Adaptation in a CBR-based Solver Portfolio for the Satisfiability Problem

Barry Hurley and Barry O'Sullivan

Cork Constraint Computation Centre,
Department of Computer Science, University College Cork, Ireland
{b.hurley|b.osullivan}@4c.ucc.ie

Abstract. The satisfiability problem was amongst the very first problems proven to be NP-Complete. It arises in many real world domains such as hardware verification, planning, scheduling, configuration and telecommunications. Recently, there has been growing interest in using portfolios of solvers for this problem. In this paper we present a case-based reasoning approach to SAT solving. A key challenge is the adaptation phase, which we focus on in some depth. We present a variety of adaptation approaches, some heuristic, and one that computes an optimal Kemeny ranking over solvers in our portfolio. Our evaluation over three large case bases of problem instances from artificial, hand-crafted and industrial domains, shows the power of a CBR approach, and the importance of the adaptation schemes used.

1 Introduction

The satisfiability (SAT) problem is defined as follows: given a propositional formula, $\phi = f(x_1, \ldots, x_n)$, over a set of variables $x_1, \ldots, x_n$, decide whether or not there exists a truth assignment to the variables such that ϕ evaluates to true.

SAT problem instances are usually expressed in a standard form, called conjunctive normal form (CNF). A SAT problem, in this form, is expressed as a conjunction of *clauses*, where each clause is a disjunction of *literals*; a literal is either a variable or its negation. The following SAT formula is in CNF:

$$\phi = (x_1 \vee x_3 \vee \neg x_4) \wedge (x_4) \wedge (x_2 \vee \neg x_3).$$

This formula comprises three clauses: the first a disjunction of literals x_1, x_3 and $\neg x_4$; the second involves a single literal x_4; the third is a disjunction of literals x_2 and $\neg x_3$. The SAT problem ϕ is satisfiable because we can set x_1, x_2 and x_4 to *true*, satisfying the first, third and second clauses, respectively.

SAT problems occur in a variety of domains such as hardware verification, security protocol analysis, theorem proving, scheduling, routing, planning, digital circuit design and artificial intelligence [1]. Deciding whether a SAT problem is satisfiable or not is usually performed by either systematic search, based on backtracking, or local search. Because the general problem is NP-Complete, systematic search algorithms have exponential worst-case run times, which has the

effect of limiting the scalability of these methods. If a SAT problem is unsatisfiable, local search algorithms, while scalable, cannot prove unsatisfiability.

Over the past decade there has been a significant increase in the number of satisfiability solving systems that have been developed. It is recognised that different solvers are better at solving different problem instances, even within the same problem class [2]. It has been shown that the best on-average solver can be out-performed by a portfolio of possibly slower on-average solvers because of complementarities amongst them, i.e. a slow on-average solver might have best performance on a particular instance. Three specific approaches that use contrasting approaches to portfolio management for the constraint satisfaction problem (CSP), SAT and quantified Boolean formula (QBF) are CPHYDRA, SATZILLA and AQME, respectively. CPHYDRA is a portfolio of constraint solvers that exploits a case base of problem solving experience [3]. CPHYDRA combines case-based reasoning with the idea of partitioning CPU-TIME between components of the portfolio in order to maximise the expected number of solved problem instances within a fixed time limit; CPHYDRA is an earlier piece of work in our research programme on portfolios, but does not consider alternative approaches to adaptation, which we study here. SATZILLAbuilds run time prediction models using linear regression techniques based on structural features computed from instances of the Boolean satisfiability problem [4]. Given an unseen instance of the satisfiability problem, SATZILLA selects the solver from its portfolio that it predicts will have the fastest running time on the instance. The AQME system is a portfolio approach to solving quantified Boolean formulae, i.e. SAT instances with some universally quantified variables [5].

The objective of the work reported in this paper is to study a simple case-based reasoning approach to a portfolio for the SAT problem. We present three large case bases of problem-solving experience with a large number of modern SAT solvers in three distinct domains, including one comprising almost 1200 industrial problems (Section 2), which we have made available online. We focus primarily on the problem of adaptation, having retrieved a suitable set of similar experiences involving problems similar to the one we wish to solve (Sections 3 and 4). Our results (Section 5) demonstrate that a case-based reasoning approach would perform close to oracle performance on the domains we evaluate, exhibiting a potential killer application domain for case-based reasoning. These results are consistent with the belief held in the SAT community that experience plays a key role in selecting a good solver for a problem instance.

2 Building Case-bases for SAT Solving

We summarise the representation, cases and similarity measure used in our three case bases for SAT solving. Our case bases relate to three domains: industrial instances, hand-crafted instances and randomly generated instances.

Feature Representation. We employed the same set of SAT instance features as those used in SATZILLA.[1] SATZILLA is a successful algorithm portfolio for

[1] `http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/`

Problem Size Features:
1. **Number of clauses**: denoted c
2. **Number of variables:** denoted v
3. **Ratio:** c/v

Variable-Clause Graph Features:
4-8. **Variable nodes degree statistics:** mean, variation coefficient, min, max and entropy.
9-13. **Clause nodes degree statistics:** mean, variation coefficient, min, max and entropy.

Variable Graph Features:
14-17. **Nodes degree statistics:** mean, variation coefficient, min and max.

Balance Features:
18-20. **Ratio of positive and negative literals in each clause:** mean, variation coefficient and entropy.
21-25. **Ratio of positive and negative occurrences of each variable:** mean, variation coefficient, min, max and entropy.
26-27. **Fraction of binary and ternary clauses**

Proximity to Horn Formula:
28. **Fraction of Horn clauses**
29-33. **Number of occurrences in a Horn clause for each variable:** mean, variation coefficient, min, max and entropy.

DPLL Probing Features:
34-38. **Number of unit propagations:** computed at depths 1, 4, 16, 64 and 256.
39-40. **Search space size estimate:** mean depth to contradiction, estimate of the log of number of nodes.

Local Search Probing Features:
41-44. **Number of steps to the best local minimum in a run:** mean, median, 10th and 90th percentiles for SAPS.
45. **Average improvement to best in a run:** mean improvement per step to best solution for SAPS.
46-47. **Fraction of improvement due to first local minimum:** mean for SAPS and GSAT.
48. **Coefficient of variation of the number of unsatisfied clauses in each local minimum:** mean over all runs for SAPS.

Fig. 1. A summary of the features used to describe SAT instances in our case base. These are the same features used in SATzilla [4].

SAT, i.e. a system that uses machine learning techniques to select the fastest SAT solver for a given problem instance. That system uses a total of 48 features, summarised in Figure 1 [4]. These features can be summarised under nine different categories: problem size features; variable-clause graph features; variable graph features; balance features; proximity to Horn formula; DPLL probing features; and local search probing features. The first category of features are self explanatory, and simply relate to the number of variables and clauses in the SAT instance. The next two categories relate to two different graph representations of a SAT instance. The variable-clause graph is a bipartite graph with a node for each variable, a node for each clause, and an edge between them whenever a variable occurs in a clause. The variable graph has a node for each variable and an edge between variables that occur together in at least one clause. The balance features are self explanatory and relate, primarily, to the distribution of positive and negative literals within the SAT instance. Another category measures the proximity to a Horn formula. This captures how close the SAT instance is to an important polynomial class of SAT that can be solved using the standard inference method used in all systematic SAT solvers (i.e. unit propagation). The DPLL probing features are related to statistics that a standard systematic search algorithm gathers while testing the difficulty of the instance [6]. The local search features are the non-systematic analogue of the latter category.

Cases. We built three case bases from the training data used by the SATzilla system [4].[2] Each case in the case base represents one SAT problem instance and the individual performance of a set of solvers when applied to it. For each

[2] SATzilla data: `http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/`

Table 1. The number of problem instances and solvers in each of our three case bases.

Case base	# Instances	# Solvers
Handcrafted (HAN)	1181	19
Industrial (IND)	1183	19
Random (RAN)	2308	27

benchmark instance in the dataset, there is a record of whether each solver solved the instance within a specified cut-off time (1 hour), the time taken by each to solve the instance, and whether the solver crashed during execution. Each solver is run independently of each other. Thus, each case is a pair, (F, S), where F is a set of feature values and S is a set of pairs encoding the performance of each solver on the query instance.

The problem instances were originally taken from the benchmark suites associated with the annual International SAT Competition.[3] We combine the instances from each year of the SAT competition into a single combined dataset. Each instance is assigned to one of three categories: handcrafted (HAN), industrial (IND) and randomly generated (RAN) problem instances. The instances are additionally separated into what SATzilla classified as satisfiable and unsatisfiable instances. For our evaluation, instances from these two classifications were combined into a single dataset. Table 1 gives the resulting size of each case base.[4] Table 1 also shows the number of solvers in each. Note that the random category contains eight additional solvers that are not available in the dataset for the handcrafted and industrial categories.

Similarlity Metric. The features that encode the cases are all numeric. For the purposes of this paper we assume that the similarity between two cases is computed using the unweighted normalised Euclidean distance. Feature values are normalised to the interval $[0, 1]$. Specifically, for the ith feature of case γ, we compute the normalised value $\eta(\gamma[i])$ of feature value $\gamma[i]$ as follows:

$$\eta(\gamma[i]) = \frac{\gamma[i] - min(i)}{max(i) - min(i)}.$$

where $min(i)$ and $max(i)$ are the minimum and maximum values respectively for feature i across all cases.

When evaluating a test case, one finds the k cases with highest similarity (smallest Euclidean distance) in the case base. The challenge is then, given the set of performance data for each solver, how should one adapt this experience to the current problem instance. We frame this problem as a label ranking task [7], which we will discuss in greater detail in the following section.

[3] `http://www.satcompetition.org`
[4] The case bases are available at `http://osullivan.ucc.ie/datasets/iccbr2012/`

3 Adaptation Strategies

The specific task of adaptation in our portfolio context is to decide which solver should be used to solve a given instance. We will consider a setting in which we are allocated one hour to solve an instance. The objective is, given a set of problem instances, to solve as many of them as possible within the cutoff in the shortest time. In other words, we lexicographically order these two objectives: maximise the number of solved instances, and tie-break by running time, which we prefer to minimise. In our setting, each nearest neighbour can be seen as giving an ordering over the performance of the available solvers. We can interpret this order as a ranking, possibly time-weighted.

In traditional classification, we are interested in assigning one or more labels from a finite set of labels, to a case. In contrast, label ranking deals with assigning a total ordering of labels to a case. This ranking of labels can be much more useful than assigning a single label, e.g. rank aggregation methods have been used to combine query results from multiple search engines [8]. In this paper, the labels represent different algorithms in our portfolio solver and the ranking of labels represents the expected order of run time on an instance.

Label ranking may also be seen as a generalization of multi-label classification. Instead of classifying a case with a subset of the classes, we instead assign a totally ordered ranking of the classes. Multi-label ranking is the task whereby in addition to producing a total ordering of the labels for an instance, the task is to also identify a partition of the labels into relevant and irrelevant labels [9,10]. This introduces an additional layer of complexity to the task. Methods for learning pairwise preferences between labels have been proposed [7]. It has been shown that case-based label ranking compares well against model-based approaches [11]. An in depth survey of additional label ranking methods is given in [12].

We consider a variety of voting-based approaches for label ranking and consensus ranking; we refer the reader to the literature for further details of the various methods [13]. We will use examples throughout, based on the sample data presented in Table 2. In this table, we present an example of the retrieval set from our case-based system, but do not present the running times. Instead we simply order the solvers by running time.

Table 2. An example list of label rankings, which we will use as a running example. Each ranking contains a total ordering of the labels a, b, c, d. The operator $\prec$ can be read as 'faster than'.

Name	Label Ranking
A	$a \prec b \prec c \prec d$
B	$c \prec b \prec a \prec d$
C	$b \prec a \prec d \prec c$
D	$a \prec c \prec d \prec b$
E	$b \prec a \prec c \prec d$

Kendal-Tau Distance. To compare two rankings, we define a function to compute the distance between them. The Kendal-Tau distance between two rankings A and B is the number of discordant pairs. Let L be the set of all labels and let A and B be two complete rankings of these labels. Formally:

$$\text{KT-distance}(A, B) = \sum_{c,d \in L,\, c \neq d} \begin{cases} 1 & \text{if } A \text{ and } B \text{ rank } c \text{ and } d \text{ in a different order} \\ 0 & \text{otherwise}. \end{cases}$$

Example 1. Using the example rankings given in Table 2, the Kendal-Tau Distance between rankings A and B is:

$$\text{KT-distance}(A, B) = 3$$

This is because the pairs of candidates (a, b), (a, c) and (b, c) are ranked differently by the two rankings. A and B both rank the other pairs, e.g. (c, d), in the same order. □

Kemeny Consensus Ranking. The Kemeny Score of a ranking R is the sum of all the Kendal Tau distances from R to all rankings among the votes V.

$$\text{Kemeny-Score}(R, V) = \sum_{v \in V} \text{KT-distance}(R, v)$$

Example 2. Let $R = \langle a \prec c \prec b \prec d \rangle$. If we take all rankings from Table 2 as the votes, then the Kemeny Score of R is 9. This is the sum of:

$$\begin{array}{lll} \text{KT-distance}(R, A) = 1 & \text{KT-distance}(R, B) = 2 & \text{KT-distance}(R, C) = 3 \\ \text{KT-distance}(R, D) = 1 & \text{KT-distance}(R, E) = 2 & \end{array}$$

□

The Kemeny Consensus is the ranking of the labels that minimises the Kemeny Score. This may also be referred to as the optimal Kemeny ranking. It is the ranking which minimises, among the votes, the number of disagreements on the pairwise preference between every pair of labels. Aggregating multiple rankings into a single optimal Kemeny ranking is NP-hard [8].

Example 3. For the votes given in Table 2, the optimal Kemeny ranking is $\langle b \prec a \prec c \prec d \rangle$ with a Kemeny Score of 7. All other possible permutations of the labels have a higher Kemeny Score than this. In this case the Kemeny Optimal ranking matches one of the rankings in the votes, but this may not necessarily be the case. □

The Kemeny Consensus ranking is said to satisfy the Condorcet criterion. This states that if a candidate is preferred by most voters to any other candidate, then it should be ranked first in the aggregation ranking. It expresses no condition on the remainder of the positions, however.

Borda Voting. Borda voting is a polynomial time approximation scheme for the Kemeny Consensus ranking. Each of the k nearest neighbours votes for each label in the order of which that solver performed on that case. A label in position p receives $n - p + 1$ points based on its position in the ranking. The points for each candidate are summed up. The ranking is produced by sorting these tallies in decreasing order. Borda voting does not satisfy the Condorcet criterion.

Example 4. For vote B in Table 2, candidates c, b, a and d would receive 4, 3, 2 and 1 points respectively. If we sum up the points from all the votes in this table, the a would have a score of 16 points, b of 15, c of 12 and d of 7. The resulting ranking would be $\langle a \prec b \prec c \prec d \rangle$. □

Weighted Borda Voting. Weighted Borda voting takes extra information about each neighbour into account. The vote for a particular solver is multiplied by the weight in one of the two weighting schemes we consider. In distance-weighted Borda voting (DW-BV), the weight W_D is given as $W_D = \frac{1}{1+d}$ where d is the Euclidean distance between the neighbour and the test instance.

In time-weighted Borda voting (TW-BV) the weight W_T is given as $W_T = \frac{\text{cutoff}-t}{\text{cutoff}}$, where 'cutoff' is the cut-off execution time limit and t is the time taken for the solver on a given neighbour. Time-weighted Borda voting gives a large weight to solvers that take very little time to solve the instance and a weight of zero to any that timeout or do not solve the instance. This suits our goal of choosing the solver that will perform fastest for a given instance.

Copeland Voting. Copeland voting looks at every pair of labels (a, b) and counts the difference between the number of votes that prefer a to b and those that prefer b to a. The label with the higher number of preferences gets one added to its score. The label with the lower number of preferences gets one deducted from its score. The resulting ranking of the labels is obtained by sorting on their respective scores.

Example 5. Given the votes in Table 2, the label ranking produced by Copeland voting would be $\langle b \prec a \prec c \prec d \rangle$. The scores for each label would be: (a, 1), (b, 3), (c, -1), (d, -3). □

Bucklin Voting. Bucklin voting is a means of choosing the label with the best median ranking. The algorithm first attempts to select the label that has a majority of first preference votes. The number of first preferences for each label is counted across the votes. If one of the labels has a majority, then that label is the winner. If no label has a majority, then the second preference votes are added to the first. Again, if there is a label that has a majority of votes, then that label is the winner. There may be multiple labels with a majority. In this case, the winner is the one with the highest vote tally.

Coomb's Voting. Coomb's voting is similar to Bucklin voting in that it first attempts to select the label that has the majority of first preference votes. If no label has a majority, a separate election is held between the labels that are ranked last in the votes. The label with the most last-place votes is then removed from all votes. A tally of the first preference votes is taken again and this is repeated until there is a label with a majority.

Instant Runoff Voting. In Instant Runoff voting (IRV), we again stop if there is a label with a majority of first place votes. If not, then the label with the fewest first preference votes is eliminated. This label is removed from each of the votes. For each vote where the eliminated label held a first place preference, the next preference votes are added to their respective label's tally. This is repeated until there is a candidate with a majority of votes. This voting scheme is similar to Coomb's voting except instead of eliminating the label that is ranked last, we eliminate the label that has the fewest first preference votes.

Best Average Score. Among the data for the k nearest neighbour instances is the run time for each solver on that neighbour instance. The Borda voting and distance weighted Borda voting methods above do not take this valuable data into account when performing their aggregation. Another strategy to get a ranking of the solvers from this data is to order them by their average score across these k instances. This gives us an ordering of the solvers by their performance, averaged across the k nearest neighbours. We refer to this aggregation as ordering by Best Average Score.

Very Best Ranking. The Very Best Ranking (VBR) is the ranking produced by an oracle, who knows the best ranking for each instance. We use this ranking as a benchmark to compare the rankings produced by the aggregation methods.

4 An Exact Method for Optimal Kemeny Ranking

The ranking methods presented in the previous section, with the exception of the optimal Kemeny and VBR rankings, are heuristics. In this section a Mixed Integer Programming (MIP) model is presented for computing the optimal Kemeny ranking from the k nearest neighbors of a query SAT instance as an optimization problem. This model was implemented using the combinatorial optimization system Numberjack[5] using SCIP as the underlying MIP solver.[6]

Let L be the set of all labels. Let V be the set of votes from each of the k nearest neighbors. We encode each ranking as a list where each label takes the value of the number of labels ranked higher than it. For example, if we are given a ranking of the labels $c \prec a \prec d \prec b$, then this would be converted to $\langle 1, 3, 0, 2 \rangle$

[5] Available under LGPL from `http://numberjack.ucc.ie/`
[6] SCIP: `http://scip.zib.de/`

because a has 1 candidate ahead of it, b has 3, and so on. This simplifies the process of finding the index of a label within a ranking for the MIP model. We define the MIP model as follows:

- R is the array of the rank indices in the Kemeny Consensus ranking. R_i states the number of labels that are ranked before label i in the aggregation ranking. The domain of values that each position in R can take is therefore $0 \ldots m-1$. This array contains all the decision variables.
- We add the constraint that the values taken by the variables in R are all different because only one candidate can occupy each position.
- For each pair of labels i and j, we have a binary variable r_{ij} which is encoded to take the value 1 if i is ranked higher than j in the target ranking R, 0 otherwise.
- For each pair of labels i and j in each vote V_k we have a binary variable v_{kij} which takes the value 1 if label i is ranked higher than label j in vote V_k, 0 otherwise.
- For each pair of labels i and j in each vote V_k we have the binary variable D_{kij} which is the exclusive-or between r_{ij} and v_{kij}. This means D_{kij} will take the value 1 iff R ranks i and j in a different order to V_k.
- The Kendal Tau distance to vote V_k from R is KT_k, which is the sum over all D_{kij} for every pair of candidates i and j.
- The Kemeny Score of the target ranking R is $\sum KT_k$. We attempt to minimise this value.

For a set of votes among 19 labels, this MIP model is able to solve the difficult aggregation problem in a matter of seconds. Consider that a greedy naive algorithm for commuting the Kemeny Optimal ranking may need to examine every possible permutation of the labels, which is $O(n!)$. It must compute the Kemeny Score for each permutation and choose the ranking that minimises this function. This approach quickly becomes infeasible.

5 Evaluation

We present an evaluation of both the quality of our adaptation strategies for ranking solvers by run time (Section 5.1), and the performance of our case-based reasoning-based solver portfolio for SAT (Section 5.2). We use the case bases described in Section 2. In terms of the quality of the rankings, we show that rankings that consider running time, rather than relative position in the rank, give better performance. This is somewhat unsurprising, but it is interesting to see that the effort spent in finding the optimal Kemeny ranking is not worthwhile.

Of much greater significance is our demonstration that our CBR portfolio out-performs all of its constituent solvers by a considerable margin. In fact, the superiority of the CBR approach is observed regardless of the adaptation scheme used. Again, rankings that consider time are superior to all others, and compare well in terms of performance against the oracle (VBR) that always selects the best solver for a particular SAT instance.

Methodology. In all experiments we used a 10-fold cross validation approach, studying each of our three case bases (Section 2) separately. We report averages, where appropriate. We always seek five nearest neighbours (5-NN), having observed that setting k to this value gave good typical-case performance. For the purpose of this paper, unweighted normalised Euclidean distance is used as a similarity metric throughout. All adaptation methods use the same distance measure, therefore each are tasked with aggregating the same set of neighbours.

5.1 Evaluation of the Adaptation Schemes

Given a ranking of the solvers, using a particular adaptation scheme, and their respective execution times, we can plot the cumulative execution time of each solver against its position in the ranking. Let $s(i)$ be the solver ranked in position i of a ranking, and $t(\alpha)$ be the time taken by solver α to solve the instance. The plot of the cumulative time of the solvers in a ranking is given by:

$$f(x) = \sum_{i=1}^{x} t(s(i)).$$

If the solvers are ordered in strict order of increasing run time, the area under the curve in this plot will be minimised. On the other hand, the ranking which is as poor as possible will have maximum area. We compare each of our adaptation strategies that produce a ranking in this way. Figure 2 shows an example plot of the curve for each of the label rankings produced by an adaptation method.

We performed paired t-tests to compare two label ranking methods on the basis of the area under the curves in our ranking plots. Such a paired t-test was performed between every pair of adaptation methods on every instance across the 10 splits in each dataset category. Table 3 gives the complete table of these results showing the 95% confidence interval and the p-value. In this table a confidence interval with negative lower and upper bounds, which is highlighted in bold, signifies that the ranking on the left is statistically significantly better than the ranking on the right.

On hand-crafted and industrial problems, which are really the most interesting from a practical viewpoint, the best-average-solver (BAS) and three variants of Borda voting out-perform all other methods; the statement is almost also true in the random category. It is clear, and not unsurprising, that the methods that take running time into account, out-perform all others. The Kemeny ranking never out-performs another method.

While comparing the rankings is interesting in itself, the more important question is how effective are these rankings in a CBR-based algorithm portfolio for SAT. We study this below.

5.2 Evaluation of the CBR-based Solver Portfolio for SAT

We implemented a variant of our basic CBR-based solver portfolio using each of our adaptation schemes in turn. We name these using the acronym of the

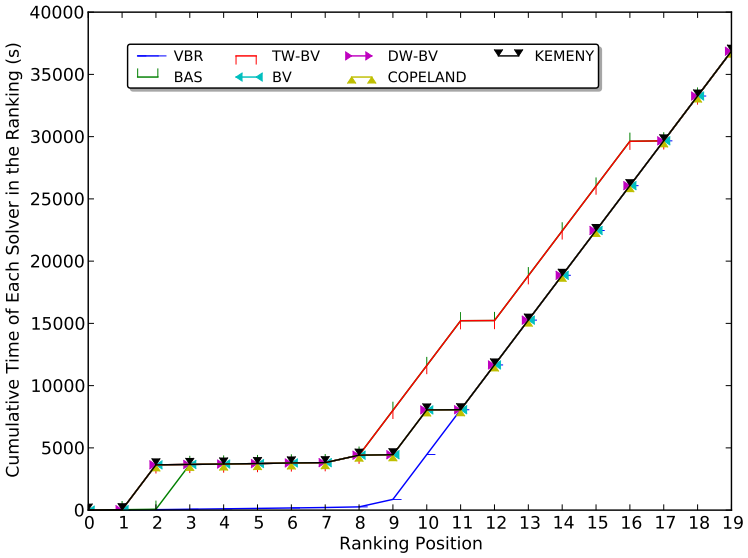

Fig. 2. An illustration of the curve produced by accumulating the time taken by each solver in a ranking. Each line represents a curve produced by the rankings from each of the adaptation methods that produce a ranking.

adaptation scheme used. Given a SAT instance, the portfolio system will apply the relevant adaptation scheme to the set of cases retrieved using our 5-NN method. The highest ranked solver is selected. We record the total number of instances solved by the chosen solvers and the cumulative time summed over all solved instances, given a cut-off time of 1 hour per instance, in a 10-fold cross validation setting. This setup is very similar to that of the International SAT Competition.

We compare this to the Very Best Ranking (VBR), which chooses the best solver for the instance given. We report the average number of instances solved and the average run time, with standard deviation in both cases. In our results tables (Tables 4, 5 and 6) we sort the variants in terms of number of instances solved, and then by run time. The VBR, the oracle, is therefore always ranked at the top. Due to space constraints, these leader boards only show the top 15 positions.

Table 3. Table of paired t-tests comparing the area under the curve for each of the methods. Results with a negative interval and p-value below the threshold of 0.05 are highlighted in **bold**. This means that the first method is statistically significantly better than the second method for that dataset. Results with a positive interval and p-value below the threshold of 0.05 are highlighted in *italics*. This means that the second method is statiscally significantly better than the first method for that dataset.

		Handcoded			**Industrial**			**Random**		
		95% Conf. Int.		p-value	95% Conf. Int.		p-value	95% Conf. Int.		p-value
VBR	BAS	**-8615.4**	**-6366.5**	**0.000**	**-7186.6**	**-5286.6**	**0.000**	**-23538.6**	**-19300.2**	**0.000**
VBR	TW-BV	**-9425.4**	**-7119.7**	**0.000**	**-7857.2**	**-5864.8**	**0.000**	**-27900.0**	**-23160.4**	**0.000**
VBR	DW-BV	**-14736.7**	**-11755.3**	**0.000**	**-10613.7**	**-8388.4**	**0.000**	**-53803.9**	**-46149.2**	**0.000**
VBR	BV	**-14902.9**	**-11866.3**	**0.000**	**-11065.8**	**-8671.7**	**0.000**	**-53795.7**	**-46126.8**	**0.000**
VBR	COPELAND	**-17103.5**	**-13394.1**	**0.000**	**-12239.6**	**-9423.4**	**0.000**	**-55608.2**	**-46523.2**	**0.000**
VBR	KEMENY	**-17173.4**	**-13463.0**	**0.000**	**-11854.5**	**-9303.8**	**0.000**	**-70833.4**	**-61940.3**	**0.000**
BAS	TW-BV	**-1196.0**	**-367.2**	**0.000**	**-876.0**	**-372.8**	**0.000**	**-5223.4**	**-2998.3**	**0.000**
BAS	DW-BV	**-7054.4**	**-4455.7**	**0.000**	**-4141.7**	**-2387.2**	**0.000**	**-31689.3**	**-25425.0**	**0.000**
BAS	BV	**-7218.2**	**-4569.1**	**0.000**	**-4600.1**	**-2664.1**	**0.000**	**-31675.3**	**-25408.4**	**0.000**
BAS	COPELAND	**-9420.1**	**-6095.6**	**0.000**	**-5787.9**	**-3401.8**	**0.000**	**-33551.5**	**-25741.1**	**0.000**
BAS	KEMENY	**-9499.7**	**-6154.9**	**0.000**	**-5396.9**	**-3288.2**	**0.000**	**-48835.7**	**-41099.2**	**0.000**
TW-BV	DW-BV	**-6203.8**	**-3743.0**	**0.000**	**-3514.4**	**-1765.7**	**0.000**	**-27354.7**	**-21538.0**	**0.000**
TW-BV	BV	**-6368.2**	**-3855.9**	**0.000**	**-3965.6**	**-2049.9**	**0.000**	**-27347.5**	**-21514.5**	**0.000**
TW-BV	COPELAND	**-8552.0**	**-5400.5**	**0.000**	**-5143.7**	**-2797.1**	**0.000**	**-29182.7**	**-21888.2**	**0.000**
TW-BV	KEMENY	**-8630.7**	**-5460.7**	**0.000**	**-4761.6**	**-2674.6**	**0.000**	**-44543.8**	**-37169.4**	**0.000**
DW-BV	BV	**-260.6**	**-16.7**	**0.026**	**-558.1**	**-177.3**	**0.000**	-136.7	167.2	0.844
DW-BV	COPELAND	**-2639.3**	**-1366.3**	**0.000**	**-1839.8**	**-821.0**	**0.000**	-2622.9	444.6	0.164
DW-BV	KEMENY	**-2719.4**	**-1425.0**	**0.000**	**-1595.1**	**-561.1**	**0.000**	**-18057.6**	**-14763.0**	**0.000**
BV	COPELAND	**-2459.8**	**-1268.6**	**0.000**	**-1361.5**	**-564.0**	**0.000**	-2625.3	416.4	0.155
BV	KEMENY	**-2543.1**	**-1324.2**	**0.000**	**-1251.7**	**-169.1**	**0.010**	**-18060.9**	**-14790.3**	**0.000**
COPELAND	KEMENY	-222.4	83.6	0.374	-339.2	843.8	0.403	**-16663.1**	**-13979.1**	**0.000**

Table 4. Leader board for the Handcrafted category of problem instances.

Approach		Solver Name	Nr. Solved	Cumulative Time on Solved Instances (s)
Oracle	1	VBR	114.9 (± 1.4)	24703.4 (± 4972.5)
CBR	2	TW-BV	110.5 (± 2.2)	25668.9 (± 5855.8)
	3	BAS	109.5 (± 2.7)	26147.9 (± 7106.6)
	4	DW-BV	109.3 (± 1.8)	26239.2 (± 6779.6)
	5	BV	109.2 (± 1.8)	25561.7 (± 6541.9)
	6	IRV	108.0 (± 1.8)	23872.8 (± 6121.4)
	7	COOMBS	107.9 (± 2.5)	24553.9 (± 6317.8)
	8	KEMENY	107.8 (± 2.4)	24163.2 (± 6683.3)
	9	BUCKLIN	107.6 (± 2.5)	22892.7 (± 6783.0)
	10	COPELAND	107.6 (± 2.5)	24060.7 (± 6140.9)
SAT	11	minisat20SAT07	88.1 (± 4.1)	33700.5 (± 9455.9)
	12	mxc08	86.2 (± 3.9)	30312.6 (± 9620.1)
	13	march_dl2004	85.1 (± 3.5)	22245.3 (± 7770.6)
	14	picosat846	84.0 (± 3.6)	28713.1 (± 8167.2)
	15	minisat2.0	82.5 (± 4.3)	32019.3 (± 7527.4)

The overall result is that the best SAT solvers are out-performed in every problem class by each of the CBR-based portfolios. The CBR portfolio compares very well against the oracle (VBR) in each category. For example, in the random problem category, the CBR portfolio solves 33% more instances than the best SAT solver on its own, and solves within 5% of the instances solved by the oracle.

Both BAS and TW-BV portfolios perform consistently well, which would not be obvious a-priori in this setting in which it is most important to solve instances within a cut-off. Once again, the Kemeny ranking is not competitive amongst the CBR-based portfolios.

These results are consistent with the expectations of experts in the field of SAT. It is regarded as a challenge to be able to select a good performing solver for a given instance, and the choice is heavily reliant on the experience of the user who makes this choice. Therefore, this domain is perfect for CBR, and the results demonstrate that it is also a very useful technique to use here.

6 Conclusions and Future Work

In this paper we studied a variety of adaptation schemes for a family of CBR-based algorithm portfolios for the SAT problem. Our results demonstrate that the choice of adaptation scheme is important for performance with schemes that consider run time rather that relative ranking gives superior performance.

We demonstrated that a CBR approach to this task is competitive, and out-performs individual high-performing SAT solvers in a wide variety of problem

Table 5. Leader board for the Industrial category of problem instances.

Approach		Solver Name	Nr. Solved	Cumulative Time on Solved Instances (s)
Oracle	1	VBR	113.1 (± 2.5)	24561.0 (± 5164.6)
CBR	2	BAS	110.3 (± 3.3)	30003.9 (± 4674.8)
	3	TW-BV	109.8 (± 3.0)	27742.0 (± 4430.4)
	4	KEMENY	105.4 (± 3.7)	26500.6 (± 5287.4)
	5	DW-BV	105.1 (± 3.4)	25699.3 (± 4611.6)
	6	BV	105.0 (± 3.5)	26364.8 (± 4243.5)
	7	COPELAND	104.5 (± 4.1)	26650.9 (± 5209.8)
	8	COOMBS	104.2 (± 3.9)	26758.2 (± 6411.9)
	9	IRV	103.6 (± 3.7)	26317.2 (± 5540.3)
	10	BUCKLIN	102.6 (± 4.5)	25574.9 (± 5617.2)
SAT	11	mxc08	101.8 (± 3.9)	30144.6 (± 6091.5)
	12	picosat846	96.4 (± 3.7)	29688.2 (± 6551.8)
	13	rsat20	93.8 (± 4.8)	34573.7 (± 6666.6)
	14	minisat20SAT07	89.8 (± 3.2)	31467.8 (± 8382.5)
	15	minisat2.0	87.2 (± 2.4)	34332.8 (± 7641.0)

Table 6. Leader board for the Random category of problem instances.

Approach		Solver Name	Nr. Solved	Cumulative Time on Solved Instances (s)
Oracle	1	VBR	227.6 (± 1.4)	28960.0 (± 5745.6)
CBR	2	BAS	216.7 (± 3.2)	30463.4 (± 5284.6)
	3	TW-BV	211.8 (± 2.6)	25250.1 (± 4005.0)
	4	COOMBS	206.2 (± 3.1)	24303.7 (± 4554.8)
	5	IRV	206.1 (± 3.9)	25405.7 (± 5249.5)
	6	COPELAND	205.8 (± 3.1)	24590.6 (± 5769.9)
	7	DW-BV	205.6 (± 3.5)	24019.9 (± 4382.9)
	8	BV	205.4 (± 3.5)	23753.5 (± 4606.5)
	9	BUCKLIN	203.2 (± 3.9)	24111.5 (± 5774.5)
	10	KEMENY	194.0 (± 4.5)	27713.1 (± 4651.7)
SAT	11	march_dl2004	149.8 (± 6.3)	38318.5 (± 4934.1)
	12	gnoveltyplus	148.1 (± 6.0)	21884.0 (± 6398.8)
	13	SATenstein_T7	146.8 (± 6.4)	23124.2 (± 3713.5)
	14	ranov	146.0 (± 6.4)	19454.8 (± 4909.5)
	15	SATenstein_swgcp	142.7 (± 7.5)	16795.1 (± 5327.9)

domains. A feature of the domain of SAT, and constraint solving in general, is that experience is important. This paper demonstrates that CBR has a lot to offer the SAT community.

Acknowledgement

This work is partly supported by Science Foundation Ireland Grant 10/IN.1/I3032 and FP7 FET-Open Grant 284715.

References

1. Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T., eds.: Handbook of Satisfiability. Volume 185 of Frontiers in Artificial Intelligence and Applications. IOS Press (February 2009)
2. Gomes, C.P., Selman, B.: Algorithm Portfolios. Artificial Intelligence **126**(1-2) (2001) 43–62
3. O'Mahony, E., Hebrard, E., Holland, A., Nugent, C., O'Sullivan, B.: Using Case-based Reasoning in an Algorithm Portfolio for Constraint Solving. In: Proceedings of AICS. (2008)
4. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla: Portfolio-based Algorithm Selection for SAT. Journal of Artificial Intelligence Research (June 2008) 565–606
5. Pulina, L., Tacchella, A.: A Multi-engine Solver for Quantified Boolean Formulas. In: CP. (2007) 574–589
6. Davis, M., Logemann, G., Loveland, D.: A Machine Program for Theorem Proving. Communications of the ACM **5**(7) (July 1962) 394–397
7. Fürnkranz, J., Hüllermeier, E.: Pairwise Preference Learning and Ranking. In: Proceedings of 14th European Conference on Machine Learning (ECML 2003). (2003) 145–156
8. Dwork, C., Kumar, R., Naor, M., Sivakumar, D.: Rank Aggregation Methods for the Web. In: Proceedings of the 10th International Conference on World Wide Web. WWW '01, New York, NY, USA, ACM (2001) 613–622
9. Brinker, K., Fürnkranz, J., Hüllermeier, E.: A Unified Model for Multilabel Classification and Ranking. In: Proceedings of the 2006 European Conference on Artificial Intelligence (ECAI 2006), IOS Press (2006) 489–493
10. Brinker, K., Hüllermeier, E.: Case-based Multilabel Ranking. In: Proceedings of the 20th International Joint Conference on Artificial intelligence. IJCAI'07, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (2007) 702–707
11. Brinker, K., Hüllermeier, E.: Case-based Label Ranking. In: Proceedings of the 17th European Conference on Machine Learning (ECML 2006). (2006) 566–573
12. Gärtner, T., Vembu, S.: Label Ranking Algorithms: A Survey. In Johannes Fürnkranz, E.H., ed.: Preference Learning. Springer–Verlag (2010)
13. Pacuit, E.: Voting Methods. In Zalta, E.N., ed.: The Stanford Encyclopedia of Philosophy. Winter 2011 edn. (2011)