Available online at www.sciencedirect.com
ScienceDirect
Expert Systems with Applications 34 (2008) 1093–1101
Expert Systems with Applications
www.elsevier.com/locate/eswa

Mining the data from a hyperheuristic approach using associative classification

Fadi Thabtah [a,*], Peter Cowling [b]

[a] Department of Computing and Engineering, University of Huddersfield, Huddersfield, UK
[b] MOSAIC Research Centre, University of Bradford, Bradford, UK

Abstract

Associative classification is a promising classification approach that utilises association rule mining to construct accurate classification models. In this paper, we investigate the potential of associative classifiers as well as other traditional classifiers such as decision trees and rule inducers in solutions (data sets) produced by a general-purpose optimisation heuristic called the hyperheuristic for a personnel scheduling problem. The hyperheuristic requires us to decide which of several simpler search neighbourhoods to apply at each step while constructing a solutions. After experimenting 16 different solution generated by a hyperheuristic called Peckish using different classification approaches, the results indicated that associative classification approach is the most applicable approach to such kind of problems with reference to accuracy. Particularly, associative classification algorithms such as CBA, MCAR and MMAC were able to predict the selection of low-level heuristics from the data sets more accurately than C4.5, RIPPER and PART algorithms, respectively.

Keywords: Associative classification; Classification; Data mining; Hyperheuristic; Scheduling

1. Introduction

Heuristic and metaheuristic approaches have been applied widely in personnel-scheduling problems (Blum & Roli, 2003; Cowling, Kendall, & Han, 2002). A metaheuristic could be defined as a recursive process which directs a simpler local search method by using different concepts for exploring and exploiting the search space in order to achieve good enough solutions (Blum & Roli, 2003). There are several different metaheuristic strategies for solving scheduling and optimisation problems such as local search, tabu search, simulated annealing and variable neighbourhood search.

Hamiez and Hao (2001) have used a tabu search-based method to solve the sport league scheduling problem (SLSP). Their implementation of the enhanced tabu search algorithm was able to schedule a timetable for up to 40 teams and its performance in term of the CPU time was excellent if compared with previous algorithms such as that of (McAloon, TretKoff, & Wetzel, 1997) that had been used for solving the same problem. Aicklen and Dowsland (2000) have used Genetic algorithms to deal with a nurse rostering problem in major UK hospitals, and Hansen and Mladenovic (1997) have showed that variable neighbourhood search is an effective approach for solving optimisation problems in which it generates good or sometimes near-optimal solutions in a moderate time.

Cowling, Kendall, and Soubeiga (2000) and Cowling et al. (2002) argued that metaheuristic and heuristic approaches tend to be knowledge rich and require extensive experience in the problem domain and the selected heuristic techniques, and therefore they are expensive in term of their implementation. A new general framework to deal with large and complex optimisation and scheduling problems, called a hyperheuristic, has been proposed

* Corresponding author.
E-mail addresses: F.Thabtah@hud.ac.uk (F. Thabtah), P.I.Cowling@-bradford.ac.uk (P. Cowling).

0957-4174/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2006.12.018

by Cowling et al. (2000). It tends to robustly find good solutions for large and complex scheduling problems and has been proven to be effective in many experiments (Cowling & Chakhlevitch, 2003; Cowling et al., 2002).

A hyperheuristic approach can be described as a supervisor, which controls the choice of which local search neighbourhood to choose while constructing a solution/schedule. A local search neighbour, also known as a low-level heuristic, is a rule or a simple method that generally yields a small change in the schedule. Often these low-level heuristics are based on normal methods of constructing a schedule such as adding an event, deleting an event or swapping two events. Fig. 1 represents the general hyperheuristic framework that at each iteration, selects and applies the low-level heuristic that has the largest improvement on the objective function, i.e. LLH5 in the figure shown below. The arrows going from and to the hyperheuristyic in Fig. 1 represent the selected low-level heuristics improvement values on the objective function obtained after trying them by the hyperheuristic.

The training scheduling problem that we consider in this paper is a complex optimisation problem for a large financial service company (Cowling et al., 2002). It involves a number of events, trainers, and locations to be scheduled over a period of time. The task is to create a timetable of courses distributed over several locations in a specific period of time using a known number of trainers. A more detailed description of the problem is presented in the next section.

In this paper, our aim is to determine an applicable data mining technique to the problem of deciding which low-level heuristic to apply in a given situation, using information about heuristic performance derived earlier. In particular, we would like to answer questions like: which learning algorithm can derive knowledge that could direct the search in order to produce good solutions?

To achieve our goal, we compare three associative classification techniques CBA (Liu, Hsu, & Ma, 1998), MCAR (Thabtah, Cowling, & Peng, 2005), MMAC (Thabtah, Cowling, & Peng, 2004) and two popular traditional classification techniques (PART (Frank & Witten, 1998) and RIPPER (Cohen, 1995)) on data sets generated using a hybrid hyperhueristic called Peckish (Cowling & Chakhlevitch, 2003), for the trainer scheduling problem.

We analyse data from several solutions of the Peckish hyperheuristic that combines greedy (best first) and random approaches. We identify that the learning task involves classifying low-level heuristics in terms whether they improved the objective function in old solutions in order to produce useful rules. These rules then will be used to decide the class attribute "low-level heuristic" while constructing new solutions. We use the classification algorithms mentioned above to learn the rules.

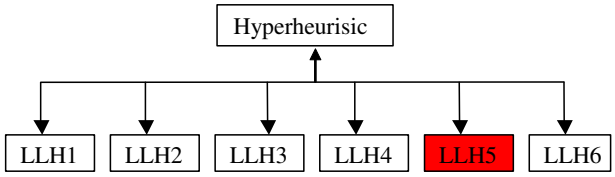

Fig. 1. Hypreheuristic general framework.

The training scheduling problem and different hyperheuristic approaches utilised to solve it are discussed in Section 2. Section 3 is devoted to the applicability of data mining classification algorithms to predict the behaviour of low-level heuristics used by the Peckish hyperheuristic. Data sets, their features and experimental results are presented in Section 4 and finally conclusions are given in Section 5.

2. The training scheduling problem and hyperheuristics

A much simpler version of the training scheduling problem has been solved in Cowling et al. (2002) using a Hyper-Genetic algorithm. A larger and more complex problem, which has been described in Cowling and Chakhlevitch (2003) is summarised in this section. It involves a number of events, trainers, and locations to be scheduled over a period of time. The task is to create a timetable of geographically distributed courses over a period of time using different trainers, and the aim is to maximise the total priority of courses and to minimise the amount of travel for each trainer. The problem is associated with a large number of constraints such as:

- Each event is to be scheduled at one location from the available number of locations.
- Each event must start within a specified time period.
- Each event can occur at most once.
- Each event to be delivered by competent trainers from the available trainers.
- Each location has a limited number of rooms and rooms have different capacities and capabilities.

The data used to build the solutions of the training scheduling problem is real data provided by a financial firm where training is given by 50 trainers over a period of 3 months in 16 different locations. Further, there are about 200 events to be scheduled and 95 different low-level heuristics that can be used to build each solution. However, solutions given to us by the authors of Cowling and Chakhlevitch (2003) have been constructed using only 10 low-level heuristics, where each low-level heuristic represents local search neighbourhoods. For example, selecting a location with the lowest possible travel penalty for a particular trainer to deliver a course as early as possible corresponds to a low-level heuristic.

In solving the trainer scheduling problem, three hyperheuristic approaches, random, greedy and hybrid, have been used. All of these approaches aim to manage the choice of which low-level heuristics to choose during the process of building the solution. The random approach

selects a low-level heuristic at random from the available ones in the problem. At each choice point in the search space, commonly all low-level heuristics have an equal chance to be selected. On the other hand, the greedy approach selects the low-level heuristic that yields the biggest improvement in the objective function. If none of the available ones improve the objective function then the algorithm will be trapped in a local optimum. The hybrid approach is named Peckish and consists of a combination of greedy and random approaches. It builds a solution by selecting a single low-level heuristic during each iteration in the search in order to apply. The choice is based on the low-level heuristic that has the largest improvement on the objective function in the problem (if one exists). In the case that none of the available low-level heuristics improve upon the objective function value, then the choice is random.

In this paper, we choose a low-level heuristic from a candidate list of good low-level heuristics. By changing the length of this candidate list and considering how it is merged, we can trade off the degree of greediness and randomness in the Peckish hyperheuristic. As a result, several different solutions produced by the Peckish hyperheuristic are investigated.

We analysed the strategy used by the Peckish hyperheuristic to construct a solution and observed that all available low-level heuristics in the problem must be tested in order to record their effect on the objective function at each iteration and apply only a single one. Data mining could provide a much quicker prediction of effective low-level heuristics at each iteration. In the next section, we investigate some of the popular data mining techniques for learning the sets of low-level heuristics that improve the objective function and have been applied by the Peckish hyperhueristic.

3. Data mining for the selection of low-level heuristics

Since we are aiming to use knowledge derived from old solutions of the problem, data mining seems an appropriate technique to extract that knowledge. The next task is to identify which data mining method is applicable to extract knowledge from solutions generated by the Peckish hyperheuristic. As mentioned earlier, the Peckish hyperheuristic usually selects and applies the low-level heuristic that leads to the largest improvement on the objective function and this is the class we want to find. In other words, we can learn rules that predict the performance of low-level heuristics in some solution runs and use these rules to forecast which low-level heuristics the hyperheuristic should choose in other runs. Since we are predicting a particular attribute (low-level heuristic), as a result, supervised learning approaches such as classification are appropriate.

There are many classification approaches for extracting knowledge from data that have been studied in the literature Cendrowska (1987), Quinlan (1993) and Cohen (1995). Three common approaches, divide-and-conquer (Quinlan, 1987), rule induction (Cohen, 1995; Furnkranz & Widmer, 1994) and associative classification (Li, Han, & Pei, 2001; Liu et al., 1998) have been selected for our base comparison. Further, five classification techniques related to such approaches have been compared, which are PART (Frank & Witten, 1998), RIPPER (Cohen, 1995), CBA (Liu et al., 1998), MCAR (Thabtah et al., 2005) and MMAC (Thabtah et al., 2004). Our choice of these methods is based on the different schemes they use in learning rules from data sets. In the next subsection, we briefly survey these algorithms.

3.1. Associative classification

Associative classification techniques employ association rule discovery methods to find the rules. This approach was introduced in 1997 by Ali, Manganaris, and Srikant (1997) to produce rules for describing relationships between attribute values and the class attribute and not for prediction, which is the ultimate goal for classification. In 1998, associative classification has been successfully employed to build classification models (classifiers) by Liu et al. (1998) and later attracted many researchers, e.g. (Yin & Han, 2003), from data mining and machine learning communities. In this subsection we survey associative classification techniques used in this paper to generate rules from the hyperheuristic data.

3.1.1. Classification based on association (CBA)

The idea of using association rule mining in classification problems was first introduced in Liu et al. (1998), in which an algorithm called CBA is proposed, which operates in three main steps. Firstly, if the intended data set contains any real or integer attributes, it is disctretised using multi-interval discretisation method of Fayyad and Irani (1993). Secondly, the Apriori candidate generation step (Agrawal & Srikant, 1994) is adopted to find the potential rules. Apriori candidate generation method necessitates multiple passes, where the potential rules found in the previous pass are used for the generation of potential rules in the current pass. This repetitive scans requires high CPU time and main memory. Once all potential rules are produced, the subset that leads to the least error rate against the training data set is selected to from the classifier. The selection of such subset is accomplished using the database coverage heuristic, which ensures that every rule in the classifier must cover correctly at least one training data object.

3.1.2. MCAR: multi-class classification based on association rule

A recently developed associative classification algorithm called MCAR (Thabtah et al., 2005) employs tid-list intersections to quickly find the rule. This algorithm consists of two main phases: rules generation and a classifier builder. In the first phase, the training data set is

scanned once to discover the potential rules of size one, and then MCAR intersects the potential rules tid-lists of size one to find potential rules of size two and so forth. This rules discovery method does no require passing over the training data multiple times. In the second phase, rules created are used to build a classifier by considering their effectiveness on the training data set. Potential rules that cover certain number of training objects will be kept in the final classifier. Finally, MCAR adds upon previous rule ranking approaches in associative classification, which are based on (confidence, support, rule length) by looking at the class distribution frequencies in the training data and prefers rules that are associated with dominant classes. Experimental results showed that MCAR rule ranking method reduces rule random selection during the process of ranking the rules especially for dense classification data.

3.1.3. Multi-class, multi-label associative classification (MMAC)

The MMAC algorithm consists of three steps: rules generation, recursive learning and classification. It passes over the training data in the first step to discover and generate a complete set of rules. Training instances that are associated with the produced rules are discarded. In the second step, MMAC proceeds to discover more rules that pass user predefined thresholds denoted by minimum-support and minimum-confidence from the remaining unclassified instances, until no further potential rules can be found. Finally, rule sets derived during each iteration are merged to form a global multi-label classifier that then is tested against test data. The distinguishing feature of MMAC is its ability of generating rules with multiple classes from data sets where each of their data objects is associated with just a single class. This provides decision makers with useful knowledge discarded by other current associative classification algorithms.

3.2. Traditional classification approaches

3.2.1. C4.5

C4.5 algorithm was created by Quinlan (1993) as a decision tree method for extracting rules from a data set. C4.5 is an extension of the ID3 algorithm (Quinlan, 1979), which accounts for missing values, continuous attributes and pruning of decision trees. As for the ID3 algorithm, C4.5 uses information gain to select the root attribute. The algorithm selects a root attribute from the ones available in the training data set. C4.5 makes the selection of the root based on the most informative attribute and the process of selecting an attribute is repeated recursively at the so-called child nodes of the root, excluding the attributes that have been chosen before, until the remaining training data objects can not be split any more (Quinlan, 1979). At that point, a decision tree is outputted where each node corresponds to an attribute and each arc to a possible value of that attribute. Each path from the root node to any give leaf in the tree corresponds to a rule. One of the major extensions of the ID3 algorithm that C4.5 proposed is that of pruning. Two known pruning methods used by C4.5 to simplify the decision trees constructed are sub-tree replacement and pessimistic error estimation (Witten & Frank, 2000).

3.2.2. Repeated incremental pruning to produce error reduction algorithm (RIPPER)

RIPPER is a rule induction algorithm that has been developed in 1995 by Cohen (1995). It builds the rules set as follows: The training data set is divided into two sets, a pruning set and a growing set. RIPPER constructs the classifier using these two sets by repeatedly inserting rules starting from an empty rule set. The rule-growing algorithm starts with an empty rule, and heuristically adds one condition at a time until the rule has no error rate on the growing set.

In fact, RIPPER is a refined version of an earlier developed algorithm called Incremental Reduced Error Pruning (IREP) (Furnkranz & Widmer, 1994) that adds a post pruning heuristic on the rules. This heuristic has been applied to the classifier produced by IREP as an optimisation phase, aiming to simplify the rule set features. For each rule r_i in the rule set, two alternative rules are built; the replacement of r_i and the revision of r_i. The replacement of r_i is created by growing an empty rule r'_i and then pruning it in order to reduce the error rate of the rules set including r'_i on the pruning data set. The revision of r_i is constructed similarly except that the revision rule is built heuristically by adding one condition at a time to the original r_i rather than to an empty rule. Then the three rules are examined on the pruning data to select the rule with the least error rate. The integration of IREP and the optimisation procedure forms the RIPPER algorithm.

3.2.3. PART

Unlike the C4.5 and RIPPER techniques that operate in two phases, the PART algorithm generates rules one at a time by avoiding extensive pruning (Frank & Witten, 1998). The C4.5 algorithm employs a divide-and-conquer approach, and the RIPPER algorithm uses rule induction approach to derive the rules. PART combines both approaches to find and generate rules. It adopts rule induction approach to generate a set of rules and uses divide-and-conquer to build partial decision trees. The way PART builds and prunes a partial decision tree is similar to that of C4.5, but PART avoids constructing a complete decision tree and builds partial decision trees. PART differs from RIPPER in the way rules are created, where in PART, each rule corresponds to the leaf with the largest coverage in the partial decision tree. On the other hand, RIPPER builds the rule in a greedy fashion, starting from an empty rule, it adds conditions, until the rule has no error rate and the process is repeated. Missing values and pruning techniques are treated in the same way as C4.5.

4. Data and experimental results

4.1. Data sets and their features

Data from 16 different solutions produced by Peckish hyperheuristic for the training scheduling problem were provided by the authors of Cowling and Chakhlevitch (2003). Each solution represents 500 iterations of applied low-level heuristics and is given in a text file. Twelve of the data files each represents only a single solution, whereas, each of the remaining four files represents ten combined solutions. Ten different low-level heuristics (LLH 1, LLH 2, LLH 20, LLH 27, LLH 37, LLH 43, LLH 47, LLH 58, LLH 68, LLH 74) have been used to produce each solution. Each file consists of 15 different attributes and 5000 instances. One iteration in a single solution is shown in Table 1 where the bold row indicates that low-level heuristic number 74 was applied by the Peckish hyperheuristic because it has the largest improvement on objective function.

In Table 1, column **LLH** represents the low-level tested, **ESM** and **RSM** columns stand for event and resource selection methods, respectively which specify how an event is scheduled. **SE** indicates the selected event number to be scheduled. **UE** reflects whether another event is a conflict with currently scheduled event, i.e. they share the same trainer, location, or timeslot. **EID** corresponds to the event identification number and **R** stands for rescheduled, which means, if swapping unscheduled event from the schedule with the selected event is possible, then, it is possible to reschedule back the removed event from the schedule.

OP, NP, OPE and **NPE** correspond to old priority, new priority, old penalty and new penalty, respectively. The new priority and new penalty values represent the total priority and penalty of the schedule after applying a low-level heuristic. The difference between the new priority and new penalty values gives the value of the objective function. **IMP** stands for amount of improvement and represents the change in value on the objective function after a low-level heuristic has been applied. **CPU** is the time in search for the low-level heuristic to be applied and SC reflects whether or not the current schedule has been changed, i.e. whether the current low-level heuristic had any effect at all, ('1' changed or '0' not changed). Finally AP column indicates whether or not the current low-level heuristic has been applied ('1' applied or '0' not applied) by the hyperheuristic.

After analysing the data in each file, we identified that six attributes have some correlation to the class attribute (LLH), which are (OP, NP, OPE, NPE, IMP, AP). However, we are interested to learn rules that represent useful sequence of applied low-level heuristics at different iterations and lead to improvement on the objective function. Therefore, solutions generated by the Peckish hyperheuristic have been filtered to retain certain iterations where improvements have occurred upon the objective function. Furthermore, a PL/SQL program has been designed and implemented to generate a new structure of each solution in order to enable the extraction of rules that represent the sequence of applied low-level heuristics. In other words, each training instance in the new structure should contain the low-level heuristics that improved the objective function in the current iteration along with others applied in the previous iterations. Specifically, in each solution run and for each iteration, we record low-level heuristics applied in the previous three iterations along with the ones that improved the objective function in the current iteration.

Table 2 represents part of a solution run (data) generated in the new structure after applying the PL/SQL program on the initial solution features, where columns **LLH_3, LLH_2** and **LLH_1** represent the low-level heuristics applied at the previous three iterations. Column **LLH** represents the current low-level heuristic that improved the objective function and column **Imp** represents the improvement on the objective function value. Finally column **Apply** represents whether or not the selected low-level heuristic has been applied by the hyperheuristic. As shown in Table 2, data generated by the hyperheuristic have multiple labels, since there could be more than one low-level heuristic that improve the objective function at any give iteration. Hence, each training instance in the scheduling data may associate with more than one class.

Table 1
One iteration of the Peckish hyperheuristic for the training scheduling problem

LLH	ESM	RSM	SE	UE	EID	R	OP	NP	OPE	NPE	IMP	CPU	SC	AP
1	0	1	1	0	−1	−1	72,999	72,999	830	830	0	0.03	0	0
2	0	2	128	0	−1	−1	72,999	72,999	830	830	0	0.02	0	0
20	4	0	12	0	−1	61	72,999	72,999	830	830	0	0.02	0	0
27	0	2	36	−1	−1	−1	72,999	72,999	830	830	0	0	0	0
37	1	2	70	−1	−1	58	72,999	72,999	830	830	0	0	0	0
43	1	8	142	−1	−1	−1	72,999	72,999	830	830	0	0	0	0
47	2	2	22	−1	−1	−1	72,999	72,999	830	830	0	0	0	0
58	3	3	18	−1	−1	−1	72,999	72,999	830	830	0	0	0	0
68	4	3	25	−1	−1	−1	72,999	72,999	830	830	0	0	0	0
74	**4**	**9**	**57**	**−1**	**−1**	**−1**	**72,999**	**73,099**	**830**	**830**	**100**	**0**	**1**	**1**

Table 2
Sample of optimisation data

Iteration	LLH 3	LLH 2	LLH 1	LLH	Imp	Apply
1	58	68	58	20	100	1
1	58	68	58	43	25	0
2	20	27	43	74	50	1
2	20	27	43	2	10	0
2	20	27	43	58	8	0
3	43	27	58	68	875	1
4	37	20	2	37	1055	1
4	37	20	2	2	950	0
4	37	20	2	74	69	0
4	37	20	2	58	9	0

4.2. Experimental results

In this section, we describe the experiments to evaluate classification accuracy and rules features produced by different rule learning algorithms from the optimisation data sets. We have performed a number of experiments using ten-fold cross validation on 16 different data files derived by the Peckish hyperheuristics for the trainer scheduling problem. The sizes of the data files after applying the PL/SQL program vary. There are 12 data files, each contains 182–750 training instances, where each file represents only one single run of the Peckish hyperheuristic. The remaining four data files contain 1500–2400 training instances and represent ten combined solutions. Two popular traditional classification algorithms (PART, C4.5) and three associative classification techniques (CBA, MCAR, MMAC) have been compared in terms of accuracy. The experiments of PART and C4.5 have been conducted using Weka software system (WEKA, 2000) and CBA experiments have been performed using a VC++ version provided by the authors of CBA (1998). Finally MCAR and MMAC algorithms were implemented in Java under Windows XP on 1.7 Ghz, 256 RAM machine.

The relative prediction accuracy, which corresponds to the difference of the classification accuracy of CBA, PART and C4.5 algorithms with respect to (MCAR, MMAC) are shown in Figs. 2–4, respectively. Fig. 2 for instance signifies the difference of the accuracy between CBA classifiers relative to those derived by (MCAR, MMAC) on the 16 optimisation data sets. The relative prediction accuracy figures shown are computed using the formulae $\frac{(\text{Accuracy}_{\text{MCAR}}-\text{Accuracy}_{\text{CBA}})}{\text{Accuracy}_{\text{CBA}}}$ and $\frac{(\text{Accuracy}_{\text{MMAC}}-\text{Accuracy}_{\text{CBA}})}{\text{Accuracy}_{\text{CBA}}}$ for CBA. The same sort of formulae has been used for PART and C4.5 algorithms, respectively. We used a *minsupp* of 2% and a *minconf* of 30% in the experiments of CBA, MCAR and MMAC algorithms. The label-weight evaluation measure (Thabtah et al., 2004) has been used to calculate the accuracy for MMAC algorithm in the figures.

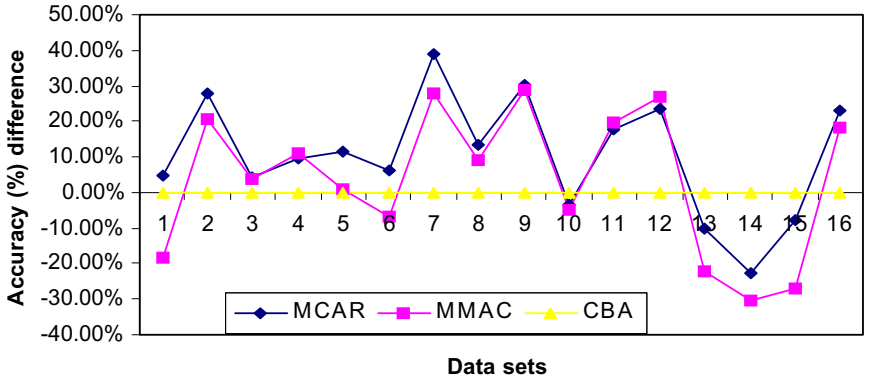

Fig. 2. Difference of accuracy between CBA and (MCAR, MMAC).

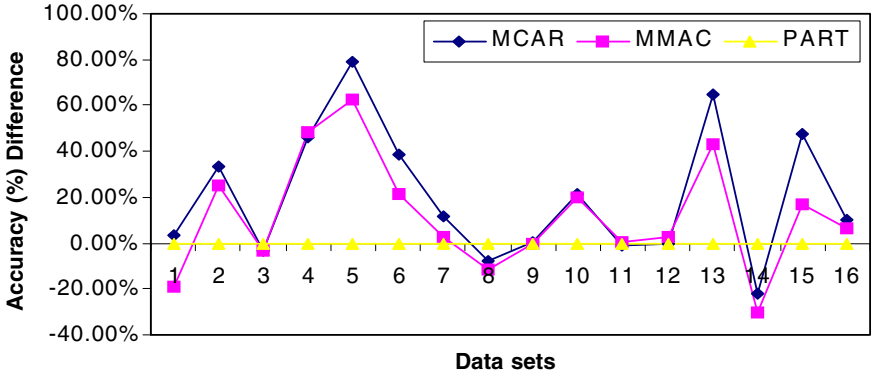

Fig. 3. Difference of accuracy between PART and (MCAR, MMAC).

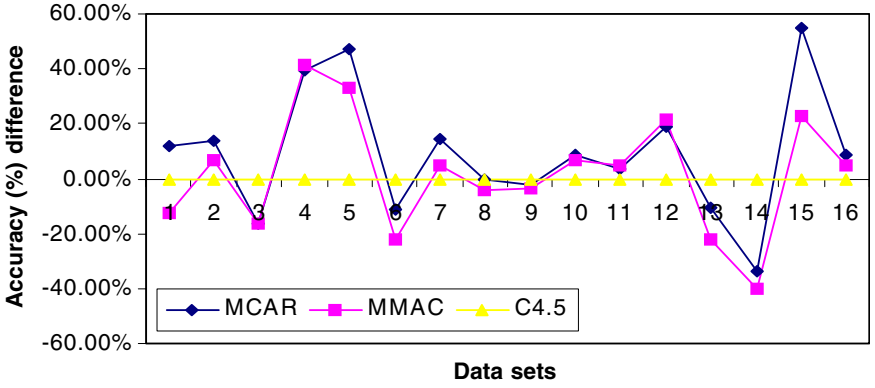

Fig. 4. Difference of accuracy between C4.5 and (MCAR, MMAC).

Label-weight evaluation method assigns each class in a multi-label rule a value based on its number of occurrences with rule's consequent (left-hand-side) in the training data. To clarify, a training object may belong to several classes where each one associated with it by a number of occurrences in the training data set. Each class can be assigned a weight according to how many times that class has been associated with the training object. Thus, unlike the error-rate method (Witten & Frank, 2000), which considers only one class for each rule in computing the correct predictions, label-weight gives a value for each possible class in a rule according to its frequency in the training data. This gives the top ranked class in a rule the highest weight and not all the weight as error-rate method does.

The accuracy results shown in the graphs indicated that associative classification algorithms outperformed the other learning techniques over the majority of test instances. Particularly, CBA, MCAR and MMAC outperformed the other learning algorithms on 5, 6, 7 and 5 benchmark problems, respectively. The won-loss-tied record of MCAR against CBA, C4.5 and PART are 11-5-0, 10-6-0 and 11-5-0, respectively. The MMAC won-loss-tied records against CBA, C4.5 and PART are, 10-6-0, 9-7-0 and 11-5-0, respectively. These figures show that associative classification approach is able to produce more accurate classifiers than decision trees and rule induction approaches, respectively.

It should be noted that CBA, PART and C4.5 algorithms outperformed (MCAR, MMAC) on data set number 14 in Figs. 2–4, respectively. After analysing the data in this particular set, it turned out that classes in this set are not evenly distributed. For example, class LLH2 and LLH20 were frequently applied by the hyperheuristic in this data set, whereas, classes LLH27, LLH37, LLH43 and LLH58 were rarely been used by the hyperheuristic. This is compound by the limited number of training instances in this particular data set (182 training data objects).

Analysis of the classifiers produced revealed consistency in the accuracy of both PART and C4.5 because the difference on average in the accuracy between them in all experiments is less than 1.6 %. This supports research works conducted in Frank and Witten (1998), where they show that despite the simplicity of PART, it generates rules as accurately as C4.5 and RIPPER. Also C4.5 and PART algorithms showed consistency in the number of rules produced.

Analysis of the rules features generated from the hyper-heuristic data has been carried out. Fig. 5 shows the number of rules extracted from nine data sets, categorised by the number of classes. MMAC is able to extract rules that are associated with up to four classes for this data. This is one of the principle reasons for improving the accuracy within applications. Fig. 5 also demonstrates that the majority of rules created from each solution are associated with one or two class labels. It turns out that this reflects accurately the nature of the hyperheuristic data, since during each iteration, normally only one or two low-level heuristics improve on the objective function in the scheduling problem. Thus, each training instance usually corresponds to just one or two classes.

The additional classes discovered by MMAC algorithm from the real data represent useful knowledge discarded by CBA, PART and RIPPER algorithms. The fact that

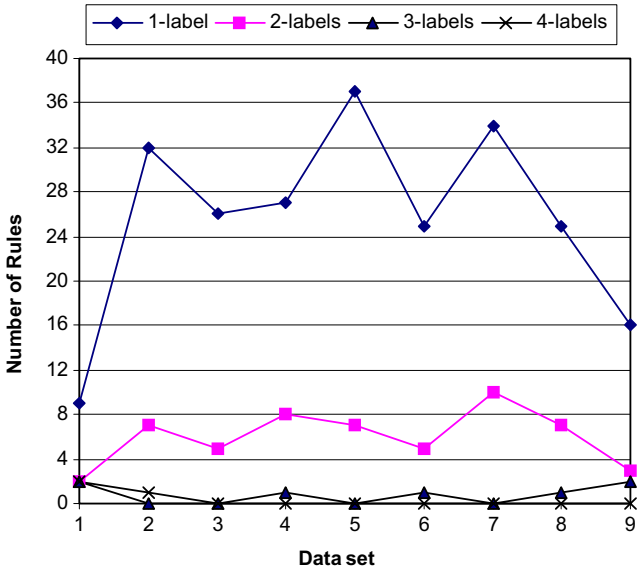

Fig. 5. Distribution of the rules with regards to their labels.

MMAC is able to extract rules with multiple classes enables domain experts to benefit from this additional useful information. In addition, these multi-label rules can contribute in the prediction step, possibly improving upon the classification accuracy.

The numbers of rules produced by C4.5, PART, CBA and MCAR algorithms are listed in Table 3. Since MMAC produces rules with multiple classes, we did not record their numbers of rules generated for fair comparison. The values in Table 3 show that C4.5 always generates more rules than PART, CBA and MCAR. This contradicts some earlier results reported in Liu et al. (1998) and Thabtah et al. (2005) on classifier sizes obtained against UCI data collection (Merz & Murphy, 1996), which show that associative classification approaches like CBA and MCAR normally generate more rules than decision trees. For this reason, we performed extensive analysis on the classifiers derived by C4.5 from the optimisation data sets.

After analysing the decision trees constructed by C4.5 algorithm at each iteration, we observed that many rules are generated, which do not cover any training data. We found out that the reason for these many useless rules appear to be the attribute that C4.5 splits the training instances on, if that attribute has many distinct values and only few of these values appear in the training data, then a rule for each branch will be generated, and hence only some of these branches cover training instances. The rest will represent rules that cover not even a single training instance. In other words, when the training data set consists of attributes that have several distinct values and a split occurs, the expected number of rules to be derived by C4.5 can be large. Since data sets used to derive the results in Table 3 contain four attributes where each one of them has 10 different values (low-level heuristics) and some of these low-level heuristics are never result in solution improvement for the hyperheuristic, this explains the large numbers of rules derived by C4.5.

Table 3
Number of rules of C4.5, PART, CBA and MCAR on the optimisation data sets

Data	C4.5	PART	CBA	MCAR
1	11	3	3	14
2	55	16	18	32
3	46	19	12	28
4	82	17	11	33
5	46	17	10	31
6	19	16	23	31
7	91	69	1	20
8	100	51	3	9
9	163	71	2	12
10	163	145	1	21
11	46	21	10	22
12	64	19	22	37
13	46	19	6	33
14	64	24	23	42
15	55	23	16	31
16	64	56	5	18

5. Conclusions

In this paper, we have studied data sets produced from a complex personnel scheduling problem, called the training scheduling problem. These data sets represent solutions generated by a general hybrid approach, called the Peckish hyperheuristic, which is a robust and general-purpose optimisation heuristic that requires us to decide which of several simpler low-level heuristic techniques to apply at each step while building the schedule. Our study focused on analysing the behaviour of low-level heuristics that were selected by the hyperheuristic and improved upon the quality of the current solution in order to extract useful rules. These rules can be used later to quickly predict the appropriate low-level heuristics to call next. For this purpose, we have compared five data mining classification algorithms, (PART, C4.5, CBA, MCAR, MMAC) on 16 different solutions produced by Peckish hyperheuristic.

The experimental tests showed a better performance for associative classification techniques (MCAR, MMAC, CBA) over decision trees (C4.5), rule induction (RIPPER) and PART algorithm with reference to the accuracy of predicting the appropriate set of low-level heuristics. Since the MMAC algorithm was able to produce rules with multiple classes, including very useful information that the hyperheuristic can use in forecasting the behaviour of low-level heuristic while constructing a new solution. It is the most applicable data mining approach, which can be used to predict low-level heuristic performance within the Peckish hyperheuristic.

Furthermore, C4.5 generated more rules than the other rule learning algorithms since useless rules were extracted by the C4.5 algorithm, which have not even a single representation in the training data. The reason of these useless rules turn out to be the training data attributes, in which when some of these attributes are associated with many distinct values and only a subset of these values have sufficient representation in the training data, there will be valid rules for this subset and the rest will represent useless rules.

References

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rule. In *Proceedings of the 20th international conference on very large data bases* (pp. 487–499).

Aicklen, U., & Dowsland, K. (2000). Exploiting problem structure in a genetic approach to a nurse rostering problem. *Journal of Scheduling, 3*, 139–153.

Ali, K., Manganaris, S., & Srikant, R. (1997). Partial classification using association rules. In D. Heckerman, H. Mannila, D. Pregibon, & R. Uthurusamy (Eds.), *Proceedings of the third international conference on knowledge discovery and data mining* (pp. 115–118).

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimisation: overview and conceptual comparison. *ACM Computing Surveys (CSUR)* (pp. 268–308).

CBA (1998). <http://www.comp.nus.edu.sg/~dm2/p_download.html>.

Cendrowska, J. (1987). PRISM: an algorithm for inducing modular rules. *International Journal of Man-Machine Studies, 27*(4), 349–370.

Cohen, W. (1995). Fast effective rule induction. In *Proceedings of the 12th international conference on machine learning* (pp. 115–123). CA: Morgan Kaufmann.

Cowling, P., & Chakhlevitch, K. (2003). Hyperheuristics for managing a large collection of low level heuristics to schedule personnel. In *Proceedings of IEEE conference on evolutionary computation* (pp. 1214–1221). Canberra, Australia.

Cowling, P., Kendall, G., & Han, L. (2002) An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In *Proceedings of congress on evolutionary computation (CEC 2002)* (pp. 1185–1190). Hilton Hawaiian Village Hotel, Honolulu, Hawaii, May 12–17, ISBN 0-7803-7282-4.

Cowling, P., Kendall, G., & Soubeiga, E. (2000). A hyperheuristic approach to scheduling a sales summit. In *Proceedings of the third international conference of practice and theory of automated timetabling (PATAT 2000)*. *LNCS* (vol. 2079, pp. 176–190). Springer.

Fayyad, U., & Irani, K. (1993). Multi—interval discretisation of continues-valued attributes for classification learning. In *Proceedings ofI JCAI* (pp. 1022–1027).

Frank, E., & Witten, I. (1998). Generating accurate rule sets without global optimisation. In *Proceedings of the fifteenth international conference on machine learning* (pp. 144–151). Madison, Wisconsin: Morgan Kaufmann.

Furnkranz, J., & Widmer, G. (1994). Incremental reduced error pruning. In *Machine learning: Proceedings of the 11th annual conference*. New Brunswick, New Jersey: Morgan Kaufmann.

Hamiez, J., & Hao, J. (2001). Solving the sports league scheduling problem with Tabu search. *Lecture notes in artificial intelligence* (vol. 2148, pp. 24–36). Springer.

Hansen, P., & Mladenovic, N. (1997). Variable neighbourhood search. *Computer and Operations Research*, 1097–1100.

Li, W., Han, J., & Pei, J. (2001). CMAR: accurate and efficient classification based on multiple-class association rule. In *Proceedings of the ICDM'01* (pp. 369–376). CA: San Jose.

Liu, B., Hsu, W., & Ma, Y. (1998). Integrating classification and association rule mining. In *Proceedings of the KDD* (pp. 80–86). New York, NY.

McAloon, K., TretKoff, C., & Wetzel, G., 1997. Sports league scheduling. In *Proceedings of the third ILOG optimisation suite international users conference*. Paris, France, 1997.

Merz, C., & Murphy, P. (1996). *UCI repository of machine learning databases*. Irvine, CA: University of California, Department of Information and Computer Science.

Quinlan, J. (1979). Discovering rules from large collections of examples: a case study. In D. Michie (Ed.), *Expert systems in the micro-electronic age* (pp. 168–201). Edinburgh: Edinburgh University Press.

Quinlan, J. (1987). Generating production rules from decision trees. In *Proceedings of the 10th international joint conferences on artificial intelligence* (pp. 304–307). CA: Morgan Kaufmann.

Quinlan, J. (1993). *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufman.

Thabtah, F., Cowling, P., & Peng, Y. (2004). MMAC: a new multi-class, multi-label associative classification approach. In *Proceedings of the fourth IEEE international conference on data mining (ICDM '04)* (pp. 217–224) Brighton, UK. (Nominated for the Best paper award).

Thabtah, F., Cowling, P., & Peng, Y. (2005). MCAR: multi-class classification based on association rule approach. In *Proceeding of the 3rd IEEE international conference on computer systems and applications* (pp. 1–7). Cairo, Egypt.

WEKA (2000): Data Mining Software in Java: <http://www.cs.waikato.ac.nz/ml/weka>.

Witten, I., & Frank, E. (2000). Data mining: practical machine learning tools and techniques association rules. In *Proceedings of the 3rd KDD conference* (pp. 283–286).

Yin, X., & Han, J. (2003) CPAR: classification based on predictive association rule. In *Proceedings of the SDM* (pp. 369–376). San Francisco, CA.