Analysis and Optimization of the Accelerated Argumentation-Based Learning Algorithm

Sander Kaatee

August 15, 2023

Abstract

Accelerated Argumentation-Based Learning (AABL) has been shown to have significant improvements in accuracy and learning speed when compared to other state-of-the-art machine learning techniques. Despite its promising performance, the algorithm has certain limitations and potential for optimization. In this paper, we provide an in-depth analysis of the AABL algorithm, examine its limitations, and propose possible optimizations. Three algorithms, the original implementation of AABL, Refactored Argumentation-Based Learning (RABL), and an implementation of AABL as it is described in the paper, are compared in various scenarios. The paper discusses the main differences between the algorithms and provides insights into the argumentation-based and machine learning aspects of AABL, as well as its limitations and potential for explainability.

1 Introduction

The increasing complexity of artificial intelligence models presents a fundamental challenge. The lack of interpretability and transparency in these systems, i.e. their 'black box' nature, raises concerns about potential biases and discriminatory outcomes and hampers users' trust and acceptance of their decisions. Consequently, Explainable Artificial Intelligence (XAI) has emerged as a research area, seeking to bridge the gap between human comprehension and AI reasoning. This paper delves into one approach within the realm of XAI, namely Accelerated Argumentation-Based Learning (AABL), which combines the power of machine learning with the structured reasoning of argumentation to provide human-understandable explanations for AI decisions.

AABL is an optimization of Argumentation-Based Learning (ABL), a machine learning (ML) technique that outperforms state-of-the-art methods in terms of accuracy and learning speed [1]. Proposed by Ayoobi et al.[1], Both methods claim to be based on the Bipolar Argumentation Framework (BAF) [2], which in turn is an extension of the Abstract Argumentation Framework (AF) by Dung [3].

Although AABL has been shown to have significant improvements over the original ABL process [1], there is a need to further investigate why this approach yields such promising results and explore its place in the machine learning/reinforcement learning literature. Additionally, for AABL we aim to assess the adaptability of AABL to other settings beyond its initial domain of application.

In this paper, we aim to address these research questions by providing a thorough examination of the AABL algorithm as proposed by Ayoobi et al. [1]. We will begin by providing a background on AABL and an outline of the algorithm. We will then replicate the original results, after which we propose an optimization as a result of refactoring the original implementation of AABL. We will also make a comparison between AABL as it was implemented and AABL as it was described, we will look at the discrepancies between these two version in the discussion section of this paper. Lastly we compare all three algorithms on a novel situation. An in-depth discusison is provided in the discussion (Section 5) as well as in the Appendix (Section 7).

2 Background

The original Argumentation Based Learning algorithm proposed by Ayoobi et al. in 2019 [1] has made improvements in online incremental learning scenarios in comparison to other methods. The original ABL exhibited good performance with increased accuracy and learning speed. However, it was limited by its memory and computational efficiency [1]. The authors addressed these limitations by introducing the AABL method, the pseudocode as given by Ayoobi et al. in [1] is shown in Section 3.2 as 'Algorithm 1: Accelerated Argumentation-Based Learning'.

The authors state that the Bipolar Argumentation Framework (BAF) [2] is the basis of the AABL method [1]. According to Dung [3], an Abstract Argumentation Framework (AF) consists of a set of arguments and a binary relation that shows attacks between arguments. The BAF builds upon Dung's concept by incorporating the notion that some arguments may support a conclusion while others may oppose it. In Algorithm 1, the support relations are used to determine what recovery label the algorithm should use. The attack relations are added to the BAF graph in Algorithm 2, note that in the provided pseudocode these attack relations are not accessed anywhere.

The purpose of the original ABL as presented in [1] is to provide a novel online incremental learning approach that can autonomously handle external failures resulting from changes in the environment. Existing research tends to offer special-purpose solutions, in [1] the authors claim that ABL generates a set of hypotheses that can "be used for finding the best recovery behavior at each failure state". Additionally, the authors claim that ABL addresses the issue of poor generalization observed in current online learning algorithms, making it applicable to various domains, including robotics. Moreover, as the approach is based on argumentation, it allows for generating an explainable set of rules suitable for human-robot interaction.

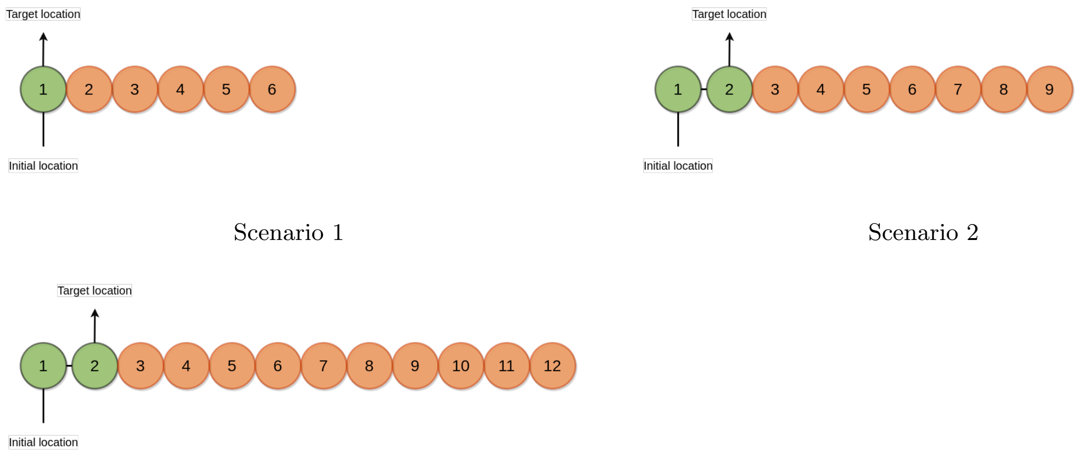
3 Methods

We compare a total of three different algorithms: Accelerated Arugmentation Based Learning (AABL), Refactored Argumentation Based Learning (RABL), and our implementation of how AABL was described in the original paper [1] (Table 1).

We test and compare the algorithms on the original scenarios as described by [1] (scenario 1, 2 and 3) in experiment 0, 1 and 2. In experiment 3 we compare all algorithms together on three newly described scenarios (i, ii, and iii).

Name	**Description**	**Detailed description at**
AABL	Accelerated Argumentation Based Learning, as implemented by H. Ayoobi	
RABL	Our refactored version of the original implementation of AABL	Section 3.1
Pseudocode	Our implementation of AABL as it is described in	Section 3.2

Table 1: The three algorithms compared in this paper



Scenario 1

Scenario 2

Scenario 3

Figure 1: The three original scenarios

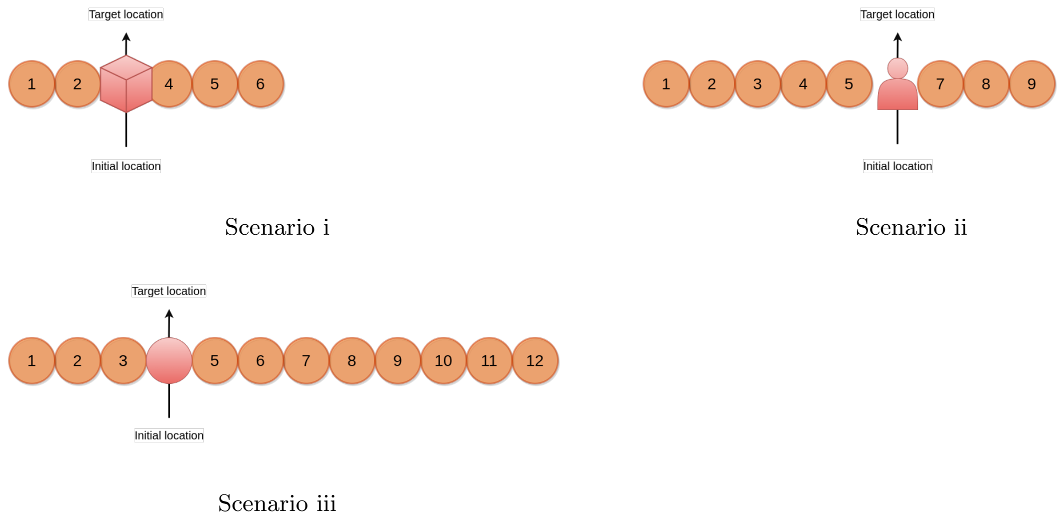


Figure 2: The three new scenarios

In the original paper, AABL is tested by having to find solutions to an obstacle course. AABL is handed a state: a field consisting of 6 to 12 locations. The amount of locations depend on the scenario: for scenario 1 there are six locations, for scenario 2 there are nine and for scenario 3 there are twelve. For each location there can be an object consisting of a color and a concept (e.g. a blue ball or a yellow person). For each state there is one correct action associated, either: ask, push, alternative route or continue. Depending on the scenario there are only either one or two locations relevant for determining the correct action, the other locations/objects function as noise for the algorithm.

Our three new scenarios are based on the original three scenarios. However, instead of the correct action being dependent on one or two locations, the correct action is dependent on whatever object is colored 'red'. Again the other locations/objects function as noise.

In theory there should be no difference in performance for the algorithms on the new scenarios as compared to the original scenarios. As the algorithms are expected to deduce what information is relevant for the correct action (i.e. the correct set of arguments), they should be able to figure out that the combination of red and a certain concept has a direct relation with the correct action.

3.1 Experiment 1: RABL vs AABL

Ayoobi's original code, as taken from his GitHub[1], contains large amounts of unnecessary and unused code. Apart from that, the original code contains hard-coded optimizations such as automatic handling of empty locations.

[1]https://github.com/H-Ayoobi/Accelerated_ABL

In response to these issues, we have performed a comprehensive refactoring of the code. We removed all unused functions and variables, and introduced variable names that conceptually align better with what the code is doing. We also changed certain types for more compact code while improving ease of understanding. Most importantly, we made a significant change to the algorithm as follows.

The original code calculated all possible subsets of features up to length L, even though it only used these subsets to determine location numbers, as opposed to using the content of the subsets themselves. Therefore, we rewrote the algorithm to keep track of a list of locations instead. It still computes all possible combinations between these locations, however it is now more generalizable as the algorithm no longer requires having the possible input values hard-coded to determine whether something is a color or a concept.

We compared RABL against AABL on scenario 1, 2 and 3 (see Figure 1). We did 10 iterations of 200 steps using the original method for generating data as provided on the authors GitHub.

The initial hypothesis is that RABL performs the same as AABL.

3.2 Experiment 2: Pseudocode vs AABL

In Chapter 3 of [1] the following pseudocode is provided for describing the AABL algorithm, see Algorithm 1. An in-depth description is presented in addition to this pseudocode. With this experiment we will compare these descriptions to the working AABL. In the Appendix (Section 7) we systematically go through the textual description as provided in [1], we provide literal quotations and then compare them to the Pseudocode as well as AABL.

For this experiment we compared 'Pseudocode' against AABL on Scenario 1,2 and 3 (see Figure 1). We did 10 iterations of 200 steps using the original method for generating data.

The initial hypothesis is that Pseudocode generates the same result as AABL since it is supposed to be the literal description of the workings of AABL.

Algorithm 1 Accelerated Argumentation-Based Learning, taken from [1]

```
Require: Current BAF graph, Data Instance $X$ entering the argumentation-based learning model,
    feature values subsets' length $L$, The class label or Best Recovery Behavior (BRB) for $X$
Ensure: The predicted label for $X$ called $Y$
 1: procedure Argumentation-Based-Learning($BAF$, $X$, $L$, $BRB$)
 2:     Extract all feature value combinations in $X$ with length $L$ and add them to a list called
        $Combs$.
 3:     Let $SNs$ be the set of supporting nodes (in form of "supporting-node $\rightarrow$ supported-node")
        in the BAF.
 4:     for all $sn$ in $SNs$ do
 5:         for all $comb$ in $Combs$ do
 6:             if sn.supporting-node == comb then
 7:                 $Y$.Add(sn.supported-node)
 8:             end if
 9:         end for
10:     end for
11:     if $Y$ is not empty then
12:         if Length($Y$) == 1 then
13:             Apply $Y$ to environment and observe the result.
14:         else
15:             Select a prediction in $Y$ at random ($Y := Y[random\ index]$) and observe the result.
16:         end if
17:     else
18:         Randomly choose a prediction from the available class labels (observed recovery behav-
        iors).
19:         Increment $L$ := Update the BAF unit (using Algorithm 2 with input parameters: current
        BAF graph, BRB, Combs, SNs).
20:     end if
21:     while (should Increment L == True) do
22:         $L := L + 1$
23:         Compute the combinations of the feature values again as $Combs$.
24:         should Increment $L$ := Update the model with Algorithm 2.
25:     end while
26:     return $Y$
27: end procedure
```

Algorithm 2 Updating the BAF Unit, taken from [1]

```
Require: Current BAF graph, class label (Best Recovery Behavior) BRB, Combinations of
    feature values for X called Combs, Set of Supporting Nodes in the BAF graph SNs
Ensure: A Boolean variable "should Increment L" that tells whether L needs to be incremented
    or not.
 1: Let RNs be the set of all the class labels (Recovery behavior Nodes) in the BAF.
 2: Let attacks be the set all the attack relations (for a ∈ RNs and b ∈ RNs the attack relations
    are in form of "a → b") among the class labels (recovery behavior nodes) in the BAF.
                                                ▷ Updating attack relations and class labels
 3: if BRB is not in BAF then
 4:     add BRB to the BAF graph;
 5:     add bidirectional attacks between BRB and all the other class labels (recovery behavior
    nodes) as follows:
 6:     for all rn in RNs do
 7:         attacks.add( BRB → rn )
 8:         attacks.add( rn → BRB )
 9:     end for
10: end if
                                                            ▷ Updating support relations
11: Let should Increment L := True
12: for all comb in Combs do
13:     Let Add Support := True
14:     for all sn in SNs do
15:         if sn.supporting-node == comb then
16:             Add Support := False
17:             if sn.supported-node ≠ BRB then
18:                 Mark comb as a non-unique node that can not support any node in future.
19:                 Remove sn from the set supporting nodes in BAF SNs.
20:             end if
21:         end if
22:     end for
23:     if comb is not Marked then
24:         should Increment L := False
25:     end if
26:     if Add Support == True then
27:         SNs.add(comb → BRB)
28:     end if
29: end for
30: return should Increment L
31:
```

Table 2: Comparison of run-times for RABL and AABL.

	Time (seconds)	
	AABL	RABL
First Scenario	2.09	1.49
Second Scenario	22.37	24.49
Third Scenario	74.47	85.22

3.3 Experiment 3: AABL vs RABL vs Pseudocode

We compared all three algorithms AABL, RABL, and Pseudocode together on the three new scenarios as described in Section 3. Again we train for 200 instances over 10 iterations.

In addition to the three algorithms, we provide a reference level through a naive algorithm, 'Most-Common' that always returns the most often seen correct recovery behavior.

The initial hypothesis is that AABL, RABL and Pseudocode perform similarly to each other, and will have accuracies on Scenarios i, ii and iii comparable to their respective Scenarios 1,2,3. The naive algorithm 'Most-Common' should be outperformed significantly by all three algorithms.

4 Results

4.1 Experiment 1: RABL vs AABL

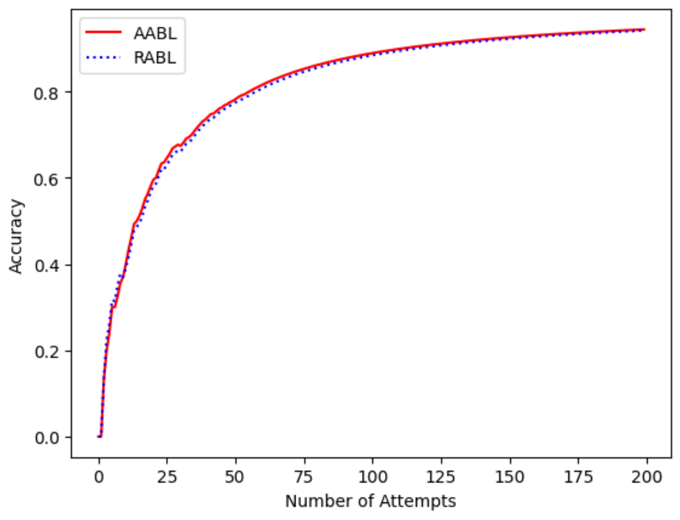
Figure 3 shows the comparison between AABL and RABL. To evaluate the performance differences, we employed the independent samples t-test, a suitable statistical test for comparing two independent algorithms on the same test scenarios. The independent samples t-test revealed no significant difference between AABL and RABL for Scenario 1 ($t(3998) = 0.245, p = .806$) and Scenario 2 ($t(3998) = 1.94, p = .052$), hence, this is consitent with our initial hypothesis that RABL and AABL should perform similarly.

However on Scenario 3 RABL performs slightly worse than AABL ($t(3998) = 2.40, p = .016$). This is likely due to RABL not discarding the empty locations.

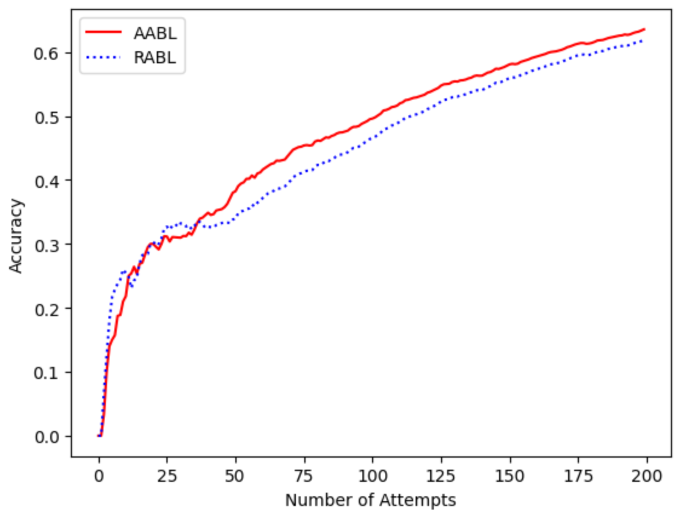
In Table 2 the run-times are presented for each algorithm per scenario.

4.2 Experiment 2: Pseudocode vs AABL

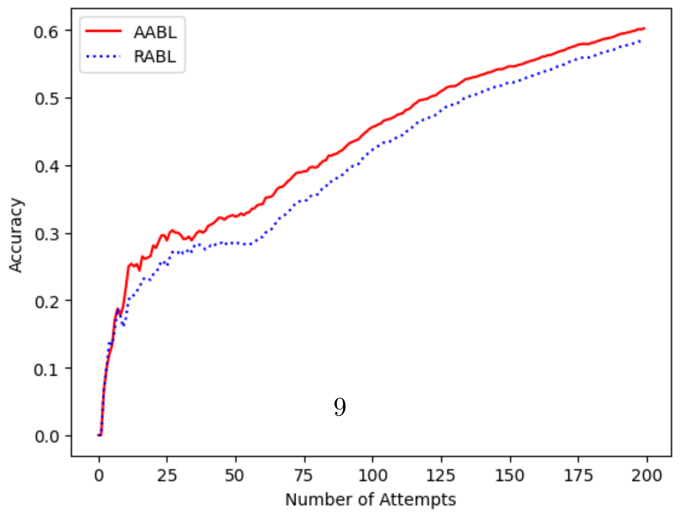
Figure 4 shows the comparison between AABL and Pseudocode. The results show that the re-implementation of the paper performed significantly worse than Ayoobi's AABL algorithm at accurately predicting correct recovery behaviors (for Scenario 1: $t(3998) = 24.02, p < .001$, for Scenario 2: $t(3998) = 15.21, p < .001$, for Scenario 3: $t(3998) = 17.29, p < .001$). This contradicts the original hypothesis that Pseudocode and AABL should perform the same.



(a) Scenario 1

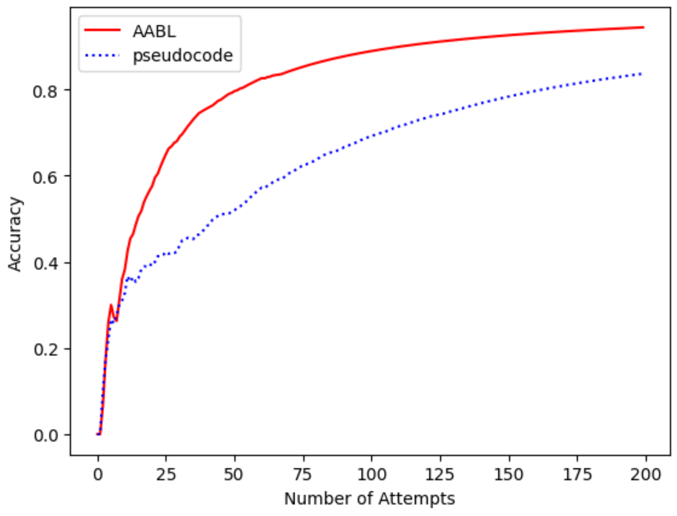


(b) Scenario 2

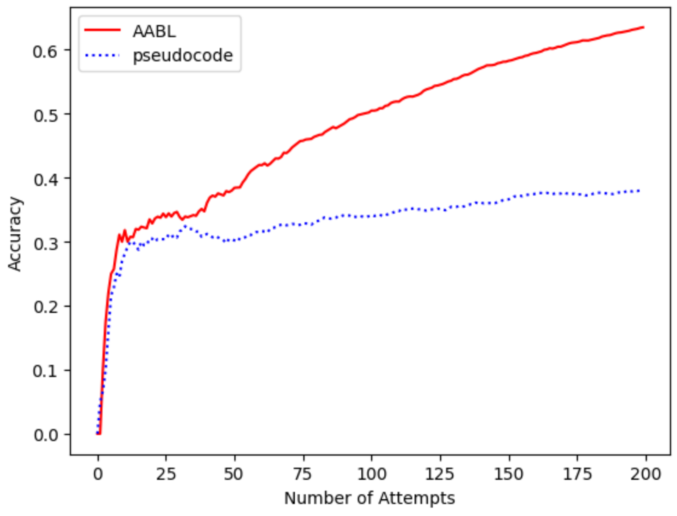


(c) Scenario 3

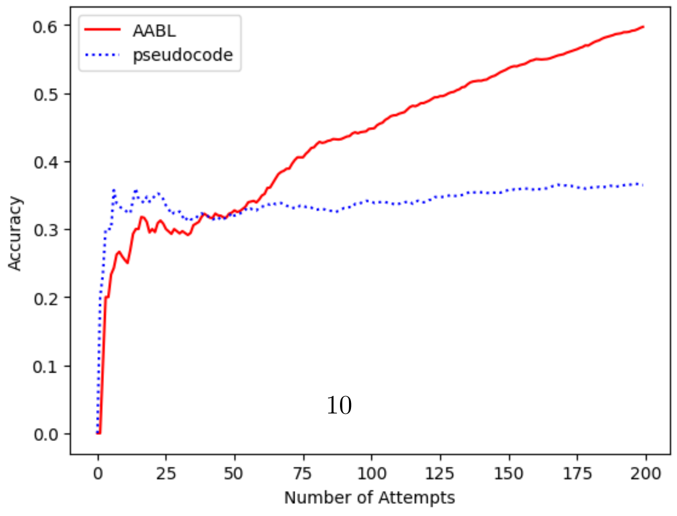
Figure 3: Performance AABL vs RABL



(a) Scenario 1

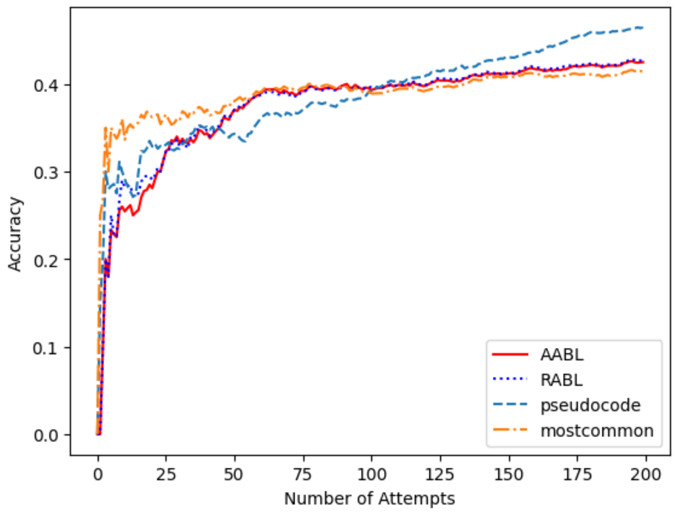


(b) Scenario 2

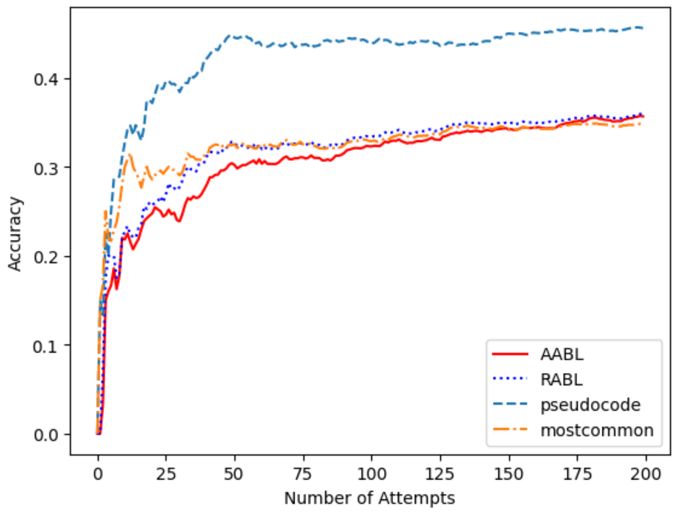


(c) Scenario 3

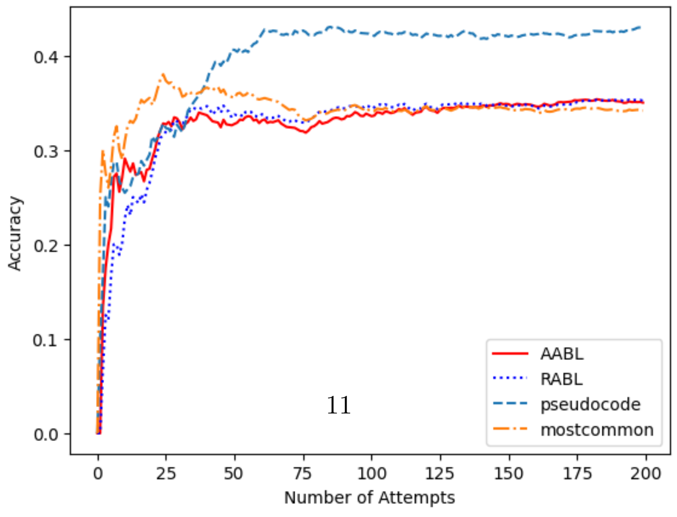
Figure 4: Performance of Pseudocode vs AABL



(a) Scenario i



(b) Scenario ii



(c) Scenario iii

Figure 5: Performance of AABL vs RABL vs Pseudocode vs Most-Common on the newly described scenarios

Table 3: Comparison of run-times for RABL, AABL and Pseudocode.

	Time (seconds)		
	AABL	RABL	Pseudocode
First Scenario	201.47	134.51	50.85
Second Scenario	3225.84	1943.28	442.88
Third Scenario	40522.76	30030.65	2820.02

4.3 Experiment 3: AABL vs RABL vs Pseudocode

Figure 5 shows the comparison between AABL, RABL and Pseudocode with Most-Common as a baseline comparison.

There is no difference between the performance of AABL, RABL and Most-Common. To evaluate the statistical significance of the (lack of) difference, we employed a One-Way ANOVA, an appropriate test for comparing the means of three or more independent groups. The degrees of freedom between groups, within groups, and the total degrees of freedom were 2, 1997, and 1999, respectively. The obtained F-statistic of 0.11 (p = 0.90) for Scenario i, 0.94 (p = 0.39) for Scenario ii, and 0.07 (p = 0.93) for Scenario iii, indicates that there is no significant difference in the categorization accuracy among the algorithms.

Pseudocode performs better than AABL on Scenario ii (t(3998) = 14.36, p ¡ .001) and Scenario iii (t(3998) = 10.02, p ¡ .001). There seems to be no difference in performance between Pseudocode and AABL on Scenario i (t(3998) = 1.60, p = 0.11), however we note that Pseudocode seems to still improve with more inputs, while the other three algorithms flatline.

These results cause us to reject our initial hypothesis that the algorithms should perform similar and all should outperform Most-Common.

Also note that RABL and AABL end with accuracies around 0.33 on the new scenarios, while on Scenario 1 an accuracy of about 0.95 was reached, on Scenario 2 an accuracy of about 0.65 and on Scenario 3 an accuracy of 0.60, which is quite a steep drop in performance while the scenarios are quite comparable. Pseudocode, however, seems to enjoy slightly better performances on the new scenarios.

5 Discussion

In Table 4 we present the main differences between the three tested algorithms. Neither AABL nor RABL explicitly utilize the support and attack relations common in argumentation theory. Instead these algorithms determine the relevant locations first by calculating the agreement between the ordered lists combinations of locations and the ordered list of recovery behaviors and taking the list with the highest agreement. Where-after it uses the previously seen scenarios as a lookup table.

One could argue that there is an implicit support relation between the recovery behavior and the object that was determined on the relevant location. In a similar sense there is a support

Table 4: Difference between algorithms

	AABL	RABL	Pseudocode
Support relations	Implicit	Implicit	Explicit
Attack relations	Implicit	Implicit	Explicit
Weights	Yes	Yes	No
Subsets of features	Yes	No	Yes
Hardcoded handling of empty locations	Yes	No	No
Determines relevant locations first	Yes	Yes	No

relation between the output heads of a neural network and the relevant class in a classification task, or any other machine learning algorithm for that matter.

In a similar vein, we can make the argument that there are implicit attack relations between all possible recovery behaviors as the problem is close-ended, there is only one correct recovery behavior for each generated scenario. Again, in a similar vein you can make this argument for any close-ended machine learning task.

The Pseucode algorithm explicitly implements both the attack and support relations. The support relations are used to determine what recovery behavior seems correct, however the attack relations are not used in the algorithm and are therefore unnecessary.

Both AABL and RABL utilize weights for determining the relevant locations, the weights are used as a metric for agreement between location and recovery behavior. In [1] Ayoobi et al. state that the weights are removed and when we look at the pseudocode in algorithm 1 and 2 we indeed see no mention of weights being updated.

Both AABL and Pseudocode calculate possible subsets of features, however RABL just calculates the possible combinations of locations (split between concept and color), as the features themselves are not used in AABL.

AABL has hardcoded information about the task, such as empty locations being discarded and the label values to determine whether something is a concept or a color. This hardcoded information has been removed in RABL as it was deemed unfair.

The mayor difference between AABL/RABL and Pseudocode is that AABL/RABL determines the relevant locations first. We would argue that this is over-optimized on the task of the original test scenarios. Ayoobi et al. [1] claim that ABL addresses the issue of poor generalization observed in current online learning algorithms, however we have showed with Experiment 3 that AABL does not even generalize to a small variation on the original task.

6 Conclusion

This paper explored the AABL algorithm as a part of Explainable Artificial Intelligence XAI. The research aimed to investigate the effectiveness and applicability of AABL in machine learning and reinforcement learning contexts, as well as identify possible optimizations and shortcomings.

The paper conducted a series of experiments comparing several versions of AABL, including the original implementation, a refactored version (RABL), and a pseudocode representation. The experiments replicated the original results, assessed the performance of the algorithms, and examined the adherence of the pseudocode to the actual implementation.

The results showed that the refactored version (RABL) performed similarly to AABL in terms of accuracy, while the pseudocode implementation yielded significantly lower accuracy. This finding contradicted the hypothesis that the pseudocode and AABL would produce similar results.

The comparison between the algorithms highlighted several differences, including the utilization of support and attack relations, the presence of weights, the consideration of subsets of features, and the handling of empty locations. These differences shed light on the underlying mechanisms of each algorithm and contributed to a deeper understanding of their strengths and limitations.

Although AABL provides some degree of explainability by indicating the object that influenced its decision, it falls short of providing a comprehensive explanation of the reasoning behind the selection. This limitation highlights the need for further advancements of AABL in terms of XAI.

In conclusion, this paper deepened the understanding of the AABL algorithm and its applicability in various domains. It identified areas for optimization and improvement, paving the way for future research and development in XAI.

References

[1] H. Ayoobi, “Explain what you see: argumentation-based learning and robotic vision,” Ph.D. dissertation, University of Groningen, 2023.

[2] L. Amgoud, C. Cayrol, M.-C. Lagasquie-Schiex, and P. Livet, “On bipolarity in argumentation frameworks,” *International Journal of Intelligent Systems*, vol. 23, no. 10, pp. 1062–1093, 2008.

[3] P. M. Dung, “On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games,” *Artificial intelligence*, vol. 77, no. 2, pp. 321–357, 1995.

7 Appendix

7.1 Comparison of pseudocode, textual description and implementation

In this subsection we will compare the three different forms of AABL that are available to us:

Table 5: Possible combinations of color-type with the best recovery behaviors

Order	Color	Concept	Best Recovery Behavior
1	Red	Ball	Push
2	Red	Box	Alternative Route
3	Red	Person	Ask
4	Green	Ball	Push
5	Green	Box	Alternative Route
6	Green	Person	Ask
7	Blue	Ball	Push
8	Blue	Box	Alternative Route
9	Blue	Person	Alternative Route
10	Yellow	Ball	Push
11	Yellow	Box	Alternative Route
12	Yellow	Person	Ask
13	None	None	Continue

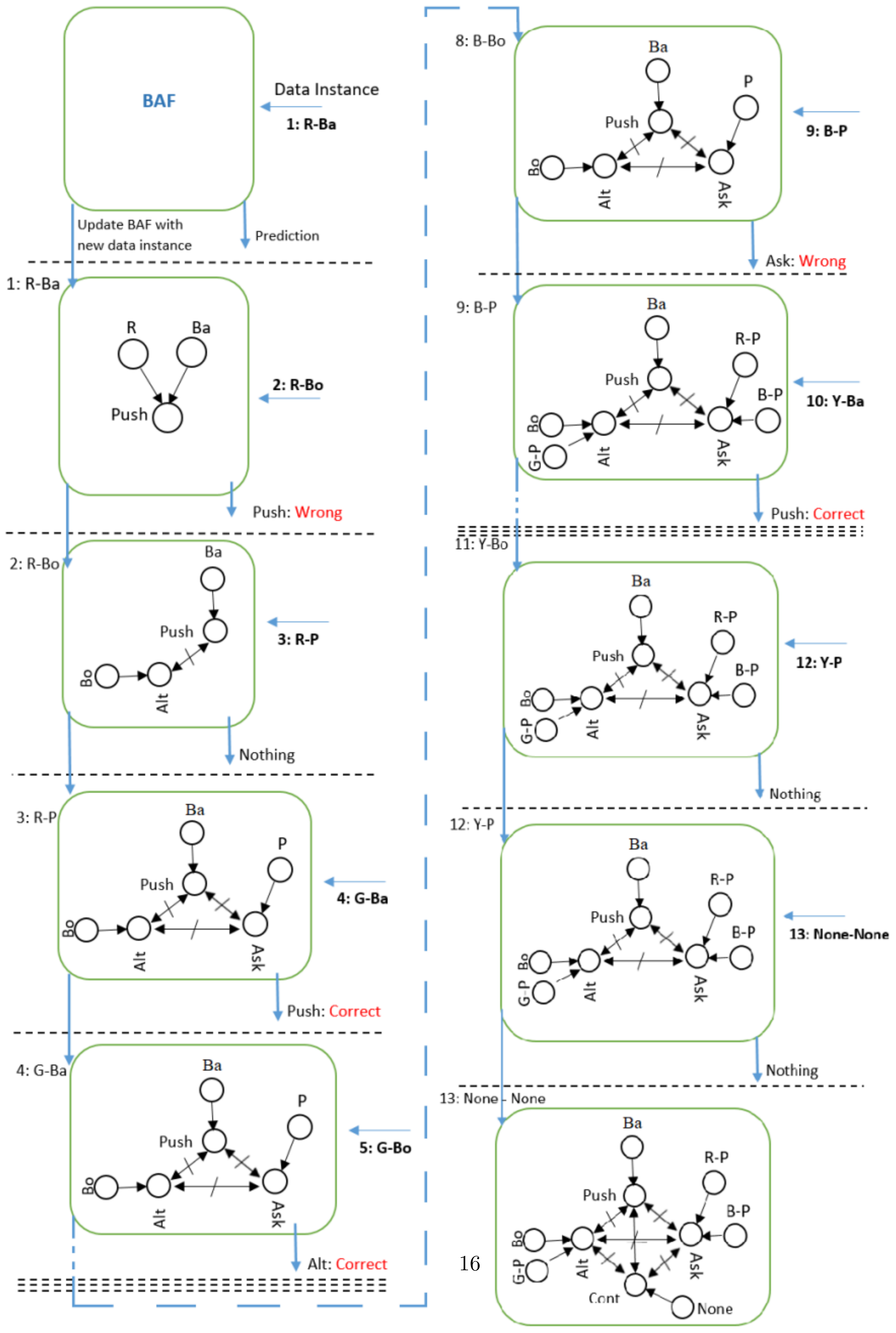
1. **The textual description -** the description of what AABL does in words as given in 3.4.1 of chapter 3 of [1] ”Explain what you see: argumentation-based learning and robotic vision” by Ayoobi

2. **The pseudocode -** the description of what AABL does in pseudocode as given in 3.4.2 of chapter 3 of [1] ”Explain what you see: argumentation-based learning and robotic vision” by Ayoobi

3. **The original implementation -** AABL as implemented by Ayoobi taken from his GitHub.

This section follows the linear example used in the textual description to compare these three versions of AABL. Since the workings of each version differ, we will sometimes need a tangent from the example in order to explain each method.

The example as given in 3.4.1 of chapter 3 of [1] is accompanied with the following table (5) and figure (6), taken directly from [1]: We implemented the scenario exactly as described in Table 5 to ensure that our theoretical understanding is the same as the practical implementation of AABL. Table 6 shows the theoretical guesses described in [1], the guesses made by AABL and RABL (they are the same), and lastly the Pseudocode (as well as the correct guesses presented in Table 5). A full walkthrough is provided in Section 7.1.1
The figure (6) supports the textual description as well as the pseudocode, as we shall see when walking through the textual description that the pseudocode and the textual description only differ in one aspect, namely: the pseudocode expects the correct answer to be handed as input while the textual description implies that AABL figures out the correct answer by trial and error. The original implementation does not correspond with the figure.



(a) Scenario 1

Figure 6: Example of Argumentation-Based Learning for the illustrating example

Table 6: Different guesses provided by the algorithms

Scenario	**Concept-Color**	**Predicted by [1]**	**AABL/RABL**	**Pseudocode**	**Correct**
1	Red-Ball	-	-	-	Push
2	Red-Box	Push	-	Push	Alt
3	Red-Person	-	Push	Random	Ask
4	Green-Ball	Push	Push	Push	Push
5	Green-Box	Alt	Alt	Alt	Alt
6	Green-Person	(correct, e.g. Ask)	Ask	Ask	Ask
7	Blue-Ball	(correct, e.g. Push)	Push	Push	Push
8	Blue-Box	(correct, e.g. Alt)	Alt	Alt	Alt
9	Blue-Person	Ask	Ask	Ask	Alt
10	Yellow-Ball	Push	Push	Random	Push
11	Yellow-Box	(correct, e.g. Alt)	-	Random	Alt
12	Yellow-Person	-	Alt	Random	Ask
13	None	-	Alt	Random	Continue

7.1.1 Walk-through comparison

The textual description in Section 3.4.1 ”Explanation of the Method with an Illustrating Example” from chapter 3 of [1] starts of with:

> We first use the simplified version of the test scenarios with only one location ahead of the agent (instead of 6, 9 or 12 locations). [...] the robot is initially confronted with a Red-Ball (R-Ba) and tries different recovery behaviors to find out that the best choice is Push.

However, in the pseudocode provided (Algorithm 1) we see that with an empty BAF graph the algorithm will reach line 18 and will randomly choose an observed recovery behavior. Since no recovery behavior has been observed yet, there will be no action. This corresponds with the original implementation: when the set of observed recovery behaviors is still empty, the original implementation immediately returns an empty string rather than trying out different recovery behaviors. Therefore on this point the pseudocode and implementation align while the textual description differs.

Instead of finding that the best choice is ’Push’, the original implementation is handed the correct recovery behavior together with the confronted scenario. Although this differs from the textual description, it matches the pseudocode: Algorithm 1 requires as input the Best Recovery Behavior (BRB) for X.

> The model initially gets updated by the subsets of feature values with size 1 (L := 1). This means that the supporting nodes R and Ba are added to the Push recovery behavior

This aligns with the pseudocode, at line 19 a new algorithm (Algorithm 2) is called, which adds the features R and Ba as support relations to the ’Push’ behavior.

However, in the original implementation no such thing happens. There is no data structure that tracks support relations, instead the features are appended as $[0, red]$ and $[0, ball]$ to a list called *subsets*, where 0 indicates the location of the object.

The recovery behavior 'Push' is uniquely added to a different list consisting of possible recovery behaviors. If the list of possible recovery behaviors were to already contain this recovery behavior then the recovery behavior is not added again: the original implementation does not track how often the recovery behavior is encountered. No relation between the subsets and recovery behaviors is tracked in any shape or form.

Instead, the the original implementation will now try to determine a list of *feature-locations* that could possibly be influencing the recovery behavior (In the original implementation the list of *possible-feature-locations* is called "*combination_feature_weights*", however we will refer to it as "*possible-feature-locations*"). To determine these *feature-locations* the algorithm requires as input a list of all previously encountered scenarios and the corresponding correct recovery behaviors. As the Red-Ball is the first scenario we encounter, this list is currently empty. Thus the list of *possible-feature-locations* stays empty as well. In the next paragraphs we will take a deeper look at how *possible-feature-locations* is updated and what the *feature-locations* entail.

> Subsequently, the agent encounters a Red-Box (R-Bo) for which the subsets of feature values with size L = 1 consist of R and Bo. Looking at the current state of the BAF, R supports the Push recovery behavior and it is chosen as the model's prediction.

This aligns with the pseudocode, at line 6 of Algorithm 1 the if-statement resolves to 'true' as '*red==red*' and thus 'Push' would be added to the possibly correct responses Y. Then, at line 12 and 13, 'Push' is applied to the scenario as Y contains only one recovery behavior.

In the original implementation there is no 'BAF' to look at, as the relations between 'Red', 'Ball' and 'Push' were not tracked. Instead the original implementation looks at the list of *possible-feature-locations* that could be influencing the correct recovery behavior. Currently this list of *possible-feature-locations* is empty.

As the list of *possible-feature-locations* is empty the algorithm again returns an empty string as recovery behavior. Thus on this point the original implementation again diverges from the pseudocode as well as the textual description.

> Since, this is a wrong choice, the agent try other recovery behaviors and find "Alternative route" (Alt) as the best recovery behavior. Therefore, the Alt node gets updated with its supporting nodes R and Bo and also a bidirectional attack among Alt and Push nodes. Since R supports both Push and Alt recovery behaviors, it is not a unique supporter for each of them and it will be pruned from both the recovery behaviors and will be marked as a node which can no longer support any recovery behavior nodes in the future

As pointed out already, neither the pseudocode nor the original implementation try other recovery behaviors. However, the pseudocode does indeed update the 'Alt' node with the supporting nodes 'R' and 'Bo' at line 27 of Algorithm 2. Next to that, 'R' is indeed pruned and marked at line 18 and 19 of Algorithm 2.

The original implementation instead empties the list of subsets and appends *[0, red]* and *[0, box]* to the list of subsets, which now consists of *[[0,red],[0,box]]* as *[0,red]* and *[0,ball]*, have been removed.

As the lists of previously encountered scenarios is now no longer empty, the algorithm will determine what *possible-feature-locations* could be influencing the correct recovery behavior. The list of previous scenarios with their corresponding recovery behaviors currently only contains one scenario, namely: *[Red, Ball, Push]* where Red is on column 0, Ball on column 1 and the recovery behavior on the last column.

We loop over each subset in *subsets*. The first subset is *[0, red]*, we take the location of this subset which is 0. When the subset contains a color we just keep the location as is, when it contains a concept we add 1 to the location. The algorithm will use these numbers to determine what column in the list of previous scenarios is important. In this case 0 corresponds with Red in *[Red, Ball, Push]*. We then count the amount of unique occurrences of features in this column as well as the amount of unique occurrences of features in combination with recovery behaviors. If we find that there is a difference between these counts then we find that the recovery behavior does not correlate with this column and we discard the column. However as the list only contains one instance we find that there is no difference between the counts and thus we add the column to the list *possible-feature-locations*, as it might be that this feature influences the correct recovery behavior.

When a column is added to *possible-feature-locations* a weight is calculated and added as well, the formula for this weight is:

$$\text{weight} = p + 1 + n * -50$$

Where p is the amount of non-unique feature-recovery occurrences

$$p = \text{TotalSelectedColumnsWithRecovery} - \text{TotalUniqueSelectedColumnsWithRecovery}$$

for this location and n is the the amount of times this object appears in this location in combination with a different recovery behavior.

$$n = \text{TotalUniqueSelectedColumnsWithRecovery} - \text{TotalUniqueSelectedColumnsWithoutRecovery}$$

The only mention of weights in the description of AABL [1] claims that the weights are removed, namely the author states " *the [total amount of] supporting weights and argument weights are reduced from 40 to 0.*". Thus there is no further explanation as to why this formula is chosen.

Note that we only add this weight in the case that there is **no difference** between the amount of unique occurrences of features in this column vs the amount of unique occurrences of features in combination with recovery behaviors, thus n will always equal 0 when adding the weight. Which means that

$$\text{weight} = p + 1$$

for every feature-location.

For *[0, red]* the weight will be 0 as p equals zero since there is only one previously seen scenario (thus the the amount of unique feature-recovery pairs is equal to the total of feature-recovery pairs).

The same thing happens for the next subset *[0, ball]*, as this subset contains a concept we look at column $location + 1 = 1$. Again there is no difference between the amount of unique items in this column compared to the amount of unique 'items+recovery behaviors'. Thus we add column 1 to the list of *possible-feature-locations* with a weight of 0 as well.

> For the third learning instance the robot is confronted with a Red-Person (R-P) and the models does not have any prediction since no current recovery behavior node in the BAF has either P or R in its supporting nodes

This corresponds with the pseudocode as we saw in the previous paragraph that R was indeed pruned, next to that P has not been encountered yet. However for the original implementation, there is no BAF but instead we now have two entries in our list of *possible-feature-locations*: column 0 and 1, corresponding to the color and concept of first object respectively.

The original implementation loops over these *possible-feature-locations* and selects the first highest weight it encounters. Since both feature-locations in the list have equal weight, the algorithm selects the first column it encountered with the highest weight, which in this case is column 0. The algorithm then takes the feature associated with this location from the state, which is 'Red' as the current state contains *[Red, Person]*. The algorithm then counts how often a recovery behavior is seen together with this feature in location 0. As we saw a Red-Ball and a Red-Box with recovery behaviors Push and Alt respectively, we get 1 for Push and 1 for Alt. If there is no single recovery behavior with the highest count, the algorithm instead returns the recovery behavior that has been most observed **overall**. As there's only been two observations: Push and Alt, the first one will be returned. Thus finally the algorithm wrongly predicts 'Push'.

> The BAF unit gets updated with only P as a supporting node for the Ask since R has been previously marked as a non-supporting node and bidirectional attack relations are added among all pairs of the recovery behaviors

This aligns with the pseudocode. However, the original implementation instead will update the *possible-feature-locations* which for now contains *[0,1]* regarding column 0 and 1 respectively.

The list of subsets gets emptied and updated to *[[0, Red],[0, Person]]*. The list of previously seen scenarios and their respective recovery behaviors will now be: $[[Red, Ball, Push], [Red, Box, Alt]]$.

We loop over the subsets and start with $[0, Red]$, converting this instance to column 0. We then look in the previous scenarios and count the unique occurrences of features in this column: which is only 1, Red as this is the feature we saw in both previously seen scenarios. Then we count the amount of unique occurrences of features in combination with the respective recovery behaviors, which is 2 as Red occurs together with Push as well as Alt. Since these counts now differ, we remove column 0 from our list *possible-feature-locations.*

The second subset *[0, Person]* translates to column 1. We do the same counting and find two different features, namely *Box* and *Alt.* Since there is no difference between the amount of unique occurrences of features in this column vs the amount of unique occurrences of features in combination with recovery behaviors, this column stays in the list of *possible-feature-locations* with a weight of 0, now being the only occurrence in this list.

Subsequently, the agent encounters with a Green-Ball (G-Ba) obstacle and since Ba supports the Push in the BAF unit, Push is chosen as a prediction for the best recovery behavior.

As the list of *possible-feature-locations* only contains column 1, we take the feature associated with this column for this scenario, which is 'Ball'. We count the recovery behaviors associated with this feature as they are found in column 1 for all previously encountered scenarios. Since only the *[Red, Ball, Push]* had the '*Ball*' feature in column 1, '*Push*' is correctly predicted as the best recovery behavior.

The BAF gets updated using G supporting node for Push recovery behavior.

The list of subsets gets emptied and updated to *[[0, Green],[0, Ball]]*. The list of previously seen scenarios and their respective recovery behaviors will now be:

$$[[Red, Ball, Push], [Red, Box, Alt], [Red, Person, Ask]]$$

We loop over the subsets and start with [0, Green], converting this instance to column 0. We then look in the previous scenarios and count the unique occurrences of features in this column: 1 as we so far have only encountered 'Red'. Then we count the amount of unique occurrences of features in combination with the respective recovery behaviors, which is 3 as Red occurs with 3 different recovery behaviors. Since both counts differ we do not add column 0 to our list of *possible-feature-locations.*

The second subset is [0, Ball] which converts to column 1. The amount of unique occurences of features in this column is 3 as we so far have seen a 'Ball' a 'Box' and a 'Person'. The amount of unique occurences in combination with the respective recovery behavior is also 3 as each concepts occurs with a different recovery behavior. Since both counts are equal we add column 1 to our list of *possible-feature-locations.*

We then calculate the weight for column 1, there are a total of 3 rows in the previous scenarios, each of which is unique thus the weight equals 0.