Leveraging Plasticity in Incremental Decision Trees

Marco Heyden (✉)[1][0000−0003−4981−709X], Heitor Murilo Gomes[2][0000−0002−5276−637X], Edouard Fouché[1][0000−0003−0157−7648], Bernhard Pfahringer[3][0000−0002−3732−5787], and Klemens Böhm[1][0000−0002−1706−1913]

[1] Karlsruhe Institute of Technology, Karlsruhe, Germany
{firstname.lastname}@kit.edu
[2] Victoria University of Wellington, Wellington, New Zealand
heitor.gomes@vuw.ac.nz
[3] University of Waikato, Hamilton, New Zealand
bernhard@waikato.ac.nz

Abstract. Commonly used incremental decision trees for mining data streams include Hoeffding Trees (HT) and Extremely Fast Decision Trees (EFDT). EFDT exhibits faster learning than HT. However, due to its split revision procedure, EFDT suffers from sudden and unpredictable accuracy decreases caused by subtree pruning. To overcome this, we propose PLASTIC, an incremental decision tree that restructures the otherwise pruned subtree. This is possible due to *decision tree plasticity*: one can alter a tree's structure without affecting its predictions. We conduct extensive evaluations comparing PLASTIC with state-of-the-art methods on synthetic and real-world data streams. Our results show that PLASTIC improves EFDT's worst-case accuracy by up to 50% and outperforms the current state of the art on real-world data. We provide an open-source implementation of PLASTIC within the MOA framework for mining high-speed data streams.

Keywords: Data streams · Incremental learning · Decision trees

1 Introduction

Data streams are infinite sequences of observations that arrive over time. They are omnipresent in everyday life (think of ad-click prediction [22], spam classification [32], or activity recognition [18]) and in industrial applications (e.g., in machine failure prediction [41] or quality control [27]). Since data streams are unbounded, it is impossible to store all data for, say, creating a training set. Furthermore, data streams can change unpredictably, a phenomenon known as concept drift. Algorithms that learn from data streams should therefore operate incrementally, so as to incorporate new information over time [3].

Decision trees are a popular way to address classification tasks. Tree-based learners come with many benefits: They yield decisions that are interpretable, they can deal with numerical and categorical data, they are robust to outliers,

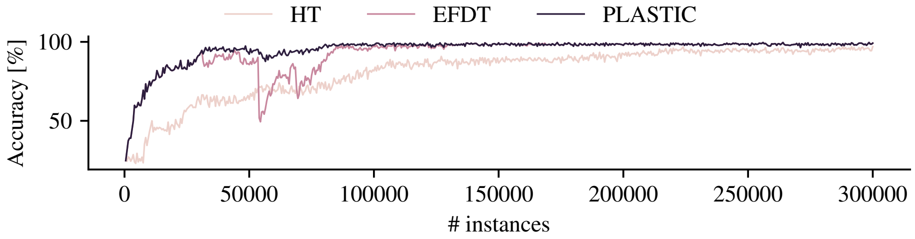

Fig. 1: Comparison of HT, EFDT, and PLASTIC using data generated by MOA's Random Tree Generator (10 binary attributes, 5 classes, default random seed).

and can scale to large data sets [8,24]. A long-standing line of research focuses on extending tree-based learners such that they can learn incrementally for the streaming setting. Hoeffding Trees (HT) and their statistically more efficient successor "Extremely Fast Decision Tree" (EFDT) are arguably the most well-known incremental tree-based learners [9,21]. Both approaches keep track of the data and label distribution at the individual nodes and split a node once they have identified a "good" split attribute with high probability. Their difference lies in the stricter leaf spitting condition of HT compared to EFDT.

HT expands a leaf if it has found the split with the highest information gain[4] with high probability. EFDT expands a leaf if it has identified a split with *greater-than-zero* information gain with high probability. As this can lead to suboptimal splits, EFDT frequently revises its split decisions and re-splits a node if it has found a better split attribute.

EFDT's split-and-revise strategy leads to faster learning compared to HT. However, when re-splitting occurs, EFDT prunes the subtree under the affected node. Depending on the size of the subtree, this process can lead to sudden and significant accuracy drops, as shown in Figure 1. Here, accuracy decreased by almost 50%, with recovery taking around 30,000 instances. Our evaluation in Section 5 demonstrates that this behavior is common rather than rare.

Deciding between HT and EFDT poses a dilemma: choosing between a learning method that is "slow but steady" and one that offers fast learning but may lead to sudden and unpredictable accuracy drops. Our algorithm, PLASTIC, offers a solution that is "fast and steady", effectively solving this dilemma.

Contributions. (1) Our main contribution is PLASTIC, an incremental decision tree algorithm for data streams. We introduce the notion of *decision tree plasticity*: restructuring a decision tree without affecting its predictions. With this, PLASTIC alleviates the negative effects of sudden and unexpected subtree pruning during split revision. We also propose a simple yet effective variant, PLASTIC-A, which is adaptive to concept drift. (2) We compare our proposals to state-of-the-art incremental decision tree algorithms on 9 synthetic and

[4] For the sake of clarity, we use the term information gain interchangeably with other splitting heuristics such as Gini index.

14 real-world data streams. Our results on synthetic data show that PLASTIC is able to avoid EFDT's sudden accuracy drops, leading to benefits of up to 50% accuracy. Real-world data validates our findings: non-adaptive PLASTIC remains highly competitive against even adaptive decision trees. However, our adaptive variant, PLASTIC-A, surpasses both, achieving accuracy gains of up to 10% on data streams with strong temporal dependencies. (3) We provide an open-source implementation[5] of PLASTIC within the popular MOA framework for mining high-speed data streams [5].

2 Setting

We consider the setting of data stream classification. A data stream S is an infinite sequence of observations $x_1, x_2, \ldots, x_t, \ldots$ that arrive over time, where t denotes the current time step. We assume that all $x_i \in S$ are from the same d-dimensional domain $\mathcal{X}$, which usually represents the real numbers, is categorical with a finite number of possible categories, or a combination of both. We call the j-th attribute a_j. For an instance x_i, its j-th attribute value is v_{ij}. If a_j is categorical, the number of its distinct categories is c_j. Each x_t has a label y_t that belongs to a finite set of categories $\mathcal{Y}$. A learner's goal is to learn a function f such that $y_i = f(x_i)$. We make the common assumption that x_t and y_t are revealed simultaneously, i.e., in each time step the learner can access the tuple (x_t, y_t) to construct f [4]. However, both the semi-supervised setting (only some y_t values are disclosed to the learner) and the delayed-labels scenario (y_t values are not immediately available) are promising directions for future research [14].

3 Preliminaries

A widely adopted decision tree algorithm for data streams is the Hoeffding Tree algorithm (HT) [9]. The algorithm, outlined in Alg. 1, builds a decision tree incrementally: At each leaf, HT keeps track of counters n_{ijk}, i.e., one counter for each attribute (a_i), value (v_j), class (y_k) combination.[6] Based on the counters, HT can make predictions and compute the information gain IG for each possible split. The algorithm splits a leaf once the IG difference between the best and second-best possible split exceeds a threshold $\epsilon = \sqrt{R^2 \ln(1/\delta)/2n}$. This threshold is based on Hoeffding's inequality and depends on the IG range, the number of observations n observed at the leaf, and the confidence level δ.

"Extremely Fast Decision Trees" (EFDT) [21], particularly related to our approach, improve HT's statistical efficiency by relaxing the condition under which they split a leaf. Compare Line 6 in Alg. 1 to Line 5 in Alg. 2: In comparison to HT, EFDT already splits a leaf if the IG of any attribute is significantly larger than 0. EFDT thus requires less data to build a tree, leading to better predictions

[5] https://github.com/heymarco/PLASTIC

[6] In the case of numerical attributes one typically relies on discretization, e.g., histograms, to maintain memory efficiency. Refer to [4] for additional information.

after fewer data points. The downside is that such eager splitting leads to many suboptimal splits. Thus, EFDT revises its former split decisions and re-splits a node if the split turns out to be suboptimal. In the process, the subtree below that node is lost, causing sudden and unexpected accuracy drops (recall Fig. 1).

Algorithm 1 Hoeffding Tree

```
procedure LEAF NODE
    for each new instance $x_i$ arriving at leaf $l$ do
        Use $x_i$ to update counters in $l$
        $a_1 \leftarrow$ attribute with the highest $IG$
        $a_2 \leftarrow$ attribute with the 2nd-highest $IG$
        if $IG(a_1) > IG(a_2)$ with probability $> 1 - \delta$ then
            Split $l$ at $a_1$
```

Algorithm 2 Extremely Fast Decision Tree

```
procedure LEAF NODE
    for each new instance $x_i$ arriving at leaf node $l$ do
        Use $x_i$ to update counters in $l$
        $a_1 \leftarrow$ attribute with the highest $IG$
        if $IG(a_1) > 0$ with probability $> 1 - \delta$ then    ▷ Relaxed splitting criterion
            Split $l$ at $a_1$
procedure INTERNAL NODE
    for each new instance $x_i$ arriving at internal node $s$ do
        Use $x_i$ to update counters in $s$
        $a_1 \leftarrow$ attribute with the highest $IG$
        $a_c \leftarrow$ current split attribute
        if $IG(a_1) > IG(a_c)$ with probability $> 1 - \delta$ then    ▷ Split revision
            Remove subtree below $s$
            Split $s$ at $a_1$
```

Over time, HT have been subject to further developments focusing on semi-supervised learning [40], vague data [15,10], stability [23], its statistical foundation [31,29,28,30], and prediction quality [13]. Other research has addressed time and memory consumption [6], or implemented adaptivity to concept drift [16,2,12,40,20]. Regarding prediction quality, [13] proposes to use classifiers, e.g., Naive Bayes, at the leaves of a tree. This approach is widely applicable, for example in HT, EFDT, and our own approach. An effective approach to achieve adaptivity to concept drift is to monitor accuracy at the nodes in the tree [2] and to build a "background tree" when accuracy decreases. The background tree replaces the current subtree once it is more accurate. The most recent "Extremely Fast Hoeffding Adaptive Tree" (EFHAT) [20] does this and combines it with

EFDT's relaxed initial splitting mechanism. EFHAT is the state of the art in adaptive incremental decision trees.

Some early Machine Learning research, ID5R [35] and ITI [36], has used subtree restructuring to build decision trees incrementally. However, these algorithms are unsuitable for data streams: (1) The algorithms are intended to build a tree from some given database. Hence, the approches assume data to follow the same distribution and be finite in volume. (2) Both approaches are slow, due to greedy tree revision (ID5R) and recursive restructuring of subtrees (ITI). (3) They also consume large amounts of memory, as every leaf has to store all its data points to facilitate future restructuring. PLASTIC in contrast relies on statistical testing to decide whether restructuring is beneficial and only stores the necessary information to perform future splits.

We compare PLASTIC to HT, the most common default choice, EFDT, our approach's closest relative and state of the art in stationary data streams, and EFHAT, the best-performing approach for data streams with concept drift.

4 PLASTIC

Our approach adopts EFDT's split-and-revise strategy: it splits a node early and revises the split when more data becomes available. Thus, both approaches share the same structure shown in Alg. 2. However, while split-revision may trigger subtree pruning in EFDT, it leads to subtree restructuring in PLASTIC. To give an intuition, Sec. 4.1 describes subtree restructuring for categorical attributes on a conceptual level. Sec. 4.2 introduces the algorithm on a more technical level and discusses adaptations for numeric and binary categorical splits.

4.1 Concept

PLASTIC exploits the fact that the order in which an instance traverses the nodes of a decision tree from root to leaf does not affect prediction.

Example 1 (Decision tree plasticity). Consider the left-most branch ■—●—○ from root (■) to leaf (○) in Fig. 2. Any instance with attribute values ■ = 0 and ● = 0 will arrive at ○. Hence, from the viewpoint of the leaf, ■—●—○ ≡ ●—■—○.

For a given branch, this means that one can freely re-arrange the order of nodes without affecting any predicted label. Similarly, by re-arranging the branches of a decision tree systematically, one can change its structure without affecting its predictions. We refer to this concept as *decision tree plasticity*.

Our idea is to leverage the concept of plasticity during split revision to facilitate the desired split without discarding the affected subtree. In the example, assume that split revision has identified ● as the currently best split for the root node in Fig. 2, instead of splitting at the current attribute ■.

In response, EFDT would prune the subtree and re-split the root. PLASTIC in turn restructures the subtree. It first creates a representation of the subtree in which its branches are disconnected from each other. PLASTIC then restructures each branch and reconnects them. Our algorithm has the following steps:

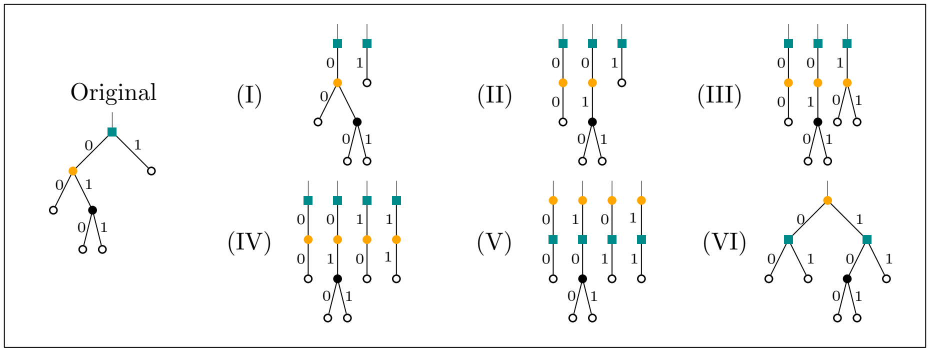

Fig. 2: Illustration of PLASTIC

Steps (I) and (II). Create a disconnected representation of the subtree. If a node already contains the desired split attribute (e.g., the ●-node in Fig. 2) the branches below remain unchanged.

Steps (III) and (IV). Before restructuring can occur, PLASTIC requires each branch to contain a node with the desired split (●). Thus, our approach enforces the desired split at the leaf of a branch if necessary. Then it disconnects those new splits, similarly to the transition between Steps (I) and (II). As a result, all branches now contain the desired split at their second-to-last node.

Step (V). Move the second-to-last node in each branch to the root position.

Step (VI). Re-build the subtree. The first branch serves as the initial decision tree prototype for attaching all other branches iteratively: For each branch, except for the first one, the algorithm traverses the prototype until there is a divergence. For example, the second branch in Step (V) diverges already at the root, so PLASTIC attaches it as a new child to the root. The third branch follows the left path of the subtree and diverges at ■. So the approach creates another branch under the left ■-node. The last branch follows the right path and diverges at ■ and thus becomes another child under the right ■-node.

4.2 Algorithm

Alg. 3 describes decision tree restructuring. Restructure requires the root r of the subtree that needs restructuring, and the new desired split attribute a^*. In case of a numeric or binary categorical split, the procedure also requires the split threshold or split category v^*. In the following, we simply call v^* "split value".

Our algorithm starts by creating a copy of the subtree's root node (line 2) and initializing two queues Q and F (line 3 and 4): Q contains the disconnected branches. Each such branch is represented as a list of tuples. The i-th tuple contains the i-th node of a branch together with the split value that identifies the $i+1$-th node of the branch. After initialization, each branch only has one tuple. It contains the root and a split value that identifies the respective child. F will contain the finished branches, i.e., the ones that have the structure as shown in step (V), which our algorithm can incorporate in the restructured tree.

In the while-loop, our algorithm extends the branches in Q until they have the structure shown in step (V) (line 6). The algorithm then moves these branches to F. As soon as F contains branches, PLASTIC incorporates them into the restructured decision tree (lines 7–9). After PLASTIC has restructured all branches, it resets the counters at the internal nodes which have become invalid. This does not affect prediction, which is done by the leaf nodes. At last, our approach replaces the original subtree with its restructured counterpart (line 12).

Numeric and binary categorical splits. When dealing with numeric and binary categorical splits, not only the split attribute must be the desired one, but also the split value must match with the desired split value. Since the split values of numeric splits typically differ, our algorithm adjusts the node's split values to the desired value and resets the invalid counters within the subtree. By adjusting the split value, some part of the tree might have become unreachable. This occurs when the split value of a successor node, which splits at the same attribute, falls outside the adjusted range. Our algorithm eliminates such unreachable branches.

PLASTIC supports binary categorical splits through a set of transformations, illustrated in Fig. 3 and described with rigor in Appendix B. In the figure, v_i is the current split value of a binary split, v^* is the desired split value, and shaded branches stand for "multiway" splits with one child per category. Our algorithm applies these transformations to nodes that have already split at the desired attribute but whose split type (binary/multiway) or split value does not match v^*. This allows our algorithm to preserve most of the subtree without requiring pruning to accommodate the desired split.

Algorithm 3 PLASTIC

```
 1: procedure RESTRUCTURE($r, a^*, v^*$)
 2:     $r' \leftarrow$ a copy of $r$
 3:     $Q \leftarrow \{[(r, v_i) : v_i$ that identify the children of $r]\}$        ▷ Unfinished branches
 4:     $F \leftarrow \{\}$                                  ▷ Branches with structure as in step (V)
 5:     while $Q$ or $F$ not empty do
 6:         DECOUPLEBRANCHES($Q, F$)                                      ▷ Steps (I) – (V)
 7:         while $F$ not empty do
 8:             $b \leftarrow F$.popleft()
 9:             ADDBRANCHTOTREE($r', b$)                                       ▷ Step (VI)
10:     Reset counters in restructured nodes (excl. $r'$)
11:     Replace $r$ with restructured subtree $r'$
```

Runtime discussion. During inference, each new data point traverses the tree until it reaches a leaf. So the time for inference depends on the depth of the

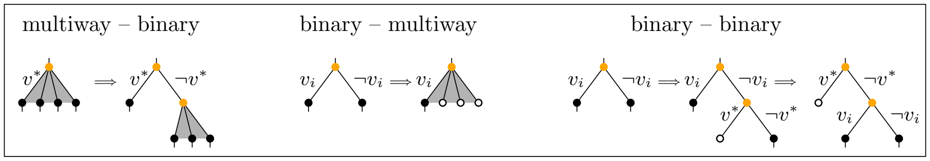

Fig. 3: Transformations to handle different types of categorical splits

respective branch. This is the same as in HT and EFDT. The same holds when updating the counters. The complexity of restructuring a given subtree is linear with regard to its size (number of nodes). See Appendix C, which contains the derivation. To limit PLASTIC's worst-case runtime for large decision trees, we configure it to traverse the restructured subtree only until a maximum depth ν (the auxiliary procedures in Appendix A show this optional parameter). When fixing ν, the worst case complexity becomes constant. If runtime is critical, one can also restructure the subtree on a separate thread and use the old subtree in the meantime. In our experiments with $\nu = 5$, runtime has not been an issue.

5 Evaluation

5.1 Algorithms

We compare PLASTIC to its closest competitors HT [9] and EFDT [21] and against EFHAT [20], which is the current state-of-the-art adaptative incremental tree for data streams with concept drift. We also include a simple adaptive baseline-version of our approach: PLASTIC-A detects changes in accuracy and starts training a background tree upon change detection. The background tree replaces the current tree once it is more accurate. This is conceptually similar to EFHAT, but much simpler: EFHAT relies on fine-grained monitoring at each tree node. PLASTIC-A only detects changes at the root of the tree.

For all approaches, we use a confidence of $\delta = 0.001$. In the case of EFDT and PLASTIC(-A), we re-evaluate splits with the same frequency as HT evaluates if a leaf should be split (the *grace period* parameter). We leave the grace period at its default choice of 200. We use this interval in EFHAT and PLASTIC-A to determine the number of instances these approaches should have seen before considering the background tree as replacement. As in [20], we instantiate EFHAT with an ADWIN change detector in its default configuration. For PLASTIC and PLASTIC-A, we use a default value of $\nu = 5$ that seemed to provide a good runtime-accuracy-tradeoff in general. Majority voting is used for prediction.

5.2 Data streams

We evaluate the approaches on 9 synthetic and 15 real-world data streams using a test-then-train methodology. The synthetic data streams are free of concept drift and instances are independent and identically distributed (iid). The real-world data streams may contain concept drift and violate the non-iid assumption. This allows to examine the algorithms' behavior in controlled environments and in real-world scenarios. For synthetic data, we generate 2×10^5 instances, repeat each experiment 30 times and set the generators' random seeds to the experiment repetition index. We do not require multiple experiment repetitions on real-world data as all approaches are deterministic and the order of data points is given.

Synthetic generators: We use MOA's synthetic data stream generators for classification. Unless stated otherwise, we use their default configuration. **Agrawal** creates a binary classification problem with 9 attributes, out of which 6 are numeric and 3 categorical. **Hyperplane** creates a binary classification problem in which the class label specifies whether a data point lies above or below a randomly parameterized hyperplane. In **LED** the task is to predict the digits displayed on a seven-segment display. **RandomRBF** produces a radial basis function stream. Samples cluster normally-distributed around a set of randomly chosen centroids. An instance's class label is the label of the closest centroid. **RandomTree-(c, m, n)** are variants of MOA's random tree generator that create a 5-class classification problem by sampling instances from a randomly created decision tree with 10 binary categorical attributes (c), 5 binary categorical and numerical attributes (m), and 10 numerical attributes (n), respectively. **SEA** generates a binary classification problem from 3 numerical attributes of which only 2 are relevant for classification. By default, the generator adds 10% noise to the data. **Waveform** generates a 3-class classification problem with 21 numeric attributes based on a random differentiation of waveforms.

Real-world data streams: We evaluate our approach on the same real-world data streams as used by our closest competitor EFDT. Exceptions are the datasets Skin and Font: we refrain from using them as they do not represent real-world streaming data and are (almost perfectly) sorted by label. Instead, we add a more recent activity recognition data stream, HARTH, as well as RIALTO, INSECTS, and SENSORS, the largest data streams from the USP change point detection benchmark [33]. Refer to Table 1 for a summary of the streams.

5.3 PLASTIC vs. EFDT

We first compare PLASTIC to its predecessor EFDT. This lets us showcase the effect of PLASTIC's restructuring procedure vs. EFDT's subtree pruning. Fig. 4 shows the accuracy difference between both approaches on synthetic data. Fig. 5 shows the same for real-world data. For example, a value of 10 in the plots means that PLASTIC outperformed EFDT by 10% at that point in the stream. Respectively, a negative value means that EFDT outperformed PLASTIC. Note that we will compare the average accuracy between all approaches in Sec. 5.4.

We also provide the maximum and minimum values across the experiment repetitions for synthetic data, which are indicated by the shaded area. For real-world data, we omitted the results for POKER and SUSY from the figure. In

Table 1: real-world data streams

Dataset	Classification task	# Inst.	# Num.	# Cat.	# Class
RIALTO [19]		82,250	27	–	10
SENSORS [42]	Environment monitoring	2,219,803	5	–	54
COVTYPE [7]		581,012	41	11	7
HARTH [18]		6,461,328	6	–	12
PAMAP2 [25]	Human activity recognition	3,850,505	52	–	25
WISDM [37]		15,630,426	1	3	6
HEPMASS [39]		10,500,000	28	–	2
HIGGS [1]	Physics particle detection	11,000,000	28	–	2
SUSY [38]		5,000,000	18	–	2
AIRLINES [17]	Flight delay (yes, no)	539,383	4	3	2
GAS [11]	Gas with highest concentration	4,178,504	16	–	3
INSECTS [33]	Flying insects classification	905,145	33	–	22
KDD [34]	Network intrusion detection	494,021	34	7	23
POKER [26]	Prediction of Poker hands	1,025,010	–	10	10

POKER, no tree-revision took place, resulting in identical results for PLASTIC and EFDT; the results for SUSY yield similar results to HEPMASS.

We observe in Fig. 4 that EFDT's performance decreases frequently and suddenly compared to PLASTIC. This is due to EFDT's subtree pruning which can lead to removal of large parts of the decision tree. PLASTIC, on the other hand, restructures the branches and thus remains able to predict with comparably high accuracy. EFDT's subtree pruning occurs even on supposedly easy problems such as LED. Though the difference in terms of average accuracy between both approaches may only be small on some data streams (e.g., a mere 0.5% on LED), large temporary drops in predictive performance are nonetheless problematic, for instance in safety- or security critical applications.

Our results on real-world data are similar: our approach exhibits fewer and less extreme accuracy drops compared to EFDT on 10 out of 14 data streams, except on AIRLINES, HEPMASS, HIGGS and SUSY. This is surprising, as data streams with concept drift should favor algorithms that implement some kind of forgetting-mechanism — for instance, the subtree pruning in EFDT. It seems that, in reality, many concept drifts do not or only partially affect the decision boundary. In such cases, keeping information through restructuring remains advantageous, explaining PLASTIC's superior performance over EFDT. On AIRLINES, EFDT outperforms our approach in particular at the beginning. Both approaches perform similar on HEPMASS, SUSY, and HIGGS, although it seems that our approach has a slight edge over EFDT.

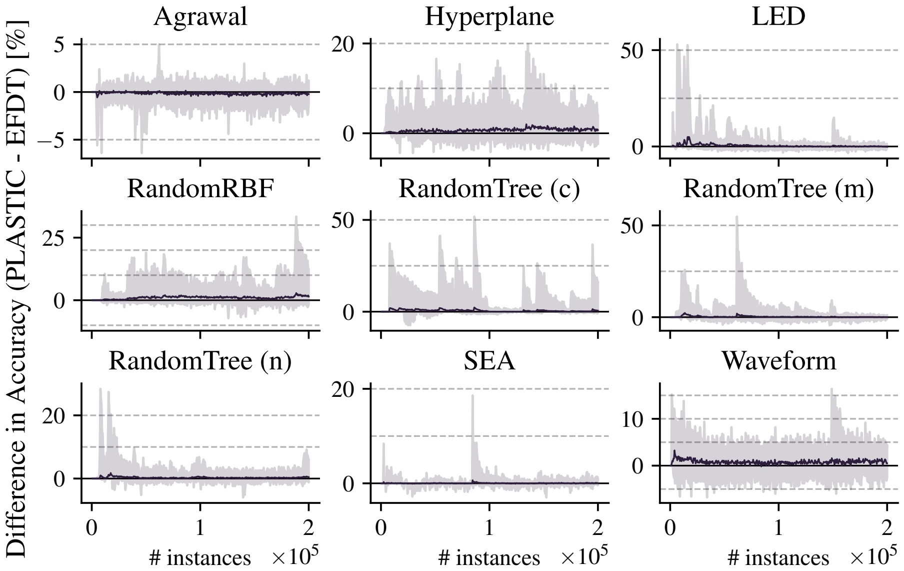

Fig. 4: Difference in accuracy between PLASTIC and EFDT on synthetic data. Shaded area represents maximum and minimum across experiment repetitions.

5.4 Average accuracy and ranks

Tables 2 and 3 respectively show the average accuracy and ranks of all approaches on synthetic and real-world data. The last column, NC, shows the accuracy of a no-change classifier that always predicts the last seen label.

In terms of average accuracy, EFHAT and PLASTIC perform equally well on synthetic data (86.6%). PLASTIC-A achieves 86.5%, EFDT 86.1%, and HT 79.7%. EFHAT and PLASTIC also have similar ranks (2.1 and 2.2), followed by PLASTIC-A (2.6). The good performance of EFHAT on these data streams is in line with the findings in the original paper [20] which already suggest that EFHAT performs well even on stationary data. HT's splitting mechanism seems overly conservative, leading to slower learning and lower accuracy on average. EFDT performs re-splitting too frequently, resulting in sudden accuracy drops (cf. Fig. 4). Last, we explain the slight difference between PLASTIC and PLASTIC-A with false alarms of the change detection mechanism. This can happen because of slight decreases in accuracy after restructuring (remember that we prune the restructured subtree at a depth of $\nu = 5$).

On real-world data, PLASTIC-A performs best, beating its nonadaptive counterpart by 1.1% and EFHAT by 1.4% on average. PLASTIC outperforms EFDT by 2.3%. Our approaches also achieve the best rank (2.29), followed by EFHAT (2.5), EFDT (3.71) and HT (4.21). The competitive performance of PLASTIC is surprising given that approaches with drift adaptation mechanisms should have a clear advantage. In particular, PLASTIC performs well on the

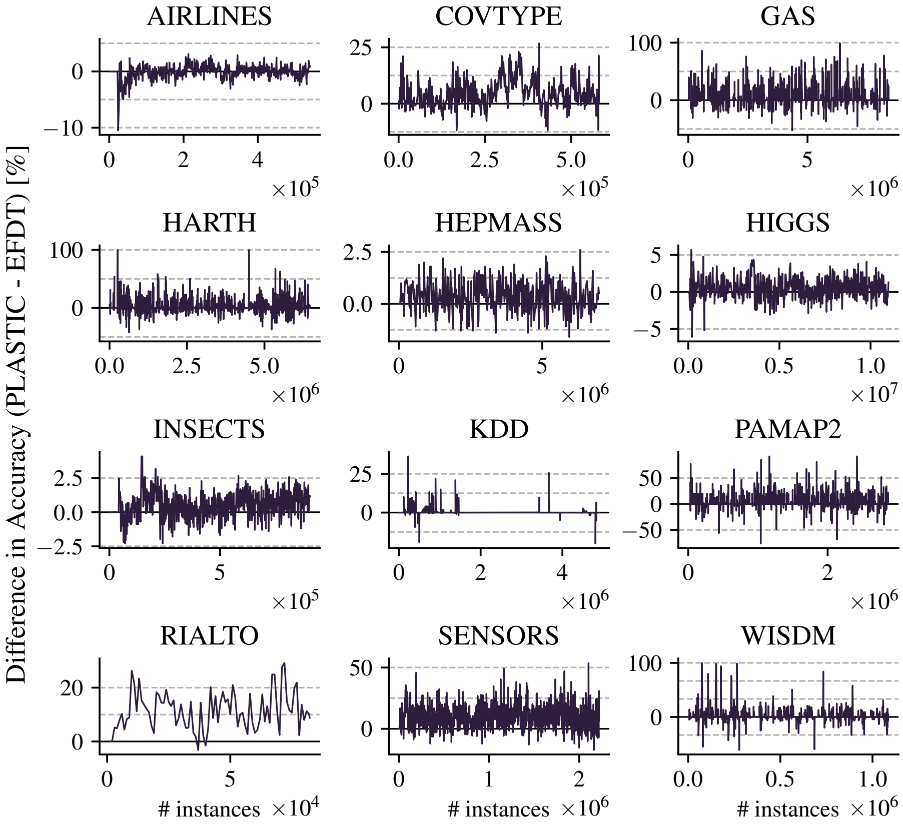

Fig. 5: Difference in accuracy between PLASTIC and EFDT on real-world data

environment monitoring data streams, where it beats EFHAT by up to 6.9%. We assume that these streams contain recurring concept drift (e.g., due to day and night) and therefore favor algorithms that retain information.

Overall, we notice that change adaptation is particularly effective when subsequent observations mostly have the same label (see the data streams where the accuracy of a no-change classifier is 99.9%). On such streams, a newly created background tree will have a similar performance to NC, and hence quickly replace the current tree. (Note, however, that this replacement also causes forgetting of the old concepts and leads to poor performance if the concepts re-occur.)

5.5 Runtime

The last rows in Table 2 and Table 3 show the geometric average wall-clock time of all approaches. We use the geometric mean to ensure that the dataset with the highest runtime does not dominate the reported number. As expected, HT is the fastest approach as it does not perform split revision.

Table 2: Average accuracy on synthetic data

Approach	HT	EFDT	EFHAT	PLASTIC	PLASTIC-A
Agrawal	**94.7**	94.6	93.9	94.4	94.5
Hyperplane	76.4	76.9	77.7	77.6	**77.8**
LED	48.0	72.6	73.0	**73.1**	**73.1**
RandomRBF	83.5	84.9	**86.1**	85.9	85.0
RandomTree (c)	82.5	96.0	**96.5**	**96.5**	**96.5**
RandomTree (m)	87.4	94.8	**95.2**	95.1	95.1
RandomTree (n)	84.3	91.8	**92.3**	92.1	91.7
SEA	87.4	**88.0**	**88.0**	**88.0**	87.7
Waveform	72.7	75.8	76.3	76.6	**76.8**
Accuracy	79.7	86.1	**86.6**	**86.6**	86.5
Standard dev.	9.89	5.55	5.40	5.35	**5.29**
Rank	4.6	3.6	**2.1**	2.2	2.6
Runtime	**0.57**	1.63	1.51	1.85	2.60

On synthetic data, EFHAT is faster than EFDT and PLASTIC, although the difference is rather small. The additional runtime is due EFDT's split revision and PLASTIC's restructuring. PLASTIC-A is the slowest approach on synthetic data. The approach not only performs restructuring, but also grows a background tree if it detected a change. Since the synthetic data streams are iid and stationary, it is harder for the background tree to outperform the current tree. Therefore, PLASTIC-A might maintain the current and the background tree simultaneously for an extended time period.

On real-world data, EFDT is about 25% faster than PLASTIC. EFHAT, on the other hand, now takes almost twice as long as EFDT. We assume is due to the large number of background trees that EFHAT maintains. EFHAT is also slower than PLASTIC-A, which maintains at most one background tree.

6 Conclusion

We presented PLASTIC, an incremental decision tree algorithm for data streams. PLASTIC leverages the concept of decision tree plasticity to overcome the effects of sudden and detrimental subtree pruning during split revision. Our experiments on commonly used synthetic and real-world data show that our approach has better worst-case accuracy than its predecessor EFDT and outperforms the current state of the art on real-world data.

We identify multiple directions for future work: First, one could investigate how our approach reacts to different types concept drift and develop an improved version of PLASTIC-A. Second, one could work on PLASTIC's runtime efficiency, e.g., by defining parameter ν as a function of the relevance of the affected subtree for inference. Third, we assumed that all instances have labels and that all labels are readily available. In practice, however, labels can be sparse

Table 3: Average accuracy on real-world data

Approach	HT	EFDT	EFHAT	PLASTIC	PLASTIC-A	NC
RIALTO	24.2	37.8	42.3	**49.2**	47.4	0.0
SENSORS	15.8	38.2	42.7	**48.1**	47.1	0.1
COVTYPE	68.3	77.4	79.6	**82.1**	81.3	95.1
HARTH	79.5	86.5	89.2	88.3	**90.9**	99.9
PAMAP2	58.4	94.5	98.3	96.6	**98.6**	99.9
WISDM	65.6	80.6	89.0	82.6	**93.1**	99.9
HEPMASS	**85.3**	84.7	85.2	85.0	85.0	50.0
HIGGS	69.4	68.4	**70.0**	68.9	68.3	50.2
SUSY	**78.9**	78.7	**78.9**	78.8	78.8	50.4
AIRLINES	62.2	63.7	62.7	63.6	**64.2**	58.0
GAS	91.5	96.7	99.1	98.2	**99.2**	99.9
INSECTS	55.6	64.9	65.3	**65.4**	64.8	13.1
KDD	99.5	**99.8**	99.3	**99.8**	99.7	99.9
POKER	53.6	67.4	67.4	67.4	**70.7**	43.2
Accuracy	64.8	74.2	76.4	76.7	**77.8**	61.4
Rank	4.21	3.71	2.50	**2.29**	**2.29**	–
Runtime	**61.8**	110.6	198.6	141.6	175.0	27.1

or costly to obtain (semi-supervised / active learning setting), only arrive after some time (delayed-labeling setting), or both [14]. Moving forward, one could investigate PLASTIC in those more challenging settings.

Acknowledgments. This work was supported by the German Research Foundation (DFG) as part of the Research Training Group GRK 2153: Energy Status Data – Informatics Methods for its Collection, Analysis, and Exploitation and by the Baden-Württemberg Foundation via the Elite Program for Postdoctoral Researchers.

This preprint has not undergone any post-submission improvements or corrections. The Version of Record of this contribution is published in Lecture Notes in Computer Science ((LNAI,volume 14945), and is available online at https://doi.org/10.1007/978-3-031-70362-1_3.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Baldi, P., Sadowski, P., Whiteson, D.: Searching for exotic particles in high-energy physics with deep learning. Nature Communications **5**(1), 4308 (Jul 2014). https://doi.org/10.1038/ncomms5308
2. Bifet, A., Gavaldà, R.: Adaptive learning from evolving data streams. In: IDA. Lecture notes in computer science, vol. 5772, pp. 249–260. Springer (2009). https://doi.org/10.1007/978-3-642-03915-7_22

3. Bifet, A., Gavaldà, R., Holmes, G., Pfahringer, B.: Big Data Stream Mining. In: Machine Learning for Data Streams: with Practical Examples in MOA, p. 0. The MIT Press (Mar 2018), https://doi.org/10.7551/mitpress/10654.003.0006
4. Bifet, A., Gavaldà, R., Holmes, G., Pfahringer, B.: Classification. In: Machine Learning for Data Streams: with Practical Examples in MOA, p. 0. The MIT Press (Mar 2018), https://doi.org/10.7551/mitpress/10654.003.0011
5. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: Massive online analysis. Journal of Machine Learning Research **11**, 1601–1604 (2010). https://doi.org/10.5555/1756006.1859903
6. Bifet, A., Holmes, G., Pfahringer, B., Frank, E.: Fast perceptron decision tree learning from evolving data streams. In: PAKDD. vol. 6119, pp. 299–310. Springer (2010). https://doi.org/10.1007/978-3-642-13672-6_30
7. Blackard, J.A., Dean, D.J.: Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. Computers and Electronics in Agriculture **24**(3), 131–151 (1999). https://doi.org/10.1016/S0168-1699(99)00046-0
8. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth (1984)
9. Domingos, P.M., Hulten, G.: Mining high-speed data streams. In: KDD. pp. 71–80. ACM (2000)
10. Ducange, P., Marcelloni, F., Pecori, R.: Fuzzy hoeffding decision tree for data stream classification **14**(1), 946–964 (2021). https://doi.org/10.2991/ijcis.d.210212.001
11. Fonollosa, J.: Gas sensor array under dynamic gas mixtures (2015). https://doi.org/10.24432/C5WP4C
12. Gama, J., Fernandes, R., Rocha, R.: Decision trees for mining data streams. Intelligent Data Analysis **10**(1), 23–45 (2006), http://content.iospress.com/articles/intelligent-data-analysis/ida00234
13. Gama, J., Rocha, R., Medas, P.: Accurate decision trees for mining high-speed data streams. In: KDD. pp. 523–528. ACM (2003). https://doi.org/10.1145/956750.956813
14. Gomes, H.M., Grzenda, M., de Mello, R.F., Read, J., Nguyen, M.H.L., Bifet, A.: A survey on semi-supervised learning for delayed partially labelled data streams. ACM Comput. Surv. **55**(4), 75:1–75:42 (2023). https://doi.org/10.1145/3523055
15. Hashemi, S., Yang, Y.: Flexible decision tree for data stream classification in the presence of concept change, noise and missing values. Data Mining and Knowledge Discovery **19**(1), 95–131 (2009). https://doi.org/10.1007/s10618-009-0130-9
16. Hulten, G., Spencer, L., Domingos, P.M.: Mining time-changing data streams. In: SIGKDD. pp. 97–106. ACM (2001). https://doi.org/10.1145/502512.502529
17. Ikonomovska, E., Gama, J., Džeroski, S.: Learning model trees from evolving data streams. Data Mining and Knowledge Discovery **23**(1), 128–168 (Jul 2011). https://doi.org/10.1007/s10618-010-0201-y
18. Logacjov, A., Bach, K., Kongsvold, A., Bårdstu, H.B., Mork, P.J.: HARTH: a human activity recognition dataset for machine learning. Sensors **21**(23), 7853 (2021). https://doi.org/10.3390/s21237853
19. Losing, V., Hammer, B., Wersing, H.: KNN classifier with self adjusting memory for heterogeneous concept drift. In: ICDM. pp. 291–300 (2016). https://doi.org/10.1109/ICDM.2016.0040
20. Manapragada, C., Salehi, M., Webb, G.I.: Extremely fast hoeffding adaptive tree. In: ICDM. pp. 319–328. IEEE (2022). https://doi.org/10.1109/ICDM54844.2022.00042

21. Manapragada, C., Webb, G.I., Salehi, M.: Extremely fast decision tree. In: KDD. pp. 1953–1962. ACM (2018). https://doi.org/10.1145/3219819.3220005
22. McMahan, H.B., Holt, G., Sculley, D., Young, M., Ebner, D., Grady, J., Nie, L., Phillips, T., Davydov, E., Golovin, D., Chikkerur, S., Liu, D., Wattenberg, M., Hrafnkelsson, A.M., Boulos, T., Kubica, J.: Ad click prediction: a view from the trenches. In: KDD. pp. 1222–1230. ACM (2013). https://doi.org/10.1145/2487575.2488200
23. Pfahringer, B., Holmes, G., Kirkby, R.: New options for hoeffding trees. In: AI. pp. 90–99. Springer (2007). https://doi.org/10.1007/978-3-540-76928-6_11
24. Quinlan, J.R.: C4.5: Programs for machine learning. Morgan Kaufmann (1993)
25. Reiss, A.: PAMAP2 Physical Activity Monitoring (2012). https://doi.org/10.24432/C5NW2H
26. Robert, C., Oppacher, F.: Poker Hand (2006). https://doi.org/10.24432/C5KW38
27. Rožanec, J.M., Trajkova, E., Dam, P., Fortuna, B., Mladenić, D.: Streaming machine learning and online active learning for automated visual Inspection. IFAC-PapersOnLine **55**(2), 277–282 (2022). https://doi.org/10.1016/j.ifacol.2022.04.206
28. Rutkowski, L., Jaworski, M., Pietruczuk, L., Duda, P.: The CART decision tree for mining data streams **266**, 1–15 (2014). https://doi.org/10.1016/j.ins.2013.12.060
29. Rutkowski, L., Jaworski, M., Pietruczuk, L., Duda, P.: Decision trees for mining data streams based on the gaussian approximation. IEEE Trans. Knowl. Data Eng. **26**(1), 108–119 (2014). https://doi.org/10.1109/TKDE.2013.34
30. Rutkowski, L., Jaworski, M., Pietruczuk, L., Duda, P.: A new method for data stream mining based on the misclassification error. IEEE Trans. Neural Networks Learn. Syst. **26**(5), 1048–1059 (2015). https://doi.org/10.1109/TNNLS.2014.2333557
31. Rutkowski, L., Pietruczuk, L., Duda, P., Jaworski, M.: Decision trees for mining data streams based on the McDiarmid's bound. IEEE Trans. Knowl. Data Eng. **25**(6), 1272–1279 (2013). https://doi.org/10.1109/TKDE.2012.66
32. Sedhai, S., Sun, A.: Semi-supervised spam detection in twitter stream. IEEE Transactions on Computational Social Systems **5**(1), 169–175 (2018). https://doi.org/10.1109/TCSS.2017.2773581
33. Souza, V.M.A., dos Reis, D.M., Maletzke, A.G., Batista, G.E.A.P.A.: Challenges in benchmarking stream learning algorithms with real-world data. Data Mining and Knowledge Discovery **34**(6), 1805–1858 (Nov 2020). https://doi.org/10.1007/s10618-020-00698-5
34. Stolfo, S., Fan, W., Lee, W., Prodromidis, A., Chan, P.: KDD Cup 1999 Data (1998). https://doi.org/10.24432/C51C7N
35. Utgoff, P.E.: Incremental induction of decision trees. Machine Learning **4**, 161–186 (1989). https://doi.org/10.1023/A:1022699900025
36. Utgoff, P.E., Berkman, N.C., Clouse, J.A.: Decision tree induction based on efficient tree restructuring. Machine Learning **29**(1), 5–44 (1997). https://doi.org/10.1023/A:1007413323501
37. Weiss, G.: WISDM smartphone and smartwatch activity and biometrics dataset (2019), https://doi.org/10.24432/C5HK59
38. Whiteson, D.: SUSY (2014), https://doi.org/10.24432/C54606
39. Whiteson, D.: HEPMASS (2016). https://doi.org/10.24432/C5PP5W
40. Wu, X., Li, P.P., Hu, X.: Learning from concept drifting data streams with unlabeled data. Neurocomputing **92**, 145–155 (2012). https://doi.org/10.1016/j.neucom.2011.08.041

41. Zhang, D., Xu, B., Wood, J.: Predict failures in production lines: A two-stage approach with clustering and supervised learning. In: Big Data. pp. 2070–2074 (2016). https://doi.org/10.1109/BigData.2016.7840832
42. Zhu, X.: Sensor Stream (2010), http://www.cse.fau.edu/~xqzhu/stream.html

A Additional Pseudocode

Alg. 4 contains the complete pseudocode of PLASTIC's restructuring procedure. See also Sec. 4.2 in the paper.

B Description of Categorical Split Transformations

This section augments Sec. 4.2 in the paper. Here, we refer to "root" as the ●-node that is visible in Fig. 6, not to the root node of the restructured subtree.

Case 1: Multiway – Binary. PLASTIC enforces the desired split at the root. It re-attaches all previous children (except the v^*-child) to the newly created $\neg v^*$-child. This operation increases subtree depth by 1.

Case 2: Binary – Multiway The root currently splits at the desired split attribute but uses split value v_i instead of v^*. PLASTIC enforces the desired multiway split at the root and re-attaches the children of the original v_i-child.

Case 3: Binary – Binary. The root currently splits at the desired split attribute but uses split value v_i instead of v^*. PLASTIC enforces the desired binary split at the $\neg v_i$-child. This increases subtree depth by 1. Next, the approach attaches the subtree under the original $\neg v_i$-child to the $\neg v^*$-child. Last, the approach swaps the newly created v^*-child with the original v_i-child that was still attached to the root.

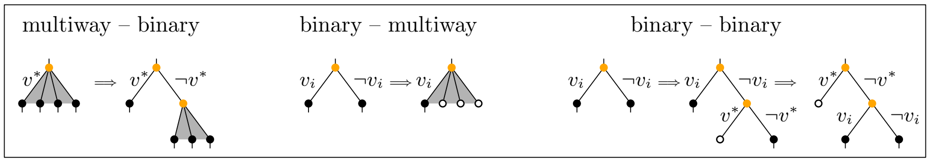

Fig. 6: Categorical split transformations (same as Fig 3)

C Runtime complexity

We make two worst-case assumptions in our derivation: (1) The subtree is balanced with depth of k. (2) The nodes in the subtree split at the k highest-cardinality attributes. Let $\mathcal{C}$ be the set of all c_j for $j = 1, 2, \ldots, d$. For numeric attributes, let $c_j = 2$. Now define

$$C = \{c_j \mid c_j \in \mathcal{C}, \text{ and } c_j \text{ is among the } k \text{ largest values in } \mathcal{C}\}.$$

Then, $\prod_{c_j \in C} c_j$ is a limit on the number of subtree leaves. Last, let c^* be the number of distinct categories of the desired split attribute. Since PLASTIC enforces a split of the leaves at the desired split attribute, the worst-case maximum number of restructured branches is $c^* \prod_{c_j \in C} c_j \leq c^* c_{max}^k$ where $c_{max} = \max(C)$. The quantity $c^* c_{max}^k$ is also the maximum size of any restructured subtree. Hence, restructuring grows linearly with subtree size.

Algorithm 4 PLASTIC – all procedures

```
procedure RESTRUCTURE($r, a^*, \nu, v^*$)
    $r' \leftarrow$ a copy of $r$
    $Q \leftarrow \{[(r, v_i) : v_i$ that identify the children of $r]\}$    ▷ Unfinished branches
    $F \leftarrow \{\}$    ▷ Branches with structure as in step (V)
    while $Q$ or $F$ not empty do
        DECOUPLEBRANCHES($Q, F, \nu$)    ▷ Steps (I) – (V)
        while $F$ not empty do
            $b \leftarrow F$.popleft()
            ADDBRANCHTOTREE($r', b$)    ▷ Step (VI)
    Reset counters in restructured nodes (excl. $r'$)
    Replace $r$ with restructured subtree $r'$

procedure DECOUPLEBRANCHES($Q, F, \nu, a^*, v^*$)
    for $b \in Q$ do
        (lastNode, –) $\leftarrow b$
        isFinished $\leftarrow$ lastNode splits at $a^*$ or b.length $\geq \nu$
        if isFinished then
            Force desired split at lastNode if necessary
            newBranches $\leftarrow$ EXTENDBRANCH($b$)
            Move 2nd-last element in $b$ to position 0
            $F$.add(newBranches)
            $Q$.remove($b$)
            return
        else
            newBranches $\leftarrow$ EXTENDBRANCH($b$)
            $Q$.remove($b$)
            $Q$.add(newBranches)

procedure ADDBRANCHTOTREE($r', b$)
    current node $c \leftarrow r'$
    currentDepth $\leftarrow 0$
    while currentDepth $< b$.length do
        (node $n$, key $v$) $\leftarrow b$[currentDepth]
        if $c$ and $n$ have the same split attribute then
            if key $v$ already in successors of $c$ then    ▷ Move to successor
                $c \leftarrow$ get successor of $c$ for key $v$
            else    ▷ Add new successor to $c$, then move to new successor
                (newSuccessor, –) $\leftarrow b$[currentDepth + 1]
                Add newSuccessor to $c$ with key $v$
                $c \leftarrow$ newSuccessor
        Increment currentDepth

procedure EXTENDBRANCH($b$)
    newBranches $\leftarrow \{\}$
    $l \leftarrow$ last node in $b$
    for all child keys $v_i$ of $l$ do
        newBranches.add($b + (l, v_i)$)
    return newBranches
```