Case-Based Learning and Reasoning Using Layered Boundary Multigraphs

Thomas Gabel(✉) and Fabian Sommer

Faculty of Computer Science and Engineering, Frankfurt University of Applied Sciences, Frankfurt am Main, 60318 Frankfurt, Germany
{tgabel,fabian.sommer}@fb2.fra-uas.de

Abstract. Instance-based and case-based learning algorithms learn by remembering instances. When scaling such approaches to datasets of sizes that are typically faced in today's data-rich and data-driven decade, basic approaches to case retrieval and case learning quickly come to their limits. In this paper, we introduce a novel scalable algorithm for both, the retrieval and the retain phase of the CBR cycle. Our approach builds an efficient graph-based data structure when learning new cases which it exploits in a stochastic any-time manner during retrieval. We investigate its characteristics both, theoretically and empirically using established benchmark datasets as well as a specific larger-scale dataset.

1 Introduction

Retrieval in Case-Based Reasoning (CBR) takes time. For some applications, specifically data intensive ones, retrieval times can quickly become the system's bottleneck. Since this has been known for decades, a lot of research effort has been put into the development of CBR approaches to increase retrieval efficiency while retaining its efficacy. Those approaches tackle the problem in different ways, including but not limited to (a) reducing the number of cases stored in the case base, (b) using hierarchical, filtering, or multistep retrieval methods to reduce the number of query-case comparisons, (c) constructing and utilizing index structures to guide the search for similar cases, (d) improving case representation and developing tailored methods for efficient similarity assessments, or (e) using anytime algorithms whose retrieval efficacy grows with available computation time.

The core contribution of the paper at hand is a novel retrieve and retain procedure that combats the aforementioned challenges using a combination of the strategies (a), (c), and (e) listed above. According to [17], indexing cases is one of the most challenging tasks in CBR. To this end, we aim at building up a *graph-based* index structure that enhances the similarity-driven search for nearest neighbors (c), while also reducing the number of cases stored overall (a). Additionally, we design the method to be an anytime approach (e) which means that the retrieval process can be stopped at any time and that the quality of the results obtained is likely to be better if more time has been allocated to the retrieval process.

M. T. Keane and N. Wiratunga (Eds.): ICCBR 2022, LNAI 13405, pp. 193–208, 2022.
https://doi.org/10.1007/978-3-031-14923-8_13

At the heart of our contribution is the construction of an index structure that we call a *boundary graph* or, respectively, a *labeled boundary multigraph* extension of it. We build up the graph structure from case data, i.e. its topology is not fixed a priori. It is worth noting that the construction process can be applied in an online setting, i.e. no batch access to the full dataset is needed and, hence, the graph index structure can be extended as more and more cases come in. Both, the buildup as well as the employment of that graph-based index structure are inherently stochastic – a fact that we found to substantially improve the robustness of the approach as well as to reduce its dependency on other factors like the order of case presentation during learning.

We start this paper with a brief literature review on related work. Section 3 introduces boundary graphs and labeled boundary multigraphs and presents corresponding retain and retrieve procedures for constructing such graph structures as well as for utilizing them during the actual retrieval. We have tested our algorithms on a variety of classical benchmark datasets as well as on a large scale dataset from the application field of robotic soccer simulation (RSS). In Sect. 4, we present the corresponding empirical results as well as further analyzes of our algorithms' properties and their scaling behavior.

2 Background and Related Work

Since retrieval takes such a prominent position in CBR, the optimization of its efficacy and efficiency has attracted a lot of research in the past. Providing a comprehensive overview on these issues is beyond the scope of this paper, which is why we only point to work that is strongly related ours.

Case Base Maintenance (CBM [13]) aims to control the number of cases in the case base while guaranteeing a high competence of the system. Following the initial proposal of the nearest neighbor rule [6], several authors have proposed ideas to reduce the set of stored cases [12]. Among those, the family of *Instance-Based Learning* algorithms (IBL [1]) is a classic whose IB2 variant centered around the idea of adding a new case only, if its problem part would not be solved by the so far existing case base. This simple but powerful idea is also fundamentally embedded into the core of the algorithms we are proposing in this paper. Other CBM algorithms take the opposite approach and iteratively decide which cases to delete from a case base [21] which comes at the cost of requiring batch access to the case data. Another technique to limit the case base size utilizes the notion of coverage and reachability of cases [18] which henceforth was exploited by the incremental COV-FP [19] algorithm.

Index Structures for Efficient Retrieval are supposed to guide the search for similar cases. Thus, before the actual retrieval utilizing an index structure can take place that structure must be generated. Tree-based structures have frequently been employed to speed up the access to large datasets (e.g. geometric near-neighbor access trees [5] or nearest vector trees [14]). Tree-based algorithms

that do also feature online insertion capabilities include cover trees [4], boundary trees [16] (see below), or kd-trees [20] where the latter have the additional advantage of not requiring full similarity calculations at tree nodes. Many more complex retrieval methods and belonging index structures do exist, including, for example, case retrieval nets [15] or retrieval using Bayesian networks [7].

Boundary Trees [16] are a powerful tree-based index structure for similarity-based search. They consist of nodes representing training cases connected by edges such that any pair of parent and child node belongs to different classes[1]. This fact is eponymous as with each edge traversal a decision boundary is crossed.

Given a boundary tree $\mathcal{T}$ and a new query q, the tree is traversed from its root by calculating the similarity between q and all children of the current node, moving to that child which has the highest similarity to q. Boundary trees use a hyper parameter $k \in [1, \infty]$ that determines the maximal number of children any node is permitted to have. The retrieval is finished, if a leaf node has been reached or if the current (inner) node v has less than k children and the similarity between q and v is larger than the similarity between q and all children of v. This way, a "locally closest" case c^* to the query is found, meaning that neither the parent(s) of c^* nor the children of c^* are more similar.

The tree creation procedure for boundary trees is inspired by the classical IB2 algorithm [1] (see above). The next training case c_i is used as query using the so far existing boundary tree $\mathcal{T}_{i-1}$. If the result of the tree-based retrieval returns a case c^* whose solution does not match the solution of c_i, then c_i is added as a new child node of c^*. In [16], Mathy et al. propose to extend the described approach to an ensemble of boundary trees, a so-called boundary forest (BF). They train an ensemble of (in that paper, usually, ten or 50) boundary trees on shuffled versions of the training data and employ different voting mechanisms using the retrieval results of the boundary trees. The Boundary Graph approach we are presenting in the next section takes some inspiration from boundary trees which is why we also use it as a reference method in our empirical evaluations.

3 Boundary Graphs and Labeled Boundary Multigraphs

We propose a case-based technique that covers both, a method to decide which cases to store in the case base and which not as well as algorithms to build up and employ an index structure that facilitates an efficient retrieval. In what follows, we assume an attribute value-based case representation over a problem space $\mathcal{P}$ and a solution space $\mathcal{S}$ where each case $c = (p, s)$ consists of a problem part $p \in \mathcal{P}$ and a solution part $s \in \mathcal{S}$ which we can both access using the dot operator (i.e. $c.p$ or $c.s$). As usual, similarity between cases is measured using a problem similarity measure $sim_p : \mathcal{P} \times \mathcal{P} \to [0, 1]$, while the similarity between cases' solutions can be assessed using a solution similarity measure $sim_s : \mathcal{S} \times \mathcal{S} \to [0, 1]$. Note that we do not impose any further requirements on $sim_{s/p}$ throughout the rest of the paper, except that, for ease of presentation, we assume it to be symmetric.

[1] While the definition given here focuses on classification tasks, a straightforward generalization to other tasks like regression or mere retrieval can easily be made.

3.1 Boundary Graphs

We now introduce the concept of boundary graphs which we will later extend to so-called labeled boundary multigraphs. These graph structures, plus techniques to create and utilize them, represent the backbone of our entire approach.

3.1.1 Notation

For a given case base CB, a *Boundary Graph (BG)* $\mathcal{B} = (V, E)$ is an undirected graph without loops with a set of nodes $V \subseteq CB$ and a set of edges

$$E \subseteq \{(c_i, c_j) | c_i, c_j \in V \text{ and } i \neq j\}, \tag{1}$$

where, by construction, each edge from E connects only cases with differing solutions. This means, for each $(c_i, c_j) \in E$ it holds

$$sim_s(c_i.s, c_j.s) < 1 - \varepsilon \tag{2}$$

where $\varepsilon \in [0, 1]$ is a threshold that defines when two solutions are considered to be different. The definition given so far and the relations in Formula 1 and 2 are not finalized, since Eq. 1 gives just a subset specification. We concretize this specification in the next paragraphs, emphasizing that the graphs we are creating will be a sparse representation of the data and contain a tiny fraction of the edges that would be allowed to be in E according to Eqs. 1 and 2.

3.1.2 Retrieval and Reuse

Given a query $q \in \mathcal{P}$ and a boundary graph $\mathcal{B} = (V, E)$, the retrieve algorithm moves *repeatedly* through the graph structure, calculating the similarity between q and the current node $c \in V$ as well as between q and the neighbors of c, i.e. for all $v \in V$ for which an edge $(c, v) \in E$ exists. It successively "moves" onwards to the node with the highest similarity to q until some maximum $c^\star$ has been reached, which means that $sim_p(q, c^\star) \geq sim_p(q, c) \forall (c^\star, c) \in E$.

Importantly, this procedure is repeated for r times, where the starting node is selected randomly from V each time. Hence, r determines the number of random retrieval starting points from which the similarity-guided search is initiated. Consequently, as retrieval result a vector $\mathcal{N}_q^r = (n_1, \ldots, n_r)$ of r estimated nearest neighbors is obtained.

The delineated retrieve step (function BG_Retrieve in Algorithm 1) is embedded into the superjacent function BG_Predict for boundary graph-based prediction which performs both, the retrieve and revise step of the classic CBR cycle. The mentioned vector of r nearest neighbor estimates are combined to form an overall prediction $\mathcal{R}(q)$ using some amalgamation function $\mathcal{A}$, such that

$$\mathcal{R}(q) = \mathcal{A}(\mathcal{N}_q^r) = \mathcal{A}((n_1, \ldots, n_r)). \tag{3}$$

For classification we can use a simple majority vote

$$\mathcal{A}((n_1, \ldots, n_r)) \in \arg\max_{l \in \mathcal{S}} |\{n_j | n_j.s = l, j = 1, \ldots, r\}| \tag{4}$$

BG_Predict(q, $\mathcal{B}$, r)
Input: query $q \in \mathcal{P}$,
boundary graph $\mathcal{B} = (V, E)$,
number r of random retrieval restarts,
amalgamation function $\mathcal{A}$
Output: BG-based prediction $\mathcal{R}(q)$
1: *// retrieve step*
2: $\mathcal{N}_q^r \leftarrow$ BG_Retrieve$(q, \mathcal{B}, r)$
3: *// reuse step (cf. Eq. 4-6)*
4: $\mathcal{R}(q) \leftarrow \mathcal{A}(\mathcal{N}_q^r)$
5: **return** $\mathcal{R}(q)$

BG_Retrieve(q, $\mathcal{B}$, r)
Input: query $q \in \mathcal{P}$,
boundary graph $\mathcal{B} = (V, E)$ with $V \neq \emptyset$,
number r of random retrieval restarts
Output: r-dimensional vector $\mathcal{N}_q^r$ of potential nearest neighbors
1: $\mathcal{N}_q^r \leftarrow$ r-dimensional vector
2: **for** $i = 1$ **to** r **do**
3: $c^\star \leftarrow$ random node from V
4: $stop \leftarrow false$
5: **while** $stop = false$ **do**
6: $c \leftarrow \arg\max_{v \in V \text{ s.t. } (c^\star, v) \in E} sim_p(q, v)$
7: **if** $sim_p(q, c) > sim_p(q, c^\star)$
8: **then** $c^\star \leftarrow c$
9: **else** $stop \leftarrow true$
10: $\mathcal{N}_q^r[i] \leftarrow c^\star$
11: **return** $\mathcal{N}_q^r$

Algorithm 1: Boundary Graph-Based Prediction and Retrieval

or a similarity-weighted voting scheme, like

$$\mathcal{A}((n_1, \ldots, n_r)) \in \arg\max_{l \in \mathcal{S}} \sum_{j=1}^{r} \begin{cases} sim_p(q, n_j) & \text{if } n_j.s = l \\ 0 & \text{else} \end{cases}. \tag{5}$$

In a similar manner, for regression tasks the estimated value becomes

$$\mathcal{A}((n_1, \ldots, n_r)) = \frac{\sum_{j=1}^{r} sim_p(q, n_j) \cdot n_j.s}{\sum_{j=1}^{r} sim_p(q, n_j)}. \tag{6}$$

A pseudo-code summary of the entire case-based retrieval and revise approach utilizing a boundary graph as index structure is given in Algorithm 1.

3.1.3 Graph Construction

We assume that the cases c_1 to c_n from the case base CB are presented to the boundary graph construction algorithm successively. Given a single training case c_i, the algorithm first queries the boundary graph $\mathcal{B}_{i-1} = (V_{i-1}, E_{i-1})$ which has been trained for the preceding $i-1$ cases, yielding a vector $\mathcal{N}_{c_i}^r = (n_1, \ldots, n_r)$ of r possible nearest neighbors. The algorithm then iterates over these n_j $(j = 1, \ldots, r)$ and, if $sim_s(c_i.s, n_j.s) < 1 - \varepsilon$ (i.e. n_j does "not solve" c_i, which for classification tasks boils down to $c_i.s \neq n_j.s$), then c_i is added as a new node to V_{i-1} and a (bidirectional) edge (c_i, n_j) is added to E_{i-1}. The resulting, extended boundary graph is denoted as $\mathcal{B}_i$. To sum up, training cases are added as nodes to the graph (including connecting edge), if the algorithm stochastically discovers a random retrieval starting point for which the currently existing boundary graph's prediction would be wrong and where, hence, a correction is needed.

Again, pseudo-code for constructively building up a boundary graph for a sequence of training cases is provided in Algorithm 2, denoted as BG_Construct. Note that the algorithm can be easily deployed in an online setting where new cases arrive during runtime by simply calling the BG_Retain function.

The left part of Fig. 1 is meant as an attempt to visualize an exemplary boundary graph for a synthetic two-dimensional, linearly separable two-class problem. Using 80 randomly created training cases, the BG_Construct algorithm created a boundary graph of 19 nodes and 40 edges. Using 20 independently sampled test cases, the resulting graph-based classifier yields an average classification error of 9.42% for 1000 random repetitions of the experiment.

BG_Construct(CB, r)
Input: case base $CB = \{c_1, \ldots, c_n\}$, number r of random retrieval restarts
Output: boundary tree $\mathcal{B}$
1: $\mathcal{B} \leftarrow (\emptyset, \emptyset)$
2: *// loop over all cases*
3: **for** $i = 1$ **to** n **do**
4: $\quad \mathcal{B} \leftarrow$ BG_Retain$(\mathcal{B}, c_i, r)$
5: **return** $\mathcal{B}$

BG_Retain($\mathcal{B}$, c, r)
Input: single (new) case c, boundary graph $\mathcal{B} = (V, E)$, number r of random retrieval restarts
Requires (global variables): amalgamation function $\mathcal{A}$, solution discrimination threshold ε
Output: (possibly extended) boundary graph $\mathcal{B}$
1: **if** $V = \emptyset$ **then** $V \leftarrow V \cup c$
2: **else**
3: $\quad \mathcal{N}_c^r \leftarrow$ BG_Retrieve$(c, \mathcal{B}, r)$
4: $\quad$ **for** $i = 1$ **to** r **do**
5: $\quad\quad \sigma \leftarrow sim_s(c.s, \mathcal{N}_c^r[i].s)$
6: $\quad\quad$ **if** $\sigma < 1 - \varepsilon$ **then**
7: $\quad\quad\quad V \leftarrow V \cup c$, $E \leftarrow E \cup (c, \mathcal{N}_c^r[i])$
8: (**return** (V, E)):

Algorithm 2: Construction of and Retain Procedure for Boundary Graphs

3.2 Labeled Boundary Multigraphs

We observed that increasing the value of the parameter r for the number of random retrieval restart improves the prediction accuracy and the stability of predictions. However, just like IB2 (cf. Sect. 2), the presented approach suffers from a dependency of the resulting boundary graph on the order in which the cases are presented to the retain procedure. The boundary forest algorithm (cf. Sect. 2) mitigates this dependency by forming an ensemble of (tree-based) classifiers. This is a straightforward idea whose impact we will also analyze for the IB2 algorithm below. At this point, however, we take inspiration from these ideas to extend our boundary graph algorithm such that it becomes more robust with respect to the order of the training data, while the computational overhead is acceptable and a single holistic retrieval index structure is retained.

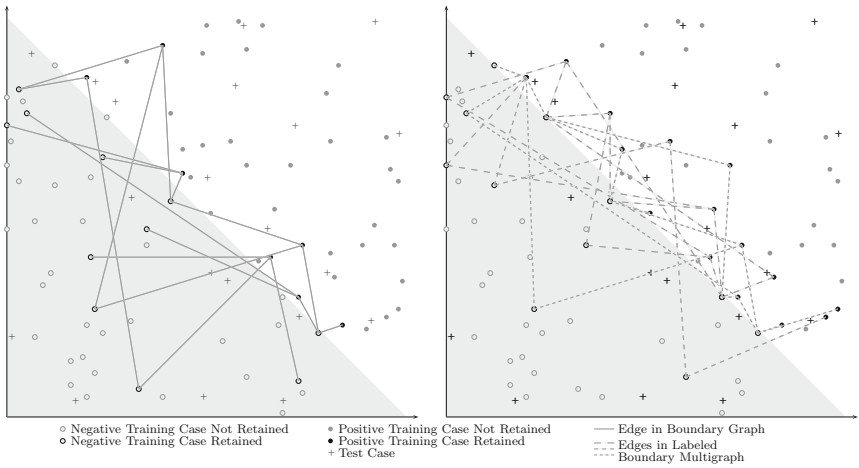


Fig. 1. Left: Visualization of an Examplary Boundary Graph: From the 80 training cases, 19 are included in the graph's set of vertices (11 from the negative class, 8 from the positive one), when trained using $r = 9$. Among that set of nodes, there are 88 possible edges that would cross the decision boundary. Out of those, 40 are included in the graph's set of edges. Right: Examplary Labeled Boundary Multigraph trained on the same data set for $\lambda = 3$ different labels and, accordingly, using $r_t := \frac{r}{\lambda} = \frac{9}{3} = 3$. In total, 27 nodes are included in the multigraph's set of vertices which are interconnected using 71 edges (32/8/31 for the three labels).

The core idea is to use uniquely identifiable labeled edges in the graph structure. This means we allow for multiple edges between vertices (which turns the graph into a multigraph), but make them distinguishable by attaching a label out of a set of λ labels to each edge. This promotes a more efficient dispersion of the retrieval knowledge across the graph structure. Then, instead of performing r random retrieval restarts (as we did before), the algorithm now (on average) performs $r_t := r/\lambda$ such random retrieval restarts per edge label. In so doing, the total amount of inner-graph retrieval computations stays roughly the same.

In the following, we first introduce the necessary notation, then present the retain part and, afterwards, conclude with the retrieve/revise procedures.

3.2.1 Notation

We define a *Labeled Boundary Multigraph (LBM)* $\tilde{\mathcal{B}} = (\tilde{V}, \tilde{E}, \tilde{L})$ to be a labeled undirected multigraph [2] with $\tilde{V}$ a set of vertices, $\tilde{E}$ a multiset of bidirectional edges, and $\tilde{L} : \tilde{E} \to \mathcal{L}$ a labeling function mapping to each edge a label out of a finite set of λ unique labels $\mathcal{L} = (l_1, \ldots, l_\lambda)$. We use the notation $\tilde{\mathcal{B}}|_l$ to indicate the submultigraph of $\tilde{\mathcal{B}} = (\tilde{V}, \tilde{E}, \tilde{L})$ for some specific edge label l. Note that, due to the restriction to edge label l, $\tilde{\mathcal{B}}|_l$ represents a boundary graph, no longer a

labeled boundary multigraph. Formally,

$$\tilde{\mathcal{B}}|_l = (V_l, E_l) \text{ where } E_l = \{e \in \tilde{E} | \tilde{L}(e) = l\} \text{ and} \\ V_l = \{v \in \tilde{V} | \exists (x,y) \in \tilde{E} \text{ with } \tilde{L}((x,y)) = l \text{ and } x = v \text{ or } y = v\} \tag{7}$$

Figure 1 (right) conveys an impression of a labeled boundary multigraph for a toy domain. Most importantly, the multigraph is factored by the edge labels and, retrieval and retain decisions do always take place in a label-specific manner.

LBM_RETAIN($\tilde{\mathcal{B}}$, CB, r)
Input: labeled boundary multigraph $\tilde{\mathcal{B}} = (\tilde{V}, \tilde{E}, \tilde{L})$, case base $CB = \{c_1, \ldots, c_n\}$, number r of random retrieval start points
Output: (possibly extended) labeled boundary multigraph $\tilde{\mathcal{B}}$

```
1:  repeat λ times  // λ = |L|
2:    CB ← shuffle cases in CB
3:    for i = 1 to n do
4:      l ← random edge label from L = {1, ..., λ}
5:      (V_l, E_l) ← B̃|_l  // induced subgraph for label l, cf. Equation 7
6:      (V_l', E_l') ← BG_RETAIN((V_l, E_l), c_i, r/λ)
7:      forall e ∈ Ẽ ∪ E_l' do if e ∈ E_l' then L̃'(e) = l else L̃'(e) = L̃(e)
8:      B̃ ← (Ṽ ∪ V_l', Ẽ ∪ E_l', L̃')
9:  return B̃
```

Algorithm 3: Construction of Labeled Boundary Multigraphs

3.2.2 Multigraph Construction

The LBM_RETAIN algorithm, provided in Algorithm 3, takes as input the number r of random retrieval restarts, a set of cases $CB = \{c_1, \ldots, c_n\}$ as well as a (possibly empty) labeled boundary multigraph $\tilde{\mathcal{B}}$ which it extends and returns. It thus works for both scenarios, either being given a set of training cases and starting out with an empty graph or in an online setting where a non-empty labeled multigraph is extended for a single or a few further training case(s).

When considering the next training case c_i, the algorithm first selects a random edge label l. If the algorithm later finds that this case should be added to the graph, then it will be connected to other nodes by edges that are labeled with l. Accordingly, line 5 determines the induced boundary subgraph for label l (cf. Eq. 7), consisting of edges with label l exclusively and of vertices that are adjacent to those edges. For this subgraph $\tilde{\mathcal{B}}|_l = (V_l, E_l)$ the BG_RETAIN algorithm is invoked, handing over the current case c_i plus the number of random retrieval restarts to be made during that retain step. Note that the latter number is reduced by a factor of λ (i.e. $\frac{r}{\lambda}$ is handed over in line 6) which is balancing the fact that the loop over all training cases $(c_1, \ldots, c_n)$ is repeated for as many times as we do have labels (i.e. for λ times, repeat loop from lines 1 to 8). The result of the BG_RETAIN call on the subgraph is finally incorporated into the

labeled boundary multigraph $\tilde{\mathcal{B}}$ (lines 7 and 8) before the algorithm continues with the next iteration. Note that each case is handled for λ times (each time using a randomly selected label) and might, thus, be connected with other nodes by a number of possibly differently labeled edges. Moreover, the contents of CB is shuffled at the beginning of each of the λ outer loop repetitions in order to mitigate against the dependency on the presentation order of the instances.

3.2.3 Multigraph Retrieval

Retrieval in an LBM $\tilde{\mathcal{B}}$ traverses the graph structure – searching greedily for more similar cases to the query q, jumping to neighboring vertices, if their similarity to q is higher than the similarity q and the current vertex – for a total of r times, starting at a random node v each time. Moreover, each of these traversals sticks to using edges with a specific label l exclusively. So, the r repetitions are equally distributed among the subsets of $\tilde{E}$ that contain edges with label l only. We point to the fact that lines 5 to 9 in the LBM_Retrieve procedure (Alg. 4) could be rewritten using a call to BG_Retrieve on $\tilde{\mathcal{B}}|_l$ with $\frac{r}{\lambda}$ random retrieval restarts, but that we decided for the more lengthy write-up for better comprehension.

As a result, an r-dimensional vector of estimated nearest neighbors is formed which, finally, can be utilized by some prediction function to form an overall estimate (reuse step) using some amalgamation function. To this end, we may alter the function BG_Predict (Algorithm 1) to LBM_Predict with the only difference that it calls LBM_Retrieve instead of BG_Retrieve in line 2.

As mentioned, the right part of Fig. 1 shows an exemplary labeled boundary multigraph (for $\lambda = 3$ labels) for the same toy problem as in Sect. 3.1.3. The increase in number of vertices and edges stored brings (while staying computationally at nearly the same level) a reduction of the classification error by almost 17% compared to the basic boundary graph approach: Using 20 independently sampled test cases, the resulting boundary graph-based classifier yields an average classification error of 7.87% (for 1000 random repetitions of the test).

LBM_Retrieve(q, $\tilde{\mathcal{B}}$, r)
Input: query $q \in \mathcal{P}$, LBM $\tilde{\mathcal{B}} = (\tilde{V}, \tilde{E}, \tilde{L})$, number r of random retrieval restarts
Output: r-dimensional vector $\mathcal{N}_q^r$ of potential nearest neighbors
1: $\mathcal{N}_r^q \leftarrow r$-dimensional vector
2: **for** $l = 1$ **to** λ **do**
3: $(V_l, E_l) \leftarrow \tilde{\mathcal{B}}|_l$
4: **for** $i = 1$ **to** $\frac{r}{\lambda}$ **do**
5: $v \leftarrow$ random node from V_l, $stop \leftarrow false$, $c^* \leftarrow v$
6: **while** $stop = false$ **do**
7: $c^* \leftarrow \arg\max_{(v,c)\in E_l} sim_p(q, c)$
8: **if** $sim_p(q, c^*) > sim_p(q, v)$ **then** $v \leftarrow c^*$ **else** $stop \leftarrow true$
9: $\mathcal{N}_r^q[(l-1)r/\lambda + i] \leftarrow v$
10: **return** $\mathcal{N}_q^r$

Algorithm 4: Labeled Boundary Multigraph Retrieval

3.3 Discussion

For both algorithms described, the boundary graph as well as the labeled multigraph extension, we find empirically that the retrieval time using such a graph as index structure scales either as a power law βn^{α} in the amount of training cases n with a power value $\alpha < 1$ or logarithmically For the most comprehensive data set we analyzed we found the complexity to be in $O(\log n)$. Accordingly, training time scales as $n^{1+\alpha}$ or $n \log n$, respectively, because each retain step essentially includes a retrieve step, rendering the complexity of training to be a loop of n repetitions wrapped around the retrieval procedure.

The boundary graph approach has the favorable characteristic to be an *anytime retrieval algorithm*. By handling the parameter r of random retrieval starting points within the graph different during training (r_t) and the application (r_a) of the learned graph, i.e. separating $r_t := \frac{r}{\lambda}$ from r_a, one can gain a significant performance boost by letting $r_a > r_t$, given that a sufficient amount of time is available for the system to respond to a query. This is a desirable property in real-time and online application settings since the accuracy of the retrieval grows with r_a as we will show in the next section.

4 Empirical Evaluation

Our experimental investigations were driven by two questions: What final performance can be achieved using labeled boundary multigraphs as index structure in case-based retrieval? And at what costs can these results be obtained? We are going to address these questions in the next subsections.

4.1 Experimental Set-Up

Our experiments focus on classification problems. Hence, no specific value for ε must be set as two cases' solutions are considered different, if their class labels do not match. We selected a variety of established classification benchmark datasets from the UCI Machine Learning Repository [8] with varying amounts of case data, classes, and numbers and types of features. Additionally, we used a larger (500 k instances) nine-dimensional dataset from the field of robotic soccer simulation where the goal is to predict the type of an opponent agent's next action (e.g. kick or dash or turn, cf. [9]) In all our experiments we employ a knowledge-poor default similarity measure according to the local-global principle [3] with uniform attribute weights, identity similarity matrices for discrete features and linear difference-based similarity functions for numeric ones. We are fully aware that the classification accuracy might be improved significantly, when applying knowledge-intensive, domain-specific similarity measures or even learned ones [10]. For each domain, we split the available case data into two randomly created disjoint sets, using the first (80%) one for construction the graph-based index structure and the second (20%) for applying (testing) it. To account for the stochastic nature of our proposed algorithms, all experiments were repeated 100 times, forming average values of performance indicators.

4.2 Classical Benchmark Data Sets

In this series of experiments, we compared our proposed methods (BG and LBM) to several other retrieval mechanisms. As a baseline, we employed the k-nearest neighbor classifier with $k = 1$, yielding a classification error μ of 26.07% averaged over the 24 datasets[2] (cf. Table 1). Searching for the best value of k for that set of domains in a brute force manner, we found that $k^* = 13$ yields best results ($\mu = 21.07\%$). We proceeded in the same way for the boundary tree/forest approach (cf. Sect. 2). Using forests of 10 and 50 trees (as the authors in [16] propose), a classification error of 23.32 and 21.32%, respectively, is obtained, while the best one found (brute force search) is 20.83% for a forest of 325 trees.

With respect to storage requirements it is worth noting that the only algorithm that yields a substantial case base compactification γ (last line in Table 1) is the classical IB2 algorithm which retains 34.6% of the given case data on average. To this end, we also tested a straightforward extension of IB2, called $IB2_e$, which runs IB2 on e shuffled versions of the training data, forms an ensemble of e compactified case bases and classifies cases by applying a majority vote as in Eq. 4. As expected, increasing e clearly impairs the achieved level of case base compactification (γ is now determined on the basis of the union of the e IB2 instances' case bases), trading this off for lower classification errors with the lowest one found (again, in a brute force manner) for $e^* = 94$.

Using a single boundary graph as index structure (columns BG_r), that was trained using $r = 10$ and 50 random retrieval restarts (these numbers have been chosen in accordance to the boundary forests), approximately 25% of the training cases need not to be stored. However, with an average classification error of 24.10 and 23.52%, respectively, the approach does not surpass the boundary forests.

Next, we investigated the labeled boundary multigraph approach where, for reasons of comparison to the previous algorithms, we set $r = 50$. We first report the setting where these 50 random retrieval restarts are distributed over $\lambda = 7$ labels during training *and* application, i.e. $r_t = r_a = \frac{r}{\lambda} = \frac{50}{7} \approx 7$. Apparently, μ drops to 21.08%, outperforming all other approaches considered so far. On the supposition that during the application of the system was more time or computational power available, one might consider increasing r_a beyond r_t. The impact of that computational extra effort can be read from the neighboring table column where μ is reported to drop to 20.49%. We emphasize that this is a unique characteristic of the BG/LBM approach, i.e. a feature that the other algorithm do not possess, and that further increasing r_a would reduce the error even more. This particular effect of varying computational budgets during retrieval in the application phase of the system is more thoroughly analyzed in Fig. 2.

The third and second from last columns indicate that doubling the number of labels while retaining the overall effort at the same level (i.e. $\lambda = 13$ and $r_t = 4$ with $\lambda r_t = 13 \cdot 4 \approx 7 \cdot 7$) improves the average performance of the

[2] A-Balance, B-BanknoteAuth, C-Cancer, D-Car, E-Contraceptive, F-Ecoli, G-Flare, H-Glass, I-Haberman, J-Hayes, K-Heart, L-Iris, M-MammogrMass, N-Monks, O-Pima, P-QualBankruptcy, Q-TeachAssistEval, R-TicTacToe, S-Transfusion, T-UserKnowledge, U-VertebralCol, V-Wholesale, W-Wine, X-Yeast.

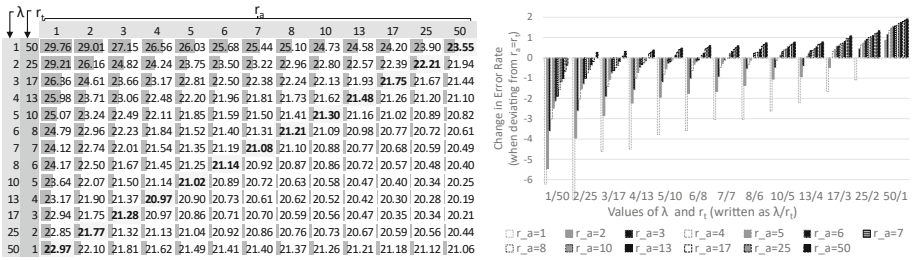
Table 1. For all considered retrieval approaches and for each of the considered domains the classification error in percent (averaged over 100 experiment repetitions) is provided, plus an average μ over all domains. Additionally, in the last line the average case base compactification γ is reported. For each domain, the two top-performing algorithms are highlighted. The first column contains domain identifiers (see footnote for plain text names).

Dom.	k-NN		BF_t			$IB2_e$			BG_r		$LBM^{\lambda=7}_{r_t,r_a}$		$LBM^{\lambda=13}_{r_t,r_a}$		$LBM^{\lambda^*=37}$
	k=1	k^*=13	t=10	t=50	t^*=325	e=1	e=13	e^*=94	r=10	r=50	r_t=7 r_a=7	r_t=7 r_a=50	r_t=4 r_a=4	r_t=4 r_a=50	r_t=6 r_a=50
A	35.47	17.12	26.11	20.47	19.13	48.59	47.05	45.75	24.43	25.18	19.54	17.24	18.39	**14.85**	**15.69**
B	0.16	0.30	0.12	0.12	0.15	0.57	0.15	0.17	0.20	0.14	0.12	**0.10**	0.11	**0.09**	0.12
C	31.48	**24.84**	29.76	27.94	26.91	32.85	29.64	28.91	30.16	31.58	27.02	26.58	25.62	**25.27**	25.29
D	22.78	9.18	10.06	**5.21**	**3.93**	22.94	15.14	14.57	12.84	5.76	8.61	6.51	11.99	9.13	6.59
E	56.34	52.00	54.87	53.06	52.67	57.18	54.69	53.90	54.57	54.81	52.43	51.24	51.84	**50.24**	**51.11**
F	20.08	16.12	20.31	19.98	19.98	28.07	23.74	23.13	20.84	21.99	16.16	16.03	15.57	**15.37**	**15.24**
G	28.86	17.52	21.51	21.37	21.21	40.65	36.60	33.82	18.36	21.16	17.25	17.17	16.99	**16.85**	**17.12**
H	28.66	33.52	27.60	27.06	**26.54**	32.61	29.62	29.41	31.07	30.02	28.19	27.98	27.90	27.10	**26.40**
I	35.34	**26.20**	35.77	35.29	35.17	40.00	37.63	38.12	35.56	36.66	33.93	34.00	**32.66**	32.79	33.39
J	33.15	43.28	27.58	**25.10**	**24.28**	32.21	29.78	29.29	32.50	31.00	33.50	34.54	34.19	35.35	35.19
K	21.27	**16.89**	23.41	20.61	20.02	28.81	24.71	23.77	23.87	24.72	19.22	18.76	18.39	18.39	**18.31**
L	5.93	**4.88**	6.43	5.72	5.79	8.75	6.70	6.46	8.13	8.87	5.93	5.90	**5.63**	**5.63**	5.70
M	26.93	20.10	23.93	22.35	21.82	32.57	28.61	27.42	22.77	23.64	20.22	**19.29**	20.61	19.49	**19.14**
N	27.37	6.11	3.97	**0.03**	**0.00**	26.57	15.74	13.03	9.49	1.03	2.64	0.47	5.56	1.06	0.22
O	30.46	26.58	31.24	28.97	28.14	36.16	35.02	34.00	30.68	32.47	27.40	27.10	26.49	**25.87**	**26.38**
P	0.20	0.40	0.60	0.67	0.63	1.88	0.42	0.28	0.52	0.20	0.26	**0.18**	0.24	0.20	**0.10**
Q	**38.42**	52.88	40.43	40.44	40.47	42.59	39.00	**38.45**	42.83	40.43	42.30	41.50	42.80	42.43	39.87
R	18.88	2.33	7.63	**1.82**	**1.07**	18.70	15.77	16.02	12.19	2.46	7.08	4.92	10.61	9.42	4.61
S	31.24	**22.30**	29.28	29.14	29.03	36.38	33.56	33.51	27.83	30.98	24.70	24.46	23.46	**23.41**	24.08
T	19.95	18.14	16.08	13.62	13.48	25.78	18.75	18.24	18.61	17.33	12.92	12.37	12.29	**11.61**	**11.51**
U	24.58	**21.26**	25.26	23.28	22.53	29.75	26.07	25.59	25.32	26.19	22.94	22.95	22.32	**21.74**	22.37
V	46.65	**29.24**	45.40	43.04	42.91	51.65	49.65	49.19	43.86	46.31	38.00	37.47	34.70	**34.56**	35.75
W	3.99	2.72	2.66	2.02	**1.60**	7.52	2.69	2.41	4.03	3.51	2.17	2.14	2.20	**1.77**	1.86
X	47.57	**41.85**	47.51	44.43	43.92	53.80	51.52	51.60	47.77	47.98	43.38	42.93	42.62	**41.86**	42.17
μ	26.49	21.07	23.23	21.32	20.89	30.69	27.18	26.54	24.10	23.52	21.08	20.49	20.97	**20.19**	**19.93**
γ	100.0	100.0	79.5	92.3	96.4	**34.6**	79.3	93.2	**72.4**	74.9	92.4	92.4	94.8	94.8	96.4

obtained index structure even further. For completeness, we also included the best possible LBM (using a set of $\lambda = 37$ edge labels, $r_t = 6$ random retrieval restarts during training and $r_a = 50$ during application) that we found by brute force search (therefore, again marked with a star) whose average classification error is 19.93%.

4.3 Scaling Analysis

When introducing novel algorithms, their space and time requirements are of high interest. In our case, space complexity boils down to the number of cases retained which can be equated with the number of nodes in a BG/LBM. Storing edges represents a negligible overhead as we realize this by storing pointers to the original case data. We measure the algorithms' time complexity in terms of



λ \ r_a	1	2	3	4	5	6	7	8	10	13	17	25	50
50	29.76	29.01	27.15	26.56	26.03	25.68	25.44	25.10	24.73	24.58	24.20	23.90	**23.55**
25	29.21	26.16	24.82	24.24	23.75	23.50	23.22	22.96	22.80	22.57	22.39	**22.21**	21.94
17	26.36	24.61	23.66	23.17	22.81	22.50	22.38	22.24	22.13	21.93	**21.75**	21.67	21.44
13	25.98	23.71	23.06	22.48	22.20	21.96	21.81	21.73	21.62	**21.48**	21.26	21.20	21.10
10	25.07	23.24	22.49	22.11	21.85	21.59	21.50	21.41	**21.30**	21.16	21.02	20.89	20.82
8	24.79	22.96	22.23	21.84	21.52	21.40	21.31	**21.21**	21.09	20.98	20.77	20.72	20.61
7	24.12	22.74	22.01	21.54	21.35	21.19	**21.08**	21.10	20.88	20.77	20.68	20.59	20.49
6	24.17	22.50	21.67	21.45	21.25	**21.14**	20.92	20.87	20.86	20.72	20.57	20.48	20.40
5	23.64	22.07	21.50	21.14	**21.02**	20.89	20.72	20.63	20.58	20.47	20.40	20.34	20.25
4	23.17	21.90	21.37	**20.97**	20.90	20.73	20.61	20.62	20.52	20.42	20.30	20.28	20.19
3	22.94	21.75	**21.28**	20.97	20.86	20.71	20.70	20.59	20.56	20.47	20.35	20.34	20.21
2	22.85	**21.77**	21.32	21.13	21.04	20.92	20.86	20.76	20.73	20.67	20.59	20.56	20.44
1	**22.97**	22.10	21.81	21.62	21.49	21.41	21.40	21.37	21.26	21.21	21.18	21.12	21.06

Fig. 2. Comparison of LBMs with Equal Computational Construction Effort: The product of λ and r_t is always (nearly) the same, i.e. $r = \lambda r_t \approx 50$. The left matrix shows classification errors averaged over all 24 UCI domains, subject to differing values of r_a (columns). While the diagonal is highlighted as a default setting with $r_t = r_a$, the diagram on the right visualizes the gains/losses in accuracy that are obtained when deviating from the default and varying the values of r_a between 1 to 50, subject to the different combinations of λ and r_t (all with $\lambda r_t \approx 50$).

similarity calculations as a function of the number of training cases n, as this is the computational bottleneck for case-based retrieval algorithms.

Part a) in Fig. 3 visualizes the average number of required similarity calculations for a test query for the 24 UCI domains we selected (trained/tested for $\lambda = 1$ and $r_t = r_a = 15$). The best power law fit of the data is nearly identical to the best logarithmic fit (in terms of the coefficient of determination R^2) leaving us somewhat undecided whether the computational effort scales with $O(n^{\frac{3}{5}})$ or logarithmically, especially since there is a lot of variation included due to the varying characteristics of the different domains.

By contrast, part b) visualizes the scaling behavior of the boundary graph approach ($\lambda = 1$, $r_t = r_a = 50$) for the mentioned robotic soccer simulation domain. Here, the number of required similarity calculations per test query seems to scale clearly logarithmically in the number of the training cases (i.e. $O(\log n)$ and $O(n \log n)$ for the overall training process).

The behavior plotted in part b) of Fig. 3 can be fully appreciated only, when relating this to the achieved case base compactification γ which can be read from part c). Apparently, the number of cases stored grows linearly with n (note the log scale abscissa) at a level of $\gamma \approx 0.6$, such that, e.g., for a training set of $n = 51200$ cases on average only 4548 similarity calculations (4.9% of n) must be performed. For $n = 409600$ cases, we observe 251 k stored cases ($\gamma = 61\%$) and 7278 required similarity calculations (1.8%) for a single test query on average. This is in strong contrast to the other algorithms where many more similarity calculations are required during retrieval. Essentially, this is achieved by the fact that the number of edges per vertex grows logarithmically with n (see dashed line in the plot of part c) in Fig. 3).

Finally, part d) highlights the relation between the training set size and the final performance of the constructed graph-based index structure in the robotic soccer domain. While it is to be expected that the classification error on the test

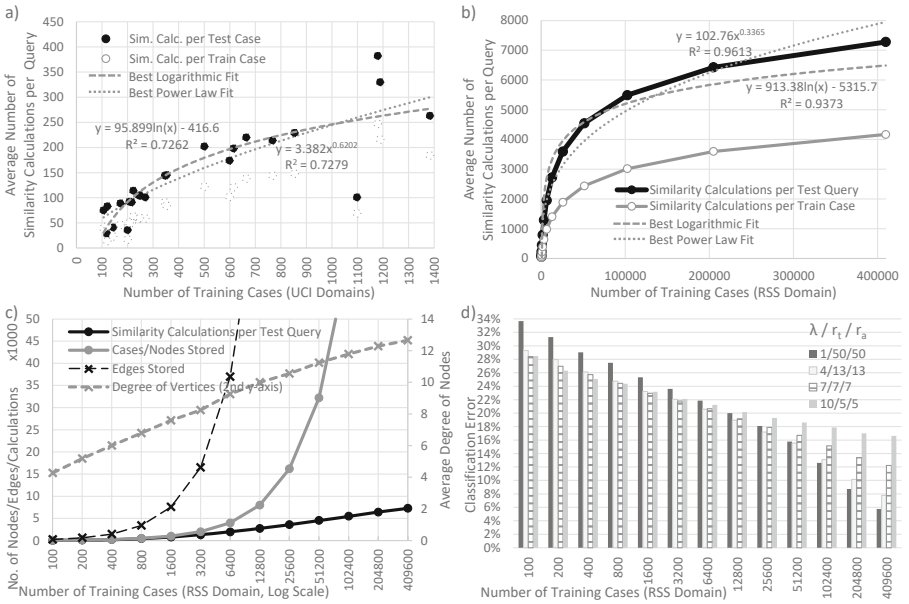


Fig. 3. Scaling Behavior of BGs/LBM: See the text for details.

set decreases with the number of training cases, it is interesting to observe that for smaller training sets a distribution among a larger number λ of label layers is less beneficial. This fact is revealed for $n \geq 100\,\text{k}$ where the simpler structured boundary graph (i.e. with $\lambda = 1$) supersedes other variations and finally achieves an excellent low classification error of less than 6%.

It is worth noting that, under these settings ($n = 409.6\,\text{k}$, $\lambda = 1$, $r_t = r_a = 50$), a case-based classification can be made within 18.6 ms of time on a single core of a contemporary 3 GHz CPU which adheres to the real-time constraints imposed by the soccer simulation environment where an agent must execute each action within 100 ms. With the already discussed additional opportunity to increase/decrease r_a online depending on necessary other computations done by the agent during decision-making, we obtain a reliable and performant module for case-based opponent modeling. Further analyses for that domain and of our algorithms' scaling properties can be found in a separate paper [11].

5 Conclusion

Retrieval efficiency in Case-Based Reasoning remains an important topic even after decades of research. In this paper, we have introduced the concept of boundary graphs which we suggest to use as an efficient index structure during case-based retrieval. The core idea of these graph structures as well as of their labeled multigraph extensions that we develop on top of them is that their

edges do always connect cases with differing solutions. We have proposed appropriate retrieve and retain algorithms and evaluated them using a selection of benchmark classification problems as well as a robotic soccer dataset. While our methods are presented in a way that makes them applicable to various problem types, our empirical evaluations focused on classification tasks, leaving the analysis of other problem types (like regression etc.) for future work.

References

1. Aha, D., Kibler, D., Albert, M.: Instance-based learning algorithms. Mach. Learn. **6**, 37–66 (1991). https://doi.org/10.1007/BF00153759
2. Balakrishnan, V.: Theory and Problems of Graph Theory. McGraw, USA (1997)
3. Bergmann, R., Richter, M., Schmitt, S., Stahl, A., Vollrath, I.: Utility-oriented matching: a new research direction for case-based reasoning. In: Proceedings of the 9th German Workshop on Case-Based Reasoning (GWCBR) (2001)
4. Beygelzimer, A., Kakade, S., Langford, J.: Cover tree for nearest neighbor. In: Proceedings of the Twenty-Third International Conference on Machine Learning (ICML), pp. 97–104. ACM Press, Pittsburgh, USA (2006)
5. Brin, S.: Near neighbors search in large metric spaces. In: Proceedings of the Twenty-First International Conference on Very Large Data Bases (VLDB), pp. 574–584. Morgan Kaufmann, Zurich, Switzerland (1995)
6. Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE Trans. Inf. Theory **13**, 21–27 (1967)
7. Djebbar, A., Merouani, H.: Optimising retrieval phase in CBR through pearl and JLO algorithms. J. Adv. Intell. Paradigms **5**(3), 161–181 (2013)
8. Dua, D., Graff, C.: UCI repository (2017). http://archive.ics.uci.edu/ml
9. Gabel, T., Godehardt, E.: I know what you're doing: a case study on case-based opponent modeling and low-level action prediction. In: Workshop on Case-Based Agents at ICCBR (CBA 2015), pp. 13–22. Frankfurt, Germany (2015)
10. Gabel, T., Godehardt, E.: Top-down induction of similarity measures using similarity clouds. In: Hüllermeier, E., Minor, M. (eds.) ICCBR 2015. LNCS (LNAI), vol. 9343, pp. 149–164. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24586-7_11
11. Gabel, T., Sommer, F.: Instance-based opponent action prediction in soccer simulation using boundary graphs. In: Paetzel, M., Lau, N., Wanichanon, T., Eguchi, A. (eds.) RoboCup 2022: Robot Soccer World Cup XXVI. Springer, Bangkok (2022)
12. Gates, G.: The reduced nearest neighbor rule. IEEE Trans. Inf. Theory **18**(3), 431–433 (1972)
13. Leake, D.B., Wilson, D.C.: Categorizing case-base maintenance: dimensions and directions. In: Smyth, B., Cunningham, P. (eds.) EWCBR 1998. LNCS, vol. 1488, pp. 196–207. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0056333
14. Lejsek, H., Jonsson, B., Amsaleg, L.: NV-tree: nearest neighbors in the billion scale. In: Proceedings of the First ACM International Conference on Multimedia Retrieval (ICMR), pp. 57–64. ACM Press, Trento, Italy (2011)
15. Lenz, M.: Case retrieval nets as a model for building flexible information systems. Ph.D. thesis, Humboldt University of Berlin, Germany (1999)
16. Mathy, C., Derbinsky, N., Bento, J., Rosenthal, J., Yedidia, J.: The boundary forest algorithm for online supervised and unsupervised learning. In: Proceedings of the 29th AAAI Conference on AI, pp. 2864–2870. AAAI Press, Austin, USA (2015)

17. Richter, M.M., Weber, R.O.: Case-Based Reasoning. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40167-1
18. Smyth, B., McKenna, E.: Building compact competent case-bases. In: Althoff, K.-D., Bergmann, R., Branting, L.K. (eds.) ICCBR 1999. LNCS, vol. 1650, pp. 329–342. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48508-2_24
19. Smyth, B., McKenna, E.: Competence guided incremental footprint-based retrieval. Knowl.-Based Syst. **14**(3–4), 155–161 (2001)
20. Wess, S., Althoff, K.-D., Derwand, G.: Using k-d trees to improve the retrieval step in case-based reasoning. In: Wess, S., Althoff, K.-D., Richter, M.M. (eds.) EWCBR 1993. LNCS, vol. 837, pp. 167–181. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-58330-0_85
21. Wilson, D., Martinez, T.: Reduction techniques for instance-based learning algorithms. Mach. Learn. **38**(3), 257–286 (2000). https://doi.org/10.1023/A:1007626913721