Hyper-heuristic Decision Tree Induction

Alan Vella, David Corne
School of MACS, Heriot-Watt University
Edinburgh, UK
mail@alanvella.com, dwcorne@gmail.com

Chris Murphy
Motorola Ltd,
Swindon, UK
Chris.Murphy@motorola.com

Abstract—**Hyper-heuristics are increasingly used in function and combinatorial optimization. Rather than attempt to solve a problem using a fixed heuristic, a hyper-heuristic approach attempts to find a combination of heuristics that solve a problem (and in turn may be directly suitable for a class of problem instances). Hyper-heuristics have been little explored in data mining. Here we apply a hyper-heuristic approach to data mining, by searching a space of decision tree induction algorithms. The result of hyper-heuristic search in this case is a new decision tree induction algorithm. We show that hyper-heuristic search over a space of decision tree induction rules is able to find decision tree induction algorithms that outperform many different version of ID3 on unseen test sets.**

Keywords- data mining, hyper-heuristics, decision trees, evolutionary algorithm.

I. Introduction

Hyper-heuristics [1] are increasingly used in function and combinatorial optimization. The essential idea of hyper-heuristics is to search for an *algorithm* rather than for a specific solution to a given problem. From the viewpoint of evolutionary computation, a hyper-heuristic can be simply regarded as a sophisticated encoding. The genotype represents an algorithm, and when we interpret it, by running the algorithm on the given problem data, the result is a solution to a given problem instance. Hence we obtain a candidate solution via a "genotype→algorithm->candidate-solution" route, rather via a direct "genotype->candidate-solution" mapping. The interesting aspect of hyper-heuristics is the potential re-use of the algorithms that emerge from the search process. With appropriate experimental design (e.g., by using many problem instances in the initial hyper-heuristic training), new, effective and fast algorithms may be discovered that apply to a wide class of problem instances.

The origin of this notion can be traced to Fisher and Thompson's work [2], which investigated combinations of basic rules for job-shop scheduling. Other work pursued similar ideas, essentially re-discovering or extending [2] during the 1990s. Most of this continued to be in the area of job-shop scheduling. E.g. Fang et al [3] used evolutionary algorithms to evolve sequences of heuristics for job-shop and open-shop problems, while Zhang and Diettrich [4, 5] developed novel job-shop scheduling heuristics within a reinforcement learning framework. Another notable study was that of Gratch et al. [6], which used hill-climbing in a space of control strategies to find good algorithms for controlling satellite communication schedules.

Hyper-heuristics have now been applied to a variety of problems, however these are almost exclusively in the area of combinatorial optimization, and therein the majority involve scheduling. A few examples outside scheduling include bin packing [7] and cutting-stock [8]. In bin-packing, for example, novel constructive algorithms were developed that outperformed standard bin-packing constructive heuristics over a wide range of unseen test instances.

Very little work has so far explored the use of hyper-heuristics in data mining. Here, the task is invariably to find a *classifier* (which might be a decision tree, a set of rules, a neural network, etc…), which has good performance in classifying test data. In other words, this is a search in classifier space for a good classifier. To align this with the possibility of using hyper-heuristics, we can consider this as a search through the space of methods that build classifiers from training data. To date, one group has started to explore this idea. In Pappa and Freitas [9], grammar-based genetic programming is used to evolve rule induction algorithms, having presented the original idea in [10]. A broad category of rule induction algorithms operates via "sequential covering": an initial rule is generated, covering some of the dataset, and additional rules are generated in order until the entire dataset is covered. There are several alternative ways to generate the initial and subsequent rules. E.g. we may start with a very general high-coverage (but low accuracy) rule, and add conditions until accuracy and/or coverage move beyond a threshold. Or, we may start with a very precise rule and gradually remove conditions. In [9], the encoding covers a vast space of possible ways to organize this process.

Here we explore a hyper-heuristic approach to decision tree induction, by searching a space of decision tree induction algorithms. We use a simpler encoding than [9], essentially restricting the algorithm space to a single overall control structure. However we make more heuristic 'components' available, and hence explore a wider range of variants of a specialized class of algorithms.

Simply put, the classic decision tree induction algorithm builds a tree step by step by deciding, at each step, how to develop the next node in the tree. This comes down to choosing a specific attribute in the dataset. E.g., if the chosen attribute is "gender", then the current node will have two children, one for the case "gender = male" and another for "gender = female". The choice of attribute is made by using a heuristic, which tests how well each proposed attribute is at discriminating between values of the target class. Our method is to search a space of rulesets, where individual rules essentially say "if the data still to be classified have property X, then use heuristic Y to decide which attribute to split on". A simple approach is used for conflict resolution between rules in a ruleset. An evolutionary algorithm evolves the rulesets, where fitness is evaluated by using it to build a decision tree on a collection of training datasets, and finding the accuracy of these trees on test datasets.

The remainder is set out as follows. Section II describes the algorithms, providing a simple introduction to ID3-style decision tree induction, describing our hyper-heuristic encoding, and then indicating the details of the evolutionary algorithm used in conjunction with this encoding. Section III describes the datasets and experiments and provides some analysis and observations of the results. We provide a concluding discussion in Section IV.

II. Hyper-Heuristic Decision Tree Induction

A. *Decision Trees*

Decision tree (DT) building algorithms build a DT quickly using a recursive, divide-and-conquer strategy. While building the tree, a typical algorithm employs a heuristic to choose which attribute to use for creating the current tree node. Imagine a dataset held by a loan company with the attributes "gender", "salary", "age" and "high-risk", where the values for high-risk are either "yes" or "no" based on past experience. A decision tree built on this data may look like that in Fig. 1.

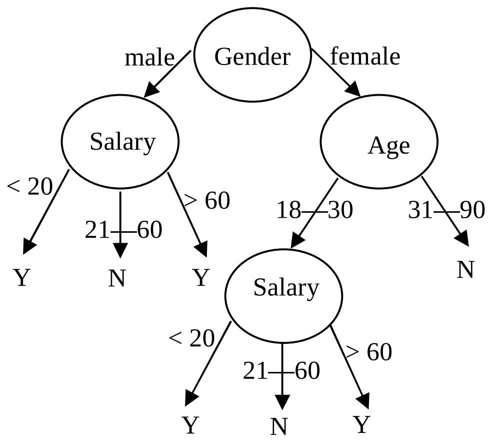

Figure 1. A simple example of a decision tree.

The contrived tree in Fig. 1 attempts to predict the value of the "high-risk" attribute (in this case, this is the target class) given the values of the other attributes. Each node uses a single attribute to "split" the data among its child nodes. E.g., the root node splits on gender; if an instance is "male", the next question comes from a node that splits on salary, and so on.

When we start to build a tree, the first decision to make is the choice of attribute for the root node. The key to this choice is to examine, for each attribute, how well it divides the data in terms of the target class. For example, if we found that all males were high risk and all females were low risk in the training dataset, then "gender" is a perfect attribute to split on. If instead we found that male and female instances contained equal proportions of high and low risk cases, then we seem to gain nothing by splitting on gender. In general, a heuristic is used to provide a score for each attribute, where that score reflects the value of using that attribute for the split. One of the most celebrated of such heuristics is information gain [11], used in the ID3 [12] and ASSISTANT [13] decision tree building algorithms.

Notice that, once we have chosen the attribute for the root node, we create a child node for each value of that attribute. In the example in Fig. 1, all of the "male" data instances are carried to the left hand child, and all of the "female" instances are carried to the right-hand child. For each of these nodes, we now have the same decision to make, and will again use a heuristic to decide which attribute to split on. However, the difference is that each node "carries" a specific set of instances, and the heuristic scores will therefore depend on the position of the node in the tree.

B. *Attribute Choice Heuristics*

Straightforward decision tree algorithms therefore repeatedly decide which attribute to split on for the current node, using a heuristic to make the decision. Although information gain is the most popular, there are several alternative possibilities. Twelve such are listed here: Chi-Square [14], Information Gain [11], Gain Ratio [15], Symmetric Gain Ratio [16], Gini index [17], Modified Gini index [18], Symmetric Gini index [19], J-Measure [20], Minimum Description Length [21], Relevance [22], RELIEF [23], Weight of Evidence [24].

Each of the above can be (and have been) used to estimate the quality of a distribution, and as such provides an alternative measure for assessing the way that the target class values of the 'remaining' instances in the dataset are distributed among the values of any given attribute available for splitting. The research literature contains various studies comparing the performance of two or more of the above heuristics on one or more data sets, e.g. [25—27].

C. *The Hyper-Heuristic Decision Tree Encoding*

We encode a decision tree building algorithm as a set of rules that indicate which heuristic to use in choosing the attribute for the current node. A single ruleset will typically therefore use multiple heuristics in each tree-building run. The intuition behind this is simply that alternative approaches may be better for different stages of tree building. E.g. the appropriate heuristics for the root node when building a many-attribute dataset may be quite different from the heuristics that will work better deep in the tree when few attributes are available for splitting. As a DT building algorithm works its way down the tree, the dataset partition which needs to be split gets smaller and smaller. Also, as indicated, the set of available candidate splitting attributes changes as fewer attributes remain. Since the 'problem state' of our DT building algorithm is continuously changing as the tree is being built, we see no reason why the heuristic that chooses the best splitting attribute has to remain static throughout the whole tree-building process. Indeed, we might get better trees if we adapt the heuristic to be used according to the data set partition that needs to be split. In such a scenario, our HH rules would be applied to any possible data set partition (instead of just the initial complete data set). Our DT building algorithm would employ a toolbox of heuristics and our HH would then pick and choose the best heuristic according to the features of the partition that needs to be split.

D. Details of the HHDT Encoding

Our HHDT encoding relies on calculating the entropy of each splitting attribute, as follows:

$$H(A) = \sum_{i=1}^{m} P(a_i) \cdot \log(P(a_i))$$

… where A is an attribute with m possible values: $a_1, \ldots, a_m$ and $P(a_i)$ is the probability of A having value a_i. $P(a_i)$ is simply the proportion of instances in the current partition of the dataset (i.e. those instances that are available at the current tree node) that have value a_i for attribute A.

The HHDT encoding simply comprises a set of rules, each of which examines (in a simple way) the entropy values among the attributes, and decides to use a specific heuristic. A single rule has the following format:

```
IF
    (x% > high)   AND   (y% < low)
THEN
    use heuristic h
```

… where: x and y are both percentage values ranging from 0 to 100, and high and low are thresholds for the entropy value H. The meaning of the above example rule is: if x% of the attributes have entropy values above *high*, and if y% of attributes have entropy values below *low*, then use heuristic h to choose the splitting attribute at the current node.

A single HHDT chromosome comprises a small set of such rules, and a *default* heuristic, representing a complete algorithm for building a decision tree, on any dataset, as follows. When a choice needs to made to split an attribute (this happens whenever the instances available at the node carry more than the value of the target attribute), the condition of each rule is tested, and consequently either 0, 1, or >1 heuristics are put forward for use. If 0 rules are triggered, the default heuristic is used. If 1 rule is triggered, then the corresponding heuristic is used. If >1 distinct heuristics are triggered, and no individual heuristic has the majority vote, then the default heuristic is used.

Note that we do not use pruning methods [28, 29] in the current research. Although this limits the generalization quality of the trees, this also simplifies the comparison studies. We expect incorporate pruning in later work; notice that it will be trivial to incorporate pruning into the interpretation of an HHDT chromosome.

E. Details of the Evolutionary Algorithms

We use a straightforward evolutionary algorithm (EA) to search the space of HHDT rulesets. An individual chromosome, representing a set of n rules, is simply a list of n 5-tuples, plus a single additional gene representing the default heuristic choice. A single 5-tuple is of the form:

$$(x, high, y, low, h)$$

with the obvious interpretation (see example rule in subsection II.*D*). The values of x and y are integers ranging from 0 to 100, and the values of *high* and *low* are real numbers in the interval [0, 1]. The heuristic gene h is simply an integer indexing the heuristics available for use in the experiment. Generally this could be the set of 12 heuristics in section II.*C*, or indeed any others. However we have found it best to focus on the heuristics that generally have the best individual performance while also keeping the heuristics in our toolbox as varied as possible, and hence in our experiments we limit the choice of heuristics to: information gain, gain ratio, symmetric gain, relief, and J-measure.

The EA uses a population of size 45—50 tournament selection with tournament size 0.4×*pop*, and generation gap replacement in which 75% of the population are replaced in each generation, maintaining the best 25% of the previous generation. This is a very 'high-pressure' EA, deliberately so to promote fast progress, since individual fitness evaluations (see section III.*B*) are quite time-consuming. One point crossover is used, with crossover preserving complete rules (i.e. the crossover point falls between 5-tuples, and not within them). Mutation is applied to each child: every gene is changed with a probability *m*. In all cases, a mutated value is a uniformly random new value from the appropriate range.

III. Datasets and Experiments

A. Datasets

We use 12 well known datasets with diverse characteristics in terms of their sizes, and their numbers and types of attributes. All are available from the UCI Machine learning repository [30]. Table I lists the datasets we use, and summarizes some key characteristics. They contain a mixture of categorical and numeric attributes. When an attribute is categorical (i.e. its values are from a small set of discrete possibilities), a node of a decision tree, which splits on that attribute, simply has one branch for each value (see the 'Gender' node in Fig. 1). In the case of a numerical attribute, we use a discretization of the attribute values that partitions the values into a small set of intervals. E.g. in Fig. 1, 'Salary' has been discretized into three intervals. There are many ways to discretize [31—33]. In prior experiments we have found that, from the viewpoint of decision tree quality, *equal-frequency-binning* [31] with five bins performs as well as most other methods. All of the numerical attributes in the datasets used in these experiments are pre-processed into five discrete categories via equal-frequency-binning. This effectively means that the smallest 20% of values of a numerical attribute are in bin 1, the next largest 20% are in bin 2, and so on. These bins might be labeled from "very small" to "very large" (for example), if we were to build an easily understandable decision tree.

TABLE I. DATASETS USED IN OUR EXPERIMENTS

Dataset Title	**Attribute distribution**	**No. instances**	**%age with target class**
Car	7 categoric, 0 numeric	1728	70%
Spect	23 categoric, 0 numeric	267	79%
votes84	17 categoric, 0 numeric	435	45%
Derma	34 categoric, 1 numeric	366	17%
Flags	25 categoric, 4 numeric	194	31%
Contrac	8 categoric, 2 numeric	1,473	43%
Credit	10 categoric, 6 numeric	690	56%
Heart	8 categoric, 6 numeric	270	44%
Ionosphere	0 categoric, 35 numeric	351	36%
Wine	0 categoric, 14 numeric	178	39%
Ecoli	0 categoric, 8 numeric	336	43%
Yeast	0 categoric, 9 numeric	1,484	31%

B. Experiments

When we apply HHDT to a dataset, the basic approach to evaluating the fitness of any HHDT chromosome is to use the ruleset specified by the chromosome to build a tree using the *training data*, and then test the accuracy of the tree on the test data. In our experiments, in all cases, we use 9-fold cross validation *within the fitness function*, thus providing a relatively robust evaluation of performance on test data during training. Essentially, this means that the above process is repeated 9 times within the fitness function, for different 'folds' of training and test sets. To estimate the quality of the best HHDT found during training, we use it to build a tree over the entire training set (i.e. all nine folds), and record the accuracy of that tree on an entirely unseen dataset that played no part in the training process – this is, in essence, the missing "10th fold". Experimental results referred to later are always results obtained on such unseen test sets. Finally, having learned from preliminary investigations (for which there is no space to summarize here), we use multiple data sets within a single training run; that is, the fitness of a single HHDT chromosome is an average over *N* datasets, each subject to 9-fold cross validation. In this paper we always have *N*=4, and the complete picture of the evaluation process for a single HHDT chromosome is as follows:

```
For each dataset:
    For each fold of the dataset:
        Build a tree on the training data.
        Evaluate the tree on the test data.
```

The fitness of the HHDT ruleset during training is taken as the mean of the 36 (9 folds, 4 datasets) test set accuracies. This setup aims to discourage overfitting, and we have found it leads to better results on the individual datasets than if they had been used as the sole dataset during training. Finally, as indicated, the ultimate result of a single HHDT run is obtained as follows: a single HHDT is returned from the training process – the one with the best training fitness (breaking ties randomly). This HHDT is then used to build a tree on the full training dataset(s), and the accuracy of these trees is recorded on entirely unseen test sets.

Here we report on two sets of experiments using two different evolutionary algorithm and HHDT setups. In each set of experiments, there are three sets of trials. In each trial, a distinct group of four datasets from Table I is used. An evolutionary algorithm is used to evolve HHDT rulesets, and the entire trial is repeated 10 times.

The fine details of the EAs and the HHDT encodings are detailed next. These are relatively arbitrary design choices, representing just two samples among the vast range of potential setups. As such they represent preliminary investigation of the feasibility of the overall approach.

In the first set of trials, the EA terminates after 100 generations, has population size *pop*=45, uses 1 point crossover, and has a mutation rate of *m*=5%. The HH encoding uses rulesets with a fixed size of 3 rules, with coarse-valued threshold genes and fine-valued percentage genes (all integers 0,…100 are available). Each trial used a set of four datasets that varied in their characteristics, ensuring that there was a mixture of datasets with all categorical, all numerical, and mixed attributes in each trial.

In the second set of trials, the EA terminates after 150 generations, has population size *pop*=50 and uses 2 point crossover. Rulesets were allowed to vary in size from 3 to 12, and hence there was an additional mutation operation, applied to every child, which either deleted a randomly chosen rule from a ruleset (if it contained 4 or more rules) or added a randomly generated rule (if it contained less than 12). Each child has a probability of 5% of having a rule added or deleted. The standard mutation operator in this case was set at m = 5%. Meanwhile, the percentage genes (x and y) were coarse-grained, allowed to take only 21 possible values (0, 5, …, 95, 100), and the threshold genes were allowed to take on any of 20 possible values: {0.05, 0.1, …, 0.95, 1}. Other details of this EA were the same as before.

C. Results

Table II summarizes the results of the first set of experiments. For example, the value of 0.779 in the HHDT row and the {yeast, votes, heart, flags} column indicates that the HHDT mean performance on unseen test sets, averaged over 10 independent trials, was 0.779 (i.e. 77.9% accuracy, to 3 significant figures) when the HHDT evolution process used those four datasets. The value of 0.770 for Information Gain in the same column indicates that (using the same training/test partitions, and unseen test sets) the equivalent value for ID3 employing the Information Gain heuristic is 77.0% accuracy. The 'overall' column gives the mean accuracy over the three sets of trials. Ranks were applied for each set of trials, ranging from 1 (best performance) to 13th (worst), and the rightmost column gives the mean rank over all trials. The methods are arranged in order of mean rank, and the best result in each column is in **bold** (where there seem to be ties, these are broken to more significant digits). We do not have space here to list the full sets of results for each of the ten independent runs, but we can report that a *T*-test comparing the HHDT ranks with those of Relevance and MDL yields a p value of < 0.05 indicating >95% confidence in the assertion that the evolved HHDT trees deliver better unseen test set performance than the ID3(h) trees for any of the other heuristics h.

HHDTs appear to have the best overall performance (the most appropriate indicator is mean rank, which is non-parametric), suggesting that the hyperheuristic training process is capable of leading to specific combinations of heuristics whose behavior generalizes well.

TABLE II. SUMMARIZED RESULTS OF FIRST GROUP OF EXPERIMENTS

Method	yeast votes heart flags	Car ecoli derma wine	contrac credit iono spect	over-all	Mean rank
HHDT	**0.779**	0.906	0.771	**0.818**	**3.33**
Relevance	0.773	**0.914**	0.765	0.818	4.33
M.D.L	0.776	0.898	**0.774**	0.816	4.33
Chi-Squ.	0.774	0.903	0.768	0.815	6
Gain Ratio	0.772	0.904	0.769	0.815	6.33
Inf. Gain	0.77	0.913	0.76	0.814	7
Gini Index	0.76	0.912	0.763	0.817	7
Sym. Gini	0.755	0.905	0.769	0.81	7.33
Sym. Gain	0.756	0.887	0.771	0.805	7.67
Wt. Evid.	0.78	0.852	0.763	0.798	7.67
J-Measure	0.772	0.891	0.764	0.809	8
Mod. Gini	0.757	0.906	0.749	0.804	9
RELIEF	0.749	0.815	0.725	0.763	13

Table III summarizes the results of the second set of experiments. The major differences in this case are that the rulesets were allowed to vary in size, and all 12 heuristics were available for use in the rulesets.

HHDT is joint second along with ID3 using the Relevance heuristic [22]. In terms of statistical significance, we note that the difference in rank between Gain Ratio and HHDT is significant with a confidence of >90%, and the superiority of HHDT over MDL is significant with confidence >95%. HHDT performs well in this set of trials, but achieves a relatively low rank in the {contrac, credit, iono, spect} trial, which dampens its overall performance. We suspect the availability of all 12 heuristics and the larger allowed sizes for the rulesets contributed to a greater degree of over-fitting in this experiments.

TABLE III. SUMMARIZED RESULTS OF SECOND GROUP OF EXPERIMENTS

Method	yeast votes heart flags	car ecoli derma wine	contrac credit iono spect	over-all	Mean rank
Gain Ratio	0.778	0.912	0.771	**0.820**	**2.33**
Relevance	0.770	0.909	**0.771**	0.817	3.67
HHDT	**0.780**	**0.915**	0.756	0.817	3.67
M.D.L.	0.778	0.900	0.766	0.815	5
Gini Index	0.763	0.907	0.764	0.811	7
Inf. Gain	0.767	0.911	0.754	0.811	7
Chi-Sq.	0.765	0.911	0.751	0.809	7.33
Wt Evid.	0.774	0.857	0.760	0.797	7.67
Sym. Gain	0.761	0.896	0.771	0.809	8
Sym. Gini	0.761	0.907	0.759	0.809	8.67
Mod. Gini	0.767	0.901	0.743	0.804	8.67
J-Measure	0.762	0.895	0.761	0.806	9
RELIEF	0.740	0.819	0.724	0.761	13

IV. CONCLUDING DISCUSSION

We have investigated hyper-heuristics for discovering decision tree building algorithms. A key aspects of any hyper-heuristic approach, when the general aim is towards deriving algorithms that may be generally useful on other problems or datasets, is to understand what are the relevant aspects of 'state' during the solution-construction process. That is, when the decision is made to use a particular heuristic, it is on the basis of specific observations of the current partial tree and the dataset. We expect that the prospects for hyper-heuristic methods in decision tree induction, and in data mining in general, will depend much on finding appropriate measures on which to base this decision. Such measures will be ones that have a chance of general salience, rather than being too specific to the dataset(s) in question. In this paper the state information is the set of entropy values among the remaining attributes, and the rules effectively suppose that the appropriate choice of heuristic can be made on the basis of a simple characterization of the distribution of entropy values.

This method has clearly yielded some degree of success. In each of three distinct trials, each using a separate collection of datasets, an evolved HHDT algorithm was able to generally outperform 12 alternatives on unseen test sets. The comparative algorithms are equivalent to ID3 using a single chosen heuristic, and included among them the most popular ID3 variants.

In terms of performance on the given datasets, the results here are not difficult to beat using alternative specialized techniques. However, in terms of developing speedy constructive DT algorithms with good performance, HHDT is clearly promising. One possibility that arises from this work is the potential for finding an HHDT ruleset (which we might call, for example, ID4) that generally outperforms the classic ID3 on arbitrary collections of datasets that were not seen during the HHDT training process. In results not reported here, tests of HHDT rulesets have found them competitive with ID3 on unseen datasets (as opposed to, as used here, unseen holdout portions of the datasets used for training), but neither better nor worse with statistical significance. But we expect that such is achievable given better understanding of how to choose appropriate collections of datasets for training, and given effective choices of state indicators to use within the rules.

Meanwhile, there is clearly promise for the current HHDT method for developing effective DT builders for datasets that are similar (in senses yet to be defined) to those used during training (indeed, Pappa and Freitas [9] come to similar conclusions). A clear application niche for such a method is where there are *recurring datasets*. This happens, for example, in scenarios where classifiers need to be kept up to date, or entirely refreshed, by constant re-learning from new data that comes from a consistent source. This is a very common situation. E.g. many financial trading systems employ classifiers that are trained on data from a fixed recent time window, and these need to be relearned daily or monthly with an updated dataset. In commerce applications, consumer buying patterns are learned from 'basket' or similar data, and classifiers must be continually regenerated from new data to maintain insight into current patterns. This notion is also discussed in the hyperheuristic combinatorial optimization context, for example in [34] where the authors suggest re-using a HH for solving timetabling problems on problems similar to the one used to evolve the HH.

In future work we will test HHDT in the context of developing a DT building ruleset that must be constantly reused on continually updated datasets. We will also examine alternative ways to represent state information in the rules, towards finding a new generally useful DT builder, or perhaps one that specializes in a wide class of datasets.

ACKNOWLEDGEMENT

We thank Motorola Ltd for sponsorship of this work.

REFERENCES

[1] P. Ross, Hyper-heuristics, Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques (E. K. Burke and G. Kendall, eds.), Springer, 2005, pp. 529-556.

[2] H. Fisher and G. L. Thompson, Probabilistic learning combinations of local job-shop scheduling rules, Factory Scheduling Conference (Carnegie Institute of Technology), 1961.

[3] H. L. Fang, P. Ross, and D. Corne, A promising genetic algorithm approach to job shop scheduling, rescheduling, and open-shop scheduling problems, 5th ICGA Morgan Kauf., 1993, pp. 375–382.

[4] Zhang, W. and Dietterich, T. G. A reinforcement learning approach to job-shop scheduling. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, 1995, pages 1114-1120.

[5] Zhang, W. and Dietterich, T. G. High-performance job-shop scheduling with a time-delay TD(λ) network. In Touretzky et al. (eds) Advances in Neural Information Processing Systems: Proc. of the 1995 Conf., 1996, pages 1024-1030, Cambridge, MA. MIT Press.

[6] J. Gratch, S. Chien, and G. DeJong, Learning search control knowledge for deep space network scheduling, Proc. 10th Int;l Conf. on Machine Learning (Amherst, MA), 1993, pp. 135–142.

[7] Ross P., Hart E., Marin-Blazquez J.G. and Schulenberg S. Learning a procedure that can solve hard bin-packing problems: a new GA-based approach to hyperheuristics. In Proc. GECCO 2003.

[8] Terashima-Marin H., Moran-Saavedra A., Ross, P. Forming hyper-heuristics with GAs when solving 2D-regular cutting stock problems. IEEE Congress on EC, v.2. 2005. pp. 1104-1110.

[9] Automatically evolving rule induction algorithms. G. L. Pappa and A. A. Freitas. In Fuernkranz, Scheffer, and Spiliopoulou, (eds), Proc.17th ECML, LNCS volume 4212 pp. 341-352, 2006.

[10] Pappa, G.L., Freitas, A.A.: Towards a genetic programming algorithm for au-tomatically evolving rule induction algorithms. In Furnkranz, J., ed.: Proc.ECML/PKDD-2004 Workshop on Advances in Inductive Learning. (2004) 93—108.

[11] Mitchell T. Machine Learning. McGraw-Hill, 1997.

[12] Quinlan J.R. Induction of Decision Trees. Machine Learning 1: 81-106. 1986.

[13] Kononenko I., Bratko I. and Roskar E. Experiments in Automatic Learning of Medical Diagnostic Rules. Technical Report, Jozef Stefan Institute, Ljubljana. 1984.

[14] Smyth P. and Goodman R.M. Rule Induction using Information Theory. Knowledge Discovery in Databases. 1991. pp.159-176.

[15] Quinlan J.R. C4.5: Programs for Machine Learning. Morgan Kaufman, Los Altos CA. 1993.

[16] López De Mántaras R. A Distance-Based Attribute Selection Measure for Decision Tree Induction. Machine Learning 6: 81-92. 1991.

[17] Brieman, L., Friedman J., Olshen R. and Stone C. Classification and Regression Trees. Chapman & Hall, Wadsworth Inc. NY. 1984.

[18] Kononenko I. Estimating Attributes: Analysis and Extensions of RELIEF. In Proc. 7th Eur. Conf. on Mach. Learn. 1994. pp. 171-182.

[19] Zhou X.J. and Dillon T.S. A Statistical-Heuristic Feature Selection Criterion for Decision Tree Induction. IEEE Trans. on Pattern Analysis and Machine Intelligence, 3(8). 1991. pp. 834-841.

[20] Smyth P. and Goodman R.M. Rule Induction using Information Theory. Knowledge Discovery in Databases. 1991. pp.159-176.

[21] Kononenko I. On Biases in Estimating Multi-Valued Attributes. 1st Int'l Conf. on KDDM. 1995. pp. 1034-1040.

[22] Baim P.W. A Method for Attribute Selection in Inductive Learning Systems. IEEE Trans. on PAMI, 10: 888-896. 1988.

[23] Kira K. and Rendell L. The Feature Selection Problem: traditional methods and a new algorithm. In Proceedings of the 10th National Conference on Artificial Intelligence. 1992. pp. 129-134.

[24] Michie D. Personal Models of Rationality. Journal of Statistical Planning and Inference, 25: 381-399. 1990.

[25] Mingers J. An Empirical Comparison of Selection Measures for Decision-Tree Induction. Machine Learning, 3(4) 1989. pp. 319-342.

[26] Breiman L. Technical note: some properties of splitting criteria. Machine Learning, v.24, n.1. 1996. pp.41-47.

[27] Badulescu L.A. The Choice of the Best Attribute Selection Measure in Decision Tree Induction. Annals of University of Craiova, Math. Comp. Sci. Ser., v. 34. 2007. pp. 88-93.

[28] Niblett T. and Bratko I. Learning Decision Rules in Noisy Domains. In Proceedings of Expert Systems, Cambridge University Press. 1986.

[29] Quinlan J.R. Simplifying Decision Trees. International Journal of Man-Machine Studies, 27(3). 1987. pp. 221-248.

[30] Asuncion, A. & Newman, D.J. (2007). UCI Machine Learning Repository [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: U. of Calif., School of Info. and Computer Science.

[31] Han J. and Kamber M. Data mining: concepts and techniques. Morgan Kaufmann, San Francisco, CA. 2000.

[32] Fayyad U.M. and Irani K.B. Multi-interval Discretization of Continuous-Valued Attributes for Classification Learning. In Proc. 13th Int'l Joint Conf. on Artificial Intelligence. 1993. pp. 1022-1027.

[33] Kerber R. ChiMerge: Discretization of Numeric Attributes. In Proc. 9th Int'l Conf. of AAAI, 1992. pp. 123-128.

[34] Terashíma-Marin H., Ross P. and Valenzuela-Rendón M. Evolution of Constraint Satisfaction Strategies in Examination Timetabling. In Proc. GECCO 1999. pp. 635-642.