An Approach To Case Combination

Lisa Purvis[1] and Pearl Pu[2]

Abstract. We present a formalization of the adaptation process, by viewing it as a constraint satisfaction problem (CSP), and show our results as obtained by our system COMPOSER, which does a simultaneous adaptation of multiple cases in the domains of assembly sequence design and configuration design.

1 INTRODUCTION

Our system, COMPOSER, solves assembly sequence and configuration design problems using a CSP engine as the adaptation mechanism. Its main goal has been to formalize the adaptation process as a CSP in order to allow a more general applicability of adaptation. The motivation for this approach arose from the observation that in complex domains, many cases must often be *combined* into a solution, but that this multi-case combination may not be convergent if not done systematically. Thus, COMPOSER's approach is to provide a formalism by which to efficiently combine several cases while also ensuring convergence. We showed that by choosing a general formalism for adaptation, the approach can be applied beyond just one domain. Our research has also resulted in some interesting observations about problem decomposition and evaluating problem adaptability.

2 DESCRIPTION OF COMPOSER'S APPROACH

In order to apply a constraint satisfaction algorithm to do adaptation, the existing cases and the new problem must all be represented as CSPs. Therefore, COMPOSER represents each case in the case base as a CSP, by storing the problem variables, constraints and solution as feature value pairs. An example of a configuration design problem as it is stored in the case base as a CSP is shown in Figure 1.

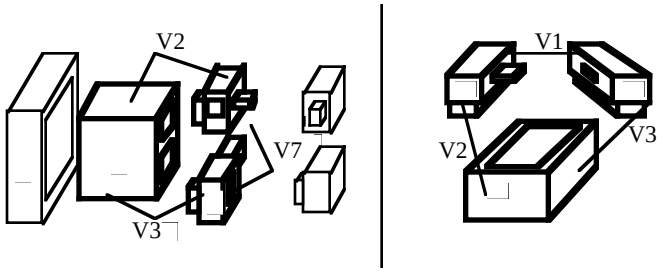
```
(MODEL MODEL-70)
(STATUS STANDARD)
(FUEL-EFF MEDIUM)
(AIRCOND AC1)
(FRAME HATCHBACK)
(ENGINE SMALL)
(BATTERY LARGE)
(SUNROOF SR1)
(GLASS NOT-TINTED)
(CONSTRAINT (AND (STATUS = STANDARD)
                 (AIRCOND ≠ AC2)))
(CONSTRAINT (AND (STATUS = STANDARD)
                 (FRAME ≠ CONVERTIBLE))))
```

Figure 1. Configuration Design Case Represented as CSP

The problem variables (MODEL, STATUS, FUEL-EFF, AIRCOND, FRAME, etc.) are case features, while their values constitute the solution to the CSP. Such a case representation allows any task that can be described as a CSP to be stored as a case, and therefore does not limit the kinds of tasks that can be solved with this approach. Furthermore, since the CSP cases are stored as traditional attribute-value pairs, conventional similarity metrics can be used to determine similarity. This stored information is used to formulate a new problem as a CSP as follows. When we find an existing case which matches a portion of the new case, its constraints are incorporated into the new CSP, along with the solution values for the corresponding variables.

Note that in COMPOSER, this multi-case coverage of a new problem occurs implicitly during the matching process. That is, COMPOSER does not explicitly decompose the new problem. The decomposition occurs because of the cases stored in the case base, and how they match portions of the new problem. For example, the decomposition given stored cases a,b,c,d would be different from the decomposition given cases e,f,g. 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.

Consider the assembly sequence problem shown in Figure 2A. COMPOSER searches the case base when presented with this new problem, and it finds the matching case shown in Figure 2B, where the highlighted components and connections show the matching structural correspondence between the two problems. The constraint for Figure 2B tells us that the two pieces of the cover must be connected before either half is attached to the bottom (otherwise, the two top pieces cannot be put together). This same principle is found in Figure 2A, only it applies to different variables. This structural similarity between the new problem and the existing case is found using a structure mapping approach[18].



Figures 2A, 2B. New Assembly Sequence Problem & Existing Case

The matching variables between the existing and the new problem, and the existing problem's constraints are then used to deduce constraints for the new CSP.

All of the existing matching cases not only contribute their *constraints* to the new CSP, but they also initialize the CSP variables with their *solution values*. Thus, if the solution to the Figure 2B case was V1 = 1, V2 = 2, indicating that connection V1 is to be made first and then connection V2 is to be made, then these same values will be assigned to V7 and V2/V3 of the new

[1] The Jackson Laboratory, 600 Main Street, Bar Harbor, ME, 05609, USA
[2] Laboratoire d'Intelligence Artificielle & Robotique, Institut de Microtechnique/DMT, Swiss Federal Institute of Technology (EPFL), MT-Ecublens, 1015 Lausanne, Switzerland.

problem. Note that connection V3 occurs simultaneously with V2, and thus it can be eliminated from the problem. This feature was accomplished in COMPOSER by allowing dynamic constraints in the CSP[1].

It is easy to imagine, however, that these local solutions to the subproblems of the new CSP, each obtained from an existing case, do not necessarily combine to create an initially consistent solution for the new problem. To clearly illustrate this, consider the well known map coloring problem, where one attempts to color all regions of a map using a specified number of colors so that no two neighboring regions have the same color. If we are trying to color a map of the US using four colors, and we have already solved the problem for the Western and the Eastern portions of the US separately, then we cannot simply put the two solutions together to find a solution for the entire US because of conflicts that appear at the border of the two solutions, as can be seen in Figure 3. Furthermore, if we do not repair the conflicts in a systematic way, then we may not even converge upon a solution at all.

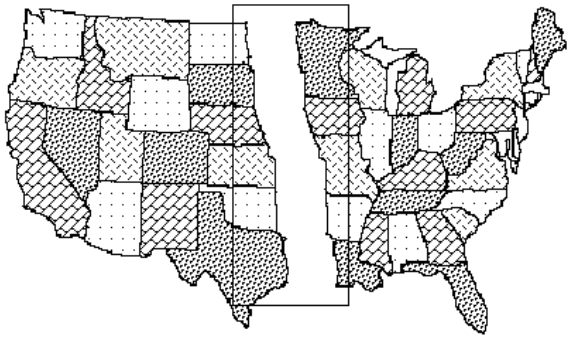


Figure 3. Merging Local Solutions

Thus, in COMPOSER, we have chosen the minimum conflicts algorithm in order to repair these initial inconsistencies using the minimum conflicts heuristic[2]. The appeal of the minimum conflicts algorithm as the adaptation algorithm is that its empirical time has been shown to grow only linearly in the number of problem variables, and that it ensures eventual convergence upon a solution if one exists.

The minimum conflicts algorithm also provides a methodology by which to assess the adaptability of the retrieved cases, which will be described in section 5. In addition, we have expanded the minimum conflicts heuristic by incorporating the possibility of dynamic constraints, thereby further broadening its applicability.

3 COMPARISON TO OTHER ADAPTATION APPROACHES

Other recent systems that have addressed the multi-case adaptation issue are EADOCS[3,4], IDIOM[5], CAPlan[6], and PRODIGY[7]. 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, and 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 the CSP during adaptation. However, dimensionality reduction is used in IDIOM in order to eliminate constraint inconsistencies, and continuous constraints are allowed, whereas the minimum conflicts repair algorithm is used in COMPOSER, 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. COMPOSER, like PRODIGY, has an implicit decomposition of the new problem, whereas in EADOCS, the decompositions are predefined based on different usages, 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, where the emphasis is on determining the adaptability of the retrieved cases. In COMPOSER, the adaptability assessment has emerged because of imposing the CSP structure onto the adaptation process, and can be determined once the set of matching cases has been retrieved, while in DEJA VU[8], assessing adaptability is being used to guide retrieval.

Other techniques that may be applicable within the framework of COMPOSER are the approach to *case abstraction* taken by PARIS, which could be especially useful in abstracting the many details of assembly sequence design cases into more general concepts that may aid the matching process. For instance, in assembly sequence design, there are several categories of assembly problems such as those that exhibit geometrical, mechanical, non-monotone, or non-linear characteristics. However, it is not clear from looking at the details of the case which of these categories the case fits into. Thus, adding an abstraction component to COMPOSER as is found in PARIS would be helpful to categorize the cases based on the domain knowledge encoded in the 'abstract planning domain'.[9] These abstracted cases may also be easier to index than would the concrete cases, since the abstracted concepts could correspond to the more general case characteristics suitable for indexing. Furthermore, the *completion rules* used in INRECA[10] might also be useable to fill in unknown details during the matching process, in order to elicit a more appropriate match.

4 ADAPTATION KNOWLEDGE REQUIRED BY COMPOSER

The difference in approach discussed in the previous section between COMPOSER and other adaptation systems also impacts the necessary adaptation knowledge in COMPOSER. 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 INRECA[11,10], for example, the required adaptation knowledge is in the form of rules which describe a possible difference between a problem and a retrieved case. In MoCAS[12], the adaptation requires a behavioral model of all components, while in DOM[13] and ToPo[14,15], the decision about what to adapt is made by referring to domain knowledge or heuristics.

In COMPOSER, the existing cases themselves provide the necessary constraints for the new problem, thus allowing 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.

The knowledge contained in the case base could also be useful for problem solving from scratch, since one could take the constraints from the existing cases and solve the CSP from scratch to come up with a solution to the new problem. However, as our results detailed in section 5 show, this approach is not as efficient as it is to use the solutions from the case base to guide the problem solving. However, the constraints contained in the case base may be useful for assessment of constrained-ness of a design, or for other assessments such as cost-effectiveness and reliability.

5 RETRIEVAL AND INDEXING VS. ADAPTATION IN COMPOSER

Adaptation is necessary in COMPOSER, again because of the use of multiple cases to solve a new problem. As discussed in the introduction, one cannot simply paste a set of local solutions together to give a consistent global solution. Thus, retrieval is not enough in domains which require combination of multiple cases, as the difficult problem of how to synthesize the global solution from a set of local solutions remains after retrieval. We have found, however, that there does exist a point at which adaptation is not cost effective. That is, there exist situations where from-scratch problem solving offers the same efficiency as does using the solutions from the case base, as is shown in Figure 4.

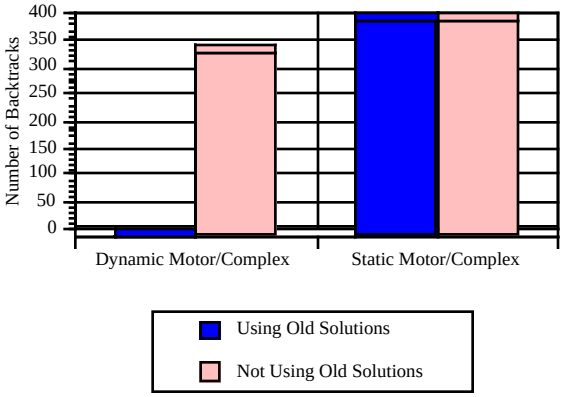


Figure 4. Cost Effectiveness of Adaptation

The deciding factor which influences the adaptability of the retrieved cases, based on COMPOSER's results, was the constrained-ness of the edge variables (the variables that overlap between the local solutions). 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. Our observations could be used to assess whether or not to attempt adaptation on the retrieved set of cases, or whether to go back into the case base to retrieve a different set of cases (i.e. a different decomposition of the problem). What we discovered through COMPOSER is that by forcing formalism onto the adaptation process, the analysis of adaptability immediately emerged as a by-product of the research. That is, one can more easily create assessments of adaptability when a formalized methodology is imposed upon the process.

6 EVALUATION

COMPOSER's adaptation methodology provides a formalized algorithm for adaptation that allows its general applicability across a wider range of problems, as it allows adaptation to apply to any problem that can be formulated as a discrete, static or dynamic CSP. Furthermore, by using the CSP methodology, COMPOSER ensures that adaptation does indeed converge upon a solution, and it provides a method by which to synthesize many cases into one new solution.

Our approach has been evaluated empirically, with three evaluation criteria: efficiency of problem solving, effectiveness of reuse, and the flexibility of the adaptation mechanism. The efficiency of problem solving was tested in regards to our problem domain: assembly sequence design. Prior approaches required extensive feasibility testing at each step of the assembly sequence planning process[16], or user involvement, in order to determine an assembly sequence[17]. By applying the case based reasoning approach with our adaptation mechanism, COMPOSER is able to solve assembly sequence problems without doing all of the feasibility calculations from scratch, and without requiring user questioning[18].

The effectiveness of the CSP as a case combination methodology is shown by the graph shown in Figure 5, which compares solving several ASPs from scratch versus solving them using the solutions in the case base. Note, however, that even when the solutions are not used, the constraints from the existing cases are used to solve the new problem. Thus, the case base provides valuable information in both instances.

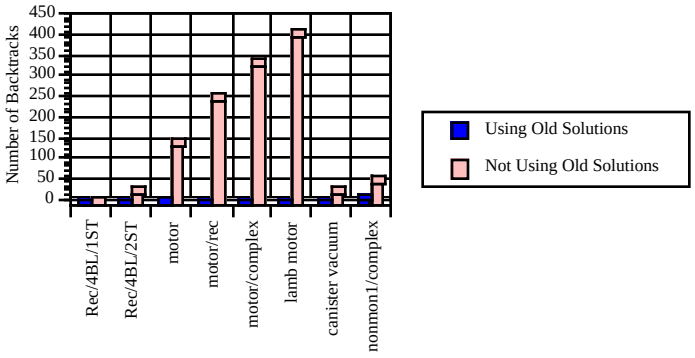


Figure 5. Using Old Solutions vs. Not Using Old Solutions

The flexibility of the adaptation mechanism stems from the dynamic capabilities of the CSP algorithm. COMPOSER is able to represent and solve assembly sequence problems which exhibit nonlinear, non-monotonic, geometric feasibility, and mechanical feasibility considerations, a wider range of problems than previous ASP approaches could accommodate.

Furthermore, we have shown that the dynamic capability also allows configuration design problems to be

easily represented, and in fact, we assert that any problem that can be formulated as a discrete, dynamic or static CSP can be solved by our approach.

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 through case combination. In this way, we hope that COMPOSER provides a methodology by which adaptation can become a more widely applicable and usable technique.

---

[1] Purvis, L. and Pu, P. *Adaptation Using Constraint Satisfaction Techniques*, in Veloso, M. & Aamodt, A. (Eds), Topics in Case-Based Reasoning Proceedings of the International Conference on Case Based Reasoning, LNAI series, Springer, 1995.

[2] 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.

[3] Netten, B.D., Vingerhoeds, R.A., Koppelaar, H., and Boullart L.: *Expert Assisted Optimization of Composite Structures*, in A. Verbraeck, E.J.H. Kerckhoffs (eds) SCS European Simulation Symp. ESS'93, Delft, October 25-28, pg.143-148.

[4] Netten, B.D., and Vingerhoeds, R.A.: *Adaptation for Conceptual Design in EADOCS*, Proceedings from the ECAI'96 workshop on case adaptation, Budapest, Hungary, 1996.

[5] Smith, I., Lottaz, C., and Faltings, B.: 1995, *Spatial composition using cases: IDIOM*, in Veloso, M., & Aamodt, A. (Eds), Topics in Case-Based Reasoning Proceedings of the International Conference on Case Based Reasoning, LNAI series, Springer, 1995.

[6] Munoz, H., and Huellen, J.: *Retrieving Cases in Structured Domains by Using Goal Dependencies*. Veloso, M., & Aamodt, A (Eds), Topics in Case-Based Reasoning Proceedings of the International Conference on Case-Based Reasoning, LNAI series, Springer, 1995.

[7] Haigh, K. and Veloso, M.: *Route planning by analogy*, in Veloso, M. & Aamodt, A. (Eds), Topics in Case-Based Reasoning Proceedings of the International Conference on Case-Based Reasoning, LNAI series, Springer, 1995.

[8] Smyth, B. and Keane, M.: Experiments on *Adaptation-Guided Retrieval in Case-Based Design*. In Veloso, & Aamodt (Eds), Topics in Case-Based Reasoning Proceedings of the International Conference on Case Based Reasoning, LNAI series, Springer, 1995.

[9] Bergmann, R., and Wilke, W.: *PARIS: Flexible Plan Adaptation by Abstraction and Refinement*, Proceedings of the ECAI'96 Workshop on Adaptation, 1996.

[10] Wilke, Wolfgang, and Bergmann, Ralph*: INRECA: Induction and Reasoning from Cases*, Proceedings of the ECAI'96 Workshop on Adaptation, Budapest, Hungary, 1996.

[11] Bergmann, R., Wess, S. Traphoener, R., & Breen, S.: *Using Background Knowledge in the Integrated System: Specification and Approach*, Deliverable D29, Esprit-Project INRECA, 1994.

[12] Bergmann, R., Pews, G., and Wilke, W.*: Explanation-based similarity: A Unifying Approach for Integrating Domain Knowledge into Case-Based Reasoning for Diagnosis and Planning Tasks*, in Topics in Case Based Reasoning, pp. 182-196, Lecture Notes in AI, Springer Verlag, 1994.

[13] Bakhtari, S., and Oertel, W.: DOM-ArC: *An Active Decision Support System for Quality Assessment of Cases*. In Aamodt, A. & Veloso, M. (eds): Proceedings of the First International Conference on Case Based Reasoning, Springer Verlag, 1995.

[14] Coulon, Carl-Helmut, *Automatic Indexing, Retrieval and Reuse of Topologies in Complex Designs*, Computing in Civil and Building Engineering, Proceedings of the Sixth International Conference on Computing in Civil and Building Engineering, A.A. Balkema, Rotterdam, 1995.

[15] Voβ, Angi, and Coulon, Carl-Helmut*: Structural Adaptation with TOPO*, Proceedings of the ECAI'96 Workshop on Case Adaptation, Budapest, Hungary, 1996.

[16] 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, 1991.

[17] T.L. DeFazio and D.E. Whitney. *Simplified Generation of All Mechanical Assembly Sequences*, IEEE Journal of Robotics and Automation, 1987.

[18] L. Purvis, *Intelligent Design Problem Solving Using Case Based and Constraint Based Techniques*, Ph.D. Dissertation, University of Connecticut, May 1995.