Improving Competence by Integrating Case-Based Reasoning and Heuristic Search

Zdenek Zdrahal, Enrico Motta

Knowledge Media Institute
The Open University
Walton Hall,
Milton Keynes, MK7 6AA
United Kingdom
{z.zdrahal, e.motta}@open.ac.uk

Abstract. We analyse the behaviour of a Propose & Revise architecture in the VT elevator design problem and we show that this problem solving method cannot solve all possible cases covered by the available domain knowledge. We investigate this problem and we show that this limitation is caused by the restricted search regime employed by the method and that the competence of the method cannot be improved by acquiring additional domain knowledge. We therefore propose an alternative design problem solver, which integrates case-based reasoning and heuristic search techniques and overcomes the competence-related limitations exhibited by the Propose & Revise architecture, while maintaining the same level of efficiency. We describe four algorithms for case-based design, which exploit both general properties of parametric design tasks and application specific heuristic knowledge.

1 INTRODUCTION

Design problems are hard due to the potentially combinatorial explosion of the design space. Designers, both human and artificial, tackle the complexity of the design process by making use of both task-specific heuristics as well as case-based knowledge. The *Propose & Revise* (P&R) problem solving method (Marcus and McDermott, 1989; Yost and Rothenfluh, 1996; Zdrahal and Motta, 1995) is a well known example of an approach to design problem solving which uses heuristic knowledge to reduce the complexity of the search space. In this paper we critically review the competence of this method by analysing the behaviour of a Propose & Revise problem solver in the VT elevator design domain (Yost and Rothenfluh, 1996) and we show that such a problem solver could not solve all the 'obvious' cases made possible by the available domain knowledge. The reason for this limitation is that the problem solving model (not the domain knowledge!) provided by domain experts and described in (Yost and Rothenfluh, 1996) is incomplete. In particular, the search algorithm used by the model is a greedy, local search procedure and therefore it rejects any search step which does not decrease the number of constraint violations, even in cases where such a search step is part of a path leading to a solution design. A similar pattern of behaviour was reported by (Selman et al., 1992) when carrying out experiments on the satisfiability of a set of propositional clauses. Selman shows that by introducing 'sideway moves' into the search algorithm the competence of the algorithm improves from about 40 – 60% to 100% and the time consumed by the algorithm dropped to about 10% of the original value. In our example, we cannot produce statistically meaningful results as we have not evaluated a large enough data sample, but we can instead provide a real-world interpretation of the troublesome cases. They seem to arise when solving the machinery subsystem of the elevator. This is indeed the most complicated part of the elevator design problem, whose solution requires either (i) more sophisticated search strategies, which can be expensive, or (ii) the use of an alternative problem solving paradigm.

Our solution to this problem combines both of the above mentioned approaches and is based on the integration of case-based reasoning and heuristic search techniques. In particular we used a best-first search algorithm to solve the ‘difficult cases’, which were then stored in a case library, and we replaced the original Propose & Revise problem solver with a case-based one.

In the paper we describe four alternative algorithms for performing case-based design and we show how we were able to integrate the pre-existing search control knowledge, originally acquired in the context of a Propose & Revise approach, with a case-based approach to design. These algorithms exploit both general properties of parametric design as well as domain specific aspects of the VT problem. In particular we used our knowledge of the relative complexity of the various subsystems of the VT domain and we designed the case-based algorithms so that the case retrieval component prefers cases whose difficult subsystems do not need any modifications. The rationale for this approach is that subsystems which are difficult to design are also difficult to adapt.

The heuristic knowledge used for adaptation, originally acquired in the context of the Propose & Revise problem solver, makes it possible to estimate the minimum and the maximum cost of adaptation. Since design is an optimisation task, the upper and lower bounds of the adaptation cost can be used for filtering out cases which cannot minimise the cost criterion. Here, adaptation is broken down to elementary corrective steps. Each step represents a modification of the case which increases the cost of the solution. The control of the adaptation algorithm applies an A* like strategy which makes use of the results and costs of the elementary corrections.

These observations, which hold in many design domains, provide the ‘design philosophy’ of the algorithms described in this paper.

2 PARAMETRIC DESIGN

The elevator design application is an example of a *parametric design* problem. Parametric design problems are a class of design problems where the solution can be described in terms an assignment of values to a set of *parameters*. This assignment typically has to satisfy a number of *design constraints* and *requirements*, and should achieve (or at least approximate) some *quality criterion*. A (possibly incomplete) set of parameter assignments is called a *design model*. The set of all design models is called a *design space*.

A constraint specifies a condition which must not be violated by a consistent design. For instance, the VT elevator design application includes constraints such as “The cab height must be between 84 and 240 inches, inclusive”. Requirements are also constraints and, as discussed in (Wielinga et al., 1995), the difference between requirements and constraints is conceptual rather than formal. Requirements describe the desired properties of a solution, while constraints limit the space of valid designs. Moreover, constraints normally specify case-independent restrictions, while requirements are typically case-specific. For instance, a requirement in the VT application specifies the desired capacity of the target elevator, which is of course case-specific.

In accordance with their role in the design process we can distinguish between *restrictive* and *constructive* constraints. Restrictive constraints eliminate certain combinations of parameter values. Constructive constraints define legal combinations of parameter values. They can be expressed in a functional notation which indicates the expected way of using the constraint in a design process.

The distinction between constructive and restrictive constraints allows us to define a class of parameters which are calculated by constructive constraints. The parameters which cannot be calculated by constructive constraints are called *key design parameters*. Key design parameters and their possible values define the degrees of freedom in the design process and, consequently, the size of the design space. In other words, the essential task during the design process is to decide upon the values to be assigned to key parameters; the values of the other parameters can then be determined by propagating the relevant functional (constructive) constraints.

In many cases we might have *preferences* over the possible values for a particular parameter. For instance, if cost is a concern, which is normally the case, we might wish to select the cheapest component which is suitable for the job. To account for this kind of knowledge our framework includes the notion of 'preference'. Preferences can be divided into two classes: *local* and *global* preferences. Local preferences express knowledge which defines a *preference order* over alternative values of a parameter. For instance, a criterion which orders motor models from the cheapest to the most expensive is an example of a local preference. Global preferences express instead desired features of a (subset of a) design solution. For example, in the Sisyphus-I room allocation problem (Linster, 1994), one of the preferences stated by the (virtual) domain expert is that "the allocation of researchers to rooms should maximise the synergy between different projects". By means of this criterion we can assess and compare two design solutions (i.e. two distinct room allocation models) in terms of those rooms which are shared by researchers.

As it can be induced from the above discussion, design is usually an optimisation problem. We are not satisfied with any design which satisfies requirements and constraints, but we want to achieve (or at least approximate) an optimal one. The *optimisation criterion* is often complex and may include a number of elements, such as the cost of the artefact, the cost of the design process, expected maintenance costs, customer satisfaction measures, etc. Because the only degrees of freedom we have are the design choices associated with key design parameters, the optimisation criterion must be based on the values of key design parameters. Typically, the optimisation criterion is characterised as a *generalised cost function* formalising preference knowledge and expressing a mapping from design solutions to real numbers. We assume that a lower score denotes a better solution.

In conclusion, the input to a parametric design task is given by: (1) a set of parameters, some of which are characterised as key design parameters; (2) a set of design requirements, (3) a set of constraints, (4) a set of values associated with key parameters; (5) local and global preferences; and (6) an overall optimisation criterion. The goal is to produce a complete design model which satisfies all requirements, does not violate any constraint, and best approximates the optimisation criterion.

3 PROPOSE & REVISE

The generic task specification given in the previous section allows us i) to define the design space and ii) to distinguish solution from non-solution designs. In order to solve a particular design task we have to decide how we are going to navigate the potentially very large space of possible designs. In other words we need to specify our *problem solving method*.

Design problem solving often follows a Propose & Revise approach. P&R problem solving consists of three main subtasks, which are: (1) proposing a *model extension*, i.e. selecting an unassigned parameter and calculating its value, (2) evaluating the current design model against the set of applicable constraints and requirements, and (3) revising an inconsistent design model. Model extensions are proposed by means of *design procedures*, while *fixes* are used during the revision stage to modify inconsistent designs. This generic framework can be operationalized into two fully specified problem solving methods which we call *Extend-Model-then-Revise* (EMR) and *Complete-Model-then-Revise* (CMR). EMR tests and fixes constraint violations after each 'propose' step; in other words, violations are resolved as soon as they are generated. In contrast with this approach, CMR first assigns values to all parameters, thus producing a complete design model, and only then attempts to fix constraint violations. A detailed analysis of P&R problem solving methods is presented in (Zdrahal and Motta, 1995).

Although this framework was proposed originally for the elevator design problem, it can be applied to other design situations. For instance, it has been used for an initial vehicle design application (Banecek and Drvota, 1995) and for sliding bearing design (Horak et al., 1995).

In the rest of the paper we will use the expressions *task knowledge* and *problem solving knowledge* to refer to concepts or procedures introduced respectively as part of a task or problem solving method specification. Both task and problem solving knowledge can be *generic* (e.g. the concept of parameter) or *domain specific* (e.g. parameter motor model in the VT domain).

4 THE ROLE OF DESIGN PROBLEM SOLVING METHODS

As already pointed out, the design process consists of assigning values to key design parameters, in such a way that the final result is a solution design, approximating as much as possible the optimisation criterion. The design process is driven by the choice of a problem solving method, which essentially fulfils two jobs:

- it provides a conceptual framework for re-interpreting task knowledge in a process-oriented way;
- it provides a mechanism for searching the design space.

For example, the P&R framework re-interprets knowledge about the values available for a particular parameter in terms of fixes and procedures. For instance, let's consider the parameter 'motor model' in the VT domain. The range of possible values for this parameter is a list of motor models, (motor-1, motor-2, ..., motor-n), ordered in accordance with their size. This order also reflects the torque and cost of the motors. This information is operationalized in terms of the P&R framework in the following way: a procedure will select the cheapest motor model providing the required torque during the 'propose' phase, while a fix will upgrade it, if the original choice turns out not to be compatible with the other components of the design. The advantage of this approach is that it compiles knowledge about design choices and preferences in such a way that choices are always locally optimal. This is a very common heuristic which often gives a good solution.

While the EMR and CMR architectures differ in the way they intertwine propose and revise steps, they both use the same approach when applying fixes. These are applied in such a way that if a fix removes the violated constraint without producing a new constraint violation the change is made permanent, otherwise the fix is rejected. The rationale for this generic heuristic is to reduce the size of the search space and therefore the complexity of the design process. However, as discussed in detail in section 6, this approach causes problems: it not only reduces significantly the competence of the problem solving method but, even worse, makes it unpredictable. Moreover, since in the EMR approach constraints are evaluated as soon as all required values are available, a consequence of this approach is that the order in which parameters are assigned becomes important. For instance, the simple example in figure 1 shows that we can achieve two different results for different orders of evaluation.

This simple network has three parameters, a, b and c; two constraints, C1 and C2; and two fixes, fix-1, associated with C1, and fix-2, associated with C2. Parameter a has a required value, say a = 1; b and c are key design parameters whose values are to be calculated. The initial choice is b = 1, c = 2.

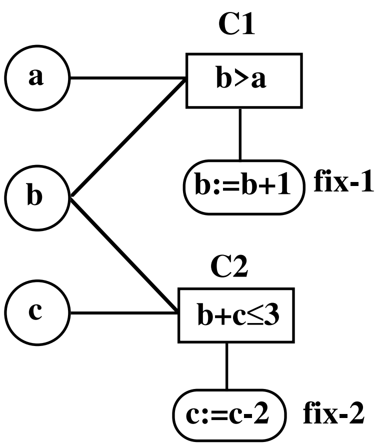

Figure 1. A simple network

There are two possible orders for calculating values: {b, c} and {c ,b}. Because of the way search is restricted in the EMR and CMR methods, the two orders produce two different results:

(a) Sequence {b, c}
step 1: choose b = 1;
step 2: Since a = 1, constraint C1 violated;
step 3: fix-1 is applied, b = 2, and constraint C1 is satisfied.
step 4: choose c = 2;
step 5: Since b = 2, constraint C2 ($b + c \leq 3$) is violated;
step 6: fix-2 is applied, c = 0, and constraint C1 is satisfied, **success**.

(b) Sequence {c, b}
step 1: choose c = 2;
step 2: choose b = 1;
step 3: Since a = 1, constraint C1 violated;
step 4: fix-1 is applied, b = 2, constraint C1 is satisfied but the fix application results in a new constraint violation – C2, therefore fix-1 is rejected, **fail**.

Sequence (a) produces the solution because constraint C2 cannot be evaluated when C1 is being fixed and therefore the new constraint violation cannot terminate the search process. Sequence (b), which is computationally equivalent to the CMR method, fails. This simple example shows that a consequence of the restricted search mechanism used by the P&R approach is that solution designs are not necessarily produced and that the order in which parameters are assigned becomes important. A possibility therefore could be that in order to use EMR it is necessary to acquire the relevant knowledge about the order of parameter assignments. However, we find this quite unlikely. First of all, no such knowledge is provided in the description of the VT application (Yost and Rothenfluh, 1996). Moreover, in the general case, it is difficult to imagine that such a control is meant to be part of a designer's problem solving knowledge. Of course, it is likely that designers use domain dependent sequences of design steps. For instance, in the case of the VT domain, we may expect that they first calculate and test all building related parameters and constraints and then proceed with designing the machinery only if the building is suitable for elevator construction. But, the search control we showed in the simple example has a different nature. It artificially reorganises parameter calculation in order to compensate another artificial assumption which says that fixes are rejected when they produce a new constraint violation, even if the original constraint violation has been solved. We may speculate that in such cases the designer would rather allow for a new constraint violation which can be fixed later.

In conclusion, the P&R approach as described in (Yost and Rothenfluh, 1996) provides a conceptual framework for operationalizing task knowledge in a process-oriented way. However, it restricts search in a way which dramatically affects the competence of the problem solver. In the next section we illustrate this point by showing the behaviour of a P&R problem solver in the VT application.

5 THE COMPETENCE OF A P&R-BASED VT PROBLEM SOLVER

An elevator in the VT domain has three main subsystems: i) the driving machinery (e.g. motor, gears, cables, counterweight), ii) the building-related components (e.g. location of the machinery, sheaves, safety, buffers, platform), and iii) the support equipment (e.g. car lantern, phone, door type). Design requirements are expressed as an initial assignment of 26 parameters and typically affect more than one subsystem. For instance, the requirements related to the machinery design include the speed and capacity of the elevator, which also affect building-related components. Similarly, some requirements affecting the building-related design also affect the machinery, e.g. the shaft dimensions, the number of floors etc. The interactions between the subsystems are expressed by constraints and common parameters.

The machinery parameters, which are constrained by physical laws, have less degrees of freedom in selecting consistent values than the other subsystems which are typically constrained only by not-so-strict technological rules. Consequently much more effort is needed to find a consistent solution for the machinery subsystem than for the other components of the elevator design. Without a good search strategy many reasonable design requirements cannot be satisfied. We tested the P&R algorithm on input specifications generated by combining all legal values of speed and capacity from (Yost and Rothenfluh, 1996). The values of the remaining 24 required parameters were always the same and correspond to the test case in (Yost and Rothenfluh, 1996). The results are displayed in table 1 which shows that the P&R problem solver is unable to provide solutions for more than 50% of the possible inputs, for details see (Motta and Zdrahal, 1996) .

	Speed [ft/min]				
Capacity [pound]	**200**	**250**	**300**	**350**	**400**
2000	**Success**	**Success**	**Fail**	**Success**	**Success**
2500	**Fail**	**Fail**	**Success**	**Success**	**Success**
3000	**Fail**	**Success**	**Success**	**Success**	**Success**
3500	**Success**	**Fail**	**Fail**	**Fail**	**Fail**
4000	**Fail**	**Fail**	**Fail**	**Fail**	**Fail**

Table 1. Competence of the P&R based solution to VT

Even worse, the results are not consistent with our intuition that if a solution (i.e. motor) is found for a certain speed and capacity, then some solution will also be found for the same speed and lower capacity, and for the same capacity and lower speed. A simple physics indicates that the solutions should be restricted by the hyperbola $mv \le Const$, where m is the elevator mass (i.e. capacity), v is the elevator speed, and *Const* is a constant proportional to the maximum torque generated by the biggest motor. The hyperbola expresses that the momentum (mv) of the moving components must not exceed the maximum torque which can be generated by the biggest available motor.

In order to improve the competence of our problem solver we tried a more powerful (and computationally expensive) best-first search algorithm, guided by cost-based heuristics. The results are shown in table 2.

	Speed [ft/min]				
Capacity [pound]	**200**	**250**	**300**	**350**	**400**
2000	**Success**	**Success**	**Success**	**Success**	**Success**
2500	**Success**	**Success**	**Success**	**Success**	**Success**
3000	**Success**	**Success**	**Success**	**Success**	**Success**
3500	**Success**	**Success**	**Success**	**Fail**	**Fail**
4000	**Success**	**Fail**	**Fail**	**Fail**	**Fail**

Table 2. Competence of the A* based solution to VT

These results are intuitively acceptable. The problem is that the search algorithm for those combinations which failed in table 1 and succeeded in table 2 is very slow. For example the combination [speed = 200 ft/min, capacity = 4000 pounds] takes 104 min on our 64 Mb Sparc 5 running Harlequin Common Lisp. For the failed combinations in table 2 the program runs out of memory and it is not clear whether the solution does not exist or whether our computational means are not powerful enough.

6 WHY DOES P&R PERFORM SO BADLY? DISCUSSION

The described experiments confirm what discussed in the previous section: the P&R approach used to design the elevator is not only incomplete, but exhibits a slightly bizarre behaviour. One possibility could be that the knowledge base is incomplete, in the sense that maybe we just need to acquire additional fixes. However, we believe this is unlikely. A more powerful, A* based search engine can do much better with the existing body of knowledge. Procedures assigning initial values to key design parameters and fixes are not born in heaven but they operationalize pre-existing task knowledge. For example, the procedure and the fixes for selecting the motor model parameter, MM, have also task-level declarative semantics. Viewed as a constraint specification mechanism they say that the constraint (MM=motor-1) $\vee$ (MM=motor-2) $\vee$ ... $\vee$ (MM=motor-n) must be satisfied. In other words we know in advance that the motor model parameter will have to have a value from the predefined list of available motors. Fixes only specify how to select the next element from the list i.e. how to make the next step in a right direction. In a nutshell we do not believe that the answer to the problem consists of acquiring some additional context-dependent, domain-specific heuristics.

Another possible explanation is that P&R faithfully models expert's problem solving behaviour and therefore produces identical results including the failures. Again, we believe this is very unlikely. It is difficult to imagine human designers showing the competence pattern exhibited by the P&R problem solver in table 1.

Our view is different and we believe that there are two problems with the P&R approach as instantiated by EMR and CMR architectures: i) it fails to capture adequately the design process followed by expert elevator designers and ii) it employs search heuristics which both reduce and randomise its competence. Designers possess a large amount of knowledge which they have difficulty to communicate. In troublesome cases, such as those failed in table 1 their design process probably combines reasoning from first principles, problem decomposition, deep search in a restricted subproblem, and synthesising and merging partial solutions. Because we do not have access to the relevant domain experts we cannot test their performance on the 'difficult cases' and we cannot therefore augment our problem solver with the result of these additional knowledge acquisition sessions. However, we can certainly assume that an expert would be able to do at least as well as our A*-based model. Armed with this assumption we use the following approach to improve our VT problem solver: we assume that an expert (actually the A*-based algorithm) has provided us with the solutions to the difficult cases, and we employ a case-based approach which exploits both the heuristic search framework provided by P&R, as well as knowledge about the relative complexity of the VT sub-systems. The result is a 'hybrid approach', integrating case-based reasoning and search, which exhibits a competence as good as the A*-based method, while performing as efficiently as the original P&R method.

7 CASE-BASED DESIGN

The modifications required to transform the CMR version of Propose & Revise into a case-based Retrieve & Adapt scheme are relatively straightforward. Propose & Revise consists of three subtasks: Propose, Evaluate and Revise and by different operationalizations we get EMR and CMR. Analogously, we can derive various CBR algorithms by different operationalizations of the Retrieve, Evaluate and Adapt subtasks shown in figure 2. In our experimental case-based design framework each case is represented by a valid design model, i.e. a complete elevator design, with associated cost. The access keys to the case library are lists of 26 initial parameters. The selected case must be adapted if it does not satisfy the task goal introduced in section 2, i.e. if the design model does not satisfy all design requirements or if it violates some constraints. The goal of the adaptation process is to fix violated constraints and, in addition, to optimise the cost criterion. Since our case library contains only complete and successful solutions, if the retrieved case satisfies all requirements specified by the new problem it also does not violate any constraint. In such a case no adaptation takes place.

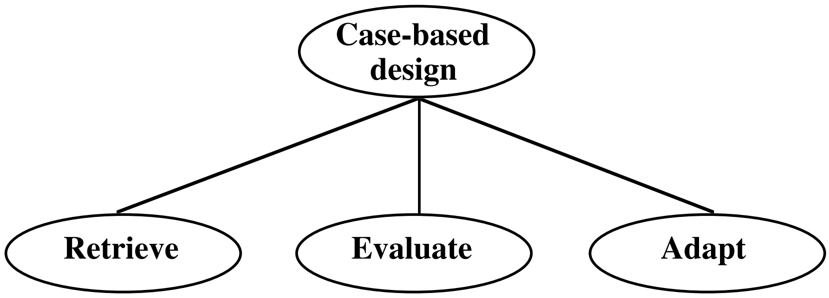

Figure 2. Subtasks of Case-based design

Typically the requirements of the new problem are different from those in the case library, and therefore the design model of the retrieved case must be evaluated and modified to satisfy new requirements. The modifications result in new constraint violations which are then fixed in the adaptation task.

The cost of the case-based design process includes both the cost of the case and the adaptation cost. Cheap adaptation of an expensive case may produce worse results than expensive adaptation of a cheap case.

We have devised and implemented four case-based design algorithms. They are described in terms of their constituent tasks and control flow.

Algorithm 1: A simple match of design requirements

Algorithm 1 operationalizes the task shown in figure 1 as follows:

Retrieve 1: The case which has the maximum number of design requirements identical with the new problem is retrieved. All requirements are taken with the same weight.

Evaluate 1: The design model of the case is modified as follows: The initial parameters of the new problem supersede those of the retrieved case. If some values are different all parameters affected by the changes are recalculated and all relevant constraints are evaluated.

Adapt 1: The adaptation is basically the same as the Revise task of P&R. A violated constraint is selected and the associated fix combinations are applied in accordance with the increasing cost. If a new constraint is violated the fix combination is rejected. This adaptation strategy is based on concepts described in Section 5 (a) - (e).

The control flow is defined by the sequence {Retrieve, Evaluate, Adapt}. Algorithm 1 works well if the case and the new problem differ only in a set of easy-to-fix parameters. If the different initial parameters are the speed or the capacity (see table 1), the adaptation often fails because of the primitive adaptation algorithm.

Algorithm 2: A subsystem sensitive match

Algorithm 2 differs from Algorithm 1 only in the retrieval phase. We have pointed out that the design of the machinery subsystem is much more complicated than the design of other elevator subsystems. The strategy applied in Algorithm 2 selects the cases where the difficult part of design has already been resolved and thus does not need any change. The design requirements are divided into groups corresponding to different subsystems of the designed artefact. The groups are ordered in accordance with the difficulty they represent for the design process.

Retrieve 2: Assign the initial parameters into groups corresponding to different subsystems. Order the groups according to decreasing design difficulty. Select the cases with the best match for the most difficult group. If more than one case is selected, apply the same selection procedure to the selected cases using the next difficulty group, and so on. If more than one case is selected after the last difficulty group was used, select at random.

In the elevator example the best results are achieved if the initial parameters are divided into two groups. The parameters from the first group are called *essential*. The retrieval algorithm requires that all essential parameters of the selected case match exactly the corresponding parameters of the new problem. Otherwise the case is not retrieved. The remaining initial parameters are called *secondary*. The maximum number of identical secondary requirements defines the best match. The essential parameters for the elevator problem are speed and capacity. Algorithm 2 resolved successfully all tested cases providing that a case with corresponding speed and capacity exists. The next two algorithms assume the complete match of the essential parameters. They differ only in the way of exploiting the secondary parameters.

Algorithm 3: Estimating design costs

Each case in the case library has a cost associated, Q_c. The design cost of the case-based design process, Q, is the sum of the cost of the case before adaptation, Q_c, plus the adaptation cost, Q_a. The adaptation cost Q_a is the sum of the costs of all fixes applied in the process of adaptation. Since the design task has been defined as an optimisation problem the solution must minimise the design cost, i.e. $Q^* = \min\{Q\}$. The case cost Q_c is known in advance for each case. The adaptation cost can be evaluated only during the adaptation process. However, after the evaluation phase and before the start of the adaptation phase, it is possible to estimate the lower and upper bounds of the adaptation cost.

Assume that after completing the Evaluate task we get a set of constraint violations, say CV. In order to remove a constraint violation we apply a fix (or a fix combination). The lower bound Q' estimate of the adaptation cost Q_a is based on the best-case scenario for satisfying constraints from CV. The lower bound Q'_a of the adaptation cost is the sum of the costs of the fixes selected under the assumption that (1) for each violated constraint, the cheapest fix combination is selected and (2) if a fix can remove multiple constraint violations it will do so. Similarly, the upper bound Q''_a estimate is based on the worst-case scenario. Each constraint violation will have to be removed independently of the others by the most expensive fix combination.

In the first step - filtering- the retrieval algorithm eliminates case C_i if a case C_j exists such that the best-case outcome of C_i is worse that the worst-case outcome of C_j, i.e. $Q_c(C_i) + Q'_a(C_i) > Q_c(C_j) + Q''_a(C_j)$. The second step of Algorithm 3 has two alternatives: a retrieval based on $Q_c + Q'_a$ or a retrieval based on $Q_c + Q''_a$. The former one is an optimistic strategy, the latter one is pessimistic. The strategy selection can be grounded on the analysis of the design domain. If the design subtasks have enough degrees of freedom and if they are only loosely coupled, then the optimistic strategy gives good results. Otherwise it is better to be pessimistic.

Algorithm 3 consists of the following new subtasks.

Retrieve 3: Retrieve all cases providing a complete match of all essential initial parameters.

This subtask is not very difficult because the essential initial parameters can be used as primary indexes to the case library.

Evaluate 3: For each retrieved case calculate Q'_a and Q''_a. Calculate $Min = \min\{Q_c + Q''_a\}$ over all retrieved cases. Delete all cases for which $Q_c + Q'_a > Min$. Select retrieve strategy (optimistic or pessimistic) based on the properties of the domain. Order cases in accordance with the selected strategy and take the first case.

Algorithm 4: Optimising adaptation costs

Algorithm 4 puts even more emphasis on the optimisation character of design. The idea is to control adaptation by continuously updating the estimated cost thus optimising over a number of cases. The result is an A* type of algorithm. The adaptation cost Q_a is divided into two components, Q_g and Q_h, $Q_a = Q_g + Q_h$. Q_g is the cost of the adaptation steps we have carried out so far, Q_h is the cost of the adaptation steps yet to be made. Unless the missing steps are made Q_h is unknown. However, we can get a lower estimate, Q_h', and an upper estimate, Q_h''. Q_h' gives an admissible search algorithm which always leads to the optimal solution. Q_h'' sometimes converges faster but the optimality is not guaranteed. The basic steps are summarised below.

Retrieve 4: The same as Retrieve 3. Denote the set of retrieved cases as R.

Evaluate 4: (Admissible search). Calculate $Q_c + Q_a'$, where $Q_a' = Q_g + Q_h'$ for all cases in R. Initially, $Q_g = 0$ and $Q_a' = Q_h'$. Order cases in terms of the increasing value of $Q_c + Q_a'$ and select the first case. If the case does not violate any constraints report success.

Evaluate 4 can be completed by the filtering technique described in Evaluate 3.

Adapt 4: Select a violated constraint to fix and apply the cheapest associated fix combination. Apply fix combination, calculate Q_g, update $Q_c + Q_g + Q_h'$, and return the result to R.

Algorithm 4 is shown in figure 3, the italicised task Evaluate indicate a recursive call.

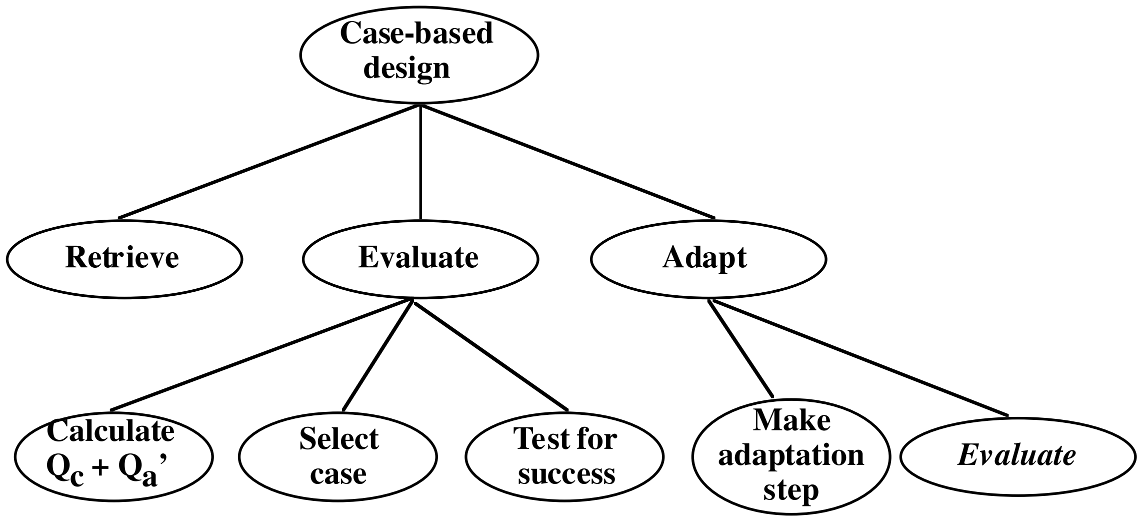

Figure 3. Optimising case-based design

8 CONCLUSIONS

In this paper we have analysed the competence of the P&R problem solving method. The experiments show that the results are counter-intuitive and unpredictable. We have argued that this problem cannot be caused by lack of domain knowledge, but such behaviour is the result of the pruning mechanism employed by P&R, which indiscriminately prunes the search when the algorithm makes a step seemingly away from the solution, see (Motta and Zdrahal, 1996). Ideally, we would like to carry out additional knowledge acquisition sessions with the original VT domain experts and analyse the problem solving techniques used when solving the cases which cannot be solved by the given P&R method. Because this is of course not possible, we have used an alternative approach, and solved the ‘difficult cases’ by means of an A* algorithm. Having done this we can then try and develop a more competent problem solving method, which extends the original P&R approach by integrating it with a case-based design module.

Designers use cases. They also use design heuristics, knowledge of domain, inventiveness and any available tool which may help them (Chandrasekaran, 1990; Fischer and Nakakoji, 1992; Gero, 1990; Maher et al., 1995). The described case-based design algorithms make use of the available knowledge originally intended for heuristic search design methods as a means for case adaptation of already existing solutions. The reason is that while the heuristic knowledge does not allow us to resolve all new problems from scratch, it is sufficient when used in combination with cases. There are two underlying principles employed by the described algorithms:

- Complex solutions can be better exploited by a case-based approach if they can be broken down into meaningful subsystems. The retrieval algorithm may be biased towards the cases whose difficult-to-design subsystems can be reused with minimum changes.
- Since design is an optimisation problem the cost element of the design can be used to guide the adaptation search. The cost of the solution integrates the cost of the retrieved case and the adaptation cost. The result can be optimised by combining partial evaluation with partial adaptation steps and dynamically selecting and adapting the most promising case.

The described algorithms transfer the computational burden from adaptation to the early stages of case-based reasoning, i.e. retrieval and evaluation. In parametric design this approach is justified. The computationally expensive task is adaptation, because it includes various design choices, i.e. a search in a potentially large space. Simplifying adaptation at the cost of more complex retrieval may significantly accelerate the whole design process. Moreover, using the condition of complete match for essential requirements as an initial filter decreases significantly the number of cases which must be taken into account.

Design is a good application domain for CBR techniques. A number of case-based systems with applications to design are reviewed in (Voss and Oxman, 1996). Some of these systems employ similar ideas and techniques with the approach described in this paper. For example, in AAAO, EADOCS, Composer and IDIOM, the design problem is represented as constraint satisfaction. EADOCS and IDIOM take into account the optimisation aspects of the problem. DÉJÀVU (Smyth and Keane, 1996) makes use of adaptation knowledge to assist in retrieving the most "adaptable" case. Algorithms 3 and 4 are based on a similar idea but their aim is retrieving not only the most adaptable but also the cheapest solutions.

Design decomposition into 'nearly independent subproblems' or 'loosely coupled subproblems' is discussed e.g. in (Maher, 1990). Composer, DÉJÀVU, IDIOM make use of this technique: they divide the problem into subproblems, resolve these subproblems by case-based methods and finally synthesise the solution. The decomposition technique used in this paper is simpler. Our algorithms identify only the most difficult design subproblem and retrieve and adapt the case whose difficult-to-design part can be reused. We assume that problem decomposition is a part of domain knowledge.

Parametric design is probably the simplest design task. Still many complex problems can be expressed as parametric design tasks. The complexity of the designed artefact plays an important role in selecting an appropriate problem solving method. For example, our experiments with the sliding bearing design (Horak et al., 1995) indicate that for simple domains the case-based approach is not needed. The sliding bearing domain consists of 44 parameters, 5 constraints and 5 fixes. The bearing can be designed only by applying design defaults and heuristic corrections.

Designers also optimise. However, the optimisation criterion is usually so complex and difficult to express that optimisation techniques are not easily applicable. The advantage of a case-based approach is that stored cases may denote already existing, i.e. field-tested optimal solutions together with their costs. The described algorithms make use of optimisation elements in guiding the adaptation.

REFERENCES

Banecek J. and Drvota J. (1995). Problem Analysis and Decomposition of Initial Vehicle Design. *TR-Encode-Dccs-1-95* (Project Encode report). DccS Engineering, Prague.

Chandrasekaran B. (1990). Design Problem Solving: A Task Analysis. *AI magazine*, Vol. 11, No. 4. Winter 1990. pp. 59 - 71

Fischer G. and Nakakoji K. (1992). Beyond the macho approach of artificial intelligence: empower human designers-do not replace them. *Knowledge-Based Systems.* Vol.5. No. 1. March 1992. pp. 15-30.

Gero J. (1990). Design Prototypes: A Knowledge Representation Schema for Design. *AI magazine*, Vol. 11, No. 4. Winter 1990. pp. 26-36

Horak J., Valasek M. and Bauma V. (1995). Knowledge Level Analysis of Bearings Design. *TR-Encode-CVUT-1-95* (Project Encode report). Czech Technical University, Faculty of Mechanical Engineering. Prague.

Maher M.L. (1990). Process Models for Design Synthesis. *AI magazine*, Vol. 11, No. 4. Winter 1990. pp. 49-58

Maher M.L., Balachandran M.B. and Zhang D.M. (1995). Case-based reasoning in design. *Lawrence Erlbaum Associates, Publishers*. Hawah, New Jersey.

Marcus, S. and McDermott, J. (1989). SALT: A Knowledge Acquisition Language for Propose-and-Revise Systems. *Journal of Artificial Intelligence*, 39(1), 1-37.

Motta E., Stutt A., Zdrahal Z., O'Hara K.and Shadbolt N.(1996). Solving VT in VITAL: a study in model construction and knowledge reuse. *International Journal of Human-Computer Studies (1996)**44.** pp. 333-371.*

Motta E. and Zdrahal Z. (1996). Parametric Design Problem Solving. In *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop* .

Selman B., Levesque H. and Mitchell D. (1992). A New Method for Solving Hard Satisfiability Problems. in *Proceedings of AAAI-92*. San Jose, California. pp. 440-445.

Smyth B. and Keane M.T. (1996). Adaptation-Guided Retrieval. In *Proceedings of the 2nd UK CBR workshop* (I.D.Watson ed.). Salford, pp. 2-15

Voss A. and Oxman R. (1996). A Study of Case Adaptation Systems. In *Artificial Intelligence in Design '96*. (J.S.Gero and F Sudweeks eds.). Kluwer Academic Publishers. pp. 172-189.

Wielinga B.J., Akkermans J.M. and Schreiber A.Th. (1995). A Formal Analysis of Parametric Design Problem Solving. In *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop* (B.R.Gaines and M.Musen eds.). pp. 37-1 - 37-15.

Yost G.R. and Rothenfluh T.R. (1996). Configuring elevator systems. *International Journal of Human-Computer Studies (1996) **44.** pp. 521-568.*

Zdrahal Z. and Motta E. (1995). An In-Depth Analysis of Propose & Revise Problem Solving Methods. In *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop* (B.R.Gaines and M.Musen eds.). pp. 38-1 - 38-20.