Adaptation Using Constraint Satisfaction Techniques[1]

Lisa Purvis

Dept. of Computer Science & Eng.
University of Connecticut
U-155, Storrs, CT 06269
Fax: 203-486-4817
Phone: 203-230-8777
E-mail: purvis@cse.uconn.edu

Pearl Pu
Laboratoire d'Intelligence Artificielle & Robotique
Institut de Microtechnique / DMT
Swiss Federal Institute of Technology (EPFL)
MT-Ecublens, 1015 Lausanne, Switzerland
Fax: 42-21-693-3866
Phone: 41-21-693-6081
E-mail: pu@lia.di.epfl.ch

Abstract. Case adaptation, a central component of case-based reasoning, is often considered to be the most difficult part of a case-based reasoning system. The difficulties arise from the fact that adaptation often does not converge, especially if it is not done in a systematic way. This problem, sometimes termed the assimilation problem, is especially pronounced in the case-based design problem solving domain where a large set of constraints and features are processed. Furthermore, in the design domain, multiple cases must be considered in conjunction in order to solve the new problem, resulting in the difficulty of how to efficiently combine the cases into a global solution for the new problem.

In order to achieve case combination, we investigate a methodology which formalizes the process using constraint satisfaction techniques. We represent each case as a primitive constraint satisfaction problem (CSP) with additional knowledge that facilitates retrieving, and apply an existing repair-based CSP algorithm to combine these primitive CSPs into a globally consistent solution for the new problem. The run time is satisfactory for providing a quick and explicable answer to whether existing cases can be adapted or if new cases would have to be created.

We have tested our methodology in the configuration design and assembly sequence generation domains. Analysis of performance and results will be shown at the end of this paper.

1 Introduction

The domain of case-based reasoning (CBR) has received much attention as a viable and natural problem solving methodology for design, because the complexities of this domain often require past design experience in order to create effective new solutions. This past experience generally must be adapted to fit the new situation, since it is rare that an existing case exactly matches the demands of a new problem. Moreover, it is not enough to simply find one old case that is similar to a new situation and adapt from there. It is more likely that multiple cases contribute information that is necessary for solving the new problem.

The solution to a new problem, then, results from merging the local solutions from previously solved problems to create a globally consistent solution for the new problem. However, the merging process is difficult since the local solutions typically exhibit conflicts when merged together. Furthermore, local solutions can be characterized by different representations, further intensifying the difficulty of synthesizing the global solution in an ad hoc way.

To overcome these problems, we investigate a methodology which formalizes the adaptation process using constraint satisfaction techniques. In our framework, each existing case is represented and stored in the case base as a solution to a primitive constraint satisfaction problem (CSP) with additional knowledge which facilitates retrieval and matching. The solutions to the primitive CSPs are combined into a globally consistent solution for the new problem using a repair-based CSP algorithm.

Our previous work[1] showed computational advantages over traditional methods when this case-based adaptation methodology is used to solve assembly sequence problems (ASP). The work described here further expands the applicability of the method to a larger class of problems by incorporating dynamic constraints into the CSP formulation and repair algorithm. This allows our adaptation process to be applied not only to static CSP's, but to any problem which can be described as either a static or dynamic, discrete CSP. This generalized formalism for case adaptation helps CBR systems achieve broader applicability and a better efficiency.

The rest of the paper is organized as follows: Section 2 reviews related work on case adaptation, Section 3 details the CSP formulation of cases, Section 4 describes the adaptation process itself, followed by current results in Section 5, and concludes with a summary and review of the key issues in Section 6.

[1] This research is sponsored by the National Science Foundation grant IRI - 9208429.

2 Related Work

Adaptation can be described as the process of changing an old solution to meet the demands of a new situation[2]. Three important, well known adaptation methods are substitution, transformation, and derivational analogy methods. Substitution methods are used in the existing case-based design aids CHEF[3], JUDGE[4], CLAVIER[5], and MEDIATOR[6]. These methods choose and install a replacement for some part of an old solution that does not fit the current situation requirements. Transformation methods use heuristics to replace, delete, or add components to an old solution in order to make the old solution work in the new situation. Transformation methods can be found in the case-based system CASEY[7], in which transformation is guided by a causal model, and JULIA[8], which uses common sense transformation heuristics to fix the old solution for the new problem context. Derivational analogy methods, found in ARIES[9] and PRODIGY/ANALOGY[10], use the *method* of deriving the old solution in order to derive a solution in the new situation.

Recently researchers have become interested in applying CBR techniques to the design domain[11]. Components of a design case often have strong relations among one another[2], and also tend to be large and therefore need to be decomposed to facilitate reuse[12]. Maher and Zhang[13] use an integration of case transformation and derivational analogies to tackle the adaptation problem in their system, CADSYN. Hua and Faltings combine multiple cases to achieve adaptation in their system, CADRE[14], where they observe that changing one feature during the adaptation process may result in non-convergent behavior for the adaptation algorithm.

3 CSP Formulation of Existing Cases

In our framework, existing cases are stored as primitive CSPs. A CSP consists of variables, values, and constraints. These CSP components are represented in a case as feature-value pairs. Along with the CSP variables, values, and constraints, a case also includes other characteristics in order to distinguish it during matching. Our cases allow not only static, but also dynamic constraints to be represented. Dynamic constraints are important in complex domains such as design, where the set of problem variables changes dynamically in response to decisions made during the course of problem solving.

Research on dynamic CSP can be found in the work of Bessiere[15], which describes an algorithm for computing arc-consistency for dynamic constraint satisfaction problems, Faltings[16], which explores dynamic constraint propagation in continuous domains, and Mittal and Falkenhainer[17], which identifies four types of dynamic constraints and implements them within an ATMS framework. Our research has implemented all four types of dynamic constraints identified in Mittal's work[17] within the minimum conflicts CSP algorithm in order to extend the applicability of our adaptation methdology to all problems which can be described as either static *or dynamic* CSP's.

4 Adaptation

Adaptation is the process of changing an existing solution to fit the new context, thereby solving a new problem. The difficulties of doing adaptation in complex domains such as design prohibit its wide use in CBR systems. We attempt to formalize the adaptation process using CSP techniques. The general methodology for our approach is illustrated by Figure 1.

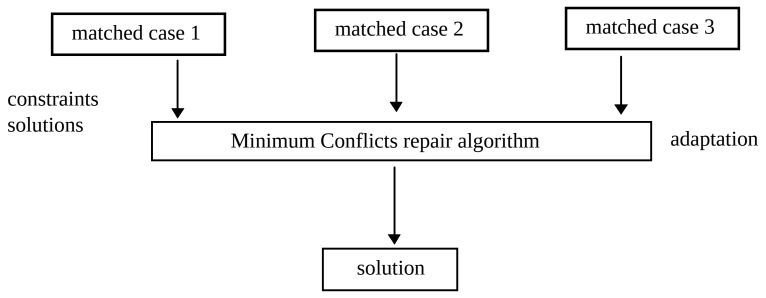

Figure 1: **Adaptation Methodology**

Cases which match portions of the new problem are retrieved from the case base, by first doing a structure mapping[18] using the spatial and geometrical features of each case to determine correspondences between variables of the old and new case, and then applying the nearest-neighbor similarity metric[2] to compute which of the structural matches are most similar. The matching cases contribute their constraints and solutions to form the new problem into a CSP, which is subsequently adapted using the minimum-conflicts adaptation algorithm to find a solution for the new problem.

The important pieces of information that are contributed by the existing cases are the old solutions and the constraints. Once a case has been identified as a match with the new problem, its variables and values can be used to initialize the corresponding variables and values in the new case, thus providing guidance from past experience to help achieve a better CSP efficiency. In addition, the newly formed CSP obtains its constraints from the matched cases, thereby eliminating reliance on user input to determine the constraints, and reducing the significant computational burden necessary in other approaches in order to calculate the constraints from first principles. For dynamic design problems such as configuration design, the old cases also contribute the essential information about what variables compose the new problem; information that would not be available without the existing cases.

To illustrate in detail how the new problem is set up as a CSP, let us consider the assembly sequence problem of the motor shown in Figure 2.

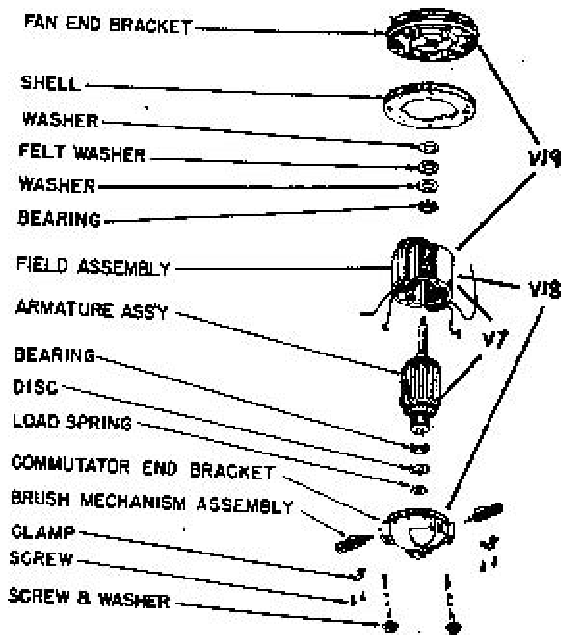

Figure 2: Motor Assembly[19]

We will look at the correspondence between a subassembly of this new problem and the existing case for the receptacle shown in Figure 3. The constraints for the receptacle are such that the stick must be placed inside the receptacle either before the handle or before the cap is attached to the receptacle, otherwise there is no geometrically feasible way to insert the stick into the receptacle. This same principle can be found in a subassembly of the motor case, in that the armature must be placed inside the field assembly before either the fan end bracket or before the commutator end bracket is attached to the field assembly, otherwise there is no geometrically feasible way to insert the armature into the field assembly. This correspondence is found in our system by doing a structure mapping on the mating-relationship feature-values of the old and new case.

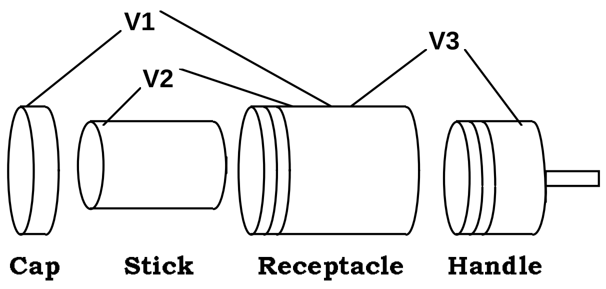

Figure 3:Receptacle

From the structure mapping, we find the following correspondences between the old and new case:

HANDLE ---> COMMUTATOR-END-BRACKET
CAP ---> FAN-END-BRACKET
RECEPTACLE ---> FIELD-ASSEMBLY
STICK ---> ARMATURE
V3 ---> V18
V1 ---> V19
V2 ---> V7

Now, the nearest neighbor similarity metric is applied using the correspondence information and the features' weights to determine whether the match is close enough to warrant using the old case as part of our new CSP. At this point, *all* of the case's features are considered, not just the mating-relationship features used during the structure mapping process. The more detailed case features allow a more accurate similarity score to be computed. In this example, additional case features of the receptacle case are:

(RECEPTACLE OPEN-CYLIDER 10)
(CAP BLIND-CYLINDER 10)
(HANDLE BLIND-CYLINDER 10)

Note that each of these detailed features has a weight of 10, out of a possible range from 1 to 10, indicating that each has significant importance to the case. In this example, the computed similarity measurement is large enough to warrant that the old case be used for the new CSP.

Now that we have found that the receptacle case is a match with part of our new problem, we take its solution and its constraints to set up the new CSP. The correspondences found provide the mapping information between old and new case.

Old case's solution: (V1 1) (V2 2) (V3 3)
Old case's constraint: (CONSTRAINT (OR (< V2 V3) (< V2 V1)))

Since V3 from the old case corresponds to V18 from the new case, the initial value for V18 in the new problem is set to 3, which was V3's value in the old case. Similarly, the constraints for the new case are obtained by substituting the new variables for their corresponding variables in the old case's constraints.

(CONSTRAINT (OR (< V7 V18) (< V7 V19)))
(V19 1)
(V7 2)
(V18 3)

In this manner, each matching case contributes its constraints and solutions to the new CSP. The minimum conflicts algorithm is then applied to synthesize all of the primitive solutions into one globally consistent solution for the new problem. The minimum conflicts repair algorithm is illustrated by Figure 4. The initial solution, made up of the primitive solutions found in the matching cases, is the starting point for the algorithm. A value that violates some of its constraints is chosen for repair from this initial solution, and repaired by finding a value for the variable which conflicts the least with the remaining values. Conflicting variable values continue to be repaired until there are no more conflicts, at which point we have found a solution to the new problem.

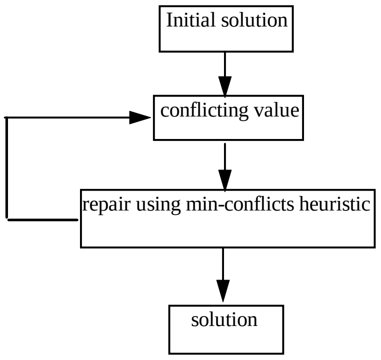

Figure 4**: Minimum Conflicts Repair Algorithm**

The empirical results for the algorithm showed that since the number of required repairs remains approximately constant as n grows, the algorithm's empirical time is approximately linear[20], as opposed to the exponential complexity of traditional constructive backtracking techniques. The effectiveness of the algorithm stems from using information about the current assignment to guide the search that is not available to standard backtracking algorithms. Our methodology capitalizes on the efficiency of the algorithm by providing it with a good initial solution based on the already solved cases in the case base.

Additional flexibility in our approach comes from the incorporation of *dynamic constraints* into the adaptation algorithm. We have modified the minimum conflicts algorithm in order to allow four types of dynamic activity constraints described in Mittal[17]. When a value is chosen for a variable, or a new variable becomes active in the problem, the activity constraints are tested using a forward-checking mechanism to identify any additionally required variables, and to eliminate any variables that are no longer required. This added flexibility has not only made the method more widely applicable, but also reduces the search space, and therefore provides a better efficiency from the minimum conflicts algorithm, as we will show in section 5 when detailing our results.

Let us now examine how a configuration design problem can be solved using our adaptation methodology. We will ne the car configuration domain detailed in Mittal[17]. Consider two existing car configurations, represented in the case base as shown in Table 1.

FEATURES	***CASE #1***	***CASE #2***
MODEL	model-80	model-70
STATUS	luxury	standard
FUEL-EFF	high	medium
BODY	convertible	hatchback
ENGINE	large	small
BATTERY	large	large
CONVERTER	cv1	
AIRCOND	ac2	
CD-PLAYER	sony	
DOORS		dr222
INSTRUMENT-PANEL		ip228
STEREO		st2
SEATS		s536
CONSTRAINT C1	(and (MODEL = model-80) (FUEL-EFF = high))	(and (BATTERY = small) (ENGINE = small) (RN CONVERTER))
CONSTRAINT C2	(and (MODEL = model-80) (RV BODY))	(and (STATUS = standard) (BODY ≠ convertible))
CONSTRAINT C3	(and (MODEL = model-80) (RV ENGINE))	(and (MODEL = model-70) (RV BODY))
CONSTRAINT C4	(and (MODEL = model-80) (RV BATTERY))	(and (MODEL = model-70) (RV ENGINE))
CONSTRAINT C5	(and (STATUS = luxury)	(and (MODEL = model-70)

	(RV AIRCOND))	(RV BATTERY))
CONSTRAINT C6	(and (STATUS = luxury)	(and (MODEL = model-70)
	(RV CD-PLAYER))	(RV DOORS))
CONSTRAINT C7	(and (FUEL-EFF = high)	(and (MODEL = model-70)
	(RV converter))	(RV INSTRUMENT-PANEL))
CONSTRAINT C8		(and (MODEL = model-70)
		(RV SEATS))
CONSTRAINT C9		and (STATUS = standard)
		(RV STEREO))

Table 1:Car Configuration Cases

Note that the cases include dynamic constraints: RV indicates a dynamic constraint meaning ‘require variable’, and RN indicates a dynamic constraint meaning ‘require not’ variable. Our new configuration problem is to configure a STANDARD, MODEL-80 car. We search the case base for cases with the requested characteristics, finding the two existing cases shown above. Case 1 matches the requested MODEL-80 feature, and Case 2 matches the STANDARD characteristic.

The variables and constraints related to the matched characteristics of the old case are used to set up the new CSP. Thus, from Case 1, all constraints having to do with MODEL-80 (C1, C2, C3, C4), and all variables encountered in those constraints (MODEL, FUEL-EFF, BODY, ENGINE, BATTERY) are added to the new CSP. Any constraints involving the added variables are also added (C7, since it involves FUEL-EFF). Any RV variables found in the added constraints are kept as reserve variables, in case any of the dynamic constraints later involve their activation. Thus, CONVERTER is kept as a reserve variable for the new CSP.

Therefore, from Case 1, we have:

(MODEL MODEL-80)
(BODY CONVERTIBLE)
(FUEL-EFF HIGH)
(ENGINE LARGE)
(BATTERY LARGE)
(CONSTRAINT (C1 (AND (MODEL = MODEL-80) (FUEL-EFF = HIGH))))
(CONSTRAINT (C2 (AND (MODEL = MODEL-80) (RV BODY))))
(CONSTRAINT (C3 (AND (MODEL = MODEL-80) (RV ENGINE))))
(CONSTRAINT (C4 (AND (MODEL = MODEL-80) (RV BATTERY))))
(CONSTRAINT (C7 (AND (FUEL-EFF = HIGH) (RV CONVERTER))))
(Reserve Variable: (CONVERTER CV1))

The same process extracts the appropriate variables and constraints from Case 2:

(STATUS STANDARD)
(STEREO ST2)
(CONSTRAINT (C2 (AND (STATUS = STANDARD) (BODY ≠ CONVERTIBLE))))
(CONSTRAINT (C9 (AND (STATUS = STANDARD) (RV STEREO))))

All of the extracted variables and constraints compose the initial solution to the new configuration design problem. The repair algorithm is now applied, finding that the dynamic constraint C7 from Case 1 is satisfied, and thus we add the reserve variable CONVERTER, along with its value CV1 to the problem. Furthermore, the repair algorithm finds that the value for BODY violates the constraint C2 obtained from Case 2. Thus, that value is repaired choosing a value for BODY that conflicts the least with the remaining values. A value of HATCHBACK is assigned to the variable BODY, resulting in a consistent final solution for the new configuration problem.

This example of a dynamic CSP illustrates how the existing cases not only provide the initial solution and constraints for the new problem, but also *formulate* the problem itself, by identifying the necessary problem variables.

5 Analysis and Results

Both configuration design using backtracking and assembly sequence generation have been characterized as NP-hard problems. Our intuition was that if a problem is hard to solve, then don't solve every one from scratch. In both domains, we found strong decompositional structures in the problems which allow application of old solutions to solve new problems. The computation time spent in adaptation is satisfactory compared to using conventional algorithms. Pu and Reschberger[21] and Pu and Purvis[1] discussed results of this framework applied in the domain of assembly sequence design. The work reported here showed that this framework can be further generalized to any discrete and static or dynamic configuration design problems. Response times for answering the question whether matched cases can be adapted to solve new problems are all within minutes.

To test the algorithm's performance against constructive backtracking, we tested using the well known n-queens problem as well as the assembly sequence design problem. These test gave us positive confirmation that the minimum conflicts algorithm outperforms chronological backtracking, as can be seen in Figure 5.

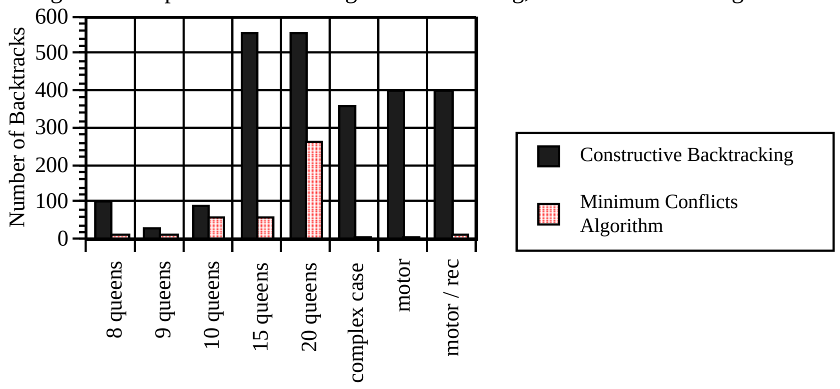

Figure 5: Difference Between Chronological BT and Min-Conflicts Algorithm

To test the hypothesis that dynamic CSP outperforms static CSP, we formulated our assembly sequence design problems both in a dynamic representation as well as a static representation. We found that being able to remove variables from the problem dynamically improved performance significantly, as can be seen in Figure 6. This result corresponds to a similar result found by Mittal[17] in the configuration design domain.

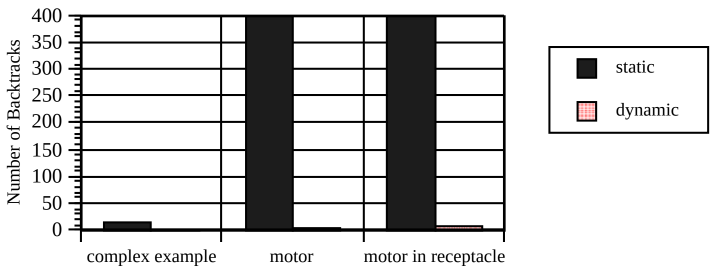

Figure 6:Difference Between Static and Dynamic Min-Conflicts Algorithm

Finally, we tested whether the initial solutions taken from existing cases provide the minimum conflicts algorithm with more guidance than the minimum conflicts algorithm applied alone. As Figure 7 shows, the initial solutions do indeed provide more guidance and therefore less backtracks during the problem solving process.

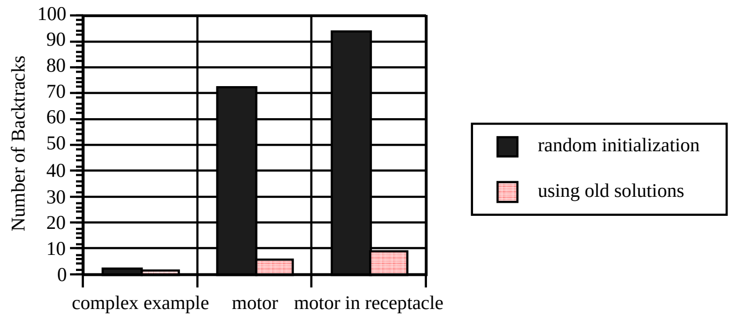

Figure 7:Comparison of using old cases vs. random initialization

These experiments have confirmed the effectiveness of the minimum conflicts algorithm as a method by which to synthesize a global solution from several primitive solutions. Adding the possibility for dynamic constraints has improved its performance further, as well as extended its applicability, and the existing solutions from the case base provide the algorithm further guidance during the problem solving process.

6 Conclusion

CBR is becoming widely recognized as a viable problem solving methodology. It is being applied to a wide range of problem solving domains such as design, diagnosis, planning, customer technical support, legal reasoning, and education. Our methodology formalizes the case adaptation process in the sense of combining multiple cases in order to make the process applicable across varied application domains.

Our methodology uses the existing cases in order to cut down the necessary search space so that each new CSP does not need to be solved from the beginning. Incorporating constraint satisfaction techniques provides formalism to the approach which makes the methodology more widely applicable to any problem which can be represented as a discrete, dynamic or static CSP. In return, the case base provides important information about design variables, constraints, functionality, and characteristics which the CSP algorithm can capitalize on in order to provide efficient performance. Together, the CBR and CSP formalisms combine to provide a methodology for adaptation that will help CBR systems achieve a wider applicability and a better efficiency.

---

[1] Pearl Pu and Lisa Purvis. Formalizing Case Adaptation in a Case-Based Design System. In *Proceedings of the Third International Conference on Artificial Intelligence in Design* (AID'94), August 1994.
[2] J. Kolodner. *Case Based Reasoning*. Morgan Kaufmann Publishers, 1993.
[3] K. Hammond. CHEF: A Model of Case-Based Planning. In *Proceedings of AAAI-86*, Cambridge, MA, 1986.
[4] W. Bain. JUDGE. In R.C.Reisbeck, C.K.Schank, eds., *Inside Case-Based Reasoning*. Erlbaum Publishers, 1989.
[5] D.H. Hennessy and D. Hinkle. Applying Case-Based Reasoning to Autoclave Loading. *IEEE Expert*, 7:21-26, 1992.
[6] J.L. Kolodner and R.L. Simpson. The Mediator: Analysis of an Early Case-Based Problem Solver. *Cognitive Science*, 13:507-549, 1989.
[7] P. Koton. Reasoning about Evidence in Causal Explanation. In *Proceedings of AAAI-88*, Cambridge, MA, 1988.
[8] T.R. Hinrichs. *Problem Solving in Open Worlds: A Case Study in Design*. Northvale Publishers, 1992.
[9] J.G. Carbonell. Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition. In *Machine Learning*, Volume 1, 1986.
[10] J.G. Carbonell and M.M. Veloso. Integrating Derivational Analogy into a General Problem Solving Architecture. In *Proceedings: Workshop on Case Based Reasoning* (DARPA), Clearwater, Florida, Morgan Kaufmann Publishers, 1988.
[11] Pearl Pu. Issues in Case-Based Design Systems. *Artificial Intelligence in Engineering Design, Analysis, and Manufacturing* (AI EDAM), pages 79-85, 1993. As guest editor for a special issue on case-based design systems.
[12] Eric Domeshek and Janet Kolodner. Finding the Points of Large Cases. *Artificial Intelligence in Engineering Design, Analysis, and Manufacturing* (AI EDAM), 1993.
[13] Mary Lou Maher and Dong Mei Zhang. CADSYN: A Case-Based Design Process Model. *Artificial Intelligence in Engineering Design, Analysis, and Manufacturing* (AI EDAM), 1993.
[14] Kefeng Hua and Boi Faltings. Exploring Case-Based Building Design - CADRE. *Artificial Intelligence in Engineering Design, Analysis and Manufacturing* (AI EDAM), 1993.

[15] C. Bessiere. Arc Consistency in Dynamic Constraint Satisfaction Problems. In *Proceedings of the 9th National Conference of AAAI*, Anaheim, 1991.
[16] B. Faltings, D. Haroud, and I. Smith. Dynamic Constraint Satisfaction with Continuous Variables. In *Proceedings of the European Conference on AI*, Wien, 1992.
[17] S. Mittal and B. Falkenhainer. Dynamic Constraint Satisfaction. In *Proceedings of the 8th National Conference of AAAI* , 1990.
[18] D. Gentner. Structure Mapping: A Theoretical Framework for Analogy. *Cognitive Science*, 7, 1983.
[19] W. Ewers. *Sincere's Vacuum Cleaner and Small Appliance Repair Service Manual* . Sincere Press, 1973.
[20] S. Minton, M. Johnston, A. Philips, and P. Laird. Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence*, 58:161-205, 1992.
[21] Pearl Pu and Markus Reschberger. Case-Based Assembly Planning. In *Proceedings of DARPA's Case-Based Reasoning Workshop* . Morgan Kaufmann, 1991.