On Retention of Adaptation Rules

Vahid Jalali and David Leake

School of Informatics and Computing, Indiana University
Bloomington IN 47408, USA
vjalalib@cs.indiana.edu, leake@cs.indiana.edu

Abstract. The difficulty of acquiring case adaptation knowledge is a classic problem for case-based reasoning (CBR). One method for addressing this problem is to use the cases in the case base as data from which to learn adaptation rules. For numeric prediction tasks, adaptation rules have been successfully learned from the case base by using the *case difference heuristic*, which generates rules based on comparisons of pairs of cases. However, because the case difference heuristic could potentially generate a rule for each pair of cases in the case base, controlling growth of adaptation rules is potentially an even more acute problem than controlling case base growth. This raises the question of how to select adaptation rules to retain. The ability to generate adaptation rules from cases also raises questions about the relative benefit of learning cases, learning the adaptation rules generated from them, or learning both. This paper proposes and evaluates a new adaptation rule retention approach and presents a case study assessing the relative benefits of learning cases versus learning adaptation rules derived from the cases, at different points in the growth of the case base.

Keywords: case adaptation learning, case-base maintenance, knowledge containers, rule retention

1 Introduction

Case adaptation is a classic challenge for case-based reasoning. Because acquiring adaptation knowledge by hand may be difficult or expensive, much research has explored machine learning methods for generating case adaptation knowledge automatically (e.g., [1–11]). The case difference heuristic approach, first proposed by Hanney and Keane [3], is a popular approach to deriving adaptation rules from cases used for numeric prediction (regression) tasks [1, 5, 7, 8, 11], by comparing pairs of cases in the case base. For each pair of cases, it assesses the problem–problem differences between the problems solved by the cases, and also assesses their solution–solution differences. These two differences are used as the basis for generating the antecedent and consequent of a new adaptation rule. The new rule applies to a retrieved case if the retrieved case and new problem have similar problem–problem differences, and it adjusts the solution of the retrieved case according to the previous solution–solution difference. The case difference heuristic approach has received much study, but two fundamental questions it raises have received little attention: How to control growth of the set of adaptation rules, and how the benefit of learning new cases compares to the benefit of learning rules derived from those cases.

The utility issues associated with case base growth are widely recognized in the CBR community, and methods for controlling case base growth are an important CBR research area (e.g., [12]). The utility issues associated with growth of automatically-generated adaptation knowledge may be even more acute. When the case difference heuristic is used to generate cases from the case base, each ordered pair of cases may result in an adaptation rule. For a case base with n cases, the number of possible ordered pairs of cases is $2\binom{n}{2}$, so as cases are added the number of candidate rules grows much more rapidly than the case base. The need to select which rules to retain was noted when the case difference heuristic was first proposed, but since then the problem has received little attention. This paper proposes a new rule retention approach and demonstrates its effectiveness compared to alternative methods.

The ability to use cases as a source for rules also raises an interesting question about the relative benefits of learning cases and/or learning adaptation rules derived from those cases. It is well known that CBR enables system developers to strategically place knowledge in different *knowledge containers*, and that knowledge in one container may compensate for lack of knowledge in another [13]. However, the question of the relative benefit of adding cases directly or applying the case difference heuristic to existing cases, to add rules, is unexplored. This paper provides a case study as a first step towards addressing these questions.

The paper proposes a general strategy for rule retention, based on a test process for credit/blame assignment to automatically-generated adaptation rules. This strategy can be used for any CBR task in which there is a trade-off between retaining adaptation versus case knowledge, regardless of how the adaptation knowledge is generated. The paper presents an evaluation of the strategy in the context of CBR for numerical prediction, a task which has been widely studied for domains such as property value estimation, using the knowledge light "case difference heuristic" method to generate the adaptation rules. Experimental results demonstrate the retention strategy's effectiveness compared to baselines and a previously-proposed frequency-based approach [4]. It also provides an initial experimental exploration of how the knowledge content contribution of adding rules varies at different points in the growth of the case base, illustrating that the knowledge contribution of adaptation rules generated using the case difference heuristic may converge before the knowledge contributions from simply adding new cases, which suggests a knowledge growth strategy of retaining a more limited set of learned adaptations but continuing to add new cases.

2 Related Work

Our research on adaptation rule retention falls within the broad category of *case-based reasoning system maintenance* (e.g., [14]). Case-based reasoning system maintenance extends maintenance considerations beyond the case base, to consider other knowledge containers as well. The work in this paper is motivated by research on adaptation knowledge generation and contributes to the study of adaptation rule ranking and retention, and of the relationship between knowledge in different case-based reasoning knowledge containers. This section briefly highlights relevant work in each of these areas.

Adaptation Knowledge Generation: The CBR literature contains numerous examples of automated adaptation knowledge generation. As a few examples, Craw et al. [15] and Shiu et al. [10] propose building rules by decision tree learning; Leake, Kinley and Wilson [6], Leake and Powell [16], and Minor, Bergmann, and Görg [17] propose capturing cases from prior adaptations; numerous projects have investigated the case difference heuristic for adaptation rule generation (e.g. [3, 5, 7, 8, 11]). D'Aquin et al. [1] combine the case difference approach with data mining methods to improve rule quality and Fuchs et al. [18] present a framework for differential adaptation. In principle, methods from the extensive CBR literature on case-base maintenance could be applied to control retention of adaptation knowledge in the form of cases. However, for systems which learn adaptation rules, methods are needed to guide rule retention.

Filtering and Ranking Adaptation Rules: Li et al. [19] propose filtering learned adaptation rules by removing duplicate rules and merging rules which conflict (*i.e.*, rules which have the same antecedents but different consequents). For rules with numeric consequents, their approach aggregates rules into new rules whose consequents are the average of those of the previous rules; for rules with symbolic features, their approach clusters rules with the same antecedent and ranks them by frequency. This leaves open the question of how to prioritize distinct rules for retention.

Leake and Dial [20] propose selecting the rules to apply to a particular adaptation based on using provenance information to assign blame to the adaptation rules involved in errors, to assess rule quality. They assume that all rules are always present; their focus is only on rule selection from the complete pool to maximize accuracy for a particular problem, rather than determining which rules to retain for best overall performance on future problems.

Hanney and Keane's [3] seminal work on the case difference heuristic proposes retaining the adaptation rules that are generated more frequently by the pair-wise comparison of cases. However, to our knowledge, this maintenance strategy has not been formally evaluated; this paper evaluates it in comparison to our new approach.

We have proposed Adaptation-Guided Case Base Maintenance (AGCBM) [21], a method for rule and case retention aimed at simultaneously controlling the number of cases and adaptation rules generated from them. AGCBM considers the contribution of each case both as (1) a source case to provide initial estimates for input queries, and (2) as a building block for adaptation rules. It selects cases to retain based on a ranking scheme considering both types of contributions. The work presented in this paper differs in three ways. First, AGCBM is computationally expensive ($O(n^3)$ in the initial size of the case base); this paper seeks an approach feasible to apply to large case bases. Second, the approach in this paper focuses solely on rule retention, aiming to minimize the adaptation knowledge container size given a fixed case base. Third, in contrast to AGCBM, and for added efficiency, the method introduced in this paper uses single adaptation rules rather than ensembles of adaptation rules. As discussed in Section 4.4, these changes make the training of the system substantially faster than AGCBM, enabling it to be applied to large case bases which would not be feasible for AGCBM.

Relationships Between CBR Knowledge Containers: Richter [13] observed that the knowledge of CBR systems resides in multiple "knowledge containers:" the case vocabulary, similarity measures, solution transformation knowledge (i.e. adaptation knowledge) and the case base, and that the knowledge containers are overlapping: added knowledge in one may reduce the need for knowledge in another (e.g., adding cases may decrease the need for adaptation knowledge). However, there has been limited research on how learning in different containers interacts.

Leake, Kinley, and Wilson [6] demonstrate that knowledge container interactions during learning may be important, by showing that uncoordinated additions to case and adaptation knowledge may interact negatively, degrading system performance even if adding the same knowledge to one container individually, with the knowledge of the other fixed, would improve performance. The benefits of both types of learning were restored when similarity knowledge was learned as well. Shiu et al. [10] propose that adaptation rules generated from cases can be used to transfer knowledge from the case base container to the adaptation knowledge container, to compact the case base, and demonstrate the approach for adaptation rules learned as decision trees.

The ability to generate adaptation rules by the case difference heuristic raises the question of when knowledge should be retained in the form of cases, when it should be retained in the form of adaptation rules derived from these cases, and when it should be retained in both forms. This paper presents a case study on a facet of this question.

Case Knowledge Maintenance: The CBR community has investigated many methods for retaining/discarding cases, as *case base maintenance*. One prominent trend of case base maintenance is the use of *footprint-based* case base maintenance approaches, originating in work by Smyth and Keane [22]. Such approaches guide maintenance according to Reachability, Coverage and Relative Coverage, which respectively refer to the set of cases that can solve a particular case, the set of cases that can be solved by a particular case, and the set of other cases in the case base that can solve cases in the coverage set of a particular case. Footprint-based methods favor retaining cases with strong competence contributions not duplicated by other cases. We will not further survey the extensive case-base maintenance literature here, but will apply footprint-based methods as our baseline case base maintenance method for a comparative study of effects of rule set and case base growth, in Section 5.2.

3 ARR: A General Approach to Adaptation Rule Retention

As the basis for our approach to adaptation rule retention, we first present a simple framework, ARR (Adaptation Rule Retainer), aimed at automatically generating a compact set of accurate case adaptation rules. ARR involves three steps:

1. Generate a preliminary subset S of possible adaptation rules, based on a given rule generator. The size of the subset may be limited, e.g., to generate x rules.
2. Do leave-one-out testing of the CBR system with S as its adaptation rule set, applying the system to the problem parts of a subset of the cases in the case base. For each test, assess error, assign credit/blame for the error to the adaptation rule used

Algorithm 1 ARR's adaptation rule scoring algorithm

Input:
x: number of candidate adaptation rules to generate
k: number of source cases to adapt to solve each query (each generates a different solution)
CB: case base
Output: Scored adaptation rules

$AdaptationRules \leftarrow$ RuleGenerator(CB,x)
for r in $AdaptationRules$ **do**
  $r.score \leftarrow 0$
end for
for c in CB **do**
  for i in 1 to k **do**
    $RuleToApply \leftarrow$ SelectRule($AdaptationRules$,$Neighbor_i(c)$,$c.problem$)
    $ValEst(c) \leftarrow$ Adjust($Neighbor_i(c)$, RuleToApply)
    $EstErr(c.problem, Neighbor_i(c), r) \leftarrow |c.value$ - $ValEst(c.problem)|$
  end for
end for
for r in $AdaptationRules$ **do**
  $r.score \leftarrow ErrorScore(EstErr, r)$
end for
return $AdaptationRules$

to generate the solution, and retain error data. To enable generating data about multiple rules from each trial, multiple solutions may be generated for each problem, one for each of the k nearest cases to the test problem, for some fixed value of k.
3. Rank the rules according to a scoring function applied to their error data and retain y highest-ranked rules, for some user-selected y.

The parameters x and k adjust the amount of data considered in the generation process. The parameter y adjusts the final number of rules retained.

Alg. 1 presents ARR's method in more detail. $Neighbor_i(c)$ denotes the i^{th} closest neighbor of c in the case base. $RuleGenerator$ is a method for generating a subset of the possible candidate adaptation rules (e.g., $RuleGenerator$ could apply the case difference heuristic to a desired number of the possible case pairs, randomly selected without replacement). $Adjust(case, rule)$ applies an adaptation rule to a case. $c.problem$ is the problem part of a case, and $c.value$ its stored solution value. $ErrorScore$: $[0, \infty) \rightarrow [0, \infty)$ is a function that maps raw error values to a transformed value reflecting domain-specific error importance characteristics (e.g., $ErrorScore$ could assign the same scores to different error levels if the differences are deemed inconsequential).

4 ARR1: An Instantiation of the ARR Approach

ARR is a general framework. For experimental tests, it is necessary to instantiate the framework with specific choices for the rule generation procedure, rule selection, and error scoring. ARR1 is an instantiation of ARR in which $RuleGenerator$ applies a

specific form of the case difference heuristic, $SelectRule$ is based on the rule selection process developed in our previous work on CAAR (Context-Aware Adaptation Retrieval) [23], and $ErrorScore$ is designed to balance observed accuracy of each rule by the amount of evidence acquired about that rule during testing. These functions are described in the following sections.

4.1 ARR1's Rule Generator

ARR1 generates adaptation rules using the case difference heuristic approach proposed by Hanney and Keane [4] and further explored by others (e.g., [5, 7, 8, 11]). The case difference heuristic approach builds new adaptation rules by comparing problem parts and respectively solution parts of cases in the case base. Each rule maps the observed differences in the problem descriptions of a pair of cases to the observed difference in their solutions. For example, in apartment rental domain, if two apartments differ in that one has an additional bedroom, and its price is higher, the case difference heuristic could generate a rule which would increase the rent estimated for a new apartment when estimating based on a previous apartment with one bedroom fewer.

Applying the case difference approach depends on addressing questions such as which pairs of cases will be used to generate adaptation rules, what function will be derived from a given difference between the values estimated by the two cases (e.g., in the rental domain, a $100 difference could prompt a rule to adjust the price by $100, to adjust the price by the same percentage difference reflected by the $100, or any of many other alternatives), and how to select the rule to be applied to a given new problem.

ARR1 is designed for cases whose problem parts are described by numeric feature vectors. ARR1's rule generator generates rules whose antecedents are the vector difference between the problem parts of the two cases from which the rule was generated, and whose consequents are the numerical difference of those two cases' solution values. However, we note that nothing about ARR or the basic ARR1 approach precludes applying alternative methods.

The number of rules generated by ARR1's $RuleGenerator$ is determined by a user-selected parameter k, a small fixed value which determines the number of neighbor cases to which each source case should be compared to generate rules. For each source case s in the selected source cases to consider, if $\{c_i\}_{i=1,...k}$ is the set of the k nearest cases to s, ARR1 generates one adaptation rule to adapt s to each c_i. For a case base with n cases, if each case is compared with its top k neighbors, $n \times k$ adaptation rules are generated.

4.2 ARR1's Value Estimation

ARR1's rule scoring is based on the errors in estimated values resulting from applying the rules. Given a query, ARR1 generates an estimated value by adapting the k nearest cases (for a fixed value of k) and averaging the adapted values, following Alg. 2. $NeighborhoodSelector$ is a function for selecting cases in the neighborhood to adapt. ARR's $SelectRule$ function applies the context-based method for adaptation rule selection developed in our previous work on CAAR [23] (Context-Aware Adaptation Retrieval), which we selected because in previous evaluations it outperformed alternative

Algorithm 2 ARR1's Algorithm for Value Estimation

Input:
$Rules$: input adaptation rules Q: input query
k: number of source cases to adapt to solve query
CB: case base
Output: Estimated solution value for Q

$CasesToAdapt \leftarrow$ NeighborhoodSelector(Q,k,CB)
for c in $CasesToAdapt$ **do**
    RuleToApply $\leftarrow$ SelectRule($Rules$,c,Q)
    $ValEst(c) \leftarrow$ Adjust(c, $RuleToApply$)
end for
return Average$_{c \in CasesToAdapt} ValEst(c)$

methods. The specific rule selection procedure used is not significant to the lessons of our experiments, so for reasons on space, we do not describe the rule selection procedure further here. However, we refer interested readers to previous work on CAAR for the details [23].

4.3 ARR1's Rule Scoring

ARR ranks adaptation rules based on an $ErrorScore$ function assigning blame/credit scores to adaptation rules, based on the error that results when they are used to adapt source case values. ARR1's $ErrorScore$ function is designed to favor rules believed to have low error from more extensive testing. Depending on the problems used in the testing phase, some rules may be used multiple times, while others may be used seldom or never, giving less information for predicting their performance. ARR1's $ErrorScore$ favors rules which showed reasonable accuracy for multiple training cases over a rule which was only applied in a single trial even if it showed excellent accuracy there; the rationale is the expectation that results of multiple tests will be more reliable predictors of future performance.

Specifically, ARR1's $ErrorScore$ function is defined as follows. Let $EstErr$ be an array of estimated error values, with $EstErr(q, c, r)$ the calculated error when case c was adapted by adaptation rule r to solve query q. If M represents the number of times r is applied to different source cases (c_i's) during ARR1's error estimation phase, and q_i and c_i, for $i = 1, ..., M$ represent the queries and source cases used in those applications, ARR1's error score is calculated as follows:

$$ErrorScore(EstErr, r) = \sum_{i=1}^{M} \frac{1}{EstErr(q_i, Neighbor_i(c), r) + \epsilon} \qquad (1)$$

where ϵ is a small positive value that determines a non-zero minimum value for perfect predictions. Rules are ranked by their $ErrorScore$ values, in order of ascending $ErrorScore$. Ties are broken arbitrarily.

4.4 Time Complexity of ARR1

Given that ARR1 is aimed at reducing potentially large rule sets, its time complexity is an important consideration. As explained previously, for a case base of n cases and for ARR1's adaptation rule generation method, which considers a neighborhood of m cases (for some small fixed m) around each case in the case base, ARR generates $n \times m$ rules, so the time complexity of its adaptation generation process is $O(n)$.

In Alg. 1, leave-one-out testing used for rule scoring will adapt $n \times k$ source cases, for k a small fixed value. For each adaptation, the most relevant adaptation rule must be selected. ARR1 does this by simply comparing the difference between the pair (input query, source case) with all generated adaptation rules to select the adaptation to apply, so its time complexity for the entire training process will be $O(n^2)$. (We assume that k is set to a small value that is significantly less than n.) However, more efficient rule retrieval methods could reduce rule retrieval complexity.

In addition, we note that the training process is not a routine process which means it could only happen once for the life of the system, and that leave-one-out testing could be replaced by sampling methods which could decrease the processing resources required for ARR1.

5 Experiments

Our experiments address four questions:

1. Performance of ARR1: How does final accuracy compare using (1) adaptation rule retention by ARR1, (2) frequency-based rule retention, (3) random rule retention, and (4) k-NN using value averaging rather than adaptation rules?
2. Performance of $ErrorScore$: How does accuracy using a reduced rule set selected based on ARR1's $ErrorScore$ function compare to accuracy using a reduced rule set selected based on simple averaging of errors?
3. Sensitivity to number of source cases adapted per problem during training: How does the number of cases adapted per problem during training affect final system accuracy?
4. Relative benefit of rule and case learning: How does the knowledge content contribution of adding rules and cases compare, at different points in the knowledge acquisition process?

5.1 Experimental Design

We evaluated ARR1's performance on four sample domains from the UCI repository [24]: Automobile (Auto), Auto MPG (MPG), Housing, and Computer Hardware (Hardware). All records and features were used for Housing (506) and Hardware (209). Auto and MPG contained some records with unknown feature values, which were removed (46 out of 205 for Auto and 6 out of 398 for MPG). However, we note that the value imputation methods used to enable k-NN to handle missing features could equally well be applied to ARR1 to enable it to handle such records. CAAR, part of ARR1's rule selection process, uses locally weighted linear regression for defining context, so requires

numeric features. All features of MPG and Housing were numeric, but 10 non-numeric features were removed from Auto and 2 from Hardware, in order for all methods to be provided with the same amount of information. We note that the numeric features are not required by the general ARR method. For each feature, values were standardized by subtracting that feature's mean value from each individual feature value and dividing the result by the standard deviation of that feature. For the Auto, MPG, Housing and Hardware domains, the respective values to estimate are price, MPG (automobile fuel efficiency in miles per gallon), MEDV (median value of owner-occupied homes in $1000's), and PRP (published relative performance).

For all experiments, the set of adaptation rules was generated before query processing, following the ARR process of Alg. 1, and values were estimated following Alg. 2. The value of k, the number of cases to adapt, was set independently for each algorithm to maximize its performance. In Alg. 1, we expect it to be desirable for k to be set to a higher value, so that more adaptations have the chance to participate in the case value estimation process, resulting (on average) in more data being available on the performance of each adaptation rule. In Alg. 2, where k is used for estimating the input query value, we expect better accuracy for a smaller value of k (as well as reducing computation time), by focusing on more similar cases.

In our experiments, we tested a range of k values, 10, 20, 40, 80, 100 and 200 for Alg. 1, and 3, 8 and 13 for Alg. 2. Best performance was achieved for k in Alg. 1 set to 80, 100, 80, and 80 for the Auto, housing, MPG and hardware domains respectively, and in Alg. 2 to 3, 8, 8, and 8, for all domains in the same order. Same process with different k values (ranging from 1 to 10 for all domains) was used to determine the optimal k value for k-NN. The lower value of k for the auto domain compared to the other domains parallels the observation that even for k-NN, using lower numbers of nearest neighbors yields higher accuracy in the Auto domain, suggesting more locality for that domain compared to the other tested domains. The percent of possible rules to generate for Auto, MPG, Housing and Hardware domains was set to 8%, 3%, 4% and 8%, meaning that only a small portion of all possible rules was generated for each domain. For questions 1 and 2, performance was compared for varying numbers of rules retained from the set of possible rules, beginning at 20 rules and increasing to 1990 rules. All experiments used ten-fold cross validation. Adaptation rules were assigned scores by applying Alg. 1.

We compared ARR1 to two alternative retention methods. The first method, *Random*, replaces ARR1's ranking strategy with randomly selecting adaptation rules to retain. The second method, *Frequency-Based Rule Retention* (FBRR), follows Hanney and Keane's [4] proposed approach of retaining adaptation rules based on their frequency of occurrence. Because the case difference heuristic may generate multiple rules with extremely small differences, to have a reasonable indication of frequency we defined a threshold on the distance between the antecedents of the adaptation rules, and treated rules with differences below that threshold as identical. Because the antecedents of each rule are simply vectors of the numeric differences between the corresponding features of the two cases from which the rule was generated, we used Euclidean distance to measure rule similarity.

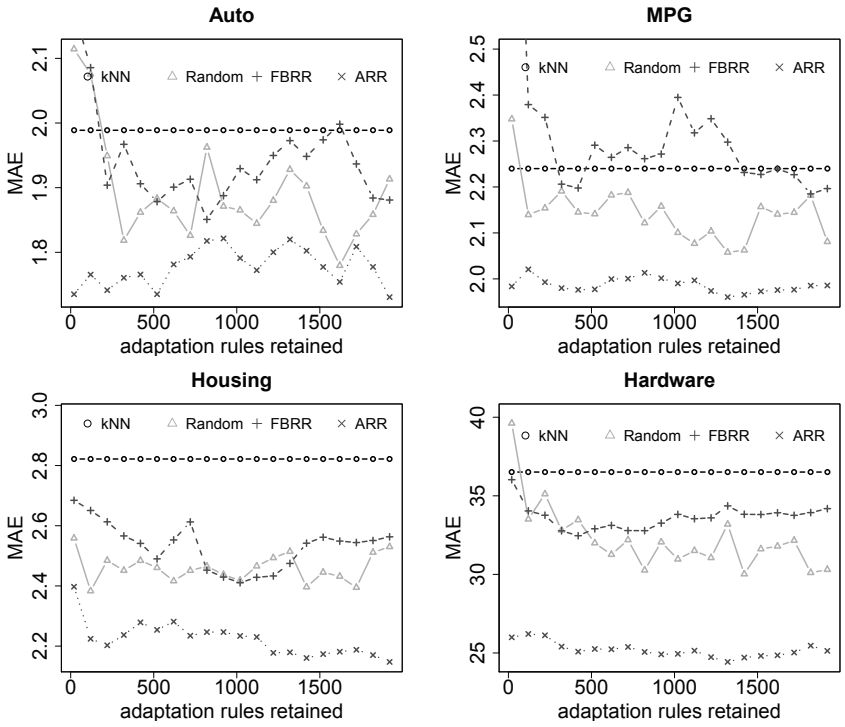

Fig. 1. MAE of the tested methods for different numbers of retained adaptations

5.2 Experimental Results

Adaptation Retention Evaluation: To address question 1, how rule retention by ARR1 affects final system accuracy compared to rule retention by other methods, we compared the Mean Absolute Error (MAE) of our test system using adaptation rules generated and retained by ARR1 to that of k-NN using value averaging rather than adaptation rules, random rule retention, and frequency-based rule retention. Figure 1 shows MAE of all methods in four domains.

In all domains, ARR1 provides the highest accuracy. In all domains except Auto, performance with ARR1 tends to improve as the number of retained rules increases. However, the improvement is not significantly different from the accuracy achieved by retaining a minimum number of adaptation rules. We discuss this further below. For the Hardware and MPG domains, after a certain point increasing the number of adaptation rules slightly degrades the performance. We hypothesize that this deterioration is due to the introduction of less accurate rules, and that this issue could be ameliorated by setting a lower bound threshold on the ranking score for rules to retain.

An interesting observation is that for three of the four domains, regardless of the rule retention method, learning and applying adaptation rules provides substantially better performance than simply averaging results with k-NN, supporting the value of adap-

tation rule learning. Even for the exception domain, MPG, two of the three retention methods provide substantially better performance than k-NN.

A surprising result is that frequency-based rule retention for the tested domains usually performs worse than random rule retention. One possible explanation is that the frequency-based approach may sacrifice rule diversity, resulting in poor coverage of parts of the space. However, further study is needed.

Assessing ARR1's Error Scoring for Blame/Credit Assignment: To investigate question 2, on the impact of ARR1's error scoring method for blame/credit assignment to rules during the retention process, we compared accuracy using rule sets selected by ARR1 and selected by an ablated version of ARR1, ARR1-AVG, that ranked rules by lowest average error rather than ARR1's $ErrorScore$. ARR1's $ErrorScore$ favors rules that are more frequently used during the test phase and at the same time yield accurate estimations; ARR-AVG's scoring favors rules that yield accurate estimations on average, regardless of the number of times the rules were applied in the training phase. Fig. 2 shows that in all test domains, retention by ARR1 resulted in lower MAE than retention by ARR-AVG, supporting the benefit of ARR1's scoring mechanism. The greatest gain over ARR-AVG is observed for the Hardware domain and the gain is least for MPG and Housing domains.

Sensitivity of Resulting Accuracy to Number of Adaptations Per Problem During ARR's Training: As discussed previously, cost of ARR1 depends on the number k of source cases adapted for each test source case. Also, there is a tradeoff in adapting larger numbers of source cases for a given test case: Adapting additional source cases provides more data about rule performance, but because the additional source cases are less similar, the additional adaptations may result in more error, penalizing rules which might have had higher rankings if only applied to more similar cases—and which might only be applied to more similar cases in practice.

To study the effect of k's value (in the training phase) on final system accuracy with the retained rule set, we tested accuracy for 5 k values, 10, 20, 40, 80 and 100. Fig. 3 shows results for the Auto and MPG domains. Best performance for both domains was achieved for a k value of 80. The performance of ARR1 for different k values across MPG and Auto domains shares some general patterns, with some variation. For example, although for the MPG domain the second best performance is achieved when k is set to 100, the same k value in the Auto domain yields lower performance, especially when the number of retained adaptation grows. The results suggest that the performance of ARR1 improves as k's value increases up to a certain point, determined by the domain, and then decreases as k increases further. The housing and MPG domains are not shown for reasons of space, but also show this pattern.

We explain this pattern by the balance between the advantage of judging adaptation rules based on their effectiveness for local cases (because, for a well populated case base, it is more reasonable to select the source cases from the local neighborhood of the input query), versus the previously-noted drawback that using a small number of source cases (lower values for k) in the training phase decreases the number of rules examined by ARR.

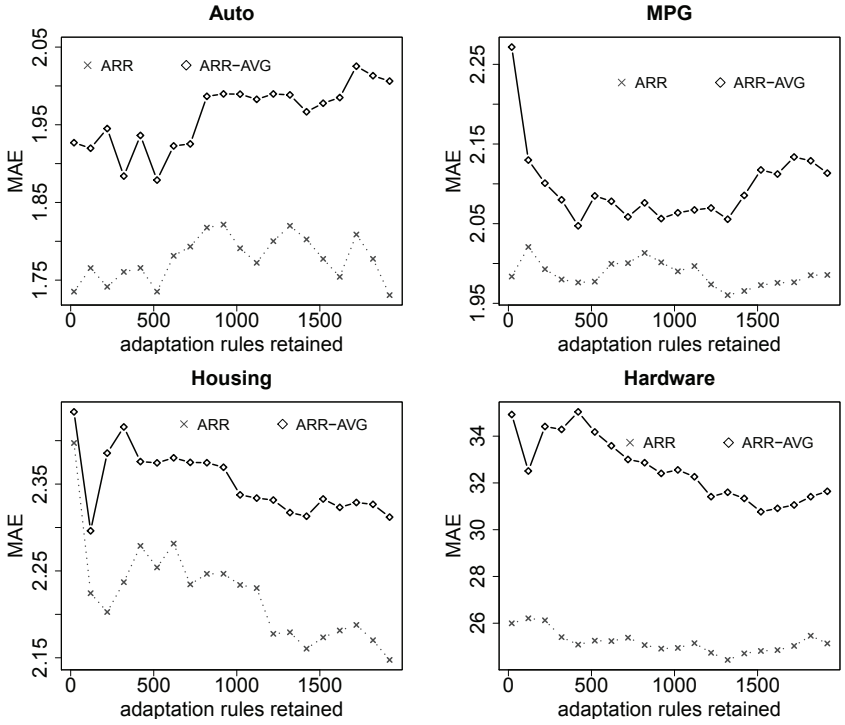

Fig. 2. MAE for retention by ARR1 and ARR-AVG

Comparative Benefit of Increasing Case or Adaptation Knowledge: To investigate question 4, on the comparative benefit of adding case or adaptation knowledge, we tested how increases in the number of adaptation rules generated by ARR1 affect final performance, versus how increases in the number of cases affect final performance. This experiment explores how sensitive performance is to additions of either type of additional knowledge, which gives an indication of how rapidly the knowledge in that container "converges" to an adequate set. In particular, the experiment measures the gain in accuracy per addition to a target knowledge container (the case base or adaptation knowledge container), compared to the performance of the system for the target knowledge container with minimum size.

Because the effect of case additions depends on the strategy used for selecting cases to add, we tested three methods: (1) Smyth and McKenna's [25] (RelCov), as well as two simplified methods, (2) Cov, which added the new case with the highest competence (Cov) and (3) Reach, which added the new case with the lowest reachability.

Fig. 4 summarizes the results. In the lefthand two graphs, the X axis shows the case base size; in the righthand two graphs, the X axis shows the adaptation rule set size. The Y axis shows the incremental percent improvement from adding another case (in the lefthand graphs) or another rule (in the righthand graphs). The minimum knowledge

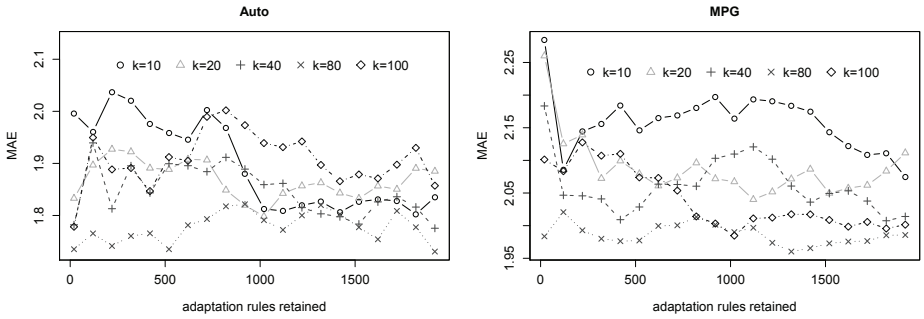

Fig. 3. The effect of the number of source cases used in the training phase on ARR's performance

container size for Auto, MPG, Housing and Hardware domains is 10, 20, 20, and 10 respectively.

As expected, increases in size of either target knowledge container generally improve performance. However, ARR1 tends to show less improvement per addition than Cov, Reach, and RelCov. We conclude that once a minimum number of adaptation rules/cases critical for system performance has been retained, performance is less sensitive to additions to the adaptation knowledge container than to additions to the case base.

6 Conclusion and Future Directions

This paper introduced ARR, a general approach to guiding adaptation rule retention. ARR uses a blame/credit assignment mechanism for guiding rule retention based on testing sample adaptation problems. Empirical results for ARR1, a specific instantiation of ARR, showed improvement over k-NN, two other alternative rule retention methods, and an ablated version of ARR with a baseline blame/credit assignment method. Experimental results also showed that ARR1's rule retention process may converge faster than case retention, suggesting a coordinated adaptation and case acquisition strategy of retaining a limited number of rules but continuing case addition.

Future directions for this work include studying ARR's performance for simultaneous maintenance of source case and adaptation knowledge containers (ARR assumes that the source case knowledge container is fixed), and investigating how local coverage characteristics of the case base may affect the choice of whether to retain cases or rules. Another direction is to extend ARR1 by applying ensembles of adaptations and studying the resulting trade-offs in accuracy and time complexity of the algorithm, as well as examining more sophisticated score assignment methods to balance the accuracy and usage frequency of the applied rules in the training phase. Because the applicability of the general ARR method is not restricted to adaptation knowledge generated by any particular method, another interesting avenue would be to explore its use for adaptation knowledge retention in other contexts and for other tasks.

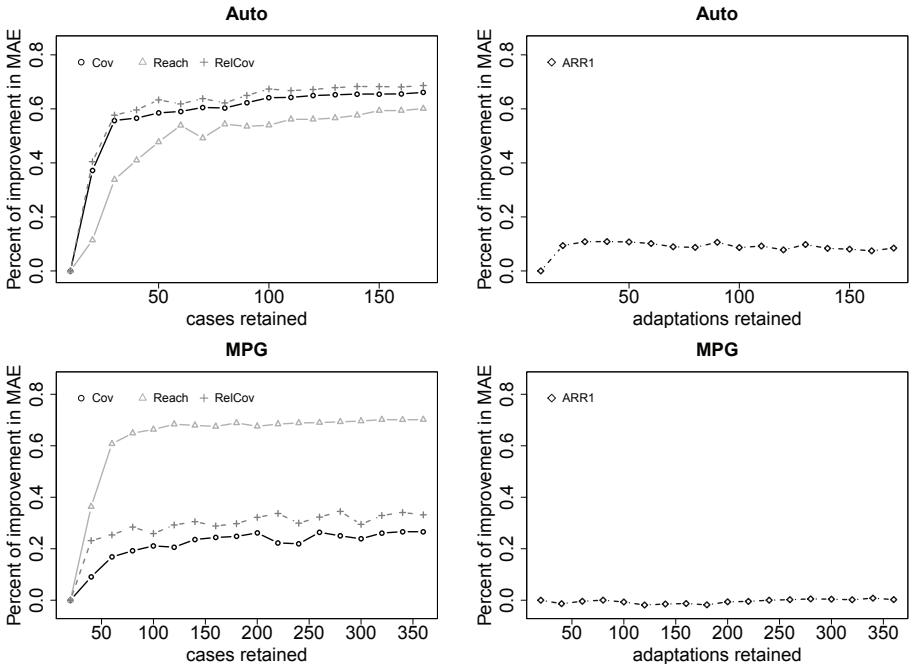

Fig. 4. Percent of improvement in MAE compared to the minimized knowledge container

References

1. d'Aquin, M., Badra, F., Lafrogne, S., Lieber, J., Napoli, A., Szathmary, L.: Case base mining for adaptation knowledge acquisition. In: Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07), San Mateo, Morgan Kaufmann (2007) 750–755
2. Wiratunga, N., Craw, S., Rowe, R.: Learning adaptation knowledge to improve case-based reasoning. Artificial Intelligence **170** (2006) 1175–1192
3. Hanney, K., Keane, M.: Learning adaptation rules from a case-base. In: Proceedings of the Third European Workshop on Case-Based Reasoning, Berlin, Springer Verlag (1996) 179–192
4. Hanney, K., Keane, M.: The adaptation knowledge bottleneck: How to ease it by learning from cases. In: Proceedings of the Second International Conference on Case-Based Reasoning, Berlin, Springer Verlag (1997) 359–370
5. Jalali, V., Leake, D.: Extending case adaptation with automatically-generated ensembles of adaptation rules. In: Case-Based Reasoning Research and Development, ICCBR 2013, Berlin, Springer (2013) 188–202
6. Leake, D., Kinley, A., Wilson, D.: Learning to integrate multiple knowledge sources for case-based reasoning. In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann (1997) 246–251

7. McDonnell, N., Cunningham, P.: A knowledge-light approach to regression using case-based reasoning. In: Proceedings of the 8th European conference on Case-Based Reasoning. ECCBR'06, Berlin, Springer-Verlag (2006) 91–105
8. McSherry, D.: An adaptation heuristic for case-based estimation. In: Proceedings of the Fourth European Workshop on Advances in Case-Based Reasoning. EWCBR '98, London, UK, UK, Springer-Verlag (1998) 184–195
9. Patterson, D., Anand, S., Dubitzky, W., Hughes, J.: Towards automated case knowledge discovery in the M^2 case-based reasoning system. In: Knowledge and Information Systems: An International Journal, Springer Verlag (1999) 61–82
10. Shiu, S., Yeung, D., Sun, C., Wang, X.: Transferring case knowledge to adaptation knowledge: An approach for case-base maintenance. Computational Intelligence **17**(2) (2001) 295–314
11. Wilke, W., Vollrath, I., Althoff, K.D., Bergmann, R.: A framework for learning adaptation knowledge based on knowledge light approaches. In: Proceedings of the Fifth German Workshop on Case-Based Reasoning. (1997) 235–242
12. Leake, D., Smyth, B., Wilson, D., Yang, Q., eds.: Maintaining Case-Based Reasoning Systems. Blackwell (2001) Special issue of *Computational Intelligence*, 17(2), 2001.
13. Richter, M.: Introduction. In Lenz, M., Bartsch-Spörl, B., Burkhard, H.D., Wess, S., eds.: CBR Technology: From Foundations to Applications. Springer, Berlin (1998) 1–15
14. Wilson, D., Leake, D.: Maintaining case-based reasoners: Dimensions and directions. Computational Intelligence **17**(2) (2001) 196–213
15. Craw, S.: Introspective learning to build case-based reasoning (cbr) knowledge containers. In Perner, P., Rosenfeld, A., eds.: Machine Learning and Data Mining in Pattern Recognition. Volume 2734 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2003) 1–6
16. Leake, D., Powell, J.: Mining large-scale knowledge sources for case adaptation knowledge. In Weber, R., Richter, M., eds.: Proceedings of the Seventh International Conference on Case-Based Reasoning, Berlin, Springer Verlag (2007) 209–223
17. Minor, M., Bergmann, R., Gorg, S.: Case-based adaptation of workflows. Information Systems **40** (2014) 142–152
18. Fuchs, B., Lieber, J., Mille, A., Napoli, A.: Differential adaptation: An operational approach to adaptation for solving numerical problems with CBR. Knowledge-Based Systems (2014) In press.
19. Li, H., Hu, D., Hao, T., Wenyin, L., Chen, X.: Adaptation rule learning for case-based reasoning. In: Semantics, Knowledge and Grid, Third International Conference on. (2007) 44–49
20. Leake, D., Dial, S.: Using case provenance to propagate feedback to cases and adaptations. In: Proceedings of the Ninth European Conference on Case-Based Reasoning, Springer (2008) 255–268
21. Jalali, V., Leake, D.: Adaptation-guided case base maintenance. In: Proceedings of the Twenty-Eighth Conference on Artificial Intelligence, AAAI Press (2014) In press.
22. Smyth, B., Keane, M.: Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems. In: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, San Mateo, Morgan Kaufmann (1995) 377–382
23. Jalali, V., Leake, D.: A context-aware approach to selecting adaptations for case-based reasoning. In: Modeling and Using Context. Springer, Berlin (2013) 101–114
24. Frank, A., Asuncion, A.: UCI machine learning repository (2010) `http://archive.ics.uci.edu/ml`.
25. Smyth, B., McKenna, E.: Building compact competent case-bases. In: Proceedings of the Third International Conference on Case-Based Reasoning, Berlin, Springer Verlag (1999) 329–342