Robotica (1998) volume 16, pp. 285–295. Printed in the United Kingdom © 1998 Cambridge University Press

COMPOSER: A case-based reasoning system for engineering design

Lisa Purvis* and Pearl Pu†

SUMMARY

The frequent use of past experience by human engineers when solving new problems has led to an interest in the use of case based reasoning (CBR) to help automate engineering design. In engineering design it often occurs that *many* past experiences must be combined to solve a new problem, and thus the process of case based adaptation must efficiently and systematically combine information from many sources. We have developed a constraint based methodology for case combination that allows its application across a wide range of problems. We have shown that our approach provides an efficient adaptation methodology that ensures convergence upon a solution if one exists, provides a uniform representation of cases, and is generalizable beyond just one domain. Our technique is implemented in a case based reasoning system called COMPOSER, which has been tested in two design domains: assembly sequence design and configuration design.

KEYWORDS: COMPOSER; Engineering design; Case-based reasoning; Automated design; Constraint satisfaction algorithm.

1. INTRODUCTION

Engineering design poses many challenges for researchers developing computational models of the process. Artificial Intelligence techniques have received much attention in the last decade as being important for making progress in our understanding of the design process and in the development of computer-based tools to aid designers. However, the complexities of engineering design often inhibit the computational feasibility of AI approaches. The large amounts of potentially inconsistent and constantly changing requirements in complex domains complicate the knowledge acquisition and representation that is critical to developing a knowledge based system. Furthermore, the required knowledge is often domain specific, limiting the applicability of a knowledge based system to one domain. In addition, the scale of problems in complex domains is often large enough to be prohibitive to typical AI techniques which become computationally infeasible as problem size grows.

The reliance on past experience that is such an integral part of human problem solving has motivated the use of case based reasoning (CBR) techniques. A case based reasoning system stores its past problem solving episodes as cases which later can be retrieved and used to help solve a new problem. The process of *adaptation*, or changing the old solutions to fit the new problem requirements, is one component of CBR that has not yet been well addressed. Without adaptation, a CBR system is simply a storage and retrieval tool, leaving the difficult data synthesis to the user. Adaptation is particularly difficult for complex problem domains such as design, where several past experiences must often be combined to solve the new problem. If this combination is not performed in a systematic manner, it is difficult to tell whether a solution is being converged upon at all. Our goal has been to formalize the adaptation process in order to make it efficient and reliable even in complex domains, and to allow its applicability beyond just one problem domain.

We have chosen a constraint satisfaction algorithm to provide the formal methodology for adaptation. Our results, described in the following sections, show that using constraint satisfaction as the adaptation engine provides many advantages:

- An efficient and generalizable methodology for adaptation, applicable across a general class of problems.
- A uniform representation of cases alleviates the knowledge representation issue often problematic when combining information from various sources.
- Allows efficient combination of many cases.
- Ensures convergence upon a solution if one exists.
- Provides a way to implicitly decompose a large problem into smaller sub-problems.
- Provides an assessment of problem adaptability.
- Reduces the reliance on domain specific knowledge.

We have shown the effectiveness of our approach by implementing a design problem solving system COMPOSER which has been tested in two design domains: assembly sequence design and configuration design. We compare and contrast our work with other similar research in Section 2, and continue in Section 3 with a description of the problem representation and its impact on our methodology. Section 4 describes the adaptation process itself, Section 5 shows a short example of the process, Section 6 details our results, and we conclude the paper in Section 7 with a summary and some directions for future work.

2 RELATED WORK

2.1 Adaptation methodologies

Many researchers have examined and proposed methodologies to address the CBR process of adaptation. Three well-known adaptation methods are derivational analogy, substitution, and transformation. In derivational analogy, a

* Xerox Corporation, 800 Phillips Road, 207-01Z, Webster, NY 14580 (USA). lpurvis@wrc.xerox.com
† Laboratoire d'Intelligence Artificielle & Robotique, Swiss Federal Institute of Technology (EPFL) MT-Ecublens, 1015 Lausanne (Switzerland). pu@lia.di.epfl.ch

https://doi.org/10.1017/S0263574798000368 Published online by Cambridge University Press

new solution is computed using the same method by which the old solution was computed. Derivational analogy is found in the existing systems ARIES[1] and PRODIGY/ANALOGY.[2] Substitution methods choose and install a replacement for some part of an old solution that does not fit the current situation requirements, as in CHEF,[3] JUDGE,[4] CLAVIER,[5] and MEDIATOR.[6] 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, as in CASEY[7] and JULIA.[8]

The main difference between many existing adaptation approaches and the one developed for COMPOSER is that COMPOSER begins with a *set* of cases, each of which addresses some component of the new problem. Existing approaches often begin with one old solution which is then changed to solve the new problem.

More recent systems that have addressed the multi-case adaptation issue are EADOCS,[9] IDIOM,[10] CAPlan,[11] and PRODIGY.[12] In COMPOSER, the many matching cases are retrieved at one time from the case base during retrieval, and then these cases are all used simultaneously by the repair algorithm to find a solution to the new problem. In EADOCS, each case addresses one feature of the new problem, and each case is used to adapt the corresponding solution feature. In PRODIGY, cases are replayed at specific choice points during the plan generation. In CAPlan, the problem is analyzed into goals, the goals are used to retrieve cases, and each retrieved case replays its decisions. IDIOM is similar to COMPOSER because of the use of constraint satisfaction during adaptation. However, dimensionality reduction is used in IDIOM in order to eliminate constraint inconsistencies, and continuous constraints are allowed. In COMPOSER on the other hand, the minimum conflicts repair algorithm is used to repair inconsistencies, and discrete, static or dynamic constraints are allowed.

Another important comparison between COMPOSER and other multi-case adaptation systems is in the decomposition of the new problem. Design problems are typically large and complex, necessitating decomposition in order to facilitate reuse of existing cases. However, decomposition is often difficult because the many components of a problem have strong relations among one another.[13] Therefore, it is helpful if a case based reasoner has an efficient and accurate method by which to decompose a new problem before matching it against the cases in the case base. COMPOSER, like PRODIGY, has an implicit decomposition of the new problem. In EADOCS, the decompositions are predefined, in IDIOM, the decomposition is delegated to the user, and in CAPlan, the decomposition is static and domain specific, and it is done before retrieval by decomposing the set of goals.

A similarity in motivation exists between COMPOSER's approach and the approach taken in DEJA VU,[14] where the emphasis is on determining the adaptability of the retrieved cases. In COMPOSER, the adaptability assessment has emerged because of imposing the constraint satisfaction problem (CSP) structure onto the adaptation process. Adaptability can be determined once the set of matching cases has been retrieved, while in DEJA VU,[14] assessing adaptability is used to guide retrieval.

The multi-case approach taken by COMPOSER is unique in that it *simultaneously* combines solutions to several cases, and solves the local inconsistencies between these cases to produce a global solution to the new problem. We have found this approach to be especially suitable to the design domain, where one existing case rarely addresses all of the complexities found in a design problem. Rather, many existing cases must be efficiently and systematically *combined*, and therefore COMPOSER uses a CSP algorithm as the case combination mechanism.

2.2 Constraint satisfaction problems

The general solution to the constraint satisfaction problem is NP-complete,[15] and thus many algorithms have been developed to tame the computational complexity of the CSP. Arc consistency, path consistency, and k-consistency algorithms attempt to eliminate inconsistencies in the constraint network before the search for a solution begins, with the hope of avoiding excessive backtracking.[16]

Other studies have focused on the constraint network *structure*, to determine if any easily solved cases may exist. Freuder investigated situations in which no backtracking is required[17] and in which the bounds on backtracking can be determined.[18] Dechter and Pearl[19] investigated directional arc/path consistency pre-processing techniques, which take into account the direction in which backtracking will eventually search the problem. As a result, they avoid processing many constraints which are unnecessary for the search. Truth Maintenance Systems (TMS) remember their reasoning and use dependency directed backtracking to look at the previous dead-end choices in order to change only those values that are relevant to the current error.[20] The worst-case analysis of all of these algorithms, however, does not reveal the merits of these techniques, as they cannot be shown to consistently provide good results or outperform other methods.

Dynamic CSP has been explored in the work of Bessiere,[21] which describes the algorithm for computing arc-consistency for dynamic constraint satisfaction problems. Faltings[22] explores dynamic constraint propagation in continuous domains. Mittal and Falkenhainer[23] identified four types of dynamic constraints and implemented them within an ATMS framework. Our research has identified a promising CSP algorithm called the minimum conflicts algorithm[24] whose empirical computational time has been shown to grow only linearly in the size of the problem. We have expanded this algorithm to include dynamic constraint capabilities and integrated it into the CBR framework as the adaptation algorithm.

2.3 Assembly sequence problems

Assembly sequence generation is the problem of finding a valid sequence by which to assemble a set of parts into the final product. Typically, the assembly sequence generation problem is viewed as a planning problem, where each step in the sequence is planned. Two well known assembly sequence generation methods are the feasibility testing method[25] and the user questioning method.[26] Each of these methods views the assembly sequence problem as one of

https://doi.org/10.1017/S0263574798000368 Published online by Cambridge University Press

finding the disassembly sequence and then reversing it.

The feasibility testing method relies on first computing all possible decompositions of the assembly and then checking each decomposition in turn to determine whether it is a feasible one. The user questioning method also begins with a set of all possible decompositions of the assembly, and then it continues by asking the user whether or not each decomposition is feasible.

The intent of COMPOSER was to show that through the use of CBR and CSP, the costly feasibility testing and the user questioning could be eliminated, since the necessary precedence constraints are stored in the case base with the previously solved assembly sequence design. COMPOSER addresses both linear and non-linear problems as well as problems with non-monotone characteristics by formulating them uniformly as CSPs.[27] COMPOSER also uses the old solution to generate one good assembly sequence based on experience, eliminating the need for a human expert to later view and edit the assembly sequence generated.

Other work on CBR in assembly sequence generation has been addressed by using the case base at each plan step order to determine which would be the most feasible next connection to make.[28] This again implies searching the case base for the best match at each plan step, whereas COMPOSER's approach attempts to perform a simultaneous adaptation so that only one pass through the case base must be made.

3 REPRESENTATION ISSUES

3.1 CSP as a formulation for design problems

The CSP is a natural formulation for design problems, where the many interacting design constraints and design ideas can be uniformly represented by the constraint satisfaction paradigm, and further automatically integrated and synthesized into a feasible design by correct and complete CSP algorithms, thereby ensuring convergence upon a solution if one exists.

In our test application domain of assembly sequence design, we have formulated the assembly sequence problem as a design problem for two reasons. First, the typical formulation of the assembly sequence problem as a planning problem requires extensive feasibility testing at each plan step in order to determine the valid subassemblies.[25] We found that most assemblies are decomposable, and thus lend themselves to a case combination process, which allows us to eliminate the costly feasibility computations necessary when the problem is formulated as a planning problem.

Second, by formulating the assembly sequence problem (ASP) as a design problem, we can more closely integrate the design decisions with the assembly sequence decisions, which traditionally are separated. The assembly sequence decision is usually made in isolation and does not have any influence on the design decisions, even though a modification of the design could very well simplify the assembly sequence. By storing the information about an assembly in the case base, one can store both design information and assembly information, thus providing a tighter link between the two processes. For instance, separate cases may be stored showing assembly sequences that are more appropriate for slight variations in part designs, thus allowing a designer to potentially look at the outcome of COMPOSER to help see the impact of the design decisions on the assembly sequence.

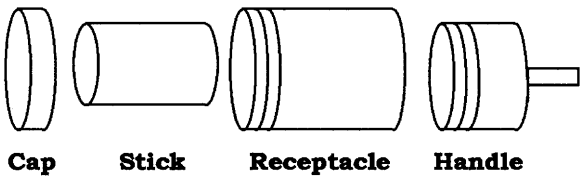

Fig. 1. Receptacle product.

In our representation, the constraints specify the various feasibility constraints placed on the product during its assembly. Consider, for example, the receptacle product shown in Figure 1. One of its feasibility constraints is that the stick must be placed inside the receptacle before both the cap and the handle are attached, otherwise there is no geometrically feasible way to properly include the stick in the final assembled product. The CSP graph of a product closely resembles its connectivity graph which describes the relationships between the parts, known as the relational model[25] shown in Figure 2.

The CSP is formulated by defining each CSP variable to be a connection between two parts. Thus, as shown in Figure 3, the CSP variables for the receptacle device are V1, representing the connection between the cap and the receptacle, V2 (connection between stick and receptacle), and V3 (connection between handle and receptacle).

The variable takes on an integer value representing the step number in the assembly sequence. For example, if a variable is assigned the value 2, the two parts sharing the connection are to be joined as the second step in the

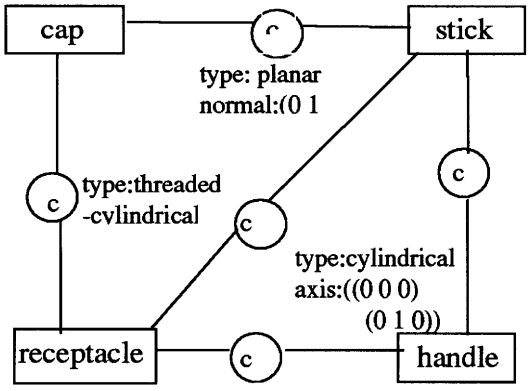

Fig. 2. Relational model of Figure 1.

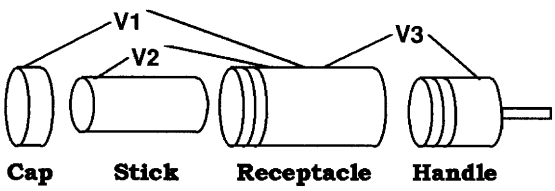

Fig. 3. CSP Representation of the Receptacle Product.

https://doi.org/10.1017/S0263574798000368 Published online by Cambridge University Press

assembly process.

Constraints express precedences between operations. For the receptacle device, there is one spatial and one operational constraint formulated as:

$$C1: (V2 < V3)\ OR\ (V2 < V1)$$
$$C2: Vi \neq Vj$$

C1 captures the knowledge that the stick must go inside the receptacle before both the cap and the handle are attached to the receptacle, and C2 indicates that no two connections may be made at the same time.

The solution to the CSP is then a valid ordering of all of the mating connections so that none of the constraints are violated. A valid assembly sequence for the receptacle is (V1 = 1, V2 = 2, V3 = 3), indicating that the cap and the receptacle are assembled first (connection V1), then the stick is inserted into the receptable (connection V2), and finally the handle is attached to the receptacle (connection V3). Note that this is only one of the possible solutions. COMPOSER's CSP engine is capable of finding more than one solution if it is not stopped after one result.

This CSP representation of design problems allows varied designs to all be uniformly represented, thereby allowing a common adaptation mechanism for cases coming from different sources. The CSP formulation also sets the stage for the use of a CSP algorithm to do the adaptation, which ensures convergence, and provides an efficient and generalizable method by which to combine several cases, as we will show in Section 6 when discussing the results.

3.2 Further expansion of the CSP paradigm to accommodate engineering design

Engineering design problems are often ones which do not fit into a static formalism, because new design requirements often appear later in the design process, and many design decisions made *during* the design process have significant influence on other portions of the design. For these reasons, we chose to augment the general CSP formalism to one of a dynamic CSP formulation, where problem values and constraints may be added or deleted from the problem as the problem solving progresses. An example of this dynamic characteristic can be seen in the 4-blocks assembly diagram shown in Figure 4.

The four blocks cannot be assembled simply by sequentially inserting one block into the next, because the last block will not fit. Therefore, the product must be assembled by creating two sub-assemblies, and then putting the sub-assemblies together. In the four blocks example, we must either put together blocks [A&B], [C&D] and then put the two sub-assemblies together, or alternatively, we must put blocks [B&C], [A&D] together, and then put those two sub-assemblies together. Now consider the situation where we have chosen to make connection V1 [A&B], and then V3 [C&D], and now we choose to make connection V2. Connection V4 then occurs automatically, and can therefore be eliminated from further consideration in the problem—it can in fact be *deleted* from the problem. This type of dynamic deletion (or addition) of variables to the CSP during problem solving is known as a dynamic constraint satisfaction problem. We have incorporated dynamic capability into COMPOSER's adaptation algorithm.

There are four types of dynamic constraints implemented in COMPOSER, based upon those introduced by Mittal and Falkenhainer.[23] COMPOSER's adaptation mechanism has been designed and implemented to include these dynamic constraint capabilities, in order to better represent a wider variety of engineering design problems.

3.3 Case based reasoning (CBR) as a description of the design process

While the CSP is an effective and formal method by which to describe the design product itself, it does not necessarily address all of the aspects involved in the design *process.* The design process is one which inherently relies on expertise, rules of thumb, and specialized knowledge, which each designer accumulates though experience. This type of accumulated expertise is best approximated by a CBR system, where the past design cases are stored in the case base for future reference. In this way, the case based reasoner may draw upon this experience to help it automatically solve new problems, just as the experienced designer draws on accumulated experience when developing a new design.

In COMPOSER, the CSP representation of a design case is stored in the case base with the variables and constraints stored as feature-value pairs, along with the solution to the case. The case representation of the receptacle CSP is shown in Figure 5.

We found that by storing small cases, each of which addresses some important design consideration and represents its solution, the case based reasoner can implicitly decompose the new problem into its constituent parts during matching, and then combine the many matching cases into a design product which satisfies its requirements.

The effectiveness of the stored cases in helping to solve new problems lies in discovering a systematic methodology

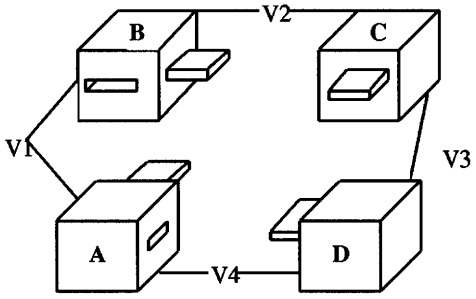

Fig. 4. 4-Blocks Example.

```
(NAME (CAP-STICK-RECEPTACLE-HANDLE))
(PART CAP)
(PART STICK)
(PART RECEPTACLE)
(PART HANDLE)
(MATING-REL ((INSIDE STICK RECEPT) V2))
(MATING-REL ((SHAFT-HOLE RECEPT CAP) V1))
(MATING-REL ((SHAFT-HOLE HANDLE RECEPT) V3))
(CONSTRAINT (C1 ((V2 < V1) OR (V2 < V3))))
(SOLUTION (S1 ((V1 1) (V2 2) (V3 3))))
```

Fig. 5. Case representation of the receptacle CSP.

https://doi.org/10.1017/S0263574798000368 Published online by Cambridge University Press

by which to combine this stored expertise into a new design product. Our research has identified the CSP as being a well-suited formalism for this task. It allows the cases to be uniformly represented, and allows the rigor and completeness of CSP algorithms to guide the incorporation of the various pieces of design information stored in the many cases in the case base. The combination of formalism and versatility allowed by the CSP enables our methodology to apply across a wide range of problems.

4 THE ADAPTATION PROCESS

4.1 Adaptation described

Adaptation is often considered to be the most difficult component of a case based reasoning system. This is especially true in complex domains such as engineering design, where it may not be immediately apparent how a set of cases should be combined to result in an effective new design product. For this reason, we have chosen a formal CSP algorithm, the minimum conflicts algorithm[24] to accomplish adaptation in our system. The issues that CSP addresses solve many of the inherent difficulties of adaptation. For instance, the CSP provides a common case representation, allowing cases from many different sources to be integrated seamlessly into a solution for a new problem. Secondly, choosing the CSP formalism allows the application of systematic and complete CSP algorithms, thus ensuring convergence upon a solution which is often problematic in adaptation methodologies.

In addition, using a CSP algorithm for adaptation reduces the often domain-specific adaptation knowledge that must be stored with the system. Traditionally, in order to accomplish adaptation, a system must evaluate which portions of a new problem need to be adapted, and must choose *how* to adapt in order to fit the new problem requirements. The formulation of adaptation as a CSP eliminates both of these knowledge requirements, freeing the system from being dependent on domain specific heuristics.

In COMPOSER, the existing cases themselves provide the necessary constraints for the new problem. This allows the minimum conflicts repair algorithm itself to determine what values need to be changed/adapted according to these constraints. Furthermore, the decision about how to adapt a piece of information from an old case is eliminated in COMPOSER, since the minimum conflicts algorithm adapts by choosing the value that conflicts the least with the remaining values. Because this method remains constant across all problem domains, the need for domain specific heuristics is eliminated.

A further advantage of using the CSP formulation for the cases is that it allows an implicit decomposition of the new problem into its constituent cases in the case base. Thus, COMPOSER can avoid the problem of an a priori decomposition that is useless because the subdivided problem has no corresponding cases in the case base. Lastly, we found that imposing the formalism of a CSP on the adaptation process allows us to make assessments about the adaptability of the retrieved cases, a concept that has been elusive in case based reasoning up until now. The adaptability assessment can be used to determine whether or not it is computationally worthwhile to do the adaptation. If not, one approach that we are currently pursuing is to improve the adaptability of the retrieved cases.[29]

The integration of CSP and CBR allows both methodologies to draw from each other's strengths. The cases help to minimize the computational complexity of the CSP, while CSP helps to fully utilize the information in the cases to do effective problem solving. COMPOSER's overall problem solving methodology is shown in Figure 6.

4.2 Motivation for COMPOSER's adaptation mechanism

To illustrate the motivation for using the minimum conflicts algorithm as the adaptation mechanism, let us look at an example of a typical CSP, the map coloring problem.

If we are trying to color a map of the US with four colors so that no two neighboring states have the same color, we pose this as a CSP and then backtrack through all possible combinations of colors, trying to find a combination in which no two neighboring states have the same color. This is a computationally complex task which grows exponentially in the number of states/problem variables. To utilize past knowledge, and to alleviate the computational complexity problem, our methodology takes solutions to previously solved subproblems and *combines* them into a solution to the new problem. Thus, consider that there have been colorings designed for the western and the eastern portions of the US. As Figure 7 shows, these two sub-

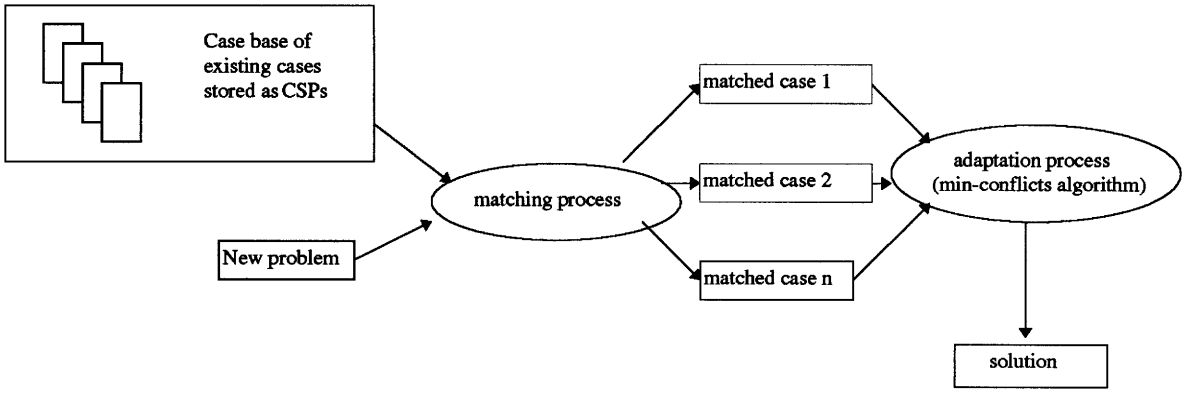

Fig. 6. COMPOSER's Problem Solving Methodology.

https://doi.org/10.1017/S0263574798000368 Published online by Cambridge University Press

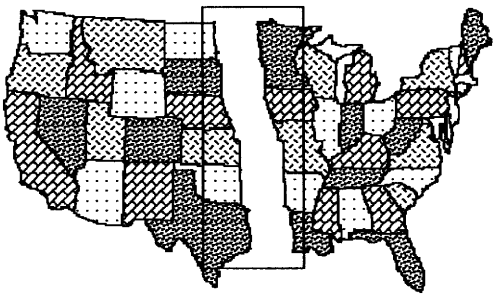

Fig. 7. Eastern and Western US Colorings Combined.

solutions cannot simply be pasted together to form a consistent coloring for the entire United States because of conflicts that occur at the border of the two solutions. Furthermore, if the conflicts are not resolved in a systematic manner, then we cannot even tell if we are converging upon a solution at all.

Thus, in COMPOSER's methodology, these sub-solutions are retrieved from the case base and used as an initial starting point for the minimum conflicts repair algorithm. The minimum conflicts algorithm then repairs these initial conflicts using the minimum conflicts heuristic described in Section 4.3. The appeal of the minimum conflicts algorithm is that the number of repairs necessary to come to a solution remains constant as the size of the problem grows, thereby providing a mechanism which can be applied even to large complex problems such as engineering design.

4.3 Minimum conflicts algorithm as the adaptation mechanism

The minimum conflicts algorithm is the means by which COMPOSER combines all of the solutions to the matched cases into a consistent solution for the new problem. Recall that even though the stored cases each have a consistent local solution, when combined they exhibit global conflicts that must be repaired in order to find a consistent global solution for the new problem. The minimum conflicts algorithm was chosen as the adaptation algorithm because it has been shown to be more effective than traditional chronological backtracking techniques, and it starts with an initial solution, which in this framework comes from the case base. The minimum conflicts repair strategy is shown in Figure 8.

The original algorithm begins with an initial solution,

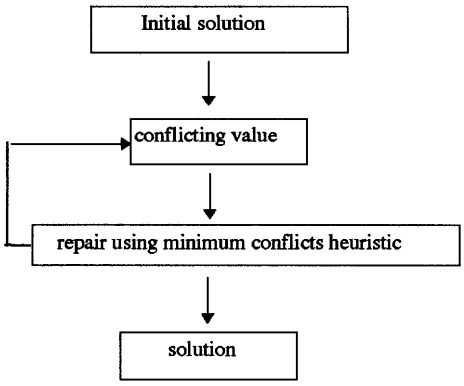

Fig. 8. Minimum Conflicts Algorithm.

```
< When a dead-end is reached (i.e. no valid value can be
  found)
  < Until encountering a choice point with an unexplored
    alternative
        < Withdraw the most recently made choice
        < Undo all consequences of the withdrawn choice
        < Move forward again making choices
```

Fig. 9. Chronological Backtracking.

obtained by choosing values for each variable using a greedy strategy in order to attempt to find a good initial solution. It then chooses a variable that violates some of its constraints and repairs that variable by choosing a value that conflicts the least with the remaining values.

Conflicting variable values continue to be chosen and repaired until no more conflicts remain, at which time the algorithm has found a solution to the problem. Although the general solution to the CSP has been shown to be NP-complete, and thus traditional methods for solving the CSP are computationally prohibitive, the empirical time of the minimum conflicts algorithm grows only *linearly* with the size of the problem, and is thus practical for use even with large problems. This linear empirical time is attributed to the fact that the number of repairs required remains constant as the size of the problem grows.[24] However, the worst case complexity is still exponential, as for the other CSP algorithms.

The minimum conflicts algorithm as implemented in COMPOSER is the minimum conflicts heuristic embedded within a backtracking algorithm. The initial values for the variables come from the retrieved matching cases. Traditional chronological backtracking proceeds by trying to assign a value to every variable so that all constraints are satisfied. When no value can be found that satisfies the necessary constraints, backtracking results, as is shown in Figure 9.

In minimum conflicts backtracking, all variables begin with an initial value, and the difference in the algorithm is seen when moving ahead to make new choices: the algorithm does not blindly choose the next value available. Instead, it looks at the values chosen so far and takes that choice that conflicts the least with current values, as is shown in Figure 10.

The dynamic component of the backtracking algorithm that has been added for COMPOSER affects the point in the algorithm that undoes the consequences of withdrawing a choice. Since variables may have been added or deleted based on a certain choice, those variables must be put back into or deleted from the problem as appropriate. The changes to the algorithm are shown in Figure 11.

```
< When a dead-end is reached (i.e. no valid value can be
  found)
  < Until encountering a choice point with an unexplored
    alternative
        < Withdraw the most recently made choice
        < Undo all consequences of the withdrawn choice
        < Move forward again making choices that conflict
          the least with choices made so far.
```

Fig. 10. Minimum Conflicts Backtracking.

https://doi.org/10.1017/S0263574798000368 Published online by Cambridge University Press

```
< When a dead-end is reached (i.e. no valid value can be
  found)
    < Until encountering a choice point with an unexplored
      alternative
          < Withdraw the most recently made choice
          < Undo all consequences of the withdrawn choice by
            adding or deleting all appropriate variables and
            updating problem constraints accordingly.
          < Move forward again making choices that conflict
            the least with choices made so far.
```

Fig. 11. Dynamic Minimum Conflicts Algorithm.

By using the minimum conflicts algorithm augmented in this way, we are able to utilize the past experience stored in the case base, which alleviates the necessity of solving every CSP from scratch, and it also provides a systematic way to put more than one matching case together to solve the new problem.

We have applied this methodology to both assembly sequence problems and to configuration design problems,[30] and the results are detailed in Section 6.

5 EXAMPLE

Consider the following example of a motor, shown in Figure 12.

Consider that we have stored in the case base the receptacle example shown in Figure 3. Recall that the constraints for the receptacle were such that the stick must be placed inside the receptacle before both the handle and the cap are attached. This same principle can be found in the motor shown in Figure 12, where the armature assembly must be placed inside the field assembly before both the fan end bracket and the commutator end bracket are attached. COMPOSER finds this correspondence between the old and the new case using structure mapping[31] and nearest neighbor similarity metrics.

The matching process returns the variables and connections which correspond between the old and the new case.

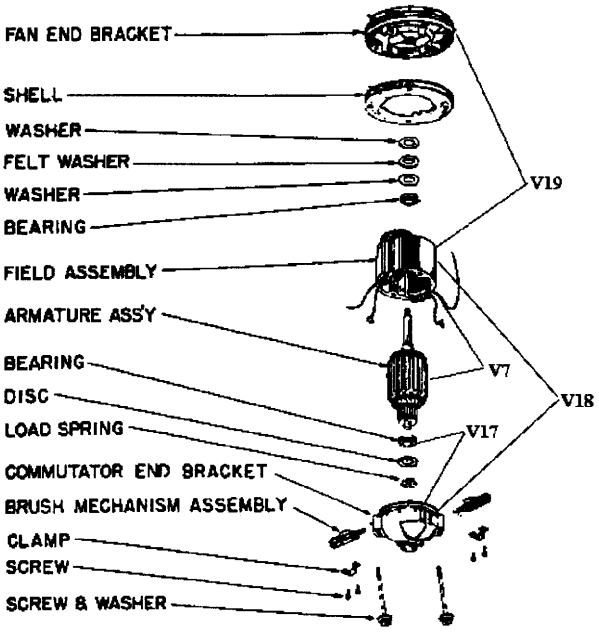

Fig. 12. Motor Example.

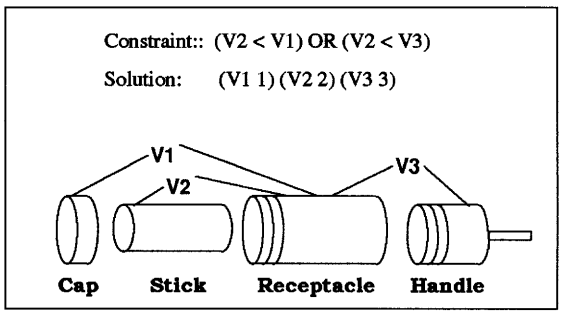

Fig. 13. Receptacle Case Information in Case Base.

In the current example, the matching information obtained is:

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

Since COMPOSER has the appropriate constraint and solution for the existing receptacle case in the case base as shown in Figure 13, it can automatically deduce the constraint that exhibits the same principle in the motor case simply by substituting the appropriate matching variables into the old case's constraint.

Since V3 from the receptacle case matches V18 from the motor case, COMPOSER substitutes V18 for all V3's in the receptacle case's constraint, substitutes V19 for all V1's and substitutes V7 for all V2's, obtaining the following constraint that exhibits the geometric feasibility constraint for the motor:

$$C1 = (\text{OR } (V7 < V19)\ (V7 < V18))$$

COMPOSER also uses the matching variable information to provide initial values for the motor problem's variables. As V1 was assigned the value 1 in the matching receptacle case, V19 is assigned the value 1 in the motor problem (because V1 was found to have matched V19), and similarly, all of the matching variable values are transferred for use in the new problem, resulting in the following variable assignment obtained from the receptacle case:

$$(V19\ 1)(V7\ 2)\ (V18\ 3)$$

Now consider that there exists the following case in the case base, as shown in Figure 14.

This example case is called the press-fit case, which exhibits a mechanical feasibility constraint because part B must be press-fit into part A before part C is attached to A, otherwise the press-fit connection cannot be properly made. There is a match between the press-fit case and a subassembly of the motor problem, in that the bearing must be press-fit into the commutator before the field-assembly is attached to the commutator, otherwise, the press-fit connection cannot be properly made between the commutator and the bearing. COMPOSER finds this match as before, by structure-mapping and nearest-neighbor matching methodologies, to determine that the matching variables are as

https://doi.org/10.1017/S0263574798000368 Published online by Cambridge University Press

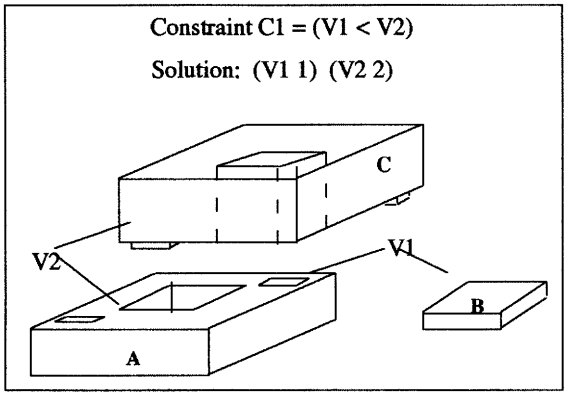

Fig. 14. Press-Fit Case.

follows:

$$V1 \rightarrow V17$$
$$V2 \rightarrow V18$$

The resulting constraint for the motor case based on the matching variables is: C1 = (V17 < V18), and the initial values obtained from the press-fit case are: (V17 1) (V18 2).

COMPOSER continues in this manner, assigning all appropriate constraints and variable values from matching cases in the case base. As Figure 15 shows, there may be overlaps between the variables that are matched by more than one existing case in the case base. We call these variables *edge variables*, because they occur at the boundary of 2 sub-cases.

In our example, variable V18 is matched by both the press-fit and the receptacle existing cases. COMPOSER does not re-assign values that have been matched by previous cases, and thus in our example, COMPOSER only takes the value for variable V17 from the press-fit case, as the value for V18 has already been assigned from the receptacle case.

Another interesting characteristic to notice is that because of the ordering nature of assembly sequence problems, there can exist what we call *unnatural overlaps* between cases. These unnatural overlaps occur when two cases each have numbered a connection with the same value, even though the two connections have no relation to one another.

For example, the receptacle case has numbered V19 as the 1st connection, and the press-fit case has numbered V17 as the 1st connection. This causes an overlap between the two cases, even though V19 and V17 are not the same connection, and do not have any constraints in common. In order to alleviate this unnecessary initial conflict, COMPOSER begins numbering each successive case's contributed values at 1 plus the current maximum value of the initial solution. So in our example, the initial values from the receptacle case were (V19 1) (V7 2) (V18 3), and COMPOSER now begins the matched values from the press-fit case at '4', assigning (V17 4) (and not re-assigning V18 to 5, since it has already been given a value from the receptacle case). The initial values for these two sub-assemblies of the motor are therefore:

(V19 1) (V7 2) (V18 3) (V17 4)

indicating that we must first assemble the fan-end-bracket to the field assembly, then insert the armature into the field assembly, then attach the commutator to the field assembly, and then finally to press fit the bearing into the commutator. This initial solution, however, is not valid, as it does not satisfy all of the new problem's constraints. The minimum conflicts algorithm now performs its repair, by choosing one of the variables that violates its constraints. Recall that the new problem's constraints are:

$$C1 = (OR\ (V7 < V19)\ (V7 < V18))$$
$$C2 = (V17 < V18)$$
$$C3 = V_i \neq V_j$$

By examining each initial variable in turn, COMPOSER finds that variable V18 violates its constraints, and thus chooses to repair V18 by evaluating each possible value for V18, and choosing the one that violates the least constraints. As shown in the chart in Figure 16, any of the values 5 through 9 would be a good choice for variable V18, and thus COMPOSER randomly chooses one of these, say 5.

After making this repair to the edge-variable V18, the solution for these two sub-assemblies of the motor is:

(V19 1) (V7 2) (V17 4) (V18 5)

indicating that we should first attach the fan-end-bracket to the field assembly, then insert the armature into the field-assembly, then press-fit the bearing into the commutator, and finally to attach the commutator to the field-assembly. This one repair has resulted in a solution for this portion of the motor problem that is consistent with all of the constraints. In this way, COMPOSER combines the information from the case base into valid solutions for new problems.

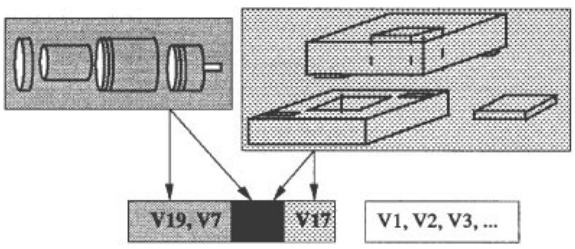

Fig. 15. Matching Cases Contribute Initial Values.

6 RESULTS

6.1 CSP as a case combination/adaptation method

Various observations can be made about the behavior of the minimum conflicts algorithm as a case combination methodology. One observation confirms a previously found result, that the minimum conflicts algorithm outperforms

Value	#Violated	Which Ones
1	3	C1,C2,C3
2	2	C2,C3
3	1	C2
4	1	C2
5-9	0	

Fig. 16. Values and their Constraint Violations.

https://doi.org/10.1017/S0263574798000368 Published online by Cambridge University Press

chronological backtracking.[24] The result which was discovered by applying COMPOSER to solve ASPs is that using the solutions from the case base outperforms the traditional minimum conflicts algorithm, thereby providing the evidence that the combined CSP/CBR methodology is an effective method by which to solve design problems that require past experience and encompass large amounts of variables and constraints. It must be remembered that even in the tests where the old solutions from the case base are *not* used to guide the minimum conflicts algorithm, the case base still provides essential information (in the form of constraints) to the algorithm. Thus, the case base is a key input component for the minimum conflicts algorithm in order for it to solve various design problems. The concrete test results found in our tests run within COMPOSER are shown in the following sections.

6.2 Minimum conflicts algorithm outperforms chronological backtracking

We first wanted to confirm the previously found result that the minimum conflicts algorithm outperforms traditional chronological backtracking. This observation was confirmed by our experiments in COMPOSER and shows that the minimum conflicts heuristic guides the search better than does traditional chronological backtracking, resulting in less backtracks and thus better performance from the algorithm. The graph shown in Figure 17 summarizes this result as found by running various n-queens and assembly problems in COMPOSER, where performance was evaluated based on the number of backtracks. This is the traditional evaluation measure for such backtracking algorithms, since the number of backtracks is the factor which most affects the time to find a solution. As Figure 17 shows, the number of backtracks required by the minimum conflicts algorithm is much smaller than the number of backtracks required for chronological backtracking.

6.3 Minimum conflicts algorithm with solutions from case base outperforms minimum conflicts algorithm with greedy initialization

In our next experiments, we wished to determine whether or not the solutions from the case base further improved the performance of the minimum conflicts algorithm. The

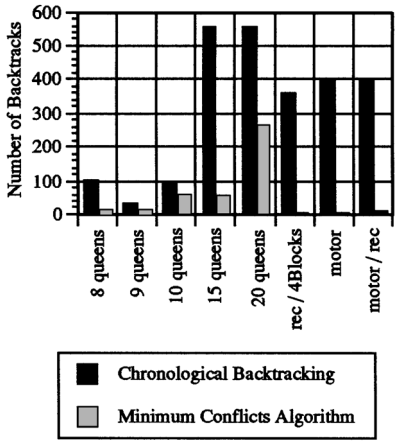

Fig. 17. Chronological BT vs. Minimum Conflicts Algorithm.

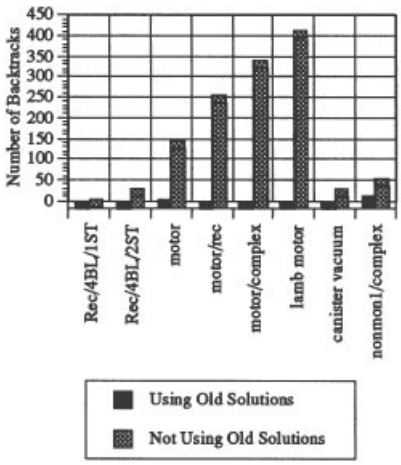

Fig. 18. Using Old Solutions vs. Greedy Initialization.

experiments shown in Figure 18 confirm that starting with solutions from the case base indeed requires fewer backtracks to find a solution than does starting with an initial solution obtained by a greedy algorithm. This result justifies the use of the minimum conflicts algorithm as a means by which to perform case-based adaptation in the sense of combining cases. It provides a systematic methodology which ensures that a solution will be found if one exists, while also providing good performance.

We have found, however, that the old solutions do not *always* provide better performance than does the traditional minimum conflicts algorithm, as shown in Figure 19. We found that there are two factors which influence the effectiveness of the old solutions: the number of initially inconsistent edge variables, and the number of choices remaining for the other edge variables. When there is a high amount of initial inconsistency coupled with very tight constraints on the initially inconsistent variables, the old solutions *do not* provide more guidance than does the minimum conflicts algorithm without using old solutions—that is, there is a point at which adaptation is not cost effective. There are situations where from-scratch problem solving offers the same efficiency as does using the solutions from the case base.

The deciding factor based on COMPOSER's results was the constrained-ness of the edge variables. If the edge variables were overly constrained, then using the solutions from the case base did not offer an improvement in efficiency over the from-scratch method. To understand why

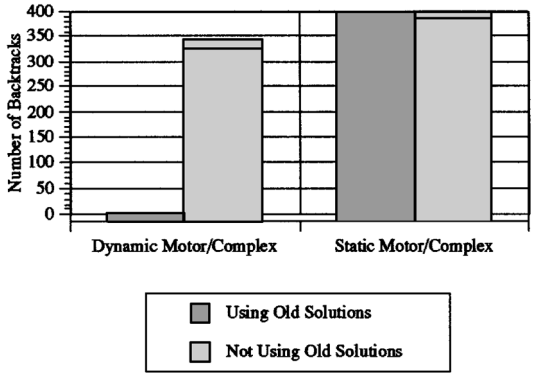

Fig. 19. Cost Effectiveness of Adaptation.

https://doi.org/10.1017/S0263574798000368 Published online by Cambridge University Press

this is so, one must consider the edge variables of the new problem—those that are matched by many existing cases, and thus are at the 'edge' between two existing solutions. The factor that reduces the performance of using the old solutions is the number of consistent choices available for the remaining edge variables in the problem, which directly relates to how constrained the edge variables are. If there are very few remaining choices for the other edge variables, AND there are a high number of initially inconsistent edge variables, this means that it is likely that more variables than just the edge variables will need to be repaired, thus destroying the case solutions from the case base.

Our observations can be used to assess the adaptability of the retrieved set of cases. Since considerable effort is expended to retrieve matching cases, we would ideally like to use the retrieved cases in all instances—even in those situations where the retrieved cases are difficult to combine. We have begun work on improving case combination in these difficult situations by employing a genetic algorithm.[29]

7. CONCLUSION

This research has investigated a methodology which allows a case based reasoning system to be used not only as a storage and retrieval tool, but also as a problem solving tool through the use of the adaptation mechanism. By employing a formalized methodology (CSP), the adaptation method described here can be applied to any problem which can be described as a static or dynamic, discrete CSP, thereby providing a generalized mechanism for CBR problem solving. We have described the system extensively in reference 32. This article has provided a further analysis of the results obtained by our system. We have shown that formalizing the adaptation process allows measurement of the adaptability of the retrieved cases—a concept that has been previously difficult to quantify. We have also shown that the decomposition of a new problem can occur implicitly as a result of the matching process, and that by using a formal CSP methodology, we were able to eliminate some of the domain specific knowledge typically required in case based reasoners.

Future work on COMPOSER will involve creating a more robust matching methodology so that fewer existing cases will be necessary in order to cover a wide range of problems. Another aspect of the adaptation ready for further expansion is to allow hard and soft constraints, as well as the ability to incorporate continuous constraints.

COMPOSER has been shown to be a tool for adaptation that can accomplish implicit decomposition of the new problem, assessment of adaptability, guaranteed convergence upon a solution if one exists, and a formalized and generalizable approach to case representation and adaptation. In this way, COMPOSER provides a methodology by which adaptation can become a more widely applicable and usable technique.

References

1. J.G. Carbonell, *Machine Learning: An Artificial Intelligence Approach.* (Morgan Kaufmann Publishers, San Mateo, CA, 1986).
2. J.G. Carbonell and M.M. Veloso, *Proceeding of the Workshop on Case Based Reasoning (DARPA)*, Clearwater, Florida, Morgan Kaufmann Publishers (1988) pp. 153–159.
3. K. Hammond, ''CHEF: A Model of Case-Based Planning'' *Proc. AAAI-86*, Cambridge, MA (1986) pp. 267–271.
4. W. Bain ''Case-based Reasoning: A computer Model of Subjective Assessment'' *Ph.D. thesis* (Yale University, 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'' *Proc. AAAI-88*, Cambridge, MA (1988) pp. 256–263.
8. T.R. Hinrichs *Problem Solving in Open Worlds: A Case Study in Design* (Lawrence Earlbaum Publishers, Hillsdale, NJ, 1992).
9. B.D. Netten, R.A. Vingerhoeds, H. Koppelaar and L. Boullart ''Expert Assisted Optimization of Composite Structures'' *SCS European Simulation Symp. ESS'93* (Verbraeck, A. and Kerckhoffs, E.J.H., Eds.) (1993) pp. 143–148.
10. I. Smith, C. Lottaz and B. Faltings, ''Spatial Composition Using Cases: IDIOM'' *Proc. 1st International Conference on Case Based Reasoning* (Veloso, M. and Aamodt, A., Eds.), LNAI series, Springer Verlag Publishers (1995) pp. 88–92.
11. H. Munoz and J. Huellen, ''Retrieving Cases in Structured Domains by Using Goal Dependencies'' *Proc. 1st International Conference on Case Based Reasoning* (Veloso, M. and Aamodt, A., Eds.), LNAI series, Springer Verlag Publishers (1995) pp. 241–252.
12. K. Haigh and M. Veloso, ''Route Planning by Analogy'' *Proc. 1st International Conference on Case Based Reasoning* (Veloso, M. and Aamodt, A., Eds.), LNAI series, Springer Verlag Publishers (1995) pp. 169–180.
13. J.L. Kolodner, *Case Based Reasoning* (Morgan Kaufman Publishers, 1993).
14. B. Smyth and M. Keane, ''Experiments on Adaptation-Guided Retrieval in Case-Based Design, In Domains by Using Goal Dependencies'' *Proc. 1st International Conference on Case Based Reasoning* (Veloso, M. and Aamodt, A., Eds.), LNAI series, Springer Verlag Publishers (1995) pp. 313–324.
15. E.C. Freuder, ''Synthesizing Constraint Expressions'', *Communications of the ACM* **21**(11), 958–966 (1978).
16. B. Nadel, ''Tree Search and Arc Consistency in Constraint Satisfaction Problems'' *Search in Artificial Intelligence* (Springer Verlag Publishers, 1988) pp. 287–342.
17. E.C. Freuder, ''A Sufficient Condition for Backtrack Free Search'' *J. ACM* **29**(1), 24–32 (1982).
18. E.C. Freuder, ''A Sufficient Condition for Backtrack-Bounded Search'' *J. ACM* **32**(4), 755–761 (1985).
19. R. Dechter and J. Pearl, ''Network-Based Heuristics for Constraint Satisfaction Problems'' *Artificial Intelligence* **34**, 1–38 (1988).
20. J. Doyle, ''A Truth Maintenance System'', *Artificial Intelligence* **12**, 231–272 (1979).
21. C. Bessiere, ''Arc Consistency in Dynamic Constraint Satisfaction Problems'', *Proc. AAAI-91* (1991) pp. 221–226.
22. B. Faltings, D. Haroud and I. Smith, ''Dynamic Constraint Satisfaction with Continuous Variables'' *Proc. of the 10th ECAI* (1992) pp. 754–758.
23. S. Mittal and B. Falkenhainer, ''Dynamic Constraint Satisfaction'' *Proc. AAAI-90* (1990) pp. 25–32.
24. 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).
25. L.S. Homem de Mello and A.C. Sanderson, ''A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences'' *IEEE Transactions on Robotics and Automation*, 228–240 (1991).

https://doi.org/10.1017/S0263574798000368 Published online by Cambridge University Press

26. T.L. DeFazio and D.E. Whitney, ''Simplified Generation of All Mechanical Assembly Sequences'' *IEEE Journal of Robotics and Automation* **3**(6), 640–658 (1988).
27. L. Purvis, ''Intelligent Design Problem Solving Using Case Based and Constraint Based Techniques'' *Ph.D. Thesis* (The University of Connecticut, 1995).
28. P. Pu and M. Reschberger, ''Case Based Assembly Planning'', *Proc. DARPA's Case-Based Reasoning Workshop*, Morgan Kaufmann Publishers (1991) pp. 245–256.
29. L. Purvis and S. Athalye, ''Towards Improving Case Adaptability with a Genetic Algorithm'' *Proceedings of the 2nd International Conference on Case Based Reasoning* (Leake, D. and Aamodt, A., Eds.), LNAI series, Springer Verlag Publishers (1997) pp. 403–412.
30. L. Purvis and P. Pu, ''Adaptation Using Constraint Satisfaction Techniques'' *Proc. 1st International Conference on Case Based Reasoning* (Veloso, M. and Aamodt, A., Eds.), LNAI series, Springer Verlag Publishers (1995) pp. 289–300.
31. D. Gentner, ''Structure Mapping: A Theoretical Framework for Analogy'', *Cognitive Science* **7**(2), 155–170 (1983).
32. P. Pu and L. Purvis, ''Formalizing the Adaptation Process for Case Based Design'' In: *Issues and Applications of Case Based Reasoning to Design* (Maher, M.L. and Pu, P., Eds.) (Lawrence Erlbaum Associates, 1997) pp. 221–239.

https://doi.org/10.1017/S0263574798000368 Published online by Cambridge University Press