Appears in:
Proceedings 12th IEEE International Conference on Tools with Artificial Intelligence, pp. 376–383, 2000.
© 2000 by IEEE, ISBN 0-7695-0909-6

Self-Optimising CBR Retrieval*

Jacek Jarmulak[1] Susan Craw[1] Ray Rowe[2]

[1]School of Computer and Mathematical Sciences,
The Robert Gordon University,
St Andrew Street, Aberdeen, AB25 1HG, Scotland
`{jacek|s.craw}@scms.rgu.ac.uk`

[2]AstraZeneca
Silk Road Business Park,
Macclesfield, Cheshire, SK10 2NA, UK
`Ray.Rowe@astrazeneca.com`

Abstract

One reason why Case-Based Reasoning (CBR) has become popular is because it reduces development cost compared to rule-based expert systems. Still, the knowledge engineering effort may be demanding. In this paper we present a tool which helps to reduce the knowledge acquisition effort for building a typical CBR retrieval stage consisting of a decision-tree index and similarity measure. We use Genetic Algorithms to determine the relevance/importance of case features and to find optimal retrieval parameters. The optimisation is done using the data contained in the case-base. Because no (or little) other knowledge is needed this results in a self-optimising CBR retrieval. To illustrate this we present how the tool has been applied to optimise retrieval for a tablet formulation problem.

1. Introduction

Case-based reasoning (CBR) is a problem-solving methodology that finds solutions to new problems by analysing previously solved problems [12]. At the centre of a CBR system is a collection of solved cases, a case-base. Given a new problem to be solved, the most similar problems from the case-base are retrieved. Their solutions may be directly applicable to the new problem, although some adaptation of these solutions may be necessary to fit the new problem better. The proposed solution may then be stored in the case-base and in turn be used to solve future problems.

CBR has become widely popular because of a number of advantages. Among them, the ability to utilise existing data as cases provides an opportunity to reduce development time. Nevertheless, knowledge engineering effort required for constructing CBR systems for some types of problems may still be significant [6]. Therefore, there is a need for tools and methods which contribute to reducing this effort.

The knowledge contained in a CBR system can be considered as distributed over several "knowledge containers" [17]: cases themselves, case representation, indexing knowledge, similarity knowledge, adaptation knowledge. Some of this knowledge is easily available, some has to be entered by the expert, and some can be discovered using AI and statistical techniques. In this paper we describe how the knowledge needed in CBR retrieval may be acquired automatically using just the available data, requiring no, or only minimal, expert input. We use cases contained in the case-base to determine the relevance/importance of the case features and the best parameters for the retrieval. This results in self-optimising retrieval. Our longterm goal is to apply these methods both to the (initial) knowledge acquisition as well as to the refinement of knowledge already contained in a CBR system.

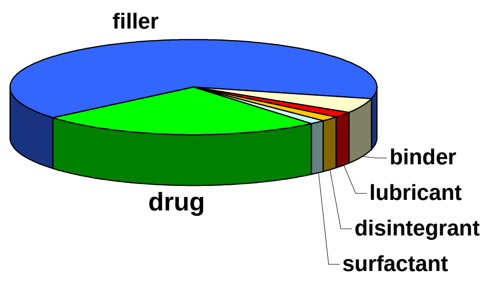
In Section 2 we describe both CBR and present our problem domain, tablet formulation. Section 3 discusses CBR retrieval in more detail and shows ways of optimising it. Details of our optimisation approach are discussed in Section 4, and some experimental results are presented in Section 5. We finish by discussing related research in Section 6, followed by Conclusions in Section 7.

2. CBR for Tablet Formulation

The tool we have developed for optimising CBR retrieval is generic and can be applied to many different problem domains. In this paper we illustrate its use on a tablet formulation domain. This domain is multi-faceted and shows well the strengths as well as some weaknesses of this optimisation approach.

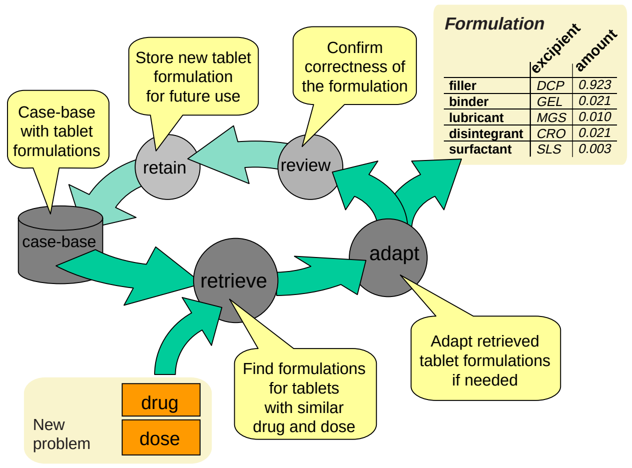
The design of a new tablet involves identifying inert substances called *excipients* to balance the properties of the

*This work is supported by EPSRC grant (GR/L98015) awarded to Susan Craw.



DRUG:
🟩 Active ingredient (typically 25%).
FILLER:
🟦 To increase bulk in order to produce a tablet of practical weight for compression (typically 65%).
BINDER:
⬜ To impart cohesive properties to the powders by the formulation of granules.
LUBRICANT:
🟥 To reduce interparticulate friction, prevent adhesion of powder to the surfaces of punches and dies and to facilitate tablet ejection from the die.
DISINTEGRANT:
🟨 To facilitate rapid breakup and disintegration after administration.
SURFACTANT:
⬜ To aid wetting and dissolution of the drug.

Figure 1. Tablet and its ingredients.


Case-base with tablet formulations
Store new tablet formulation for future use
Confirm correctness of the formulation
retain
review
case-base
retrieve
adapt
Find formulations for tablets with similar drug and dose
Adapt retrieved tablet formulations if needed
New problem
drug
dose

Formulation

	excipient	*amount*
filler	*DCP*	*0.923*
binder	*GEL*	*0.021*
lubricant	*MGS*	*0.010*
disintegrant	*CRO*	*0.021*
surfactant	*SLS*	*0.003*

Figure 2. Case-Based Reasoning for tablet formulation.

drug (the medically active component) so that a tablet is manufactured in a robust form, and the desired dose of drug is delivered and absorbed by the patient. Excipients play the role of fillers, binders, lubricants, disintegrants and surfactants in the tablet, Figure 1. The difficulty of this formulation task arises from the need to select a set of mutually compatible excipients, whilst at the same time satisfying a variety of other constraints. A formulation also specifies the quantity of each of the added excipients. Thus, full formulation requires the determination of 5 nominal and 5 numeric values.

Our collaborator AstraZeneca currently uses a rule-based system TFS [18] to create tablet formulations. This application required a large knowledge engineering effort [5] and therefore we now work on constructing an equivalent system using a CBR methodology. Our goal is to develop techniques which ease the building of CBR systems in general. These techniques would then be used to build other CBR systems for AstraZeneca (e.g. injection formulation or tablet coating) or other industries.

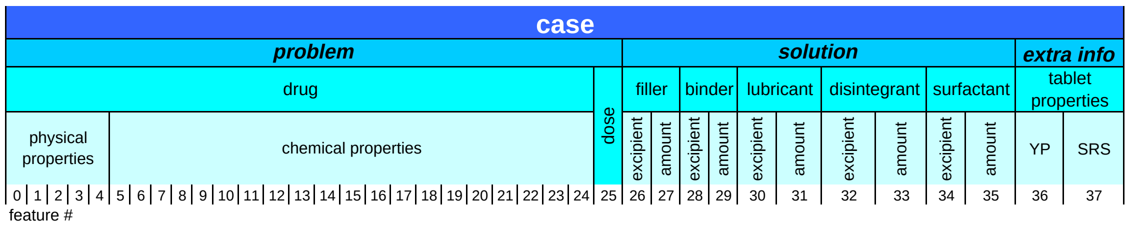
The idea of case-based tablet formulation is depicted in Figure 2. When a new tablet for a given drug and dose is to be formulated, we first retrieve similar tablet formulations from the case-base. Our case-base contains feature-value vectors representing previously formulated tablets, Figure 3. The formulations suggested by the retrieved tablets may be adapted to give a formulation compatible with our drug and having acceptable mechanical properties. The suggested formulation may be reviewed – this may include actually making a tablet according to the formulation – and if found to be correct the new formulation is stored in the case-base.

It is possible to determine all the excipients and their amounts in a *single* retrieval. However, current practice with tablet formulation suggests that the excipients and their amounts can be chosen sequentially, e.g., filler, filler amount, binder, binder amount, etc. Experiments have shown that this indeed gives more accurate results. But this also means that retrieval has to be done 10 times for a single problem; each retrieval, in principle, requiring different retrieval parameters. That is where the availability of a tool which automatically determines optimal parameters for retrieval is particularly useful.

3. CBR Retrieval and its Optimisation

In this paper we are concerned with automatically optimising the retrieval stage of CBR, so that the most useful cases are retrieved from the case-base, thereby reducing the need for adaptation. Our retrieval stage consists of decision-tree index and similarity measure, see left part of Figure 4, and is typical for CBR. This type of retrieval is widely used in commercial CBR tools, such as: ReCall (ISoft, www.isoft.fr)[1], Kate (AcknoSoft, www.acknosoft.com), and The Easy Reasoner (The Haley Enterprise, www.haley.com).

[1]ReCall is the CBR tool we use and we thank ISoft for its software contribution.



case																																					
problem																										solution										extra info	
drug																									dose	filler		binder		lubricant		disintegrant		surfactant		tablet properties	
physical properties					chemical properties																					excipient	amount	excipient	amount	excipient	amount	excipient	amount	excipient	amount	YP	SRS
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37

feature #

Figure 3. Contents of a case.

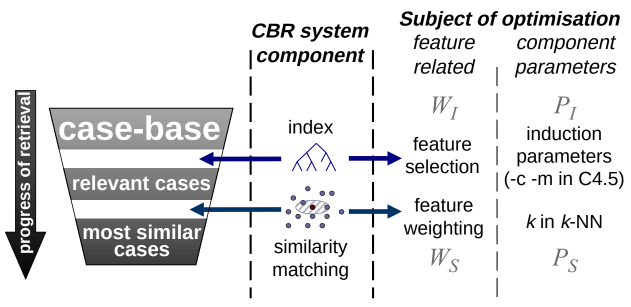


Figure 4. Optimising the index and similarity measure for CBR retrieval.

3.1. Decision-Tree Index

In our CBR system, the C4.5 induction algorithm [16] is used to build a decision tree that is then used not as a classifier but as an index for the case-base. A classifier would assign to each leaf of the tree a class corresponding to the majority class of the data examples for that leaf and the training data examples would then be discarded. Instead, when a decision-tree is used as an index, the training data examples (cases) are kept in the leaves. During retrieval, this index guides the search to the group of cases stored in a leaf node. These cases are deemed to be relevant since the chain of decisions that discriminated them from others when building the tree is also true of the new problem. From these cases we then select the most similar ones using a further similarity matching step.

Using an index serves two purposes: the retrieval is faster because fewer cases have to undergo similarity matching; and the retrieval accuracy is increased because partitioning the cases by the index prevents irrelevant cases from being considered in the similarity matching. The accuracy of retrieval is influenced by the form of the index, which in turn depends on the case data, the parameters used to induce the index (e.g. `-m` and `-c` in C4.5), and which case features are actually used in the induction, as we may exclude some of the features (e.g. the irrelevant ones) from being considered in the decision nodes. We shall improve the index by finding the optimal feature selections (binary weights), denoted by W_I, and the optimal parameters `-c` `-m` for index induction, denoted by P_I, see Figure 4.

3.2. Similarity Matching – k Nearest Neighbour

After relevant cases have been selected by the index we select those that are most similar to the new problem. The similarity can be determined in many ways, however, when cases are represented as feature vectors, calculating the weighted sum of feature distances (e.g. a Hamming or Euclidean distance) is the most common approach [22].

Usually, several most similar cases are retrieved, this is called k nearest neighbour (k-NN) retrieval. Their solutions are combined; for example, using some voting technique. Values of k larger than 1 are used to improve the generalisation properties of the retrieval, reduce sensitivity to noise, and obtain more accurate results (by interpolation). The accuracy of k-NN retrieval depends on the choice of the value of k as well as on the feature weights used in the similarity measure. We shall improve accuracy of this component of retrieval by finding the optimal feature weights for the similarity measure, denoted W_S, and by finding the optimal k parameter for k-NN, denoted P_S, see Figure 4.

3.3. Self-Optimising Retrieval

We have seen that CBR retrieval typically contains two components: a decision-tree index and similarity matching. We want to optimise both of them to achieve best retrieval results. It is important to note that C4.5 is a learning algorithm intended for classification problems. Using an index which corresponds to the maximum accuracy classifier for the case-data does not always lead to the optimal *retrieval* results. This also means that optimising of the index should be done when it is part of the retrieval, as the index has to choose good cases for similarity matching. Also the weights needed by the similarity measure may be different when the similarity is calculated for only a selection of cases in the leaves of the index and not for all cases. The best retrieval

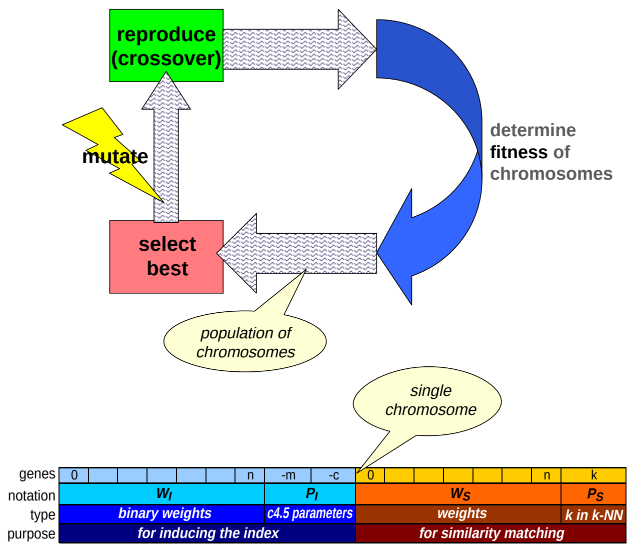


Figure 5. GA for optimising retrieval.

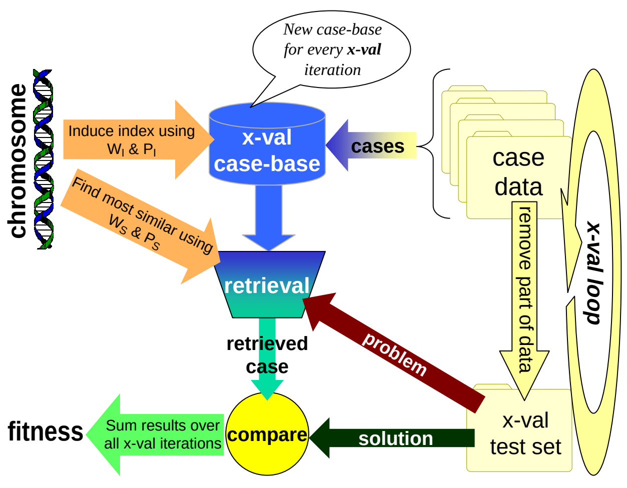

New case-base for every x-val iteration
chromosome
Induce index using W_I & P_I
x-val case-base
cases
case data
Find most similar using W_S & P_S
retrieval
remove part of data
x-val loop
retrieved case
problem
fitness
Sum results over all x-val iterations
compare
solution
x-val test set

Figure 6. Calculating fitness.

results will be achieved when the two components are optimised simultaneously. Earlier work has shown that this is indeed true [7].

Thus, the tool simultaneously optimises feature weights[2] for the index and similarity measure (W_I, W_S) and the other parameters for index induction and similarity matching (P_I, P_S). Notice that feature weighting for the index induction and similarity matching are not related: a feature may be ignored (weight zero) for constructing the index, while it may have a non-zero weight in the similarity measure.

If the cases contained in the case-base represent the problem domain reasonably well, one can use these cases to optimise the retrieval in a way that will result in improved performance of the system on new cases. Basing the optimisation only on the cases from the case-base means that the retrieval stage will be self-optimising, which is a particularly attractive feature of this approach.

4. Optimisation Approach

Optimisation is performed as a search in the space of possible parameters. Finding only the optimal parameters P_I & P_S would be trivial because of the small size of the search space. However, feature weighting is more difficult because of the large size of the search space – for the tablet formulation problem we need weights for about 30 features both for the index and similarity measure.

[2]In the following we consider feature selection as a special case of feature weighting, with just 2 weights, and do not address it separately.

4.1. GA Search

We have chosen Genetic Algorithms (GA) [13] as the search method. GAs are well suited to the high dimensionality of the search space and the combination of the crossover operator and selection preserves successful groups of feature weights. Our GA represents parameters (P_I, P_S) and feature weights (W_I, W_S) as real-valued genes in the GA chromosome, Figure 5. In each iteration of the GA, the population of chromosomes undergoes the process of selection, mutation, and reproduction. The selection type used is rank selection with elitism. New chromosomes replace old members of the population if their fitness is higher. We next define a fitness function that estimates the retrieval quality from the case-base using a decision-tree induced using feature weights W_I and parameters P_I and applying a k-NN algorithm with the feature weights W_S and parameters P_S. By supplying fitness feedback from the complete CBR retrieval process we achieve simultaneous optimisation of both indexing and similarity matching.

4.2. GA Fitness Calculation

In Figure 6 we schematically present how the fitness of a chromosome is determined. Because optimisation is highly susceptible to data over-fitting we need to incorporate cross-validation in our fitness function. Therefore, we apply cross-validation every time we evaluate the fitness of a chromosome, the “x-val loop” in Figure 6. A leave-n-out cross-validation partitions the original case-base repeatedly into an x-val test set containing data for n cases, and an x-val case-base containing the remaining cases. In each cross-validation experiment a new decision-tree index is induced for the x-val case-base using the feature weights W_I and parameters P_I from the chromosome. The similarity matching applies the weights W_S and the parameter P_S from the chromosome. In this way we have defined an x-val CBR system which is now evaluated on the x-val test set. The solution predicted by the x-val CBR system is compared with the actual solution; this is known since the x-val test set is a subset of the cases from the original case-base. Finally, the average over all cross-validation experiments returns the fitness for the given feature weightings W_I, W_S and parameters P_I, P_S.

4.3. Using the Tool

The proper way to use the tool involves two phases: (1) first we have to determine if the optimisation can be safely applied to the given combination of the case-base data and the target problem; and if this is true then (2) the actual optimisation can be performed. The reason why such an approach is required is that optimisation involves a danger of data over-fitting. This is true even though cross-validation is built into the calculation of the fitness, as we could see in the previous Section. Whether the optimisation will suffer from over-fitting will depend mainly on the problem type and the amount of available data, but may also depend on, for example, the number of possible feature weight values. In order to see if optimisation can be safely applied to the case-base data, we have to perform testing using a separate evaluation set. The tool does it by running a cross-validation, as shown in Figure 7. The figure shows the already described optimisation cross-validation (x-val loop, also in Figure 6) and the cross-validation for evaluation (e-x-val loop). It is important to stress that the e-x-val cross-validation loop is performed in order to evaluate the results of optimisation, and not to do the optimisation itself.

A single run of the evaluation phase, for a given case-base, will return the retrieval accuracies on the evaluation sets for each of the e-x-val cross-validation iterations. The average of these accuracies will show us the expected gain from optimisation, and the variance will tell us how confident we can be that these gains will actually be achieved. Each of the e-x-val iterations also returns a set of feature weights and parameters found to be optimal for that optimisation set. The degree of correlation between the found weights and parameters may give us an indication how well they represent underlying domain knowledge such as the importance of features. We can also compare the gains in accuracies, relative to the non-optimised retrieval, for the optimisation sets versus the evaluation sets; this is another method to determine if our target problem is susceptible to over-fitting. Once we are convinced that the optimisation will bring the desired results, without over-fitting the data, we can run the optimisation on all the case-base data to obtain the final W_I, W_S, P_I, P_S. Alternatively, we might also

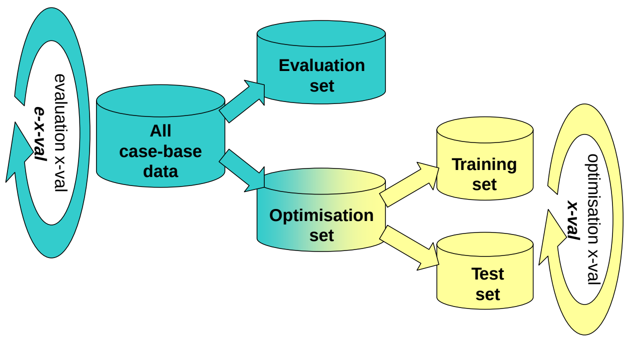


Figure 7. Partitioning of the case-base data for optimisation and evaluation.

obtain these parameters as the average of those found during each e-x-val loop of the evaluation phase.

5. Experiments and Results

In the experiments we have used our tool to optimise retrieval for predicting five excipients and their amounts, as well as two tablet properties YP and SRS that will be needed for the CBR adaptation stage. Because all tablets in our data set contained the same lubricant we omit results for lubricant prediction, although lubricant amount predictions are still included. Predicting surfactant amount has turned out to be very simple, with even non-optimised 1-NN retrieval returning 100% accuracy, so we also do not present these results here. The data set we used contained 156 tablet formulations for 39 different drugs. This data was generated by the rule-based TFS (see Section 2) and it represents a realistic set of correct formulations. Earlier experiments described in [7] suggested that use of a smaller number of possible weights may reduce the danger of over-fitting for small data sets. In experiments described here we used only two possible weight values: 0 and 1. Later, for some of the prediction problems which are not affected by over-fitting much, we plan to use more possible weight values. It is interesting to note that the tablet-formulation expert would use only two possible weights when asked to specify feature importance.

For each optimisation the GA population contained 100 individuals. We ran the GA through 100 cycles, each cycle consisting of a mutation and crossover phase. In each mutation phase 50 "mutants" were added to the population, and in each crossover phase 50 "children" were added, before selection of the most fit individuals reduced the population back to 100. The e-x-val cross-validation was 9-fold. Time needed for optimisation is not a critical issue, because optimisation will be done off-line, and is only applied rarely.

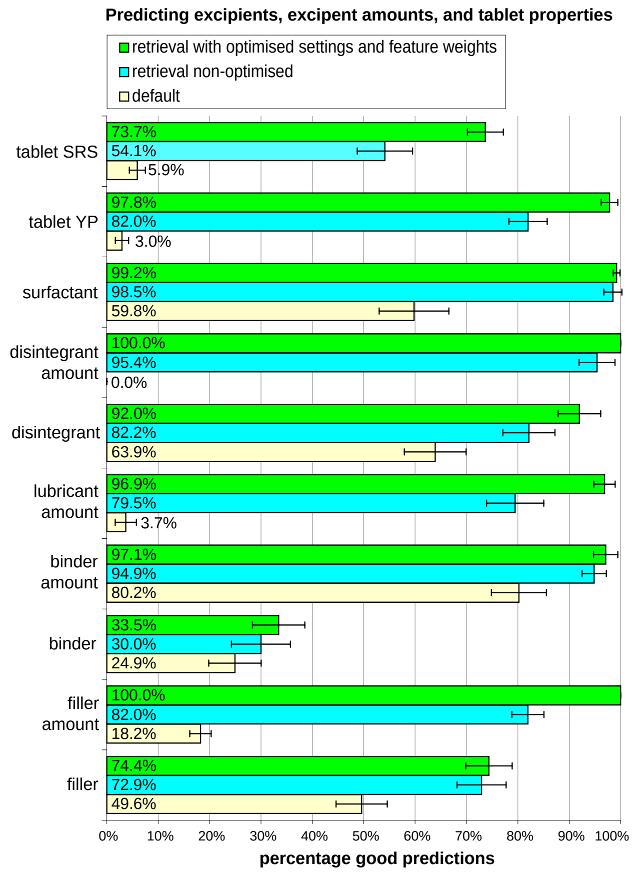


Figure 8. Results for excipient type, excipient amount and tablet property prediction.

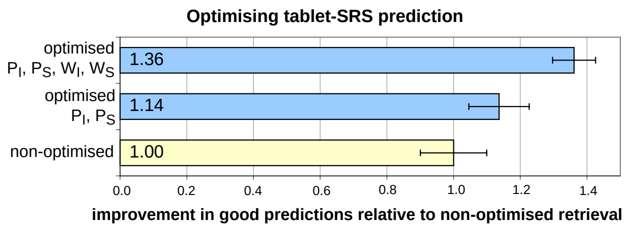


Figure 9. Results for tablet SRS prediction.

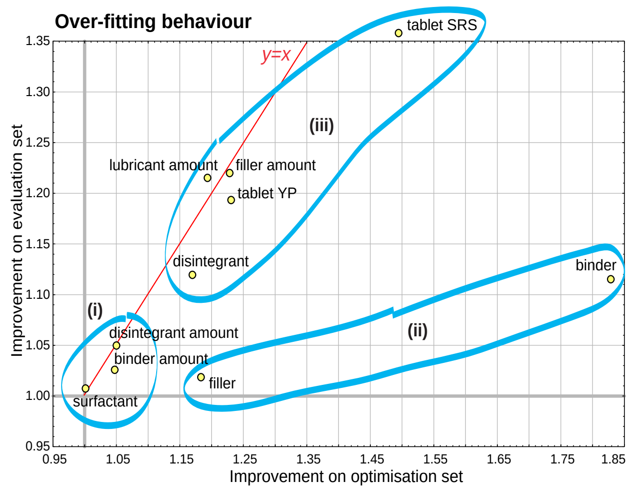


Figure 10. Analysis of over-fitting.

Moreover, the extensive e-x-val cross-validation has to be done mainly for the first use of a given case-base, since maintenance optimisation does not necessarily require evaluation cross-validation. The average time required for a single evaluation run (9 times e-x-val loop) was about 5 hours on 400MHz SPARC workstation, a single optimisation run took about half an hour. We note that we made no attempt to optimise the GA used and shorter processing times are possible.

The results of experiments are summarised in Figure 8. It shows the accuracy on the evaluation set averaged over 5 runs of the tool, where the e-x-val splits are random, in this way we obtained meaningful 95% confidence intervals. For each retrieval target we show: (a) default accuracy, either the majority class or the average value; (b) accuracy of retrieval with standard C4.5 settings and 1-NN, no feature weighting; (c) accuracy with optimised `-m`, `-c`, k and feature weights.

The results from Figure 8, can be divided into 3 groups: (1) where the optimisation does bring an improvement that is statistically significant, e.g. for tablet YP and SRS; (2) where the results without optimisation are already so good that optimisation brings no significant improvement, e.g. for surfactant and disintegrant amounts; (3) where optimisation brings hardly any improvement due largely to over-fitting, e.g. for filler and binder. We should note that even when optimisation brings no improvement in accuracy, as in case (2), it may be useful because it may identify the relevant features and thus discover useful knowledge. In our experiments we found, for example, that there was a perfect correlation between the type of disintegrant used and its amount, and that the chosen surfactant depended only on the value of one of the physical drug properties. Generally, the gains from optimisation are the highest where we know that there are many irrelevant features, like in the case of filler amount prediction. The benefit of optimising all the parameters and weights, compared to optimising only parameters, is illustrated in Figure 9.

In Figure 10 we analyse the problem of over-fitting more closely. It shows the average improvement in accuracy relative to the non-optimised retrieval for optimisation set versus the same for the evaluation data set. In the figure, we would ideally like our results to lie along the line y=x, meaning that the improvement on the evaluation set corresponds to that on the optimisation set. Results too far below the line y=x suggest data over-fitting by the optimised retrieval because the improvement for the optimisation set is not reflected in the evaluation set. Results lying above the line y=x are unlikely and are obtained purely by chance. In the plot of the results of our experiments we can distinguish three groups: (i) results around the point (1,1) show little danger of over-fitting but also minimal gain from optimisation; (ii) results with great danger of over-fitting, like the filler and binder prediction; and (iii) results which do not suffer much from over-fitting. We can use a chart like this to decide which retrieval targets should be optimised before being used in a full CBR system. In this case, we would apply optimisation for the retrievals in group (iii) to derive retrieval parameters for our tablet formulation CBR system. We could also apply the optimisation to some retrievals from group (i) if we are interested in feature relevance in addition to simply increasing retrieval accuracy. For the other problems (ii) it seems that the amount of data is not sufficient to represent the underlying problem well, and applying optimisation might reduce system accuracy due to over-fitting.

6. Related work

The subject of optimising decision trees or k-NN classifiers by feature selection or weighting has been extensively treated in Machine Learning literature. The novel element in our tool is combining the decision-tree and k-NN optimisation into optimising CBR retrieval as a whole. Previously we studied simultaneous optimisation for the filler-selection formulation sub-problem [7]. Here, we extended it to the whole formulation problem, using the optimisation approach that was shown to give the best results in the experiments for filler selection.

In our experiments the greatest improvement came from optimising feature weights, see Figure 9. Indeed, it is well-known that inducing decision trees for classification of data examples with many irrelevant features results in sub-optimal results [3], even though induction itself performs feature selection. This has made many researchers look for methods for identifying and removing irrelevant features before induction starts [8]. The benefit of feature selection is even more pronounced for nearest-neighbour classifiers [2, 19] and these are often further improved by choosing feature weights for the similarity measure [20, 10, 1]. This is especially useful for problems where the relative importance of features plays a role in addition to simple feature relevance.

The preferred way to determine the feature selections or weights is to perform a search [8, 2, 20] using the ML algorithm under optimisation to evaluate the intermediate results. This method of optimisation is called a *wrapper* [4]. Genetic Algorithm (GA) wrappers are frequently used to search for feature selections or weightings for k-NN algorithms [9, 21, 23, 15]. In [15], optimisation of the similarity weights for k-NN retrieval in a genuine CBR system is described. The GA fitness is the degree of match between the ranking of the retrieved cases and that defined by the expert. However, this approach is expert intensive and was subsequently abandoned [14], and this underlines the advantage of a fully automatic (self-)optimisation.

7. Conclusions

We have presented a tool which uses a Genetic Algorithm to determine the optimal parameters and feature weights for the retrieval stage of a CBR system. The optimisation is done using no other knowledge apart from that contained in the cases already present in the case-base, resulting in automatic self-optimisation of the retrieval, thus reducing construction cost and effort. We use a GA because of its search capabilities and the flexibility of defining the fitness measure to suit the optimisation goal. We found that using cross-validation in the fitness measure was critical to obtaining good results.

The experiments have shown that using the tool indeed improves the accuracy of CBR retrieval. The results are particularly good for domains which do contain many irrelevant features; this is a common feature for cases derived from a database. Our tool can be used safely because the accuracies on the evaluation set(s) give a good indication if over-fitting may occur. If this is the case then probably the number of cases is not sufficient to cover the problem domain well. To achieve useful results even in such situations, we are going to study whether adding jitter [11] during optimisation helps to reduce the over-fitting.

As far as scalability of our approach is concerned, it is difficult to give clear cut answers. A lot depends on the underlying real complexity of the problem represented by the data set. For example, the same problem represented by more data may require comparable time for optimisation if a cross-validation setup with fewer folds is used. We have also observed that for large data sets with many irrelevant features the optimisation is relatively fast as the search system quickly identifies the irrelevant features. Analysis of scalability is made even more difficult by the random character of the search. Our approach leaves many possibilities for speed improvements, like caching indexes and a “smart” choice of the initial GA population.

We expect the techniques implemented in the tool to be particularly useful when built into a CBR shell. We are currently working on integrating it with ReCall, first by means of Tcl scripts, and when shown successful on a wider range of problems, probably into ReCall itself. We are convinced that the existing CBR shells would certainly benefit from including the option to automatically generate a retrieval stage with optimal settings, based just on the cases already present in the case-base. This would reduce the development time and cost for new CBR systems, but it could also be useful for re-optimising retrieval during maintenance of a CBR system.

References

[1] D. Aha. Feature weighting for lazy learning algorithms. In H. Liu and H. Motoda, editors, *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Norwell MA: Kluwer, 1998.

[2] D. W. Aha and R. L. Bankert. Feature selection for case-based classification of cloud types: An empirical comparison. In *Proceedings of the AAAI-94 Workshop on Case-Based Reasoning*, pages 106–112. AAAI Press, Seattle, 1994.

[3] H. Almuallim and T. G. Dietterich. Efficient algorithms for identifying relevant features. In *Proceedings of the Ninth Conference on Artificial Intelligence*, pages 38–45. Morgan Kaufman, Vancouver, 1992.

[4] A. L. Blum and P. Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2):245–271, 1997.

[5] S. Craw, R. Boswell, and R. Rowe. Knowledge refinement to debug and maintain a tablet formulation system. In *Proceedings of the 9TH IEEE International Conference on Tools with Artificial Intelligence (TAI'97)*, pages 446–453, Newport Beach, CA, 1997. IEEE Press.

[6] P. Cunningham and A. Bonzano. Knowledge engineering issues in developing a case-based reasoning application. *Knowledge-Based Systems*, 12:371–379, 1999.

[7] J. Jarmulak, S. Craw, and R. Rowe. Genetic algorithms to optimise CBR retrieval. In E. Blanzieri and L. Portinale, editors, *Advances in Case-Based Reasoning: Proceedings of EWCBR-2K*, Trento, Italy, 2000.

[8] G. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In W. W. Cohen and H. Hirsh, editors, *Machine Learning: Proceedings of the 11th International Conference*, pages 121–129. Morgan Kaufmann, 1994.

[9] J. D. Kelly and L. Davis. A hybrid genetic algorithm for classification. In *Proceedings of the 12th IJCAI*, pages 645–650, Sidney, Australia, 1991.

[10] R. Kohavi, P. Langley, and Y. Yun. The utility of feature weighting in nearest-neighbor algorithms. In *Proceedings of the European Conference on Machine Learning (ECML-97)*, 1997.

[11] P. Koistinen and L. Holmstrom. Kernel regression and backpropagation training with noise. In J. E. Moody, S. J. Hanson, and R. P. Lippman, editors, *Advances in Neural Information Processing Systems 4*, pages 1033–1039. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[12] D. B. Leake, editor. *Case-Based Reasoning: Experiences, Lessons & Future Directions*. AAAI Press, Menlo Park, CA, 1996.

[13] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1998.

[14] G. Oatley. *An Investigation of Case-Based Reasoning for Decision Support of Diagnosis in a Large-Scale Ill-Structured Domain*. PhD thesis, University of Sunderland, 2000.

[15] G. Oatley, J. Tait, and J. MacIntyre. A case-based reasoning tool for vibration analysis. In R. Milne, A. Macintosh, and M. Bramer, editors, *Applications and Innovations in Expert Systems VI: Proceedings of the BCS Expert Systems '98 Conference*, pages 132–146, Cambridge, December 1998, 1998. Springer-Verlag.

[16] J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[17] M. M. Richter. Introduction. In M. Lenz, B. Bartsch-Sprl, H.-D. Burkhard, and S. Wess, editors, *Case-Based Reasoning Technology: From Foundations to Applications*, Lecture Notes in Artificial Intelligence 1400. Springer Verlag, 1998.

[18] R. Rowe. An expert system for the formulation of pharmaceutical tablets. *Manufacturing Intelligence*, 14:13–15, 1993.

[19] D. B. Skalak. Prototype and feature selection by sampling and random mutation hill-climbing algorithms. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 293–301, New Brunswick, New Jersey, 1994.

[20] D. Wettchereck and D. W. Aha. Weighting features. In *Proceedings of the 1st International Conference on CBR (ICCBR-95)*, pages 347–358, 1995.

[21] D. R. Wilson and T. R. Martinez. Instance-based learning with genetically derived attribute weights. In *Proceedings of the International Conference on Artificial Intelligence, Expert Systems, and Neural Networks (AIE'96)*, pages 11–14, 1996.

[22] D. R. Wilson and T. R. Martinez. Improved heterogenous distance functions. *Journal of Artificial Intelligence Research*, 6:1–34, 1997.

[23] J. Yang and V. Honavar. Feature subset selection using a genetic algorithm. In Motoda and Liu, editors, *Feature Extraction, Construction and Selection - A Data Mining Perspective*. Kluwer, 1998.