experiments. The results are presented in Table V. The first column in the table presents the name of the instance while the second gives the number of nodes in the instance. The next three columns represent the area found for each instance; the original, the area found using the PC heuristic, and the lower bound found by the HYPER tool. We present the percentage improvement of the CP area over the HYPER area and the percentage of the CP area over the lower bound next.

The eight, ninth, and tenth columns present the number of buses, for the HYPER tool, the PC approach, and the lower bound found by the HYPER tool. The last two columns contain the percentage difference for the buses. On average, we observe a 10.45% improvement in terms of area and a 14.4% improvement on the number of buses over the HYPER tool. The runtime at average was less than $\frac{1}{10}$ of the runtime of the Hyper default scheduler.

VI. Conclusion

We introduced a new PC algorithm paradigm. We search for a small part of the problem that can be solved efficiently and in such a way that the remaining problem is as much as possible amenable 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 generic NP-complete problems (maximum independent set and graph coloring) and two design problems (sequential code covering and scheduling). Extensive experimentation indicates that the new algorithm is capable of achieving competitive or better results than previously published approaches, often with shorter runtimes.

References

[1] E. Aarts, J. Korst. Simulated Annealing and Boltzmann Machines: a stochastic approach to combinatorial optimization and neural computing. New York: Wiley, 1989.

[2] A.V. Aho, M. Ganapathi, S.W.K. Tjiang. "Code generation using tree matching and dynamic programming." ACM Transactions on Programming Languages and Systems, vol.11(4), pp.491-516, 1989.

[3] N. Alon, M. Krivelevich, B. Sudakov. "Finding a large hidden clique in a random graph." Random Structures & Algorithms, vol.13(3-4), pp.457-66, 1998.

[4] C.J. Alpert, A.B. Kahng. "Recent Directions in Netlist Partitioning: A Survey." Integration: The VLSI Journal, vol.19(1-2), pp.1-81, 1995.

[5] L. Babel, G. Tinhofer. "A branch and bound algorithm for the maximum clique problem." ZOR-Methods and Models of Operations Research, vol.34(3), pp.207-217, 1990.

[6] J. Bitner, E. M. Reingold. Backtrack programming techniques. Communications of the ACM, vol.18, pp. 651-655, 1975.

[7] B. Bollobas. Random Graphs. London, UK: Academic Press, 1985.

[8] I. M. Bomze, M. Budinich, P. M. Pardalos, M. Pelillo. "The maximum clique problem." Handbook of Combinatorial Optimization, vol.4(1), pp.1-74, 1999.

[9] G. Brassard, P. Bratley. Algorithmics: theory and practice. Englewood Cliffs, N.J.: Prentice Hall, 1988.

[10] D. Brelaz. "New methods to color the vertices of a graph." Communications of the ACM, vol.22(4), pp.251-256, 1979.

[11] F.Y. Busaba, P.K. Lala. "A Graph Coloring based approach for self-checking logic circuit design." Asian Test Symposium, IEEE Computer Society Press, pp.327-330, 1995.

[12] E. G. Coffman et al. Computer and job-shop scheduling theory. New York: Wiley, 1976.

[13] J. Cong, M. Hossain, N.A. Sherwani. "A provably good multilayer topological planar routing algorithm in IC layout designs." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.12(1), pp.70-78, 1993.

[14] M.R. Corazao, M.A. Khalaf, L.M. Guerra, M. Potkonjak, J.M. Rabaey. "Performance optimization using template mapping for datapath-intensive high-level synthesis." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.15(8), pp. 877-888, 1996.

[15] T. H. Cormen et al. Introduction to algorithms. New York: McGraw-Hill, 1990.

[16] O. Coudert. "Exact coloring of real-life graphs is easy." ACM/IEEE Design Automation Conference, pp.121-126, 1997.

[17] G. De Micheli. Synthesis and optimization of digital circuits. New York: McGraw-Hill, 1994.

[18] ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmark/. Benchmark for Graph Coloring and clique, 1993.

[19] C. M. Fiduccia, R.M. Mattheyses. "A Linear Time Heuristic for Improving Network Partitions." ACM/IEEE Design Automation Conference, pp.175-181, 1982.

[20] L.J. Fogel, A.J. Owens, M.J. Walsh. Artificial Intelligence through Simulated Evolution. New York: John Wiley, 1966.

[21] D. Gajiski, N.D. Dutt, A.C-H. Wu, S.Y-L. Lin. High Level Synthesis-Introduction to Chip and System Design. Boston, M.A.: Kluwer Academic Publishers, 1992.

[22] M.R. Garey, D.S. Johnson. "Computers and Intractability: A Guide to the Theory of NP-Completeness." San Francisco: Freeman, 1979.

[23] F. Gavril. "Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph." SIAM Journal on Computing, vol.1(2), pp.180-187, 1972.

[24] F. Glover. "Tabu search part I." ORSA Journal on Computing, vol.1(3), pp.190-206, 1989.

[25] F. Glover. "Tabu search Part II." ORSA Journal on Computing, vol.2(1), pp.4-32, 1990.

[26] D.E. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning. Reading, Mass.: Addison-Wesley, 1989.

[27] C. A. R. Hoare. "Quicksort." The Computer Journal, vol.5(1), pp.10-15, 1962.

[28] C. Hoffman, M.J. O'Donnell. "Pattern matching in trees." Journal of the ACM, pp.68-95, 1982.

[29] J. Holland. "Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence." Ann Arbor: University of Michigan Press, 1975.

[30] D. S. Johnson, C. R. Aragon, L. A. McGeoch, C. Schevon. "Op-

Number Templates	Templates Sequence	Trial	Seq. Length	Optimal Solution	Greedy	GPC	Delayed Decision
20	1000	1	4720	0.96864	0.63257	0.92389	0.92840
20	1000	2	4741	0.98397	0.72002	0.94790	0.94790
20	1000	3	4306	0.96656	0.83621	0.87181	0.93939
20	2000	1	8017	0.96857	0.69382	0.96408	0.95239
20	2000	2	8947	0.98089	0.58577	0.93652	0.95057
20	4000	1	19904	0.99131	0.79584	0.92248	0.96554
20	6000	1	25557	0.97766	0.40704	0.92112	0.91572
40	2000	1	7498	0.98133	0.53873	0.93653	0.92380
40	2000	2	9595	0.98802	0.63985	0.92876	0.91937
40	4000	1	14686	0.98495	0.36827	0.93416	0.92499
Average	-	-	-	0.97919	0.621812	0.928725	0.93681

TABLE IV

Experimental Results for Sequence Covering.

Name	Ops	Hyper Area	CP Area	LB Area	% Improv.	% LB	Hyper Buses	CP Buses	LB Buses	% Improv.	% LB
Sqrt	11	0.52	0.47	0.44	8.7	6.8	4	4	4	0	0
Wdf8	23	2.05	1.84	1.72	10.3	7.0	23	20	18	13.04	11.1
lir7	33	5.46	5.04	4.66	7.7	8.2	28	26	24	7.14	8.3
Cascade8	34	2.41	2.08	2.06	13.6	1.0	29	23	20	20.69	15.0
modem	39	1.39	1.19	1.1	14.2	8.2	12	12	12	0	0.0
parallel8	39	2.85	2.64	2.48	7.4	6.5	37	34	32	8.11	6.3
conv5	45	3.6	3.24	2.88	9.9	12.5	68	51	46	25	10.9
sine	49	1.45	1.34	1.2	7.8	11.7	30	22	19	26.67	15.8
Wavele	53	4.33	3.97	3.48	8.3	14.1	55	44	39	20	12.8
Dsrect25	62	7.46	6.97	6.41	6.6	8.7	66	54	48	18.18	12.5
DSfir51	127	10.88	9.45	8.4	13.1	12.5	137	102	89	25.55	14.6
fir133	130	11.96	9.84	8.98	17.7	9.6	18	16	16	11.11	0.0
fir100	302	20.59	18.43	18.2	10.5	1.3	17	15	14	11.76	7.1

TABLE V

Experimental Results for Scheduling. LB represents Lower Bound.

timization by simulated annealing: An experimental evaluation, part II, Graph Coloring and number partitioning." Operations Research, vol. 39(3), pp.378-406, 1991.

[31] D. Joseph, J. Meidanis, P. Tiwari. "Determining DNA sequence similarity using maximum independent set algorithms for interval graphs." Algorithm Theory- SWAT, pp.326-337, 1992.

[32] A. Juels, M. Peinado. "Hiding cliques for cryptographic security." Designs, Codes and Cryptography, vol.20(3), pp.269-280, 2000.

[33] R.M. Karp, A. Wigderson. "A fast parallel algorithm for the maximal independent set problem." Journal of ACM, vol.32(4), pp.762-773, 1985.

[34] D. Karger, R. Motwani, M. Sudan. "Approximate Graph Coloring by semi-definite programming." Journal of the ACM, vol.45 (2), pp.246-265, 1998.

[35] B. W. Kernighan, S. Lin. "An Efficient Heuristic Procedure for Partitioning Graphs." Bell System Technical Journal, vol. 49(2), pp. 291-308, 1970.

[36] K. Keutzer. "DAGON: Technology Binding and Local Optimization." ACM/IEEE Design Automation Conference, pp.341-347, 1987.

[37] S. Khanna, K. Kumaran. "On wireless spectrum estimation and generalized Graph Coloring." IEEE INFOCOM, Conference on Computer Communications, vol.3, pp.1273-1283, 1998.

[38] S. Kirkpatrick, C. Gelatt, M. Vecchi. "Optimization by Simulated Annealing." Science, Vol.220, pp.671-680, 1983.

[39] D. Kirovski, M. Potkonjak. "Efficient coloring of a large spectrum of graphs." ACM/IEEE Design Automation Conference, pp.427-432, 1998.

[40] E.A. Lee, D.G. Messerschmitt. "Static scheduling of synchronous data flow programs for digital signal processing." IEEE Transactions on Computers, vol.C-36(1), pp.24-35, 1987.

[41] F.T. Leighton. "A Graph Coloring Algorithm for Large Scheduling Algorithms." Journal of Research National Bureau Standards, vol.84, pp.489-506, 1979.

[42] S. Lin, B.W. Kernighan. "An effective heuristic algorithm for the traveling-salesman problem." Operations Research, vol. 21(2), pp.498-516, 1973.

[43] H. Massalin. "Superoptimizer—a look at the smallest program." Architectural Support for Programming Languages and Operating Systems (ASPLOS II), pp.122-126, 1987.

[44] M.C. McFarland, A.C. Parker, R. Camposano. 'The high-level synthesis of digital systems." Proceedings of the IEEE, vol.78(2), pp.301-318, 1990.

[45] N. Metropolis, A. Rosenbluth, R. Rosenbluth, A. Teller, E. Teller. "Equation of state calculations by fast computing machines." Journal Chemical Physics, vol.21, pp.1087-1092, 1953.

[46] C. Morgenstern. "Distributed Coloration Neighborhood Search." Discrete Mathematics and Theoretical Computer Science, vol.26, pp.335-358. 1996.

[47] R. Motwani, P. Raghavan. Randomized algorithms. New York: Cambridge University Press, 1995.

[48] S. Note, F. Catthoor, G. Goossens, H. De Man. "Combined hardware selection and pipelining in high-performance data-path design." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.11(4), pp.413-423, 1992.

[49] P.G. Paulin, J.P. Knight, "Force-directed scheduling for the behavioral synthesis of ASICs." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.8(6), pp.661-679, 1989.

[50] J. Pearl. Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley, 1984.

[51] P.W. Purdom. Search rearrangement backtracking and polynomial average time. Artificial Intelligence, vol. 21, pp.117-133, 1983.

[52] G. Qu, N. Kawabe, K. Usami, M. Potkonjak. "Function-level power estimation methodology for microprocessors." ACM/IEEE Design Automation Conference, pp.810 - 813, 2000.

[53] J.M. Rabaey, C. Chu, P. Hoang, M. Potkonjak. "Fast prototyp-

ing of datapath-intensive architectures." IEEE Design & Test of Computers, vol.8 (2), pp.40-51, 1991.
[54] J. Rabaey, M. Potkonjak, "Estimating Implementation Bounds for Real Time Application Specific Circuits", IEEE Transaction on CAD, vol.13(6), pp. 669-683, 1994.
[55] D.S. Rao, F.J. Kurdahi. "On clustering for maximal regularity extraction." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.12(8), pp.1198-1208, 1993.
[56] S. J. Russell, P. Norvig. Artificial intelligence : a modern approach. Englewood Cliffs, N.J. : Prentice Hall, c1995.
[57] L.A. Sanchis. "Multiple-way network partitioning." IEEE Transactions on Computers, vol.38(1), pp.62-81, 1989.
[58] A. Singh, M. Marek-Sadowska. "Circuit clustering using Graph Coloring." International Symposium on Physical Design, pp.164-169, 1999.
d
[59] R.E. Tarjan, A.E. Trojanowski. "Finding a maximum independent set." SIAM Journal on Computing, vol.6(3), pp.537-546, 1977.
[60] C.W. Wu. "Graph Coloring via synchronization of coupled oscillators." IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, vol.45(9), pp.974-978, 1998.