Complexity of Adaptation in Real-World Case-Based Reasoning Systems

Barry Smyth,
Hitachi Dublin Laboratory,
Trinity College Dublin, Dublin 2

Pádraig Cunningham,
Department of Computer Science,
Trinity College Dublin, Dublin 2

Abstract

The essence of Case-Based Reasoning (CBR) as a problem solving paradigm is that solutions are generated by adapting the solutions of similar problems rather than solving the problem from first principles. In this paper we present a categorisation of problem solving tasks, arranged according to complexity. In addition we categorise CBR systems according to the complexity of the adaptation process involved. We describe three CBR systems; a system for property valuation, a system for software design and a system for modelling in engineering analysis. We discuss the manner in which the advantage of a CBR solution to these problems shifts as the task becomes more complex and the complexity of the adaptation process changes.

Complexity of Adaptation in Real-World Case-Based Reasoning Systems

1 Introduction

Case-Based Reasoning (CBR) is a reasoning method that uses experiential knowledge, in the form of cases, to solve problems. When faced with a new problem a CBR system will retrieve a case that is similar, and, if necessary, adapt it to provide the desired solution. Its current popularity as a problem solving paradigm is representative of the shift that has occurred in automated problem solving research; researchers are beginning to move from simple, "toy" domains to complex, real world domains. With this move has come a recognition of the many inadequacies of traditional problem solving approaches, and assumptions previously deemed reasonable have proved invalid in complex, dynamic real world domains. Case-based reasoning methods have proved useful in dealing with many of these problems and have had considerable initial success in real world problem solving endeavours.

Researchers have organised problem solving into three main classes. In order of increasing complexity, these are: *simple*, *routine*, and *innovative*. Simple problem solving activities are characterised by domains whose simplicity facilitate solution generation in a fairly straightforward fashion; there is a relatively direct inference path from problem specification to problem solution. With routine problem solving, domains are more complex and often incomplete, problem specifications are conceptually quite distant from their solutions, and additional computational expense is incurred due to conflicts between dependent domain objects. The most difficult activity, innovative problem solving, is identified by patchy domain models, intricate solution paths through the problem-space, and a considerable amount of backtracking due to interaction problems.

The aim of this paper is to characterise the relationship between case-based reasoning and problem solving tasks of differing complexity. This relationship is discussed in detail in the next sections. Sections three, four and five solidify this discussion by introducing three real CBR systems concerned with simple, routine, and innovative problem solving.

2 Problem Solving Complexity and CBR

Our perspective on problem solving is concerned with search-based problem solving techniques. Such problem solvers search a space of “world models” or “problem states” (the problem-space) to find one in which all the problem goals have been achieved (the goal state). Search proceeds through the problem-space by the application of state-transforming operators. The task of the problem solver is then to find some sequence of operators that transform the initial state into the goal state. The complexity and cost of automated problem solving is well studied and there are many ways that it can be estimated. The classic measure is the amount of search which has to be carried out in developing a solution to a problem.

Essentially, there are two search related problems associated with conventional problem solving methods. The first, which we term the *operator chaining problem*[1], is basically the problem of searching forward through the problem-space to find the appropriate sequence (chain) of operators with which to solve the target problem. Relieving this problem has been (and still is) the objective of much research and has resulted in a range of relatively successful techniques (e.g., heuristic search, hierarchical planning, least commitment strategy, goal regression etc.). Further computational complexity arises from the *operator conflict problem*[2]. As a planner interacts with its domain it receives a stream of conjunctive goals. Were these goals independent of one another the cost of problem solving would be linear with the number of goals. The reality is somewhat different. Operator conflicts can result in unfavourable interactions between goals. Under such conditions the problem solver must backtrack over earlier work in order to plan a new solution which avoids these interactions. If somehow the problem solver could learn specific sequences of actions that overcome interaction problems in certain situations, the cost per goal could be reduced in the long run.

Case-based reasoning methods attempt to address both of these problems. By constraining the problem-space search, it reduces the operator chaining problem; the retrieval of a similar case constitutes a considerable 'jump' into the problem space thereby eliminating many potential routes from the search. Furthermore, CBR is potentially a more efficient constraint method than conventional approaches (e.g., heuristic evaluation etc.) whose scope is more local, pertaining to individual problem space states rather than whole areas. Backtracking due to bad interactions may also be avoided, thereby relieving the operator conflict problem. Effectively, cases can be viewed as 'canned' solutions where operator conflicts have been resolved in certain well defined situations As long as the retrieved case is sufficiently similar to the target relatively few conflicts should arise due to modifications made during adaptation.

2.1 An Overview of CBR

In CBR, problem solving knowledge is characterised as a set of episodes each representing the solution to a specific problem situation. A new problem (the *target*) is solved by retrieving a similar episode (*base case*) from memory, and its solution is then modified to conform with the target situation. Cases are represented by features and relationships and case retrieval depends on matching these case representations. It can be seen from the following descriptions of the stages in CBR that the retrieval stages is a two stage process:-

- **Representation:** Cases are represented by features and operators (relationships). The features may be surface or abstract features and typically the more predictive features will be selected for indexing.

[1]This is analogous to Marks' "Immediate Complexity" problem [Marks, Hammond, & Converse '88]
[2]This is analogous to Marks' "Asymptotic Complexity" problem.

- **Base Filtering:** A small number of candidate episodes which are considered contextually similar to the target situation are selected from the case base. The case base will often be organised as a discrimination network (D-net) to facilitate this.

- **Case Selection / Mapping:** This second stage will select a case from this candidate set based on a more detailed comparison of the cases. A mapping between the base and target cases may also be produced at this stage.

- **Adaptation:** Once the best case from the case base has been selected it must be adapted to fit the problem in hand. In the simplest situations, for instance diagnosis, this adaptation may be trivial and the base case may apply without modification. Adaptation may be very complex in more difficult problem domains, e.g. non routine design.

2.3 A CBR Perspective on Complexity

Earlier we introduced the notion of a number of types of problem solving: simple, routine, and innovative. In fact, we can view these as a continuum of problem solving tasks (Figure 1) organised in terms of computational complexity. As illustrated, the relative contribution of the operator chaining and operator conflict problems varies across this continuum. Due to the simplicity of their domains, simple problem solving tasks incur the bulk of their cost from the operator chaining problem; the lack of domain complexity leads to few conflicts so little time is spent backtracking. At the other end of the continuum, the complexity of the domains of innovative problem solving tasks results in the operator conflict problem being very prominent (and expensive to resolve), with proportionately less time spent on the operator chaining problem.

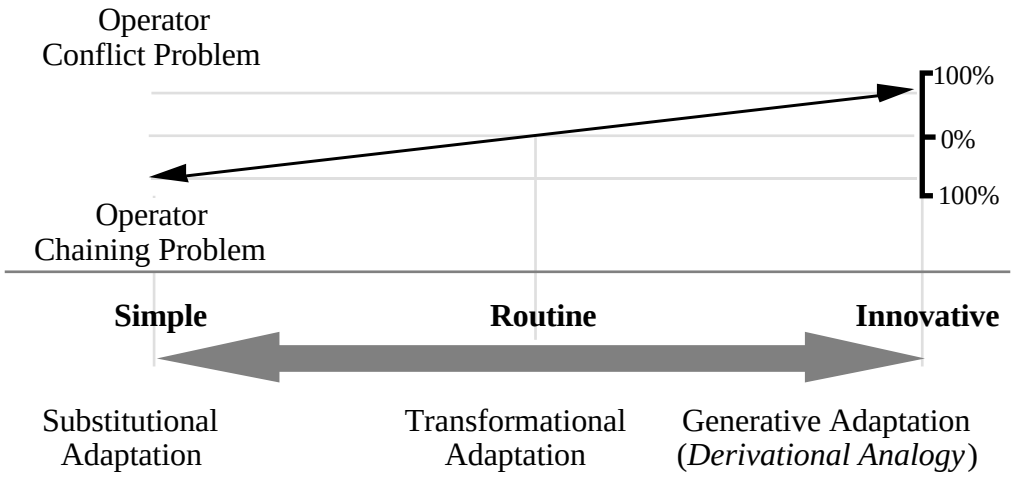

Figure 1. A continuum of problem solving tasks

The basic tenet of CBR is that, rather than solve a problem from first principles, it may be easier to retrieve a similar problem and transform the solution to that problem. In Figure 2 we attempt to illustrate these trade-offs graphically. SP' represents the specification for a new problem and SL' is the solution to that problem. FP' represents the search process that establishes this solution from

first principles - the task we wish to avoid. A CBR solution is worthwhile if the retrieval task R, and the adaptation task A are simpler than FP'. In the later sections of this paper we will examine three CBR systems that differ in the complexity of the transformation task A:-

- **Substitution Adaptation:** This is the simplest type of adaptation and merely involves substituting some of the parameters in the solution.

- **Transformational Adaptation:** This adaptation is more complex and will involve structural changes to the solution.

- **Generative Adaptation:** This is the most complex adaptation and is not perfectly represented by the diagram. The adaptation process involves a rework of the reasoning process FP in the context of the new problem situation represented by SP'.

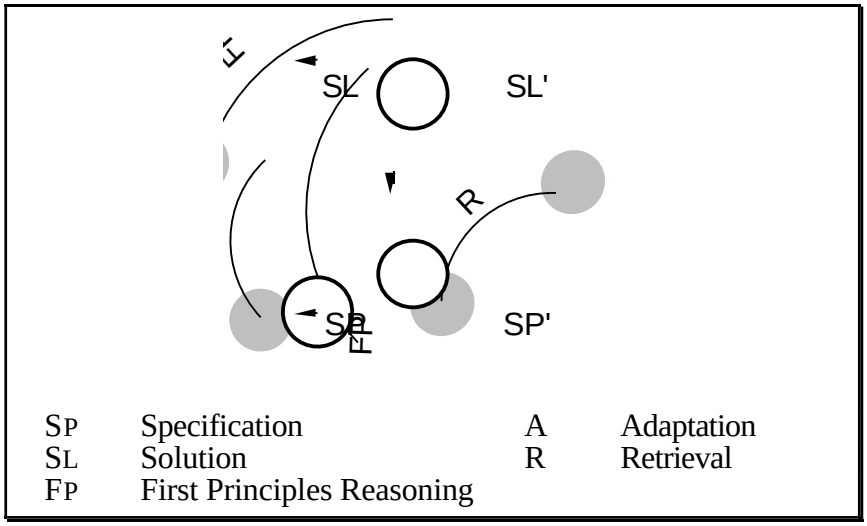

Figure 2. The transformation processes in CBR and in reasoning from first principles.

From a case-based reasoning perspective there is a relationship between the complexity of the problem solving task and the complexity of the CBR adaptation process. CBR systems concerned with simple problem solving tasks will typically require simple, substitutional adaptation; the retrieved case will typically be very close to the target and consequently will require only very basic substitutive adaptation, with little or no chance of bad interactions occurring. With routine problem solving tasks, the retrieved case, while being similar to the target , will probably require substantial modification possibly changing some structural elements of the retrieved solution (transformational adaptation), and consequently bad interactions are likely to occur. With more complex, innovative problem solving tasks extensive modifications are necessary during adaptation so interactions are unavoidable and difficult to resolve. To accommodate this adaptation complexity it is generally necessary to augment cases with detailed knowledge such as the reasoning trace structures of Carbonell's Derivational Analogy.

3 Rachman: Substitutional Adaptation

Simple problem solving tasks are characterised by well-defined domain models, and a relatively direct inference path from problem specification to problem solution. Case-based reasoning methods are particularly well suited to such tasks as there is often a rich set of cases to draw upon during problem solving. The retrieved case is typically very similar to the target problem and consequently the adaptation problem is relatively uncomplicated; in general modifying the attributes of solution elements will suffice without the need to change the overall solution structure.

3.1 A Brief Overview of Rachmann

Figure 3 shows two example cases from a case-based system called Rachman that can predict the selling value of a house given some details about it. The system contains a large case base of houses and their selling prices and, given a house description, it will retrieve a case or a set of cases describing similar houses and their selling prices. These prices can be adjusted depending on differences between the target and base cases to estimate the price of the target house. This system is comparatively straightforward but is equivalent to a host of potential CBR applications, for example loan risk assessment and help-desk assistants. The complexity of this problem is greatly reduced by having a well populated case base, so that good matches can be found and the required adaptation is not difficult.

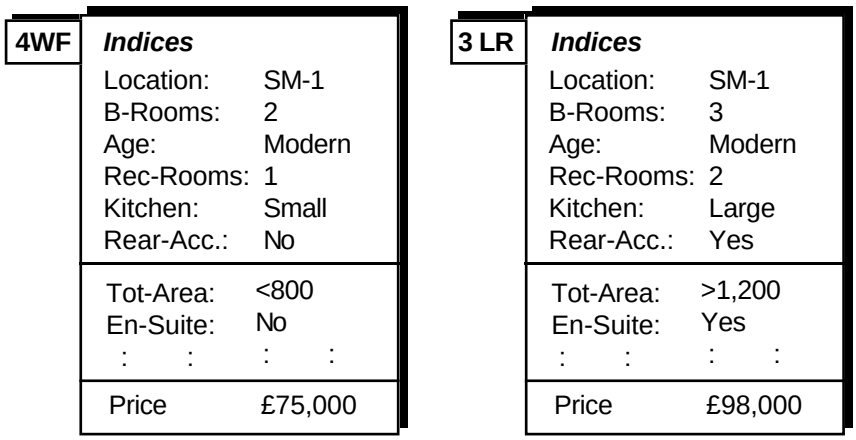

Figure 3. Two sample cases from the Rachman Case-Base

The cases are divided into two sets of features, the index features and the internal features. The index features are the most strongly predictive features and form the basis for the D-Net (Figure 4). The main problem with the D-Net approach is that it forces a strict ordering of the index features, in this example the cases might be organised first under location, then number of bedrooms, etc. However, different users may have different priorities; some, for instance, might consider the number of bedrooms to be more important than location. In addition, it will not be possible to retrieve matches for cases that have missing features as the retrieval process will not be able to search below the level of that feature in the network.

These problems are largely solved by introducing redundancy into the D-Net. This means that the network supports alternative orderings on the index features (see Figure 4). The extent to which redundancy can be introduced into the network is limited because the size of the network grows in proportion to the number of orderings supported. Retrieval in Rachman returns clusters of cases and the cases are ranked according to their frequency of occurrence in these clusters.

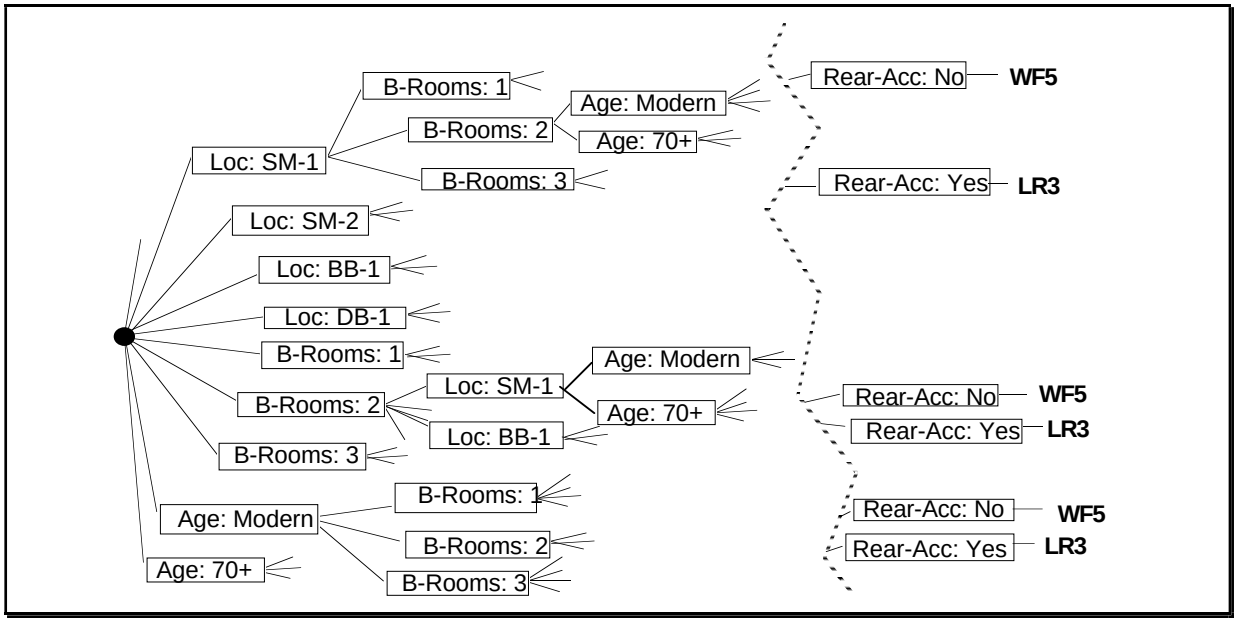

Figure 4. A portion of the Rachman Case-Base organised as a D-Net with some redundancy.

3.2 Substitutional Adaptation in Rachmann

It is assumed with this system that the case base is well populated so that near or exact matches can be found. This means that the adaptation is comparatively straightforward using substitution specialists that adjust the price based on differences between the base and target cases. For instance a substitution specialist may add or remove the value of having a conservatory. The important point is that the effect of this adjustment is quite local and does not cause complex interactions.

4 Déjà Vu: Transformational Adaptation

In a CBR system for routine problem solving, retrieved cases, while being comparatively similar to the target, will probably require substantial structural modification. Consequently a limited opportunity exists for bad interactions to occur. Carbonell (Carbonell, 1983) characterises such problem solvers as belonging to the realm of *Tranformational Analogy*. Essentially, the adaptation stage can be considered as a solution transformation problem. Finding the appropriate transformation is itself a problem-solving process, but in a different problem space. The states of this transformation-space (T-space) are problem

solutions, as opposed to solution states. The transformation-space operators (T-operators) are atomic solution transformation operators (e.g., Substitute Solution Step, Delete Solution Step, Insert Solution Step, Reorder Solution Steps etc.). And the job of adaptation is to find the appropriate sequence of T-operators which will transform the retrieved solution into the desired target solution (See Figure 5).

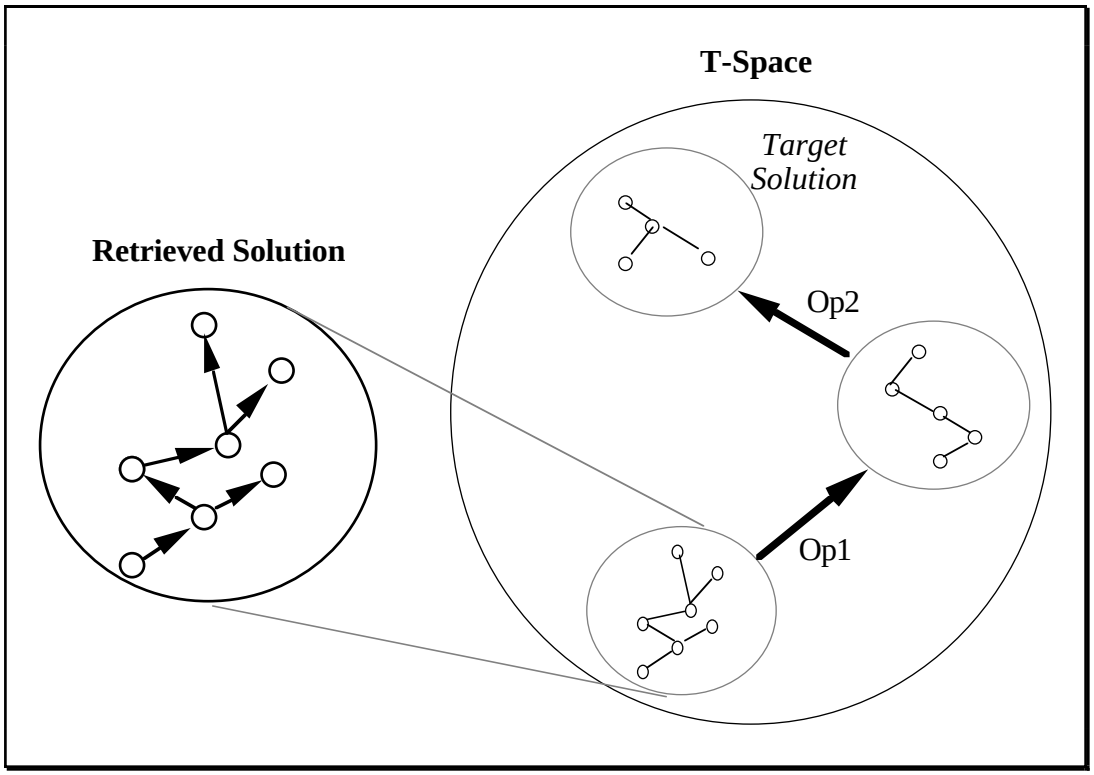

Figure 5. Transformational Adaptation.

In this section we discuss Déjà Vu, a case-based reasoning system for software design (a routine problem solving task), paying particular attention to its adaptation mechanism.

4.1 An Brief Overview of Déjà Vu

Déjà Vu's application domain is that of Plant-Control software (see Smyth & Cunningham, 1992a, 1992b); it is concerned with producing control software for autonomous vehicles in a plant or mill environment.

Figure 6 illustrates an important class of programs in a steel-mill environment that are concerned with controlling vehicles in the loading and unloading of metal coils during the milling process[3]. Déjà Vu's case-base consists of control programs for carrying out tasks such as these Load/Unload operations. The

[3]Tension Reels and Skids are used to hold spools or coils of steel during the milling process. Coil-cars are vehicles which can carry spools or coils about and perform load/unloading operations.

programs are written in a high-level, graph-based language which can be compiled into executable code.

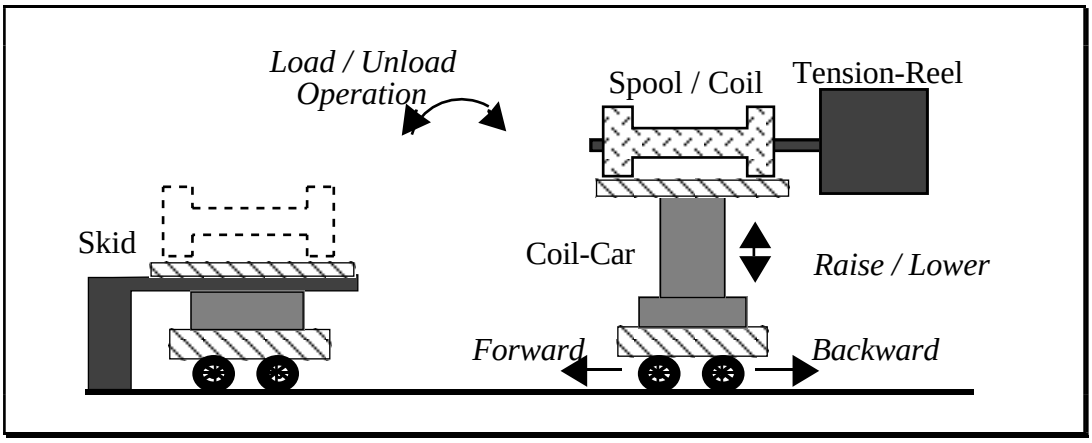

Figure 6. Load / Unload Plant-Control Tasks

Briefly, Déjà Vu uses a hierarchical approach to software design; during problem solving a number of cases are retrieved at different levels of abstraction, and these are adapted to provide solutions to the various sub-tasks of the target problem. Problem solving activity is co-ordinated using a blackboard control architecture with dedicated knowledge sources handling the various problem solving stages of analysis, problem decomposition, retrieval, adaptation, and solution integration.

4.2 Transformational Adaptation in Déjà Vu

Déjà Vu adopts a two stage approach to adaptation; *adaptation specialists* are used to transform components of the retrieved solution in the desired component of the target while more general *adaptation strategies* are used to handle any bad interactions arising out of specialist activity. Both types of adaptation knowledge are based on a primitive set of transformation operators.

During retrieval the features of the target case are mapped against the features of the base case. Non-exact matches provide the adaptation process with a set of conjunctive adaptation goals which must be satisfied in order to transform the base solution into the desired target solution. During adaptation these goals are handled by the appropriate set of specialists, each making its own local changes to the base solution structure.

As an example of the type of adaptation carried out by Déjà Vu consider the following scenario. The target specification is given as "MOVE BUGGY BACKWARD TO SKID USING 2 SPEEDS". The retrieval stage selects a base case conforming to the following specification, "MOVE BUGGY FORWARD TO TENSION-REEL USING 1 SPEED". Figure 7 shows the mappings between the target features and the base features that are computed during retrieval.

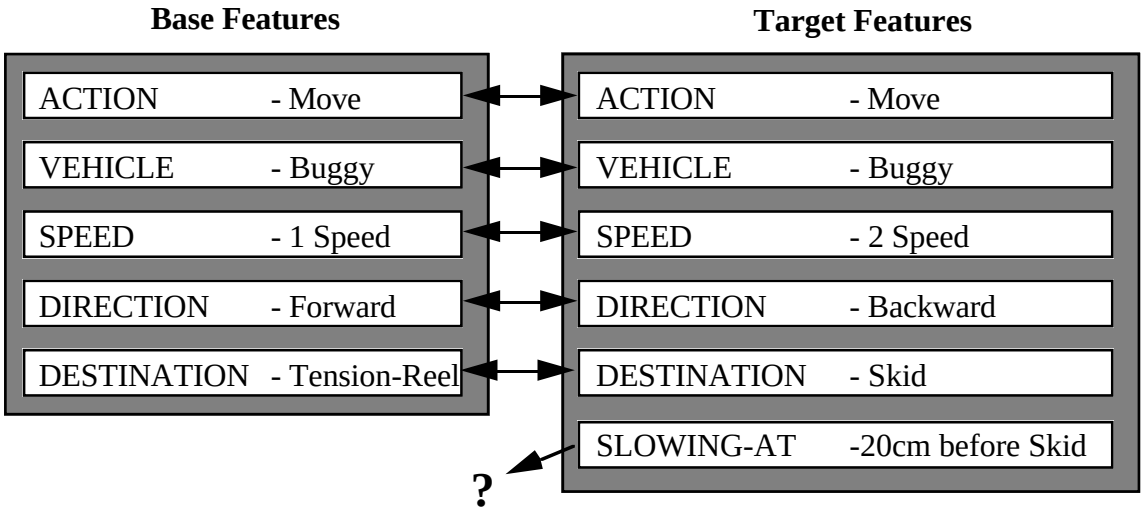

Figure 7. The mappings generated between the base and target features.

Clearly the direction and destination of the base case solution must be altered, and a command must be added to accommodate two-speed motion. While the direction and destination modifications are simple substitutive changes (like in Rachman), the accommodation of two speed motion is somewhat more complex. In the base case the buggy travels at a constant speed, but in the target it must travel some of the distance at its fast speed and then decelerate to its slower speed before reaching its destination and stopping. This means adding extra solution components, not just modifying existing ones.

Déjà Vu's adaptation specialists are organised according to the type of modification that they can perform. For example, there is a speed specialist capable of transforming a 1-Speed motion case into a 2-Speed motion case (see Figure 8). In this way the mappings set up during retrieval are used to select the appropriate adaptation specialists.

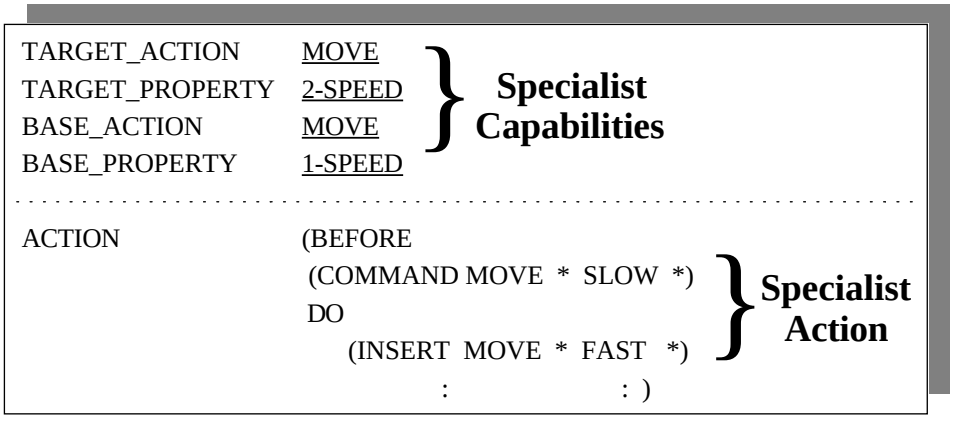

Figure 8. An example SPEED specialist

Figure 9 shows the "before-and-after" view of the adaptation process; the direction and destination attributes have been modified and the additional speed nodes added.

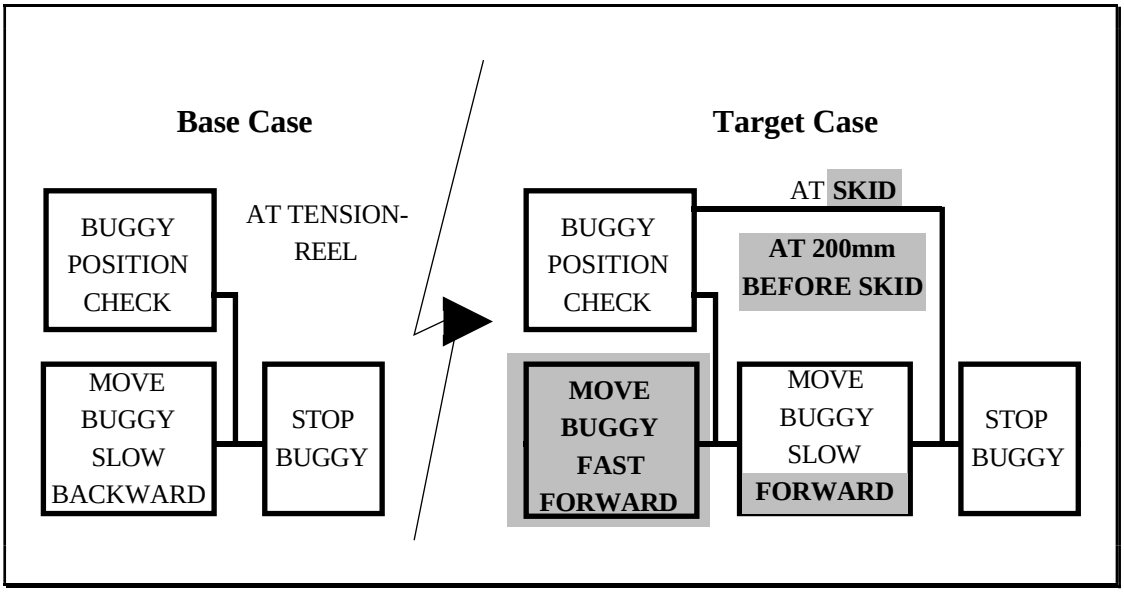

Figure 9. The base to target transformation.

Although we did not discuss interaction problems in the example they can easily occur. For example, a buggy's fuel restriction would mean that an increase in speed would require a corresponding increase in fuel. Rather than encoding such knowledge into its adaptation specialists, Déjà Vu uses a set of strategies to capture the abstract dependencies that exist between features such as speed and fuel, and during adaptation these strategies will cater for any goal interactions that arise.

5 CoBRA: Generative Adaptation

Innovative problem solving is the most difficult task category that we consider. We argue that, in a CBR system for tasks of this complexity, it is not enough to transform the solutions of existing problems. Instead the adaptation process must re-enact the reasoning trace of similar cases. This is generative adaptation or derivational analogy as presented by Carbonell [Carbonell '86].

5.1 A Brief Overview of CoBRA

CoBRA is a CBR system for creating physical models in engineering analysis. The task being addressed is the generation of simplified models suitable for numerical analysis. This process of model simplification is an important initial stage in thermal analysis in engineering. The objective is to produce a simplified geometric model suitable as a basis for a mathematical model. This simplified model must be a reasonable approximation to the actual physical system. For the human designer this process involves a series of assumptions and justifications that produce the simplified model.

CoBRA has been implemented to work on a case base of cooling fins, a typical example of which is shown in Figure 10 Each case is made up of a representation of the basic model, the simplified model and a reasoning trace of the

justifications for the transformations in going from the basic to the simplified model.

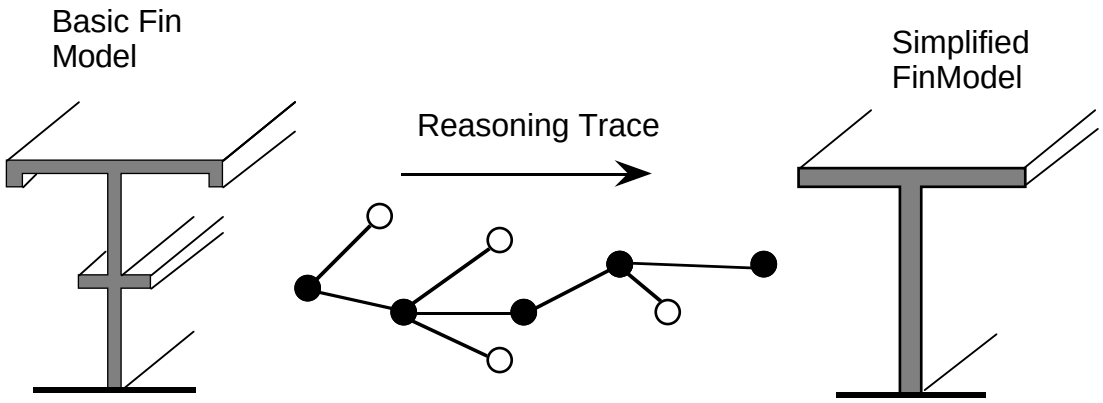

Figure 10. Example case from CoBRA, the thermal modelling system

5.1 Generative Adaptation in CoBRA

A modelling episode with CoBRA begins with a target case containing only the basic model to be simplified. The retrieval mechanism retrieves a ranked set of cases that have similar basic model features. Each of these retrieved cases contains a reasoning trace; this trace is the heart of the generative adaptation process. This adaptation process attempts to reinstantiate these reasoning traces using the target case.

Each reasoning trace has an action part and a decision part (after [Carbonell '86]). The decision part contains:-

- Alternatives considered and rejected
- Reasons for decisions taken
- Starts of false paths
- Dependencies of later decisions on earlier ones

The action part holds the steps taken as a result of the reasons held in the decision part. The actual functions used to express these actions must be abstract enough to allow their application to cases similar to the one with which they are stored. A typical action would be, "Remove the extended surface which faces into the flow". The two main actions in CoBRA are REMOVE and RESIZE. Both the decision and action parts use a set of operators which are coded to be useful over a range of model fins, e.g.:-

- **altitude:** the altitude of the shape
- **towards-back:** true if the shape is towards the back of the shape
- **area-of-face:** Return the area of the face.
- etc.

In summary, each reasoning trace is made up of a set of actions and justifications for those actions. The reasoning trace can be reinstantiated for the new case if these justifications are valid in the new situation.

6. Conclusion

We have described a categorisation of problem solving tasks arranged according to increasing complexity. To parallel this we have presented descriptions of different types of adaptation in CBR that are appropriate for tasks of different complexity.

It is often argued that the great advantage of CBR is in the ease of setting up the knowledge base of CBR systems. This advantage is manifest in the Rachman system described here. However, CBR systems for more complex tasks do not enjoy this advantage to the same extent. The case representation becomes more elaborate as the adaptation task becomes more complex. The example systems for transformation and generative adaptation described here illustrate that CBR can make a contribution in complex domains. However, these complex adaptation strategies need to be backed up with more complex case representations and a stronger domain theory.

An alternative view of CBR is as a technique for constraining the computationally expensive search associated with conventional problem solvers. As well as reducing the search through the problem space, CBR facilitates a reduction in the time spent backtracking due to interaction problems. The example systems have varied in the amount of search they have required, and real efficiency improvements have been observed when compared to first-principles approaches, especially in innovative problem solving tasks.

References

[Carbonell '83]
Carbonell, J. G., "Learning by Analogy: Formulating and Generalising Plans from Past Experience.", in *Machine Learning*, Vol. 1, ed. Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, Morgan Kaufmann, pp. 137 - 161, 1983.

[Carbonell '86]
Carbonell, J. G., "Derivational Analogy: A Theory of Reconstructive Problem Solving and Expert Acquisition", in *Machine Learning*, Vol. 2, ed. Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, Morgan Kaufmann, pp. 371 - 391, 1986.

[Marks, Hammond, and Converse '88]
Marks, M., Hammond, K., and Converse, T. "Planning in an Open World: A Pluralistic Approach, in *Proceedings of the Case-Based Reasoning Workshop,* Florida, U.S.A., ed. Janet Kolodner, Morgan Kaufmann, pp. 271 - 285, 1988.

[Smyth & Cunningham '92a]
Smyth B., Cunningham P., "A Blackboard-Based, Multi-Stage Case-Based Reasoning System for Software Development", in *Proceedings of 5th. Irish Conference on Artificial Intelligence and Cognitive Science,* Limerick, Ireland, 1992.

[Smyth & Cunningham '92b]
Smyth B., Cunningham P., "Déjà Vu: A Hierarchical Case-Based Reasoning System for Software Design", in *Proceedings of 10th. European Conference on Artificial Intelligence,* Vienna, Austria, ed. Bernd Neumann, Wiley & Son, pp587-589, 1992.