Learning Adaptation Rules From a Case-Base

Kathleen Hanney and Mark T. Keane

Dept. of Computer Science, Trinity College Dublin, Dublin, Ireland
E-mail: kathleen.hanney, mark.keane@cs.tcd.ie

Abstract. A major challenge for case-based reasoning (CBR) is to overcome the knowledge-engineering problems incurred by developing adaptation knowledge. This paper describes an approach to automating the acquisition of adaptation knowledge overcoming many of the associated knowledge-engineering costs. This approach makes use of inductive techniques, which learn adaptation knowledge from case comparison. We also show how this adaptation knowledge can be usefully applied. The method has been tested in a property-evaluation CBR system and the technique is illustrated by examples taken from this domain. In addition, we examine how any available domain knowledge might be exploited in such an adaptation-rule learning-system.

1 Introduction

Part of the success of case-based reasoning (CBR) has been attributed to its reduced knowledge engineering requirements. Case-based reasoning researchers point out that cases are often available for free in existing database records. Furthermore, these cases contain a lot of implicit knowledge about causal dependencies and the relative importance of different features in a domain. However, this optimistic picture of CBR can really only be sustained for retrieval-only systems. Unlike cases, adaptation knowledge is not usually readily available and the development of such knowledge involves knowledge acquisition difficulties akin to those found in traditional expert system design. The simplest and most widely-used form of adaptation knowledge acts to resolve feature differences between a target problem and a retrieved case [3]. For example, in the Déja vu system, which deals with the design of plant control programs for steel mills [7, 8], many feature differences can arise between problem specifications and retrieved cases. A problem specification could require a two-speed buggy to pick up a load from one location and deliver it to another location, whereas the best retrieved case might only describe how a one-speed buggy carries out these actions. Déja vu has a specific adaptation rule to deal with this buggy-speed feature-difference; this rule notes that two-speed buggies have to be slowed before they stop, whereas one-speed buggies do not and modifies the solution of the one-speed buggy case to reflect this difference. This speed-difference rule is typical of CBR adaptation rules. These rules tend to be domain-specific and have to be hand-coded by the system developer. Hence, the knowledge-engineering effort expended in one domain tends not to be re-usable in other domains (although see [8, 10] for some exceptions). Furthermore, the knowledge engineering task in coding adaptation

knowledge is non-trivial. The knowledge engineer has to predict the feature-differences that are likely to arise between known cases and all possible future problems and then determine the solution changes produced by these differences. Indeed, the latter will often require a deep understanding of the problem domain. If CBR is to deliver on the promise it clearly shows, then solutions to this knowledge engineering problem will have to be found. This view is supported by the current interest in adaptation and the worries being raised about its tractability (see e.g. [11]). In this paper, we look at one possible solution to this problem, we pursue the possibility that adaptation knowledge might be acquired automatically from case knowledge. We are aware that there are many types of adaptation rule that carry out different roles in CBR systems; however, as a first pass, we tackle the learning of feature-difference-based adaptation rules. In the next section, we review this and other approaches to adaptation-knowledge acquisition, before describing a technique for learning adaptation knowledge (section 3) and presenting some preliminary experimental evidence on this technique (section 4). Later, we show how domain knowledge aside from the case-base aids adaptation rule learning (section 5).

2 Learning Adaptation Knowledge

Ideally, CBR systems should be able to learn adaptation knowledge automatically from knowledge sources that have minimal knowledge-engineering requirements. There appear to be three main methods for learning adaptation knowledge; methods that exploit (i) other domain knowledge, (ii) an expert user or (iii) the case-base. The first two solutions have been examined in the literature but we know of no attempt to use the last method to learn adaptation rules.

2.1 Using Domain Knowledge to Learn Adaptation Rules

Leake et al. (see [4, 5]) report on one method that uses other domain knowledge to learn adaptation knowledge. The system is given general adaptation knowledge strategies which when applied result in specific adaptation cases and these adaptation cases are stored for future use. Adaptation is viewed as a combination of transformation and search. When differences arise between the current problem and the retrieved case, case-based adaptation is attempted and if no suitable prior cases are found rule-based adaptation is used. For example, when using rule-based adaptation to adapt a reaction-plan for the environmental problems in a factory building to an air-quality problem in a school, DIAL uses a general role/filler mismatch transformation to adapt the 'notify-union' step and finds 'parents' to be a suitable substitute for 'union' based on role similarities. DIAL can then create an adaptation case from this information for future use (see [2] for another example of adaptation rule learning from domain knowledge). The problem with this method is that it does not appear to reduce the knowledge engineering load. In particular, the handcoding of general transformation strategies may be too expensive. One possible advantage of this approach is that

domain knowledge could already be available in another knowledge based system developed for other purposes.

2.2 Interactive Adaptation Rule Learning

DIAL supplements its learning of adaptation rules with the second method of exploiting the user's knowledge. An expert user can judge whether certain feature differences matter and may even identify the need for further plan adaptation (see e.g., repair in Persuader [10]). It make a lot of sense to keep users in the learning loop as a "free" source of knowledge, as long as they are asked the right sort of questions. At the very least, an interactive approach to adaptation can reduce the knowledge engineering load. However, if the intended user of the system is non-expert then we may want to constrain what the system will learn. The user is a very useful source of adaptation knowledge but it may often be possible to supplement this type of learning with learning from other sources.

2.3 Learning Adaptation Rules from the Case-base

A third possible solution, which, to our knowledge, has not been examined before in the CBR literature, is to use the knowledge that is already available in the system, namely the case-base itself. (Related work in the growing area of knowledge discovery in databases includes the generation of association rules from databases [1]). It is possible to compute all the feature-differences that exist between cases in the case-base and examine how these differences relate to differences in case solutions. From this analysis, it should be possible to automatically learn a set of adaptation rules. The implicit assumption here is that the differences that occur between cases in the case-base are representative of the differences that will occur between future problems and the case-base (see [9], for more on this assumption). As we shall see, further processing might have to be carried out to determine which of these rules are actually used by the system. However, even if this solution is only approximate, it may be sufficient because it reduces significantly the knowledge engineering load. So far, the approach has been implemented for flat cases with ordinal, boolean or symbolic features and numeric, boolean or symbolic solutions. In the next section, we describe the method used in more detail.

3 Learning Adaptation Rules from a Case-base and Applying them

We have developed an algorithm, for learning and applying adaptation rules. It takes a case-base and performs pair-wise comparisons of its cases. The feature differences between each pair of cases are noted and become the antecedent part of a new adaptation rule, with the consequent part of this rule being the difference between the solutions of both cases. Methods for constraining case-comparison and subsequent refinement and generalisation of the generated rule set as well as rule application are described below.

3.1 Representing Adaptation Rules

Information about the adaptation rules created by case comparison is contained in two sets of rules: C-rules and F-rules. C-rules are the basic adaptation rules that associate a change in case feature values with solution changes. Each rule has seven slots: rule identifier, features, values, context, solution-change, gen, and confidence-rating. The 'features' slot contains those features with different values in the case pair. The 'values' slot contains the value pairs of the features in the features slot. The 'context' slot contains those feature values that the case pair share that have an impact on solution change. The 'solution-change' is the rule consequent and is an expression of the change is the solution. In the case of numeric solutions it could be simply additive. For symbolic solution it is a transition from one class to another class. The 'gen' slot notes whether the rule was generalised. The main advantage of generalisation is that it broadens the applicability of the specific adaptation rules. The system induces a more general adaptation rule given several similar but non-identical, specific rules. The 'confidence-rating' entry has further significance later on; at present, it is sufficient to say that it is important to weight the confidence in adaptation rules based on several factors. Its role can be accounted for by the the fact that the more frequently an association is observed, the more likely it is to be true.

F-rules just record the frequency of groups of feature value differences. There may be more than one consequent change associated with the same feature changes. In this domain, two rules are considered to be duplicates if they have exactly matching antecedents (i.e., feature differences) and broadly similar consequents. If the solution is symbolic duplicates must have both matching antecedents and consequents. Whether or not context must match depends on the domain. There is no requirement for the solutions of duplicates to match exactly because it is important for the system to handle noisy data. Each C-rule has a corresponding F-rule which has information about the number of times its set of value differences was found, the case-pairs on which it is based and the consequents of each occurence of the rule. The motivation for the use of F-rules is to curtail the size of the C-rule list while keeping account of rule frequency and consequent variation.

3.2 Reducing the Number of Adaptation Rules Generated

A key consideration in adaptation rule generation by case comparison is to limit the number of adaptation rules generated. First, one can limit the number of cases considered by the algorithm; in the experiments carried out we limit the number of cases compared, to reduce the number of rules generated. If a similarity metric is available it may be used to find cases for comparison. Case pairs within a given measure of similarity are compared. If a similarity metric is not available and requires too much effort to obtain another test for case comparison is proposed in which we make use of the fact that the retriever returns a case within a certain degree of similarity to the target for adaptation. The solution is

to let the retriever determine candidates for case comparison. A threshold number of differences between cases is determined below which case pairs will be compared. This threshold is found by using each case in the case-base as target and finding the number of differences between the target and highest ranked retrieved case. The threshold for identification of candidates for comparison can be gauged by examining the number of differences found between the target and retrieved cases in n retrievals where n is the case-base size.

The rule-set can also be reduced by deleting duplicate rules. The knowledge expressed by a set of duplicated rules can be expressed by a single rule which characterises the deleted duplicates; this rule inherits the 'features', 'values' and 'gen' entries from the duplicate rules and the value of the soln-change entry is some average of the solutions in the individual rules (clearly, what constitutes an average solution will change from one domain to another). The rule also receives a confidence rating that reflects the number of duplicates it captures (the higher the rating the more duplicates it captures).

3.3 Generalisation of Adaptation Rules

Generalisation has an important role to play in filling gaps in the rule-set; it allows a general pattern to be induced from a set of specific rules, producing a wider coverage of possible differences. One of the generalisation heuristics used for generalising rule antecedents is Michalski's [6] closing interval rule. However, it should be remembered that even the applicability of this rule depends on feature-values being ordinal or continuous. The above rule states that if two descriptors of the same class differ in the values of only one linear descriptor, then the descriptions can be replaced by a single description whose reference is the interval between the two values. Rules may also be generalised by removing constraints on context. Application of this generalisation rule results in rule antecedents like:

if the no-of-bedrooms changes by 2 in the range of 2-bed-rooms to 6-bed-rooms

Rule consequents can be generalised by taking the average (for numeric solutions) and the most common class change for symbolic solutions. Rules may be generalised by removing unnecessary contextual constraints. For example, a given feature difference might always result in the same solution change regardless of the context in which it occurs.

Note, that since the space of generalisations is huge, the system does not perform generalisations over all of the rules it creates. Rather it only finds generalisations of rules when applying the adaptation rules to a given problem. That is, it only generalises on demand when applying its adaptation rules (see next section).

3.4 Adaptation Rule Application

Apart from learning adaptation rules, we have devised mechanisms for applying these rules when given a target problem. When a case has been retrieved for a

target problem and the differences between the two noted, an attempt is made to find rules in the rule-base that match the general form of these differences. Initially, the system searches for the full list of differences occurring together in a single rule. If no single rule can resolve all the differences it looks for a set of rules. The set of differences is reduced by looking at all possible ways of dropping a difference from the list of original differences. Then a search is made for all possible shorter antecedents. Reduction of the differences continues by dropping one difference from the list of differences and finishes with a search for rules handling one difference only. This search method gives all the rules that could potentially resolve one or more differences between the base and target case. Generalisation is then applied to this set of rules. The generalised rules are then filtered to exclude rules that are no longer applicable due to scope constraints on their feature values. The resulting set contains the building blocks for solving the adaptation; that is, the list of rules that resolve the differences between the retrieved and target cases. This set contains those rules that match differences in the difference list exactly and those generalised rules applicable to the differences. Whether or not the adaptation rules need to be ordered depends on the solution type and the domain.

The rule set may contain rules with matching antecedents and conflicting consequents. When applying such rules, the rule whose base cases are closest to the target are chosen. When more than one rule is applicable, we need some way to select a rule from the set of applicable rules. Where the differences between the target and best case are only partially resolved, the set of rules which handle the most differences is chosen. Where all the differences between target and best match case have been resolved this decision is based on three factors (listed in order of precedence):

- *specificity*: Given a list of differences to resolve, preference is given to rules that include as many of these differences as possible. For example, if four differences are to be resolved we prefer a rule that can tackle all four differences as opposed to four rules that handle one difference each. (This heuristic is important in systems where the feature values interact.)
- *confidence rating*: the rule with the highest confidence rating is used.
- *concreteness*: Specific rules have primacy over general rules, because generalisation is not guaranteed to be correct.

Using these criteria an adaptation path is constructed from selected rules. If unhandled-differences still remain then other methods must be used, like relying on an expert user to institute the changes. Alternatively, a new adaptation path may be constructed from the set of rules applicable to the differences. If the best match case cannot be fully adapted the user can choose to try the next best match until a satisfactory solution is found. There is a trade-off here between a full adaptation of a similar previous case and a partial adaptation of a case more similar to the target problem. This iterative method is especially useful when adaptation rule application is ordered since it was found that the requirement for finding a full adaptation path with a certain ordering is harder to meet.

4 Learning Adaptation Knowledge from Cases: Experiments using a Pure Inductive Approach

To determine whether the method was at all feasible a relatively simple system was used in the empirical tests. In Hanney et al's [1] taxonomy of CBR systems, the simplest CBR system performing adaptation has an atomic solution (i.e., a single value, numeric or symbolic) and solves problems using a single adapted case. Therefore, the chosen system had these characteristics with the added constraint that feature changes in a case had local, non-interacting effects on the solution value. This type of system is representative of many simple systems currently used in CBR industrial applications. The CBR system was created from an expert system for property evaluation; the expert system was used to create the case-base and to determine correct answers to problems. The CBR version of this system had 1000 cases in its case-base and used a standard, spreading-activation retrieval algorithm. The 1000 cases were generated randomly; that is, features values were selected for each possible slot of a case and then the expert system was used to provide the solution answer for this set of features. Figure 1 shows a sample case:

feature	value
location	loc3
nr-bed-rooms	3-bed-rooms
nr-rec-rooms	2-rec-rooms
kitchen	medium-kitchen
structure	detached
nr-floors	1-floor
condition	excellent-condition
age	mature-age
facilities	facilities-far
price	75000

Fig. 1. Sample Case from the Property Evaluation Domain

It is probably true to say that a "real" case-base for this system might have a greater clustering of cases than our randomly generated case-base. However, a more clustered case-base would probably make adaptation-rule learning easier because there would be better structure in it. For present purposes, it is sufficient to note that this case-base might be a little harder to learn rules from than a real one. For illustration purposes, an adaptation rule generated by the system is shown below. The basis for this rule is case 1000 compared with case 792 (see figure 2).

If kitchen changes from bad-kitchen to good-kitchen and number of reception rooms changes from 3 reception rooms to 6 and condition changes from medium condition to bad condition then the house value decreases by 4000

Case_1000		Case_792	
location	loc4	location	loc4
nr-bed-rooms	6-bed-rooms	nr-bed-rooms	6-bed-rooms
nr-rec-rooms	3-rec-rooms	nr-rec-rooms	6-rec-rooms
kitchen	bad-kitchen	kitchen	good-kitchen
structure	semi	structure	semi
nr-floors	2-floors	nr-floors	2-floors
condition	medium-condition	condition	bad-condition
age	mature-age	age	mature-age
facilities	facilities-far	facilities	facilities-far
price	73500	price	69500

Fig. 2. A Compared Case Pair

4.1 Experimental set-up

In all the experiments, the CBR property-evaluation system was used. It has 1000 cases and was presented with 3 sets of 100 randomly-generated test problems. These problems were solved by retrieving and adapting cases from the case-base, using the learned adaptation-rules. In order to rate the performance of the CBR system, the dependent variable was how close the CBR system's answer was to the expert system's answer. The formula used to measure this deviation is

$$E = \frac{\sum_{i=1}^{N} \frac{|estimate(i) - rbs(i)|}{rbs(i)}}{N} \quad (1)$$

N is the number of test cases, rbs(x) is the the expert system's solution and estimate(x) is the CBR system's solution. If the CBR system is being successful then most of the test problems should deviate minimally from the expert-system's solution.

4.2 Experiment 1

In the first experiment, the system does simple case comparison with no generalisation and uses no other domain knowledge. A threshold value of 4 differences

was used to identify candidate case pairs. Alternatively, a similarity metric could be used. We can justify the usage of this threshold by the assumption that case coverage is good and the retriever retrieves a case sufficiently close to the target. The threshold is estimated by using each case in the case-base as target and finding the number of differences that exist between the case retrieved from the case-base with this target removed. It was found that there was almost always no more than 4 differences between the target and retrieved cases so only rules with a cardinality of 4 or less were generated.

Three test sets of 100 cases were used to evaluate the relative performance of the retrieval only system and the system with adaptation. The percentage error (calculated using formula 1 above) for the retrieval only system ranged from 0.077 to 0.133 whereas the percentage error of the system performing adaptation was considerably less ranging from 0.001 to 0.003. It is also interesting to examine the number of exactly correct results given by the system performing adaptation. Whereas the retrieval only system found on average only 3 exactly correct solutions the system performing adaptation found the exactly correct solution on 95 per cent of test problems. The above experiment shows that adaptation knowledge can be acquired automatically and that it results in significant performance improvements over a system lacking such rules. In the next section, we describe how the addition of domain knowledge effects adaptation rule learning.

5 Using Domain Knowledge to Support Adaptation Rule Learning from Cases

Often, there are known regularities in the domain which are not evident from an examination of the case base. We demonstrate how this knowledge can be utilised to guide adaptation rule learning. First, each of the general categories of domain knowledge types is described. Second, we show how this domain knowledge can be effectively exploited. Third, the effects of domain knowledge on adaptation rule application are examined.

5.1 Types of Domain Knowledge

Four types of domain knowledge are considered: known adaptation rules, knowledge of feature relevance, knowledge of contextual constraints and knowledge of interactions between adaptation rules. Each of these knowledge types is described below and one example of each type is provided.

- *Known Adaptation Rules* are adaptation rules already known by the system designer. For example, a change in structure from terraced to semi-detached results in a price increase of 7000
- *Irrelevant Features* are features that have no effect on the solution. In our domain, some features are always irrelevant while other features are irrelevant if certain contextual constraints hold. An example of a rule containing knowledge about feature irrelevancy is the rule, *if location is location 1 or location 2 then distance from facilities has no effect on the solution*

- *Contextual Dependencies* Sometimes, the solution changes arising from a feature value change depend on the context. For example, in our experimental domain, the addition of a bedroom in location 1 (a good location) results in a larger price increase than in location 6 (a medium location).
- *Feature Interaction* It is known that the effects of some features are non-interacting. Features are non-interacting if differences in those feature values occurring together have the same effect as the sum of the effects of those differences occurring separately.

5.2 Domain Knowledge Constrains Adaptation Rule Learning

Previously known adaptation rules prevent the acquisition of rules produced by case comparison if a predefined adaptation rule matches exactly the antecedent of a rule suggested by case comparison. Here, case comparison is not productive since the rule is already known and no learning occurs. Adaptation knowledge has an effect on what is learned from case comparison when the antecedent list of the predefined adaptation rule is contained in the antecedent list of the suggested rule. Here learning is possible by subtracting the known adaptation rule from the rule suggested by case comparison. For example, if the result of case comparison is the rule [1], and [2] is a known adaptation rule, then the new rule [3] may be inferred.

if d1 and d2 then soln change is X [1]
if d1 then soln change is Y [2]
if d2 then soln change is X-Y [3]

In this way, the differences handled by the known adaptation rule along with its effects on the solution are removed from the rule obtained from case comparison. The result is a new rule which might not be obtained using cases alone. The existence of irrelevant case features complicates the rule acquisition process because feature relevance has a direct effect on the contents of a rule's antecedent. The value of knowledge about feature relevance is illustrated with an example. Consider case[1] and case[2] in figure 3 below.

In case[1], features *d, a f* are irrelevant (given in feature relevance rules). In case[2], features *d, a, f* and *r* are irrelevant. The change in condition from *gc* to *bc* is responsible for *r's* being irrelevant in case[2]. Now a simple comparison of case[1] and case[2] might suggest the rule [c [gc $\rightarrow$ bc]] $\rightarrow$ decrease 4000.

This rule is erroneous because it includes the side effects of feature relevance by failing to note that feature *r's* irrelevance in case[2] has a positive effect on the solution. (In fact the difference in solution for the single change [gc $\rightarrow$ bc] is a decrease of 7000). The problem then is to find whatever context features are responsible for the attenuated effect of the change in context in this case pair. Clearly, it is not desirable, during case comparison, to put all the differences and context features in the rule antecedent. In order to maximise rule applicability, we require only the feature differences and context changes that justify the solution change. Knowledge of any other feature differences or context is

[1]			[2]		
	c	gc		c	bc
	l	ll		l	ll
	a	ya		a	ya
	b	4b		b	4b
	k	mk		k	mk
	f	lf		f	lf
	r	3r		r	3r
	s	ss		s	ss
	d	ff		d	ff
	p	107000		p	103000

Fig. 3. Case Pair Illustrating the Effect of Irrelevant Features

unnecessary and imposes undesired constraints on applicability. We use a simple technique that relies on knowledge about causal dependencies between features to find minimal rules. Like feature relevance knowledge, contextual knowledge plays a role in the selection of rule context components. Knowledge about contextual constraints guides the generation of rule context components. Failure to take these constraints into account may result in rules with over generalised applicability. The problem of finding the right context during case comparison is illustrated by the case pair in figure 4.

[3]			[4]		
	c	mc		c	mc
	l	l8		l	l6
	a	ya		a	ya
	b	lb		b	lb
	k	bk		k	bk
	f	lf		f	lf
	r	2r		r	2r
	s	ss		s	ss
	d	fv		d	fv
	p	29500		p	40500

Fig. 4. Case Pair Illustrating the Effect of Contextual Dependencies

In case [3], the impact of the values of the features b and d are governed

by contextual constraints. In case [4] they have the usual effect on the solution. Case[4] is identical to case[3] apart from the location. However, this change from *l8* in case[3] to *l6* in case[4] means that the values of features *b* and *d* have different effects in these two cases. The rule associating the change (l8 → l6) with the solution differences is erroneous since it does not take into account the different contributions of features *b* and *d* to the final solutions in both cases. Straight comparison is therefore insufficient for this case pair. The problem is to identify the context features responsible for the reduced increase in solution caused by this solution change. (It is sufficient to note that normally, this change in feature *l* results in a bigger solution increase). The procedure used to identify necessary context features relies on knowledge of causal dependencies between features. In the case of non-interacting features, the sum of the effects of differences occurring separately is equal to the effect of those differences had they occurred together. Knowledge about non-interacting features helps constrain the number of adaptation rules learned. For example, if the rules [9] and [10] below have already been learned and if a case comparison suggests the rule [11] there is no need to retain this proposed rule since it may be deduced that the consequent of rule [11] is the sum of the consequents in rules [9] and [10]. Note that it is more desirable to have a group of rules with small antecedents than an equivalent longer rule since smaller rules have more extensive applicability.

Rule 9: diff1 and diff2 → effect on soln X
Rule 10: diff3 → effect on soln Y
Rule 11: diff1 and diff2 and diff3 → effect on soln Z

So, in general, if the differences between two compared cases are all non-interacting, a search is made of the adaptation rule-base to find a group of rules whose collective applicability is equivalent to the consequent of the proposed rule.

5.3 Effects of Domain Knowledge on the Adaptation Rule Application Process

The rule application process is effected by domain knowledge in the following ways:

- *irrelevant features* may be removed from the list of differences between the target and best matching case and only the remaining differences need be resolved.
- *contextual dependency* means that when assessing the applicability of a rule, context too must be checked. A rule may be applied if either the feature value constraints in the rule antecedent are satisfied or the context of the rule is unspecified.
- *adaptation rules* allow the reduction of differences because the system applies domain adaptation rules before learned rules.

There are six possible outcomes for adaptation rule application in this system. First, if the target case is already in the case-base the solution of the duplicate

case in the case-base is given and no further action is necessary. Second, the differences between the target and best match cases may be resolved entirely by domain knowledge. For example, the differences between the best-match and target cases may be found to be irrelevant or solvable by domain adaptation rules. Third, a sufficiently similar best match case may not be found for a target case making adaptation impossible. Fourth, there may be no applicable adaptation rules. This happens if none of the differences between the best match and target cases are resolved by the learned adaptation rules. Five, the differences between the target and best match cases may be only partially resolved by adaptation rules. In this case, the remaining unresolved differences are presented to the user for further processing. Six, all the differences between the best match and target cases may be resolved by adaptation rules (either learned rules or a combination of learned and domain adaptation rules).

5.4 Experiment 2

A new case-base containing 1000 cases was randomly generated for this experiment and the same 3 sets of 100 test cases were used for the three conditions of this experiment. The major difference between this case base and the earlier one was that it contained implicit contextual constraints. In addition, the cases also contained features that were sometimes or always irrelevant. In this way, two types of domain knowledge, contextual constraints and feature irrelevance directly influence case generation. The experiment examines the effect of domain knowledge on rule learning and application in three conditions involving separate tests on a retrieval-only system, a system which learns adaptation rules with no domain knowledge, and a system that uses all four types of domain knowledge during adaptation rule learning.

Using formula 1 the percentage error was measured for each of the three conditions. The percentage error of the retrieval only system ranged from 0.071 to 0.098. The application of adaptation rules generated without using domain knowledge brought the percentage error to within the range 0.019 to 0.032. Interestingly, the third set of tests which used adaptation rules generated using domain knowledge gave a higher percentage error; between 0.030 and 0.042. From these results it would seem that the addition of domain knowledge has a negative effect on results. However if we examine the number of exactly correct solutions the benefits of domain knowledge are more evident. Using adaptation rules generated without domain knowledge the average number of exactly correct solutions was 38 per test set. The addition of domain knowledge brought this figure to 71 per test set. The results show that a good approximation of the solution may be obtained without domain knowledge but that domain knowledge can significantly improve adaptation accuracy.

6 Conclusions

A method for adaptation rule learning has been described. The technique manipulates cases alone or a combination of cases and domain knowledge. Our

results show that case-comparison can improve on the performance of a retrieval system without costly knowledge engineering. In the domain illustrated here, rule consequents are additive. The approach can be extended to handle other types of solution change, e.g. multiplicative changes. We then showed how domain knowledge can be exploited in adaptation rule learning. The results show that best adaptation accuracy is obtained when all domain knowledge is used during adaptation rule learning. The use of domain knowledge can help prevent incorrect adaptation by allowing the system to enforce constraints on adaptation rule applicability. The success of our approach relies on the existence of a well populated case-base. Further tests are necessary to see if this conclusion holds for other more complex domains and an extension of the approach to other types of adaptation knowledge would be useful.

References

1. Agrawal R., Mannila H., Srikant R., Toivonen H., Verkamo A.: Fast Discovery of Association Rules. In Fayyad U., Piatetsky-Shapiro G., Smyth P., Uthurusamy R. (Ed.) Advances in Knowledge Discovery and Data Mining. *AAAI Press / The MIT Press* (1996)
2. Hammond K.:Case-Based Planning: Viewing Planning as a Memory Task. *Boston: Academic Press* (1989)
3. Hanney K., Keane M.T., Smyth B., Cunningham P.: Systems, Tasks and Adaptation Knowledge: Revealing some Revealing dependencies. In Proceedings of the First International Conference on Case-based Reasoning (1995) 461–470.
4. Leake D.: Combining Rules and Cases to Learn Case Adaptation. In Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society, (1995)
5. Leake D., Kinley A., Wilson D.: Learning to Improve Case Adaptation by Introspective Reasoning and CBR. In Veloso M., Aamodt A.: (Eds.) Case-based Reasoning Reasoning Research and Development: Lecture Notes in Artificial Intelligence 1010. *Springer Verlag* (1995) 229-240.
6. Michalski R.: A Theory and Methodology of Inductive Learning. In R. Michalski, J. Carbonell, T. Mitchell (Ed.) Machine Learning: An Artificial Intelligence Approach Vol. 1. *Morgan Kaufmann* (1983)
7. Smyth B., Cunningham P.: Déja vu: A Hierarchical Case-Based Reasoning System for Software Design. In Proceedings of the 10th European Conference on Artificial Intelligence. Vienna, Austria (1992)
8. Smyth B., Keane M.T.: Retrieving Adaptable Cases: The Role of Adaptation Knowledge in Case Retrieval. In Topics in Case-Based Reasoning: Lecture Notes in Artifical Intelligence 837. *Springer Verlag* (1994) 209–220
9. Smyth B., Keane M.T.: Remembering to Forget: A Competence-Preserving Deletion Policy in Case-Based Systems. In Proceedings International Joint Conference on Artificial Intelligence, Montreal. (1995)
10. Sycara E.P.: Using Case-Based Reasoning for Plan Adaptation and Repair. In Proceedings: Case-Based Reasoning Workshop (1988) 425-434.
11. Veloso M., Aamodt A.: (Eds.) Case-based Reasoning Reasoning Research and Development: Lecture Notes in Artificial Intelligence 1010. *Springer Verlag* (1995)