Application of Constraint-Based Heuristics in Collaborative Design

Juan Antonio Carballo
IBM Austin Research Laboratory
11400 Burnet Road, Austin, TX 78758, USA
+1 (512) 838-8914
jantonio@us.ibm.com

Stephen W. Director
College of Engineering, University of Michigan
1221 Beal Avenue, Ann Arbor, MI 48109, USA
+1 (734) 647-7010
director@umich.edu

ABSTRACT

Significant acceleration of today's complex collaborative design processes can be achieved if team members are able to apply search heuristics that consider the simultaneous effect of all design constraints. We present the Active approach to Design Process Management (ADPM), whereby designers receive constraint-based feedback that enables them to apply these search heuristics effectively. To evaluate ADPM, we developed a design process evaluation environment called TeamSim. Evaluation results suggest that ADPM can reduce costly design iterations at the expense of extra, less costly, verification tool executions.

1. INTRODUCTION

Complex electronic designs are subject to ever tighter time-to-market requirements and thus involve ever larger teams, where multiple subsystems are developed in parallel by different groups. Unfortunately, this concurrent design often results in the late detection of conflicts involving multiple subsystems. These conflicts tend to be found upon system integration and thus are very costly to resolve. If one views the design as a group of variables subject to a set of constraints, conflicts can then be seen as violations of constraints. Late conflict detection occurs because, until system integration, each group of designers typically considers only a subset of all constraints relating their subsystem to other subsystems. We can substantially reduce costly rework by aiding consideration of the simultaneous effect of all constraints. The amount and variety of constraints makes computer support for this task essential. For this support to be most powerful, it must give designers direct instructions or "clues" to improve their design space search throughout the design process.

This paper presents *Active Design Process Management* (ADPM), a state-based design process model whereby team members receive constraint-based feedback on their operations and use it to apply design space search heuristics effectively. This guidance reduces and helps resolve conflicts. We also present *TeamSim*, a design process evaluation environment developed on top of the Minerva III design process manager [3] to evaluate ADPM.

Design can be viewed as a search process in a design space restricted by constraints. Constraint-based search heuristics can substantially improve search algorithms [2,6,9] and thus may significantly accelerate design convergence. Several types of constraint-based information can help effectively apply these heuristics, including:

- *Infeasible design subspaces*. The design process may be accelerated by focusing first on areas of the design space that have the smallest subspaces not found to be infeasible.
- *Strongly constrained subspaces.* Another heuristic is to focus first on design subspaces affected by the most constraints.
- *Efficient conflict resolution strategies.* Design convergence may also be accelerated by (a) making use of trade-offs produced by constraint margins to fix violations and (b) executing design operations that will fix many violations at a time.

While heuristics are often used by designers and CAD tools to search for design solutions, design environment work has not focused on providing the constraint-based guidance described above [4,7,8,10,11,12]. A key challenge is the complexity of the required constraint management infrastructure. A methodology called CCM [3] was introduced whereby this infrastructure is developed on the basis of constraint-based systems [1,9] and CAD tools. We leverage this work by computing constraint-related information using CCM's constraint generation and propagation techniques, and then "mining" the results into data that directly supports search heuristics (e.g., the number of violations related to each design variable). This heuristic support data accounts for the simultaneous effect of all constraints and thus may significantly reduce design iterations.

ADPM has been implemented in the Minerva III design process manager. However, measuring ADPM's value requires also quantitatively estimating its impact on design process performance and CAD resource consumption. Historically, prior design environment work has not been quantitatively evaluated[1]. We have addressed this issue by developing a design process simulation environment called *TeamSim* using Minerva III's infrastructure. Simulation facilitates controlling the evaluation process and enables a large number of evaluations. Designers are simulated by making the user interface automatically generate design operations. TeamSim can be configured to simulate a design process using either ADPM or conventional approaches and is customizable to any given application. TeamSim results were obtained for the design of a sensing system and the design of a MEMS-based wireless receiver.

Section 2 describes ADPM, Section 3 describes TeamSim and the evaluation results, and Section 4 draws conclusions.

2. ACTIVE DESIGN PROCESS MANAGEMENT

2.1. Background

ADPM is based on a design process modeling framework that is built on previous work [3,8,10] and emphasizes the role of constraints. In this framework, a design is characterized by a set of variables called properties. A design *property,* denoted by a_i, is a variable that can take one or more values from a range $E_i=\{v_j^i, j=1,...,N_i^V\}$. Values may be numbers, strings, tuples, or complex descriptions. A property a_i to which a single value has been assigned is said to be *bound*; otherwise, it is *unbound* with an implicit value of $a_i \equiv E_i$. The properties of a correct design must satisfy a set of constraints. A *design constraint* is a relation, c_i,

[1] Project data tracking systems [5] capture tool run data but do not provide control to evaluate high-level process and constraint management issues.

DAC 2001, Las Vegas, Nevada.

among a set of properties:

$$c_i(\boldsymbol{a}_i): S_i \rightarrow \{T, F\}, \quad (1)$$

where $\boldsymbol{a}_i=\{a_{i_j}, j=1,..., N_i^A\}$ denotes the *arguments* of c_i, and S_i denotes the cross-product of all possible argument values, i.e., the design subspace restricted by c_i. For example, constraint c_1, given by $P_f + P_s \le P_M$, relates a receiver circuit's power consumption requirement, P_M, its analog front-end power, P_f, and its digital deserializer power, P_s. A constraint c_i is said to be *satisfied* if it holds for all combinations of the current argument values; *violated* if it returns *False* for all combinations; and *consistent* otherwise. The *status* of c_i, denoted by $s(c_i)$, indicates whether c_i is satisfied ($s(c_i)=T$), violated ($s(c_i)=F$), or otherwise ($s(c_i)=Unknown$).

A *design problem*, denoted by p_i, is given by (I_i, O_i, T_i), where I_i is the set of *input properties*, O_i is the set of *output properties*, and $T_i = \{c_{i_j}, j=1,..., N_i^C\}$ is a set of constraints relating a subset of p_i's properties. A *solution* for p_i is an assignment for p_i's outputs that satisfies all constraints in T_i. Each problem has a *status* indicating its level of accomplishment (e.g., "Solved"). A *design operator*, denoted by f_j, is a function that helps solve a problem p_i by (a) computing values for p_i's outputs (*synthesis* and *optimization* operators), (b) verifying that a solution meets one or more constraints in T_i (*verification* operators), or (c) decomposing p_i into a partially-ordered subproblem set (*decomposition* operators). In practice, operators are typically implemented by CAD tools. An operator f_j may take one or more parameters, e.g., for a synthesis tool, a parameter may determine whether area or delay is optimized. A *design operation,* denoted by θ, is given by an operator f_j, a problem p_i to which f_j is applied, and f_j's parameter values.

A design process is a state-based system that goes through a series of design states. The *design process history* at stage *n* is given by $H_n=\{(<s_i, \theta_i>, i=1,..., n\text{-}1) \cup s_n\}$, where s_i and θ_i denote the *design process state* and the applied operation at stage *i*, respectively. Each s_i consists of: the *design object hierarchy,* i.e., the set of all *design objects* currently under design, where each object is a set of properties that represents a part of the design; the *design problem hierarchy*, i.e., the set of all formulated design problems; and the *network of constraints,* denoted by $C_i=\{c_{i_j}, j=1,..., N_i^C\}$, where N_i^C is the total number of design constraints. The *design space* at stage *n* is given by the cross product of all property value ranges in s_n. A *design transition,* denoted by t_n, is a pair of consecutive states (s_n, s_{n+1}). s_{n+1} results from applying the *next-state function*, δ, to s_n:

$$s_{n+1} = \delta(s_n, \theta_n), \quad (2)$$

where θ_n is the operation executed at stage *n*. The function δ applies θ_n's operator to a problem in s_n, and updates the state to s_{n+1}. δ's implementation depends on how the design process is managed.

2.2. The ADPM design process model

ADPM's transition model is graphically compared with conventional approaches in Fig. 1. For conventional approaches, the implementation of δ features a design process manager (DPM) component. In practice, the DPM connects the user with conventional CAD tools and may range from a raw OS interface to a complete process management system such as Minerva II [11]. The implementation of δ in ADPM adds a Design Constraint Manager (DCM) and a Notification Manager (NM). To address a problem p_i, a designer sends an operation request θ_n to the DPM, which takes as input θ_n and the previous state s_n. After applying θ_n's operator on p_i, the following tasks are undertaken:

- **Update of design state**. The DPM updates the problem hierarchy in s_n, including p_i, based on the operation result.

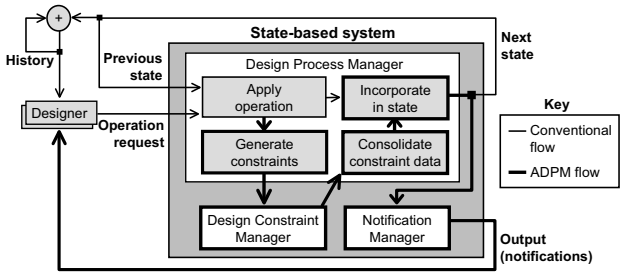

Fig. 1. Transition model for Active Design Process Management.

  However, unlike in conventional approaches, this DPM also generates any necessary constraints and incorporates them in C_n. The resulting C_{n+1}, including the current values of C_{n+1}'s properties, is then sent to the DCM for evaluation. The DCM then runs a constraint propagation algorithm to compute infeasible property values and the status of all constraints. Constraint evaluation details are delegated to constraint-based systems and CAD tools [3]. The result is sent back to the DPM, which properly updates C_{n+1} and the status of design problems. Constraint information is consolidated into data that explicitly supports heuristics (see Section 2.3), and the design state is properly labeled with this data. The new state s_{n+1} is included in the design history and made available to designers.

- **Communication of state information**. The NM alerts designers of constraint-related events, including violations and reductions of a property's feasible subspace. It selects subsets of H_{n+1} relevant to each designer and includes them in notifications. Notifications alert designers of key information that might otherwise go unnoticed, thereby encouraging them to use that information when choosing operations.

ADPM may require more computer resources than conventional approaches. While each CAD tool is executed only upon a designer's request in conventional approaches, additional tool runs are typically performed within ADPM's constraint propagation algorithm. This extra computation, though, allows ADPM to directly support constraint-based heuristic application. Key constraint-related information is automatically generated in a timely manner, and is organized to provide direct heuristic guidance. Notifications encourage designers to use the most relevant portions of this information when choosing an operation.

2.3. Constraint-based heuristic application support

ADPM directly supports constraint-based heuristics by virtue of several types of information. We describe some of these types next.

2.3.1 Heuristics based on feasible subspaces

For each property a_i, its *feasible subspace* $v_F(a_i)$ is given by the values that were not found to be infeasible by constraint evaluation. Feasible value information helps designers prune substantial design subspaces and thus quickly meet specifications. Design operations should be intended to bind problem outputs to values from their feasible subspace. Additionally, this information can help choose the order in which properties are bound. The following heuristic is supported: focus first on problems that target properties with the smallest feasible subspaces. By using this heuristic, it is expected that most violations happen early, since difficult subspaces are given priority. Similar variable ordering heuristics exist in constraint satisfaction algorithms [2,9].

2.3.2 Heuristics based on number of constraints

Another helpful heuristic based on existing constraint satisfaction

heuristics [6] is to execute operations that target properties connected to many constraints. It is intended to help focus first on very "constrained" properties. In ADPM designers can apply this heuristic as they receive information about (a) constraints involved in each design problem and (b) constraints where each property appears. To help apply this heuristic, we associate a variable, denoted by β_i, with each property a_i. β_i is the number of constraints where a_i appears: $\beta_i = |\{c_j \mid a_i \in \boldsymbol{a}_j\}|$. Extensions of this heuristic are possible. Specifically, β_i may also include constraints indirectly related to a_i by an intermediate constraint.

2.3.3 Heuristics based on constraint violations

Timely constraint violation information allows backtracking to start early. It can also be used as the basis of a heuristic for fixing violations; specifically, to modify values of properties connected to many violations. This heuristic may help resolve multiple conflicts with a single operation and thus exit the infeasible part of the design space fast. ADPM supports this heuristic by providing designers with the following information: (a) for each problem, all conflicts affecting any of its properties; and (b) for each property, all conflicts where the property is involved. To help apply this heuristic, we associate a variable, denoted by α_i, with each property a_i. α_i is the number of violated constraints where a_i appears:

$$\alpha_i = |\{c_j \mid (a_i \in \boldsymbol{a}_j) \wedge (s(c_j) = \mathrm{F})\}| \quad (3)$$

2.4. Constraint-based heuristics in Minerva III

We have implemented constraint-based heuristic support in the Minerva III design process manager [3]. We describe this support with an example: the team-based design of a MEMS-based wireless receiver front-end subject to gain, power, bandwidth, and frequency precision constraints. The example focuses on the concurrent design of (a) the low-noise amplifier (LNA) and mixer and (b) a MEMS filtering device. The team includes a leader, a device engineer, and an analog circuit designer. (Although ADPM is envisioned for use by larger teams, this example is large enough to highlight the differences between ADPM and traditional approaches.) Using Minerva III's user interface, the leader defines a top-level system design problem, and decomposes it into the analog portion and the MEMS filter. The device engineer, who is assigned to work on the filter, starts by focusing on its required center frequency. Since this frequency is determined by the device's beam length, the engineer adjusts this length to 13 μm and then completes an initial version of the filter.

2.4.1 Using feedback about infeasible design subspaces

Minerva III provides information that clarifies the impact of the device engineer's operation on the analog portion of the design. Using Minerva III's object browser (see Fig. 2), the circuit designer can view property values not found to be infeasible (including design variables and performance parameters), related to his LNA and mixer. This feature helps choose operations that bind problem outputs to values from their feasible subspace. It also supports a heuristic: to focus first on properties with the smallest feasible subspaces. As Fig. 2 shows, all values for the frequency inductor property ("Freq-ind") are infeasible except for the interval (0.17, 0.5) μH. This value set is small when compared with the feasible set for the differential pair width property ("Diff-pair-W")[1], which encourages the circuit designer to focus on the inductor design first.

2.4.2 Using feedback about constrained subspaces

Before committing to a design operation, the designer considers

[1] Note: value set size is unit-dependent and subject to designer judgement.

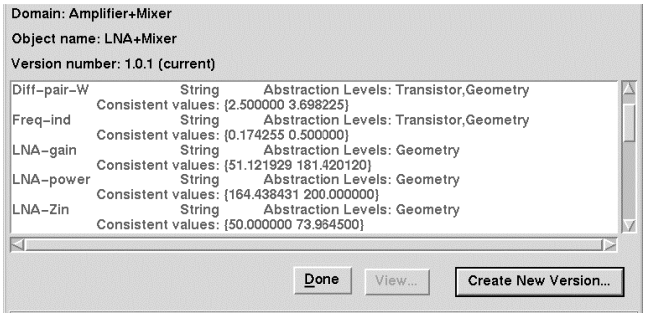

Fig. 2. Subspaces not found to be infeasible (circuit designer view)

other constraint-related information. Using Minerva III's constraint and property browser (see Fig. 3), the designer views in what constraints each property appears. This information supports another heuristic: to give priority to properties that appear in many constraints. As the PROPERTIES pane shows, the differential pair width property ("Diff-pair-W") appears in 3 constraints: power consumption, input impedance, and gain. Thus $\beta_2 = 3$, where β_2 is the number of constraints where this property appears.

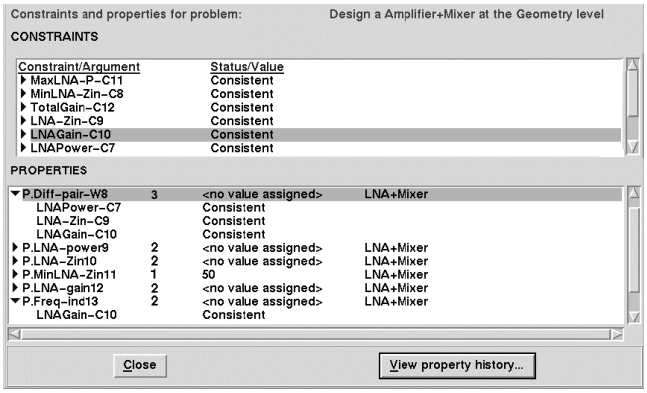

Fig. 3. Properties and their related constraints (circuit designer view)

The designer uses the constraint-related information shown in Fig. 2 and Fig. 3 when working on the LNA. Of the many tasks on which the designer could focus, two are suggested as important by this information. The designer first focuses on the design of the load inductor, because its feasible value set is very small. By invoking a schematic editor from Minerva III a value of 0.2 μH is chosen, which does not result in any detected conflict. The differential pair transistors are then sized. A size of 2.5 μm is chosen because it is the smallest potentially feasible value (see Fig. 2), and will reduce power consumption. Unfortunately, the chosen values lead to a violation of the global gain requirement, which concerns both the circuit designer and the device engineer. The team leader worsens the situation by tightening the input impedance requirement to 40 Ω, which leads to an impedance violation as well.

2.4.3 Using feedback for conflict resolution

The designer invokes the constraint and property browser again to try to resolve these conflicts (see Fig. 4). In this case, the number of violations related to each property is examined, shown in the "Connected violations" column of the PROPERTIES window pane. This information supports another heuristic: to backtrack on a property connected to many violations. Based on this heuristic, the designer chooses to work on the differential pair width, as this property is connected to two violations, i.e., $\alpha_2 = 2$. Since larger transistors will improve gain and input impedance matching, the

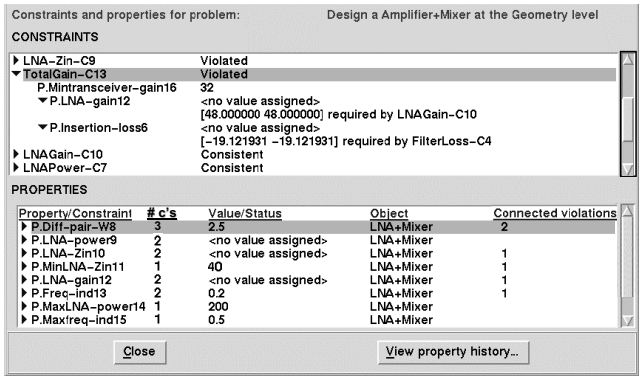

Fig. 4. Minerva III's support for circuit designer to resolve conflicts.

designer decides to increase the value of the differential pair width to 3.5 μm. Constraint propagation is run again and no conflicts are found. Both violations have been fixed with a single iteration.

3. EVALUATION RESULTS

3.1. Design process evaluation with *TeamSim*

TeamSim is a simulator whose architecture (see Fig. 5) extends the Minerva III architecture by incorporating the following features:

- **Simulation statistics capture.** A simulation engine within the DPM dynamically captures, stores, and consolidates simulation statistics for on-line visualization and post-simulation analysis.
- **Simulation of designers.** A simulated designer engine sits within each client and is integrated with its graphical user interface. This engine can automatically react to server feedback by requesting design operations.
- **Visualization of simulation statistics.** A graphical interface dynamically displays the captured simulation statistics. It includes a constraint network viewer and a statistics viewer that shows constraint statistics and value assignments. This interface was built by connecting Minerva III with existing visualization programs (Gnuplot and Lefty). Minerva III's interactive windows can also be viewed and used during simulations.

3.1.1 Designer model

Simulating designers requires a designer model that emulates an engineer's view of the design, as derived from the information received, and the choices made based on this information. Our model satisfies this requirement (see Fig. 6). A designer is viewed

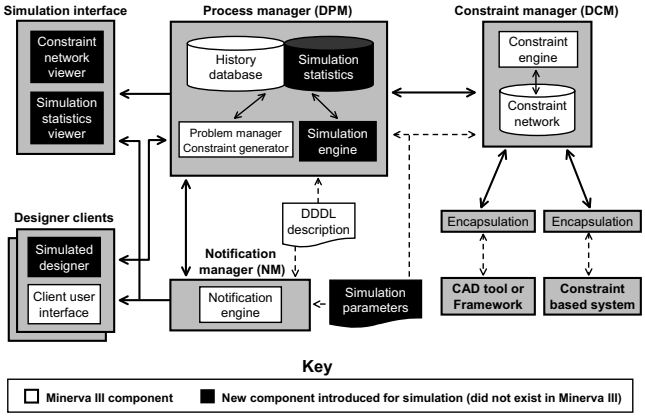

Fig. 5. Architecture of the TeamSim evaluation environment.

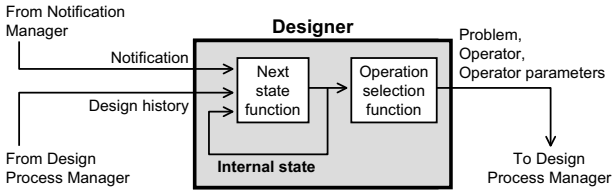

Fig. 6. Simulated designer model used to evaluate ADPM.

as a state-based system whose goal is to solve design problems. The designer has an internal view of the design, the *internal state,* based on the information received from the DPM and NM. The designer must select a problem to address, a set of outputs to which values are assigned, and the values themselves. The internal state includes the necessary information to make this selection, including:

- **Design problems.** A list of problems assigned to the designer, including their status information, outputs and current state.
- **Constraint information used in search heuristics.** For each property a_i that a designer is concerned with, the internal state contains its feasible values $v_F(a_i)$, the number of connected constraints β_i, the number of connected violations α_i, a list of constraints monotonically increasing in a_i, and a list of constraints monotonically decreasing in a_i [1].

The internal state is updated based on the current state itself and the messages received from the DPM and the NM. When a new message arrives, a *next-state function* updates the state. The process whereby each designer d_i chooses an operation to be executed can be seen as the application of a function f_o, called the *operation selection function,* on the internal state. f_o can be viewed as the composition of three functions, f_p, f_a, and f_v:

- **Problem selection function** (f_p)**.** This function selects all problems assigned to d_i that do not have a *Waiting* status. If d_i knows of no violations based on the internal state, and all problems assigned to d_i are solved, an empty set is returned.
- **Target property selection function** (f_a)**.** This function selects an output property from all addressable problems based on the following heuristics (ties are resolved randomly):
  - *Focus on most difficult subspaces*. If problems with unbound outputs exist and no constraint violations are known of, d_i chooses the output with the smallest feasible subspace.
  - *Focus on properties that enable efficient conflict resolution.* If there are violations, a property is selected for which a value modification is likely to fix many violations. The number of violated constraints related to each property is examined and preference is given to properties involved in many violations. For violated monotonic constraints, we also determine what direction of value change is likely to fix most violations. If moving the value of a property a_i in a given direction is likely to fix many violations, then a_i is given preference.
  - *Complete design.* If all problems are complete and no violations are known of, an empty property set is returned.
- **Value selection function** (f_v)**.** This function chooses a value for a selected property a_i based on the following heuristics[2]:
  - *Choose from feasible subspace.* If $v_F(a_i) \neq \varnothing$, a value is chosen from $v_F(a_i)$. For ordered value sets, we choose the top or bottom value based on what may satisfy most constraints.

[1] A constraint c_i is monotonic in a_i if moving a_i's value in a given direction helps satisfy the design requirement implied by c_i.

[2] While using these heuristics, the design history is consulted to avoid combinations of assignments that have previously led to violations.

- *Choose from initial subspace*. If $v_F(a_i)=\varnothing$, the value is chosen from the initial range E_i. For bound properties with ordered value sets, we increase or decrease the current value based on what direction is likely to fix the most violations. The size of this delta (increase or decrease) can be controlled in software by a parameter. In our simulations, delta values around 100 times smaller than the size of E_i worked well.

3.1.2 Collection and visualization of simulation data

Each simulation has an initial *problem scenario* given by a top-level problem formulation, an initial decomposition into subproblems; a set of designers, an assignment of subproblems to designers, and initial values for top-level requirements. A script automatically initializes this scenario, and designers start requesting operations independently. A simulation terminates when the top-level problem is solved (and thus all of its subproblems are too), all problem outputs have a value, and no constraints are violated.

TeamSim is configured for the scenario's design area using the DDDL language [3,10]. Types of properties, constraints, problems, decompositions, ordering among design problems, and constraint monotonicity can be specified. For example, the DDDL code below states that filter loss constraints are monotonic decreasing in the resonator length, but are monotonic increasing in the beam width:

```
constraint "Filter-loss-constraint" {
    arguments {
        select {
            "Insertion-loss";
            "Resonator-length"; decrease to satisfy;
            "Beam-width"; increase to satisfy;
        }
    }...
}
```

ADPM can be compared with conventional approaches by setting a Boolean parameter. When λ=F, the conventional approach is simulated. Constraint propagation is not run. Simulated designers can know of constraint violations and infeasible values only by requesting verification operations (e.g., simulations). Verification operators are executed only when their inputs are bound, typically when a subsystem is complete. Constraints relating multiple subproblems are evaluated only when all subproblems involved are solved and no internal constraints are violated. When λ=T, ADPM is simulated. Constraint propagation is run, and the simulated designers can make use of timely constraint-related information. Constraints relating multiple subproblems are propagated beginning when these constraints are generated.

Upon the execution of a design operation θ, TeamSim captures and displays the number of constraint violations found immediately after θ's execution, the number of constraint evaluations executed due to θ, the cumulative number of executed operations (including θ), and the value assignments done as a result of θ. Fig. 7 depicts a typical profile for a simplified design case showing two key metrics as a function of operation number: number of constraint violations, and number of constraint evaluations. Fig. 7 (a) shows the number of violations found upon each executed operation. The solid line corresponds to a simulation run with the new ADPM features, including constraint propagation, turned off. The dotted curve corresponds to a simulation run with all features turned on. Observe that using ADPM a smaller number of violations is found, violations start later, and violations stop happening earlier (and thus fewer design operations are required to complete the design). These trends can be explained by ADPM's constraint-based heuristic support. This support supplies a timely, precise view of the feasible design space that helps quickly place the design in subspaces that

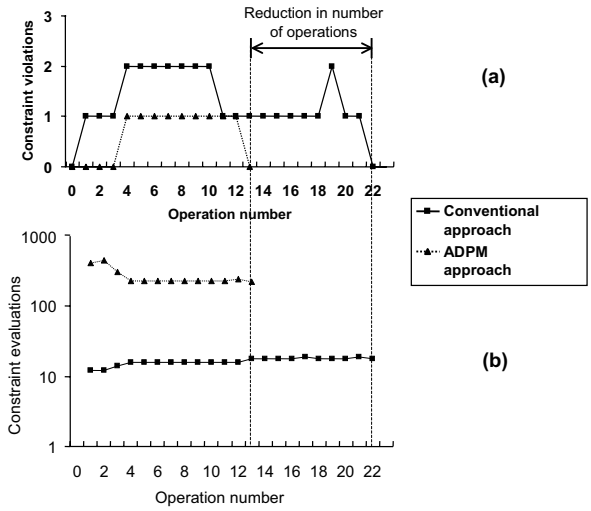

Fig. 7. Typical simulation statistics in TeamSim.

are likely to be feasible. However, as Fig. 7 (b) shows, ADPM requires more constraint evaluations (i.e., more tool runs) per executed operation in exchange for the reduced number of human designer operations. In terms of the *total* number of constraint evaluations, though, ADPM presents a smaller penalty. The total number of evaluations, denoted by N_T, is given by $N_T = N_E \times N_O$, where N_E denotes the average number of evaluations per design operation, and N_O denotes the total number of executed operations. The total penalty is smaller than the per-operation penalty, because the number of evaluations is given by the area under the two curves in Fig. 7 (b) and the number of operations is smaller using ADPM.

Fig. 8 shows TeamSim's design process statistics window. Key statistics are dynamically displayed, including the number of constraints, the number of violations, the number of constraint evaluations, and the cumulative number of design *spins*. A design spin is an executed operation due to at least one violation involving properties from multiple subsystems. Spins can be viewed as expensive design iterations performed upon system integration.

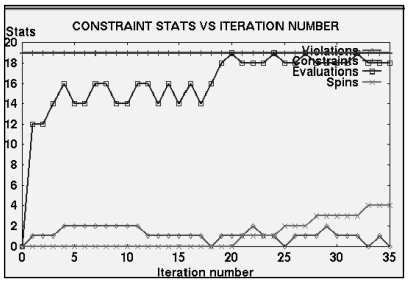

Fig. 8. Snapshot of design process statistics window in TeamSim.

3.2. Simulation results

We simulated two different design cases. The first case is the design of a MEMS-based pressure sensing system, composed of a capacitive pressure sensor and a mixed-signal interface circuit that are designed concurrently. This case includes top-level constraints on sensing resolution, estimated yield, and achievable pressure range. During simulations, the entire network contains up to 26 properties and 21 constraints, most of them linear and monotonic. The second case is the design of a MEMS-based wireless receiver front-end, composed of mixed-signal circuitry and a MEMS-based

channel-selection filter that are designed concurrently. This case includes constraints on channel bandwidth, system gain, input impedance, frequency selection precision, and power consumption. During simulations, up to 35 properties and 30 constraints exist, most of which are non-linear. Thus this case can be viewed as "harder" than the sensing system case. Although relatively small, both simulated cases present multiple complex design trade-offs among the decisions of different team members and thus emulate real multidisciplinary design processes.

Design process performance was estimated by examining the number of design operations required to complete each case. Over 60 simulations were executed varying the value of the random seed. As Fig. 9 (a) shows, at least twice as many operations on average were required to complete the designs using the conventional approach compared to ADPM. Since each operation requires a direct request from a designer, this result suggests that ADPM may reduce expensive designer effort, thereby improving designer productivity. The reduction in the number of operations is more significant for the receiver problem. This result can be supported by ADPM's constraint-based guidance. The harder a design problem, the more difficult it is to make "guesses" about values that should be assigned to problem outputs, and thus the more helpful ADPM's guidance should be. Further analysis showed that the average number of spins performed using ADPM was 7% of the number of spins performed using the conventional approach. This data suggests that ADPM may significantly reduce late design iterations. Finally, we estimated design process predictability by comparing the variability or standard deviation of the number of design operations. ADPM's results were at least 3 times less variable (see Fig. 9 (a)). Assuming that variability implies predictability, this data suggests that ADPM's guidance may result in a more predictable design process.

ADPM's computational penalty was estimated by examining the number of executed constraint evaluations, which provides an estimation of the number of times that verification tools, simulation tools, and constraint-based systems are run. Although this metric does not directly account for differences among tools, it supports an adequate *comparison* between the conventional approach and ADPM. As Fig. 9 (b) shows, the average number of evaluations required by ADPM in our simulations was much higher than those required by the conventional approach. This result indicates that ADPM may require a substantial extra computational effort due to its constraint propagation algorithm. While this algorithm's worst-case complexity is at least polynomial in the number of constraints and variables [3], the conventional approach is at most linear in the number of constraints. The computational penalty is smaller for the wireless receiver problem. This result seems reasonable as the

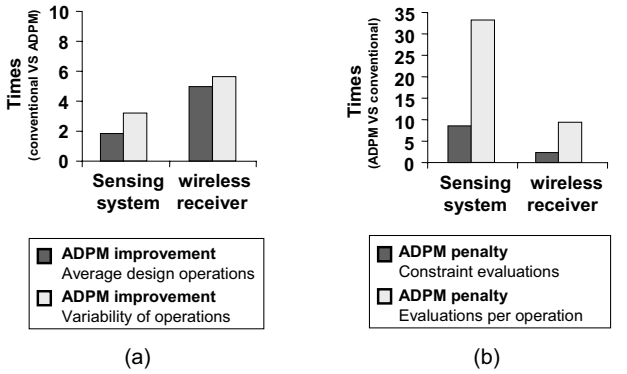

Fig. 9. Results for (a) design operations and (b) constraint evaluations.

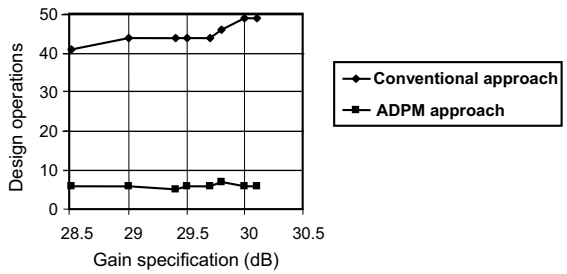

Fig. 10. Variation of design operations with specification tightness.

harder a problem, the greater the likely improvement in the number of operations provided by ADPM, and thus the smaller the total computational penalty. Finally, as Fig. 9 (b) shows, the average number of evaluations per executed operation reflects a larger penalty than the penalty given by the total number of evaluations. This difference is consistent with Fig. 7 (b) and in practice may require purchasing extra computational infrastructure.

To examine ADPM's robustness with respect to problem hardness, we swept the tightness of top-level requirements. Fig. 10 shows the variation in the number of executed operations with the tightness of the gain requirement in the receiver problem. This variation appears to be larger when using the conventional approach, which suggests that the new ADPM approach is more robust.

4. CONCLUSIONS

Current design teams suffer ever tighter staffing and time-to-market requirements. Our quantitative results suggest that ADPM's constraint-based heuristic support may improve productivity and predictability. These improvements come at the expense of a computational cost penalty. Fortunately, for more complex design problems ADPM may provide a more substantial design process acceleration for a proportionally smaller computational penalty. Future work should evaluate other types of problems and heuristics.

5. ACKNOWLEDGEMENTS

The authors are grateful to the reviewers and to Anne Gattiker for their helpful comments. This work has been funded in part by a scholarship from Spain's Science and Education Department.

6. REFERENCES

[1] C. Bessiere and J. Regin, "Arc consistency for general constraint networks: preliminary results", *Proc. IJCAI'97*: 398-404.

[2] J. Bitner and E. Reingold. Backtrack programming techniques. *Communications of the ACM*, (18):651-655.

[3] J.A. Carballo and S. Director, "Constraint Management for Collaborative Electronic Design", *Proc. 36th DAC*, June 1999.

[4] F.L. Chan, M.D. Spiller, and A.R. Newton, "Weld - an environment for web-based electronic design", *Proc. 35th DAC*, June 1998.

[5] S. Fenstermarker et al., "METRICS, A System Architecture for Design Process Optimization", *Proc. 37th DAC*: 705-710, June 2000.

[6] E. Freuder and M. Quinn, "Taking advantage of stable sets of variables in constraint satisfaction problems." In *Proc. IJCAI*, 1076-1078, 1985.

[7] S.T. Frezza, S.P. Levitan, and P.C. Chrysanthis, "Requirements-based Design Evaluation", *Proc. 32nd DAC*, June 1995.

[8] M. Jacome and S. Director, "A formal basis for design process planning and management", *IEEE Trans. CAD,* 15(10):1197–1211, Oct. 1996.

[9] V. Kumar, "Algorithms for Constraint Satisfaction", *AI Magazine*, 13(1):32–44, 1992.

[10] P. R. Sutton and S. W. Director, "A Description Language for Design Process Management", *Proc. 33rd DAC*, 1996.

[11] P. R. Sutton and S. W. Director, "Framework Encapsulations: A New Approach to CAD Tool Interoperability", *Proc. 35th DAC*, June 1998.

[12] K.O. ten Bosch et al., "Design Flow Management in the Nelsis CAD Framework", *Proc. 28th DAC*: 711–716, June 1991.