Evolutionary approach for automated component-based decision tree algorithm design

Miloš Jovanović, Boris Delibašić*, Milija Vukićević, Milija Suknović and Milan Martić
Faculty of Organizational Sciences, University of Belgrade, Belgrade, Serbia

Abstract. This paper proposes a framework for automated design of component-based decision tree algorithms. These algorithms are being constructed by interchanging components extracted from decision tree algorithms and their partial improvements. Manual selection of the best-suited algorithm for a specific problem is a complex task because of the huge algorithmic space derived from component-based design. The proposed framework searches through the algorithmic space with an evolutionary algorithm by interchanging components and tuning parameters, and finds a near optimal algorithm for a specific problem. Through experiments we show that using this meta-heuristic is justified in automated component-based algorithm design. This approach is useful not only as an algorithm design help, but also as a technology enhanced learning tool, which aids the understanding of the algorithms.

Keywords: Decision tree, classification, evolutionary algorithm, automated algorithm design, reusable components, component-based algorithms

1. Introduction

A lot of effort has been put in the algorithm selection problem since the seminal paper of [37]. These efforts were systematically described in [41]. In the same paper, Smith-Miles proposed a framework for automated algorithm selection and algorithm design.

We propose a framework for automated algorithm design that could be used as an extension to the Smith-Miles framework. This idea is shown in Fig. 1, where the automated algorithm designer is imputed.

The problem with designing new and refining existing algorithms is that implementation usually needs a lot of time because it is often done from scratch. The main cause of this problem is the lack of standards for algorithms implementation and therefore difficult reuse of existing algorithms and their parts. The need for standardized data mining algorithms that could be fairly tested as wholes as well as their integral parts is reported in [42]. Additionally, as stated by No Free Lunch theories [50], there isn’t an algorithm that suits all problems soundly, but one can always make an appropriate algorithm for a specific problem.

To fulfill these goals (standardize algorithms, and extend the space of available algorithms), component-based design of data mining algorithms was proposed in [12,13]. This approach allows users...
to design and evaluate their own component-based algorithms, with the help of intuitive GUI by simply assembling components. In [13] 80 component-based decision tree algorithms have been proposed and evaluated on 15 datasets. Results of components’ influence on algorithm performance, as statistical significance of the differences found were presented. That study suggested that it is justified, for a specific dataset, to search for the optimal component interplay (component-based algorithm) instead of looking for the optimal among predefined (original) algorithms.

However, manual component-based algorithm design and parameter tuning is inefficient when the algorithm and parameter spaces are huge. Automated algorithm design (AAD), proposed in this paper, generates component-based algorithms and can be used after algorithm refinement (Fig. 1). Algorithm refinement, on the other hand, can propose either a new algorithm, that doesn’t have to be component-based, or an algorithm part that can be easily used in AAD as a component.

For a newly proposed algorithm part (component), AAD generates the near-optimal component interplay with already existing components in the AAD component repository (Fig. 2), thus designing a new algorithm which can extend the meta-learning algorithm repository.

In this paper, we propose an evolutionary algorithm for automated design of decision tree induction algorithms. Generally speaking, any meta-heuristic could be used for algorithm design, and could be applied to other component based data mining algorithms like e.g. [12].

The proposed framework is shown in Fig. 2. Currently, it doesn’t use empirical rules gained from meta-learning, but searches the space of component-based algorithms and proposes a near-optimal algorithm by classification accuracy.

This paper is further structured as follows. In Section 2, related work in this area of research is presented. Section 3 explains component-based design and the motivation behind it. In Section 4, we present our proposed framework based on evolutionary algorithm meta-heuristics. Section 5 gives several experimental results of the proposed framework, and the conclusion and further research directions are discussed in Section 6.
2. Related work

Combining meta-heuristics and data mining (DM) algorithms is often found in the literature. In [16] authors presented a survey paper on how data mining algorithms can be used for hybridizing meta-heuristics. Survey paper about combining operations research and data mining algorithms was presented in [30].

Meta-heuristics are often used as search tools in the solution (model) space of classification algorithms. E.g. [32] used genetic algorithms to evolve classification decision trees. Genetic algorithms based on a lexicographic multi-objective criterion for the problem of decision tree induction are proposed by [4]. In [11] the authors proposed a genetic algorithm for constructing compact binary decision trees. [26,33,43] used ant colony and particle swarm optimization techniques for generating classification models.

Meta-heuristics are also used for improving DM classifier performance. A hybrid “decision tree/genetic algorithm” method that solves the problem of small disjoint sets in decision tree models was proposed in [10]. Genetic algorithms for improving decision tree models with respect to misclassification and test cost was implemented in ICET system [46].

Another popular approach of incorporation of optimization techniques in data mining algorithms is parameter space search. [1] used genetic algorithms for parameter tuning of fuzzy classifiers based on decision tree initialization. [23] utilized simulated annealing approach for parameter determination of support vector machine and feature selection. Simulated annealing and genetic algorithms for choice of kernel parameters for SVM are combined in [34]. In [21] evolutionary sampling as a solution for the class imbalance problem is used. In the same work, authors proposed a research prototype, eVann that implements a genetic algorithm based optimizer for modeling machine learning algorithm parameters. Improving classifier performance by selection of representative examples and attributes with help of genetic algorithms is presented in [38].

Automated algorithm design is, however, a relatively new research topic. Still, it can be found in the operations research, as well as in the data mining community. In [40] a framework for designing hybrid combinatorial optimization algorithms is proposed. [18] developed a design framework for local
search algorithms. The genetic programming approach [22] and directions for future research on automatic design of data mining algorithms is described in [31]. In [47] a hyper-heuristic for decision tree algorithm construction is proposed, where decision tree algorithm had been built with different split evaluation measures. The choice of an evaluation measure was defined by hyper-heuristics (set of rules for evaluation measure selection).

Our approach for decision tree automated algorithm design is based on meta-heuristic search in the space of algorithms which are designed by different combinations of reusable components (RCs).

Reusable component design is a research topic that originated from architecture [2], but its biggest success was in software engineering [14]. Meta-heuristics can be used to identify the right reusable software components, e.g. [5] use an ant colony to discover rules for representation of various domains of reusable components. Further, these rules help in retrieving an appropriate component from the RC repository.

A framework for generic component-based decision tree algorithm design was proposed in [44]. That framework was further evaluated in [13]. Because of the large space of algorithms in component-based algorithm design, the need for AAD emerged. In [44], crucial sub-problems in decision tree algorithm design were identified. RCs were implemented in a repository with the goal to solve sub-problems. Easy extendable RC repository, sub-problem structure and generic algorithms provided a large algorithm space that has to be searched through in order to find the best suited algorithm for the data at hand. The whole framework was made open-source, making it more possible to integrate community efforts in data mining algorithm development [48].

We propose extending the Smith-Miles framework [41] with automated algorithm design thus enabling the extension of the algorithmic space. This is also supported by [9], where authors concluded that further development of meta-learning frameworks should be directed towards "extension of classifier pool, available to the user, and addition of new features for dataset characterization".

3. Component-based design

The data analysis process is changing rapidly, and will be more and more standardized in the years to come. The need for standardized data mining algorithms and component interchange between those algorithms is reported in [42]. That article supports the development of open-source frameworks that will serve the machine learning and data mining community for fast algorithm development and fair performance comparison between algorithms and their integral parts. It is also emphasized that such an approach would speed up the development and application of new data mining algorithms because it would enable:

- Combining advantages of various algorithms,
- Reproducing scientific results,
- Comparing algorithms in more details,
- Building on existing resources with less re-implementation,
- Faster adoption in other disciplines and industry,
- Collaborative emergence of standards.

The question of whether the combination of components could improve algorithm performance asked by [42] is positively answered in [13]. This approach is strongly supported by the series of No Free lunch theories [50] where authors proved that there is no optimal algorithm for every problem, but one can always design an optimal algorithm for the data at hand. According to these theories, it is the task of great importance to provide a large space of algorithms and enable efficient search.
Reusable components (RCs) are often used in software engineering. They are defined as triplets consisting of concept, content and context [45]. The concept is the description of what a component does with its inputs and outputs. The content describes how the RC is realized, which is encapsulated and hidden from the end user. The context explains the application domain of the component, which helps to find the right component for a specific problem.

Reusable components are algorithmic units on a lower level of granularity. Designing algorithms with RCs allows easier refinement of algorithms by replacing or adding new RCs with the same functionality without having to recode the rest of the algorithm.

In data mining algorithms, this proves to be the right thing to do because of the following reasons:
1. There are many different implementations for the same algorithm in open-source and commercial software.
2. The algorithms are coded mostly as black-boxes where it is hard to refine an algorithm without having to implement the algorithm from scratch.
3. Adoption and implementation of algorithms and their partial improvements found in the literature usually need a lot of time, and many of these are neither implemented and made available to users nor tested in a fair way.

We will show how RC based design helps in automatic algorithm design. In [44] the main principle of RC identification was to discriminate algorithm design structure from specific algorithm solutions. The design structure was saved in a generic algorithm shell while the specific solutions were identified as RCs. This will be explained in more detail in the next subsection on component-based decision trees. Besides component-based design, there are also other approaches for algorithm characterization and standardization. E.g. an ontology-based meta-heuristic framework for better algorithm characterization is proposed in [15]. Component-based algorithm characterization could probably be used for improvement of data mining ontologies for meta-learning, which should be further explored in the future.

3.1. Component-based decision trees

In [13,44] RCs from decision tree algorithms ID3 [35], C4.5 [36], CART [8], and CHAID [19] were identified. The choice of these algorithms was guided by their availability within popular software, together with a survey that points out more popular algorithms [51]. Further, partial algorithm improvements in [24,25] have been analyzed as well, RCs were classified according to frequently occurring sub-problems in decision tree design. Sub-problems define the design structure of algorithms. RCs are solutions for sub-problems within the process of decision tree induction. For each of these sub-problems, several RCs were identified. The sub-problems play an important role in the decision tree design because there are many possible solutions for each sub-problem, i.e. RCs and the algorithm designer can choose which RC should be used for solving a specific sub-problem.

In [13] the following sub-problems (design structure) were identified for decision tree algorithms:
1. Remove insignificant attributes – a sub-problem that is used in every node during the tree growing process to reduce the number of potential attributes that could contain useful splits. It can improve the run time of algorithms, while keeping the accuracy of the tree high, and sometimes even improve accuracy [13].
2. Create split – this sub-problem is used in every node for creating possible node splits.
3. Evaluate split – this sub-problem is used to evaluate the quality of created split candidates.
4. Stop criterion – is used for stopping tree growth before the tree grows completely.
5. Prune tree – is used after tree growth on the decision tree model to shrink the tree with the goal to reduce the complexity of the tree while reducing generalization error.
Table 1 shows all sub-problems, RCs and their parameters that have been used in AAD in this paper, as well as default, minimal and maximal values for the parameters.

Component-based design includes identifying RCs in algorithms and algorithm improvements and allowing them to be used in other algorithms. For example, CHAID includes an RC that groups attribute categories and splits tree nodes on grouped categories (This RC was named “Significant” in Table 1). In decision tree growth, instead of trying to branch a categorical attribute on all categories (as in ID3 or C4.5), or make binary groupings (as in CART), CHAID tries to estimate the optimal grouping of attribute categories using the chi-square test for estimating category differences. For an attribute with 5 values, CHAID can decide to group the values into 2, 3, 4 or 5 separate value groups, based on the difference in class distribution.

This behavior can be used independently within any decision tree induction algorithm, as an RC. It could solve the problem of an overly detailed number of categories of an attribute that are not informative for a decision. In [13] using the CHAID grouping attribute RC showed benefits in terms of accuracy and execution time.
Table 2
Benchmark datasets from UCI repository used in this study

<table>
<thead>
<tr>
<th>ID</th>
<th>Dataset</th>
<th>Size</th>
<th>Attributes</th>
<th>Classes</th>
<th>Missing values</th>
<th>Class distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bre</td>
<td>Breast</td>
<td>286</td>
<td>9</td>
<td>2</td>
<td>Yes</td>
<td>201/85</td>
</tr>
<tr>
<td>Car</td>
<td>Car</td>
<td>1728</td>
<td>6</td>
<td>4</td>
<td>No</td>
<td>1210/384/69/65</td>
</tr>
<tr>
<td>Cmc</td>
<td>Contraceptive</td>
<td>1473</td>
<td>9</td>
<td>3</td>
<td>No</td>
<td>629/511/333</td>
</tr>
<tr>
<td>Cra</td>
<td>Credit-a</td>
<td>690</td>
<td>14</td>
<td>2</td>
<td>Yes</td>
<td>383/307</td>
</tr>
<tr>
<td>Crg</td>
<td>Credit-g</td>
<td>1000</td>
<td>20</td>
<td>2</td>
<td>Yes</td>
<td>700/300</td>
</tr>
<tr>
<td>Har</td>
<td>Hayes roth</td>
<td>160</td>
<td>5</td>
<td>3</td>
<td>No</td>
<td>65/64/31</td>
</tr>
<tr>
<td>Hep</td>
<td>Hepatitis</td>
<td>155</td>
<td>19</td>
<td>2</td>
<td>Yes</td>
<td>123/32</td>
</tr>
<tr>
<td>Hyp</td>
<td>Hypothyroid</td>
<td>3772</td>
<td>21</td>
<td>3</td>
<td>Yes</td>
<td>3488/191/93</td>
</tr>
<tr>
<td>Krk</td>
<td>King-rook-vs-King-pawn</td>
<td>3169</td>
<td>36</td>
<td>2</td>
<td>No</td>
<td>1669/1527</td>
</tr>
<tr>
<td>Lid</td>
<td>Liver-disorders</td>
<td>345</td>
<td>7</td>
<td>2</td>
<td>Yes</td>
<td>200/145</td>
</tr>
<tr>
<td>Lyn</td>
<td>Lymphography</td>
<td>148</td>
<td>18</td>
<td>4</td>
<td>No</td>
<td>81/61/42</td>
</tr>
<tr>
<td>Nur</td>
<td>Nursery</td>
<td>12960</td>
<td>8</td>
<td>5</td>
<td>No</td>
<td>4320/4266/4044/328/2</td>
</tr>
<tr>
<td>Pos</td>
<td>Post-operative</td>
<td>90</td>
<td>8</td>
<td>3</td>
<td>Yes</td>
<td>64/24/2</td>
</tr>
<tr>
<td>Spe</td>
<td>Spectrometer</td>
<td>531</td>
<td>102</td>
<td>2</td>
<td>Yes</td>
<td>172/15</td>
</tr>
<tr>
<td>Sta</td>
<td>Statlog heart</td>
<td>270</td>
<td>13</td>
<td>2</td>
<td>No</td>
<td>150/120</td>
</tr>
<tr>
<td>Vot</td>
<td>Voting</td>
<td>435</td>
<td>16</td>
<td>2</td>
<td>Yes</td>
<td>267/168</td>
</tr>
</tbody>
</table>

RCs are combined following a generic structure presented in Fig. 3 [13]. It is important to notice that the units in Fig. 3 should not be regarded as algorithmic steps, but as sub-problems that define an algorithm structure.

The generic algorithm, sub-problem structure and RC repository [13] provide large space of component-based algorithms and can supposedly allow better adaptation of decision tree algorithms for data at hand.

4. Algorithm space search by evolutionary algorithms

We defined the algorithm search space over all possible RC combinations on sub-problems (960 algorithms) from Table 1. Since there are no constraints in combining decision tree RCs, the search space size grows super-linearly with addition of new RCs. Furthermore, as RCs can have parameters as well, this broadens the algorithm search space even more.

We used classification accuracy based on 10-fold cross-validation test for the evaluation of the solutions (decision tree algorithms). Since classification accuracy over the algorithmic space has no known theoretical properties (e.g. linearity, convexity, etc.), it is reasonable to use the meta-heuristic approach when searching for the best solution. Furthermore, as decision trees are greedy algorithms (also heuristics), and this approach could also be considered as hyper-heuristic, as in [47].

Algorithm space search doesn’t need any prior knowledge about the behavior of the algorithms, but relies on algorithms evaluation for the problem at hand, which sometimes can be costly. Therefore, this approach could benefit from reduction of the search space based on empirical rules from meta-learning databases. We leave this integration for future studies.

4.1. Evolutionary algorithm setup

In this study, we employed evolutionary approach for searching a near-optimal decision tree algorithm for a selected dataset. Genetic coding of chromosomes, as well as genetic operators, is fully customized to conform to the component-based algorithms. Each sub-problem is represented by a custom gene, whose values (alleles) represent an RC with its parameters. An example of a chromosome is given in
Table 3
Parameters of the evolutionary algorithm

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>15</td>
</tr>
<tr>
<td>Generations</td>
<td>30</td>
</tr>
<tr>
<td>Mutation rate</td>
<td>12.5%</td>
</tr>
<tr>
<td>Crossover rate</td>
<td>35%</td>
</tr>
<tr>
<td>Percentage of generations with surrogate function</td>
<td>70%</td>
</tr>
<tr>
<td>Surrogate dataset sample size</td>
<td>30%</td>
</tr>
<tr>
<td>Random seed in RapidMiner</td>
<td>1</td>
</tr>
</tbody>
</table>

Fig. 3. Generic decision tree structure.

Fig. 4. An example of the chromosome containing genes for the ID3 algorithm.

Fig. 4, and it shows the gene representation of the ID3 algorithm reconstructed with the components. It uses “Multiway categorical” splitting strategy (component), evaluates possible splits with “Information gain”, and in this example has “Tree depth” stopping criteria with parameter 10. It doesn’t use any RIA and Prune tree components (denoted by ‘!’). Examples of chromosomes could also be seen in Table 5.

All sub-problem genes can react specifically to applied genetic operators, meaning that the custom genes define which components vary, range of parameter values, and other consistency issues which ensure that the gene modified by a genetic operator is valid.

For genetic operators, we used standard roulette-wheel selection, uniform crossover, and gene dependent mutation. The probability that genetic operators will affect the solution is additionally controlled by rates of appliance. Each chromosome has an equal chance for being selected. The initial population is created at random, with each component (allele) having the same probability of being a part of the genes from the initial population.

Selection also uses elitism to preserve the single best chromosome in the population. Uniform crossover exchanges randomly chosen genes with defined parameters from two parent chromosomes and produces two child chromosomes. Mutation is switching RCs in a sub-problem gene randomly, and also changes the parameters of an RC with some chosen probability. Mutation can affect both component choice and parameter setting, and it is possible to define separate mutation rates for RCs and their parameters. Thus, the search can either be more focused on RCs or on parameters.
Table 4

<table>
<thead>
<tr>
<th>Data-set</th>
<th>Proposed algorithm</th>
<th>Max (Brute force) – EA accuracy</th>
<th>Max (Brute force)</th>
<th>Min (Brute force)</th>
<th>No. of evaluations with surrogate function (out of 960)</th>
<th>No. of evaluations with fitness function (out of 960)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bre</td>
<td>C-S-C-S-P</td>
<td>2.77%</td>
<td>76.60%</td>
<td>64.06%</td>
<td>130</td>
<td>63</td>
</tr>
<tr>
<td>Car</td>
<td>!-A-GR-!-!</td>
<td>0.17%</td>
<td>98.09%</td>
<td>76.85%</td>
<td>80</td>
<td>38</td>
</tr>
<tr>
<td>Cmc</td>
<td>C-B-I-C-C</td>
<td>0.13%</td>
<td>55.73%</td>
<td>43.18%</td>
<td>94</td>
<td>49</td>
</tr>
<tr>
<td>Cra</td>
<td>C-A-G-C-S</td>
<td>0.00%</td>
<td>86.38%</td>
<td>77.83%</td>
<td>115</td>
<td>64</td>
</tr>
<tr>
<td>Crg</td>
<td>!-M-D-S-R</td>
<td>0.00%</td>
<td>74.90%</td>
<td>65.10%</td>
<td>79</td>
<td>32</td>
</tr>
<tr>
<td>Har</td>
<td>!-A-GR-C-!</td>
<td>0.62%</td>
<td>83.75%</td>
<td>37.50%</td>
<td>122</td>
<td>43</td>
</tr>
<tr>
<td>Hep</td>
<td>!-A-D-S-P</td>
<td>2.54%</td>
<td>84.46%</td>
<td>76.08%</td>
<td>163</td>
<td>45</td>
</tr>
<tr>
<td>Hyp</td>
<td>!-A-I-!-!</td>
<td>0.03%</td>
<td>99.79%</td>
<td>95.44%</td>
<td>132</td>
<td>53</td>
</tr>
<tr>
<td>Krk</td>
<td>!-B-I-!-!</td>
<td>0.00%</td>
<td>99.69%</td>
<td>94.99%</td>
<td>109</td>
<td>35</td>
</tr>
<tr>
<td>Lid</td>
<td>!-M-C-C-P</td>
<td>7.55%</td>
<td>68.41%</td>
<td>57.97%</td>
<td>97</td>
<td>32</td>
</tr>
<tr>
<td>Lym</td>
<td>!-B-C-D-M</td>
<td>0.05%</td>
<td>83.19%</td>
<td>54.81%</td>
<td>106</td>
<td>43</td>
</tr>
<tr>
<td>Nur</td>
<td>!-S-D-!-P</td>
<td>0.00%</td>
<td>99.88%</td>
<td>83.05%</td>
<td>88</td>
<td>40</td>
</tr>
<tr>
<td>Pos</td>
<td>C-S-I-C-S</td>
<td>0.00%</td>
<td>71.11%</td>
<td>50.00%</td>
<td>136</td>
<td>52</td>
</tr>
<tr>
<td>Spe</td>
<td>!-B-I-S-S</td>
<td>0.00%</td>
<td>91.99%</td>
<td>81.73%</td>
<td>155</td>
<td>72</td>
</tr>
<tr>
<td>Sta</td>
<td>C-B-C-C-R</td>
<td>5.92%</td>
<td>81.85%</td>
<td>55.56%</td>
<td>89</td>
<td>25</td>
</tr>
<tr>
<td>Vot</td>
<td>C-S-GR-!-M</td>
<td>0.01%</td>
<td>96.55%</td>
<td>61.38%</td>
<td>123</td>
<td>41</td>
</tr>
</tbody>
</table>

The fitness function is the algorithm accuracy on cross-validation test. The WhiBo framework [48] is used to evaluate a designed component-based algorithm (chromosome). Since algorithm evaluation can be time expensive, several strategies are used to address this problem. First, implementation of a cache for evaluated function values enables equal algorithms (chromosomes) to be evaluated only once. This is important, since evolutionary approach, within populations and generations, often replicates and reconstructs points in space that has been previously evaluated.

Another strategy used to relax the burden of costly algorithm evaluation is the introduction of surrogate fitness function that enables fast estimation of the fitness function. In this study, surrogate function evaluates an algorithm on a sample of a specific dataset, since this can reduce the run time. Sampling is done in advance, using stratified random sampling, which preserves the class distribution of the sample.

The evolutionary algorithm runs in two phases. In the first phase, the evolutionary algorithm runs using surrogate function instead of full fitness function for a defined percentage of evolutions (generations). In the second phase, the algorithm uses full fitness function. This strategy enables quick identification of solutions (algorithms) which are estimated to have good accuracy in the first phase. In the second phase, selected algorithms from phase one are refined. This way, only a small number of algorithms get evaluated on a complete dataset, which reduces the execution time.

The evolutionary algorithm is implemented using JGAP framework [27], and is using WhiBo plug-in [48] for RapidMiner [28] in order to evaluate component-based algorithms.

5. Experimental results

To evaluate the usefulness of the proposed approach, we conducted two experimental studies. The first set of experiments are conducted to test how good the algorithms found by the evolutionary algorithm (EA) were compared to the best possible algorithms on a specific dataset.

First we conducted a brute-force search on the space of 960 algorithms representing all possible combinations of RCs (with default parameters) that form a decision tree algorithm (like in [13], but using more RCs).
Then we ran the evolutionary algorithm over the same algorithmic space to see how good the algorithm proposed by EA performed, comparing the results to the best possible algorithm found by brute-force search method. If the EA produces good results, it could be used to search for a suitable algorithm in a lot larger algorithm space, where the brute-force method is inefficient.

The second set of experiments widens the algorithmic search space because both RCs and parameters are varied. Because the brute-force method would show inefficient in this case, we compared the results of the EA to the benchmark results on decision trees [7]. The datasets used in both studies are shown in Table 2, and were taken from UCI repository [3]. Table 2 also contains basic dataset properties, and also the class distribution for each dataset, which can be used to judge on the minimal required accuracy of the classifiers, set by the majority class relative frequency.

The goal of this study is to use a search procedure (evolutionary algorithm), that can find a near optimal algorithm with as little evaluations of algorithms as possible. The authors in [13] also suggest that component-based algorithms form a space of algorithms that is prominent to search, since their study showed that popular black-box decision tree algorithms were not the top performing ones.

The evolutionary algorithm used for the first experiment set is parameterized as shown in Table 3. Note that the mutation rate stands for the probability that the whole chromosome (individual) gets mutated, which might affect more than one gene of that chromosome. Because the EA is a stochastic procedure, the random seed is also reported, to enable experiment reproducibility.

Results of this experiment set are shown in Table 4. For each dataset, the evolutionary algorithm proposes a decision tree algorithm, which is further compared with the brute-force search results. “Max (Brute force) – EA accuracy” describes how far the proposed algorithm accuracy is from the best algorithm accuracy found by brute-force search. The table also shows maximal and minimal accuracies of all algorithms on a specific dataset. Finally, the last two columns show what part of the solution space is actually evaluated. This is done separately by the surrogate and the fitness function since surrogate function is calculated a lot faster than the original fitness function. This shows how intensive this search is compared to brute-force search.

Table 5

<table>
<thead>
<tr>
<th>Data-set</th>
<th>Best algorithm design found by EA</th>
<th>Accuracy</th>
<th>Std. Dev.</th>
<th>Best accuracy found in the first experiment</th>
<th>Best decision tree from [7]</th>
<th>Accuracy from best decision tree [7]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bre</td>
<td>C - M - D - C (0.50) - R</td>
<td>75.24%</td>
<td>1.49%</td>
<td>73.83%</td>
<td>C4.5</td>
<td>76.22%</td>
</tr>
<tr>
<td>Car</td>
<td>1 - B - C - I - P (0.05)</td>
<td>98.32%</td>
<td>0.27%</td>
<td>97.92%</td>
<td>C4.5</td>
<td>97.28%</td>
</tr>
<tr>
<td>Cmc</td>
<td>1 - B - C - S (727) - C (0)</td>
<td>56.61%</td>
<td>0.72%</td>
<td>55.60%</td>
<td>C4.5</td>
<td>56.08%</td>
</tr>
<tr>
<td>Cra</td>
<td>C - M - G - C (0.21) - S (86)</td>
<td>86.67%</td>
<td>0.23%</td>
<td>86.38%</td>
<td>ADTree</td>
<td>85.51%</td>
</tr>
<tr>
<td>Crg</td>
<td>C - I - C (0.05) - C (1)</td>
<td>75.10%</td>
<td>0.88%</td>
<td>74.90%</td>
<td>LMT</td>
<td>75.90%</td>
</tr>
<tr>
<td>Har</td>
<td>1 - A - D - D (12) - !</td>
<td>85.62%</td>
<td>1.21%</td>
<td>83.13%</td>
<td>C4.5</td>
<td>71.43%</td>
</tr>
<tr>
<td>Hep</td>
<td>C - M - I - S (731) - R</td>
<td>83.88%</td>
<td>0.01%</td>
<td>81.92%</td>
<td>C4.5</td>
<td>85.16%</td>
</tr>
<tr>
<td>Hyp</td>
<td>C - S (0.076; 0.042) - GR - D (9) - M (13)</td>
<td>97.79%</td>
<td>(dog)</td>
<td>99.76%</td>
<td>C4.5</td>
<td>99.00%</td>
</tr>
<tr>
<td>Krk</td>
<td>1 - B - I - P (0.26)</td>
<td>99.69%</td>
<td>0.07%</td>
<td>99.69%</td>
<td>LMT</td>
<td>99.75%</td>
</tr>
<tr>
<td>Lid</td>
<td>C - S (0.24; 0.03) - G - I - M (424)</td>
<td>69.86%</td>
<td>(dog)</td>
<td>60.86%</td>
<td>C4.5</td>
<td>70.72%</td>
</tr>
<tr>
<td>Lym</td>
<td>1 - B - G - D (10) - M (221)</td>
<td>83.19%</td>
<td>1.55%</td>
<td>83.14%</td>
<td>LMT</td>
<td>83.11%</td>
</tr>
<tr>
<td>Nur</td>
<td>1 - S (0.28; 0.018) - D - I</td>
<td>99.88%</td>
<td>0.04%</td>
<td>99.88%</td>
<td>C4.5</td>
<td>99.51%</td>
</tr>
<tr>
<td>Pos</td>
<td>1 - M - G - S (719) - S (305)</td>
<td>71.11%</td>
<td>0.00%</td>
<td>71.11%</td>
<td>Rep-tree</td>
<td>71.11%</td>
</tr>
<tr>
<td>Spe</td>
<td>C - A - D - S (50) - C (1)</td>
<td>91.99%</td>
<td>0.01%</td>
<td>91.99%</td>
<td>C4.5</td>
<td>91.99%</td>
</tr>
<tr>
<td>Sta</td>
<td>1 - B - I - S (343) - P (0.06)</td>
<td>81.85%</td>
<td>0.11%</td>
<td>75.93%</td>
<td>LMT</td>
<td>84.43%</td>
</tr>
<tr>
<td>Vot</td>
<td>C - B - GR - I - M (204)</td>
<td>96.54%</td>
<td>0.01%</td>
<td>96.54%</td>
<td>LMT</td>
<td>96.78%</td>
</tr>
</tbody>
</table>

Decision tree component-based algorithms composed by the evolutionary algorithm, and compared to benchmark results. Best accuracies are underlined.
A proposed component-based algorithm can be read as follows: E.g. on the “cmc” dataset the proposed algorithm is C-B-I-C-C. That means that the algorithm is using for “Remove insignificant attributes” the “chi square/anova f test” RC, for “Create split (categorical)” the “Binary” RC, for “Evaluate split” the “Information gain” RC, for “Stop criteria” the “Leaf label confidence” RC and for “Prune tree” the “Cost-complexity pruning” RC. The RC for “Create split (numerical)” is not shown, because it always uses “binary” RC, as currently no other RC for solving this sub-problem is used. As the sub-problems “Remove insignificant attributes”, “Stop criteria” and “Prune tree” are optional to use, it is also possible not to use an RC (denoted with “!”) for these sub-problems. On the other hand, the sub-problem “Create split (categorical)” allows using more than one RC for solving it. In Table 1, an RC named “all” is available and it uses at the same time “binary”, “multiway”, and “significant” RCs.

The results show that on most (12) datasets the algorithms proposed by the EA differ less than 1% from the results of the brute-force search, from which on six datasets an optimal algorithm was found. The worst results are achieved on the “Lid” dataset where EA didn’t manage to find a near-optimal algorithm but found an average performing algorithm on that dataset.

The results in Table 4 show that the proposed evolutionary algorithm can be utilized to discover well performing decision-tree algorithms for a selected dataset. We notice that the evolutionary algorithm suggested an algorithm for each dataset that very often gives accuracy similar or equally good as the best algorithm on the dataset. Furthermore, the algorithm is suggested by the EA in less than 45 calculations of the fitness function on average, in the space of 960 algorithms. This gives us indications that it is reasonable to utilize EA in an even larger search space.

The employed cache for already calculated fitness evaluations also proved beneficiary. For each dataset, from 111 to 227 evaluations were performed, and total number of evaluated chromosomes was 450 (30 generations * 15 chromosomes per generation). This means that the cache was boosting time performance from 98 to 305%.

The second experiment set was aimed at giving evidence on how well a component-based decision tree algorithm can perform, optimizing both components and parameters which effect the performance of the algorithm.

The search space was much bigger this time, and no previous study can be used to evaluate how far a found algorithm is from the ideal one. Still, we used the best benchmark results of decision-tree classifiers found in the experimental database [7], which are not component-based, but still a close match to our decision tree algorithm structure. This database contains performance results of well-known algorithms on publicly available datasets. It is important to note that the evaluation of the algorithms used in this database is based on cross-validation, which is the same evaluation we used in our study, so comparison is possible. The results are shown in Table 5. Aside from the found algorithm, the table also shows the results of the best decision tree algorithm from the experimental database which are from Weka [49]. Additionally, we provide comparison information to the best algorithms found in the first experiment, when the search space was more restrict (without parameter optimization). We also report the standard deviation of the accuracy for the found algorithm, after re-evaluating it 30 times with different random seeds for sampling.

The evolutionary algorithm ran for 50 generations with population of size 30. Cross-over rate was kept at 35%, while the mutation rate was increased to 16.67%. In the first 30 generations only components were mutated, while on the last 20 generations parameters were mutated 5 times more than components. This strategy was developed since mutation of components strongly affects the parameters, because if component changes, the parameters for that components become obsolete. The found algorithms are shown in code using abbreviations from Table 1, and the symbol “!” is used to note when no component
was used for a sub-problem. For example, on “Har” dataset the best found algorithm doesn’t use RCs for “Remove insignificant attributes” and “Prune tree”. It uses for “Create split (categorical)” the “All” RC and for “Evaluate split” the “distance measure” RC. It is interesting that it uses for “Stop criteria” the “maximal tree depth” RC with tree depth set to 12, which is highly specific for the dataset.

On the “Lid” dataset, for which the EA in the first experiment found the worst accuracy (when compared to the best brute-force algorithm accuracy), the EA in the second experiment found an algorithm that differed less than 1% compared to the best algorithm in [7] and had a 9% accuracy improvement compared to the first experiment. This could motivate searching through the algorithm space varying both components and parameters.

The results show that the decision tree component-based algorithm found by the evolutionary algorithm is comparable to the best algorithms in the database for a dataset. For some of the datasets, new best algorithms are discovered and set as new benchmarks. On other datasets EA yielded comparable results, and several times it underperformed, usually comparing to the LMT (Logistic Model Tree) classifier from the database. The datasets where LMT showed better results are probably better suited for the LMT’s different internal algorithm structure and logic, and is indicating that it is prominent to try to introduce some new algorithmic components (RCs) originating from LMT to WhiBo.

As expected, this experiment showed that evolutionary algorithm can optimize both components and parameters, since there was consistent accuracy improvement compared to the results in the first experiment, when parameters had been set to default values. Still, the improvements of the parameter optimizations are not large, giving evidence that the selection of components affects the accuracy far more than parameter tweaking, as also suggested by [44].

The experiment shows that by intelligently searching the component-based algorithm space, one can construct an algorithm that matches its dataset without manual selection of algorithms and adjusting of parameters. The results are not yet conclusive but give strong evidence that this area of research is prominent.

6. Conclusion and future work

Component-based data mining algorithm design provides a larger algorithmic space that can be used for better algorithm adaption to concrete problems. In [13] authors showed that decision trees designed with RCs extracted from popular algorithms, had performed significantly better on the majority of tested problems than popular decision trees had. Since there is no optimal algorithm for every problem, and one can always design optimal algorithm for the data at hand [50], it is important that data mining practitioners have a galore of algorithms to achieve good adaption to data. Still, such large algorithmic space is hard to search since the objective functions are computationally very expensive (e.g. accuracy calculated by cross-validation). This is especially true for large scale problems, where brute-force searching for the best component-based algorithm is infeasible, which was not addressed by the previous study [13]. To fully use benefits of component-based design and the enlarged algorithmic space, we propose meta-heuristic approach for algorithm design. This paper describes a framework for automated algorithm design that utilizes algorithmic space search based on evolutionary algorithms. The implemented framework uses the WhiBo platform [48] for component-based design and evaluation of these algorithms. It allows automatic design of component-based algorithms and RCs parameter tuning.

The proposed framework can be further developed in several directions:

1. Applying the proposed evolutionary algorithm on other component-based algorithms (e.g. partitioning cluster algorithms proposed in [12]).
2. Implementation of other meta-heuristic algorithms like Simulating annealing, Variable neighborhood search, Taboo search etc. to find the most appropriate meta-heuristic for component-based algorithm search.

3. Optimizing other evaluation measures instead of accuracy (like Area Under Curve), which have different properties and are often more appropriate than accuracy. This could also incorporate some multi-attribute goals, where the EA could optimize several measures at once, including the stability measured by standard deviation.

4. Extending the automated-algorithms design framework proposed in this paper with use of empirical rules from meta-learning in order to integrate into the Smith-Miles framework. The extended framework for AAD could probably be a valuable part for automatic modeling of complete data mining workflows [20,39].

5. Integrate the evolutionary algorithm in WhiBo plug-in for RapidMiner [28], and use it as decision support for users’ component-based algorithm selection.

6. Implementation in meta learning systems like MiningMart [29], MetaL Data Mining Assistant [17], IDEA [6].

7. Usage of this approach as a technology enhanced learning (TEL) tool, for students who learn algorithms and performance influence of its parts.

Experimental evidence show that utilization of evolutionary algorithms for design of well performing component-based algorithms is useful because it automates the process of finding the appropriate algorithm for data at hand by finding the right component interplay and by adjusting parameters of the assembled components.

Acknowledgements

This research was supported by a grant from the Serbian Ministry of Science and Technological Development, III 47003. The authors are also very grateful to the anonymous referees, whose comments and suggestions helped to improve the article.

References


