Analyzing the Impact of Search Orders in Extended State Machine Learning for Anomaly Detection

Sam van Beek
Responsible Professor: Dr.ir. S.E. (Sicco) Verwer
EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 21, 2026

Name of the student: Sam van Beek
Final project course: CSE3000 Research Project
Thesis committee: Sicco Verwer, Kubilay Atasu

Abstract

Anomaly detection is very important in this day and age. Finding anomalies in software can help fix bugs or prevent malicious intent from third parties. One way to detect these anomalies is by making use of pdfa's. Pdfa learning an extension of dfa learning can be used to construct a model of an application given some logs of the application. To build a pdfa, a probabilistic deterministic finite automata, various algorithms in combination with extra parameters can be used. In this paper we will use the FlexFringe framework to look into these so called search orders to answer the question: *What are good search orders in extended state machine learning for anomaly detection?* We looked into the Alergia and RTI+ algorithm on data with and without time information. Our main conclusions are that Alergia together with the blueblue parameter works best for creating a model for anomaly detection.

1 Introduction

Anomaly detection is very important in this day and age. Finding anomalies in software can help fix bugs or prevent malicious intent from third parties. One way to detect these anomalies is by making use of pdfa's. Pdfa's or probabilistic deterministic finite automata are a variant of dfa's that instead of accepting or rejecting a sequence will return a probability. The FlexFringe framework made by Sicco Verwer and Christian Hammerschmidt [4] is a framework which implements multiple different algorithms to construct these pdfa's. They show that given traces/logs from an application a pdfa can be build which acts as a sort of model of the application. This model can then be used to find anomalies which is also shown by Serentellos [3] who did a master thesis on *Anomaly Detection in Network Traffic using Multivariate State Machines* in which he managed to detect malware in network traffic with a low false positive rate. Other related work is the original paper on the Alergia algorithm by Carrasco and Oncina [1]. And the paper on the RTI+ algorithm by Verwer, Weerdt, and Witteveen [5]. These algorithms will be discussed briefly in the preliminaries. Although both the paper by Serentellos [3] and Verwer and Hammerschmidt [4] have shown pdfa's to be useful for anomaly detection neither of them has fully researched how well we can make these pdfa's by tweaking parameters of various different algorithms. We call these modified algorithms: search orders. Looking more into these search orders and how they function could result in creating even better models.

The main question we will answer in this paper is: *What are good search orders in extended state machine learning for anomaly detection?* To this end we will be running various search orders on the HDFS_v1 dataset found on loghub [2, 6, 7]. This question will be answered using various sub-questions.

1. What are good search orders for the Alergia algorithm when ran on untimed data.
2. What are good search orders for the RTI+ algorithm when ran on untimed data.
3. What are good search orders for the RTI+ algorithm when ran on timed data.

To answer these sub-questions we will run various search orders to generate pdfa's which will then be tested on performance giving an indication of which parameter configurations perform well and which do not. With these results it can become easier to create good performing models since there is more insight into what algorithms perform well.

The report will be structured as follows. Chapter 2 will contain the most important background information, definitions, algorithms and parameters. In chapter 3 the general setup of the experiments will be explained. In chapter 4 experiments are explained followed by chapter 5 which will showcase the results. Chapter 6 will contain a breakdown of the ethical aspects and reproducibility of the research. Chapter 7 will give a discussion of the results followed by the conclusions in chapter 8.

2 Preliminaries

This chapter will contain some basic definitions followed by a brief overview of the algorithms we will be using. After that we will explain the parameters FlexFringe provides. Basic knowledge of normal dfa's and how they function is expected.

2.1 Basic definitions

- **Traces:** traces are sequential data that a system may keep track of during it's processes. The most basic trace consists of an id, a list of symbols and a label. The extension of this is keeping track of an extra value for every symbol in the trace. This can for example be time. We refer to data without the time as untimed and with the time as timed.
- **Pdfa's:** probabalistic deterministic finite automata are a variant of dfa's where instead of either accepting or rejecting a sequence they instead give the probability of getting that given sequence when run on the pdfa.
- **Epdfa's:** extended pdfa's are pdfa's that also use guards for the transitions. This means that only if the guard condition is met the edge can be taken. See figure 1 below.

2.2 FlexFringe

The FlexFringe framework made by Sicco Verwer and Christian Hammerschmidt [4] is a framework which implements multiple different algorithms to construct pdfa's. Given traces/logs from an application a pdfa can be build which acts as a sort of model of the application. This is done using the red-blue framework. It starts by building a prefix tree from the given data. It colors the root node red and it's children blue. It then computes a score for all merges and splits between red and blue nodes. Where a merge merges to states together and a split splits an edge into two edges with guard conditions on both of them. How the scores of these merges and splits are computed depends on the algorithm used. If none of the merges meet the lower-bound and none of the splits meet the splitting upper-bound then we will extend thus coloring the first blue node red. Otherwise it will conduct the best merge/split. After an extend, merge or split any new children of red nodes will be colored blue and the process repeats until all states are red. An example of a prefix tree, merge, split and extend are shown in appendix A.

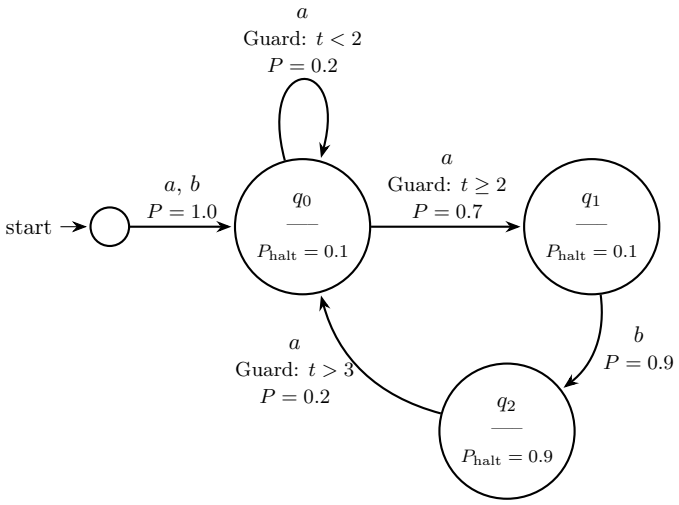


Figure 1: Example of an epdfa showcasing how guards and transitions work together with the probabilities. The inside of nodes give the probability of ending in that node. A transition can only be taken if it adheres to both the symbol and the guard condition. It remains deterministic because there is no overlap in the guards. Transitions not shown will end up in the dead state. Which has probability 0.0.

2.3 Algorithms

2.3.1 Alergia

The Alergia algorithm makes use of the Hoeffding bound to compute the quality of a merge. [1, 4] A general interpretation is that if the future distribution of 2 states are similar than they can be merged. For Alergia you can use the Alpha parameter to indicate how similar 2 states need to be to be merged.

2.3.2 RTI+

The RTI+ algorithm uses a different metric to compute the score. [5]. To calculate if a merge is good it uses the likelihood-ratio test. But RTI+ was also designed with splitting in mind. It uses the inverse of the log-likelihood test to compute how good a split is. To determine how good a split/merge has to be can be tuned by tweaking the significance level parameter.

2.4 Parameters

Below are some parameters which the FlexFringe framework provides to further specify the way you want the search order to go [4]. These parameters are interesting cause they change the way it builds the pdfa.

- **finalprob:** the finalprob parameter when enabled will transform the generated pdfa to also take ending probabilities into account. This means that the probability of ending in a state plus the probabilities of all outgoing transitions will together sum up to one instead of the outgoing transitions summing up to one. When running predict mode

this parameter also incorporates the ending probability for the final state making it well suited to detect traces that end prematurely.

- **confidence bound:** the confidence bounds functionality depends on the algorithm being used. For the Alergia algorithm the confidence bound is the alpha value and for RTI+ it's the significance level. Changing the confidence bound will allow us to tune how generalized we want the pdfa to be.
- **shallowfirst:** when shallowfirst is enabled the blue states are sorted on depth. The nodes with the lowest depth will get the most priority. Enabling this parameter will allow us to see if merges to nodes closer to the start will result in a better pdfa in the end.
- **largestblue:** when largestblue is enabled only the highest scoring blue node is compared to all red states instead of comparing all blue states to all red states. This drastically decreases the runtime of the algorithm. It will be interesting to see how much it affects the final quality of the pdfa.
- **blueblue:** when enabled allows blue states to merged with other blue states. Which means the selected blue node(s) will be compared to all red and blue nodes instead of only the red ones. This will increase the runtime but it will look at more options making it possibly perform better.
- **redfixed:** when enabling redfixed, red states are final and unmodifiable. This means that if an edge is added when merging the merge is considered inconsistent. It also entails that the values of the outgoing edges will not be changed when a merge is conducted. Enabling this parameter tends to give more insightful models.

3 Experimental Setup

To answer the research question: *What are good search orders in extended state machine learning for anomaly detection?* We will be building a binary classifier which distinguishes between two cases: anomalies or normal traces. To determine the quality of a binary classifier we will use the ROC-AUC. The Receiver Operating Characteristic - Area Under the Curve or ROC-AUC is a metric used to classify how well a binary classifier segments the data into 2 distinct groups. A ROC-AUC of 0.5 means it performs equal to random whereas a ROC-AUC of 0.9 means it performs really good. Using this metric we will get a good idea of how well the pdfa performs as a classifier and thus how good it is at anomaly detection. In this chapter we will explain the pipeline used to get the classifier. We start by explaining the HDFS dataset [2, 6, 7] which was used to conduct the experiments. After that we discuss how the test and training sets are made. Followed by an explanation of how we generate and run the automata. And finally we mention the values we will use as the classifier.

3.1 HDFS dataset

For the experiments the HDFS_v1 dataset was used [2, 6, 7]. The Hadoop Distributed File System (HDFS) logs are very useful for testing and training the automata. The data is labeled making the distinction between sequences of events that are anomalies and sequences that are not. Below is an example of a log as seen in table 1. As you can see it contains a

BlockId indicating that all events are part of that block. The Label indicates either Fail or Success with Fail being an anomaly. Features is a list of events Labeled from E1 up to E29. These lists of events will be the symbols we use to train an automata using FlexFringe [4]. After that TimeInterval contains the time intervals between the events so when you have n events you will have n-1 time intervals. These will be useful for the extended automata we are going to be building later on. We ignore the Type and Latency of each block since they are irrelevant for the testing we are doing.

Table 1: Example of a HDFS_v1 trace which includes the id, the Label which is either Success or Fail, the sequence of events and the time intervals between those events. The Type and Latency are irrelevant for our use cases.

BlockId	**Label**	**Type**	**Features**	**TimeInterval**	**Latency**
blk_-310	Fail	16	[E5,E5,E22,E7]	[0.0, 60.0, 657.0]	718

3.2 Building test and train datasets

When creating the training and testing sets we first separate the data to make sure we are only training on normal traces and not on anomalies. The distribution of the HDFS_v1 dataset [6] is 97.07% normal traces and 2.93% anomalies. Since the amount of anomalies is very low we decided to use only normal traces to train to not shrink our testing set any further.

We create 10 training sets of 10.000 traces each and 1 testing set of 28.000 traces. Where the training sets contains only normal traces whereas the testing set consist of 50% anomalies and 50% normal traces. By dividing the testing set 50/50 we will be able to better see how anomalies perform when compared to normal traces.

When creating these training and testings sets we also make the distinction between timed and untimed data. Untimed data will not contain time information whereas timed data will. This means that when we convert the data to a FlexFringe parsable format. Which means splitting the trace over multiple lines as seen in table 2. The untimed version will not have the column containing ff_sattr:s/val which normally contains the time information.

Table 2: Example of the HDFS_v1 trace above converted to the timed FlexFringe format which includes the id of the trace, the current event, the label which is either 1 for normal or 0 for anomaly traces and the time since the last event in the trace.

ff_id	**ff_symb**	**ff_ttype**	**ff_sattr:s/val**
1	5	0	0.0
1	5	0	0.0
1	22	0	60.0
1	7	0	657.0

3.3 Generate and running automata

We run the algorithm to build an automata for every one of the 10 training sets. We do this by running FlexFringe with the ini file containing the algorithm. We do this because there is a lot of variance in the complexity, size and performance between the automata when they are generated using different data. By running the algorithm on multiple training sets we circumvent this issue by looking at the overall effect the algorithms have. After all automata are generated they are then run on the test set using FlexFringe's predict mode [4]. This predict mode will give various values from which we found 3 to be interesting. Below are the metrics we looked at when ran on the test set. Their distributions are shown in figure 2.

1. **Transition Mean:** The mean of the scores of the transitions taken. It doesn't work well as an anomaly detection metric since a very long trace with a lot of standard scores and one really bad one will still have a normal mean.
2. **Transition Sum:** The sum also isn't good at detecting anomalies since it heavily depends on the length of a trace. Whereas anomalies can be long or short.
3. **Transition Minimum:** The minimum transition taken does work well as an anomaly detection metric since it shows if a very unlikely transition was taken. When the finalprob parameter is enabled it also adds an extra probability to the final transition to account for the state it ends in making it very good at detecting traces that end prematurely.

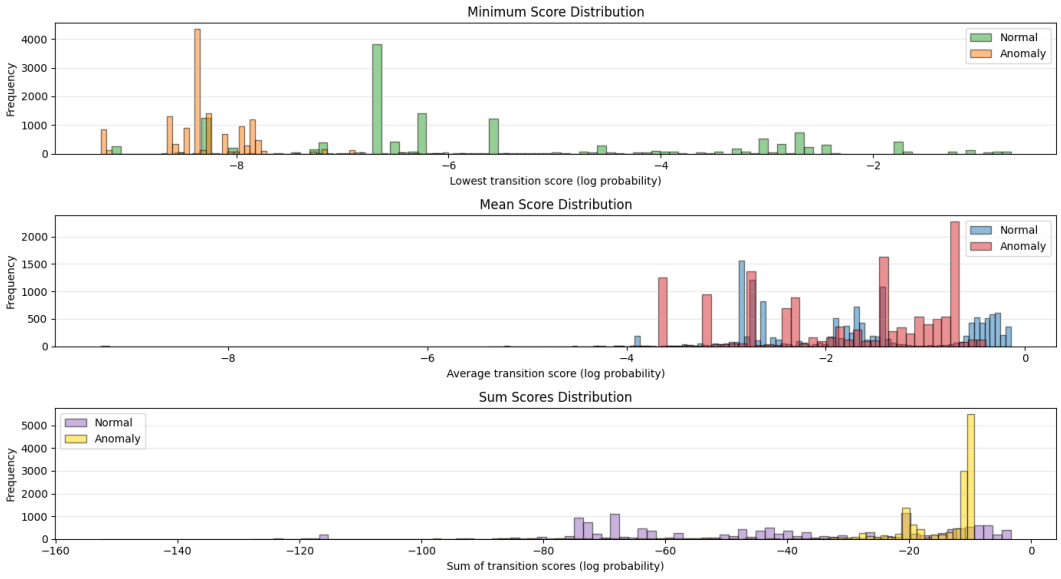


Figure 2: Distribution of the Minimum Score, Mean Score and Sum Scores values when run on the test set. It shows that Minimum Score is best at separating the data.

We will use the Minimum Score as the classifier to separate the anomalies from normal traces. We calculated the ROC-AUC to get the final value to determine how well a pdfa performs.

4 Experiments

To answer our research question we will first be answering our sub-questions using 2 experiments per question. We will first determine the optimal confidence bound for a given algorithm to get a general solution. Then in the following experiment we try the different parameter configurations to see how it effects the performance as a classifier. Below is a list with a small description of all experiments. In appendix B the tables are shown for the parameters used in experiment 2, 4 and 6.

1. **Experiment 1:** For the first experiment we will be running the Alergia algorithm using various confidence bounds (alpha values) on untimed data. From the range [1e-7, 1e-1]. We decided on this range since both size and AUC performance only shows significant difference across such a large range.

2. **Experiment 2:** For the second experiment we will be running the Alergia algorithm on untimed data using the optimal confidence bound found in experiment 1 in combination with various parameters.

3. **Experiment 3:** For the third experiment we will instead be looking at the RTI+ algorithm to see what confidence bound it performs best with.

4. **Experiment 4:** For the fourth experiment we will be trying the different parameters together with the best confidence bound found in experiment 3.

5. **Experiment 5:** For the fifth experiment we will once again be trying to find the optimal confidence bound for the RTI+ algorithm, but this time we will be using timed data. We will try the confidence bounds in the range [1e-8, 1e-5]. The reason this range is shorter than the other ones is because using higher confidence bounds will make the algorithm take far too long to run without largestblue enabled.

6. **Experiment 6:** For the sixth experiment we will once again be looking at the different parameters when used with the confidence bound found in experiment 5.

5 Results

In this chapter we will show the results and briefly discuss what they mean and what conclusions we can draw from them. A full overview of all results can be found in appendix C.

5.1 Experiment 1

When determining the optimal confidence bound for the Alergia algorithm the thing that we looked for is the one with the highest AUC score on average. You can see when looking at the AUC scores in figure 3 that they increase until a certain point and then they drop again. This is also reflected in the size shown in figure 4 since it is the point where the size doesn't make large jumps any more.

The best performing confidence bounds are 1e-6, 1e-5 and 1e-4 when looking purely at the AUC score, but when also looking at size 1e-6 is significantly smaller than the other two so we decided to use 1e-6 as the confidence bound for experiment 2.

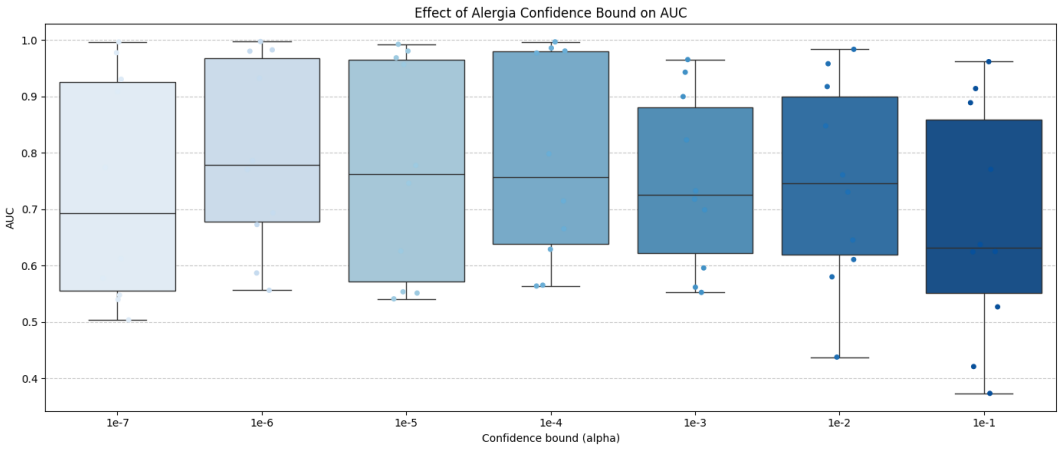


Figure 3: ROC-AUC values of the Alergia algorithm ran on untimed data with various confidence bounds as described in experiment 1.

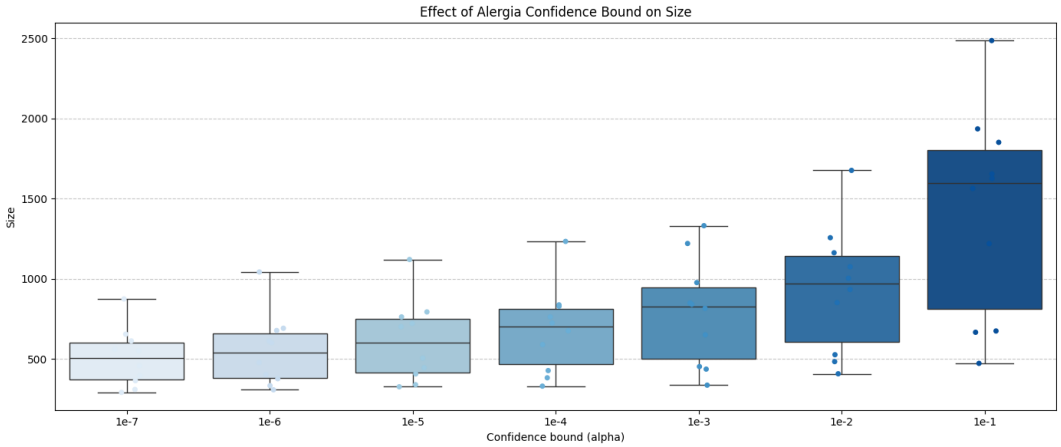


Figure 4: Size values of the pdfa's generated by the Alergia algorithm ran on untimed data with various confidence bounds as described in experiment 1.

5.2 Experiment 2

When looking at the AUC values there are only a few parameter configurations that perform equal or better than base Alergia (A0) which is shown in figure 5. These include A1, A4, A7 and A9. The other configurations perform significantly worse. Both A1 and A7 both perform better than A0. Where A1 has blueblue and shallowfirst enabled and A7 has both blueblue and shallowfirst enabled. It seems blueblue by expanding the amount of options during the merging stage does improve the actual performance of the final pdfa.

Largestblue which is enabled in A2 and together with shallowfirst in A9 performs significantly worse on it's own than if combined with shallowfirst. This is to be expected since largestblue significantly reduces runtime at the cost of only comparing a single blue node instead of all blue nodes. shallowfirst does seem to improve the performance by quite a significant margin and despite it not seeming to do much when combined with other parameters.

You can see that redfixed which is enabled in A3, A6, A8, A10, A11, A13, A14 and A15 significantly tanks the performance. By not allowing edges to be transformed it doesn't create the more generalized models needed for proper classification.

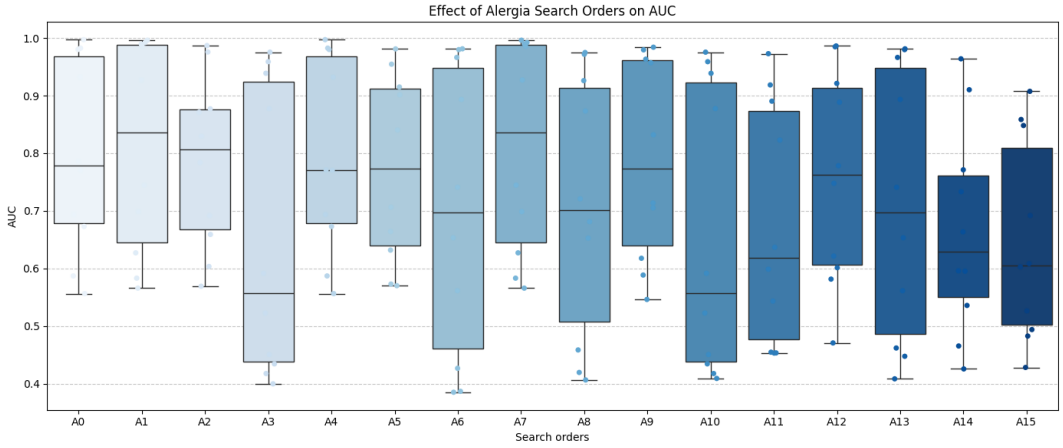


Figure 5: ROC-AUC values of the Alergia algorithm ran on untimed data with various different parameter configurations as described in experiment 2. The parameters used can be found in table 3 in appendix B.

5.3 Experiment 3

When looking at the RTI+ algorithm we use the same approach as for the Alergia algorithm this showed 1e-6 and 1e-5 to be roughly equal in performance. We decided to continue with 1e-6 since the generated pdfa's are significantly smaller than those generated by 1e-5. This is shown in figure 17 and 18 in appendix C.

5.4 Experiment 4

In contrast to how the parameters affected the Alergia algorithm they do seem to have a slightly different effect on the RTI+ algorithm. The most significant change from the Alergia

algorithm search order variations is that the different parameter configurations seem to give far more extreme outliers. As you can see in figure 6. Almost all algorithms that are not base RTI+ seem to have multiple pdfa's that have AUC values lower than 0.5. This means they perform worse than random if used as a classifier.

Another interesting difference is that largestblue seems to perform much better in combination with RTI+. It does make the spread of results significantly larger as seen in R2. It outperforms base RTI+, but performs worse when combined with other parameters such as blueblue (R5) or shallowfirst (R9).

The rest of the results are similar as they were for Alergia. Blueblue makes it perform better as seen in R1. Shallowfirst doesn't seem to make any significant difference and redfixed ruins the performance.

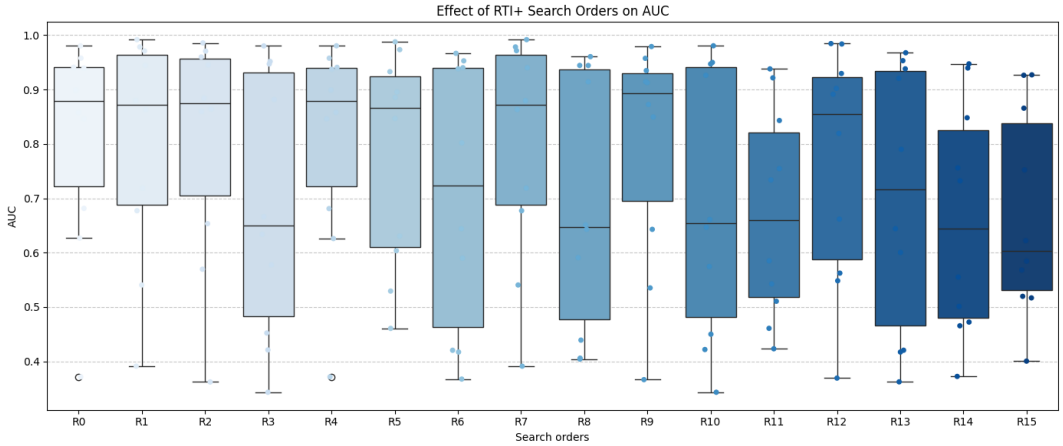


Figure 6: ROC-AUC values of the RTI+ algorithm ran on untimed data with various different parameter configurations as described in experiment 4. The parameters used can be found in table 4 in appendix B.

5.5 Experiment 5

For RTI+ on timed data we once again repeated the procedure of trying to find the optimal confidence bound. We expected the confidence bound to be around the same as for untimed RTI+ so we looked around that range expecting the same confidence bound to be optimal. This was not the case since the optimal confidence bound we found for timed RTI+ is 1e-7 instead of 1e-6 as shown in figure 21 in appendix C.

5.6 Experiment 6

When looking at the results for the different search orders they are relatively similar to those of untimed RTI+. But it seems to have even worse performance for some configurations given AUC score's as low as 0.3 as shown below in figure 7.

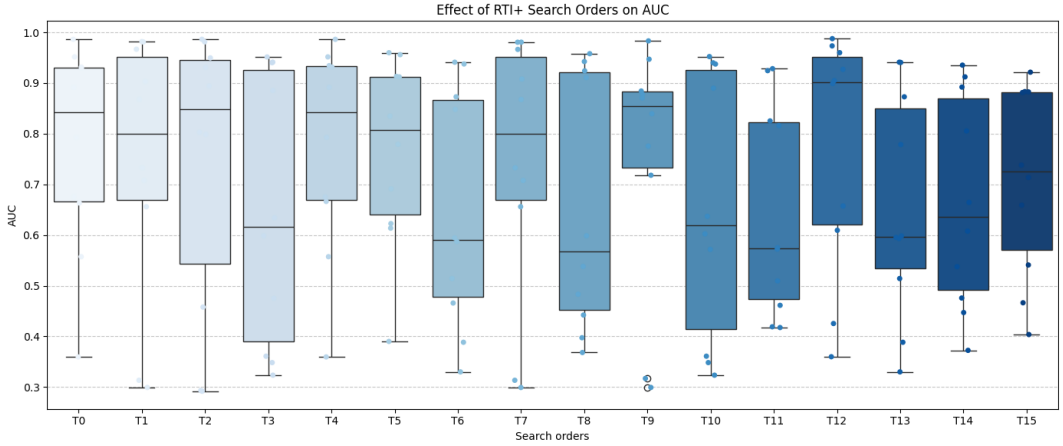


Figure 7: ROC-AUC values of the RTI+ algorithm ran on timed data with various different parameter configurations as described in experiment 6. The parameters used can be found in table 5 in appendix B.

6 Responsible Research

6.1 Reproducibility

To ensure that the research is both reproducible and repeatable we have documented the full setup and pipeline used in the experimental setup. All data and software used is open-source and can be used to recreate the experiments. The code we used which includes the parameters for all experiments and the code to create the plots can also be found at https://github.com/sam24351/Research_project.

6.2 AI usage

We did use AI for certain aspects of the research. Gemini was used to help with the generation of code. The code generated was mostly used to help create plots and visualizations. The structure and layout of certain tables and figures were generated using AI, but the content was made by hand.

7 Discussion

7.1 Algorithms

Overall we found the results to be quite surprising when looking at the best base performance and the best overall performance. Alergia was most successful in creating highly performative pdfa's as shown in figure 8. Which came as a surprise since you would expect timed automata to excel at anomaly detection since weird timings are often an indication of foul play. But timed RTI+ also performed worse than the untimed version. We also tested how RTI+ performed when only looking at the timing and not the symbols of the traces. This resulted in a AUC of around 0.90 as shown in figure 25 in appendix D. So the timings can be used to detect anomalies.

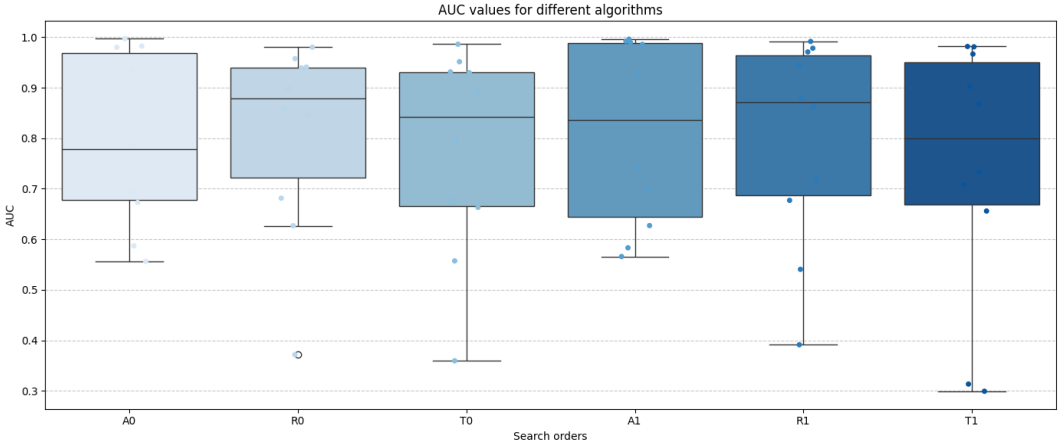


Figure 8: Comparison of the best performing algorithms for Alergia untimed, RTI+ untimed and RTI+ timed. A0, R0 and T0 are the best performing base algorithms and A1, R1 and T1 are the same algorithms but with blueblue enabled.

7.2 Confidence bounds

The trend which the confidence bound seems to show for untimed data is that once the size of the automata doesn't decrease that much the optimum has been reached for the performance since further merges will be so different they will harm the quality of the automata. When looking at the performance of the automata the most indicative characteristic seems to be the average of the amount of outgoing edges per node. Good performing pdfa's getting an AUC score higher than 0.95 have an average of around 14 edges per node whereas lower scoring algorithms tend to have an average around 8. This is shown in figure 26 in appendix D.

7.3 Parameters

The parameters can also be explained by looking at the node density of the pdfa. When enabling blueblue we see a direct increase in the node density which results in better pdfa's. Shallowfirst and largestblue do not show any significant change which is also reflected in the AUC values and finally redfixed heavily decreases the node density making the AUC score significantly worse. You can see these effects in figures 27, 28, 29 and 30 in appendix D.

8 Conclusions and Future Work

The main question this paper answers is: *What are good search orders in extended state machine learning for anomaly detection?* To answer this question we created a pipeline to build a model of an application which can be used as a binary classifier to detect anomalies. These models are created using different search orders were search orders are algorithms in combination with some extra parameters. We created the classifier using the FlexFringe framework developed by Verwer and Hammerschmidt [4] and tested it on the HDFS_v1 dataset [2, 6, 7].

To answer the research question we looked into 3 different algorithms: Alergia on untimed data, RTI+ on untimed data and RTI+ on timed data. We tried running these algorithms with different parameters enabled to see how they affect performance of the final classifier. We concluded that Alergia on untimed data was the best performing one with some classifiers even getting a ROC-AUC score as high 0.997. RTI+ on untimed data performed a bit worse and the worst performing one was RTI+ on timed data. As for the parameters used we found that the only parameter which seems to significantly improve performance is the blueblue parameter which increases the runtime but makes more considerations when creating the model.

It was very interesting that the RTI+ algorithm on untimed data outperformed the one on timed data. The timing does contain information since building a classifier on only the timings and not the symbols did provide decent results as shown in figure 25. Future work may include looking more into this. Testing on different datasets could give more insight into timed RTI+ to see if timed always performs worse or if this was a dataset specific case.

References

[1] Rafael Carrasco and Jose Oncina. "Learning Stochastic Regular Grammars by Means of a State Merging Method". In: Nov. 2002. ISBN: 978-3-540-58473-5. DOI: 10.1007/3-540-58473-0_144.

[2] Zhihan Jiang et al. *A Large-Scale Evaluation for Log Parsing Techniques: How Far Are We?* 2024. arXiv: 2308.10828 [cs.SE]. URL: https://arxiv.org/abs/2308.10828.

[3] Vasileios Serentellos. *Anomaly Detection in Network Traffic using Multivariate State Machines*. 2020. URL: https://repository.tudelft.nl/file/File_650384ca-986c-434e-8486-d51abc997505?preview=1.

[4] Sicco Verwer and Christian Hammerschmidt. "FlexFringe: Modeling Software Behavior by Learning Probabilistic Automata". In: *Logical Methods in Computer Science* Volume 21, Issue 3 (2025). ISSN: 1860-5974. DOI: 10.46298/lmcs-21(3:31)2025. URL: http://dx.doi.org/10.46298/lmcs-21(3:31)2025.

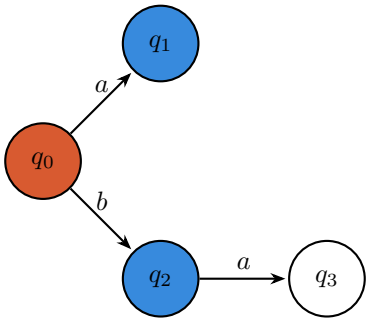
[5] Sicco Verwer, Mathijs Weerdt, and Cees Witteveen. "A Likelihood-Ratio Test for Identifying Probabilistic Deterministic Real-Time Automata from Positive Data". In: vol. 6339. Sept. 2010, pp. 203–216. ISBN: 978-3-642-15487-4. DOI: 10.1007/978-3-642-15488-1_17.

[6] Wei Xu et al. "Detecting Large-Scale System Problems by Mining Console Logs". In: *Proc. of the 22nd ACM Symposium on Operating Systems Principles*. SOSP '09. 2009. URL: https://people.eecs.berkeley.edu/~jordan/papers/xu-etal-sosp09.pdf.

[7] Jieming Zhu et al. *Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics*. 2023. arXiv: 2008.06448 [cs.SE]. URL: https://arxiv.org/abs/2008.06448.

A Red-blue framework

A.1 Prefix tree

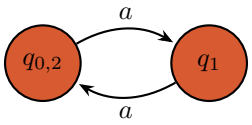


(a) Prefix tree

Figure 9: This figure showcases a prefix tree generated from the strings a, b, ba. The root node is colored red and it's children blue.

A.2 Merge

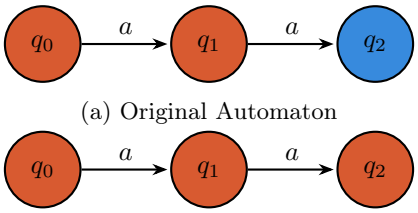
(a) Original Automaton



(b) Automaton after merge

Figure 10: This figure showcases a blue state q_2 being merged with a red state q_0. All pointers pointing to q_0 will now point to q_2 resulting in a loop.

A.3 Extend

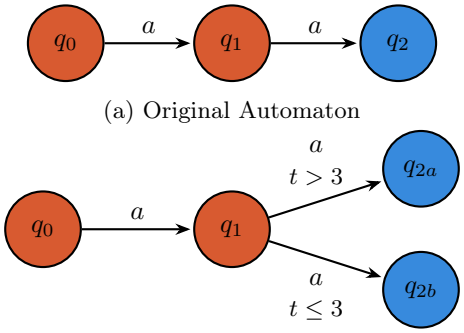


(a) Original Automaton

(b) Automaton after extend

Figure 11: This figure showcases an extend which means the highest ranking blue node will be colored red which in this case is q_2.

A.4 Split



(a) Original Automaton

(b) Automaton after split

Figure 12: This figure showcases a split which splits an edge into 2 edges with guards on them. In this case if the time is greater than 3 you will end up in q_{2a} otherwise you end up in q_{2b}.

B Search orders

Search order	confidence bound	blueblue	largestblue	redfixed	shallowfirst
A0	1e-6	×	×	×	×
A1	1e-6	✓	×	×	×
A2	1e-6	×	✓	×	×
A3	1e-6	×	×	✓	×
A4	1e-6	×	×	×	✓
A5	1e-6	✓	✓	×	×
A6	1e-6	✓	×	✓	×
A7	1e-6	✓	×	×	✓
A8	1e-6	×	✓	✓	×
A9	1e-6	×	✓	×	✓
A10	1e-6	×	×	✓	✓
A11	1e-6	✓	✓	✓	×
A12	1e-6	✓	✓	×	✓
A13	1e-6	✓	×	✓	✓
A14	1e-6	×	✓	✓	✓
A15	1e-6	✓	✓	✓	✓

Table 3: Search order matrix for experiment 2

Search order	confidence bound	blueblue	largestblue	redfixed	shallowfirst
R0	1e-6	×	×	×	×
R1	1e-6	✓	×	×	×
R2	1e-6	×	✓	×	×
R3	1e-6	×	×	✓	×
R4	1e-6	×	×	×	✓
R5	1e-6	✓	✓	×	×
R6	1e-6	✓	×	✓	×
R7	1e-6	✓	×	×	✓
R8	1e-6	×	✓	✓	×
R9	1e-6	×	✓	×	✓
R10	1e-6	×	×	✓	✓
R11	1e-6	✓	✓	✓	×
R12	1e-6	✓	✓	×	✓
R13	1e-6	✓	×	✓	✓
R14	1e-6	×	✓	✓	✓
R15	1e-6	✓	✓	✓	✓

Table 4: Search order matrix for experiment 4

Search order	confidence bound	blueblue	largestblue	redfixed	shallowfirst
T0	1e-7	×	×	×	×
T1	1e-7	✓	×	×	×
T2	1e-7	×	✓	×	×
T3	1e-7	×	×	✓	×
T4	1e-7	×	×	×	✓
T5	1e-7	✓	✓	×	×
T6	1e-7	✓	×	✓	×
T7	1e-7	✓	×	×	✓
T8	1e-7	×	✓	✓	×
T9	1e-7	×	✓	×	✓
T10	1e-7	×	×	✓	✓
T11	1e-7	✓	✓	✓	×
T12	1e-7	✓	✓	×	✓
T13	1e-7	✓	×	✓	✓
T14	1e-7	×	✓	✓	✓
T15	1e-7	✓	✓	✓	✓

Table 5: Search order matrix for experiment 6

C Results

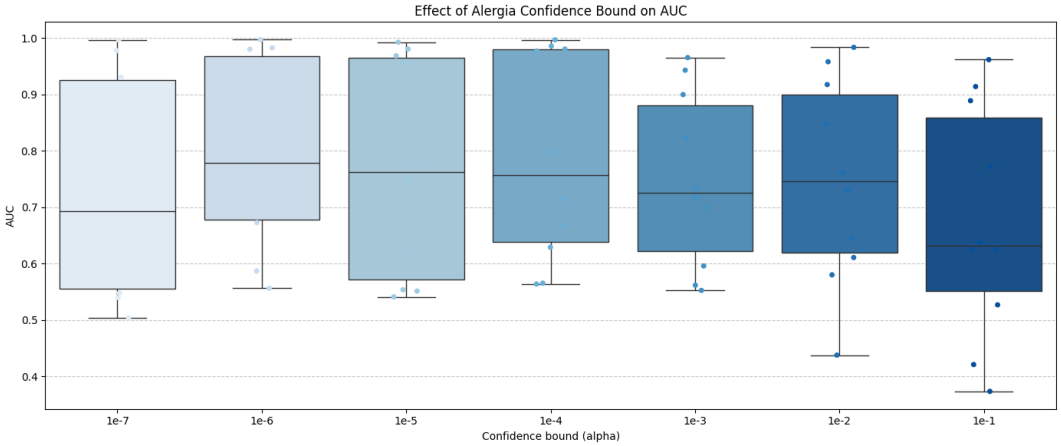


Figure 13: ROC-AUC values of the Alergia algorithm ran on untimed data with various confidence bounds as described in experiment 1.

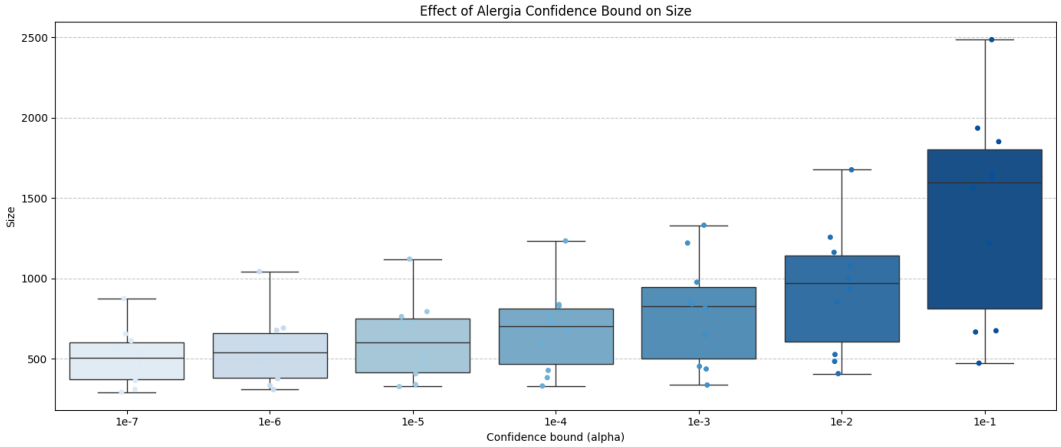


Figure 14: Size values of the pdfa's generated by the Alergia algorithm ran on untimed data with various confidence bounds as described in experiment 1.

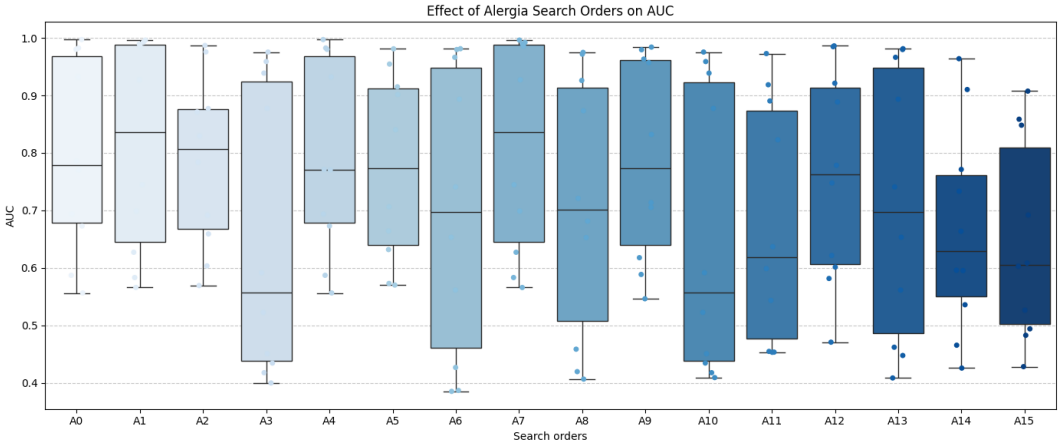


Figure 15: ROC-AUC values of the Alergia algorithm ran on untimed data with various different parameter configurations as described in experiment 2. The parameters used can be found in table 3.

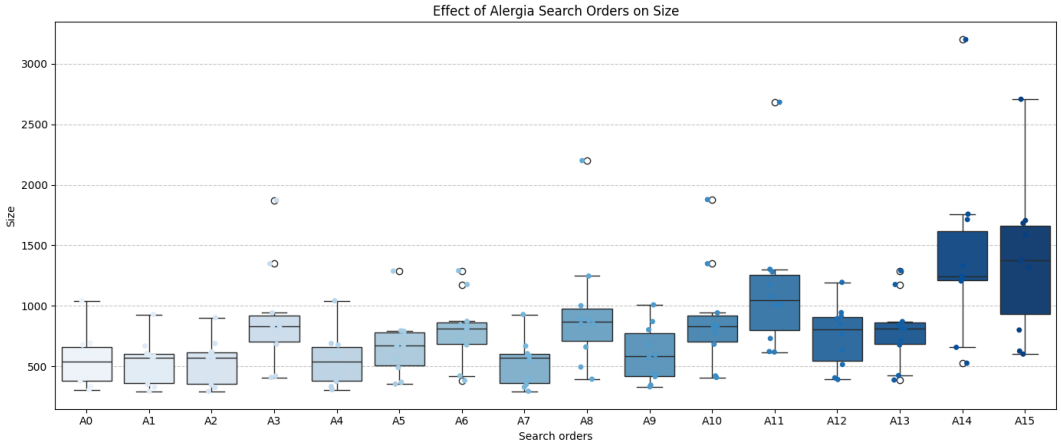


Figure 16: Size values of the Alergia algorithm ran on untimed data with various different parameter configurations as described in experiment 2. The parameters used can be found in table 3.

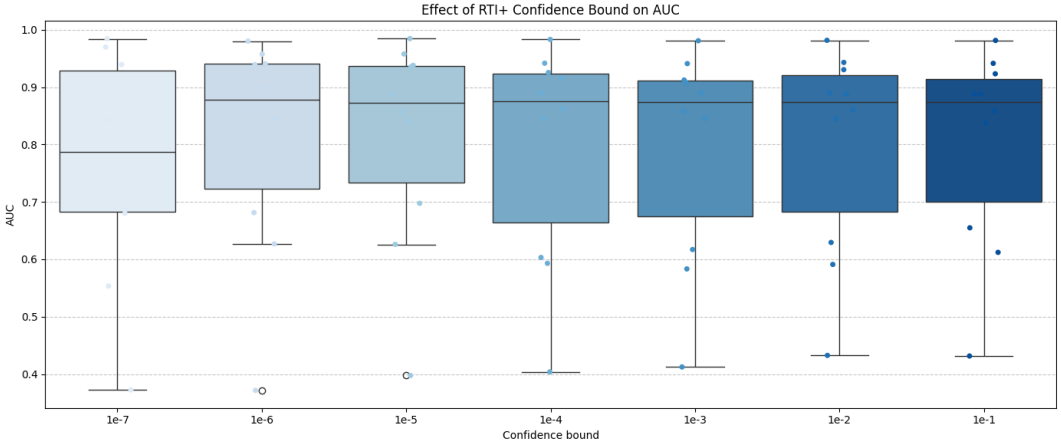


Figure 17: ROC-AUC values of the RTI+ algorithm ran on untimed data with various confidence bounds as described in experiment 3.

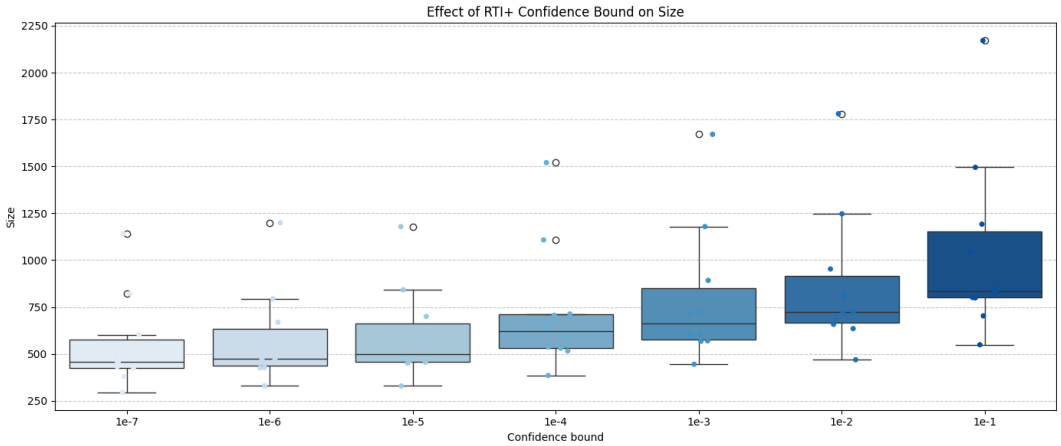


Figure 18: Size values of the pdfa's generated by the RTI+ algorithm ran on untimed data with various confidence bounds as described in experiment 3.

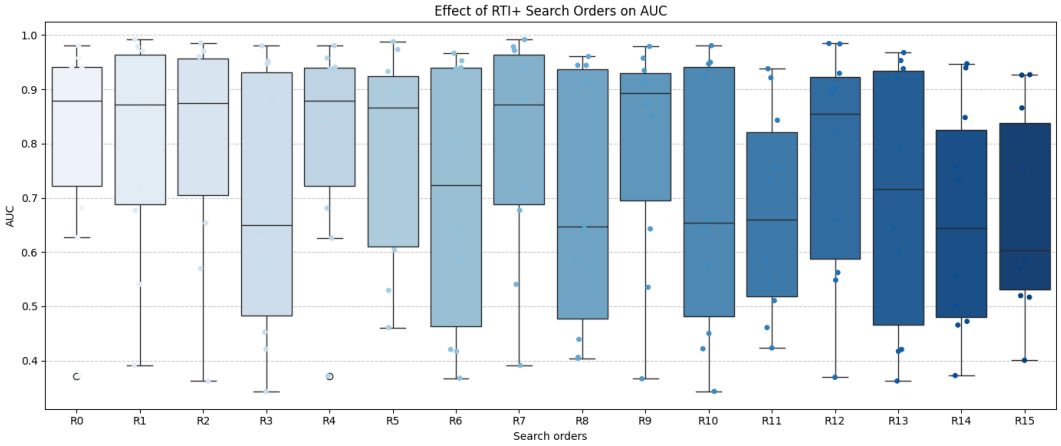


Figure 19: ROC-AUC values of the Alergia algorithm ran on untimed data with various different parameter configurations as described in experiment 4. The parameters used can be found in table 4.

Figure 20: Size values of the Alergia algorithm ran on untimed data with various different parameter configurations as described in experiment 4. The parameters used can be found in table 4.

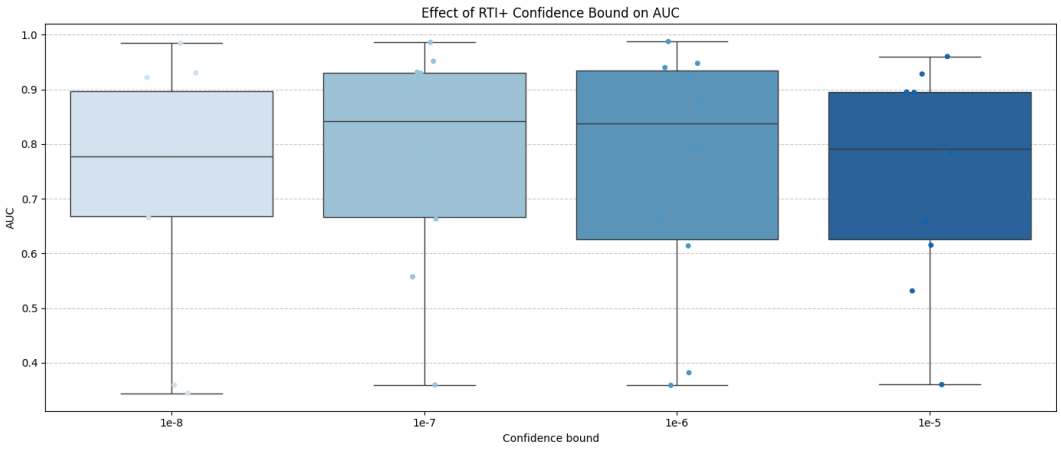


Figure 21: ROC-AUC values of the RTI+ algorithm ran on timed data with various confidence bounds as described in experiment 5.

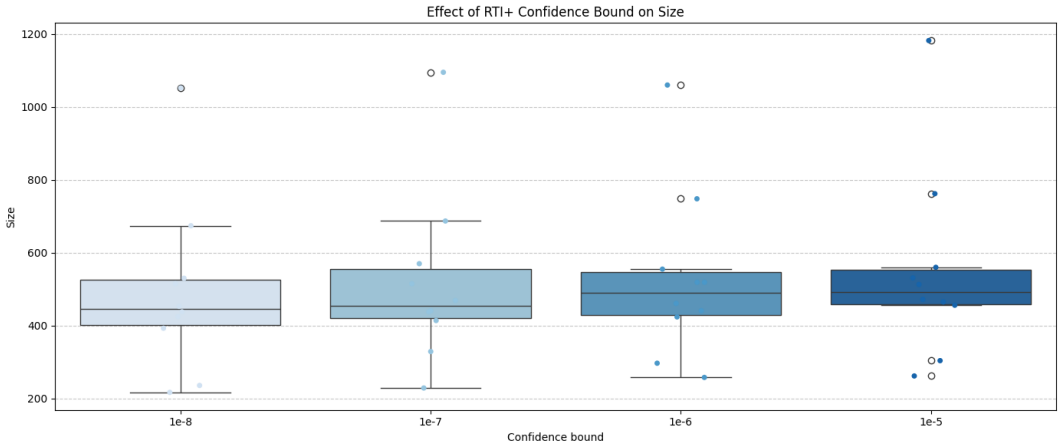


Figure 22: Size values of the pdfa's generated by the RTI+ algorithm ran on timed data with various confidence bounds as described in experiment 5.

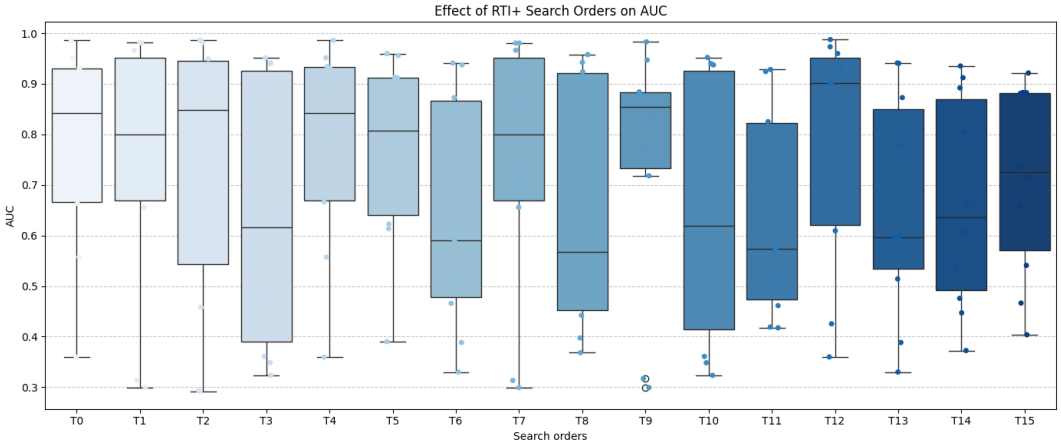


Figure 23: ROC-AUC values of the Alergia algorithm ran on untimed data with various different parameter configurations as described in experiment 6. The parameters used can be found in table 5.

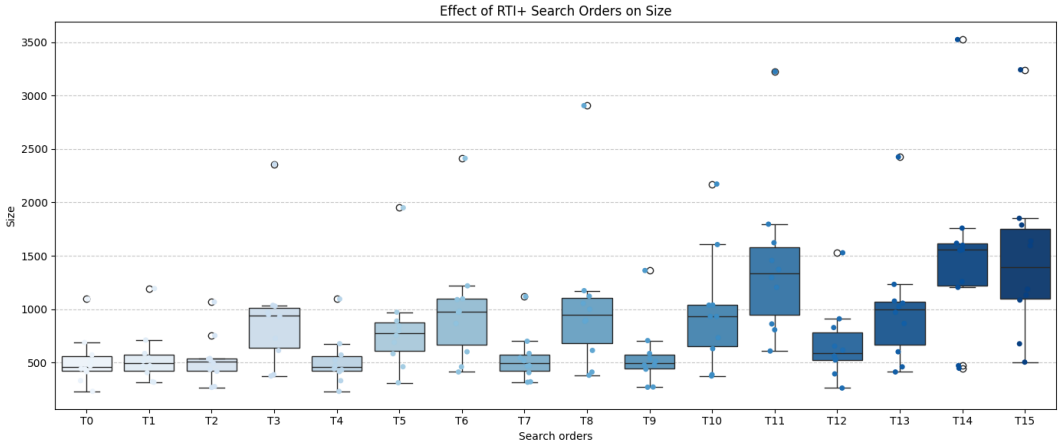


Figure 24: Size values of the Alergia algorithm ran on untimed data with various different parameter configurations as described in experiment 6. The parameters used can be found in table 5.

D Additional figures

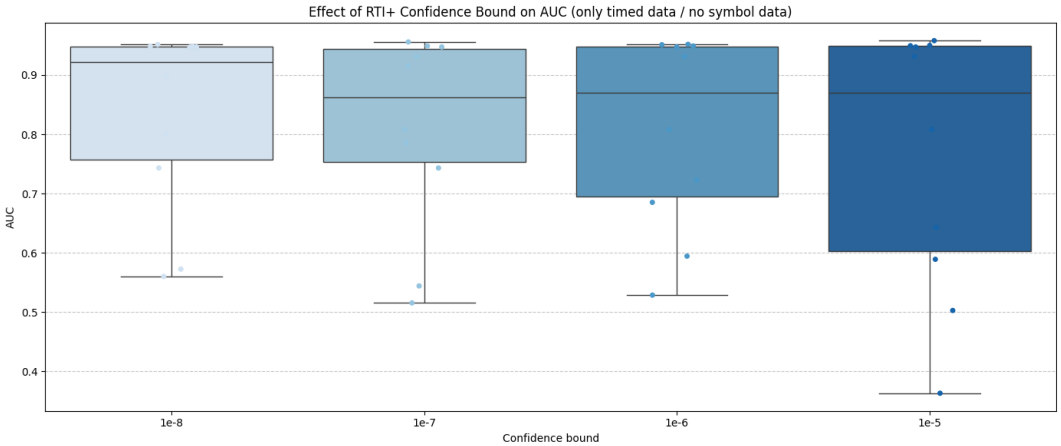


Figure 25: ROC-AUC values of the RTI+ algorithm ran on only timed data. with various confidence bounds. Only timed data means that it didn't look at the symbols of the traces but only at the timings.

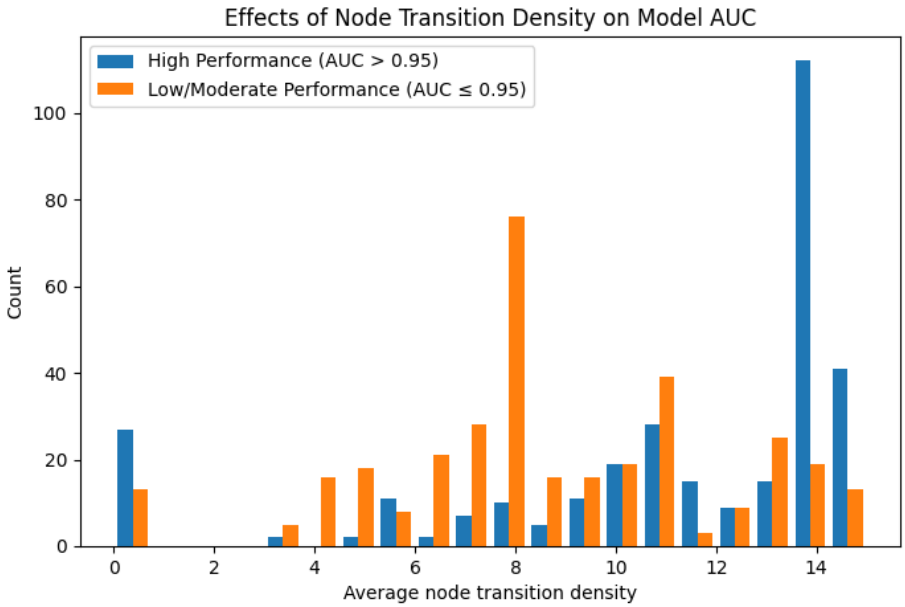


Figure 26: Distribution of the average amount of edges per node per pdfa. Divided by good performing AUC > 0.95 and less good AUC ≤ 0.95. Values are taken from all pdfa's across experiments 1-6.

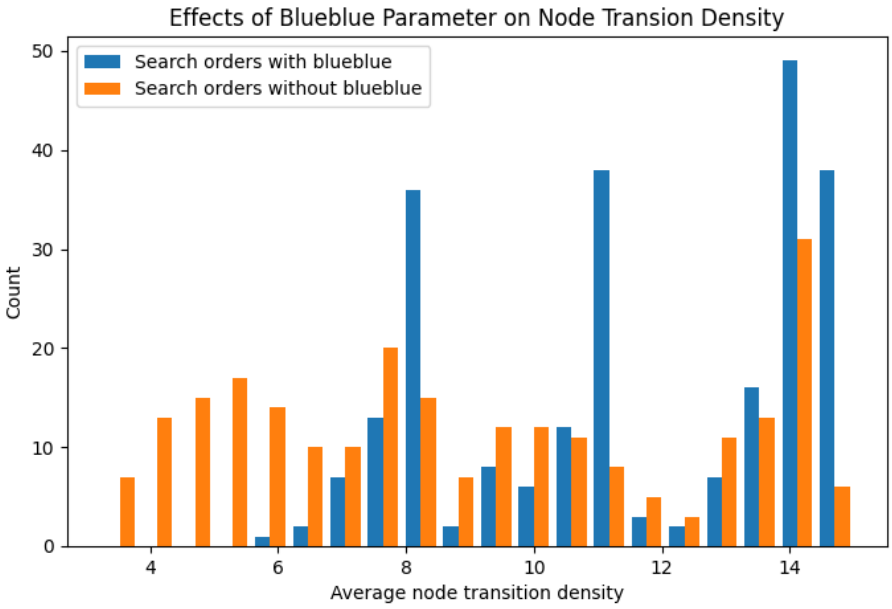


Figure 27: Distribution of the average amount of edges per node per pdfa for search orders with and without blueblue. The plot was generated using only results from experiment 2, 4 and 6.

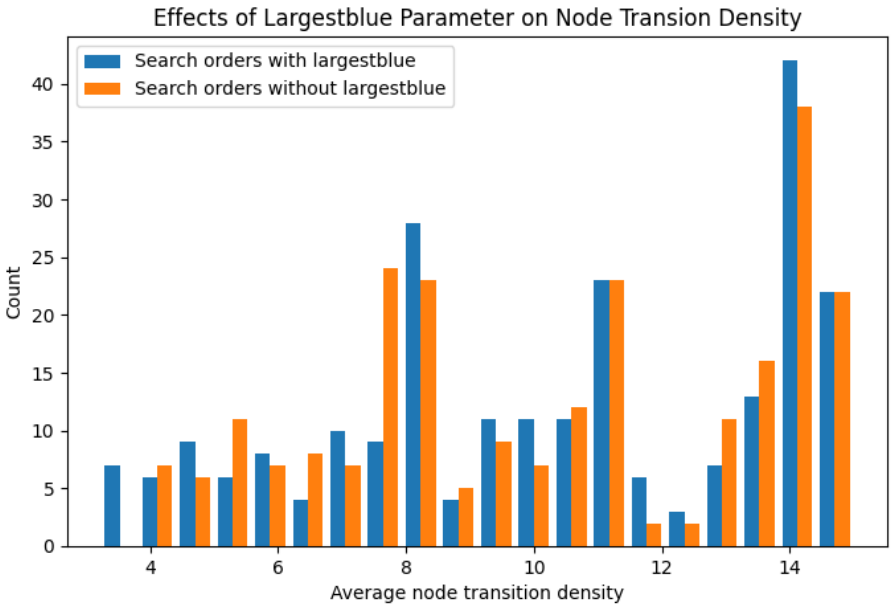


Figure 28: Distribution of the average amount of edges per node per pdfa for search orders with and without largestblue. The plot was generated using only results from experiment 2, 4 and 6.

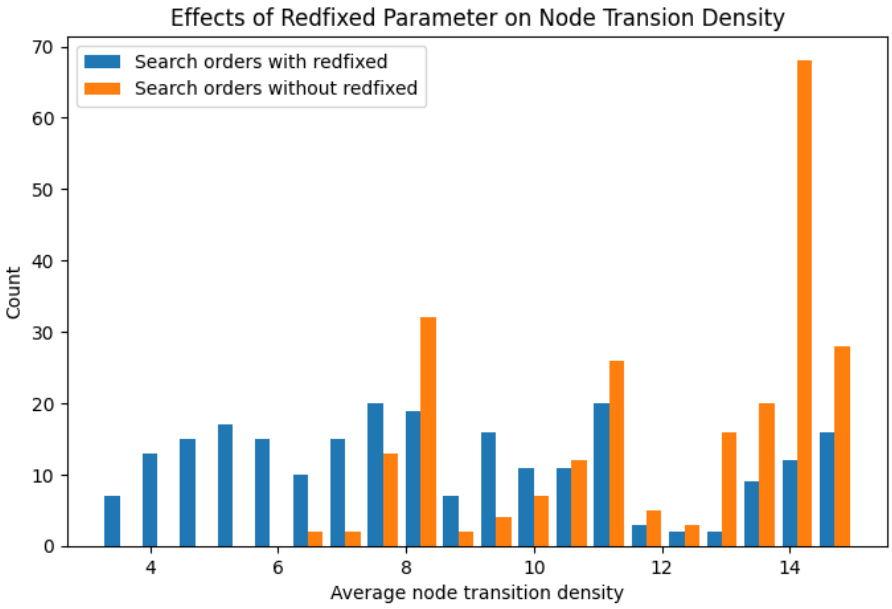


Figure 29: Distribution of the average amount of edges per node per pdfa for search orders with and without redfixed. The plot was generated using only results from experiment 2, 4 and 6.

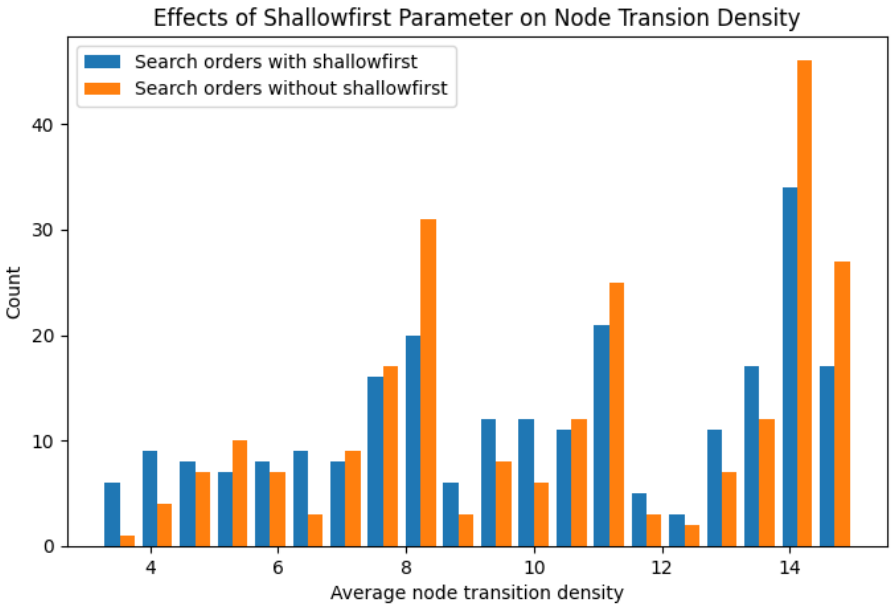


Figure 30: Distribution of the average amount of edges per node per pdfa for search orders with and without shallowfirst. The plot was generated using only results from experiment 2, 4 and 6.