Gradual Relaxation Techniques with Applications to Behavioral Synthesis*

Zhiru Zhang, Yiping Fan, Miodrag Potkonjak, Jason Cong

Computer Science Department, University of California, Los Angeles

Los Angeles, CA 90095, USA

{zhiruz, fanyp, miodrag, cong}@cs.ucla.edu

ABSTRACT

Heuristics are widely used for solving computational intractable synthesis problems. However, until now, there has been limited effort to systematically develop heuristics that can be applied to a variety of synthesis tasks. We focus on development of general optimization principles so that they can be applied to a wide range of synthesis problems. In particular, we propose a new way to realize the most constraining principle where at each step we gradually relax the constraints on the most constrained elements of the solution. This basic optimization mechanism is augmented with several new heuristic principles: minimal freedom reduction, negative thinking, calibration, simultaneous step consideration, and probabilistic modeling.

We have successfully applied these optimization principles to a number of common behavioral synthesis tasks. Specifically, we demonstrate a systematic way to develop optimization algorithms for maximum independent set, time-constrained scheduling, and soft real-time system scheduling. The effectiveness of the approach and algorithms is validated on extensive real-life benchmarks.

1. MOTIVATION

We have two strategic objectives in this paper: development of general optimization principles and applications to a wide range of synthesis problems. The optimization goal is to advance the state of the art in the design of heuristic algorithms. Heuristics are widely used for solving computational intractable synthesis problems. However, until now, there has been limited effort to systematically develop heuristics that can be easily applied to a variety of synthesis tasks.

In this paper, we propose a new heuristic optimization paradigm that can be applied on a broad spectrum of computationally intractable problems. While the traditional most constraining principle always addresses the most constrained part of the problem first, we employ the most constraining principle where at each step we make a decision that maximally relaxes the constraints on the most constrained elements of the solution. This basic optimization mechanism is augmented with several new heuristic insights: minimal freedom reduction, negative thinking, calibration, simultaneous step consideration, and probabilistic modeling. We call them the ***gradual relaxation techniques***.

The minimal freedom reduction principle aims to make the minimal possible quantum of decision at each step. The rationale is that after a small step is made, one can better evaluate its impacts and prevent the heuristic from following a greedy mode of optimization to produce local optimal solutions. The main way to realize this principle is to use negative thinking, i.e., to decide what the optimization process will not do at the next step, instead of what to do. The options that stay at the end of this process form a set of decisions to yield a high quality solution. Calibration is a step where the chances for optimization along a particular direction are evaluated and compared. If a particular direction is not promising, all efforts along that line are terminated or assigned a lower priority. A typical example is resource allocation. If the design is such that the number of required adders cannot be reduced below some bound, and there is room for minimization of the number of multipliers, the scheduling is conducted so that the operation that has to be assigned to multipliers preserves maximum slack. Moreover, in some cases, it is advantageous to simultaneously consider several small steps applied on several parts of the problem in order to evaluate their compound impact. Finally, in order to further increase the effectiveness of the proposed heuristic, we also propose a new, more realistic technique for probabilistic modeling during the optimization process.

Optimization techniques can rarely be developed completely out of context. Even more difficult is to evaluate their effectiveness, unless they are applied on important, real-life problems and compared with existing methods. Therefore, our second and actually main goal is to develop a system of effective and fast optimization techniques for common behavioral synthesis tasks. Specifically, we demonstrate a systematic way to develop optimization algorithms for maximum independent set, time-constrained scheduling, and soft real-time system scheduling. The problems are selected not only because of their importance, but also because they have a very different nature. For example, in static schedule one has to map all of the operations to control steps while in the MIS problem the goal is to select only a subset of nodes.

While behavioral synthesis is still looking to establish itself, the intuition is that this will make algorithms more accurate. Consider the simple motivational example shown in Figure 1. We list all the possible configurations of a CDFG with 3 operations scheduled to 5 available control steps. We denote *prob(i, t)* to be the probability of assigning the operation *i* in control step *t*. According to traditional approaches, uniform distribution is assumed. For example, we have *prob(1,1) = prob(1,2) = prob(1,3)* =0.33, However, the realistic probabilities should be *prob(1,1)* = 0.6, *prob(1,2)* = 0.3 and *prob(1,3)* = 0.1, which are significantly different from the previous ones.

* This research is partially supported by National Science Foundation under award CCR-0096383.

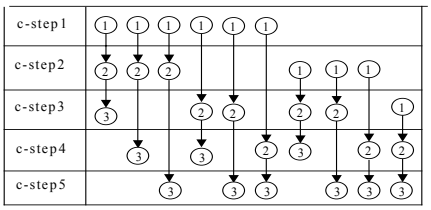

Figure 1. A simple example of the non-uniform distribution

2. RELATED WORK

In this section, we survey the related work along the following four directions: maximally constrained minimally constraining heuristics, force-directed optimization, static and soft real-time system scheduling, and maximum independent set.

Bitner and Reingold [2] were the first to propose the systematic use of the maximally constrained variable selection paradigm. They use it to guide the search within a generic backtracking search for the optimal solution. The first to use the maximally constrained rule as part of fast heuristic program was Brelaz for graph coloring [3]. Another popular family of heuristics is based on notion of slack [7][9]. Slack is usually defined as value proportional to the cardinality of a set of still viable options for assigning a specific variable from the problem formulation. In CAD literature, the most constrained least constraining paradigm has been widely used, mostly under the name of force-directed heuristics [20]. A more global picture of the role of a maximally constrained minimally constraining approach as an efficient heuristics is given in [21].

Scheduling is a mandatory task in behavioral synthesis and compilation [17][18]. In behavioral synthesis, scheduling is closely related to resource allocation, a task where the hardware required for a specific application is allocated. We target two important and widely required types of scheduling: static and soft real-time systems. For the former, we assume a synchronous data flow model of computation [15]. Our goal is to minimize the required hardware for a given sampling rate.

The proliferation of multimedia applications and soft real-time system is gaining rapidly increasing attention. The basis for soft-real system scheduling was formed in the work done in the Garcia-Molina research group [13][1]. In multimedia research, soft real-time system are most often used to model video and WWW servers [12]. Real time system community has placed emphasis on a sound formal definition of requirements associated with soft real-time systems [6]. In CAD and embedded system literature notable soft real-time related efforts include [28][27][22]

Maximum Independent Set (MIS) is one of the most popular generic NP-complete problems [8]. For example, it was among the first set of problems proved to be NP-complete [8]. MIS is mainly used in one of its modifications as a set during graph coloring. For example, it has been experimentally demonstrated that in many domains, finding an MIS is sufficient to make coloring popular benchmarks both fast and provably optimal [14].

3. GENERIC TECHNIQUE

In this section, we introduce the seven main new insights that can be used to build a new generation of efficient heuristics for a variety of computationally intractable problems. The insights are as follows:

(1) *Most constrained principle*. The traditional most constrained paradigm for developing heuristics looks to first resolve the components (e.g., nodes or variable) that participated in the largest number of constraints or the constraints that are very strict. The rationale is that it is better to resolve this situation early, while there is still enough freedom to resolve them. The components are usually resolved in such a way that the difficulty of still unresolved constraints is minimally impacted. In practice, the effectiveness of these heuristics is mainly due to the second insight. We propose to use a new insight that resolves the components that participate in the same constraints as the most difficult components. Note that these are not necessarily the most difficult components themselves. The rationale is that these components can often be resolved in such a way as to make the most difficult constraints more relaxed.

(2) *Minimal freedom reduction*. The key impediment to effectiveness of a heuristic is the often greedy behavior of optimization, where short term benefits are achieved at the expense of increased difficulty and cost in later stages of optimization. One way to avoid such behavior is to make a small gradual atomic decision and evaluate its individual impact before committing to large decisions.

(3) *Negative thinking*. This idea is a specific way to realize the previous paradigm. Essentially, it is often more beneficial to state what will not be considered as option instead of resolving a specific component of the solution. The similar ideas appeared in [26] for a special case of scheduling and in [4] for the standard cell global routing (i.e., the iterative deletion method). We generalize it other problems. For example, for the MIS problem, it means that we will eliminate nodes as candidate for MIS one by one, until the remaining set of nodes does not have any incident edges. This is in contrast to the standard procedure of selecting nodes for MIS one by one.

(4) *Compounding Variables.* At first sight it seems that it is impossible to apply the previous two paradigms to problems where variables can be assigned only to binary values. For example, in the SAT problem, the negative decision to not assign a variable to 1 is that the variable is assigned to 0. This difficulty can be resolved, if we consider two or more variables simultaneously. For example, we may decide that variable x_i and x_j will not be simultaneously assigned to value 1. Therefore, after this decision, there are still three options how these two variables can be simultaneously assigned.

(5) *Simultaneous steps consideration*. The previous idea can be further generalized even when there is no intrinsic need to create compound variables. The practice indicates that it is often advantageous to simultaneously consider a small negative decision on a set of variables.

(6) *Calibration.* In many situations, one can conclude that the way a subset of variables is resolved will not have an impact on the quality of the final solution. In that situation, there is no need to preserve the options for these variables. The solution to the resource allocation problem discussed in Section 1 is a typical example of calibration.

(7) *Probabilistic modeling*. Our motivational example indicates this assumption is rarely close to reality, even when very small instances are considered. Therefore, we propose to model options on how to resolve each variable in a non-uniform way as a function of all constraints that are imposed on that particular variable.

4. WHEN IS GRADUAL RELAXATION MOST EFFECTIVE?

It is well know that different types of algorithms are best suited for different types of optimization problems. For example, greedy algorithms are optimal and the fastest for the problems that have matroid structure [19]. Similarly, in order to apply dynamic programming, the problem of interest has to have the property that the optimal solution is composed of optimal sub-solutions. At the same time, it is important to realize that the effectiveness of a particular algorithm also greatly depends on the structure of the instance. For example, it is well known that the graph coloring problem for sparse graphs can be almost always solved optimally [25][5].

In general, however, it is not easy to identify classes of problems and instances that can be solved using a particular technique efficiently. Most often these insights are obtained by comprehensive studies and statistical analysis [10][11]. Nevertheless, it is possible to obtain some important insights by considering specific characteristics of the problem or the instance. We first focus our attention on the type of instances where the system of gradual relaxation performs well.

Force directed approaches work well when there exist significant difference in slacks (i.e., range of value that a particular variable can be assigned to) and strictness of constraints that are associated with each variable. The technique of gradual relaxation through negative thinking provides additional effectiveness when a large number (or percentage) of variables have significant slack and when variables have more complex interactions among a large number of constraints. This observation is a consequence of the fact that in these situation gradual relaxation will better reveal the impact of each optimization decision.

Compounding variables and simultaneous steps consideration techniques are most effective when each variable itself has a set of potential values with small cardinality. In particular, it is effective for binary variables. Calibration is most effective for instances where the final solution only involves relatively few occurrences of each type of resources. Finally, probabilistic modeling is most relevant for large and complex instances.

Furthermore, we expect that problems where each variable can be assigned to many values (e.g. coloring and scheduling) are more amenable for optimization using gradual relaxation than problems that are defined on binary variables (e.g. SAT).

5. DRIVER EXAMPLES

In this section, we demonstrate how a properly selected subset of the proposed heuristic optimization principles can be applied to common behavioral synthesis tasks.

5.1 Maximum Independent Set and Graph Coloring

In this subsection, the MIS problem is used to demonstrate the most constraining, minimal freedom reduction, and negative thinking principles.

In behavioral synthesis, the resource sharing problem is usually transformed to the graph coloring problem for the corresponding resource conflict graph. Unfortunately, graph coloring is an NP-complete problem in general [8], defined as:

Problem: ***Graph k-Coloring***

Given: (1) A graph $G\ (V, E)$ with vertex set V and edge set E, (2) A positive integer $K \leq |V|$.

Objective: To determine whether there exists a coloring of G using no more than k unique colors so that for every edge $(u, v) \in E$, u and v have different colors.

The ***graph coloring*** problem is to find the minimum number of colors needed to color a given graph.

An efficient heuristic to solve this problem was presented in [14]. Its approach is to divide the whole problem into serial independent set problems, i.e., in every step the algorithm searches a constrained independent set and assigns one color to its vertices. The independent set search is basically an iterative improvement algorithm, which randomly generates an initial solution and defines the move as vertex exclusion/inclusion. The maximum independent set is also an NP-complete problem stated as:

Problem: ***Maximum Independent Set***

Given: a graph $G\ (V, E)$ with vertex set V and edge set E.

Objective: To find the maximum-size independent set in G. An independent set of G is a subset $V' \subseteq V$ of vertices such that if $u \in V'$ and $v \in V'$, then u and v are not adjacent.

We focus on applying the gradual relaxation technique to solve the MIS problem. We observe that typically in a real-life graph, the MIS size is much smaller than the total graph size. Thus the decision to choose a vertex into an independent set imposes much more constraints to the other node than the decision to choose a vertex to be out of the independent set. Following the minimal freedom reduction principle, we drop off vertices gradually from an initial vertex set to obtain an independent set.

The algorithm, described in Figure 2, is fairly straightforward. We start with an initial solution set containing all of the vertices. At each step, we select the maximally constrained vertex and remove it from the current vertex set until the solution is legal. We use *force* to denote the cost function. Note that a vertex with a large number of neighbors is unlikely to be in the resulting independent set. Hence, we have the following *Level-1* heuristic for the force calculation:

$$force\ (v) = Num\_of\_Neighbors\ (v)$$

However, by looking forward one level, we have the observation that for a vertex, if the number of its neighbors' neighbors is very large, it tends to be in the resulting set. Figure 3 shows this situation. The black vertex has a maximum degree, but it should be in the MIS with the eight leaf vertices.

```
S ← V
while S is not a independent set
        update force for every vertex
        select the vertex v with maximal force
        remove v from S
return S
```

Figure 2. The gradual relaxation algorithm for MIS

To describe this look-ahead force, we define the *Level-2* heuristic as follows

$$force\ (v) = \sum_{u \in Neighbors\ (v)} (1 / Num\_of\_Neighbors\ (u))$$

The force update is in O(|V|) time for the Level-1 heuristic and $O(|V|^2)$ for the Level-2 heuristic. In every step of the algorithm, we remove one vertex. Hence the time complexity is $O(|V|^2)$ and $O(|V|^3)$.

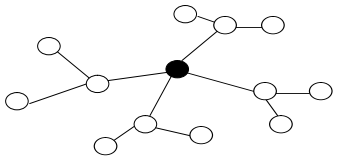

Figure 3. An example of the Level-2 heuristic

5.2 Time-Constrained Scheduling for Behavioral Synthesis

Problem: Time-Constrained Scheduling

Given: (1) A CDFG *G (V, E)*. (2) A timing constraint *T*.

Objective: to schedule the operations of *V* into *T* control steps such that the resource usage is minimized.

In this subsection we show how to use the negative thinking principle to improve the widely accepted force-directed scheduling (FDS) algorithm [20].

5.2.1 Enhancement by Negative Thinking

The essence of FDS algorithm is to reduce the resource usage by evenly distributing the operations to the available control steps. The FDS algorithm uses the uniform probability that an operation will be scheduled within its ALAP-ASAP range.

At each step of FDS algorithm, the time frame of the most constrained operation with least forces will be reduced to a single control step. That is a significant step with potentially high impact. Following the most constraining and negative thinking principles, we first select the operation that has options to be scheduled to the most congested time slot. After that, we prune only one control step, the most congested one, from the operation's time frame. Therefore, we always keep a global picture before scheduling an operation by gradually shrinking the time frames and postpone the decision to a later stage when more accurate estimates are available.

5.2.2 Efficiency Improvement

Since the above method tends to increase the timing complexity, we employ the Gradual Time-Frame Reduction (*GTFR*) technique proposed in [26] to speedup the process. Suppose the time frame of the operation *n* is denoted as $[a_n, b_n]$. In each iteration, *GTFR* only computes for each node *n* the forces of assigning *n* to a_n and b_n instead of all its feasible control steps. The assignment incurring the most force will be identified and the corresponding control step will be removed. As pointed out in [26], the time complexity of this algorithm is $O(T^2N^3)$, where *T* is the timing constraint and *N* is the number of operations. It is the same as that of the basic force-directed scheduling. Although the removal of each control step increases the number of iteration to *O(TN)*, the tentative assignments has to be calculated reduce to 2, instead of *O(T)* times.

5.3 Soft Real-Time Scheduling

In this subsection, we use soft real-time scheduling to illustrate the effectiveness of minimal freedom reduction and probabilistic modeling principles.

Problem: Soft Real-Time Scheduling

Given: (1) A set of $\Psi=\{\tau_1, \tau_2, \ldots \tau_n\}$ tasks and each task $\tau_i=(a_i, d_i, e_i)$ is characterized by an arrival time a_i, a deadline d_i and an execution time e_i. (2) A single processor *P*. (3) A timing constraint *T*.

Assumptions: All tasks are non-preemptive and independent.

Objective: To schedule a subset of tasks in Ψ on processor *P* within the available time *T* so that the number of tasks scheduled is maximized.

5.3.1 Basic Definitions

The time frame of a task is the time interval within which it can be scheduled. Time frame of task τ_i is denoted as *time-frame*$=[a_i, d_i]$. The mobility of a task determines the maximum number of different schedules for it. We denote the mobility of a task τ_i as $mobility(\tau_i)=|time\text{-}frame(\tau_i)|-e_i+1$. We denote the probability of the task τ_i at a time slot *t* as $prob(\tau_i, t)$. It can be derived by the equation shown in the left side of Figure 4. The probability distribution is non-uniform as shown by the trapezium-shape curve in the right side of Figure 4.

$$prob(\tau, t) = \begin{cases} \dfrac{(t - a_i + 1)}{mobility(\tau_i)} & t - a_i + 1 < e_i \\ \dfrac{(d_i - t + 1)}{mobility(\tau_i)} & d_i - t + 1 < e_i \\ \dfrac{e_i}{mobility(\tau_i)} & Otherwise \end{cases}$$

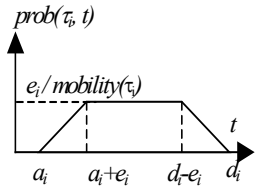

Figure 4. Definition and the curve of *Prob* (τ_i, t)

A distribution graph is defined to capture the likelihood of the potential competitions among different tasks for a single time slot. The distribution at the time slot *t* is computed by $DG(t) = \sum_{i=1}^{n} prob(\tau_i, t)$. Two tasks τ_i and τ_j are said to be in conflict when their time frames overlap. It is denoted by

$$conflict(\tau_i, \tau_j) = (time\text{-}frame(\tau_i) \cap time\text{-}frame(\tau_j) \neq \varnothing).$$

5.3.2 Solution Overview

Observe that the less the conflicts exist, the more chances we may have to schedule a larger number of the tasks. We use a two-step heuristic to solve the problem. In the first step, the tasks are scheduled to minimize the number of conflicts. Since this initial solution may not be feasible, the legalization is performed in the second step to get the best legal schedule in terms of the number of scheduled tasks.

```
Soft-Real-Time-Scheduling (Ψ)
/*step1: conflict minimization*/
while there exists unlocked task in Ψ
    compute the distribution graph
    for each unlocked task τ_i
        a-force(i)←self-force-arrival(τ_i)
        d-force(i)←self-force-deadline(τ_i)

    choose task τ_k which incurs the maximal force
    //shrink the time frame of τ_k
    if a-force(k)≥d-force(k)
        then Time-Frame(τ_k)←[a_k+1,d_k]
        else Time-Frame(τ_k)←[a_k,d_k-1]
    if |Time-Frame(τ_k)|=e_k
        then set τ_k locked
```

```
/*step2: legalization*/
sort all τ_i in the non-decreasing order of a_i
V←{v_1 .. v_n}
for i←1 to n
    for j←i+1 to n
        if all the predications hold
            then add an edge <v_i, v_j> into G
schedule T by traversing the longest path in G
```

Figure 5. Soft real-time scheduling algorithm

5.3.3 Conflict Minimization

In order to minimize the conflicts, we again apply the minimal freedom reduction technique to gradually shrink the time frames. The force concept is also employed to balance the conflicts along the available time slots. We only consider the self-force of relating a task τ_i to its arrival time and its deadline. They are denoted by *self-force-arrival(τ_i)* and *self-force-deadline(τ_i)*, respectively.

self-force-arrival(τ_i)=

$$\sum_{t=a_i}^{a_i+e_i-1} DG(t)\left(1-\frac{prob(\tau_i,t)}{tasks(t)}\right)+\sum_{t=a_i+e_i}^{d_i} DG(t)\left(0-\frac{prob(\tau_i,t)}{tasks(t)}\right)$$

self-force-dealine(τ_i)=

$$\sum_{t=d_i-e_i+1}^{d_i} DG(t)\left(1-\frac{prob(\tau_i,t)}{tasks(t)}\right)+\sum_{t=a_i}^{d_i+e_i} DG(t)\left(0-\frac{prob(\tau_i,t)}{tasks(t)}\right)$$

At each step, the assignment of a task to a time slot which results in highest force will be rejected, and the time frame of the corresponding task τ_i will be shrunk accordingly. When the size of a task's time frame is reduced to its execution time, this task will be locked (marked as *semi-scheduled*). The first step algorithm ends when all tasks are locked with the hope that they are evenly distributed. The first part of Figure 5 illustrates the algorithm.

5.3.4 Legalization

After the conflict minimization, all tasks have been *semi-scheduled*, namely, the time frame of each task is shrunk exactly to its execution time.

A compatibility graph G can be built from the initial solution. Define $G=(V, E)$ where $V=\{v_i|1\le i\le n\}$. An edge $<\tau_i,\tau_j>\in$E if and only if all the following three predications hold:

(1) $\neg conflict(\tau_i, \tau_j)$: τ_i *and* τ_j *are compatible*

(2) $a_i \le a_j$: τ_i arrives earlier than τ_j

(3) $\neg\exists k <v_i,v_k>\in E\wedge<v_k,v_j>\in E$: *edge* $<v_i, v_j>$ *is irredundant.*

It can be easily shown that G is a directed acyclic graph and we can get the best legal schedule by traversing the longest path in G. A task τ_i is scheduled in its time frame if v_i appears in the longest path, otherwise it is dropped. Note that the longest path in a DAG can be optimally found in linear time. The algorithm is illustrated in the second part of Figure 5.

5.4 Experimental Results

5.4.1 Maximum Independent Set

To evaluate our MIS algorithm, we take a set of DIMACS benchmark graphs for the *Clique* problem challenge. Since the maximum clique of a graph is the maximum independent set in the complementary graph, we complemented these benchmarks and apply the algorithm discussed in Section 5.1, as well as Kirovski's algorithm [14] for comparison. Our programs are implemented with C++/UNIX environment. Kirovski's program is the open source from his personal website. The experiments are run with a Sun Blade 1000 workstation with 750 MHz frequency.

The experimental results are shown in Table 4. The first column lists the name of each benchmark. The second and third columns are the corresponding vertex and edge numbers. In column Opt, we list the optimal solution if known. The next three columns are independent set sizes produced by three algorithms. Columns A and B are the results from our algorithm with the Level-1 and Level-2 heuristic respectively. Column C lists the results from Kirovski's iterative improvement algorithm.

In most of the cases, our Level-2 heuristic produced better solutions than the Level-1 heuristic. Compared with Kirovski's algorithm, our Level-2 algorithm achieves only marginally worse result but with a dramatic improvement in runtime. As shown in Table 5, on average, our Level-1 algorithm is more than 50 times faster than the Level-2 algorithm, which is more than 30 times faster than the iterative improvement algorithm.

5.4.2 Behavioral Synthesis Scheduling

Table 1. Scheduling results comparison under critical-path time constraint

	Node	Cycles	ALU	ALU*	MULT	MULT*
FFT	27	7	3	3	2	2
WANG	48	7	5	5	8	8
MCM	94	8	11	10	9	9
HONDA	97	7	8	9	14	14
DIR	148	9	10	10	9	8
CHEM_PLANT	347	16	15	13	13	13
AIRCRAFT_S	422	20	15	12	16	12
11_u5ml	497	17	20	17	22	17
12_u5ml	547	19	21	17	16	16
FEIG_DCT	548	69	9	7	3	2
GSM	617	41	10	9	6	6
AIRCRAFT_L	2283	35	43	36	48	35
Total	-	-	170	148	166	142
Improvement	-	-	-	13%	-	14%

We implemented both the force-directed algorithm and our gradual relaxation scheduling approach in C++/STL. The selected DFG benchmarks were taken from [23] with several DCT algorithms (WANG, DIR, FEIG_DCT), and a set of real-time

DSP programs. For the experimental results listed in Table 1, we used the critical path as the time constraint.

The second and the third columns list the size of DFG and its critical path length, respectively. The following columns list the number of required function unit ALU and MULT for different algorithms. Columns named ALU and MULT are the results from the force-directed algorithm, and columns ALU* and MULT* are from our algorithm.

For small examples (FFT and WANG), these two algorithms produced the same results. As the DFG size and parallelism became larger, our algorithm consistently outperformed the force-directed algorithm. On average, our algorithm achieved about 13% resource reduction compared with the force-directed algorithm.

Table 2. Scheduling results comparison under time constraint with 1.5 times of the critical path

	Node	Cycles	ALU	ALU*	MULT	MULT*
FFT	27	10	2	2	1	1
WANG	48	10	4	4	8	3
MCM	94	12	8	7	6	6
HONDA	97	10	8	6	7	7
DIR	148	13	8	7	6	6
CHEM_PLANT	347	24	12	8	8	9
AIRCRAFT_S	422	30	10	8	16	9
11_u5ml	497	25	15	11	14	12
12_u5ml	547	28	15	11	12	11
FEIG_DCT	548	103	8	5	4	2
GSM	617	60	7	6	3	3
AIRCRAFT_L	2283	52	33	23	31	26
Total	-	-	130	98	116	95
Improvement	-	-	-	25%	-	18%

Another set of experimental results in Table 2 is produced when we set the timing constraint as the 1.5 times of the critical path length. Our algorithm achieves 25% ALU reduction and 18% MULT reduction compared with the force-directed algorithm. This indicates that when we relax the time constraint, our gradual relaxation technique performs even better with more slack on each operation node.

5.4.3 Soft Real-Time Scheduling

Table 3. Soft real-time scheduling results

T	Task No.	Period	EDF.	No.	Busy	Util.
p1_ts9	20	56	8	12	48	0.8571
p2_ts9	20	56	7	9	49	0.8750
p3_ts9	20	56	5	6	53	0.9464
p4_ts9	20	56	4	6	47	0.8393
p1_ts14	25	145	7	10	119	0.8207
p2_ts14	25	145	7	12	128	0.8828
p3_ts14	25	145	10	18	115	0.7931
p4_ts14	25	145	6	9	124	0.8552
Average	22.5	100.5	6.75	10.25	85.38	0.8587

A set of benchmarks from [16] was applied to evaluate our soft real-time scheduling algorithm. As shown in Table 3, every cell in the first column is a task set executed in a particular processor. The second and third columns are the task numbers and scheduling periods. The fourth column lists resulting scheduled task numbers from the Earliest-Deadline First (EDF)[1] scheduling [24] for comparison. The next three columns list resulting scheduled task number (No.), processor busy time (Busy), and utilization ratio (Util.) from our algorithm. On average, our algorithm schedules 3.5 more tasks than EDF, which is being used in many real-time applications. Our algorithm achieved about 86% utilization ratio for this benchmark set.

[1] The full schedulability is not achievable in our testcases. Otherwise, EDF would generate optimal results.

6. CONCLUSIONS

We introduce a suite of new heuristic principles for design of algorithms for computational intractable synthesis tasks. The effectiveness of these principles is demonstrated on three important behavioral synthesis problems: maximal independent set, static time-constrained scheduling, and synthesis of soft real-time systems.

7. REFERENCES

[1] B. Adelberg, H. Garcia-Molina; and B. Kao, "Emulating Soft Real-Time Scheduling Using Traditional Operating System Schedulers," *in Proceedings of Real-Time Systems Symposium*, pp. 292-298, Dec. 1994.

[2] J. Bitner and E. Reingold, "Backtrack Programming Techniques," *Communications of the ACM,* vol. 18(11), pp. 651-655, Nov. 1975.

[3] D. Brelaz, "New Methods to Color the Vertices of a Graph," *Communications of the ACM,* vol. 22(4), pp. 251-256, Apr. 1979.

[4] J. Cong and Patrick H. Madden, "Performance Driven Global Routing for Standard Cell Design," *in Proceedings of International Symposium on Physical Design,* pp. 73-80, Apr. 1997.

[5] O. Coudert, "Exact Coloring of Real-Life Graphs is Easy," *in Proceedings of the 34th Design Automation Conference*, pp. 121-126, Jun. 1997.

[6] P. D'Argenio, J.-P. Katoen, and E. Brinksma, "Specification and Analysis of Soft Real-Time Systems: Quantity and Quality," *in Proceedings of the 20th IEEE Real-Time Systems Symposium*, pp. 104-114, Dec. 1999.

[7] R. I. Davis, K. W. Tindell, and A. Burns, "Scheduling Slack Time in Fixed Priority Pre-emptive Systems," *in Proceedings of Real-Time Systems Symposium*, pp. 222-231, Dec. 1993.

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

[9] M. Goldwasser, "Patience is a Virtue: The Effect of Slack on Competitiveness for Admission Control," *to appear in Journal of Scheduling*, 2003.

[10] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, "Optimization by Simulated Annealing: An Experimental Evaluation; Part 1, Graph Partitioning," *Operations Research*, vol. 37(6), pp. 865-893, 1989

[11] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, "Optimization by Simulated Annealing: Part II, Graph Coloring and Number Partitioning," *Operations Research*, vol. 39(3), pp. 378-406, 1991.

[12] M. Jones, D. Rosu, and M.-C. Rosu, "CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities," *in Proceedings of the 16th ACM Symposium on Operating System Principles*, vol. 31(5), pp. 198-211, Dec. 1997.

[13] B. Kao and H. Garcia-Molina, "Subtask Deadline Assignment for Complex Distributed Soft Real-Time Tasks," *in Proceedings of International Conference on Distributed Computing Systems*, pp. 172-181, Jun. 1994.

[14] D. Kirovski and M. Potkonjak, "Efficient Coloring of a Large Spectrum of Graphs," *in Proceedings of the 35th Design Automation Conference,* pp. 427-432, Jun. 1998.

[15] E. A. Lee and D. G. Messerschmitt, "Synchronous Data Flow," *in Proceedings of the IEEE*, vol. 75(9), pp. 1235-1245, Sep. 1987.

[16] C. Lee, M. Potkonjak, and W. H. Wolf, "Synthesis of Hard Real-Time Application Specific Systems," *Design Automation for Embedded Systems,* vol. 4(4), pp. 215-242, Oct. 1998.

[17] M. C. McFarland, A. C. Parker, and R. Camposano, "The High-Level Synthesis of Digital Systems," *in Proceedings of the IEEE*, vol. 78(2), pp. 301-318, Feb. 1990.

[18] G. D. Micheli, "Synthesis and Optimization of Digital Circuits," *McGraw-Hill*, 1994.

[19] C. Papadimitriou and K. Steiglitz, "Combinatorial Optimization: Algorithms and Complexity," *Prentice Hall,* 1982.

[20] P. G. Paulin and J. P. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASICs," in *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,* vol. 8(6), pp. 661-679, Jun. 1989.

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

[22] K. Richter, D. Ziegenbein, M. Jersak, and R. Ernst, "Model Composition for Scheduling Analysis in Platform Design," *in Proceedings of the 39th Design Automation Conference*, pp. 287-292, Jun. 2002.

[23] M. B. Srivastava and M. Potkonjak, "Optimum and Heuristic Transformation Techniques for Simultaneous Optimization of Latency and Throughput," *in IEEE Trans. on Very Large Scale Integration Systems,* vol. 3(1), pp. 2-19, Mar. 1995.

[24] J. A. Stankovic, M. Spuri, M. D. Natale, and G. C. Buttazzo, "Implications of Classical Scheduling Results for Real-Time Systems," *IEEE Computer,* vol. 28(6), pp. 16-25, Jun. 1995.

[25] J. S. Turner, "Almost All k-Colorable Graphs are Easy to Color," *Journal of Algorithms*, vol. 9, pp. 63-82, Mar. 1988.

[26] W. F. J. Verhaegh, P. E. R. Lippens, E. H. L. Aarts, J. H. M. Korst, J. L. van Meerbergen, and A. van der Werf, "Improved Force-Directed Scheduling in High-Throughput Digital Signal Processing," in *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,* vol. 14(8), pp. 945-960, Aug. 1995.

[27] D. Verkest, P. Yang, C. Wong, and P. Marchal, "Optimisation Problems for Dynamic Concurrent Task-Based Systems," *in Proceedings of the 2001 International Conference on Computer-Aided Design*, pp. 265-268, Nov. 2001.

[28] D. Ziegenbein, J. Uerpmann, and R. Ernst, "Dynamic Response Time Optimization for SDF Graphs," *in Proceedings of the 2000 International Conference on Computer-Aided Design*, pp. 135-141, Nov. 2000.

Table 4. Solution quality comparison of three MIS algorithms

Example	V	Edges	Opt	A	B	C
brock200_1	200	14834	?	18	19	20
c-fat200-1	200	1534	12	12	12	12
c-fat200-5	200	8473	58	58	58	58
c-fat500-1	500	4459	14	14	14	14
c-fat500-10	500	46627	126	126	126	126
c-fat500-5	500	23191	64	64	64	64
hamming10-2	1024	518656	512	512	512	512
hamming10-4	1024	434176	?	32	34	38
hamming6-2	64	1824	32	32	32	32
hamming8-2	256	31616	128	128	128	128
johnson32-2-4	496	107880	?	16	16	16
johnson8-4-4	70	1855	14	14	14	14
MANN_a27	378	70551	?	125	125	125
MANN_a45	1035	533115	?	342	342	341
MANN_a81	3321	5506380	?	1096	1096	1089
MANN_a9	45	918	?	16	16	16
p_hat1000-1	1000	122253	?	9	10	10
p_hat1000-2	1000	244799	?	43	46	43
p_hat1000-3	1000	371746	?	60	63	38
p_hat1500-2	1500	568960	?	58	61	65
p_hat1500-3	1500	847244	?	85	91	92
p_hat300-2	300	21928	25	24	25	23
p_hat300-3	300	33390	36	30	34	28
p_hat500-2	500	62946	36	34	36	32
p_hat500-3	500	93800	?	44	48	47
p_hat700-1	700	60999	?	9	9	9
p_hat700-3	700	183010	?	59	59	58
sanr200_0.7	200	13868	18	14	16	18
sanr200_0.9	200	17863	?	38	40	40
sanr400_0.5	400	39984	?	10	12	12
sanr400_0.7	400	55869	?	17	21	20

Table 5. Runtime comparison of three MIS algorithms (sec)

Example	V	Edges	Time A	Time B	Time C
brock200_1	200	14834	0	0.1	6.02
c-fat200-1	200	1534	0.01	0.36	87.95
c-fat200-5	200	8473	0.01	0.21	432
c-fat500-1	500	4459	0.04	5.95	334.22
c-fat500-10	500	46627	0.14	3.64	2627.11
c-fat500-5	500	23191	0.08	4.92	1491.21
hamming10-2	1024	518656	0.19	0.62	1545.38
hamming10-4	1024	434176	0.19	9.36	25.88
hamming6-2	64	1824	0	0	6.37
hamming8-2	256	31616	0.01	0.03	98.71
johnson32-2-4	496	107880	0.04	0.77	7.75
johnson8-4-4	70	1855	0	0.01	3.03
MANN_a27	378	70551	0.02	0.04	1.94
MANN_a45	1035	533115	0.17	0.33	4.38
MANN_a81	3321	5506380	2.11	3.8	13.59
MANN_a9	45	918	0	0	0.85
p_hat1000-1	1000	122253	0.18	38.22	98.42
p_hat1000-2	1000	244799	0.24	22.27	72.86
p_hat1000-3	1000	371746	0.23	11.12	31.36
p_hat1500-2	1500	568960	0.57	78.5	112.26
p_hat1500-3	1500	847244	0.54	38.64	114.88
p_hat300-2	300	21928	0.02	0.57	25.07
p_hat300-3	300	33390	0.01	0.29	8.54
p_hat500-2	500	62946	0.05	2.54	47.03
p_hat500-3	500	93800	0.05	1.29	27.92
p_hat700-1	700	60999	0.1	12.31	44.54
p_hat700-3	700	183010	0.11	3.59	42.07
sanr200_0.7	200	13868	0.01	0.11	7.81
sanr200_0.9	200	17863	0.01	0.04	12.34
sanr400_0.5	400	39984	0.03	1.56	38.03
sanr400_0.7	400	55869	0.03	0.94	13.29
Total	-	-	5.19	242.13	7382.81