Acta Universitatis Sapientiae

Informatica

Volume 13, Number 1, 2021

Sapientia Hungarian University of Transylvania
Scientia Publishing House

Contents

Acta Univ. Sapientiae, Informatica **13,** 1 (2021) 1–20

DOI: 10.2478/ausi-2021-0001

Differential privacy based classification model for mining medical data stream using adaptive random forest

Hayder K. FATLAWI
ELTE University, Budapest, Hungary
University of Kufa, Najaf, Iraq
email: `hayder@inf.elte.hu`

Attila KISS
J. Selye University, Komarno, Slovakia
email: `kissae@ujs.sk`

Abstract. Most typical data mining techniques are developed based on training the batch data which makes the task of mining the data stream represent a significant challenge. On the other hand, providing a mechanism to perform data mining operations without revealing the patient's identity has increasing importance in the data mining field. In this work, a classification model with differential privacy is proposed for mining the medical data stream using Adaptive Random Forest (ARF). The experimental results of applying the proposed model on four medical datasets show that ARF mostly has a more stable performance over the other six techniques.

1 Introduction

A series of researches and projects in medical science, and information technology (IT) are starting a relationship between the healthcare industry and the IT industry that rapidly leads to a better and interactive relation among patients, their doctors, and health institutions. Data mining has a significant

Computing Classification System 1998: H.2.8, I.2.1
Mathematics Subject Classification 2010: 68P25, 97R40
Key words and phrases: ensemble methods, bagging, privacy-preserving protocol

role in medical data processing and analysis that mostly aims to predict the possibility of diseases or diagnose them [11]. One of the most remarkable challenges facing data mining is privacy preservation.

Privacy is an important component of medical data processing, as many health institutions refrain from providing this data to the public, due to the fear of compromising patient privacy. Therefore, providing a mechanism to carry out data mining operations, without revealing the patient's identity has recently taken place in the interest of researchers.

1.1 Problem statement

Privacy can be provided using many techniques that aim mostly to make a data modification to hide the identity of the objects in data and enable performing the mining operations on the data stream. This modification may destroy the distribution of the data, hence, the effectiveness of data will weaken for data mining techniques. Therefore, the combination of privacy and utility of the data for the mining process represents an interesting challenge. On the other hand, stream data mining techniques are characterized by fast response and ability to adapt to change in data distribution, while privacy-preserving techniques can cause delays in response time or/and difficulty in detecting the drift, which can lead to failure to adapt the mining model properly.

Also, the differential privacy-preserving technique performs data modification in which the average of added noise values for an attribute equal to zero, and that keeps the overall distribution of the data of this attribute. On the other hand, with a data stream, this can not be applicable because only some data instances in a specific time moment are available, which represents another challenge.

Therefore, the mining stream privacy-preserving model should satisfy the following conditions:

1. Data modification should be performed in which the presence or absence of any data element doesn't affect the statistics of the query. This condition aims to make any attacker can't ensure if any identity contributes to the data or not.
2. The modification should preserve the distribution changes in the stream samples to avoid decreasing classification accuracy.
3. Modification time should be fast as much as possible to avoid the response delay of the stream mining technique.

This work aims to design and implements a data stream classification model that satisfies these conditions. It should be capable of building a classifier based on modified data to maintain privacy with minimal impact on response time and classification accuracy.

1.2 Related works

Chaudhuri et al. [5] addressed the tradeoff between privacy and learnability by focusing on privacy-preserving logistic regression. Their work involved disturbing the classifier with noise proportional to the sensitivity. They claimed that their technique didn't depend on the sensitivity of the function, and can be extended to a class of convex loss functions. Kadampur et al. [12] applied noise addition after building the decision tree from data in which for each path from the root to a leaf the noise values were added to the attributes of that path. Although the capability of handling categorical and numerical attributes, the classification accuracy was degraded after applying their model with three datasets.

Dwork et al. [7] constructed a privacy-preserving synopses using boosting for a set of queries over an input database, their algorithm obtains a synopsis that is good for all of these sets in which the privacy is guaranteed for the rows of the database while boosting is performed on the queries. They also provided synopsis generators for arbitrary sets of arbitrary low sensitivity queries. Vaidya et al. [20] utilized a random decision tree and random encryption to develop a distributed data mining framework with privacy-preserving. Their model had slower performance compared to a non privacy-preserving version though the accuracy exactly the same.

Two approaches from a combination of quasi-identifier and sensitive attribute (equal and unequal) were proposed by Bhaladhare et al. [3]. To minimize information loss that happened as a result of applying privacy-preserving, their model utilized systematic clustering for clusters generation. Although the loss of information and the execution time was better compared with Greedy k-member and Systematic clustering algorithms, their model had a moderate level of data utility. Homomorphic encryption scheme with cloud-aided association rule mining proposed by Li et al. [14] to achieve privacy-preserving with frequent itemset mining. According to their experiment results using many data sets, the model had fewer information leaks but higher computational time. Wang et al. [21] proposed a randomized response based approach for privacy-preserving in data collection. The implementation of their approach using data of patients showed less utility loss than the standard Laplace approach.

A distributed framework for preserving privacy using clustering in Hadoop was proposed by Nayahi et al. [16]. It used Hadoop Distributed File System and tried to overcome some attacks such as similarity attacks. The computational time of their model increased as the number of clusters increased, and they claimed that their algorithms were highly scalable with the size of the data set. Zhang et al. [22] used two mechanisms of noise: Laplace and exponential for providing privacy. They utilized lower noise sensitivity to avoid a high impact on split point choosing. They applied the proposed model on only one dataset, and the results showed more stability in classification accuracy compared with three other algorithms.

Beck et al. [19] proposed a data analytics system for privacy-preserving of a data stream, it provided zero-knowledge privacy guarantee for users, a data analysts interface to explore the output accuracy with the query execution budget, and a close real-time stream processing based on a scalable distributed architecture. Manikandanet al. [15] utilized a code-based threshold scheme with fuzzy c-means clustering for creating distributed privacy-preserving.

Table 1 summarizes the characteristics of those related works which have been mentioned in this section. Most of the related works mentioned in Table 1 were involving classification tasks and only one of them was working with stream data. This points to the lack of research works in privacy preservation for stream data mining. Also, those classification works were mostly lacking in utilizing ensemble classifiers which have a preferable performance with real-world datasets.

This paper is an extension to our paper [8], the extension utilizes the robustness of Adaptive Random Forest (ARF) ensemble classifier against small changes in the distribution of stream data, and build a classification model with the ability of privacy-preserving using Laplace distribution instead of normal distribution. The extension includes mentioning and analysis for additional related works, also the evaluation of the proposed model including one addition algorithm, two new datasets, a different range of noise values that added to the data, and a new comparison for distribution changes and adaptive window sizes. The implementation of the proposed model in this extension produces 364 experiments and confirms the better performance of ARF.

Article	Data Mining Tech.	Dataset	Privacy Preserving Technique	Advantages	Disadvantages
Chaudhuri et al. [5]	Logistic Regression	Artificial Dataset	Differential Privacy	Sensitivity Independent	Only Simulation Results
Kadampur et al. [12]	Decision Tree	Boston Housing Price,Census Income,Car Evaluation	Noise Addition	Handle Categorical and Numerical Data Types	Less Accuracy than Original Classifier
Dwork et al. [7]	Boosting	-	Differential Privacy	Stronger Bounds on Expected Privacy Loss	No Experiment on Real Datasets
Vaidya et al. [20]	Random Decision Tree	Mushroom, Nursery, Image Segmentation, and Car	Random Encryption	Fast Distributed Mining with Same Accuracy	Slower than Non Privacy-preserving Version
Bhaladhare et al. [3]	Systematic Clustering	Benchmark Adult	Combination of Quasi-identifier and Sensitive Attribute	Lesser Information Loss	Moderate Level of Data Utility
Li et al. [14]	Frequent Itemset	Retail and Pumsb Datasets	Vertically Partitioned Databases	Leak Less Information	Slower than Algorithms with Low Privacy Levels.
Wang et al. [21]	Data Collection	YesiWell	Randomized Response	Fewer Utility Loss with High Sensitivity of Functions	Depending on Only One Dataset
Nayahi et al. [16]	J48 , Naive Bayes , K-NN	Benchmark Adult d	K-anonymization	Scalability on Increasing Dataset Size	Time Increasing when Number of Clusters Increasing
Zhang et al. [22]	Decision Tree	Census Income	Laplace and Exponential Noise	More Stable Accuracy	Depending on Only One Dataset
Beck et al. [19]	Sampling	NYC Taxi Ride,Household Electricity Consumption	Randomized Response	Distributed Real-time Stream Processing	Accuracy Loss doesn't Always Decrease when Second Randomization Parameter Increases
Manikandan et al. [15]	Fuzzy C-Means	Plant Cell Signaling	Code Based Technique with Threshold Estimation	Less Number of Iterations and No Cross Trust is Required	Focus Only on Efficiency

Table 1: Comparison of some research works on privacy-preserving data mining

2 Basic concepts in stream data mining

2.1 Data stream constraints

Unlike with batch data, stream data faces many constraints as follow: (1) infinite arrival of data samples make storing them impossible, (2) the fast arrival of data samples requires dealing with each sample in real-time, (3) the possibility of changing items' distribution overtime in which the old data would be useless for the current status. Generally, the perfect classification model should produce maximum accuracy in the fastest time and minimum computational resources [2].

2.2 Concept drift

Concept drift refers to that the data is being gathered may change from time to time, every time according to some minimum persistence. Changes may occur during the time in which the old training examples become irrelevant to the current state, and the learning system should forget such kind of information. There are two important issues related to the change: causes of change and the rate of change [9].

2.3 Adaptive sliding window (ADWIN)

It is an estimation technique that aims at detecting the change in a data stream based on a sliding window with adaptive size. It has a qualified and significant method for tracking the average of bits in the stream. In this technique, the length of windows is not updated as long as the average value inside the window doesn't change [9].

2.4 Hoeffding tree

Hoeffding Tree or Very Fast Decision Tree (VFDT) is a variation from the typical decision tree designed for stream data. The learning of these techniques depends on replacing leaves of the tree with decision nodes. Each terminal node (leaf) in the tree stores enough information statistics about features values that are used by a heuristic function to perform a splitting test. After reaching a new data instance, it transfers starting from the root until reaching a specific leaf node. At this point, the statistics information then is evaluated and a new decision node may be created based on this evaluation [9]. It is very popular to utilize VFDT as a base learner for the ensemble classification model, thereby, the ensemble techniques in this work used VFDT as well.

VFDT depends on the concept of Hoeffding Bound [9] which states that the probability of the difference between the expected value and the actual value of the mean of data elements to be more than ϵ value shouldn't exceed a specific small value as follows: let $F_1, F_2, \ldots, F_n$ be an independent random variable and each F_i is bounded in which

$$P\left(F_i \in R = [x_i, y_i]\right) = 1. \tag{1}$$

Let

$$H = \frac{1}{n}\sum_{i=1}^{n} F_i$$

with expected value $E(H)$. Then for any $\epsilon > 0$,

$$P[H - E[H] > \epsilon] \leq e^{-\frac{2n^2\epsilon^2}{R^2}}. \tag{2}$$

2.5 Adaptive random forest

Models based on a single classifier have some weakness points, such as model instability which means any slight changes in data may make a change in the structure of the tree classifier. To overcome that, ensemble methods have been developed which combine many weak classifiers [13, 1]. Ensembles have more power predictive performance than a single tree, so they became general techniques for both classification tasks and numeric prediction [17, 13]. The methodology of an ensemble model is to combine a set of single models, each one tries to solve the same original task, aiming to obtain a better integrated global model [10]. Two points should be taken into account when using ensembles: (i) the size of an ensemble (ii) the mechanism of combination among the results of trees [23]. Many techniques are developed for ensemble models such as bagging, boosting, and stacking. Bagging combines the decisions of multiple trees by using the voting concept for binary (and multi) class predictive tasks, and for a numerical predictive task, bagging calculates the average. A popular example of bagging techniques is the random forest [1].

Ensemble modeling aims at building a strong accumulative classifier from many weak classifiers. Adaptive Random Forest is a variation from the typical random forest algorithm for data stream mining tasks. The main idea is to utilize Hoeffding trees, which have the ability to adapt to distribution changes, as the base classifier for the bagging ensemble method [2]. For detecting the change in a data stream, ADWIN is used in these techniques. It depends on Online Bagging as a resampling method and a drift monitor for change detection per each tree [2].

2.6 Differential privacy

Differential privacy aims at learning information about the whole data while preserving the privacy of each data sample. The differential privacy model assumes that although the availability of the knowledge about all data records except one, the adversary is not be able to extract the information of that record. It can be resistant to background attack in comparison with other privacy models, also, privacy guarantee of differential privacy is provable [24].

In general, the system with differential privacy should have the same performance without any consideration for to presence or absence of any data sample, and this can be performed by keeping the probability distribution of data [7]. According to [5], differential privacy can be provided using a randomized mechanism RM if for all databases $\mathsf{DB1}$ and $\mathsf{DB2}$ that differ by one element for any t,

$$\frac{\mathsf{P[RM(DB1) = t]}}{\mathsf{P[RM(DB2) = t]}} \leq e^{\epsilon} \tag{3}$$

The privacy guarantee level of the differential privacy model is controlled by the parameter ϵ which represents the privacy budget. Sensitivity is another aspect related to differential privacy, it indicates the required amount of perturbation for this mechanism by calibrating the volume of noise. There are two types of sensitivity used in differential privacy: (i) global sensitivity represents the largest value for the difference between results of the query on different related datasets, (ii) local sensitivity concern with calibrating the difference between query results based on records. Queries with relatively low values are preferable with global sensitivity [24].

3 Methodology

The main aim of this work is to design and implement a classification model for stream data based on adaptive random forest, including differential privacy. Thereby, there are two main stages; the first one is to apply some of the preprocessing procedures to prepare the medical data for the mining task. The second stage is to build an ensemble classifier which includes many very fast decision trees, and finally compare the performance based on streaming real batch datasets. Figure 1 illustrates all the steps of our work.

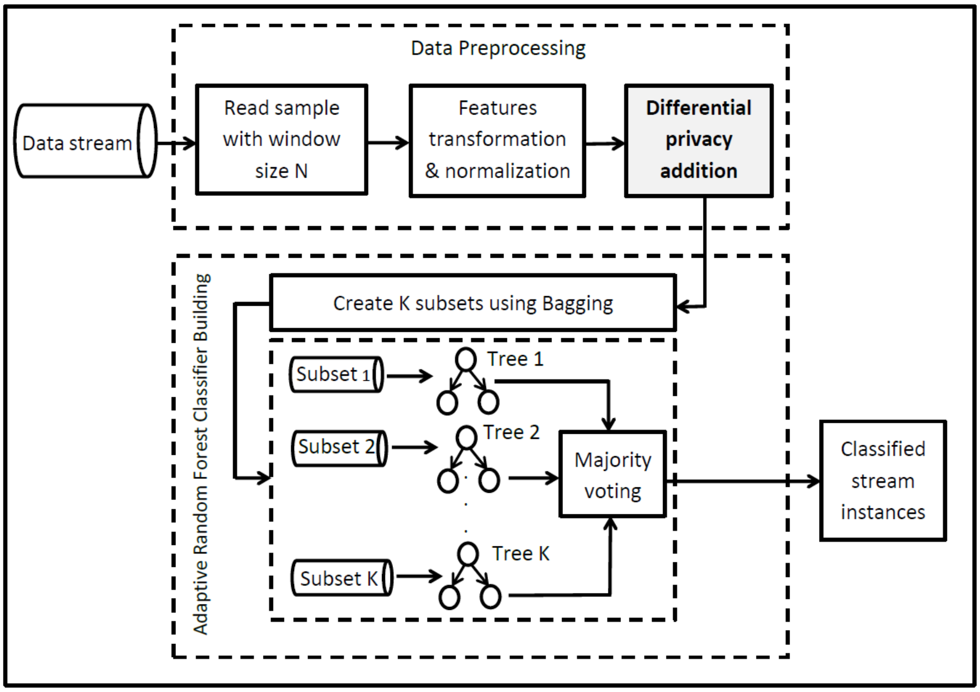

Data Preprocessing
Data stream
Read sample with window size N
Features transformation & normalization
Differential privacy addition
Adaptive Random Forest Classifier Building
Create K subsets using Bagging
Subset 1
Subset 2
Subset K
Tree 1
Tree 2
Tree K
Majority voting
Classified stream instances

Figure 1: The classification model with ϵ-differential privacy for medical data stream

3.1 Stage one: data preprocessing

The first stage concerns with preparing and generating a new dataset from the original one by using features transformation, normalization, and the white noise concept. This stage can be described as following steps:

3.1.1 Categorical to numerical transformation

To add the noise values to the data, features should be in a numeric form. So, in this step, every Categorical (Textual values) was converted to numerical values. For binary features values like (Yes, No), the simplest coding is used and the values became (1,0). For multiple values (more than two values), frequency of each distinct value (1..N) for each feature was calculated. After that coding was used in which the most frequent distinct textual value converted to N and least frequent converted to 1.

3.1.2 Data normalization

The range of feature values can be different, such as age has values between 1 - 150 while the yearly income can be between (1-10000000). For that, we need to apply normalization to prevent any dominant from one of the features during statistical calculation that performed during classifier building. The new range for all feature values were between -1 and 1 in which for each feature f:

$$f(i) = 2\frac{f(i) - minf}{maxf - minf} - 1 \tag{4}$$

3.1.3 ϵ-Differential noise generation

For each data set, noise value was added in which the mean of those values for each feature is zero. The standard deviation (STD) represents the intensity of the noise and the gradual increase of it will be used in the proposed model to investigate the suitable value for the input data. The random values should satisfy the condition of the random mechanism Eq. (3). To obtain these noise values, Laplace Mechanism as presented in [24] was utilized in which the values were generated from the Laplace distribution, which has zero center and scale q. Large q value produces a higher noise value z as the following:

$$Lab\,(z) = \frac{1}{2q} exp\left(\frac{-|z|}{q}\right). \tag{5}$$

3.2 Stage two: building ensemble model

In this stage, we utilized online Bagging of K base classifier [18], each base classifier built using Hoeffding Tree as presented by [6] . The stage started with setting the size of ARF ensemble model, then number of data subsets produced from resampling the data sample that resulted from the previous stage. The number of subsets was equal to the number of base classifiers, each subset was used to train a Hoeffding Tree classifier, and finally, voting among all base classifiers was used to classify each data element.

3.2.1 Initializing the size of ensemble model

While Online Bagging was used for ensemble model building, the size of this model i.e. the number of base classifiers was a user-defined parameter that needs its value before the building operation started. The importance of this parameter comes from it represents a stopping condition for each learning step, which is related to the complexity of required computational resources. The value of this parameter in the proposed model prefers to be in low range value (around 10 base classifier) for the following reasons:

1. Stream mining classifier is expected to have a fast response in learning and classifying process as a stream element reach continuously.
2. The number of data rows in each data sample inside the current window is relatively low, thereby, the resampling step in Online Bagging doesn't need for large ensemble model for preserving the diversity of each base classifier.

3.2.2 Ensemble model building

In online Bagging, any new data sample was chosen according to a Poisson(1) distribution. The classification decision of the ensemble bagging model is based on the voting of all K base classifiers with equal weight for all of them. It gives every new data example an initial weight w=1, then it is passed to the first weak learner. If this data example is misclassified, it's weight is increased before passing it to the next weak learner. The base learner in our comparison was Hoeffding Tree classifier and the size of the ensemble that used was ten learners. Adaptive Random forest was built depending on [4] which utilized Online Bagging's resampling method but the difference was in the adaptive method.

3.2.3 Hoeffding tree classifier building

It includes two types of nodes: internal (or decision) and terminal (or leaf) nodes. Each terminal node in the tree stores enough statistical information about features values. This information is used by a heuristic function to perform a splitting test. After reaching a new data instance, it transfers starting from the first node (root) until reaching a specific leaf node. In this point, if the class value of new instance isn't seen before, the instance then is classified according to the majority class of the current leaf node, otherwise, the statistics information is evaluated and a new decision node may be created based on this evaluation.

The evaluation includes computing the gain for all features in all possible split points. For each split point, the impurity of class distribution of the current node and the possible child nodes will be computed using Entropy, according to the following equation, for node a:

$$\mathrm{Entropy}(a) = -\sum_{i=0}^{c-1} p\,(i|a)\,\log_2 p\,(i|a)\,, \tag{6}$$

where c refers to the number of classes. The difference between the Entropy of the current node and the average of Entropy of its possible child nodes after splitting represents the gain of that splitting operation. A splitting that produces a more homogeneous class distribution i.e. higher gain is preferable. Hoeffding bound computes using Eq. (2), and if the difference between the highest two features is more than Hoeffding bound value, the current leaf node will replace by an internal decision node depending on the highest feature, also, for each split branch of this new node, a new empty leaf node will be added. Algorithm 1 summarizes all the steps of the proposed model.

4 Implementation and experimental results

4.1 Data analysis platform

In this work, three major tools were utilized to perform the comparison; Waikato Environment for Knowledge Analysis (Weka), Massive Online Analysis (MOA), and Sklearn. Weka Platform is open-source software for data analysis tasks including Classification, Clustering, and Association Rules. It is developed by the University of Waikato using Java programming language. It was utilized in this work for preprocessing operations (Transformation and Normalization).

MOA Platform is an improvement for the Weka platform for the mining data stream. It provides many popular mining techniques, stream generator, and concept drift detection techniques, in our comparison, it performed the data streaming and implementation of classification techniques. Sklearn is a python free library for machine learning tasks. It contains many classification techniques such as random forest and boosting. Sklearn was used in this work for adding white noise values to data.

Algorithm 1 Stream data classification model with ϵ-differential privacy

```
 1: procedure ARF
 2:     St = current stream samples in ADWIN window
 3:     From St Create K data subsets using Bagging resampling
 4:     for each subset S do
 5:         for each feature f in S do
 6:             if IsCatogorical(f) = True then
 7:                 for each distinct value d in f do
 8:                     f(d)= frequency(d)
 9:                 end for
10:             end if
11:             Apply normalization on S according to Eq. (4)
12:         end for
13:         for each data instance dt in S do
14:             Generate random value R for differential privacy    ▷ Based on Eq. (3),(5)
15:             dt = dt+R
16:             Trace Hoeffiding tree reaching to a specific terminal node tn
17:             if Class value of dt != ? then
18:                 dt(y)= major class of tn
19:             else
20:                 for each feature f in tn do
21:                     G1= Class impurity in tn                ▷ Based on Eq. (6)
22:                     G2= Class impurity in tn 's possible child nodes    ▷ Based on Eq. (6)
23:                     G= G1-G2
24:                 end for
25:                 Rank every features based on its gain
26:                 Choose two features bf1,bf2 with highest gain
27:                 Compute HB= Hoeffiding Bound based on Eq. (2)
28:                 if G(bf1)-G(bf2) > HB then
29:                     Replace tn with a decision node based on split test of bf1
30:                     Add new terminal nodes for each possible split value
31:                 end if
32:             end if
33:         end for
34:     end for
35:     Return ARF classifier, Classified sample instances
36: end procedure
```

4.2 Applying adaptive random forest with differential privacy

This step includes applying ensemble classifier against gradually increasing in differential privacy strength of data by using white noise. For this task, the Weka platform is used to perform two preprocessing steps; features transformation and normalization. Then MOA is used to convert batch datasets to a data stream, then to train the classifier based on that stream. Four measurements are used for evaluating the performance of the classification techniques; mean of correctly classified instances, mean of F1 score, mean of precision, and mean of Recall. Figure 2 and Table 2 clarify the performance of the proposed model using different Standard Deviation STD randomize values for Differential Privacy.

Diff. Noise STD	0.000	0.001	0.002	0.003	0.004	0.005	0.01	0.015	0.02	0.025	0.05	0.075	0.1
EEG State	98.75	98.28	98.17	98.29	98.11	98.08	98.03	98	97.90	98.06	98.06	98.09	97.84
Skin Seg.	100.00	99.99	99.99	99.99	99.99	99.99	99.99	99.99	99.99	100.00	99.99	99.99	99.99
MIT-BIH	90.93	82.72	82.71	82.71	82.70	82.71	82.71	82.70	82.70	82.71	82.71	82.72	82.70
Breast Cancer	99.35	99.35	99.35	99.35	99.35	99.35	99.35	99.35	99.35	99.35	99.35	99.35	99.35

Table 2: Accuracy of ARF with different STD values

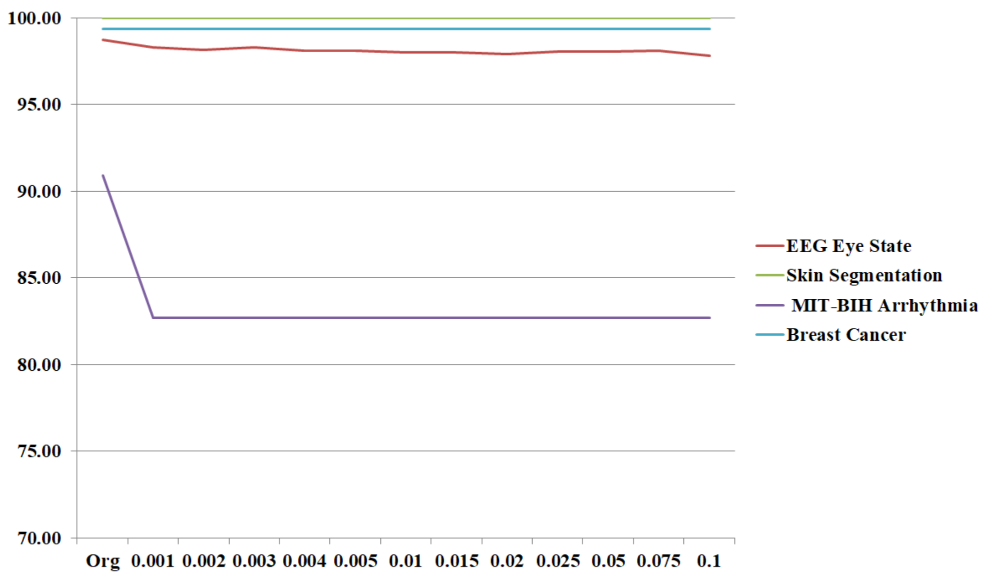


Figure 2: Accuracy of ARF with range of STD values for differential privacy

In both Table 2 and Figure 2 , we can observe the following:

1. Ability of ARF to preserve the accuracy of classification after adding the noise values with the first two and the fourth datasets.
2. Stability of ARF with different values of additional noise with the first two and the fourth datasets.
3. There was a decrease in the classification accuracy with the third dataset after adding the minimum magnitude of the noise, however, ARF recovered its stability with the rest of the range's values.

The main difference between the first two and the fourth dataset aside, and the third dataset from another side is that the number of features in the third dataset is more, this leads to a question that if the high dimensionality can affect the utility of ARF after adding the differential privacy.

The preference of the proposed model using ARF compared with many other techniques can be observed in Table 3 and Figure 3, however, there was a close performance between ARF and Ozabagging. The similarity of those two techniques that are both of them is a bagging ensemble classifier, a question arises if the strength of ARF with privacy-preserving in the medical data stream can be generalized to other bagging techniques. Also, we can observe that Naive Bayesian had unstable performance, in which it had the worst performance in most cases. All results in Table 3 were using the minimum STD value for differential privacy.

Technique	Heoffman	ARF	OzaBagg	OzaBoost	K-NN	N. Baysain	Random Hoeffman
EEG State	72.56	98.28	93.33	85.56	88.4	48.42	65.48
Skin Seg.	99.94	99.99	99.97	79.06	99.97	95.29	99.93
MIT-BIH	82.7	82.72	82.72	87.87	82.67	14.61	82.70
Breast Cancer	99.31	99.35	99.35	99.19	99.35	94.33	99.3

Table 3: Accuracy comparison of classification algorithms based on streaming four medical datasets

Other interesting findings from the experimental results are illustrated in Figure 4 and Figure 5. We can observe that the number of drifts i.e change in the distribution of data streams for all features in Skin Seg. and EEG Eye datasets was reduced significantly after adding differential privacy. As a result of this reduction, the size of ADWIN window which illustrated in Figure 6 and Figure 7 was maximized to contain all stream elements in EEG state and Skin Seg. datasets. This smoothness of the data stream leads to reduce the number of changes in ARF model to adapt to change in distribution, thereby, the computational time of ARF has been reduced, and that could overcome the addition time of differential privacy step.

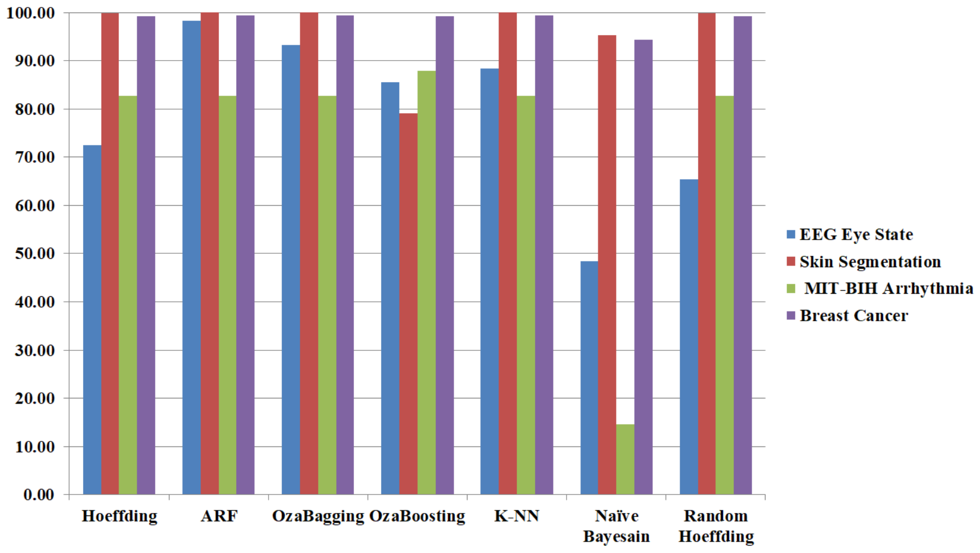


Figure 3: Comparison of the proposed model accuracy with other six techniques

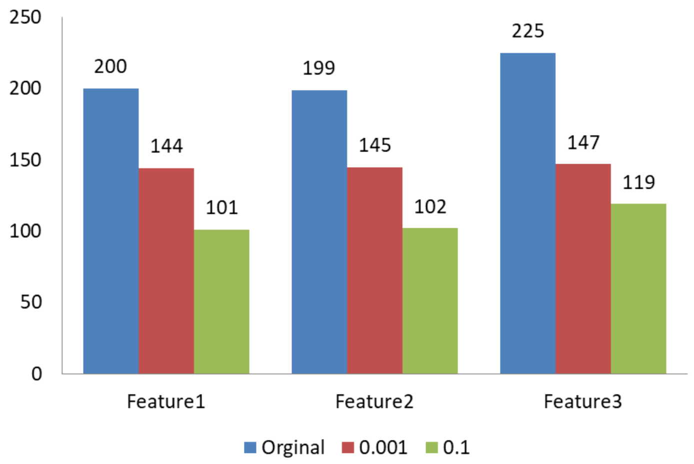


Figure 4: Comparison of drifts in skin sig. dataset with two STD values

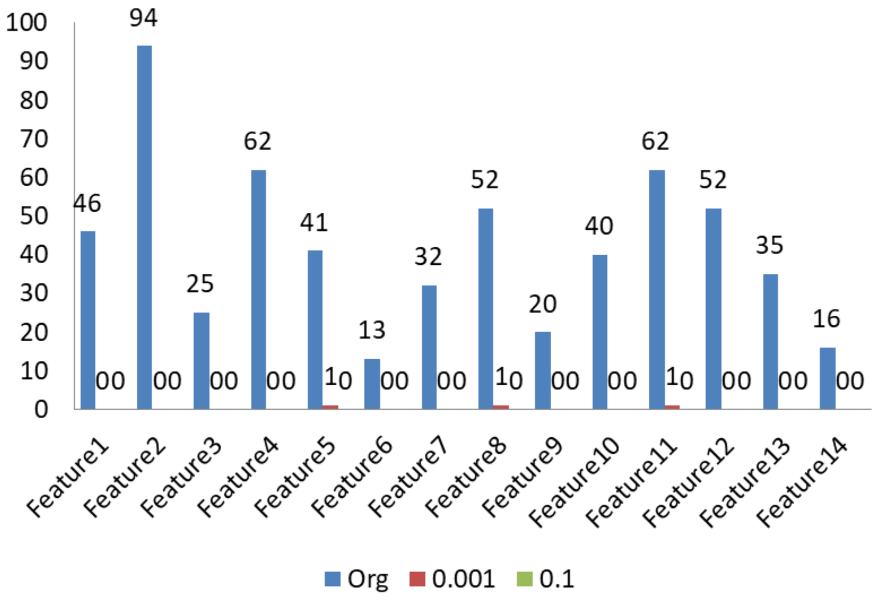


Figure 5: Comparison of drifts in EEG state dataset with two STD values

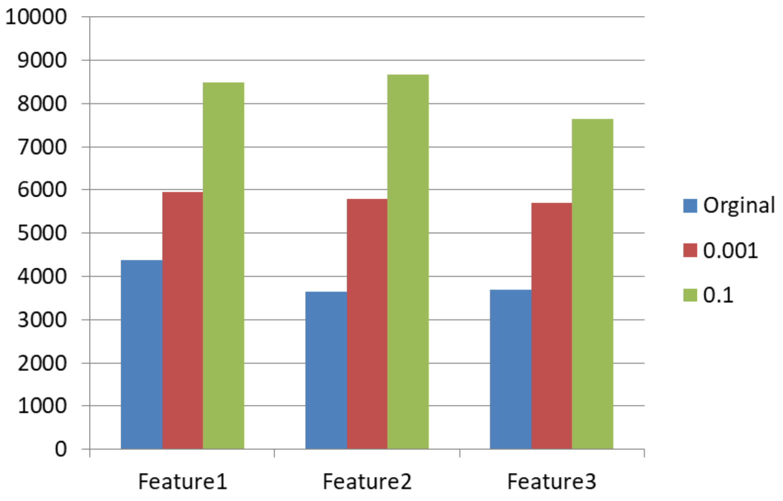


Figure 6: Comparison of ADWIN size in skin sigm. dataset with two STD values

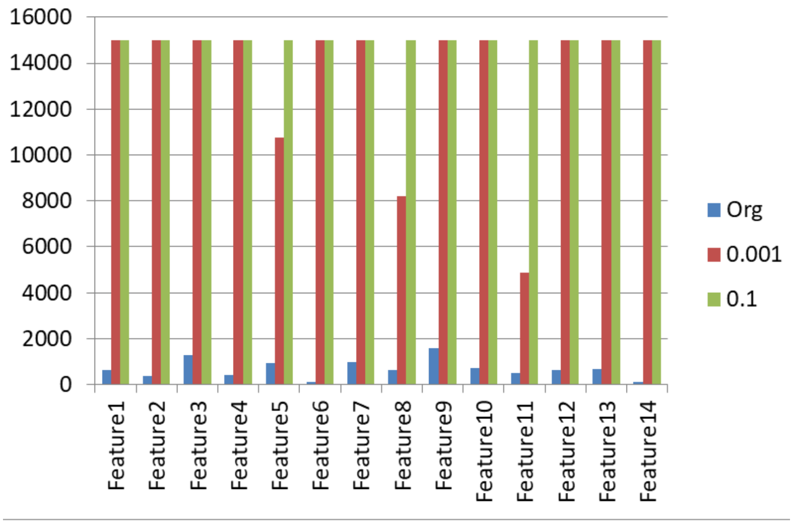


Figure 7: Comparison of ADWIN size in EEG state dataset with two STD values

5 Conclusion

In this work, a privacy-preserving classification model for the mining data stream was proposed. It utilized the robustness of Adaptive Random Forest classifier to handle the randomized values that added to the original data stream using differential privacy. The proposed model had the best accuracy compared with the other six techniques applied on four real medical datasets, OzaBagging also has notable performance, and both techniques are bagging ensemble methods. Also, the number of drifts in the distribution of data streams was reduced significantly after adding differential privacy, as a result, that the size of ADWIN window was maximized. These results obtained by applying 364 experiments using a gradual increase of STD of randomizing values for proving the differential privacy.

Acknowledgments

The project was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).

References

[1] A. Al-Fatlawi, H. Fatlawi, S. H. Ling Recognition physical activities with optimal number of wearable sensors using data mining algorithms and deep belief network, *2017 39th annual international conference of the IEEE engineering in medicine and biology society (EMBC)* Seogwipo, South Korea, 2017, pp. 2871–2874. ⇒7

[2] B. Babenko, MH. Yang, S. Belongie, A family of online boosting algorithms, *2009 IEEE 12th international conference on computer vision workshops, ICCV workshops* Kyoto, Japan, 2009, pp. 1346–1353. ⇒6, 7, 8

[3] P. R. Bhaladhare, D. C. Jinwala, Novel Approaches for Privacy Preserving Data Mining in k-Anonymity Model, Journal of information science and engineering, **32** (2016) 63–78. ⇒3, 5

[4] A. Bifet, G. Holmes, B. Pfahringer, G. Bernhard, Improving adaptive bagging methods for evolving data streams, *Asian conference on machine learning* Nanjing, China, 2009, pp. 23–37. ⇒11

[5] K. Chaudhuri, C. Monteleoni, Privacy-preserving logistic regression, *Advances in neural information processing systems* Vancouver, Canada, 2009, pp. 289–296. ⇒3, 5, 8

[6] P. Domingos, G. Hulten, Mining high-speed data streams, *KDD00: the second annual international conference on knowledge discovery in data* Boston Massachusetts, USA, 2000, pp. 71–80. ⇒10

[7] C. Dwork, G. N. Rothblum, S. Vadhan, Boosting and differential privacy, *2010 IEEE 51st annual symposium on foundations of computer science* Las Vegas, Nevada USA, 2010, pp. 51–60. ⇒3, 5, 8

[8] H. Fatlawi, A. Kiss, On Robustness of Adaptive Random Forest Classifier on Biomedical Data Stream, *Asian Conference on Intelligent Information and Database Systems (ACIIDS 2020)*, *Lecture notes in computer science* Springer, **12033** (2020) 332-344. ⇒4

[9] J. Gama, *Knowledge discovery from data streams*, The CRC Press, 2010. ⇒6, 7

[10] G. Giovanni, J. F. Elder, *Ensemble methods in data mining: improving accuracy through combining predictions*, The Synthesis lectures on data mining and knowledge discovery Morgan & Claypool Publishers, 2010. ⇒7

[11] G. Hulten, L. Spencer, P. Domingos, Mining time-changing data streams, *Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining* San Francisco California, USA, 2001, pp. 97–106. ⇒ 2

[12] M. A. Kadampur, D.V.L.N Somayajulu, A noise addition scheme in decision tree for privacy preserving data mining, Journal of computing **2,** 1 (2010) 137–144. ⇒3, 5

[13] M. Kuhn, K. Johnson, *Applied predictive modeling*, Springer, 2013. ⇒7

[14] L. Li, R. Lu, K. R. Choo, A. Datta,J. Shao, Privacy-preserving-outsourced association rule mining on vertically partitioned databases, IEEE transactions on information forensics and security, **11,** 8 (2016) 1847–1861. ⇒3, 5

[15] V. Manikandan, V. Porkodi, A. S. Mohammed, M. Sivaram, Privacy preserving data Mining using threshold based fuzzy C-Means clustering, ICTACT journal on soft computing, **9,** 1 (2018) 1820–1823. ⇒4, 5

[16] J. J. v. Nayahi, V. Kavitha, Privacy and utility preserving data clustering for data anonymization and distribution on Hadoop, Future Generation Computer Systems, **74** (2017) 393–408. ⇒4, 5

[17] T. Ngo, *Data mining: practical machine learning tools and technique, by ian h. witten, eibe frank, mark a. hell*, The ACM SIGSOFT Software Engineering Notes, **36,** 5 2011. ⇒7

[18] N.C. Oza, Online bagging and boosting, *2005 IEEE international conference on systems, man and cybernetics* Waikoloa, HI, USA, 2005, pp. 2340–2345. ⇒10

[19] D. L. Quoc, M. Beck, P. Bhatotia, R. Chen, Christof Fetzer, Thorsten Strufe, PrivApprox: privacy-preserving stream analytics, *2017 annual technical conference (USENIX ATC '17)* Santa Clara, CA, USA 2017, pp. 659–672. ⇒4, 5

[20] J. Vaidya, B. Shafiq, W. Fan, D. Mehmood, D. Lorenzi, A random decision tree framework for privacy-preserving data mining, IEEE transactions on dependable and secure computing, **11,** 5 (2013) 399–411. ⇒3, 5

[21] Y. Wang, X. Wu, D. Hu, Using Randomized Response for Differential Privacy Preserving Data Collection, EDBT/ICDT Workshops, Bordeaux, France, **1558** (2016). ⇒3, 5

[22] L. Zhang, Y. Liu, R. Wang, X. Fu, Q. Lin, Efficient privacy-preserving classification construction model with differential privacy technology, Journal of systems engineering and electronics BIAI, **28,** 1 (2017) 170–178. ⇒4, 5

[23] Z. Zhou, *Ensemble methods: foundations and algorithms*, The CRC press, 2012. ⇒7

[24] T. Zhu, G. Li, W. Zhou, S. Y. Philip, *Differential privacy and applications*, Springer, 2017. ⇒8, 10

Received: January 23, 2021 • Revised: February 28, 2021

Acta Univ. Sapientiae, Informatica **13,** 1 (2021) 21–38

DOI: 10.2478/ausi-2021-0002

Confluence number of certain derivative graphs

Johan KOK
Independent Mathematics Researcher,
City of Tshwane, South Africa &
Visiting Faculty at CHRIST (Deemed to be a University), Bangalore, India
email: jacotype@gmail.com

Joseph SHINY
Mathematics Research Center,
Mary Matha Arts and Science College
Mananthavady, Kerala, India
email: shinyjoseph314@gmail.com

Abstract. This paper furthers the study on the confluence number of a graph. In particular results for certain derivative graphs such as the line graph of trees, cactus graphs, linear Jaco graphs and novel graph operations are reported.

1 Introduction

Concepts, notation and graph parameters without formal definitions can be clarified in [3, 4, 12]. Unless stated otherwise, graphs will be finite, undirected and non-complete, connected simple graphs. A shortest path having end vertices u and v is denoted by $u-v_{(\text{in } G)}$. If $d_G(u,v) \geq 2$ then a vertex w on $u-v_{(\text{in } G)}$, $w \neq u$, $w \neq v$ is called an internal vertex on $u-v_{(\text{in } G)}$. When the context is clear the notation such as $d_G(u,v)$, $deg_G(v)$ can be abbreviated to $d(u,v)$, $deg(v)$ and so on.
The notion of a confluence set (a subset of vertices) of a graph G was introduced in [11]. For a non-complete graph G a non-empty subset $\mathcal{X} \subseteq V(G)$ is said to be a confluence set if for every unordered pair $\{u,v\}$ of distinct vertices

Computing Classification System 1998: G.2.2
Mathematics Subject Classification 2010: 05C07, 05C12
Key words and phrases: Confluence set, minimum confluence set, confluence number.

(if such exist) in $V(G)\backslash\mathcal{X}$ for which $d_G(u,v) \geq 2$ there exists at least one $u - v_{(\text{in } G)}$ with at least one internal vertex $w \in \mathcal{X}$. Also any vertex $u \in \mathcal{X}$ is called a *confluence vertex* of G. A minimal confluence set $\mathcal{X}$ has no proper subset which is a confluence set of G. The cardinality of a minimum confluence set denoted by $\mathcal{C}$ (also called confluence set $\mathcal{C}$ when context is clear) is called the *confluence number* of G and is denoted by $\zeta(G)$. To distinguish between different graphs the notation $\mathcal{C}_G$ may be used for a minimum confluence set of G.

2 Preliminaries

Recall that the line graph $L(G)$ of graph G is obtained by letting $V(L(G)) = \{e_i' : e_i'$ a vertex representation of the edge e_i iff $e_i \in E(G)\}$ and $E(L(G)) = \{e_i'e_j' :$ iff e_i, e_j share a common end-vertex$\}$. A number of known results related to a line graph and which are important for this paper are recalled throughout.

Theorem 1 [10] *For a path* P_n, $n \geq 2$, $L(P_n) = P_{n-1}$.

A corollary is immediate from Theorem 1.

Corollary 2 *For a path* P_n, $n \geq 2$*:*

$$\zeta(L(P_n)) = \begin{cases} 0, & \textit{if } n = 2; \\ \lfloor \frac{n-1}{3} \rfloor, & \textit{if } n \geq 3. \end{cases}$$

Proof. The result follows from Theorem 1 read together with Theorem 5 in [11] i.e.:

$$\zeta(P_n) = \begin{cases} 0, & \text{if } n = 1 \text{ or } 2; \\ \lfloor \frac{n}{3} \rfloor, & \text{if } n \geq 3. \end{cases}$$

□

A subsequent corollary is trivial.

Corollary 3 *For a path* P_n, $n \geq 2$ *it follows that* $\zeta(P_n) > \zeta(L(P_n))$ *if* $n = 3t$, $t = 1, 2, 3, \ldots$ *Else,* $\zeta(P_n) = \zeta(L(P_n))$.

Theorem 4 *A path* P_n *has a unique minimum confluence set* $\mathcal{C}_{P_n}$ *if and only if* $n = 5 + 3i$, $i = 0, 1, 2, \ldots$

Proof. Let path P_n be on the consecutive vertices $v_1, v_2, v_3, \ldots, v_n$. For P_5 it is trivial that $\{v_3\}$ is the unique minimum confluence set.

It is easy to verify that if $v_i, v_j \in \mathcal{C}_{P_n}$ then $d(v_i, v_j) \leq 3$. By ensuring $d(v_i, v_j) = 3$ to a maximum results in a minimum confluence set for P_n. In particular for $n = 5 + 3i$, $i = 1, 2, 3, \ldots$ the selection of a minimum confluence set yields a unique minimum confluence set for P_n. This settles the "if".
The converse follows inherently from the minimum confluence set selection procedure and the well-defined value $n = 5 + 3i$, $i = 0, 1, 2, \ldots$ in the first part. □

Counter example. For illustrative purpose consider $P_6 = v_1v_2v_3v_4v_5v_6$. It is easy to see that amongst others, the sets $\{v_3, v_4\}$, $\{v_3, v_5\}$ and $\{v_3, v_6\}$ are non-unique minimum confluence sets of P_6, $6 \neq 5 + 3i$ for any i.

Theorem 5 [10] *A graph G is isomorphic to its line graph if and only if G is a cycle C_n, $n \geq 3$.*

Theorem 5 implies that $C_n \cong L(C_n) \cong L(L(C_n)) \cong \cdots \cong L(L \cdots L(C_n))$. Therefore, $\zeta(C_n) = \zeta(L(C_n))$, $n \geq 3$. With only a change in notation we adopt the result from [11].

Theorem 6 *For a cycle C_n, $n \geq 3$:*

$$\zeta(L(C_n)) = \begin{cases} 0, & \textit{if } n = 3; \\ 1, & \textit{if } n = 4; \\ \lceil \frac{n}{3} \rceil, & \textit{if } n \geq 5. \end{cases}$$

The next lemma is a consequence of the definition of a minimum confluence set.

Lemma 7 *Consider a graph G with at least two vertices u, v, $d_G(u, v) = 2$. Amongst all possible $u - v_{(\text{in } G)}$ there exists at least one such path say, uwv such that u or w or v an element(s) of $\mathcal{C}_G$.*

A vertex v to which a leaf (pendent) vertex u is attached is called the *pre-leaf* of u or simply, *pre-leaf* v.

Lemma 8 *For any graph G of order $n \geq 3$ which has a leaf u there exists a confluence set $\mathcal{C}$ (a minimum) such that $u \notin \mathcal{C}$.*

Proof. Let a leaf $u \in \mathcal{C}$ and let v be the pre-leaf vertex of u. Clearly if both $u, v \in \mathcal{C}$ then $\mathcal{C}$ is not minimal thus not minimum. Therefore, $v \notin \mathcal{C}$ hence $(\mathcal{C} - u) \cup \{v\}$ remains a minimum confluence set of T. This suffices to settle the result. □

Lemma 8 implies that since a pendent vertex cannot be an internal vertex on a path it need not by necessity be an element of a minimum confluence set $\mathcal{C}$ of any graph.
Let the *pendent degree* of vertex $u \in V(G)$ denoted by $\deg_p(u)$ be the number of pendent vertices adjacent to u. The vertex set $V(G)$ can be partitioned into $X_1 = \{u : \deg_p(u) = 0\}$, $X_2 = \{v : \deg_p(v) = 1\}$ and $X_3 = \{w : \deg_p(w) \geq 2\}$. A trivial lower bound establishes i.e. $\zeta(G) \geq |X_3|$. Put differently, there exists a minimum confluence set $\mathcal{C}$ of T such that $X_3 \subseteq \mathcal{C}$. The aforesaid is true because the unique shortest path between any two pendent vertices sharing a common pre-leaf w permits $w \in \mathcal{C}$. For a tree T this lower bound can be useful and equality can hold. For example for a star, $S_{1,n} \cong K_{1,n}$. Let $X'_1 = \{\text{leafs of } T\}$. Note that $X'_1 \subseteq X_1$.

Proposition 9 *If for a tree* T *the vertex set partition is such that* $V(T) = X'_1 \cup X_3$ *then* $\zeta(T) = |X_3|$.

Proof. Since $V(T) = X'_1 \cup X_3$ the vertex set $V(T)$ consists of only leafs and pre-leafs. Furthermore, each pre-leaf u has at least two leafs say, v, w. There exist a unique $v - w_{(in\ T)}$ with the unique internal vertex u so it follows by necessity that $u \in \mathcal{C}$ so, $\zeta(T) \geq |X_3|$. By Lemma 8 a leaf is not by necessity in $\mathcal{C}$ so $\zeta(T) \leq |X_3|$. Therefore $\zeta(T) = |X_3|$. □

2.1 Heuristic method to obtain $\zeta(T)$

The path P_n is called a n-path. For paths P and Q we define $P \cap Q = V(P) \cap V(Q)$. For different paths and not necessarily of different order, P_{n_1}, P_{n_2}, $P_{n_3}, \ldots$, P_{n_k} we define $\bigcap\limits_{i=1}^{k} P_{n_i} = (((P_{n_1} \cap P_{n_2}) \cap P_{n_3}) \cap \cdots \cap P_{n_k})$.
Heuristic A. For a vertex labeled tree T of order $n \geq 3$ do:
Step 1A. Let $V(T) = \{v_1, v_2, v_3, \ldots, v_n$ with v_n a leaf$\}$. Minimizing by isomorphism let $\mathfrak{P}$ be maximal with $\mathfrak{P} = \{$all 3-paths $v_1v_iv_j,\} \cup \{$all 3-paths $v_2v_kv_l,\} \cup \cdots \cup \{$all 3-paths $v_{n-1}v_mv_q\}$.
Step 2A: Consider the vertex subset of $V(T)$ i.e. $X_3 = \{w : \deg_p(w) \geq 2\}$.
Step 3A: For all $x \in X_3$ remove the 3-paths in $\mathfrak{P}$ which contains vertex x to obtain $\mathfrak{P}'$.
Step 4A. Partition $\mathfrak{P}'$ into a minimum partition of 3-path subsets such that, within a 3-path subset all 3-paths share at least a common vertex except for singleton subsets. Denote this partition by $\mathcal{P}(\mathfrak{P}')$.
Step 5A. Let the number of 3-path subsets in $\mathcal{P}(\mathfrak{P}')$ be ℓ. Then $\zeta(T) = \ell + |X_3|$.

Furthermore, for a singleton 3-path subset in $\mathcal{P}(\mathfrak{P}')$ any one vertex on the 3-path is permitted in $\mathcal{C}_T$. For 3-path subsets with two or more 3-paths which share two common vertices, any one of the two vertices is permitted in $\mathcal{C}_T$. For 3-path subsets with two or more 3-paths which share one common vertex, only the common vertex is permitted in $\mathcal{C}_T$.
Note that prescribing v_n to be a leaf is only a matter of convenience. Furthermore, besides obtaining the confluence number the heuristic yields a valid minimum confluence set. In real world applications this additional result can be worthy.

Theorem 10 *Heuristic A is valid.*

Proof. Since the 3-path between any two vertices in T is unique, every 3-path in Step 1A is unique. Lemma 7 implies that no 3-path can exists without at least one vertex in $\mathcal{C}_T$. Therefore a set $\mathcal{X} = \{v_i : v_i$ a central vertex of a 3-path in $\mathfrak{P}\}$ is a confluence set of T. Hence, the validity of the Heuristic A follows immediately from minimizing $\mathfrak{P}$ by isomorphism in Step 1A and the minimization of partition $\mathfrak{P}'$ in Step 4A to yield $\mathcal{C}_T$. □

Applying Heuristic A to a general graph will yield a confluence set denoted by $\mathcal{X}_h$. It follows trivially that $\zeta(G) \leq \mathcal{X}_h$. The advantage of the aforesaid approach for a real world application is that after a $\mathcal{X}_h$ has been obtained, $\mathcal{X}_h$ can be minimized to yield a valid minimum confluence set $\mathcal{C}_G$.
Example: Consider the tree T of order 12 in figure 1.
Step 1A. Minimizing by isomorphism the maximum set $\mathfrak{P}$ of 3-paths (elements) is,
$\mathfrak{P} = \{v_1v_3v_2,\ v_1v_3v_4,\ v_2v_3v_4,\ v_3v_4v_5,\ v_3v_4v_6,\ v_4v_6v_7,\ v_5v_4v_6,\ v_6v_7v_8,\ v_6v_7v_9,\ v_7v_9v_{10},\ v_7v_9v_{11},\ v_7v_9v_{12},\ v_8v_7v_9,\ v_{10}v_9v_{11},\ v_{10}v_9v_{12},\ v_{11}v_9v_{12}\}$.
Step 2A. $X_3 = \{v_3, v_9\}$.
Step 3A. After removing all 3-paths which have vertices v_3 or v_9 the set $\mathfrak{P}' = \{v_4v_6v_7, v_5v_4v_6, v_6v_7v_8\}$ is obtained.
Step 4A. In respect of $\mathfrak{P}'$ a minimum partition of 3-path subsets such that, within a 3-path subset all 3-paths share a single vertex is
$\mathcal{P}(\mathfrak{P}') = \{\{v_4v_6v_7, v_5v_4v_6, v_6v_7v_8\}\}$. Therefore, $\ell = 1$.
Step 5A. The result $\zeta(T) = 1 + 2 = 3$. Moreover, a minimum confluence set is $\mathcal{C}_T = \{v_3, v_6, v_9\}$.

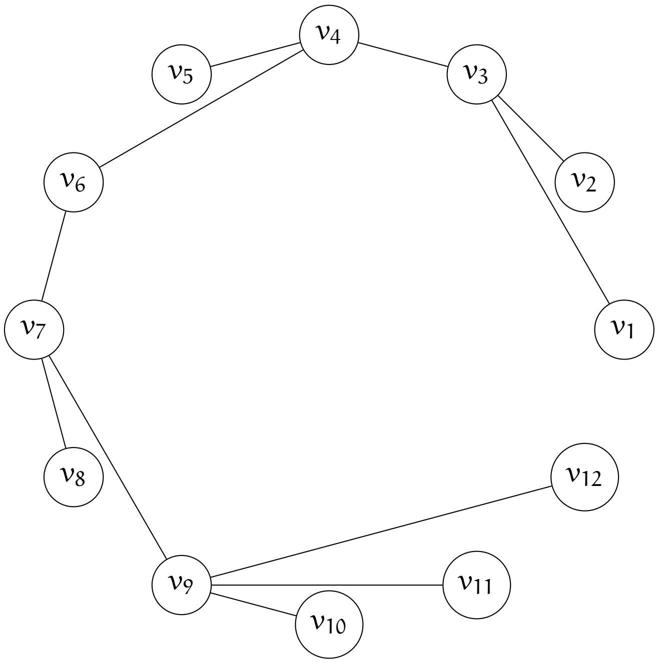


Figure 1: Tree T of order 12.

3 Line graph of trees

From an intersection graph perspective we recall an important definition.

Definition 11 *Let $\mathfrak{C}$ be a non-empty set of non-empty subgraphs of G. Then let each element (subgraph) in $\mathfrak{C}$ be represented by a unique vertex say v_i. Hence, $v_i \in \mathfrak{C}$, $i = 1, 2, 3, \ldots, |\mathfrak{C}|$ has well-defined meaning. Define the derivative graph $G(\mathfrak{C})$ on the vertex set $\mathfrak{C}$ with edge set $E(G(\mathfrak{C})) = \{v_iv_j :$ if and only if $v_i \neq v_j$ and v_i, v_j satisfy some adjacency condition$\}$.*

Note that an *adjacency condition* can be a condition such as, conventional adjacency between vertices or edges incident with a common vertex. However, in an abstract sence, it could be subsets $X_i \subset V(G)$, $i \geq 2$ such that $\bigcup\limits_{all\ i} X_i = V(G)$ and $X_i \cap X_i = \emptyset$, $i \neq j$. Applications in abstract algebra present numerous innovative 'adjacency conditions'.
For a graph G let $\mathfrak{C} = E(G)$. Then the line graph of G denoted by $L(G)$ is defined by $L(G) = G(\mathfrak{C})$ with $V(L(G)) = \mathfrak{C}$ and $E(L(G)) = \{v_iv_j :$ if and only if v_i, v_j are adjacent in $G\}$.

In the literature there exist different views on the distinction between *block graphs* and *cactus graphs*. This section will accept the view that a block graph is a graph whose blocks are cliques. Furthermore, in a block graph the intersection between any two distinct blocks are either empty or a cut vertex. Note that a tree T is a block graph. See a good characterization of block graphs in [2]. A path P_n, $n \geq 3$ which has an end-vertex merged with a vertex of G which is not a path itself and the other end-vertex of the path remains pendent is a *beam* of G.

For a graph $G \ncong P_n$, $n \geq 3$ the line graph $L(G)$ typically has a combination of the structural elements (not necessarily all), (i) cliques of *order* ≥ 3 which share a common vertex (called a clique-cut vertex) and/or (ii) cliques of *order* ≥ 3 which are connected by an edge (a clique-cut edge) and/or (iii) cliques of *order* ≥ 3 which are connected by a 3-path (end-vertices in the respective cliques) and/or (iv) cliques of *order* ≥ 3 which are connected by a 4-path (end-vertices in the respective cliques) and/or (v) cliques of *order* ≥ 3 which are connected by a k-path, $k \geq 5$ (end-vertices in the respective cliques) and/or (vi) cliques of *order* ≥ 3 with beams and/or (vii) pendent vertices.

For a tree $T \ncong P_n$, $n \geq 4$ and $T \ncong K_{1,m}$, $m \geq 3$ the line graph has a "tree-like" graphical embodiment i.e. besides cycles in cliques of *order* ≥ 3, the line graph $L(T)$ is acyclic. It typically has a combination of the structural elements (not necessarily all), (i) cliques of *order* ≥ 3 which pairwise share a unique common vertex (called a clique-cut vertex) and/or (ii) cliques of *order* ≥ 3 which are pairwise connected by an unique edge (a clique-cut edge) and/or (iii) cliques of *order* ≥ 3 which are pairwise connected by an unique k-path, $k \geq 3$ (end-vertices in the respective cliques and called a clique k-path) and/or (iv) cliques of *order* ≥ 3 with beams such that, one beam is attached to one clique vertex (clique-beam) and/or (v) cliques of *order* ≥ 3 with pendent vertices such that one pendent vertex is adjacent to one clique vertex (clique-pendent vertex).

Let a clique (or block) $Q_{n,i}$ of order $n \geq 2$ and $i \in \mathbb{N}$ an identifier be represented by a vertex $v_{Q_{n,i}}$. Derive the following graph from $L(T)$. Replace each clique $Q_{n,i}$ with a vertex $v_{Q_{n,i}}$ and add the edge between adjacent $v_{Q_{n,i}}$ and $v_{Q_{n,j}}$. If a clique $Q_{n,i}$ has only one cut vertex then add all non-cut vertices as leafs to $v_{Q_{n,i}}$. If a clique $Q_{n,j}$ has two or more cut vertices then delete all non-cut vertices and all edges from $Q_{n,i}$. Clearly a tree results from this operation. This tree is called the *confluence tree* of $L(T)$ and denoted by $\mho(L(T))$.

3.1 Heuristic B for minimum confluence set of line graph of tree

For the line graph $\mathsf{L}(\mathsf{T})$, $\mathsf{T} \ncong \mathsf{P}_n$, $n \geq 4$ and $\mathsf{T} \ncong \mathsf{K}_{1,m}$, $m \geq 3$ do:
Step 1B. Construct the confluence tree $\mho(\mathsf{L}(\mathsf{G}))$ and relabel the vertices accordingly, $u_1, u_2, u_3, \dots, u_t$.
Step 2B. Apply Heuristic A to obtain $\zeta(\mho(\mathsf{L}(\mathsf{T})))$
Step 3B. Yield $\zeta(\mathsf{L}(\mathsf{T})) = \zeta(\mho(\mathsf{L}(\mathsf{T})))$.

Theorem 12 *Heuristic B is valid.*

Proof. The result for $\zeta(\mathsf{L}(\mathsf{P}_n))$, $n \geq 2$ is provided by Corollary 2. Hence, the exclusion of paths is justified. The line graph $\mathsf{L}(\mathsf{K}_{1,m})$, $m \geq 3$ is complete. Therefore the exclusion is justified.

Furthermore, in $\mathsf{L}(\mathsf{T})$ all 3-paths, if any, from a non-cut vertex of a clique must transverse through a cut vertex of the clique. Therefore, the vertex $v_{\mathsf{Q}_{n,i}}$ represents all cut vertices in $\mathsf{Q}_{n,i}$ definitively with regards to the definition of a confluence set. Therefore, Heuristic B is valid. Hence, $\zeta(\mathsf{L}(\mathsf{T})) = \zeta(\mho(\mathsf{L}(\mathsf{T})))$. □

3.2 Novel graph operations

Two finite sets X and Y of equal cardinality is said to be *identical* if and only if $|X \cap Y| = |X|$, (or $|X \cap Y| = |Y|$). Otherwise the sets are distinct. Let $\mathbb{C}(\mathsf{G}) = \{$distinct minimum confluence sets of $\mathsf{G}\}$. Clearly, for a finite graph G the set $\mathbb{C}(\mathsf{G})$ exists and is finite. If a pendent vertex u is attached to vertex w of graph G the operation is denoted by $\mathsf{G}(w) \multimap u$.

Theorem 13 *For any graph* G *it follows that*

$$\zeta(\mathsf{G}) \leq \zeta(\mathsf{G}(w) \multimap u) \leq \zeta(\mathsf{G}) + 1.$$

Proof. Consider $\mathsf{G}(w) \multimap u$. If a confluence set $\mathcal{C} \in \mathbb{C}(\mathsf{G})$ exists such that $\forall v \in V(\mathsf{G})$ some shortest path $\mathsf{P} = v - u_{(\text{in } \mathsf{G}(w) \multimap u)}$ exists such that P has a confluence vertex $x \in N_{\mathsf{G}}[w]$ then $\zeta(\mathsf{G}(w) \multimap u) = \zeta(\mathsf{G})$. If no such set exists then for any $\mathcal{C} \in \mathbb{C}(\mathsf{G})$ a set $\mathcal{C} \cup \{w\}$ is a minimum confluence set of $\mathsf{G}(w) \multimap u$. Therefore, $\zeta(\mathsf{G}(w) \multimap u) = \zeta(\mathsf{G}) + 1$. Hence, $\zeta(\mathsf{G}) \leq \zeta(\mathsf{G}(w) \multimap u) \leq \zeta(\mathsf{G}) + 1$. □

For graphs G and H let $u \in V(\mathsf{G})$, $v \in V(\mathsf{H})$. By adding the edge uv to connect graphs G and H the graph $\mathsf{G}(u) \leftrightsquigarrow \mathsf{H}(v)$ is obtained. The edge uv is said to *edge-merge* graphs G and H. Reversing the operation is simply called *edge-unmerging* or *unmerging* if the context is clear.

Theorem 14 *For* $G(u) \leftrightsquigarrow H(v)$ *provided that* $G \neq K_1$ *and* $H \neq K_1$ *it follows that*

$$\zeta(G) + \zeta(H) \leq \zeta(G(u) \leftrightsquigarrow H(v)) \leq \zeta(G) + \zeta(H) + 1.$$

Proof. (a) If a confluence set $\mathcal{C}_1 \in \mathbb{C}(G)$ or $\mathcal{C}_2 \in \mathbb{C}(H)$ exists such that $u \in \mathcal{C}_1$ or $v \in \mathcal{C}_2$ then $\zeta(G(u) \leftrightsquigarrow H(v)) = \zeta(G) + \zeta(H)$.
(b) If in G and H respectively, $N_G(u) \subseteq \mathcal{C}_1$ for some $\mathcal{C}_1 \in \mathbb{C}(G)$ and $N_H(v) \subseteq \mathcal{C}_2$ for some $\mathcal{C}_2 \in \mathbb{C}(H)$ then $\zeta(G(u) \leftrightsquigarrow H(v)) = \zeta(G) + \zeta(H)$.
(c) If both (a) and (b) fail it implies that $\forall \mathcal{C}_1 \in \mathbb{C}(G)$ and $\forall \mathcal{C}_2 \in \mathbb{C}(H)$ there exist vertices $x \in N_G(u)$, $y \in N_H(v)$ such that the (x, y)-path (shortest) does not contain a confluence vertex of either G or H. Therefore, and without loss of generality a set $\mathcal{C}_1 \cup \mathcal{C}_2 \cup \{u\}$ is a minimum confluence set of $\zeta(G(u) \leftrightsquigarrow H(v))$. The reasoning through (a), (b), (c) suffices to settle the result. $\square$

Definition 15 *Let* $S = (G_1, G_2, G_3, \ldots, G_\ell)$*,* $\ell \geq 2$ *be an ordered string of graphs. Each graph corresponds to order* n_i*,* $1 \leq i \leq \ell$ *with corresponding vertex sets* $V(G_i) = \{v_{i,j} : 1 \leq j \leq n_j\}$*. For ordered pairs of vertices* $(v_{t,j}, v_{t+1,k})$*,* $1 \leq t \leq \ell - 1$ *and* $j \in \{1, 2, 3, \ldots, n_t\}$*,* $k \in \{1, 2, 3, \ldots, n_{t+1}\}$ *the graph* $G_{\boxdot \ell} = (((G_1 \boxdot G_2) \boxdot G_3) \cdots \boxdot G_\ell)$ *is obtained by merging each pair of ordered vertices* $(v_{t,j}, v_{t+1,k})$*. This new graph is called a* ℓ*-sliced graph.*

Note that Definition 15 implies that all edges of the respective graphs G_i, $1 \leq i \leq \ell$ are retained. The merged vertex corresponding to say, $v_{t,j}$ and $v_{t+1,k}$ serves as a common end-vertex for all edges incident to $v_{t,j}$ and $v_{t+1,k}$. Reversing the merging operation is simply called *vertex-unmerging* or *unmerging* if the context is clear. By convention any graph G *per se* and in particular a graph without a cut vertex are said to be a 1-sliced graph.
Let T be a tree on at least $n \geq 3$ vertices. Let $n = q + p$ with q the number of pendent vertices. The next corollary is stated without proof as it is deemed to be self-evident.

Corollary 16 *For any tree* T *of order* $n \geq 3$ *(a lower bound for convenience) the line graph* $L(T)$ *is a block graph which is a* p*-sliced graph.*

Theorem 17 *Consider graph* G *of order* n *with at least one cut vertex. Unmerge sufficient cut vertices to obtain a maximum of* t *components. Then the graph* G *is a maximum* t*-sliced graph.*

Proof. Let v be a cut vertex of G. Let the components of $G - v$ (cut-vertex deletion) be on vertices $V(G_1)$, $V(G_2), \ldots V(G_r)$, $r \geq 2$. Consider the induced subgraphs of G i.e. $H_1 = \langle V(G_1) \cup \{v\} \rangle$, $H_2 = \langle V(G_2) \cup \{v\} \rangle, \dots, H_t = \langle V(G_r) \cup \{v\} \rangle$. Clearly, in respect of $v \in V(H_1)$ and $v \in V(H_2)$ and $\dots$, and $v \in V(H_r)$ the graph obtained recursively, $(((H_1 \boxdot H_2) \boxdot H_3) \cdots \boxdot H_r)$ is isomorphic to $\cong G$. In the case of removing $k \geq 2$ cut vertices to obtain the maximum t components the result follows through immediate induction. □

The notation $G_{\boxdot \ell} = (((G_1 \boxdot G_2) \boxdot G_3) \cdots \boxdot G_\ell)$ *per se* does not specify the pairs of vertices under operation. Definition 13 requires that the pairs be specified prior. For application we consider only 2-sliced graphs in respect of $u \in V(G)$ and $v \in V(H)$ denoted by, $G(u) \boxdot H(v)$. The result can be applied recursively to any ℓ-sliced graph, $\ell \geq 3$.

Theorem 18 *For graphs G and H and provided that $G \neq K_1$ and $H \neq K_1$ and; without loss of generality, $G \neq K_2$ and $H \neq K_1$ it follows that*

$$\zeta(G) + \zeta(H) - 1 \leq \zeta(G(u) \boxdot H(v)) \leq \zeta(G) + \zeta(H) + 1.$$

Proof. (a) If confluence sets $\mathcal{C}_1 \in \mathbb{C}(G)$ and $\mathcal{C}_2 \in \mathbb{C}(H)$ exist such that $u \in \mathcal{C}_1$ and $v \in \mathcal{C}_2$ then $\zeta(G(u) \boxdot H(v)) = \zeta(G) + \zeta(H) - 1$.
(b) If a confluence set $\mathcal{C}_1 \in \mathbb{C}(G)$ exists such that $u \in \mathcal{C}_1$ and $v \notin \mathcal{C}_2$ $\forall \mathcal{C}_2 \in \mathbb{C}(H)$ then $\zeta(G(u) \boxdot H(v)) = \zeta(G) + \zeta(H)$.
(c) If in G and H respectively, $N_G(u) \subseteq \mathcal{C}_1$ for some $\mathcal{C}_1 \in \mathbb{C}(G)$ or $N_H(v) \subseteq \mathcal{C}_2$ for some $\mathcal{C}_2 \in \mathbb{C}(H)$ then $\zeta(G(u) \boxdot H(v)) = \zeta(G) + \zeta(H)$.
(d) If (a), (b) and (c) fail it implies that $\forall \mathcal{C}_1 \in \mathbb{C}(G)$ and $\forall \mathcal{C}_2 \in \mathbb{C}(H)$ there exist vertices $x \in N_G(u)$, $y \in N_H(v)$ such that the (x, y)-path (shortest) does not contain a confluence vertex of either G or H. Therefore, and without loss of generality a set $\mathcal{C}_1 \cup \mathcal{C}_2 \cup \{u\}$ is a minimum confluence set of $\zeta(G(u) \boxdot H(v))$. Hence, $\zeta(G(u) \boxdot H(v)) = \zeta(G) + \zeta(H) + 1$.
The reasoning through (a), (b), (c) and (d) suffices to settle the result. □

Note that Theorem 18 has an intriguing application to $K_n \boxdot K_m$, $n \geq 2$, $m \geq 2$. By convention $\zeta(K_n) = 0$, $n \geq 1$. Hence, by convention the minimum confluence set $\mathcal{C}_{K_n} = \emptyset$. Therefore, Theorem 18(d) finds application.

4 Cactus graphs

For the purpose of studying cactus graphs (simply a cactus) it is noted that a cycle is meant to be a simple cycle or put differently, a chordless cycle. A cactus is a connected graph with at least one cycle and any two distinct cycle share at most one common vertex. Generally a cactus is denoted by C_{ac}. The cycle C_3 is considered a trivial cactus. A well studied family of cactus graphs called the friendship graphs or n-fan graphs denoted by F_n belongs to a wider family $\mathcal{F} = \{F^{(n)} : F^{(n)}$. These graphs consist of n cycles each of *order* ≥ 3 which share a common vertex$\}$. Note that in a $F^{(n)}$ the respective order of distinct cycles may differ.

Proposition 19 *For* $F^{(n)} \in \mathcal{F}$ *and we have:*
(a) $\zeta(F^{(n)}) = 1$ *if* $F^{(n)} = (((\underbrace{C_3 \boxdot C_3) \boxdot C_3) \cdots \boxdot C_3}_{n-\text{copies}})$,
(b) $\zeta(F^{(n)}) = n\zeta(C_m) - (n-1)$ *if* $F^{(n)} = (((\underbrace{C_m \boxdot C_m) \boxdot C_m) \cdots \boxdot C_m}_{n-\text{copies}})$, $m \geq 4$,
(c) $\zeta(F^{(n)}) = \sum\limits_{i=1}^{n} \zeta(C_{m_i}) - (n-1)$ *if* $F^{(n)} = (((C_{m_1} \boxdot C_{m_2}) \boxdot C_{m_3}) \cdots \boxdot C_{m_n})$, $m_i \geq 4$,
(d) $\zeta(F^{(n)}) = \sum\limits_{i=1}^{n-t} \zeta(C_{m_i}) - (n-t-1)$ *if* $F^{(n)} = (((C_{m_1} \boxdot C_{m_2}) \boxdot C_{m_3}) \cdots \boxdot C_{m_n})$, $m_i \geq 4$ *for* $1 \leq i \leq n-t$ *and* t *cycles are copies of* C_3.

Proof. (a) The proof will be by immediate induction. Consider two copies of C_3. Let first copy be on vertices v_1, v_2, v_3 and second copy on u_1, u_2, u_3. Without loss of generality let vertices v_1, u_1 merge as the common vertex in $C_3 \boxdot C_3$. Label this common vertex as w_1. Although $\zeta(C_3) = 0$ the existence of a 3-path say, $v_2 w_1 u_2$ necessitates $\mathcal{C}_{C_3 \boxdot C_3} = \{w_1\}$. By iteratively constructing $(((\underbrace{C_3 \boxdot C_3) \boxdot C_3) \cdots \boxdot C_3}_{n-\text{copies}})$ it follows through immediate induction that $\zeta(F^{(n)}) = 1$.
(b) Let the vertices of the i^{th}-copy of C_m be labeled $v_{i,j}$, $j = 1, 2, 3, \ldots, m$ and select a confluence set labeled $\mathcal{C}^i_{C_m}$ for each $1 \leq i \leq n$. Through stepwise rotation of the vertex labeling say, clockwise, it is possible to let $v_{1,i} \in \mathcal{C}^i_{C_m}, \forall\, i$. By merging vertices $v_{1,1}, v_{2,1}$ to yield w_1 in $C_m \boxdot C_m$ it follows that $\zeta(C_m \boxdot C_m) = 2\zeta(C_m) - 1 = 2\zeta(C_m) - (2-1)$. After following similar iterative procedure the result $\zeta(F^{(n)}) = n\zeta(C_m) - (n-1)$ follows through immediate induction.
(c), (d). These results follow through similar reasoning found in parts (a) and (b). □

Note that in the first iteration in the proof of Proposition 19(a), Theorem 18(d) applies. In the induction step, Theorem 18(b) applies. In the proofs of parts (b) through (d) Theorem 18(a) applies.
Essentially $G \boxdot H$ means that graphs G and H are connected by merging two vertices $v \in V(G)$ and $u \in V(H)$ which results in a cut vertex say w. The term *unmerging* w means to disconnect G and H by cloning w into say,

$v \in V(G)$ and $u \in V(H)$. Furthermore, if a combination of more than three cycles and/or trees share a common vertex w then unmerging w may occur for *graphical subsets*. It means that if say tree T and cycles C_n, C_m share vertex w, the graphical subsets say $\{T\}$, $\{C_n, C_m\}$ can unmerge to render $C_n \boxdot C_m$ with common vertex w and T unmerges with new vertex say v substituting w. Consider a cactus. Besides the wider meaning of connectivity we specialize to say, two connected cycles will mean *directly connected* through either a common vertex (cut vertex) or a common edge (cut edge). If a cycle is connected to a tree it means *directly connected to a maximal tree*. Hence, the maximal tree and the cycle only share a common vertex. Two cycles connected by an edge may only unmerge by edge deletion. The next corollary is deemed to be self-evident.

Corollary 20 *Any cactus C_{ac} can be unmerged into a minimum number of maximal graphical structures i.e. cycles, maximal trees and edges.*

Theorem 21 (Existence theorem) *There exists a heuristic algorithm to find both the confluence number and a minimum confluence set of a cactus.*

Proof. It is known from [11] that finding a minimum confluence set of a graph G is NP-complete. Iterative pairwise unmerging of a cactus C_{ac} in an arbitrary order to find a minimum number of maximal graphical structures i.e. cycles, maximal trees and edges is possible. Pairwise unmerging is possible in a finite number of steps and the solution (cluster of graphical structures) is unique. Let the arbitrary ordered pairwise unmerging steps be s_1,s_2,. . . , s_ℓ. Furthermore, an ordered set of prescriptions exist to orderly reconstruct the cactus through pairwise merging. The prescriptions are in the order s_ℓ^{-1},$s_{\ell-1}^{-1}$,. . . , s_1^{-1}. Therefore, the original cactus is uniquely reconstructed. Let a merging operation be $f \in \{\boxdot, \leftrightsquigarrow\}$.
Step s_ℓ^{-1}. Label the one graphical structure G_1 and the other H_1. Apply $s_\ell^{-1} = G_1 f H_1$ to obtain G_2. Depending on f, G_1, H_1 any one of Theorems 14 or 18 ensures that a minimal confluence set and the confluence number of G_2 can be found.
Step $s_{\ell-1}^{-1}$. Consider G_2 and the appropriate graphical structure H_2 prescribed by $s_{\ell-1}^{-1}$. Apply $s_{\ell-1}^{-1} = G_2 f H_2$ to obtain G_3. Depending on f, G_2, H_2 any one of Theorems 14 or 18 ensures that a minimal confluence set and the confluence number of G_3 can be found.
Through immediate induction it follows that through inversing the merges, s_ℓ^{-1},$s_{\ell-1}^{-1}$,. . . , s_1^{-1} the confluence number and a minimum confluence set of a cactus can be found. Thus a heuristic algorithm exists. □

5 Linear Jaco graphs

Linear Jaco graphs was introduced by Kok et.al, [5, 6, 7, 9]. Linear Jaco graphs are digraphs. Therefore the definitions define arcs. It is important to note that in the notation $J_n(f(x))$ means that a Jaco graph is of order n and for vertex v_i the vertex degree is bounded by $\mathrm{deg}(v_i) \leq f(i)$. For the family of linear Jaco graphs $f(x)$ is restricted to linear functions. Reference to the definitions will assist in some proofs to follow. For ease of reference we recall the formal definitions.

Definition 22 *The infinite linear Jaco graph* $J_\infty(x)$, $x \in \mathbb{N}$ *is defined by* $V(J_\infty(x)) = \{v_i : i \in \mathbb{N}\}$, $A(J_\infty(x)) \subseteq \{(v_i, v_j) : i, j \in \mathbb{N}, i < j\}$ *and* $(v_i, v_j) \in A(J_\infty(x))$ *if and only if* $2i - d^-(v_i) \geq j$.

Definition 23 *The family of finite linear Jaco graphs is defined by* $\{J_n(x) \subseteq J_\infty(x) : n, x \in \mathbb{N}\}$. *A member of the family is referred to as the Jaco graph,* $J_n(x)$.

We shall only consider the underlying graph (undirected). See figure 2. However, within the context the notions of in-degree $\mathrm{deg}^-(v_i)$ and out-degree $\mathrm{deg}^+(v_i)$ remains relevant. Since the linear Jaco graphs for $n = 1, 2, 3, 4$ are K_1, K_2, P_3, P_4 respectively. Because $d^+(v_3) = 2 \geq 2$ the lower bound $n \geq 3$ will be considered.

For $J_5(x)$ vertex v_3 is a confluence vertex therefore $\mathcal{C}_{J_5(x)} = \{v_3\}$. To constructively find a minimum confluence set of $J_n(x)$, $n \geq 3$ we call vertex v_3 the *prime confluence vertex* of all linear Jaco graphs. When we *skip or hop* from say vertex v_i to v_j (not necessarily adjacent) it is denoted by $v_{i \curvearrowright j} = v_j$. For example, let $\ell(i) = (i+2) + \mathrm{deg}^+(v_{i+1})$. Then $v_{3 \curvearrowright \ell(3)} = v_8$ and $v_{8 \curvearrowright \ell(8)} = v_{16}$. See table 1, page 71 in [5]. Let an j^{th}-iteration yielding a minimum confluence set be denoted by it_j. Define $\mathrm{it}_1 \rightarrowtail \{v_3\}$, $\mathrm{it}_2 \rightarrowtail \{v_3, v_8\}$ and $\mathrm{it}_{j+1} \rightarrowtail \{v_3, v_8, \ldots, v_{(v_i \curvearrowright \ell(i)) \curvearrowright \ell(v_i \curvearrowright \ell(i))}\}$.

Lemma 24 *For a Jaco graph* $J_n(x)$, $n \geq 3$ *a confluence set is given by*
$X = \{v_3, v_8, v_{8 \curvearrowright \ell(8)}, v_{16 \curvearrowright \ell(16)}, \ldots, v_t = v_{(v_i \curvearrowright \ell(i)) \curvearrowright \ell(v_i \curvearrowright \ell(i))}\}$, $t \leq n$.

Proof. The result may be said to be trivial because for any vertex $v_i \in X$, all vertices v_j, $(i-2) - \mathrm{deg}^-(v_{i-1}) \leq j \leq \ell(i)$ have $d(v_i, v_j) \leq 2$. Hence, by the definition of a confluence set, X is a confluence set of $J_n(x)$. □

The induced path $\langle \{v_1, v_2\} \cup X \rangle$ is called the *confluence path* of $J_n(x)$. It is denoted by $\mho_P(J_n(x))$.

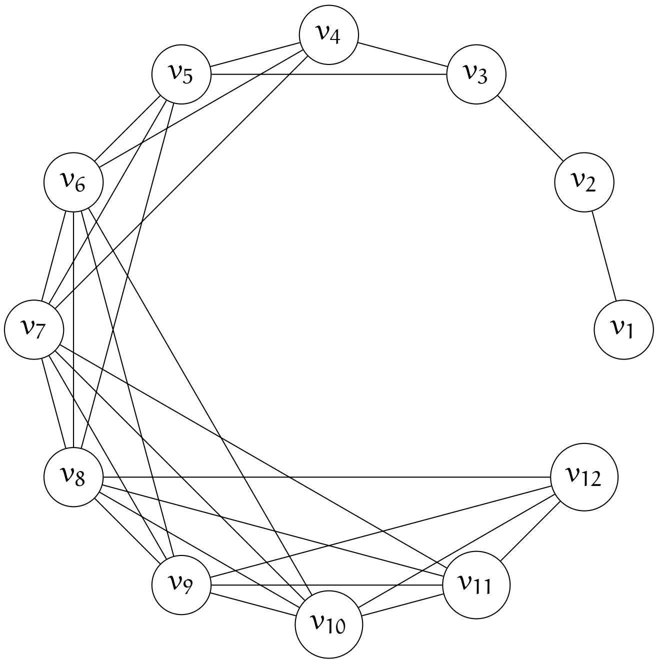


Figure 2: Linear Jaco graph $J_{12}(x)$.

Theorem 25 *For* $n \geq 3$, $\zeta(J_n(x)) = \zeta(\mho_P(J_n(x)))$.

Proof. From Lemma 24, $\zeta(J_n(x)) \leq |X|$. From Definition 22 it follows if $X' = X\backslash\{v_j\}, v_i \in X$ then some vertices v_k, $k < i$ and v_t, $t > i$ exist with $d(v_k, v_t) \geq 3$. Therefore $\zeta(J_n(x)) > |X'|$. Hence, $\zeta(J_n(x)) = |X|$. □

Define sub-families of linear Jaco graphs as follows, $\mathcal{F}_1$ on $n_{1,1} = 3 \leq n \leq 7 = n_{1,2}$ vertices, $\mathcal{F}_2$ on $n_{2,1} = 8 \leq n \leq 15 = n_{2,2}$ vertices, $\mathcal{F}_3$ on $n_{3,1} = \ell(8) \leq n \leq \ell(9) - 1 = n_{3,2}$ vertices, $\ldots, \mathcal{F}_i$ on $n_{i,1} = \ell(n_{i-1,1}) \leq n \leq \ell(n_{i,1}) - 1 = n_{i,2}$ vertices.

Corollary 26 *For* $n \geq 3$ *let* $J_n(x) \in \mathcal{F}_i$ *then* $\zeta(J_n(x)) = i$.

Proof. The results follow by similar reasoning found in the proof of Lemma 24. □

Corollary 27 *For* $J_n(x)$, $n \geq 3$, *Let the diameter of* $J_n(x)$ *be* $diam(J_n(x)) = k$. *Then* $\zeta(J_n(x)) = \lfloor \frac{k+1}{3} \rfloor$.

Proof. From Definition 22 it follows that if vertices v_i, v_j, $i < j$ are adjacent then v_k, v_l, $i \leq k < l \leq j$ are adjacent. There exists a path P_{k+1} (of length

k) such that any $v_i \in V(J_n(x)$ and any vertex v_j on P_{k+1} has $d(v_i, v_j) \leq 2$. Therefore path P_{k+1} is a path of $max\{min\}$ length in $J_n(x)$. The result finally follows from the result for paths. $\square$

To illustrate reasoning in the proof of Corollary 27 see figure 2. Since edge v_6v_{10} exists all pairs of vertices in $\{v_6, v_7, v_8, v_9, v_{10}\}$ are adjacent. Put differently, the induced subgraph $\langle\{v_6, v_7, v_8, v_9, v_{10}\}\rangle \cong K_5$.

6 Conclusion

Theorem 4 characterized paths which have unique minimum confluence sets. Based on this characterization a few research avenues are highlighted.
Problem 1. Characterize graphs with unique minimum confluence sets.
Solving Problem 1 opens an avenue to characterize forbidden graphs in relation to confluence number and minimum confluence set. The next conjecture could be the key.
Conjecture 1. Let $\zeta(G) \geq 2$. A minimum confluence set $\mathcal{C}_G$ of G has, for each confluence vertex $u \in \mathcal{C}_G$ at least one vertex $v_{\neq u} \in \mathcal{C}_G$ such that the distance $d_G(u, v) \leq 3$.
When a vertex $u \in V(G)$ is selected whereupon a set X which complies with some parameter set prescriptions and $u \in X$ it is said that, X *has been built on* u. If Conjecture 2 is found to be correct the next conjecture with motivation might be settled.
Conjecture 2. Let G be a non-complete connected graph of order $n \geq 3$ and G does not have a unique minimum confluence set. There exists a minimum confluence set $\mathcal{C}$ of G which contains a vertex u with $deg_G(u) = \Delta(G)$.

Motivation: Firstly, it is possible to select a vertex u and "build" a minimal confluence set X for G. It is claimed that amongst all vertices with $deg_G(v) = \Delta(G)$ at least one such "built" X' on say v' will be a minimum confluence set because firstly, there exists a v' such that $|X'| = min\{X : \text{built on } u,\ deg_G(u) = \Delta(G)\}$ and $|N(v')| = \Delta(G)$ contributes a maximum to Y with $Y = \bigcup\limits_{w \in X'} N[w]$.
It is easy to verify the result for the one graph order 3, the five graphs of order 4, the twenty graphs of order 5 all of which are non-complete connected graphs. Assume the result holds for all non-complete connected graphs of order k. Consider any non-complete connected graph G of order $k+1$. Amongst all vertices with degree $\delta(G)$ select a vertex w with minimum $|N_2(w)|$ such that w is not a cut-vertex. Such w always exists. Consider graph $H = G - w$.

Since H is a non-complete connected graph of order k consider a minimum confluence set $\mathcal{C}_H$ which contains a vertex x with $deg_H(x) = \Delta(H)$.
Case 1. If $deg_H(x) = deg_G(x) - 1$ then w is adjacent to x in G and the result holds.
Case 2. If $deg_H(x) = deg_G(x)$ then w is not adjacent to x in G. If w is adjacent to a vertex in $N[y]$, $y \in \mathcal{C}_H$ the result holds for G. If w is adjacent to a vertex $t \in N_2[y]$, $y \in \mathcal{C}_H$ then $X' = \mathcal{C}_H \cup \{t\}$ is necessarily a minimum confluence set of G. Therefore, the results holds for G of order $k+1$. Hence, $\zeta(G) = |X'|$.
By induction the result holds for all non-complete connected graphs G be of order $n \geq 3$. $\square$ Following Theorem 21 it is important to develop and code an efficient algorithm to further research for graphs in general. Computational research is an avenue to to find substantial results for the new notion. The three novel graph operations with the results of Theorems 13, 14, 17 & 18 provide a framework to achieve an algorithmic approach.
Problem 2. Design and code an efficient algorithm to find a minimum confluence set an the confluence number of a graph.
For the construction of $\mho(L(T))$ the pseudo entities, representative vertices for cliques were utilized. Upon utilizing Heuristic A the procedure remained blind to the pseudo entities. Such representative vertices may appear in the minimum confluence set yielded by Heuristic A. Although $\zeta(L(T))$ is valid same cannot be said of the set $\mathcal{C}_{L(T)}$.
Problem 3. If possible find an improved construction of a confluence tree for the line graph of a tree which will yield both the confluence number and a valid minimum confluence set.
If some blocks of a block graph are substituted by general graphs a *blocklike* graph G is constructed. It implies that G can be decomposed into a maximum number of graphs through unmerging operations f^{-1}, $f \in \{\boxdot, \leftrightsquigarrow, \multimap\}$. Let such decomposition result in a set of minimal subgraphs denoted by $\mathcal{D}(G) = \{G_1, G_2, G_3, \ldots, G_t\}$. Let $S = (\zeta(G_1), \zeta(G_2), \zeta(G_3), \ldots, \zeta(G_t))$. Iteratively reconstructing G with the results of Theorems 13, 14 and 18 will determine $\zeta(G)$. This avenue of research remains open.
Let $\zeta(G) = k$. For any minimum confluence $\mathcal{C}_G$ let $V'(G) = V(G) \backslash \mathcal{C}_G$. Let $Z = \{Z_i : Z_j \text{ is a k-tuple subset of vertices of } V'(G)\}$.
Conjecture 3. There exist a minimum confluence set $\mathcal{C}'$ of G such that,

$$\sum_{v \in \mathcal{C}'} deg_G(v) \geq \sum_{u \in Z_i, \forall Z_i \in Z} deg_G(u).$$

In the field of algebra it is known that posets of minimal type are related to linear Jaco graphs [1, 8]. Section 5 suggests that algebraic results may follow in respect of the minimum confluence number and confluence path of linear Jaco graphs. It is deemed a worthy avenue for research.

Dedication

This paper is dedicated to the life and career of the first author's remarkable Uncle Ben Kok. Uncle Ben was born on 18 December 1930 and at the time of writing this paper, he continues to show a keen interest in the advances in mathematics, quantum physics, nuclear physics and modern day technology. Linear Jaco graphs were named after Pieter Jaco Kok, an elder brother of Uncle Ben Kok. The dedication is to celebrate the occasion of Uncle Ben's 90th birthday.

Acknowledgment

The authors would like to thank the anonymous referees for their constructive comments, which helped to improve on the elegance of this paper.

References

[1] R. Assous, M. Pouzet, Jónsson posets, *Algebra Universalis* **79,** 3 (2018) article 74. ⇒37

[2] A. Behtoei, M. Jannesari, B. Taeri, A characterization of block graphs, *Discrete Applied Mathematics* **158** (2010) 219–221. ⇒27

[3] J.A. Bondy, U.S.R. Murty, *Graph Theory with Applications,* Macmillan Press, London, 1976. ⇒21

[4] F. Harary, *Graph Theory*, Addison-Wesley, Reading MA, 1969. ⇒21

[5] J. Kok, Linear Jaco graphs: A critical review, *Journal of Informatics and Mathematical Sciences* **8,** 2 (2016) 67–103. ⇒33

[6] J. Kok, P. Fisher, B. Wilkens, M. Mabula, V. Mukungunugwa, Characteristics of finite Jaco graphs, $J_n(1), n \in \mathbb{N}$, *arXiv: 1404.0484v1 [math.CO].* ⇒33

[7] J. Kok, P. Fisher, B. Wilkens, M. Mabula, V. Mukungunugwa, Characteristics of Jaco graphs, $J_\infty(a), a \in \mathbb{N}$, *arXiv: 1404.1714v1 [math.CO].* ⇒33

[8] J. Kok, N.K. Sudev, K.P. Chithra, U. Mary, Jaco-type graphs and black energy dissipation, *Advances in Pure and Applied Mathematics* **8,** 2 (2017) 141–152. ⇒ 37

[9] J. Kok, C. Susanth, S.J. Kalayathankal, A study on linear Jaco graphs, *Journal of Informatics and Mathematical Sciences* **7,** 2 (2015) 69–80. ⇒33

[10] A.C.M. Rooij, H.S. Wilf, The interchange graph of a finite graph, *Acta Mathematica Hungarica* **16,** 3-4 (1965) 263–269. ⇒22, 23

[11] J. Shiny, J. Kok, V. Ajitha, Confluence number of graphs, Communicated. ⇒ 21, 22, 23, 32

[12] B. West, *Introduction to Graph Theory*, Prentice-Hall, Upper Saddle River, 1996. ⇒21

Received: December 18, 2020 • Revised: January 31, 2021

Acta Univ. Sapientiae, Informatica **13,** 1 (2021) 39–59

DOI: 10.2478/ausi-2021-0003

Towards autoscaling of Apache Flink jobs

Balázs VARGA
ELTE Eötvös Loránd University
Budapest, Hungary
email: balazsvarga@student.elte.hu

Márton BALASSI
Cloudera
Budapest, Hungary
email: mbalassi@cloudera.com

Attila KISS
J. Selye University
Komárno, Slovakia
email: kissae@ujs.sk

Abstract. Data stream processing has been gaining attention in the past decade. Apache Flink is an open-source distributed stream processing engine that is able to process a large amount of data in real time with low latency. Computations are distributed among a cluster of nodes. Currently, provisioning the appropriate amount of cloud resources must be done manually ahead of time. A dynamically varying workload may exceed the capacity of the cluster, or leave resources underutilized. In our paper, we describe an architecture that enables the automatic scaling of Flink jobs on Kubernetes based on custom metrics, and describe a simple scaling policy. We also measure the effects of state size and target parallelism on the duration of the scaling operation, which must be considered when designing an autoscaling policy, so that the Flink job respects a Service Level Agreement.

1 Introduction

Apache Flink [5, 18, 10] is an open-source distributed data stream processing engine and framework. It can perform computations on both bounded and

Computing Classification System 1998: C.2.4
Mathematics Subject Classification 2010: 68M14
Key words and phrases: Apache Flink, autoscaling, data stream processing, big data, kubernetes, distributed computing

unbounded data streams using various APIs offering different levels of abstraction. A Flink application consists of a streaming pipeline, which is a directed graph of operators performing computations as nodes, and the streaming of data between them as edges.

Flink applications can handle large state in a consistent manner. Most production jobs make use of stateful operators that can store internal state via various state backends, such as in-memory or on disk. Flink has an advanced checkpointing and savepointing mechanism to create consistent snapshots of the application state, which can be used to recover from failure or to restart the application with an existing state [4, 3].

These streaming jobs are typically long-running, their usage may span weeks or months. In these cases, the workload may change over time. The application must handle the changed demands while meeting the originally set service level agreement (SLA). This changing demand may be predictable ahead of time, in case some periodicity is known, or there are events that are known to influence the workload, but in other cases, it is bursty and unpredictable. Statically provisioning resources and setting the job's parallelism at launch-time is unsuited for these long-running jobs. If too few resources are allocated (under-provisioning), the application will not keep up with the increasing workload, and start missing SLAs. If the resources are provisioned to match the predicted maximum load, the system will run over-provisioned most of the time, not utilizing the resources efficiently, and incurring unnecessary cloud costs.

Flink jobs' parallelism can not be changed during runtime. It is possible however to take a savepoint, then restart the job with a different parallelism from the snapshot. If the job is running in the cloud, it is also possible at this point to provision (or unprovision) additional resources, new instances that can perform computations. This is called horizontal scaling.

Restarting a job is an expensive operation. The state must be written to a persistent storage beforehand, which can be done asynchronously, but restoring from this savepoint after the restart can take a considerable amount of time. Meanwhile, the incoming workload is not being processed, so the restarted application has to catch up with this additional delay. Scaling decisions should therefore be made wisely. We must monitor various metrics of the running job, take into account the delays allowed by the SLA, and decide whether the trade-off of the scaling is worth it. Algorithms that make this decision automatically, reacting to the changing load dynamically, and performing the actual scaling operation are of great value, and make the operations of long-running streaming applications feasible and efficient.

Container orchestrators such as Kubernetes [9] allow us to both automate the mechanics of the scaling process, and to implement the custom algorithms that make the decisions. We have set up a system that can perform these scaling operations using Kubernetes' Horizontal Pod Autoscaler resource [20] and Google's open-source Flink operator [21].

In this paper, we discuss the architecture of this system. We describe a simple scaling policy that we have implemented, that is based on operator idleness and changes of the input records' lag. Additionally, we analyze the downtime caused by the scaling operation and how it is influenced by the size of the application state. We make observations regarding these results, that should be considered when designing an autoscaling policy to best meet a given SLA while minimizing overprovisioning.

2 Related work

Cloud computing is a relatively new field, but in the recent years it has gained a large interest among researchers.

The automatic scaling of distributed streaming applications consists of the following phases [13]: a monitoring system provides measurements about the current state of the system, these metrics are analyzed and processed, which is then applied to a policy to make a scaling decision (plan). Finally, the decision is executed, a mechanism performs the scaling operation. Most research is focused on the analytic and planning phase.

The authors of [13] have reviewed a large body of research regarding autoscaling techniques. They categorize the techniques into five categories: (1) threshold-based rules, (2) reinforcement learning, (3) queuing theory, (4) control theory, and (5) time series analysis based approaches.

The DS2 controller [11] uses a lightweight instrumentation to monitor streaming applications at the operator level, specifically the proportion of time each operator instance spends doing useful computations. It works online and in a reactive manner. It computes the optimal parallelism of each operator by a single traversal of the job graph. The authors have implemented instrumentation for Flink among other streaming systems. They have performed experiments on various queries of the Nexmark benchmarking suite to show that DS2 satisfies the SASO properties [1]: stability, accuracy, short settling time, and no overshoot. The job converges to the optimal parallelism in at most 3 steps (scalings). The resulting configuration exhibits no backpressure, and provisions the minimum necessary resources.

PASCAL [12] is a proactive autoscaling architecture for distributed streaming applications and datastores. It consists of a profiling and an autoscaling phase. In the profiling phase, a workload model and a performance model are built using machine learning techniques. These models are used at runtime by the autoscaler to predict the input rate and estimate future performance metrics, calculate a minimum configuration, and to trigger scaling if the current configuration is different from the calculated target. For streaming applications, these models are used to estimate the CPU usage of each operator instance for a predicted future workload. The authors show that their proactive scaling model can outperform reactive approaches and is able to successfully reduce overprovisioning for variable workloads. In our work, we use a different metric from the CPU load, based on how much the job lags behind the input. Our policy is a reactive approach, but it might be interesting to explore whether a proactive model could be built on these metrics.

Ghanbari et. al. [8] investigate cost-optimal autoscaling of applications that run in the cloud, on an IaaS (infrastructure as a service) platform. They propose an approach that uses a stochastic model predictive control (MPC) technique. They create a model of cloud and application dynamics. The authors define a cost function that incorporates both cloud usage costs, as well as the expected value of the cost or penalty associated with the deviation from certain service level objectives (SLOs). These SLOs are based on metrics that describe the overall performance of the application.

In our work, we aim to describe the characteristics of scaling Flink jobs, to serve as a base for an optimal scaling policy in the future. We also provide an architecture for making and executing the scaling decisions.

3 System architecture

We have implemented an autoscaling architecture on Kubernetes. This section gives an overview of the components involved in running, monitoring and scaling the applications.

3.1 Kubernetes operator

Flink applications can be executed in different ways. Flink offers per-job, session and application execution. Flink supports various deployment targets, such as standalone, Yarn, Mesos, Docker and Kubernetes based solutions. There are various managed or fully hosted solutions available by different vendors.

There are also multiple approaches of running Flink on Kubernetes. One method is to deploy a standalone cluster on top of Kubernetes. In this case, Kubernetes only provides the underlying resources, which the Flink application has no knowledge about. Flink also supports native Kubernetes deployments, where the Flink client knows about and interacts with the Kubernetes API server.

We have decided to use the standalone mode combined with Kubernetes' operator pattern [6] to manage Flink resources. The open-source operator [21] by Google defines Flink clusters as custom resources, allowing native management through the Kubernetes API and seamless integration with other resources and the metrics pipelines. The operator encodes Flink-specific knowledge and logic in its controller.

The desired state of the cluster is specified in a declarative manner, conforming to the format defined in the custom resource definition (CRD). The user submits this specification to the Kubernetes API server, which creates the *FlinkCluster* resource. The operator, installed as a deployment, starts to track the resource. The reconciliation loop performs four steps.

1. It observes the resource, and its sub-resources, such as *JobManager* or *TaskManager* deployments, ingresses, etc.

2. The controller uses the observed information to compute and update the *Status* fields of the resource through the API.

3. The desired state of the individual cluster components is calculated, based on the (potentially changed) observed specification, and the observed status.

4. Finally, the desired component specifications are applied through the API.

This loop runs every few seconds for every *FlinkCluster* resource in the Kubernetes cluster.

3.2 Scale subresource

We have modified the operator to expose the *scale* subresource on the *FlinkCluster* custom resource. This exposes an endpoint that returns the current status of the scaling, which corresponds to the number of *TaskManager* replicas and the job parallelism, as well as a selector, which can be used to identify the

Pods that belong to the given cluster. Additionally, this endpoint can be used to set the desired number of replicas in the *FlinkCluster* Spec.

The scaling process starts with this step, the desired replicas are set through the scale subresource. The scaling is done in multiple steps, with intermediate desired deployments in the process. The operator can keep track of the cluster's and the job's state, and perform the steps of a scaling operation. As the scale subresource's *replicas* specification changes, the operator first requests a savepoint and the deletion of the existing cluster. Once this is complete, it computes the desired deployment (step 3 of the reconciliation loop) with the newly desired replicas. When the cluster components are ready, it resubmits the job with the appropriate parallelism, starting from the latest savepoint.

3.3 Custom metrics pipeline

The scaling decisions are based on metrics from the Flink job. We have used Prometheus [2] to scrape the job's metrics. Flink has an established metrics system, including access to connector metrics (such as Kafka). We have used Flink's Prometheus reporter to expose the metrics to Prometheus.

To access Prometheus metrics through the Kubernetes metrics API, we have used an adapter [16]. It finds the desired time series metrics in Prometheus, connects them to the appropriate Kubernetes resources, and performs aggregations, exposing the results as queryable endpoints in the custom metrics API. The metrics we have decided to calculate will be described in detail in Section 4. Figure 1 shows the overview of the metrics pipeline.

3.4 Horizontal Pod Autoscaler

The Horizontal Pod Autoscaler (HPA) [20] is a built-in Kubernetes resource. It can control the scaling of Pods in a replication controller, such as ReplicaSet or Deployment. It can also scale custom resources whose Scale subresource is exposed and the scaling logic is implemented. Since we have done this for *FlinkCluster* resources, we can set them as scaling targets for the HPA.

The Horizontal Pod Autoscaler uses the currently observed replica count (either calculated by counting pods with certain labels, or read from the scale endpoint), as well as various types of metrics to calculate the desired number of replicas, by the following formula:

$$\mathsf{desiredReplicas} = \max_{i \in \mathsf{usedMetrics}} \left\lceil \mathsf{currentReplicas} \times \frac{\mathsf{currentMetricValue}_i}{\mathsf{desiredMetricValue}_i} \right\rceil \tag{1}$$

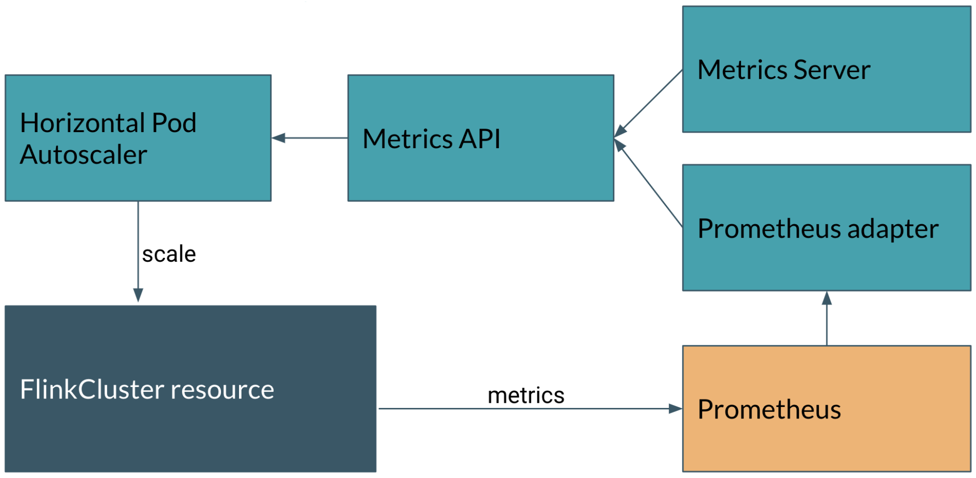


Figure 1: The custom metrics pipeline with Prometheus, used in our scaling architecture.

We have used the latest, *autoscaling/v2beta2* version of HPA, which has support for both resource, custom, and external metrics. We can specify multiple metrics, including those we have exposed through Prometheus and the custom metrics API. The HPA calculates the desired replica count based on each metric, and uses the largest value to scale the resource. In our setup, we have used two metrics through the custom metrics pipeline described above. The metrics will be described in Section 4.

The above equation assumes that the application can handle workload proportionally to the number of replicas, and that the metrics linearly change with the utilization, and therefore inversely with the number of replicas. This means that if the current metric is n times the desired metric, the desired replicas are calculated to be n times the current replicas. After the scaling, we expect the current metric value to decrease to near the desired metric value.

4 Scaling policy

We have implemented a scaling policy inspired by [15, 7]. We assume that the stateful Flink job reads a stream of data from a Kafka topic, performs some calculations on the records, and emits the results to another Kafka topic. Apache Kafka [14] is an open-source event streaming, message broker platform. Producers can write and consumers can read topics that are stored in a distributed manner among Kafka brokers. The data in the topics is broken up to partitions.

The data is assumed to be evenly distributed, i.e. there is no skew among the number of messages in the partitions, so the parallel operator instances are under a nearly equal load.

The scaling policy uses two different metrics. One is based on the relative changes in the lag of the input records, and the other on the proportion of time that the tasks spend performing useful computations.

4.1 Relative lag changes

The goal of autoscaling is for the Flink job to be able to handle the incoming workload. In distributed systems, failures are inevitable and a part of normal operation, so the processing capacity of the job should slightly exceed the rate of incoming records. This way, in the case of a failure, the job can catch up after a recovery. However, the job should not be overprovisioned by a large factor, to avoid wasting resources.

Kafka keeps track of the committed offsets in each partition (number of records that have been read by Flink), as well as the latest offset (the number of records). The difference between these is called the consumer *lag*. Flink uses a separate mechanism from Kafka's offsets to keep track of how much it has consumed, due to the way it handles checkpoints and consistency, but it is still possible to extract information about how far the consumption lags behind.

Flink's operators are deployed to tasks as multiple parallel instances. Each task of the Kafka source operator is assigned a number of partitions of the input topic. A total lag or an average lag metric would be more useful, but it is not available with this setup. Therefore, we give an upper bound for the total lag using the *records_lag_max* metric, which returns for instance the maximum of the lags in the partitions that they read. For example, consider a topic with 1 million messages evenly balanced among 20 partitions, when read from the beginning by a Flink job with 4 consumer instances. Each instance would get assigned 5 partitions, and when reading from the beginning, the lag for each partition would be 50000, hence the *records_lag_max* would also be 50000 for each instance. As the job consumes the messages (faster than they are produced), the lag in each partition would decrease, and this metric would give the largest among them for each task instance. We give an overestimation for the total lag by multiplying this metric for each instance with the corresponding *assigned_partitions* metric, and summing this value for all tasks with the source operator's instances. Equation 2 summarizes this calculation.

$$\mathsf{totalLag} = \sum_{i \in \mathsf{SourceTasks}} \mathsf{records\_lag\_max}_i \times \mathsf{assigned\_partitions}_i \qquad (2)$$

If this lag is nearly constant, the job is keeping up with the workload. If the lag is increasing, the job is underprovisioned and needs to scale up. If the lag is decreasing, the job may be overprovisioned. However, this may be a desired scenario if there was a large lag, as the latency will decrease if the job can process the lagging records.

To decide whether the current parallelism is enough to handle the workload, we take the rate of change of the lag metric (in records / second), using PromQL's (Prometheus' query language) [2] *deriv* function over a period of 1 minute.

The specific value of lag by itself is not too meaningful, and neither is its rate of change. To be useful, we compare it to the rate at which the job processes records (in records/second), which can be calculated by summing the *records_consumed_rate* metric for each operator instance of the Kafka source, which we call *totalRate*. To smooth the effect of temporary spikes, we use a 1-minute rolling average of this metric.

The ratio of these two metrics gives a dimensionless value, which represents how much the workload is increasing (positive) or decreasing (negative) relative to the current processing capabilities. After adding 1, the resulting number is the multiplier necessary for the number of replicas to keep up with the load.

$$\mathsf{relativeLagChangeRate} = 1 + \frac{\mathsf{deriv(totalLag)}}{\mathsf{totalRate}} \qquad (3)$$

This number is only meaningful for scale up decisions, since a decreasing lag might still warrant the current number of replicas, until the application catches up to the latest records. By using *relativeLagChangeRate* as the *currentMetricValue*, and 1 as the *desiredMetricValue* for the Horizontal Pod Autoscaler, as described in Equation 1, the desired replicas are properly calculated when scaling up. For downscaling, multiplying the parallelism with this value would still correctly make the job match the incoming message rate, but a downscaling is likely not warranted until the lag is below a certain threshold. The utilization portion of the policy, described in the next subsection, is responsible for making downscaling decisions.

To make the job catch up to the lag after the scaling, and to account for future failures, it might be worthwhile to slightly overshoot when scaling up.

This could be achieved by setting the desired metric value to a slightly smaller number or including a multiplier in the fraction in Equation 3.

When the job can process messages at the rate they are generated, or if the job is overprovisioned with no lag, then the change of the lag is 0, giving the *relativeLagChangeRate* metric a value of 1. These two cases need to be distinguished, as in the latter case scaling down might be possible. Since the HPA sets the parallelism to the maximum dictated by various metrics, a value of 1 for *relativeLagChangeRate* in the case of 0 lag would prevent a downscaling based on other metrics (whose values would be between 0 and 1).

To avoid this, we need some logic not to use this metric when the total lag is below a desired threshold. To simulate this, we can use the following value instead in the query for the HPA:

$$\frac{\mathsf{totalLag} - \mathsf{threshold}}{\mathsf{abs}(\mathsf{totalLag} - \mathsf{threshold})} \times \mathsf{relativeLagChangeRate} \qquad (4)$$

If the total lag is less than the threshold, then the fraction in the equation is -1, which makes the above expression negative. Then, this metric will no longer prevent the downscaling based on the other metric, since the HPA takes the maximum of the values calculated based on various metrics. If the total lag is larger than the threshold, then the fraction is 1, so this metric will be taken into account when making the scaling decision.

As noted before, this metric always overestimates, since the total lag is calculated based on the maximum lag of each subtask. If the data has a large skew, the effect of this overestimation may be very large. Therefore, it may be necessary to adjust the desired value of this metric based on knowledge about the actual data being processed. If the number of subtasks matches the number of Kafka partitions, and each subtask processes one partition, then the metric is based on the true lag value. Obviously, due to scaling, this will not be the general case, because the parallelism changes over the lifetime of a job. In the future, it would be desirable to expose a metric about the lag in each of the partitions.

4.2 Utilization

As noted in the previous subsection, another rule is necessary for the policy to distinguish between cases when the lag remains unchanged because the job's processing capabilities match the incoming rate, and when the job could process more than the incoming rate, but there are no records to be processed.

This rule is based on the *idleTimeMsPerSecond* metric. It is a task-level built-in metric in Flink, that represents the time in milliseconds that a task is idle. A task can be idle either because it has no data to process, or because it is backpressured (it is waiting for downstream operators to process their data).

If a task is idle due to a lack of incoming records, then the job is overprovisioned for the current load. We define the utilization metric for a job as the average portion of time its tasks spend in a non-idle state.

In our observations, the tasks executing the source operators (Kafka consumers) have shown an *idleTimeMsPerSecond* metric of 0 ms when there was no lag and the job was able to keep up with the messages, and 1000ms when the job was processing events at its maximum capacity, due to the source tasks being backpressured by the downstream processor operators. This is not an accurate representation of the overall utilization of the job, therefore we have excluded the tasks containing the first operator from the calculation. For this metric to be useful, the source operators should not be chained to others.

The metric can be expressed with the following formula:

$$\mathsf{utilization} = 1 - \frac{\mathsf{avg}_{\mathsf{nonSourceTasks}}(\mathsf{idleTimeMsPerSecond})}{1000} \tag{5}$$

This is a dimensionless number between 0 and 1, representing the average utilization of tasks in the job. We use this as the *currentMetricValue* in the Horizontal Pod Autoscaler, and set the desired value to a number less than 1.

If the utilization is above the desired value, it may trigger a scale up before the lag metric. Its main purpose however, is to scale down when the utilization is low. If this is the case, the lag metric is 0, so the job would be able to process more records than the incoming rate. For example, if the target utilization is 0.8, and its current value is 0.4, then a scale down should be triggered, the new parallelism should be half of the original.

This simple metric assumes that the job's tasks are under even load. A finer-grained approach could be used with more knowledge about a job's specifics, such as which operators perform heavy calculations, and consider tasks with different weights in the average.

5 The effects of scaling on performance

The scaling operation requires the snapshotting of the whole application state onto persistent storage. Additionally, in our Kubernetes operator implementation, the original job's pods are destroyed, and new pods are provisioned instead with the appropriate replica count. The Flink job then has to be restarted from the recently snapshotted state.

In this section, we analyze how the size of the state affects the downtime during the scaling operation. We perform measurements regarding the savepoint duration, the overall downtime, and the loading time of the savepoint after the scaling.

5.1 Experimental setup

We have created a simple Flink job that reads records from a topic in Apache Kafka [14], keys them by one of the fields, performs a simple calculation in a *KeyedProcessFunction*, and writes the results to another topic. The job stores a random string of configurable size in a keyed state. The number of keys is also configurable.

A separate data generator job produces the records of the input topic. The creation timestamp is stored as a field of each record. The main job's last operator before the Kafka producer calculates the difference between its own processing time and the stored timestamp, and writes this elapsed time to the emitted record. This value can be used to determine the total downtime in Section 5.3. We disregard the differences among the nodes' clocks.

We have used the operator's savepoint mechanism, as well as the scale endpoint to trigger savepoints and restarts. We have observed the duration of savepointing on the JobManager dashboard. To calculate the downtime and latency distribution, we have observed the output Kafka topic's records with the elapsed time field.

5.2 Effects of state size on savepoint duration

Flink has an aligned checkpoint mechanism [3]. When triggering a snapshot, a snapshot barrier is inserted at each input operator, which flows along the streaming pipeline along with the records. An operator takes a snapshot of its state when it has received the snapshot barriers from all of its inputs. When operators are under different loads, this can significantly delay the snapshotting of certain operators' states.

The latest versions of Flink also support unaligned checkpoints [19] (but not unaligned savepoints), where checkpoint barriers can overtake buffered records, and thus avoid this delay caused by the alignment. This does not affect the time required for the I/O operations involving the checkpoint. In future

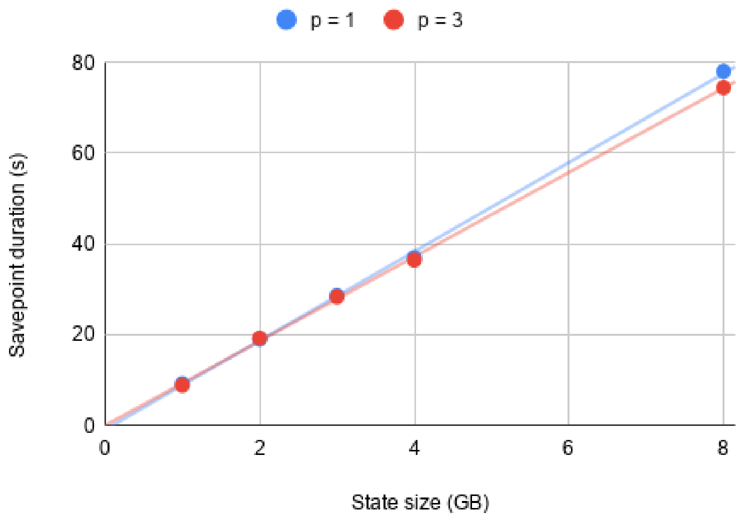


Figure 2: Duration of taking a savepoint as measured by Flink, with relation to state size, when the job runs with a parallelism of 1 or 3.

research, it might be worthwhile to investigate how using aligned or unaligned checkpoints instead of savepoints might affect performance and scaling time.

Each operator must serialize its state and write it to a persistent storage. We have measured how the size of application state affects this portion of the snapshotting process. To avoid the fluctuation caused by the barrier mechanism, we have stopped the data generator, and thus the stream of incoming records, before taking snapshots. At this point, the keyed application state had the sizes of {1.0, 2.0, 3.0, 4.0, 8.0} GB. The application has run with a parallelism of 1 and 3. We have used Amazon's Elastic File System (EFS) to store the snapshots. This is one of the storage options that can be used for Kubernetes, but there are other implementations and providers that may offer different performance characteristics. In the future, it might be worthwhile to compare the offerings of various providers.

We have manually triggered savepoints 5 times for each combination of parallelism and state size. The measurements, read from the Flink dashboard, were rounded to seconds, and an average value was taken. As seen on Figure 2, state size has a strong linear correlation to the serialization and writing time of the snapshotting phase.

The parallelism of the job did not affect the savepoint duration. It is limited by the write rate of the file system. EFS offers 100 MiB/s (105 MB/s) bursting performance. In our measurements, we have confirmed that savepoints were written at this rate.

This has been measured when the application is in a healthy state and not backpressured. The job did not need so spend time synchronizing the checkpoint barriers, as the pipeline was simple, with no multi-input operators. Backpressure does significantly increase the time to the completion of a savepoint, since the snapshot barriers must flow through all operators of the job. The beginning of writing an operator's state may thus be delayed until the barrier reaches it. The magnitude of this effect depends highly on the specific job.

The snapshotting of an operator's state happens asynchronously, with a small impact on the overall performance of the pipeline. While this effect itself may cause delays in processing and slow the pipeline, it is unlikely to cause problems in real-world scenarios due to its small magnitude, so we have decided not to focus on this effect.

5.3 Effects of state size on scaling downtime

The majority of the downtime during a scaling operation is due to the loading of state from the persistent storage. The properly partitioned state must be distributed to the operator instances from the persistent storage.

We have measured how the total downtime of the processing job is affected by the size of the state. We have manually triggered trigger scale-up and scale-down operations with different state sizes. To measure the downtime, we have used the method described in Subsection 5.1.

With state sizes of approximately 1, 2, 4 and 8 GB, we have found that the creation of the infrastructure done by Kubernetes, the deletion and initialization of pods is a factor with a large variance, that is responsible for the larger portion of the scaling time. Additionally, if the Flink image is not available on the node, or the image pull policy is set to *Always*, its pulling is another factor that we have little control over, and might dominate the scaling time.

In an extreme case, the cluster has no available resources to provision the requested number of pods, and the job is in a stopped state until resources become available, or a new node is initialized by some autoscaling mechanism. To make measurements feasible, we limit the experiments to cases when enough resources are available on the cluster, and new pods can be provisioned without delay.

However, even in this limited case, our observations have shown a large variance in the scaling times due to the above described factor of waiting for Kubernetes. Figure 3 shows the downtimes we have measured for target parallelisms of 2 to 8, with state sizes between 2 and 8 GB. Based on the observed data, we were not able to establish the effect of either the state size or the target parallelism on the scaling time.

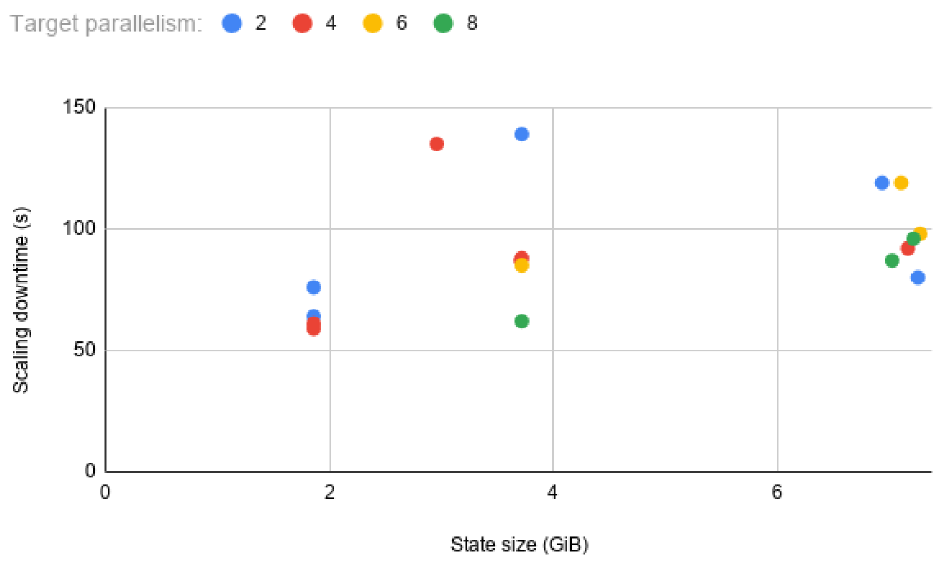


Figure 3: Total time for the scaling operation, during which the job processes no messages.

5.4 Effects of state size on savepoint restoration duration

To directly observe the effects of state size, while eliminating the large variance described in the previous section, we have taken a different approach. We have added a measurement directly to the *OperatorChain* class' *initializeStateAndOpenOperators* method. This exposes the duration of initializing each operator's state. This can either be a newly created state, or one restored from a savepoint. In our setup, only one operator had stored state, whose size we have had control over. We have exposed the load time as a gauge type metric through Prometheus, and filtered it by the operator name in a query.

We have performed measurements with state sizes up to 8 GB, and target parallelisms of 2, 4, 6, and 8. In each measurement, we have recorded the state initialization time of each stateful parallel operator instance. The operator instances work in a parallel manner, so the job is able to start processing at the full rate after the last subtask has loaded its state. We have measured the averages and the maximums of the subtask load times.

We have hypothesized the correlation between state size and load times to be linear. This was confirmed by performing a linear regression on the data points of the maximum and average values, while disregarding the target parallelism component. The R^2 values of 0.905 and 0.833 respectively indicate that state size explains a large portion of the load times' variability. Figure 4 shows the data points and the trendlines for linear regression.

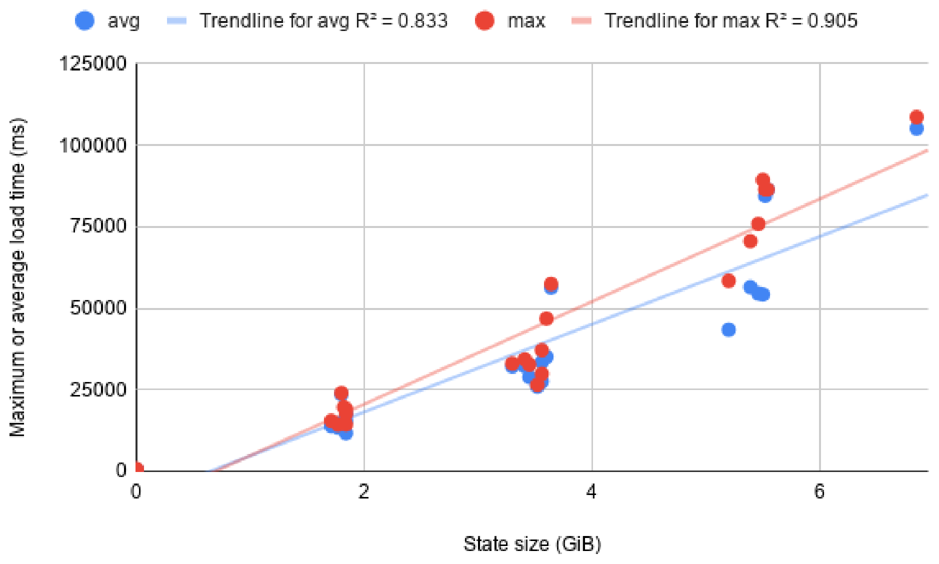


Figure 4: Maximum and average state initialization times of all instances of the stateful operator in job restarts with various state sizes.

5.5 Effects of target parallelism

Each parallel operator instance loads only a portion of the state, partitioned by some key. With a higher parallelism, the state that each operator must load is smaller, which may result in a quicker load time. However, there may be other bottlenecks (e.g. disk or network performance), that may limit the rate of loading the state.

Flink uses key groups as the smallest unit of distributing keyed state. Each key group gets assigned to a parallel instance of an operator. The number of key groups is equal to the maximum parallelism, which can only be set when the job is first started. If key groups are not distributed evenly, the state size is uneven between operator instances. It is recommended [17] to use parallelisms that are divisors of the maximum parallelism setting to avoid this issue. In our measurements, we have used a maximum parallelism setting of 720, which is divisible by most of the parallelism settings we have measured with (except for 32).

We have performed measurements to determine what the effect of the target parallelism is on the load time. We have used a cluster with 6 AWS *m5.2xlarge* instances, and ran the benchmark job with approximately 10 GiB state. We have initiated the scaling operation with target parallelisms of 4, 6, 8, 12, 16, 24, and 32, scaling from various parallelisms. The job was initially started with a parallelism of 8.

We have measured the state load times of each subtask using the same tooling as before. We have scaled the job to each parallelism 6 times, and taken the averages of the maximum and the average load times of each measurement.

Based on the results of the measurements, seen in Figure 5, we were not able to observe a clear correlation between the target parallelism and the loading time of the state.

6 Discussion and future work

We have worked on the problem of autoscaling Flink applications to adapt to the current workload. We have built and described a scaling architecture based on Kubernetes and its operator pattern.

We have focused on Flink jobs with inputs from Kafka, and detailed a simple scaling policy based on relative changes to the Kafka consumer lag, as well as the idle rate of the tasks in the job. The policy is implemented on a Horizontal Pod Autoscaler with custom metrics. We have described this scaling architecture in detail. It can be used to implement different policies with minimal modifications. For more complex setups, the Horizontal Pod Autoscaler may be replaced with a custom controller that implements an advanced scaling algorithm.

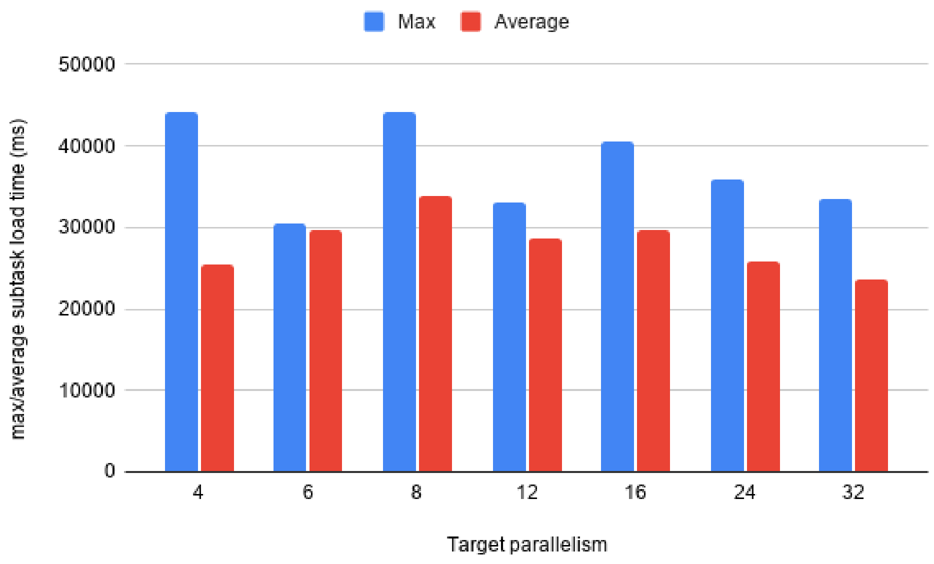


Figure 5: The average of the 6 measurements for maximum and average load times.

We have investigated the factors that affect the downtime caused by the scaling operation. Due to the steps of the scaling, the processing latencies of records may greatly increase during and after the scaling, as there is some downtime when there are no records being processed.

We have found that there is a linear correlation between the state size and the duration of savepointing to a persistent storage, excluding the time spent aligning the checkpoint barriers. While this is done asynchronously, it causes a delay between when the scaling is triggered and the beginning of the downtime.

Our measurements showed that there is a large variance in the total downtime caused by the scaling operation. We were not able to establish the effect of state size or target parallelism at this point. The time it takes for Kubernetes to reconcile the number of necessary pods, pull the images and start the Flink processes takes up a large portion of the total duration.

This means that with this setup there is a lower bound to the maximum latencies and the downtime. Therefore it is not possible to give guarantees that every record will be processed with a low latency, the SLA has to allow for occasional downtimes on the scale of a few minutes, if the application uses autoscaling.

In our measurements, we have tried to break down the factors that influence the job's restarting time. When restarting from a savepoint, portions of the state are loaded by the subtasks from the persistent storage. The duration of this loading is one of the factors that affect the total downtime. We have found that the state size is linearly correlated with this duration. We have not found a correlation between the parallelism of the restarted job and the maximum or the average load time of its subtasks.

Our described policy may serve as a basis for designing a more advanced autoscaling setup. When the policy must respect an SLA, the scaling downtime is a factor to take into consideration. A longer restart time means that the job will accrue a larger lag during the scaling operation. Thus, it might violate the SLA for a longer period. It might make sense to overprovision by a larger factor to account for this and make the job catch up quicker.

In future work, it may be worthwhile to investigate how to incorporate our results about the effects of state size to the policy. If we can calculate the scaling downtime based on state size, it can be approximated how much additional lag the restart will cause, and how much the total lag will be. Based on this, we may calculate an appropriate scaling factor that allows the job to catch up (to decrease the lag below a threshold) within the time allowed by the SLA.

With a good autoscaling architecture and policy, long-running Flink jobs will be able to keep up with changing workloads while utilizing resources effectively. Our contributions serve as a basis towards building such an architecture and designing an optimal policy.

Acknowledgements

The project has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).

References

[1] T. Abdelzaher, Y. Diao, J.L. Hellerstein, C. Lu, X, Zhu, Introduction to control theory and its application to computing systems, in: *Performance Modeling and Engineering*, Springer US, Boston, MA, 2008, 185–215. ⇒41

[2] B. Brazil, Prometheus: Up & Running : Infrastructure and Application Performance Monitoring, *O'Reilly Media, Inc.*, 2018 ⇒44, 47

[3] P. Carbone, G. Fóra, S. Ewen, S. Haridi, K. Tzoumas, Lightweight asynchronous snapshots for distributed dataflows, *arXiv preprint 1506.08603*, 2015 ⇒40, 50

[4] P. Carbone, S. Ewen, G. Fóra, S. Haridi, S. Richter, K. Tzoumas, State management in Apache Flink: Consistent stateful distributed stream processing, in *Proc. VLDB Endow.* **10,** 12 (2017) 1718–1729 ⇒40

[5] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, K. Tzoumas, Apache flink: Stream and batch processing in a single engine, *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* **38,** 4 (2015) 28–38 ⇒ 39

[6] J. Dobies, J. Wood, Kubernetes Operators: Automating the Container Orchestration Platform, *O'Reilly Media, Inc.*, 2020 ⇒43

[7] Fabian Paul, Autoscaling Apache Flink with Ververica Platform Autopilot, *Ververica Blog*, 2021, `https://www.ververica.com/blog/autoscaling-apache-flink-with-ververica-platform-autopilot` ⇒ 45

[8] H. Ghanbari, B. Simmons, M. Litoiu, C. Barna, G. Iszlai, Optimal autoscaling in a IaaS cloud, in: *Proceedings of the 9th International Conference on Autonomic Computing*, ICAC '12, Association for Computing Machinery, New York, NY, USA, 2012, 173–178 ⇒42

[9] K. Hightower, B. Burns, J, Beda, Kubernetes: Up and Running Dive into the Future of Infrastructure, *O'Reilly Media, Inc.*, 1st edn., 2017 ⇒41

[10] F. Hueske, V. Kalavri, Stream Processing with Apache Flink: Fundamentals, Implementation, and Operation of Streaming Applications. *O'Reilly Media, Inc.*, 2019 ⇒39

[11] V. Kalavri, J. Liagouris, M. Hoffmann, D. Dimitrova, M. Forshaw, T. Roscoe, Three steps is all you need, Fast, accurate, automatic scaling decisions for distributed streaming dataflows, in: *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, USENIX Association, USA, 2018, 783–798 ⇒41

[12] F. Lombardi, A. Muti, L. Aniello, R. Baldoni, S. Bonomi, L. Querzoni, Pascal: An architecture for proactive auto-scaling of distributed services. *Future Generation Computer Systems* **98** (2019), 342–361 ⇒42

[13] T. Lorido-Botrán, J. Miguel-Alonso, J. Lozano, A review of auto-scaling techniques for elastic applications in cloud environments, *Journal of Grid Computing* **12** (2014) 559–592 ⇒41

[14] N. Narkhede, G. Shapira, T. Palino, Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale. *O'Reilly Media, Inc.*, 2017 ⇒45, 50

[15] F. Paul, Towards a Flink Autopilot in Ververica platform, *Flink Forward Global (2020)*, `https://www.flink-forward.org/global-2020/conference-program#towards-a-flink-autopilot-in-ververica-platform` ⇒45

[16] S. Ross, Prometheus Adapter for Kubernetes Metrics APIs (10.04.2021), `https://github.com/DirectXMan12/k8s-prometheus-adapter` ⇒44

[17] ∗ ∗ ∗ Setting max parallelism, *Cloudera Docs / Streaming Analytics 1.2.0* (10.04.2021),
`https://docs.cloudera.com/csa/1.2.0/configuration/topics/csa-max-parallelism.html` ⇒55

[18] ∗ ∗ ∗ Apache Flink — Stateful Computations over Data Streams (10.04.2021),
`https://flink.apache.org/` ⇒39

[19] ∗ ∗ ∗ Stateful Stream Processing, *Apache Flink Documentation* (10.04.2021),
`https://ci.apache.org/projects/flink/flink-docs-release-1.12/concepts/stateful-stream-processing.html` ⇒50

[20] ∗ ∗ ∗ Horizontal Pod Autoscaler, *Kubernetes Documentation* (10.04.2021),
`https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/` ⇒41, 44

[21] ∗ ∗ ∗ Kubernetes Operator for Apache Flink (10.04.2021),
`https://github.com/GoogleCloudPlatform/flink-on-k8s-operator` ⇒41, 43

Received: March 22, 2021 • Revised: April 11, 2021

Acta Univ. Sapientiae, Informatica **13,** 1 (2021) 60–85

DOI: 10.2478/ausi-2021-0004

Animal Farm—a complex artificial life 3D framework

Attila KISS
J. Selye University
Komárno, Slovakia
email: kissae@ujs.sk

Gábor PUSZTAI
ELTE Eötvös Loránd University
Budapest, Hungary
email: afbdpk@inf.elte.hu

Abstract. The development of computer-generated ecosystem simulations are becoming more common due to the greater computational capabilities of machines. Because natural ecosystems are very complex, simplifications are required for implementation. This simulation environment offer a global view of the system and generate a lot of data to process and analyse, which are difficult or impossible to do in nature. 3D simulations, besides of the scientific advantages in experiments, can be used for presentation, educational and entertainment purposes too. In our simulated framework (Animal Farm) we have implemented a few basic animal behaviors and mechanics to observe in 3D. All animals are controlled by an individual logic model, which determines the next action of the animal, based on their needs and surrounding environment.

1 Introduction

Within the concept of ecology, we treat an ecosystem as a small part of the whole living system. The ecosystem can scale in different sizes, from a local few species environment, to highly complex continent wide coexistence.

Computing Classification System 1998: J.4
Mathematics Subject Classification 2010: 68T20
Key words and phrases: artificial life, multi-agent systems, ecosystem, animal-simulation, extinction, sustainability, unity (engine), computer-generated, 3D-simulation, competing-ecosystem, ecosystem-model

An ecosystem comprises the entirety of the living things (plants, animals) in a given territory, interacting with each other. Natural ecosystems are "balanced" structures. This means that through interactions between the individuals, ecosystems are constantly changing. With this changes, the system is constantly trying to stabilize and thus able to sustain for a long time. Every living creature has impact on the environment, like during their lifetime predators are hunting down smaller animals for food source to sustain themselves and grow their species population, while keeping prolific animal populations lower to reduce their noxiousness.

In the lifetime of an ecosystem there are many influencing factors that can change the whole image and future of the system. This factors can be abiotic (environmental impacts) like climate change, earthquakes and sea level rising; and biotic (living creatures and their interactions) like extinction of a species in the food web, new predator appearance from another neighbour ecosystem, human interference. This effects can make entire systems unstable, regardless of the complexity, size and thoroughly development by observations. Taking all aspects into account is impossible and the smallest undefined or random effects can alter the system in enormous ways.

Replicating small ecosystem environments with living beings is not an easy task. Animals' behavior can become unpredictable in a few seconds, when they are reacting to outside effects (abiotic and biotic components as well) and every individual will react differently depending on their surrounding, characteristics and logic. This changes can force species to migrate or extinct, if their natural habitat changes rapidly and they can't adopt fast enough.

Both fixed and random effects occur in the ecosystem. Fixed effects are well observed, monitored and known behaviors and characteristics of animals and the environment and most of the time simple to implement. Environmental effects can be remodelled in the system too, but hard to replicate realistically, because their strength and occurrence has to be set correctly, otherwise their impacts can be overwhelming. It is also hard to model animals' logic and their behaviors (instincts) and how they react to different situations and occurring environmental and interaction effects. Their decisions is hard to determine based on purely outside effects or pure logic. Creating an ecosystem in a simulation is really difficult, because of this many unknown (indetermined or hard to monitor or remodel) factors. To imitate and simplify unknown influential factors, the most easy way to consider and determine them as random decisions and events as well. Without calculating these complex tasks and relying only on the values of the randomness, many alternatives are being created by running same simulations multiple times. While rerunning the same simulation and getting different result may sound odd, this way we can observe many alternatives of an evolvement of our ecosystem.

A 3D environment for a simulation can be useful to monitor different events and animals real time by graphical observation as well. It can show how the animals live, like when they eat or drink, and how they react to other species, like fleeing or hunting. It's more an eye-catching experience to see the actual animals, rather than only reading the plain resulted statistics value of the populations. Also, while we can specify a lot of ways what and how to extract data form the simulations, sometimes it's hard to determine what is important to extract from a situation. We can monitor what caused the animal's death, say like it was killed by a predator, but this doesn't determines if the animal wasn't a suitable prey or just unlucky and couldn't flee away from the predator due to a terrain (mountain or water) formation. And the opposite is true, when we say an animal died from like thirst, before it's death, it could have been chased a long way by a predator, which would suggest to give the credit for the predator, but the system registered as a death of thirst, which in the plan statistics would imply a lack of water source on the terrain. Without a perfect register system, sometimes it is advised to check on the simulation's 3D environment to observe interactions and events.

2 Related work

Scientists have been trying to analyse and model real ecosystems for a century. One of the earliest, and most well-known, ecological model is the predator-prey model of Alfred J. Lotka (1925) [17] and Vito Volterra (1926) [22] called "The Lotka–Volterra equations". Volterra originally devised the model to explain fluctuations in fish and shark populations observed in the Adriatic Sea after the First World War. There are multiple alternatives to the Lotka-Volterra predator-prey model and its common prey dependent generalizations, like Richard G. Wiegert's [23] simulation model or the Arditi-Ginzburg (ratio dependent) equations [2, 3] (it's connection to Lotka-Volterra [21]) or a three species model, where two predator competing for the same prey by Reddy and Ramacharyulu [18].

In addition to mathematical models, there are also simulation models that can be used to model ecosystems, where the mathematical model is either too complex or impossible to compute. The difficulty of creating a model is, we need to include an enormous amount of factors and data (gathered or replicated), which could interact unpredictably with each other. In a simulated model, it is an always asked question, which attributes and effects should be included in the program. The best would be if we could replicate all the properties of the environment and the animals (this is what complicated mathematic models are trying to do). To reduce this complexity, simulated models typically simplify the systems they are representing, to limit the number of components. So simulation models are more widely utilized and more biologically reasonable, while analytic models are esteemed for their numerical tastefulness and elegance.

There are multiple other frameworks, which generate valuable ecosystem data, but most of them are using high abstraction in implementation (like only text-based or 2D environments and unrealistic creatures to study). In the '90s, computer simulation programs were developed, modeled on the 1984's programming game Core War [9], where two or more programs competed by switching turns to control virtual machines and get all the resources by terminating other programs. This kind of competition is similar to real-life ecosystems where the animals are rivals to each other. A simulation named Tierra [20] appeared to show this connection parallel, with the difference of evolvement, where the programs could alter themselves like an evolution approach to evolve. After a few years, another simulation called Avida [1] appeared with another addition, where "organisms" could have different execution speeds instead of a simple distributed turn switching. Many other improved and altered versions with modified rules appeared since then, like a modified model [4]. After this many variations appeared, the community decided to keep the basic approach, and an International Core War Society (ICWS) was established for the maintenance of Core War standards and running the tournaments.

Meanwhile, computer graphics improved and finally allowed to create graphic representations of the simulations. With graphics, we became closer to replicate and observe real ecosystem environments instead of just creating simulations with complex calculations showing numbers on the screens like mathematical ecosystem models. Some simulations take a 2D abstractions to simplify their model and save computer resources like Bubbleworld.Evo [19]. Bubbleworld.Evo is a multi-agent simulation where hundreds of prey (bubbles) are trying to avoid predators (triangles) to survive. The agents determined by size, energy, speed, and in their life cycle, when they reach a dedicated size or energy, they reproduce into multiple offsprings, starting their live cycles again from the start.

Creating simulations where the animals have complex logic has many cons and pros. If they are very complex, their behaviors will be hard to define and control, and also calculating this for every individual requires significant computer resources. A 3D simulation called PolyWorld [24] created trapezoid agents to search for food and mate. They needed to keep their whole population only in the hundreds to be able to run. With lower population numbers, it is hard to create any solid ecological statements.

Another 3D simulation is Framsticks [15, 10, 14, 11, 12, 13], but here, they took a very unusual approach to the animal models and evolution mechanics. The creatures contain mechanical structures ("bodies") and control systems ("brains"), which results in very unusual appearance, like stick segments with joints. Their complicated system requires a ton of physics calculations like collisions with other creatures and the environment, gravity, and mechatronics forces (friction, elastic reaction). However, if we have enough computation power, we can run different enormous, physically accurate experiments, including evolution optimizations, co-evolution, and spontaneous evolutions from gene pools.

3 Simulation environment

While scientists like to analyse data from simulations, to catch other (like younger) audiences' interest, the simulation has to be just as engaging as informative. We named our simulation framework to "Animal Farm" after George Orwell's novella as an indication of the animals' logic model, where they can behave unexpectedly (and "rebel against us" as in the novella's story).

To make our immersive 3D graphical simulation we used Unity (engine). Unity [27] is a cross-platform game engine, developed by Unity Technologies. The editor is supported on Windows, Linux, and macOS, but the engine can build applications for more than 25 different platforms. Unity integrates multiple custom rendering options with an incredible physics engine and with Mono, the open-source implementation of Microsoft's .NET libraries, with we can write scripts in the object-oriented programming language C# to create and control the running simulation. Unity is well documented and constantly upgraded and supported with many official and unofficial tutorials on the internet.

In the past, many graphical engines were used to create 3D environments, but the most popular was the Unreal engine. After the launch of Unity in 2005, many developers and projects like "Search and Rescue Game Environment" [5] moved to this new platform. The editor is a well-designed environment with a lot of features and options like drag-n-drop features and built-in windows and also custom windows, which can be easily created by ourselves to use in our project to make the work easier.

In the past, Unity had only paid options, and Unity Indie was the cheapest with limited features, but after 2009, they discontinued the Indie version and released a free Unity version with very few feature locks. Unity's policy is that Unity (Personal) is free to creators with revenue or funding (raised or self-funded) below USD $100K in one year, meaning after 2009, a large number of students, hobbyists, and Indie developer had the opportunity the start projects with Unity for free. To unlock all the features and functionalities, a Plus or Pro license is required, but year after year, many paid features have been moved into the free version, which makes Unity an amazing platform to use for smaller, Indie development and for scientific demonstration purposes. You can learn more about Unity's history in [8].

4 Our work

In our project, we are using our developed multi-agent predator-prey simulation model to create an isolated 3D environment with a few example species. From the simulations, we collect data to analyse and set up certain hypotheses. Our animals are not simply separated into 2 distinct groups (predators and prey), we created a model, which supports food chain approaches too. A medium-sized animal can be the predator of a smaller animal, but an even bigger animal could consider this medium-sized animal as prey.

In our simulation, the animals can move intelligently on a terrain with Unity's AI system, which provides a navigation mesh, by scanning the terrain and creates a walkable mesh. On this mesh, we can set a position as a destination point for the animal. The AI system will calculate the shortest path to our destination with obstacle avoidance (like hills, trees, lakes).

To provide land for our animals, we created two separate modes (called Sample and Custom scene). The Sample scene contains a premade, static terrain created with Unity's Terrain Editor and 3 lakes, many trees, and at the edges of the map a range of mountains to keep the animals on the map. The Custom scene is for self-made maps, where we can import terrain data using heightmaps (more in the designated section) and modify it in some aspects, then start the simulation on your newly created map.

Both scenes have a 'Console Prompt' feature (by pressing the TAB key) where, with specific commands, we can interact with the running simulation like, spawn and reset animals, manipulate time, get statistics of the animal.

Two camera modes are possible, the default is to fly around inside the simulation (WASD keys) to inspect the animals, and a follow mode to automatically follow an animal (by left-clicking on the animal, which you want to follow).

Above each animal, you can see primary information of the animal's thirst, hunger levels, current logical and mating states, the animal's age, age stage, and the pack's name, which the animal belongs to.

To create some of the graphical and technical parts, we used free assets from the official Unity asset website [28]. The animal models are from 2 packages, the "Voxel Animals Pack" from "Total Game Assets" [33] and "5 animated Voxel animals" from "VoxelGuy" [29]. Our vegetation models are from the "Low Poly Nature – FREE Vegetation" from "Elcanetay" [31] and the tree models "Free Trees" from "Ada_King" [30]. We have used a file browser implementation for importing heightmap images into our custom map generation "Runtime File Browser" by "Süleyman Yasir Kula" [32].

5 Featured mechanics and logical phases

5.1 Exploring mechanic:

This is the default phase mechanic, if no other mechanics' criteria are fulfilled (e.g., the animal can't see any water in view range, when thirsty), the animal will explore the area, until a needed resource or an enemy is found. If the pack mechanic is turned on, the animals will explore together, following the pack leader.

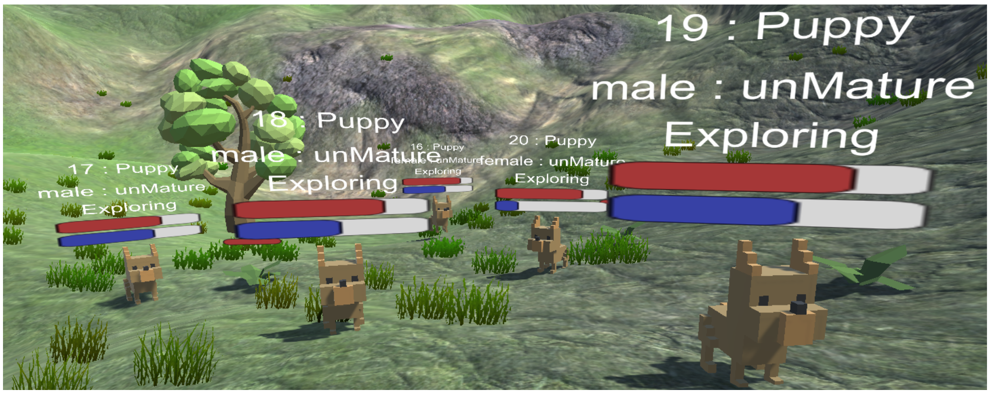


Figure 1: Exploring mechanic

5.2 Drinking mechanic:

If the animal's thirst level is below a set threshold and a designated water source found within view range, the animal goes to the source and starts to refill its thirst (lower, blue bar) meter.

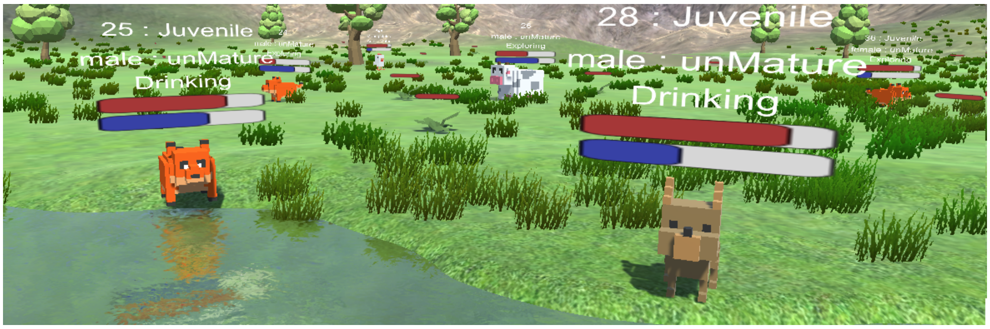


Figure 2: Drinking mechanic

5.3 Hunting mechanic:

If the animal is a predator and the animal's hunger level is below its threshold and a designated food source found within view range, the animal starts to hunt the prey by following it and if reaches close proximity, kills the prey. The prey's body becomes a corpse and the predators can consume it.

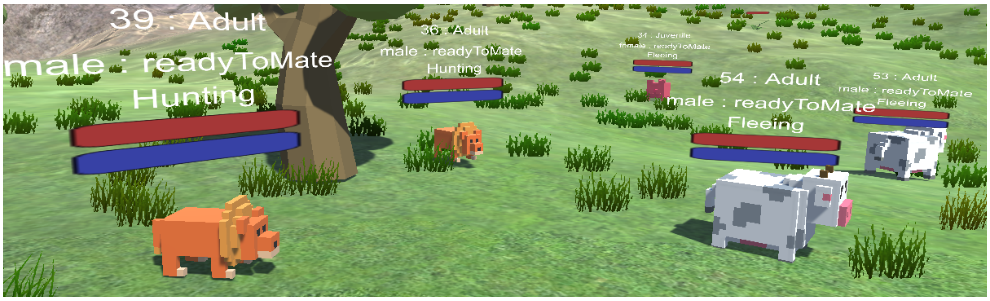
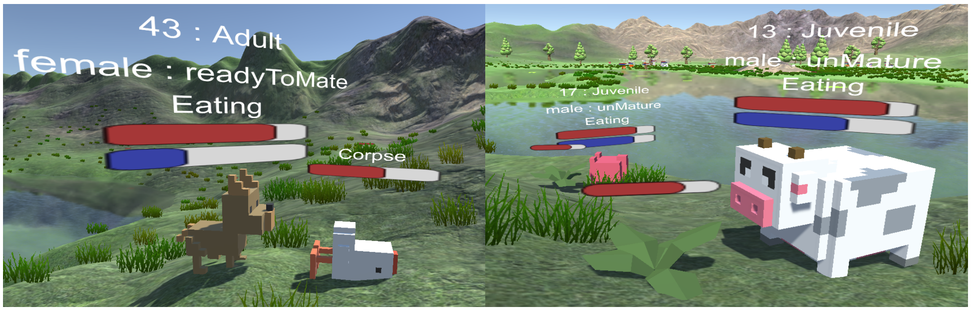


Figure 3: Hunting mechanic

5.4 Eating mechanic:

If the animal is a vegetarian or the hunting mechanic is completed, the animal goes to the food source (the hunted animal or the plant) and starts consuming it. The feeding is represented by filling up the hunger (upper, red bar) meter, and the source's meter (single, red bar) drains. Additionally, the corpse's source fades overtime, representing flesh decomposition and the vegetation's source is increases overtime as the plant's natural regeneration ability. When the source is completely depleted, it disappears from the simulation.



(a) Predator eating **(b)** Vegetarian eating

Figure 4: Eating mechanic

5.5 Fleeing mechanic:

If the animal senses a hostile animal within the view range, a fleeing movement overrides any other mechanics and the animal starts to run in the opposite direction, trying to escape from the predator. If the prey loses the predator, the natural logical cycle is restored and continued.

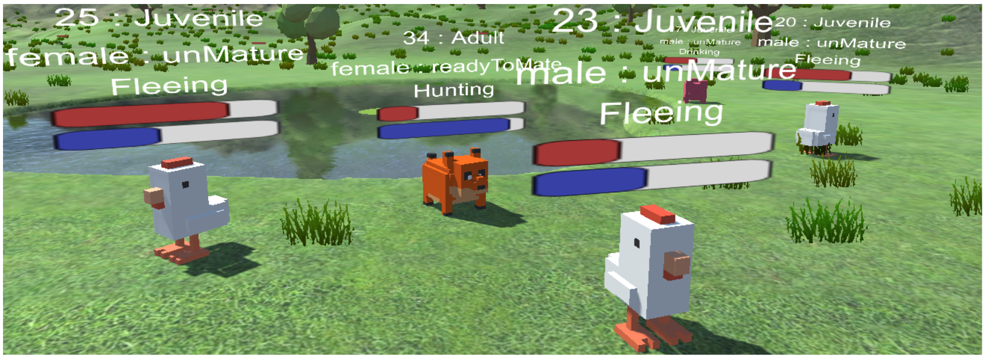
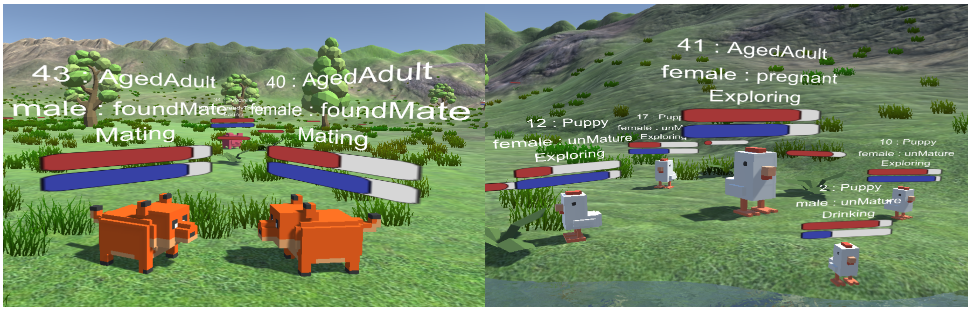


Figure 5: Fleeing mechanic

5.6 Mating mechanic:

Within the view range, if the animal finds a right (gender, mating and age status requirements are fulfilled) partner, a mating request is sent by the male to the female, and if it is accepted by the female, they will start the mating process, which will result in the female becoming pregnant. After the pregnancy time ends, new children are born (depending on the species childbirth capability).



(a) Finding a partner **(b)** Creating new life

Figure 6: Mating mechanic

5.7 Pack mechanic:

The animals can form packs with others from their own race. During the simulation, multiple packs can be formed for each race (in this simulation animals will always try to form bigger packs until a set maximum threshold number is reached), an individual can request a join (even change, which pack they belong) to an other pack, which could be accepted or rejected. Every pack has a leader, whose movement will be considered during an individual's exploring logic phase. This results in a group of animals, which moves and sticks together. This mechanic has many positive (e.g., easier to find mating partners) and negative (e.g., predators will find and catch preys easier) qualities. These assets have been analysed with statistics at the end of the article.

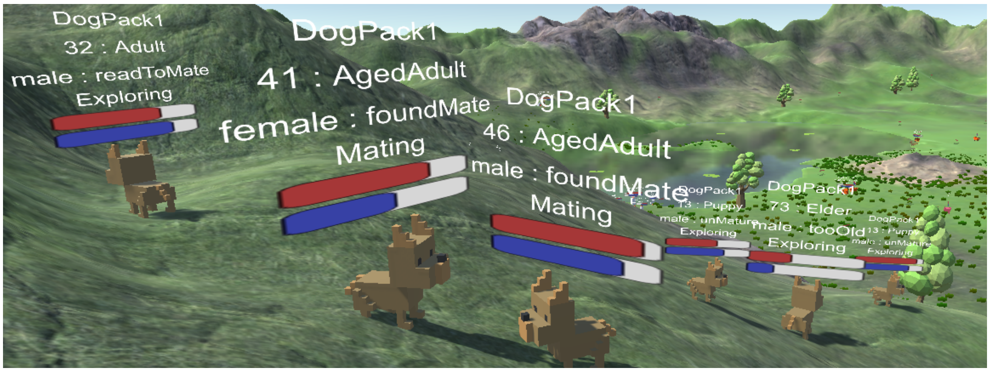


Figure 7: Pack mechanic

6 Adjustable parameters in the simulations

To modify any animal parameters, we have to alter their species' attributes. There are many attributes, which control the behaviors of the animals and they can be set for all species separately. First, we have to set a race type, so the animal knows, which animals are in the same race, to consider them as allies. Addition relations are the list of enemies, where the animals who belong here will trigger the fleeing mechanic, and the list of foods, which determines the animal's feeding type and which sources (animal or vegetation) to look for to trigger the hunting and eating mechanics.

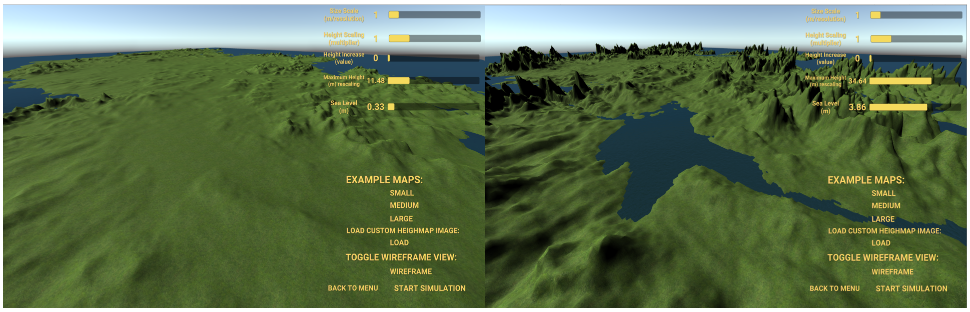
There are some parameters, which change as the animal grows. These variables can be set to animals depending on their age stages. Animals are categorised into age stages based on their current age. Our pre-determined age stages are Puppy, Juvenile, Young, Adult, Aged, and Elder. Inside every age stage, there are 4 primary variables: view distance, walking speed, hunger modifier, thirst modifier. View distance means, how far the animal can see, and from this information, the animals will determine what to do. The walking speed means how fast the animal can move on the terrain. The hunger and thirst modifiers alter the basic life requirements (metabolism), where the animals always lose hunger and thirst, while the time passes. With this, we can determine characteristics for different ages, like older animals' metabolism slows down, so their hunger and thirst bars will decrease slower than an Adult's from the same species, but their view distance and walking speed decrease as well.

The mating mechanics is determined by age stages too, where the Puppy and Elder can't reproduce, meaning they will be in a not sexually active status, but other stages can. Males and females have unique attributes here, females have pregnancy duration and max childbirth capacity, and males have spermatogenesis duration, which means the length of a state, when the male sexually inactive.

With the pack forming ability, we can set which species use this mechanics. If they are capable to form packs, the species will start to create groups and move together, if not, every animal stay an individual from others (except in the mating mechanic) and explore the map separately.

7 Custom map generation

In this simulation environment, we can choose between 2 maps to run our simulation. The first is a sample static map, which was created by hand to test the animals and illustrate a beautiful landscape, while our simulation is running, but due to its static nature of the map rerunning with the same properties multiple times, will end in slightly similar results, because of the landscape's image (lakes' and hills' position). According to solve this problem, we created a tool to import custom maps into the simulations, using heightmaps. A heightmap [6] is a black-white image file containing data from a landscape, where the pixel positions are representing the 2D surface coordinates and the pixel's grey shade will represent the height in a 0-1 scale. This allows us to convert an image file into a 3D mesh surface (terrain map), which our animals can walk on. Heightmaps include water on their representation (usually the darkest areas) but we can't be sure if that area is water or land with sea level 0, or even below sea level. In our generator section, we provided some tools to scale, increase, set a maximum height and set water level after the image is imported to modify the terrain as we wish, and see the modifications in real-time inside the environment. After we are satisfied with the map, pressing the start button will calculate any furthermore necessary technical details, and start the simulation like the sample scene, except this time with our custom-made map.


Size Scale (m/resolution) 1
Height Scaling (multiplier) 1
Height Increase (value) 0
Maximum Height (m) rescaling 11.48
Sea Level (m) 0.33

Size Scale (m/resolution) 1
Height Scaling (multiplier) 1
Height Increase (value) 0
Maximum Height (m) rescaling 34.64
Sea Level (m) 3.86

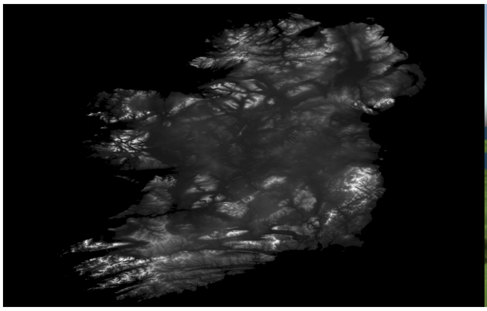
(a) Plain terrain with low sea level **(b)** High mountains with higher sea level

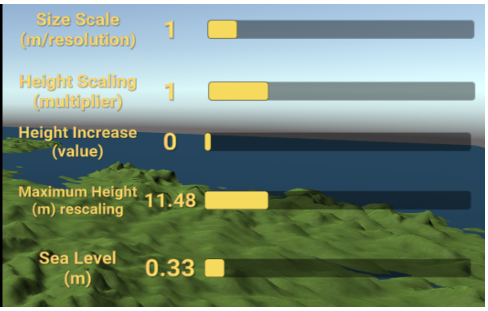
Figure 8: One of the 3 example maps provided inside the program, the same heightmap was used in both pictures, only the maximum height and the sea level was modified at runtime with the right-upper modifier sliders.

8 Heightmaps

As described in the previous section, we need an image to start a custom simulation. There are various different methods to acquire images like searching online, using any search engine, which is a great way to gather varied types and shapes to run an experiment. If we want a specific location's heightmap on Earth we can search for it by name or we can use “`https://terrain.party/`” website to create our own snapshot of a part of the Earth. Or, if we would like to run our simulation on many different (unreal) terrains we can use a noise heightmap generater like “`http://kitfox.com/projects/perlinNoiseMaker/`” to generate a random terrain to import into our environment.

Inside the map generator, we can modify a heightmap image with 5 different attributes. The “Size scale” will scale the whole image to a bigger terrain. “Height scaling” multiplies every positive height value. “Height increase” adds this fix value to every positive height value. “Maximum height” sets and translates the original heightmap’s 0-1 value into the simulation’s value system, meaning if a point is 1 (the most white color) on the heightmap, inside the simulation, it will be 11.48 (example value from [Figure 9b]) units tall. And the final attribute sets the “Sea level” to a designated value.


(a) Heightmap of the example [Figure 8]



Size Scale (m/resolution) 1
Height Scaling (multiplier) 1
Height Increase (value) 0
Maximum Height (m) rescaling 11.48
Sea Level (m) 0.33

(b) Changeable heightmap variables

Figure 9: Heightmaps

9 Demonstration of the platform

For demonstration purposes, we have created 6 prototype animals with different properties and 1 plant with a fixed 9-second reproduction rate and random spawn positions. The animals’ names are only for demonstration purposes, we named them randomly (considering some attributes as animal characteristics to strive for realistic representations of the real world animals). We named our only plant type “Grain”. We planned to create multiple types of vegetation, but decided to keep only one for this demonstration, because there are only 3 vegetarian types, and we didn’t want to complicate the experiments and decided to set them as to eat the same type of vegetation.

Parameters	Chicken	Pig	Cow	Dog	Fox	Lion
Avarage lifetime	375	700	800	650	550	700
Most childbirth	3	3	1	1	2	1
Pregnancy dur.	55	70	80	65	65	70
Spermatogenesis	50	30	30	30	30	30
Food	Grain	Grain	Grain	Chicken	Chicken,Pig	All
Enemy	Dog,Fox,Lion	Dog,Fox,Lion	Dog,Fox,Lion	Lion	Lion,Dog	None

Viewrange	Chicken	Pig	Cow	Dog	Fox	Lion
Puppy	20	22	22	23	26	23
Juvenile	22	24	24	25	28	25
Young	23	24	26	27	30	27
Adult	25	25	28	29	32	29
Aged	22	24	24	25	28	25
Elder	18	22	21	21	24	21

Walkingspeed	Chicken	Pig	Cow	Dog	Fox	Lion
Puppy	3.3	3.2	3.2	3.55	3.65	3.5
Juvenile	3.6	3.6	3.4	3.85	3.85	3.65
Young	4.1	3.9	3.6	4.05	4.15	3.85
Adult	4.3	4.1	4	4.2	4.25	4.1
Aged	3.7	3.6	3.4	3.7	3.85	3.55
Elder	3.2	3.1	2.9	3.2	3.35	3.3

Thirst/Hunger	Chicken	Pig	Cow	Dog	Fox	Lion
Puppy	1.3	1.1	1	1.1	15	1.1
Juvenile	1.55	1.35	1.35	1.3	1.35	1.22
Young	1.65	1.5	1.5	1.4	1.45	1.3
Adult	1.55	1.4	1.35	1.3	1.35	1.2
Aged	1.25	1.3	1.1	1.1	1.15	1.1
Elder	1	0.9	0.8	0.7	0.7	0.7

AgeStage bound.	Chicken	Pig	Cow	Dog	Fox	Lion
Puppy	37.5	105	120	97.5	82.5	105
Juvenile	75	154	176	143	121	154
Young	112.5	231	264	214.5	181.5	231
Adult	187.5	350	400	325	275	350
Aged	281.25	525	600	487.5	412.5	525
Elder	375	700	800	650	550	700

9.1 Sample results

We ran simulations with the same parameters and heightmap multiple times, and from the generated data we created summary graphs for each simulation. Here, we represent 5 unique situations to analyse, which were different form the major of the results.

For these simulations, we created a custom made map with balanced land-water ratio and distribution in our opinion. We placed 9 circle shaped lakes on a big square land, where the animals had plenty of space to go around and between the lakes. This way the combined water surface area was around 35-40% of the map. The water itself shaped multiple natural obstacles for the animals, so we didn't add any mountains to this terrain, to give the preys plenty of opportunities to flee in multiple directions without trapping themselves.

Our simulations start with spawning the designated number of animals (24 chicken, 24 cow, 24 pig, 18 dog, 18 fox and 12 lion) on the map with random positions. The spawned animals start as newborns on random positions, with their properties reduced to their current age stage of their species' characteristics. Because, the predators are already capable of hunting when they young, sometimes they spawn close to the preys and start to chase them in everywhere on the land, which creates a chaotic start of every simulation. From this chaos, unfortunately only a few species come out alive, usually 1-2 vegetarian and a predator type. Later the survived animals' numbers normalize and they start to create different symbiosis variations for a short time. Also, because the terrain is an isolated land from other ecosystems, there are no outside effects to our ecosystem, and if a species extinct there will be no more of them to observe in the simulation. We tried to balance the species capabilities (demonstration attributes), but this only resulted in a random selection of which animals survive the start chaos.

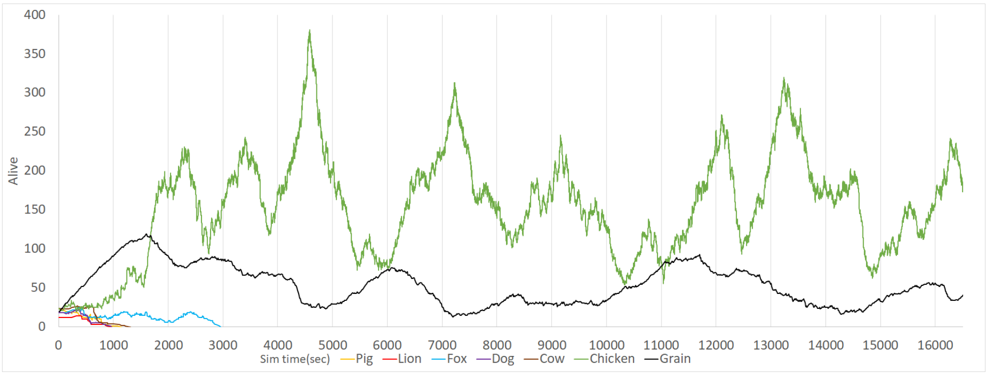


Figure 10: Symbiosis between vegetarian animals and the vegetation

[Figure 10] shows a unique case where only 1 vegetarian animal (chicken) survived in the long run of the simulation. We can observe a very balanced symbiosis between our chicken and grain species with a natural both-sided waving population increase and decrease. The chicken fluctuates in waves with the vegetation due to their rapid reproduction rate, when their numbers reach a high value they start to eat more vegetation than the vegetation's reproduction rate and the vegetation starts to decrease, which results a low food supply for the high number of chickens and they start to starve to death, lowering the chicken's number, letting the vegetation to replenish and at a point, it will be able to sustain the lower chicken population again. Without any predator the chickens' only danger is their own gluttony, meaning if the vegetation hits a very low value, maybe there won't be enough time for the vegetation to recover before the whole chicken population starve to death.

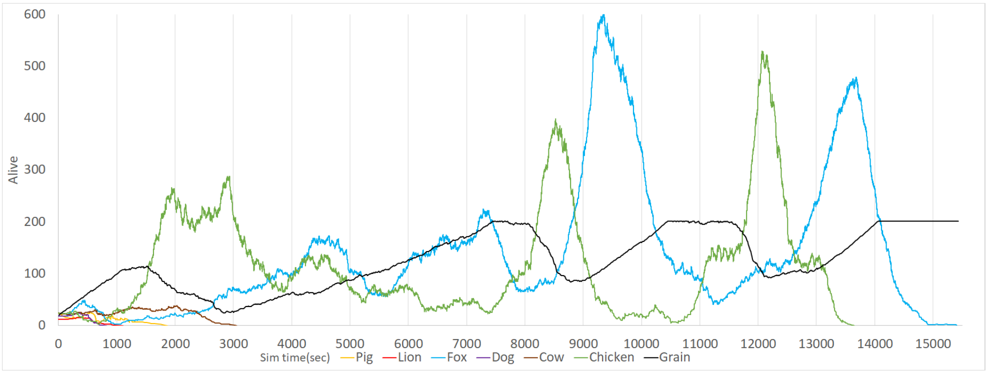


Figure 11: Symbiosis between 3 species

In this simulation [Figure 11] we can observe a more complex symbiosis between the chicken, fox, and grain. Chickens multiply rapidly, capable of a fast recovery in numbers, resulting in a grain decrease first and a fox increase secondly. The foxes overcome of the chickens' number and will decrease them significantly. If the low chicken population can survive for a brief time, the foxes number will decrease due to the food shortage, resulting in a fox population shrink and the cycle can restart. Unfortunately, this simulation ended, because after the second huge fox population wave, the chickens couldn't sustain themselves against the foxes and died out.

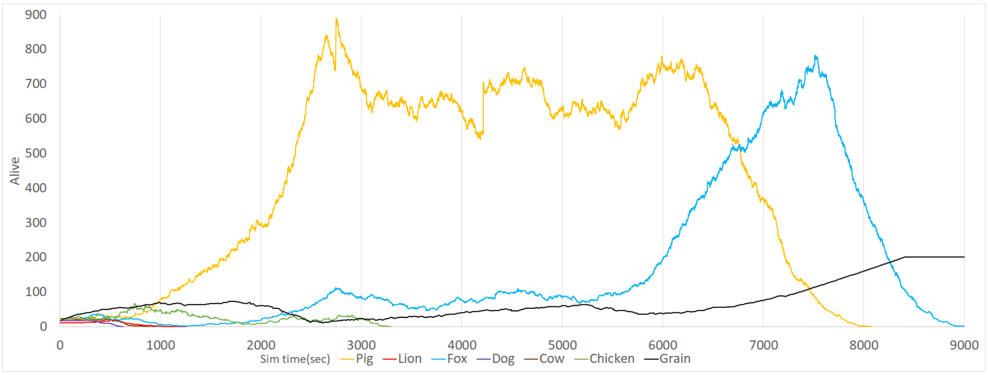


Figure 12: Pigs have eaten all the food from the chickens

Here [Figure 12] pigs overpopulated the map. Pigs have a slower reproduction rate, but because of the early low population of the foxes and the other predators' extinctions, pigs by chance could overcome of the chicken population, which extincted because of the food shortage and their low population number soon reached zero. The chickens couldn't reproduce enough, because the pregnancy and childbirth infer an extra hunger modifier, which if not enough food is available, could be fatal for the population. Foxes could barely survive, but at the early stages, pigs and chickens provided enough to reach a safe number, and after the chicken's extinction the foxes started to eat the pigs and their population started to grow rapidly since they didn't have any enemy. When the foxes population grow higher, the pigs couldn't keep up with the hunt and their numbers started to go down, which resulted in their and after a short time the foxes extinction.

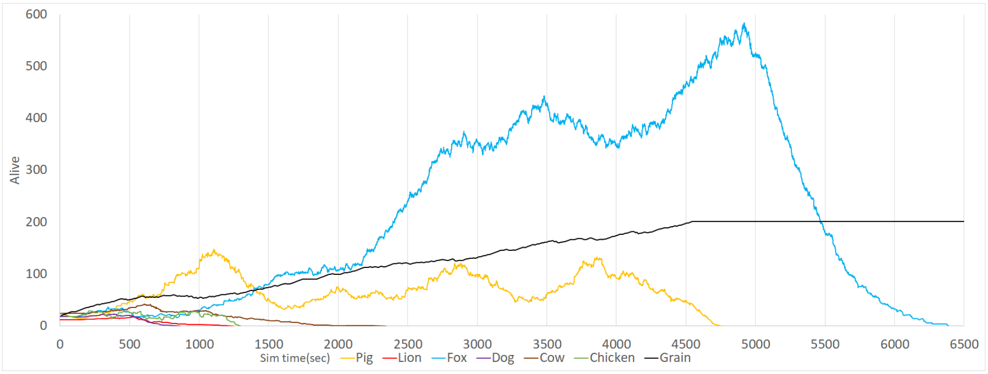


Figure 13: Unsustainable symbiosis

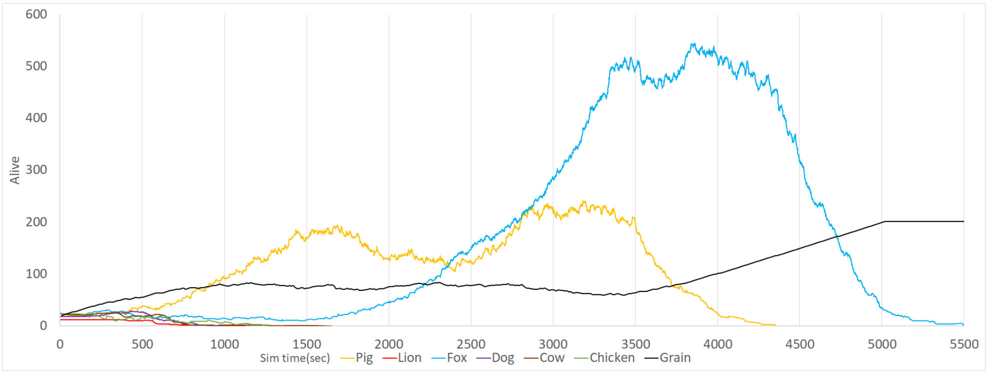


Figure 14: Unsustainable symbiosis 2

[Figure 13] and [Figure 14] is a more common result, rather than [Figure 12], because the pigs are not as prolific as the chickens and with an even slower start as before, they extincted even faster, because the foxes hunted them at the same rate, but due to the low population, the same scenario happened, just earlier. This means, that the chickens and foxes could perform better in a predator-prey symbiosis, rather than the pigs and foxes. Pigs have an advantage against the chickens with a longer lifetime attribute, but have the disadvantage in their pregnancy duration. If the predators are hunting the preys rapidly and the prey dies before reaching an older age, the extended lifetime has no real advantage, but the faster reproduction rate could mean the survival of the prey population, which the preys in the real world also use for survival.

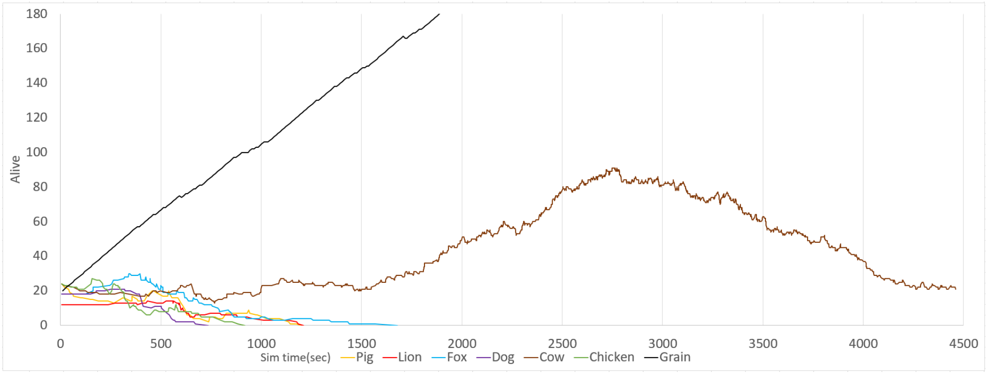


Figure 15: Bigger size is not always an advantage

In our last demonstration, the cows survived the chaos alone, like in our first demonstration [Figure 10] with the chickens and the vegetation. In the cows' characteristics we have given them a much lower reproduction and movement skills on purpose. Their only two advantage is in their size (out of our predators, only the lions can harm the cows) and their expanded lifetime. As the only survivors, without any enemy and plenty of food, they managed to survive, but unlike the chickens their numbers didn't fluctuate as much, keeping a low population despite the food abundance, due to their low reproduction rate their population kept a low value. Without predators or environmental effect this can be an acceptable solution, but if the lions were present or any catastrophe (e.g., environmental event or biological disease) occurs, while they keep a low population, they wouldn't survive for long and extinct.

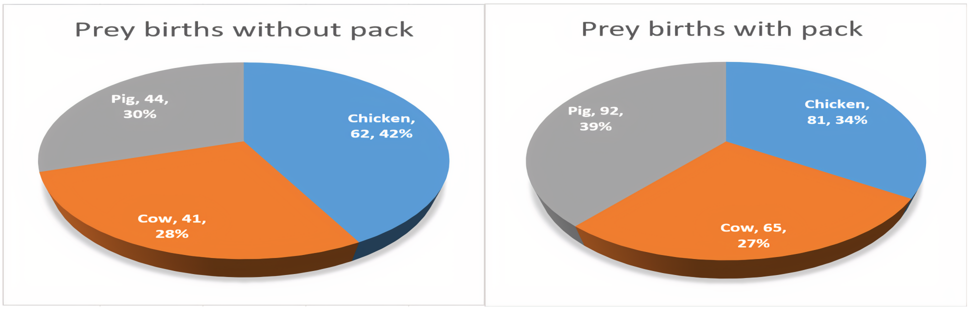
9.2 The pack forming ability and the advantages of the mechanic

To demonstrate the pack forming ability usefulness, we ran multiple simulations with the same animals (types and properties, same animal starting numbers), on the same terrain with random starting points, the only thing we changed is the pack forming ability for all or none of the animals. Without a cohesion force, the animals make their decisions fully separately from each other (ignoring each other, except in the mating process) and with the packing mechanism the pack's members always try to stay and move together.

We observed our prey animals (Chicken, Cow, Pig) more closely, because they had a more a more notable change in their populations than the preys' .

Hypothesis 1 (H1): *Forming a pack could benefit in the mating process, because the animals are close together and they don't need to search long for a mate to reproduce which will greatly increase the population.*

The simulations support this assumption. The prey animals have a default faster reproduction rate than the predators, furthermore the mating process is accelerated rapidly due to the close distance between the species, because the animal doesn't have to spend a lot of time to find a suitable mate.



(a) Without pack **(b)** With pack

Figure 16: Prey animals' birth numbers.

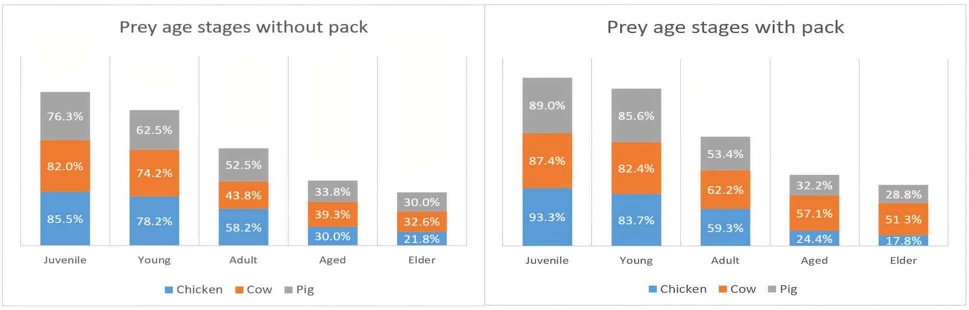
The difference between these 3 animals is we have given them different maximal child births at labor (pig 4, chicken 3, cow 1).

Hypothesis 2 (H2): *Forming a pack also could improve their lifetime or help in their species survival.*

The simulations support this assumption as well,

1. With the growing population (shown in Hypothesis 1) the predators have a lot more food source to ease their hunger, which extends the predator's lifetime.

2. In a pack, animals with different ages are staying close together. When a predator attacks, it will choose the closest prey to catch. The younger ones, because they have more speed, they can flee more quicker than the older members of the pack. The predator will likely change it's hunting target to them, because they are easier to catch. After the successful hunt (and the tragical death of an elder) the predator becomes satiated by consuming it's prey and leaves the pack alone for a while. With the survival of the younger animals, the pack has more repopulation ability, which extends the pack's and also their species' survival.



(a) Without pack **(b)** With pack

Figure 17: Prey animals' reached age stages

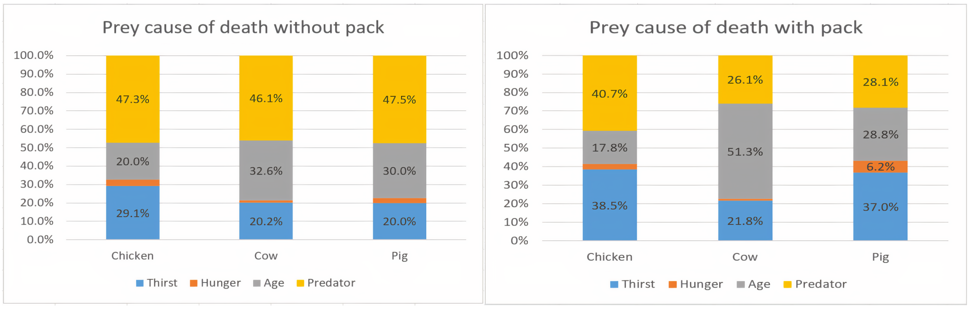
Here we can see the dispersion of the reached age groups. This means, if an animal dies in it's current (e.g., Adult stage), it will be counted in the previous (Juvenile, Young) and current (Adult) groups, but not in the following (Aged and Elder). The percentages show how much of the population has reached that designated group, for example in [Figure 17a], 85.5% at the Juvenile column's Chicken section means that 14.5% died early at the age of Puppy. The pack mechanic had a greater impact on the younger groups, but all groups had an increase in their combined (value of considering the 3 animals together) number.

Hypothesis 3 (H3): *Sticking together has it's danger, when predators attack they can surround more prey at once or in a shorter time.*

In our simulation, every animal was set as scavengers carnivores, whom consume corpses. With this presumption this hypothesis considered as false.

1. If the predators attack in a large group, they can surround an area, but they will likely target one or few of the prey at once (shown in Hypothesis 2) and most likely catch older preys, additionally, when a predator catches a prey, other predators, if they are close, they can leave their hunting target to escape and join in the corpse consumption, this way no leftover from the corpse will go to waste (predators hunger will be more efficiently satisfies) and the hunt for the pack's other members will only continue if the predators are still hungry.

2. If a member of the pack dies in natural causes, like due to thirst, hunger or age, the predators who followed the pack, but didn't attack yet, can find the recently deceased animal's corpse easier and consume it, if they are scavenger carnivores. Without the pack function, the deceased animal can die in an abandoned section of the terrain, without ever been found and consumed by a predator, leading to more hungry predators, whom will try to catch alive members of the pack, decreasing their population.



(a) Without pack **(b)** With pack

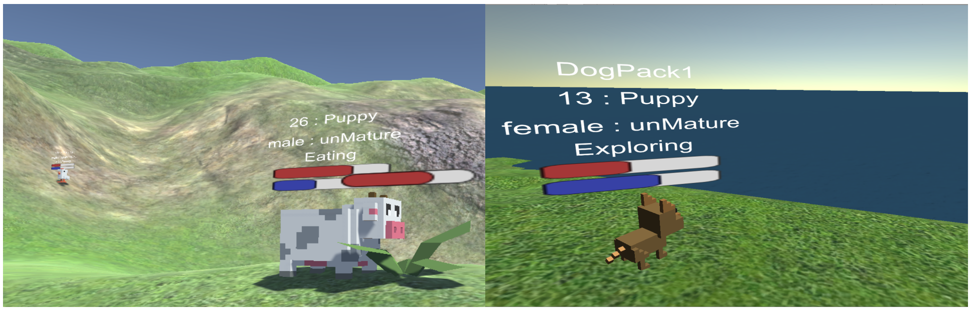
Figure 18: Vegetarian cause of death

As we can see in the above figure, the dispersion of predator death cases were drastically decreased.

Hypothesis 4 (H4): *The food can decrease drastically near the pack's area because of the many hungry mouth.*

1. Considering the predator animals, this could depend on the hunting instinct. If a predator pack's consumption is greater than a prey pack's reproduction rate or multiple predator packs hunt the same prey pack it could be a disadvantage. Other than this situation, the pack's survival rate is increased (shown in Hypothesis 2 and Hypothesis 3).

2. Considering the prey animals, this hugely depends on the landscape's layout and size. If the vegetation runs out in an area the pack will wander to a different area. This option is only available if the map's size is big enough with few natural huge obstacle formations like oceans, mountains.



(a) High mountains **(b)** Huge oceans
Figure 19: Limitations of the land

10 Conclusion and future works

With the "Animal Farm" Framework, we created a simulation platform for everyone to run personal ecosystem simulations, with customizable animals in a simplified behavior and logic system. Running and observing this simulations, not only aesthetically beautiful, but generates ecosystem data in a large quantity. Using this data helps analyse and predict animal behaviours, extinction processes and any kind of change in the system on a wide variety (real and imaginary landscapes using heightmap data) of terrains. These kind of predictions can be very useful, when we run simulations of a real landscape terrain, with the replications of the animals living in that specific area, to observe and have a slight insight into the future of the real world's ecosystem.

In the Framework, we are working on the implementation of more mechanics in animals' behaviors, functionalities of the framework and monitoring, exporting and analysing more data for different experiments. The plan is to first upgrade and optimize some of the already implemented mechanics like a more complex calculation of the escape direction vector, where the animal takes into account all the enemies and their distances and calculates the appropriate summary vector instead of the closest enemy and reworking the animal's view mechanic to a more optimized approach.

We are also want to implement multiple new different mechanics, like a gene mechanic, which will influence the mating procedure, what kind of attractive attributes are the females and the males are looking for and whether they accept other ones approach or refuse them. Offsprings inherit different combinations of parental genes, also they can mutate a little the inherited gene pool at birth, taking evolutionary steps. We consider developing a role mechanic, which would mean, within the packs there could be different roles (such as hunter, explorer etc.) based on the animal's outstanding abilities (using the gene system) to split jobs within packs. A hiding ability, smaller animals will be able to hide in specific locations (like bushes, nests) to escape from predators. New day-night cycles and sleeping mechanics. A family mechanic which represents a strong cohesion, like a pack, but this will be a closer bond, for example if a family member is in trouble, they will help immediately, the young ones may offer or share the food to the elder ones if they dangerously hungry. A basic form of communication (information transfer) between family, pack, race, to share location information like food and water location, dangerous areas and basic planned hunting process like attacking and surrounding the prey together. And an advanced fighting mechanic which replaces the current instant killing hunting mechanic of the predators. Introducing a new vitality meter (alongside the hunger and thirst) which shows, if the animal is full with energy (like not thirsty or hungry and not wounded) or lowered, because some causes, this will determine the damage they take and make to others. If taking damage or having a lower energy level, they start to suffer a penalty in attributes, like they will be reduced (e.g., the speed attribute will be lowered) until the energy reaches an optimal level and if the animal can keep it above this level the attribute will be slowly restored to the original value. With this new fighting mechanic, new battle types can be developed like intra-species struggles for food or females or during a hunt, a duel can take place where the prey can attacks back instead of just fleeing (creating a damage system, that takes down life force from both participants), where if one of the animal has a low life level he can decide to run away. Fellow animals nearby can help in the fight, if they decide to. This way not always the predators will win a fight if multiple preys unite to fight back and they kill or chase away a single predator.

Multiple functions are under consideration of adding, like an animal customization interface to create new animals and set the properties inside of the program. This custom created animals can be saved and spawned in simulations just like our 6 basic provided animals. New random/procedurally generated maps, using noises (Perlin [16] and other noise types merged) instead of using only heightmaps. For only entertainment we considered to implement one of the Unity's VR (virtual reality) [7] functions into our simulation, to walk alongside of our animals and observe their behaviour closely, instead of just flying above them. If this implementation is successful and popular, we are considering adding additional functionalities like interacting with animals (human interference), where we could chase away predators or feed gentle animals and create farms with animals to experiment with the gene system or just testing our custom created animals.

We ran a large number of simulations, but the data gathering in the current framework wasn't nearly complete. Only the major statistics were monitored like number changes (every time something happened, the program logged the current statistic numbers) in the living (distributed by age stages) and the dead (distributed by death types) and the overall animal numbers in the whole simulations. We are planning to log and analyse more information like vegetation dispersion and monitoring vegetarian feeding habits, pack wondering patterns, influence of terrain formations (like hills and lakes) on the animals' habits.

The "Animal Farm" Framework can be accessed on Github [25]. The system can freely be downloaded, extended and modified as a Unity project for personal use. To try out and use a live version of the program without any installation, visit the browser version [26].

Acknowledgements

The project has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).

References

[1] C. Adami, C.T. Brown, Evolutionary learning in the 2D artificial life system "Avida", arXiv preprint adap-org/9405003, (1994). ⇒63

[2] H.R. Akcakaya, R. Arditi, L.R. Ginzburg, Ratio-dependent predation: an abstraction that works, *Ecology*, **76,** 3 (1995) 995–1004. ⇒62

[3] A.A. Berryman, A.P. Gutierrez, R. Arditi, Credible, parsimonious and useful predator-prey models: a reply to Abrams, Gleeson, and Sarnelle, *Ecology*, **76,** 6 (1995) 1980–1985. ⇒62

[4] F. Corno, E. Sanchez, G. Squillero, Exploiting co-evolution and a modified island model to climb the core war hill, *The 2003 Congress on Evolutionary Computation*, 2003. CEC'03, IEEE, **3,** (2003) 2217–2221. ⇒63

[5] J. Craighead, J. Burke, R. Murphy, Using the unity game engine to develop SARGE: A case study, *Computer*, **4552** (2007). ⇒64

[6] C. Dachsbacher, M. Stamminger, Rendering procedural terrain by geometry image warping, *Eurographics Symposium on Rendering* (2004) 103–110. ⇒71

[7] J. Glover, Jesse J. Linowes, *Complete Virtual Reality and Augmented Reality Development with Unity: Leverage the power of Unity and become a pro at creating mixed reality applications*, Packt Publishing Ltd, 2019. ⇒83

[8] J.K. Haas, *A history of the unity game engine*, Worcester Polytechnic Institute, 2014. ⇒65

[9] D.G.Jones, A.K. Dewdney, *Core war guidelines*, Department of Computer Science, the University of Western Ontario, 1992. ⇒63

[10] M. Komosinski, The world of framsticks: Simulation, evolution, interaction, *International Conference on Virtual Worlds*, Springer, 2000, pp. 214–224. ⇒64

[11] M. Komosinski, *Framsticks: A platform for modeling, simulating, and evolving 3D creatures*, *Artificial Life Models in Software*, Springer, 2005. 37–66. ⇒64

[12] M. Komosinski, Sz. Ulatowski, Framsticks, *Artificial Life Models in Software*, 2007. ⇒64

[13] M. Komosinski, Framsticks, *Artificial Life Models in Software*, Springer, 2009. pp. 107–148. ⇒64

[14] M. Komosinski, et al., The Framsticks system: versatile simulator of 3D agents and their evolution, *Kybernetes*, MCB UP Ltd, 2003. ⇒64

[15] M. Komosinski, Sz. Ulatowski, Framsticks: Towards a simulation of a nature-like world, creatures and evolution, *European Conference on Artificial Life*, Springer, 1999. pp. 261–265. ⇒64

[16] H. Li, X. Tuo, Y. Liu, X, Jiang, A parallel algorithm using Perlin noise superposition method for terrain generation based on CUDA architecture, *International Conference on Materials Engineering and Information Technology Applications* (MEITA 2015), Atlantis Press, 2015. ⇒82

[17] A. J. Lotka, *Elements of Physical Biology*, Williams and Wilkins Company, 1925. ⇒62

[18] K. Reddy, N. Ramacharyulu, A three species ecosystem comprising of two predators competing for a prey, *International Conference on Simulation of Adaptive Behavior*, Springer, 2011. pp. 208–218. ⇒62

[19] T. Schmickl, K. Crailsheim, Bubbleworld. Evo: Artificial evolution of behavioral decisions in a simulated predator-prey ecosystem, *Advances in Applied Science Research*, **2** (2006) 594–605. ⇒63

[20] S. Tom, Ray, An approach to the synthesis of life, Physica D, 1992. ⇒63

[21] Y.V. Tyutyunov, L.I. Titova, *From Lotka–Volterra to Arditi–Ginzburg: 90 years of evolving trophic functions*, *Biology Bulletin Reviews*, Springer, **10** (2020) 167–185. ⇒62

[22] V. Volterra, *Leconssen la theorie mathematique de la leitte pou lavie*, 1931. ⇒ 62

[23] R. Wiegert, G. Richard, Simulation Models of Ecosystems, *Annual Review of Ecology and Systematics*, **1,** 6 (1975) 311–338. ⇒62

[24] L.Yaeger, Computational genetics, physiology, metabolism, neural systems, learning, vision, and behavior or Poly World: Life in a new context, *Santa Fe Institute Studies in the Sciences of Complexity*, Addison-Wesley Publishing Co. **17** (1994). ⇒64

[25] ∗ ∗ ∗ Github project repository, (2020), https://github.com/Wornox/AnimalFarmFramework. ⇒83

[26] ∗ ∗ ∗ Github runnable browser version of the program, (2020), https://wornox.github.io/AnimalFarmWebGL. ⇒83

[27] ∗ ∗ ∗ Unity (game engine), (2020), http://www.unity3d.com/. ⇒64

[28] ∗ ∗ ∗ Unity asset store, (2020), https://assetstore.unity.com/. ⇒66

[29] ∗ ∗ ∗ Unity asset: 5 animated Voxel animals by "VoxelGuy", (2020), https://assetstore.unity.com/packages/3d/characters/animals/5-animated-voxel-animals-145754. ⇒66

[30] ∗ ∗ ∗ Unity asset: Free Trees by "AdaKing", (2020), https://assetstore.unity.com/packages/3d/vegetation/trees/free-trees-103208. ⇒66

[31] ∗ ∗ ∗ Unity asset: Low Poly Nature - FREE Vegetation by "Elcanetay", (2020), https://assetstore.unity.com/packages/3d/vegetation/low-poly-nature-free-vegetation-134006. ⇒66

[32] ∗ ∗ ∗ Unity asset: Runtime File Browser by "yasirkulaa", (2020), https://assetstore.unity.com/packages/tools/gui/runtime-file-browser-113006. ⇒66

[33] ∗ ∗ ∗ Unity asset: Voxel Animals Pack by "VoxelGuy", (2020), https://assetstore.unity.com/packages/3d/characters/animals/voxel-animals-pack-133366. ⇒66

Received: January 8, 2021 • Revised: April 23, 2021

DOI: 10.2478/ausi-2021-0005

On ordering of minimal energies in bicyclic signed graphs

S. PIRZADA
Department of Mathematics, University of Kashmir, Srinagar, India
email: `pirzadasd@kashmiruniversity.ac.in`

Tahir SHAMSHER
Department of Mathematics, University of Kashmir, Srinagar, India
email: `tahir.maths.uok@gmail.com`

Mushtaq A. BHAT
Department of Mathematics, National Institute of Technology, Srinagar, India
email: `mushtaqab1125@gmail.com`

Abstract. Let $S = (G, \sigma)$ be a signed graph of order n and size m and let $x_1, x_2, \dots, x_n$ be the eigenvalues of S. The energy of S is defined as $\mathcal{E}(S) = \sum\limits_{j=1}^{n} |x_j|$. A connected signed graph is said to be bicyclic if $m = n + 1$. In this paper, we determine the bicyclic signed graphs with first 20 minimal energies for all $n \geq 30$ and with first 16 minimal energies for all $17 \leq n \leq 29$.

1 Introduction

Let $S = (G, \sigma)$ be a signed graph, where $G = (V, E)$ is the underlying graph of S and $\sigma : E \to \{-1, 1\}$ is the signing function (or signature). We represent a positive edge by a plain line and a negative edge by a dotted line. The sign of a signed cycle is defined as the product of signs of its edges. A signed cycle

Computing Classification System 1998: G.2.2
Mathematics Subject Classification 2010: 05C22, 05C50, 05C76
Key words and phrases: Bicyclic signed graph, balance, spectrum, signed energy, ordering

is said to be positive (resp., negative) if its sign is positive (resp., negative), that is, it contains an even (resp., odd) number of negative edges. A signed graph is said to be balanced if each of its cycle is positive and unbalanced, otherwise. Throughout, by C_n^+, we denote a positive cycle of order n and by C_n^- a negative cycle of order n. A connected signed graph of order n is said to be unicyclic or bicyclic according as the number of its edges is respectively n or $n+1$.

The adjacency matrix of a signed graph S with vertex set $\{v_1, v_2, \dots, v_n\}$ is the $n \times n$ matrix $A(S) = (a_{ij})$, where $a_{ij} = \sigma(v_i, v_j)$ if v_i and v_j are adjacent and zero, otherwise. The adjacency matrix $A(S)$ is real symmetric and so has real eigenvalues. Let $\psi(S, x)$ denote the characteristic polynomial of the adjacency matrix of S. The eigenvalues of $A(S)$ are called the eigenvalues of S.

Gutman [7] defined the energy of a graph as the sum of the absolute values of eigenvalues of its adjacency matrix. Germina, Hameed and Zaslavsky [6] extended this concept to signed graphs. The energy of a signed graph S with eigenvalues $x_1, x_2, \dots, x_n$ is defined as $\mathcal{E}(S) = \sum\limits_{j=1}^{n} |x_j|$. Bhat and Pirzada [2] characterized the unicyclic signed graphs with minimal energy. Bhat et al. [4], characterized the bicyclic signed graphs with minimal and second minimal energy. Similar problems for graphs, digraphs, signed graphs and signed digraphs have been studied in [3, 5, 9, 10, 11, 13, 14, 15, 16, 17].

Let S be a signed graph with vertex set V. Switching S by a set $X \subset V$ means reversing the signs of all the edges between X and its complement. Two signed graphs of the same order are said to be switching equivalent if one can be obtained from the other by a switching. Switching equivalence is an equivalence relation on the signings of a fixed graph. For more details about switching see [4, 15]. An equivalence class is called a switching class. Switching a signed graph does not change the sign of cycles (see [15]), switching equivalent signed graphs have the same set of positive cycles, and they are either both balanced or both unbalanced. Also, switching preserves the spectrum. So, as long as spectra is concerned, we use a single signed graph for a switching class and call that the representative of the switching class.

The rest of the paper is organized as follows. In section 2, we give some definitions and state preliminary results, which will be used to prove our main results. All the main results are in section 3. In that section, we compare energy by using integral formula, Descartes' rule of signs, by cut set deletion and energy change techniques.

2 Preliminaries

In this section, we give some notations, definitions and state some of the results which will be used in the sequel. A basic figure is a signed graph whose components are signed cycles or edges or both.

Theorem 1 [1] *If S is a signed graph with characteristic polynomial*

$$\psi(S,x) = x^n + a_1(S)x^{n-1} + \cdots + a_{n-1}(S)x + a_n(S),$$

then

$$a_k(S) = \sum_{L\in\mathscr{L}_k} (-1)^{p(L)} 2^{|c(L)|} \prod_{X\in c(L)} s(X),$$

for all $k = 1, 2, \ldots, n$, where $\mathscr{L}_k$ is the set of all basic figures L of S of order k, $p(L)$ denotes number of components of L, $c(L)$ denotes the set of all cycles of L and $s(X)$ the sign of cycle X.

The following is the integral formula for the energy of signed graphs.

Theorem 2 [2] *Let S be a signed graph on n vertices with characteristic polynomial $\psi(S,x) = x^n + a_1(S)x^{n-1} + \cdots + a_{n-1}(S)x + a_n(S)$. Then*

$$\mathcal{E}(S) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \frac{1}{x^2} \log\left[\left(\sum_{k=0}^{\lfloor\frac{n}{2}\rfloor}(-1)^k a_{2k}(S)x^{2k}\right)^2 + \left(\sum_{k=0}^{\lfloor\frac{n}{2}\rfloor}(-1)^k a_{2k+1}(S)x^{2k+1}\right)^2\right] dx.$$

In a signed graph S, if the even and odd coefficients respectively alternate in sign, we have two cases to consider.
Case (i). $(-1)^k a_{2k}(S) \geq 0$ and $(-1)^k a_{2k+1}(S) \leq 0$ for $k \geq 0$.
Case (ii). $(-1)^k a_{2k}(S) \geq 0$ and $(-1)^k a_{2k+1}(S) \geq 0$ for $k \geq 0$.
Put $b_k(S) = |a_k(S)|$, then for a signed graph S with even and odd coefficients alternating, above integral formula takes the form

$$\mathcal{E}(S) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \frac{1}{x^2} \log\left[\left(\sum_{k=0}^{\lfloor\frac{n}{2}\rfloor} b_{2k}(S)x^{2k}\right)^2 + \left(\sum_{k=0}^{\lfloor\frac{n}{2}\rfloor} b_{2k+1}(S)x^{2k+1}\right)^2\right] dx.$$

Let S' and S'' be two signed graphs of same order with even and odd coefficients of their respective characteristic polynomials alternating in sign. If $b_{2k}(S') = b_{2k}(S'')$ and $b_{2k+1}(S') = b_{2k+1}(S'')$ for all $k \geq 0$, then it is clear that $E(S') = E(S'')$. In this case, we write $S' \sim S''$. Further, if $b_{2k}(S') \leq b_{2k}(S'')$ and $b_{2k+1}(S') \leq b_{2k+1}(S'')$ for all $k \geq 0$, we write $S' \preceq S''$ or $S'' \succeq S'$. If $b_{2k}(S') \leq b_{2k}(S'')$ and $b_{2k+1}(S') \leq b_{2k+1}(S'')$ for all $k \geq 0$ and for some k_0, strict inequality holds in one of the two inequalities, then we write $S' \prec S''$ or $S'' \succ S'$. Clearly, $\preceq$ is a transitive relation on the coefficients. Thus, if $S' \preceq S''$, we see that $E(S') \leq E(S'')$. Moreover, if $S' \prec S''$, then $E(S') < E(S'')$.

Lemma 3 [4] *Let $e = uv$ be an edge of a signed graph S. Then*

$$\begin{aligned} \psi(S,x) &= \psi(S-\{e\},x) - \psi(S-\{u,v\},x) \\ &- 2\left(\sum_{Z\in\mathscr{C}^+_{uv}} \psi(S-V(Z),x) - \sum_{Z\in\mathscr{C}^-_{uv}} \psi(S-V(Z),x)\right). \end{aligned}$$

where $\mathscr{C}^+_{uv}$ and $\mathscr{C}^-_{uv}$ respectively denote the set of positive and negative cycles containing the edge $e = uv$ and by $V(Z)$ we mean the vertex set of Z.

From this recurrence relation, it is easy to obtain the following lemma.

Lemma 4 *If S is a signed graph with even and odd coefficients alternating in sign and if (u,v) is the pendent edge of S with pendent vertex v, then*

$$b_i(S) = b_i(S-v) + b_{i-2}(S-v-u).$$

It is well known that there are three classes of bicyclic signed graphs, which are defined as follows.

(1). For positive integers p and q with $p,q \geq 3$ and $6 \leq p+q \leq n$, we denote by $CC[n,p,q]$, the class of bicyclic signed graphs of order n and having two vertex disjoint cycles of length p and q.

(2). For positive integers p and q with $p,q \geq 3$ and $6 \leq p+q \leq n+1$, we denote by $\infty(n,p,q)$, the class of bicyclic signed graphs of order n with two cycles of lengths p and q such that these cycles have exactly one vertex in common.

(3). For positive integers p, q and r with $(p-r) \geq r$, $(q-r) \geq r$, $p,q \geq 3, r \geq 1$ and $6 \leq p+q \leq n-r+1$, we denote by $\theta(n,p,q,r)$, the class of bicyclic

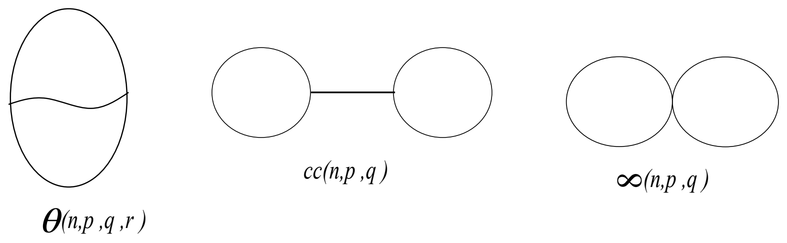


Figure 1: Three classes of bicyclic signed graphs

signed graphs on n vertices with three cycles; one has length p, and the other has length q and two cycles share r edges so that the third cycle has $p+q-2r$ edges. For illustration, see Figure 1, where we have not shown non-cyclic edges which can be present.

Lemma 5 [4] *Let* C *be a cut set of a signed graph* S. *Then* $\mathcal{E}(S-C) \leq \mathcal{E}(S)$. *Moreover, if* C *is a single edge, then* $\mathcal{E}(S-C) < \mathcal{E}(S)$.

Given a signed star S_n on n vertices, let $S_{n,n}$ denote the collection of unicyclic signed graphs on n vertices such that each element of $S_{n,n}$ is obtained from S_n by adding a single signed edge between any two non adjacent vertices. Then there are two switching classes in $S_{n,n}$, one containing unicyclic signed graphs with positive cycle C_3^+ and other containing unicyclic signed graphs with negative cycle C_3^-. In $S_{n,n}$, if a unicyclic signed graph contains C_3^+, we denote it by $S_{n,n}^1$ and if it contains C_3^-, we denote it by $S_{n,n}^2$. We denote a signed path on n vertices by P_n.

The following result characterizes unicyclic signed graphs with minimal energy [2].

Lemma 6 *Among all unicyclic signed graphs with* $n \geq 3$ *vertices,* $n \neq 4, 5$, *each signed graph in* $S_{n,n}$ *has the minimal energy. Moreover, for* $n = 4$, C_4^+ *has the minimal energy. Further, for* $n = 5$, *the signed graph* S *as shown in Figure* 5 *has the minimal energy.*

Consider the graph $K_4 - e$ and nonnegative integer $0 \leq k \leq n-4$. Let $G(K_4 - e, n, k)$ be the graph obtained from $K_4 - e$ by respectively identifying the centers of the stars S_{k+1} and S_{n-k-3} to two vertices of degree 3. Let $S_{n,n+1}^k$ denote the collection of bicyclic signed graphs on n vertices obtained from

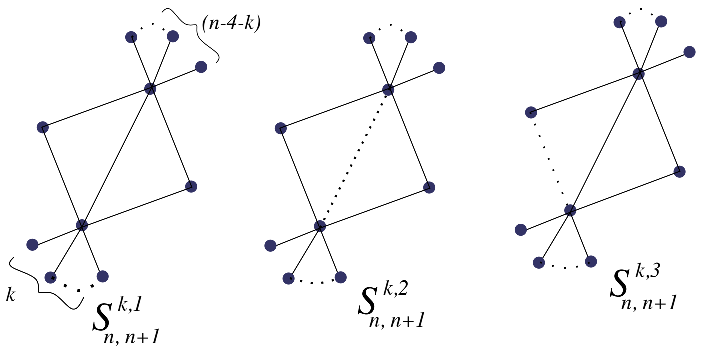


Figure 2: Three switching classes in $S^{k}_{n,n+1}$

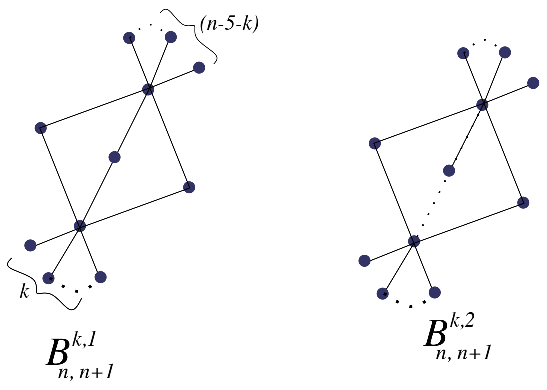


Figure 3: Two switching classes in $B^{k}_{n,n+1}$

$G(K_4 - e, n, k)$. There are three switching classes in $S^{k}_{n,n+1}$. We use $S^{k,1}_{n,n+1}$, $S^{k,2}_{n,n+1}$ and $S^{k,3}_{n,n+1}$ as representative of these three switching classes as shown in Figure 2. Note that $S^{k,1}_{n,n+1}$ is balanced, $S^{k,2}_{n,n+1}$ contains two negative cycles of length 3 and a positive cycle of length 4 while as $S^{k,3}_{n,n+1}$ has one positive cycle of length 3, one negative cycle of lengths 3 and one negative cycle of length 4. The following result characterizes bicyclic signed graphs with minimal and second minimal energy[4].

Lemma 7 *Among all bicyclic signed graphs with $n \geq 12$ vertices, $S^{0,1}_{n,n+1}$ and $S^{0,2}_{n,n+1}$ have minimal energy and $S^{0,3}_{n,n+1}$ has the second minimal energy.*

3 Main results

Given a complete bipartite graph $K_{2,3}$ and nonnegative integer $0 \leq k \leq n-5$, let $G(K_{2,3}, n, k)$ be the graph obtained by respectively identifying the centers of the stars S_{k+1} and S_{n-k-4} to two vertices of degree 3. Let $B^{k}_{n,n+1}$ denote the collection of bipartite bicyclic signed graphs on n vertices obtained from $G(K_{2,3}, n, k)$. There are two switching classes in $B^{k}_{n,n+1}$. We use $B^{k,1}_{n,n+1}$ and $B^{k,2}_{n,n+1}$ as representative of these two switching classes, for illustration, see Figure 3. $B^{k,1}_{n,n+1}$ contains three positive cycles of length 4 and $B^{k,2}_{n,n+1}$ contains two negative cycles of length 4 and one positive cycle of length 4. With these notations, we have the following observation.

Lemma 8 (i) *For all* $n \geq 6$ *and* $1 \leq k \leq 3$, $\mathcal{E}(B^{k-1,1}_{n,n+1}) < \mathcal{E}(S^{k,1}_{n,n+1}) = \mathcal{E}(S^{k,2}_{n,n+1})$.
(ii) *For all* $n \geq 6$ *and* $k \geq 0$, $\mathcal{E}(S^{k,1}_{n,n+1}) = \mathcal{E}(S^{k,2}_{n,n+1}) < \mathcal{E}(S^{k,3}_{n,n+1})$.
(iii) *For all* $n \geq 6$ *and* $k \geq 1$, $\mathcal{E}(S^{k,3}_{n,n+1}) < \mathcal{E}(B^{k-1,2}_{n,n+1})$.
(iv) *For all* $n > 2k + 12$ *and* $k \geq 1$, $\mathcal{E}(B^{k-1,2}_{n,n+1}) < \mathcal{E}(B^{k,1}_{n,n+1})$.
(v) *For all* $n \geq 12$, $\mathcal{E}(S^{0,3}_{n,n+1}) < \mathcal{E}(B^{0,1}_{n,n+1})$.

Proof. (i). By Sach's theorem, we have

$$\psi(B^{k-1,1}_{n,n+1}, x) = x^{n-4}\{x^4 - (n+1)x^2 + [(k+2)(n-k-4) + 3k - 3]\},$$

$$\psi(S^{k,1}_{n,n+1}, x) = x^{n-4}\{x^4 - (n+1)x^2 - 4x + [(k+2)(n-k-4) + 2k]\}$$

and

$$\psi(S^{k,2}_{n,n+1}, x) = x^{n-4}\{x^4 - (n+1)x^2 + 4x + [(k+2)(n-k-4) + 2k]\}.$$

It is clear that $B^{k-1,1}_{n,n+1} \prec S^{k,1}_{n,n+1}, S^{k,2}_{n,n+1}$ for all $1 \leq k \leq 3$ and $S^{k,1}_{n,n+1} \sim S^{k,2}_{n,n+1}$, therefore
$\mathcal{E}(B^{k-1,1}_{n,n+1}) < \mathcal{E}(S^{k,1}_{n,n+1}) = \mathcal{E}(S^{k,2}_{n,n+1})$ for all $n \geq 6$ and $1 \leq k \leq 3$.
(ii). The characteristic polynomial of $S^{k,r}_{n,n+1}$ for $r = 1, 2, 3$ are given by

$$\psi(S^{k,1}_{n,n+1}, x) = x^{n-4}\{x^4 - (n+1)x^2 - 4x + [(k+2)(n-k-4) + 2k]\},$$

$$\psi(S^{k,2}_{n,n+1}, x) = x^{n-4}\{x^4 - (n+1)x^2 + 4x + [(k+2)(n-k-4) + 2k]\}$$

and

$$\psi(S^{k,3}_{n,n+1}, x) = x^{n-4}\{x^4 - (n+1)x^2 + [(k+2)(n-k-4) + 2k + 4]\}.$$

Clearly, $S^{k,1}_{n,n+1} \sim S^{k,2}_{n,n+1}$ and so $\mathcal{E}(S^{k,1}_{n,n+1}) = \mathcal{E}(S^{k,2}_{n,n+1})$. Therefore, to compare the energy of $S^{k,r}_{n,n+1}$ for $r = 1,2$ and $S^{k,3}_{n,n+1}$, it is enough to compare the energy of $S^{k,1}_{n,n+1}$ and $S^{k,3}_{n,n+1}$. We see that even and odd coefficients of $S^{k,r}_{n,n+1}$ for $r = 1,2,3$, alternate in sign but coefficients are not quasi-order comparable for $r = 1$ or 2 and $r = 3$. We will compare energy using Coulson's integral formula by directly solving the integrals. We have $\mathcal{E}(S^{k,3}_{n,n+1}) - \mathcal{E}(S^{k,1}_{n,n+1})$

$$= \frac{1}{\pi}\int_0^\infty \ln \frac{\{1+(n+1)x^2+[(k+2)(n-k-4)+2k+4]x^4\}^2}{\{1+(n+1)x^2+[(k+2)(n-k-4)+2k]x^4\}^2+16x^6}dx.$$

Put

$$\alpha_1(x) = \{1+(n+1)x^2+[(k+2)(n-k-4)+2k+4]x^4\}^2$$

and

$$\beta_1(x) = \{1+(n+1)x^2+[(k+2)(n-k-4)+2k]x^4\}^2+16x^6.$$

Since $n \geq k+4$, we get $\alpha_1(x) - \beta_1(x) = 8x^4 + 8(n-1)x^6 + 8[(k+2)(n-k-4)+2k+2]x^8 > 0$ for $n \geq 6$ and $x > 0$. Thus, $\mathcal{E}(S^{k,3}_{n,n+1}) > \mathcal{E}(S^{k,1}_{n,n+1})$.

(iii). The characteristic polynomial of $B^{k-1,2}_{n,n+1}$ and $S^{k,3}_{n,n+1}$ are given by

$$\psi(B^{k-1,2}_{n,n+1}, x) = x^{n-4}\{x^4 - (n+1)x^2 + [(k+2)(n-k-4)+3k+5]\}$$

and

$$\psi(S^{k,3}_{n,n+1}, x) = x^{n-4}\{x^4 - (n+1)x^2 + [(k+2)(n-k-4)+2k+4]\}.$$

Clearly, $S^{k,3}_{n,n+1} \prec B^{k-1,2}_{n,n+1}$ for all $k \geq 1$ and therefore $\mathcal{E}(S^{k,3}_{n,n+1}) < \mathcal{E}(B^{k-1,2}_{n,n+1})$ for all $n \geq 6$ and $k \geq 1$.

(iv). Again, the characteristic polynomial of $B^{k-1,2}_{n,n+1}$ and $B^{k,1}_{n,n+1}$ are respectively, given by

$$\psi(B^{k-1,2}_{n,n+1}, x) = x^{n-4}\{x^4 - (n+1)x^2 + [(k+2)(n-k-4)+3k+5]\}$$

and

$$\psi(B^{k,1}_{n,n+1}, x) = x^{n-4}\{x^4 - (n+1)x^2 + [(k+3)(n-k-5)+3k]\}.$$

Clearly, $B^{k-1,2}_{n,n+1} \prec B^{k,1}_{n,n+1}$ for all $n > 2k+12$. Therefore, $\mathcal{E}(B^{k-1,2}_{n,n+1}) < \mathcal{E}(B^{k,1}_{n,n+1})$ for all $n > 2k+12$.

(v). The characteristic polynomial of $S^{0,3}_{n,n+1}$ and $B^{0,1}_{n,n+1}$ are given by

$$\psi(S^{0,3}_{n,n+1}, x) = x^{n-4}\{x^4 - (n+1)x^2 + [2(n-4)+4]\}$$

and

$$\psi(B^{0,1}_{n,n+1}, x) = x^{n-4}\{x^4 - (n+1)x^2 + [3(n-5)]\}.$$

Clearly, $S^{0,3}_{n,n+1} \prec B^{0,1}_{n,n+1}$ for all $n \geq 12$ and therefore $\mathcal{E}(S^{0,3}_{n,n+1}) < \mathcal{E}(B^{0,1}_{n,n+1})$ for all $n \geq 12$. □

Let $Q^{r,1}_{n,n+1}$, $r = 1, 2, 3$ and $H^{1}_{n,n+1}$ be the graphs as shown in Figure 4. Then it is easy to see that there are two switching classes on the signings of $Q^{1,1}_{n,n+1}$. Let $Q^{1,1}_{n,n+1}$ and $Q^{1,2}_{n,n+1}$ be the representative for these two switching classes, where $Q^{1,1}_{n,n+1}$ contains C_4^+, C_4^+, C_4^+ and $Q^{1,2}_{n,n+1}$ contains C_4^-, C_4^- and C_4^+. There are four switching classes on the signings of $Q^{2,1}_{n,n+1}$. Let $Q^{2,1}_{n,n+1}$, $Q^{2,2}_{n,n+1}$, $Q^{2,3}_{n,n+1}$ and $Q^{2,4}_{n,n+1}$, respectively be the representative for these four switching classes, where $Q^{2,1}_{n,n+1}$ contains C_3^+, C_4^+ and C_5^+; $Q^{2,2}_{n,n+1}$ contains C_3^-, C_4^- and C_5^+; $Q^{2,3}_{n,n+1}$ contains C_3^-, C_4^+, C_5^-; and $Q^{2,4}_{n,n+1}$ contains C_3^+, C_4^- and C_5^-. There are three switching classes on the signings of $Q^{3,1}_{n,n+1}$. Let $Q^{3,1}_{n,n+1}$, $Q^{3,2}_{n,n+1}$ and $Q^{3,3}_{n,n+1}$, respectively be the representative for these three switching classes, where $Q^{3,2}_{n,n+1}$ is the signed graphs obtained from $Q^{3,1}_{n,n+1}$, by making both triangles negative in $Q^{3,1}_{n,n+1}$. Also, $Q^{3,3}_{n,n+1}$ is the signed graph obtained from $Q^{3,1}_{n,n+1}$ by making one triangle negative and other triangle positive in $Q^{3,1}_{n,n+1}$. There are three switching classes on the signings of $H^{1}_{n,n+1}$. We use $H^{r}_{n,n+1}$ for $r = 1, 2, 3$, as the representative for these switching classes. $H^{1}_{n,n+1}$ is balanced, $H^{2}_{n,n+1}$ has both triangles negative and $H^{3}_{n,n+1}$ has one positive triangle and one negative triangle. With these notations, we have the following observation.

Lemma 9 (i) *For all* $n \geq 10$, $\mathcal{E}(B^{1,1}_{n,n+1}) < \mathcal{E}(Q^{1,1}_{n,n+1}) < \mathcal{E}(S^{2,1}_{n,n+1})$.
(ii) *For all* $n \geq 10$, $\mathcal{E}(Q^{1,2}_{n,n+1}) > \mathcal{E}(B^{2,2}_{n,n+1})$.
(iii) *For all* $n \geq 10$, $\mathcal{E}(S^{2,3}_{n,n+1}) < \mathcal{E}(Q^{2,1}_{n,n+1}) = \mathcal{E}(Q^{2,3}_{n,n+1}) < \mathcal{E}(B^{1,2}_{n,n+1})$.
(iv) *For all* $n \geq 10$, $\mathcal{E}(Q^{2,2}_{n,n+1}) = \mathcal{E}(Q^{2,4}_{n,n+1}) > \mathcal{E}(B^{2,2}_{n,n+1})$.
(v) *For all* $n \geq 10$, $\mathcal{E}(B^{1,2}_{n,n+1}) < \mathcal{E}(Q^{3,1}_{n,n+1}) = \mathcal{E}(Q^{3,2}_{n,n+1}) < \mathcal{E}(Q^{3,3}_{n,n+1}) < \mathcal{E}(H^{3}_{n,n+1})$.

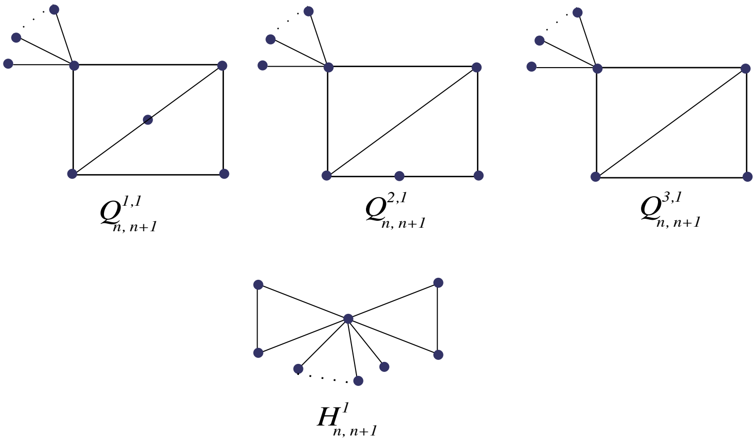


Figure 4: Signed graphs $Q^{r,1}_{n,n+1}$, $r = 1, 2, 3$ and $H^{1}_{n,n+1}$

(vi) *For all* $n \geq 10$, $\mathcal{E}(H^{3}_{n,n+1}) < \mathcal{E}(H^{1}_{n,n+1}) = \mathcal{E}(H^{2}_{n,n+1})$.
(vii) *For all* $n \geq 30$, $\mathcal{E}(H^{1}_{n,n+1}) = \mathcal{E}(H^{2}_{n,n+1}) < \mathcal{E}(B^{2,1}_{n,n+1})$.

Proof. (i). We have

$$\psi(B^{1,1}_{n,n+1}, x) = x^{n-4}\{x^4 - (n+1)x^2 + (4n-21)\},$$

$$\psi(Q^{1,1}_{n,n+1}, x) = x^{n-4}\{x^4 - (n+1)x^2 + (4n-20)\},$$

$$\psi(S^{2,1}_{n,n+1}, x) = x^{n-4}\{x^4 - (n+1)x^2 - 4x + (5n-20)\}.$$

It is easy to see that even and odd coefficients of signed graphs $B^{1,1}_{n,n+1}$, $Q^{1,1}_{n,n+1}$ and $S^{2,1}_{n,n+1}$ alternate in sign. Clearly, $B^{1,1}_{n,n+1} \prec Q^{1,1}_{n,n+1}$ and $Q^{1,1}_{n,n+1} \prec S^{2,1}_{n,n+1}$ for all $n \geq 10$. Therefore, $\mathcal{E}(B^{1,1}_{n,n+1}) < \mathcal{E}(Q^{1,1}_{n,n+1}) < \mathcal{E}(S^{2,1}_{n,n+1})$ for all $n \geq 10$.
(ii). We have

$$\psi(B^{2,2}_{n,n+1}, x) = x^{n-4}\{x^4 - (n+1)x^2 + (5n-21)\},$$

$$\psi(Q^{1,2}_{n,n+1}, x) = x^{n-6}\{x^6 - (n+1)x^4 + (4n-12)x^2 - (4n-20)\}.$$

The signed graphs $B^{2,2}_{n,n+1}$ and $Q^{1,2}_{n,n+1}$ are not quasi-order comparable. Therefore, consider the functions $\alpha_2(x) = x^6-(n+1)x^4+(4n-12)x^2-(4n-20)$ and $\beta_2(x) = x^4-(n+1)x^2+(5n-21)$. It is easy to see that $\beta_2(2) > 0$, $\beta_2(\sqrt{5}) < 0$, $\beta_2(\sqrt{n-4}) < 0$ and $\beta_2(\sqrt{n-3}) > 0$ for all $n \geq 10$. Also, $\alpha_2(\sqrt{2}) = 0$, $\alpha_2(1) < 0$, $\alpha_2(\frac{7071}{5000}) > 0$, $\alpha_2(\sqrt{n-3}) < 0$ and $\alpha_2(\sqrt{n-2}) > 0$ for all $n \geq 10$.

We observe that $\alpha_2(x) = \alpha_2(-x)$ and $\beta_2(x) = \beta_2(-x)$. Therefore $\alpha_2(x)$ has three positive and three negative zeros and $\beta_2(x)$ has two positive and two negative zeros. Recall that the energy of signed graph is twice the sum of its positive eigenvalues. Therefore, we have

$$\mathcal{E}(Q^{1,2}_{n,n+1}) > 2(\sqrt{2}+1+\sqrt{n-3}) > 2(\sqrt{5}+\sqrt{n-3}) > \mathcal{E}(B^{2,2}_{n,n+1})$$

for all $n \geq 10$, which proves part (ii).

(iii). We have

$$\psi(S^{2,3}_{n,n+1}, x) = x^{n-4}\{x^4 - (n+1)x^2 + (4n-16)\},$$

$$\psi(Q^{2,1}_{n,n+1}, x) = x^{n-4}\{x^4 - (n+1)x^2 - 2x + (4n-16)\},$$

$$\psi(Q^{2,3}_{n,n+1}, x) = x^{n-4}\{x^4 - (n+1)x^2 + 2x + (4n-16)\},$$

$$\psi(B^{1,2}_{n,n+1}, x) = x^{n-4}\{x^4 - (n+1)x^2 + (4n-13)\}.$$

Clearly, $S^{2,3}_{n,n+1} \prec Q^{2,1}_{n,n+1}$ and $Q^{2,1}_{n,n+1} \sim Q^{2,3}_{n,n+1}$ for all $n \geq 10$. Therefore, $\mathcal{E}(S^{2,3}_{n,n+1}) < \mathcal{E}(Q^{2,1}_{n,n+1}) = \mathcal{E}(Q^{2,3}_{n,n+1})$ for all $n \geq 10$. Thus, to prove the result, it is enough to show that $\mathcal{E}(Q^{2,1}_{n,n+1}) < \mathcal{E}(B^{1,2}_{n,n+1})$ for all $n \geq 10$. We see that even and odd coefficients of $Q^{2,1}_{n,n+1}$ and $B^{1,2}_{n,n+1}$ alternate in sign but the coefficients are not quasi-order comparable. We will compare the energy using Coulson's integral formula by directly solving the integrals. We have

$$\mathcal{E}(B^{1,2}_{n,n+1}) - \mathcal{E}(Q^{2,1}_{n,n+1}) = \frac{1}{\pi}\int_0^\infty \ln\frac{\{1+(n+1)x^2+(4n-13)x^4\}^2}{\{1+(n+1)x^2+(4n-16)x^4\}^2+4x^6}dx.$$

Put $\alpha_3(x) = \{1+(n+1)x^2+(4n-13)x^4\}^2$ and

$$\beta_3(x) = \{1+(n+1)x^2+(4n-16)x^4\}^2+4x^6,$$

we get $\alpha_3(x) - \beta_3(x) = 6x^4 + (6n+2)x^6 + (24n-87)x^8 > 0$ for $n \geq 10$ and $x > 0$. Thus, $\mathcal{E}(B^{1,2}_{n,n+1}) > \mathcal{E}(Q^{2,1}_{n,n+1})$.

(iv). We have

$$\psi(Q^{2,2}_{n,n+1}, x) = x^{n-6}\{x^6 - (n+1)x^4 + 2x^3 + (4n-12)x^2 - 4x - (4n-20)\},$$

$$\psi(Q^{2,4}_{n,n+1}, x) = x^{n-6}\{x^6 - (n+1)x^4 - 2x^3 + (4n-12)x^2 + 4x - (4n-20)\}.$$

Clearly, $Q^{2,2}_{n,n+1} \sim Q^{2,4}_{n,n+1}$ for all $n \geq 10$ and therefore $\mathcal{E}(Q^{2,2}_{n,n+1}) = \mathcal{E}(Q^{2,4}_{n,n+1})$ for all $n \geq 10$. Thus, to prove the result, it is enough to show that $\mathcal{E}(Q^{2,2}_{n,n+1}) > \mathcal{E}(B^{2,2}_{n,n+1})$ for all $n \geq 10$. The signed graphs $B^{2,2}_{n,n+1}$ and $Q^{2,2}_{n,n+1}$ are not quasi-order comparable. Consider the function $\alpha_4(x) = x^6 - (n+1)x^4 + 2x^3 + (4n-12)x^2 - 4x - (4n-20)$. It is easy to see that $\alpha_4(\sqrt{2}) = 0$, $\alpha_4(1) < 0$, $\alpha_4(\frac{7071}{5000}) > 0$, $\alpha_4(\sqrt{n-4}) < 0$ and $\alpha_4(\sqrt{n}) > 0$ for all $n \geq 10$. By Descartes' rule of signs, $\alpha_4(x)$ has three positive and three negative zeros. Therefore,

$$\mathcal{E}(Q^{2,2}_{n,n+1}) > 2(\sqrt{2} + 1 + \sqrt{n-4}) > 2(\sqrt{5} + \sqrt{n-3}) > \mathcal{E}(B^{2,2}_{n,n+1})$$

for all $n \geq 12$. We verified the result directly for $n = 10, 11$. This proves part (iv).

(v). We have

$$\psi(B^{1,2}_{n,n+1}, x) = x^{n-4}\{x^4 - (n+1)x^2 + (4n-13)\},$$

$$\psi(Q^{3,1}_{n,n+1}, x) = x^{n-5}\{x^5 - (n+1)x^3 - 4x^2 + (3n-12)x + 2(n-4)\},$$

$$\psi(Q^{3,2}_{n,n+1}, x) = x^{n-5}\{x^5 - (n+1)x^3 + 4x^2 + (3n-12)x - 2(n-4)\},$$

$$\psi(Q^{3,3}_{n,n+1}, x) = x^{n-5}\{x^5 - (n+1)x^3 + (3n-8)x - 2(n-4)\},$$

$$\psi(H^{3}_{n,n+1}, x) = x^{n-6}\{x^6 - (n+1)x^4 + (2n-5)x^2 - (n-5)\}.$$

First, we will show that $\mathcal{E}(Q^{3,1}_{n,n+1}) < \mathcal{E}(Q^{3,3}_{n,n+1})$. We see that even and odd coefficients of $Q^{3,1}_{n,n+1}$ and $Q^{3,3}_{n,n+1}$ alternate in sign but the coefficients are not quasi-order comparable. We will compare energy using Coulson's integral formula by directly solving the integrals, and we have $\mathcal{E}(Q^{3,3}_{n,n+1}) - \mathcal{E}(Q^{3,1}_{n,n+1})$

$$= \frac{1}{\pi}\int_0^\infty \ln \frac{\{1 + (n+1)x^2 + (3n-8)x^4\}^2 + \{(2n-8)x^5\}^2}{\{1 + (n+1)x^2 + (3n-12)x^4\}^2 + \{4x^3 + (2n-8)x^5\}^2} dx.$$

Put

$$\alpha_5(x) = \{1 + (n+1)x^2 + (3n-8)x^4\}^2 + \{(2n-8)x^5\}^2$$

and

$$\beta_5(x) = \{1 + (n+1)x^2 + (3n-12)x^4\}^2 + \{4x^3 + (2n-8)x^5\}^2,$$

we get $\alpha_5(x) - \beta_5(x) = 8x^4 + 8(n-1)x^6 + 8(n-2)x^8 > 0$ for $n \geq 10$ and $x > 0$. Thus, $\mathcal{E}(Q^{3,3}_{n,n+1}) > \mathcal{E}(Q^{3,1}_{n,n+1})$.

Next we will show that, $\mathcal{E}(Q^{3,3}_{n,n+1}) < \mathcal{E}(H^{3}_{n,n+1})$. The signed graphs $Q^{3,3}_{n,n+1}$ and $H^{3}_{n,n+1}$ are not quasi-order comparable. Therefore, consider the functions $\alpha_6(x) = x^5 - (n+1)x^3 + (3n-8)x - 2(n-4)$ and $\beta_6(x) = x^6 - (n+1)x^4 + (2n-5)x^2 - (n-5)$. It is easy to see that $\alpha_6(-2) = 0$, $\alpha_6(-\sqrt{n-3}) > 0$, $\alpha_6(-\sqrt{n-\frac{5}{2}}) < 0$ and $\beta_6(\frac{n-3}{n}) < 0$, $\beta_6(\frac{n-1}{n}) > 0$, $\beta_6(1) = 0$, $\beta_6(\sqrt{n-1}) < 0$ and $\beta_6(\sqrt{n}) > 0$. By Descartes' rule of signs, $\alpha_6(x)$ has three positive and two negative zeros and $\beta_6(x)$ has three positive and three negative zeros. As the energy of signed graph is twice the sum of its positive eigenvalues or -2 times the sum of negative eigenvalues, therefore,

$$\mathcal{E}(H^{3}_{n,n+1}) > 2(\frac{n-3}{n} + 1 + \sqrt{n-1}) > 2(2 + \sqrt{n-\frac{5}{2}}) > \mathcal{E}(Q^{3,3}_{n,n+1})$$

for all $n \geq 14$. We verified the result directly for $n = 10, 11, 13$.
Clearly, $Q^{3,1}_{n,n+1} \sim Q^{3,2}_{n,n+1}$ for all $n \geq 10$ and therefore $\mathcal{E}(Q^{3,1}_{n,n+1}) = \mathcal{E}(Q^{3,2}_{n,n+1})$ for all $n \geq 10$. Thus, to prove the result, it is enough to show that $\mathcal{E}(Q^{3,1}_{n,n+1}) > \mathcal{E}(B^{1,2}_{n,n+1})$ for all $n \geq 10$. The signed graphs $B^{1,2}_{n,n+1}$ and $Q^{3,1}_{n,n+1}$ are not quasi-order comparable, therefore consider the functions $\alpha_7(x) = x^5 - (n+1)x^3 - 4x^2 + (3n-12)x + 2(n-4)$ and $\beta_7(x) = x^4 - (n+1)x^2 + (4n-13)$ and proceeding similarly as above, we can prove that $\mathcal{E}(Q^{3,1}_{n,n+1}) > \mathcal{E}(B^{1,2}_{n,n+1})$ for all $n \geq 10$. This proves part (v).
(vi). We have

$$\psi(H^{1}_{n,n+1}, x) = x^{n-6}\{x^6 - (n+1)x^4 - 4x^3 + (2n-5)x^2 + 4x - (n-5)\},$$

$$\psi(H^{2}_{n,n+1}, x) = x^{n-6}\{x^6 - (n+1)x^4 + 4x^3 + (2n-5)x^2 - 4x - (n-5)\},$$

$$\psi(H^{3}_{n,n+1}, x) = x^{n-6}\{x^6 - (n+1)x^4 + (2n-5)x^2 - (n-5)\}.$$

It is clear that $H^{1}_{n,n+1} \sim H^{2}_{n,n+1}$ and $H^{1}_{n,n+1}, H^{2}_{n,n+1} \succ H^{3}_{n,n+1}$ for all $n \geq 10$. Therefore, $\mathcal{E}(H^{1}_{n,n+1}) = \mathcal{E}(H^{2}_{n,n+1}) > \mathcal{E}(H^{3}_{n,n+1})$ for all $n \geq 10$.
(vii). We have

$$\psi(B^{2,1}_{n,n+1}, x) = x^{n-4}\{x^4 - (n+1)x^2 + (5n-29)\}.$$

The signed graphs $H^{r}_{n,n+1}$ for $r = 1, 2$ and $B^{2,1}_{n,n+1}$ are not quasi-order comparable. Therefore consider the functions $\alpha_8(x) = x^6 - (n+1)x^4 - 4x^3 + (2n-5)x^2 + 4x - (n-5)$ and $\beta_8(x) = x^4 - (n+1)x^2 + (5n-29)$. Again, it is easy to see that $\alpha_8(-\sqrt{n}) > 0$ and $\alpha_8(-\sqrt{n-2}) < 0$ for all $n \geq 12$. Also, -1 is a zero of $\alpha_8(x)$ with multiplicity 2 and $\beta_8(\sqrt{\frac{9}{2}}) > 0$, $\beta_8(\sqrt{5}) < 0$, $\beta_8(\sqrt{n-3}) > 0$ and $\beta_8(\sqrt{n-4}) < 0$ for all $n \geq 22$. By Descartes' rule of signs, $\alpha_8(x)$ has three negative and three positive zeros and $\beta_8(x)$ has two positive and two negative zeros. Therefore,

$$\mathcal{E}(B^{2,1}_{n,n+1}) > 2(\sqrt{\frac{9}{2}} + \sqrt{n-4}) > 2(2+\sqrt{n}) > \mathcal{E}(H^{1}_{n,n+1})$$

for all $n \geq 275$. We can directly verify the result from $n = 30$ to 274. As $\mathcal{E}(H^1_{n,n+1}) = \mathcal{E}(H^2_{n,n+1})$, therefore $\mathcal{E}(H^1_{n,n+1}) = \mathcal{E}(H^2_{n,n+1}) < \mathcal{E}(B^{2,1}_{n,n+1})$ for all ≥ 30. □

Combining Lemmas 8 and 9, we have the following result.

Corollary 10 (i) *For all* $n \geq 30$*, we have*
$\mathcal{E}(S^{0,1}_{n,n+1}) = \mathcal{E}(S^{0,2}_{n,n+1}) < \mathcal{E}(S^{0,3}_{n,n+1}) < \mathcal{E}(B^{0,1}_{n,n+1}) < \mathcal{E}(S^{1,1}_{n,n+1}) = \mathcal{E}(S^{1,2}_{n,n+1}) < \mathcal{E}(S^{1,3}_{n,n+1})$
$< \mathcal{E}(B^{0,2}_{n,n+1}) < \mathcal{E}(B^{1,1}_{n,n+1}) < \mathcal{E}(Q^{1,1}_{n,n+1}) < \mathcal{E}(S^{2,1}_{n,n+1}) = \mathcal{E}(S^{2,2}_{n,n+1}) < \mathcal{E}(S^{2,3}_{n,n+1})$
$< \mathcal{E}(Q^{2,1}_{n,n+1}) = \mathcal{E}(Q^{2,3}_{n,n+1}) < \mathcal{E}(B^{1,2}_{n,n+1}) < \mathcal{E}(Q^{3,1}_{n,n+1}) = \mathcal{E}(Q^{3,2}_{n,n+1}) < \mathcal{E}(Q^{3,3}_{n,n+1})$
$< \mathcal{E}(H^{3}_{n,n+1}) < \mathcal{E}(H^{1}_{n,n+1}) = \mathcal{E}(H^{2}_{n,n+1}) < \mathcal{E}(B^{2,1}_{n,n+1}) < \mathcal{E}(S^{3,1}_{n,n+1}) = \mathcal{E}(S^{3,2}_{n,n+1})$
$< \mathcal{E}(S^{3,3}_{n,n+1}) < \mathcal{E}(B^{2,2}_{n,n+1})$.

(ii) *For all* $17 \leq n \leq 29$*, we have*
$\mathcal{E}(S^{0,1}_{n,n+1}) = \mathcal{E}(S^{0,2}_{n,n+1}) < \mathcal{E}(S^{0,3}_{n,n+1}) < \mathcal{E}(B^{0,1}_{n,n+1}) < \mathcal{E}(S^{1,1}_{n,n+1}) = \mathcal{E}(S^{1,2}_{n,n+1}) < \mathcal{E}(S^{1,3}_{n,n+1})$
$< \mathcal{E}(B^{0,2}_{n,n+1}) < \mathcal{E}(B^{1,1}_{n,n+1}) < \mathcal{E}(Q^{1,1}_{n,n+1}) < \mathcal{E}(S^{2,1}_{n,n+1}) = \mathcal{E}(S^{2,2}_{n,n+1}) < \mathcal{E}(S^{2,3}_{n,n+1}) < \mathcal{E}(Q^{2,1}_{n,n+1})$
$= \mathcal{E}(Q^{2,3}_{n,n+1}) < \mathcal{E}(B^{1,2}_{n,n+1}) < \mathcal{E}(B^{2,1}_{n,n+1}) < \mathcal{E}(S^{3,1}_{n,n+1}) = \mathcal{E}(S^{3,2}_{n,n+1}) < \mathcal{E}(S^{3,3}_{n,n+1}) < \mathcal{E}(B^{2,2}_{n,n+1})$.

The following lemma [4] will be useful in the sequel.

Lemma 11 *Let* S' *and* S'' *be two unicyclic signed graphs of order* $m_1, m_2 \geq 6$ *and let* $n = m_1 + m_2$. *Then, for* $t = 1, 2$,

$$\mathcal{E}(S' \cup S'') \geq \mathcal{E}(S^t_{m_1,m_1} \cup S^t_{m_2,m_2}) \geq \mathcal{E}(S^t_{n-6,n-6} \cup S^t_{6,6})$$

with equality if and only if $m_1, m_2 \in \{6, n-6\}$.

Now, we have the following theorem.

Theorem 12 *If* $S \in CC[n,p,q]$*, with* $n \geq 12$ *and* $p,q \geq 3$*, then* $\mathcal{E}(S) > \mathcal{E}(B^{2,2}_{n,n+1})$.

Proof. As $S \in CC[n,p,q]$, with $n \geq 12$ and $p,q \geq 3$, therefore S has a cut-edge say e, such that $S-\{e\}$ is disconnected with two components, which are unicyclic signed graphs, say S' and S''. Let m_1 and m_2 respectively be the number of vertices in S' and S''. Without loss of generality, we assume that $m_1 \geq m_2$. The following cases arise. (i) $m_1, m_2 \geq 6$, (ii) $m_1 \geq 7$ and $m_2 \geq 5$,(iii) $m_1 \geq 8$ and $m_2 \geq 4$,(iv) $m_1 \geq 9$ and $m_2 \geq 3$.
Case (i). $m_1, m_2 \geq 6$. By Lemmas 5, 6 and 11, we have

$$\begin{aligned}\mathcal{E}(S) > \mathcal{E}(S-e) &= \mathcal{E}(S' \cup S'') = \mathcal{E}(S') + \mathcal{E}(S'')\\ &\geq \mathcal{E}(S^t_{m_1,m_1}) + \mathcal{E}(S^t_{m_2,m_2}) = \mathcal{E}(S^t_{m_1,m_1} \cup S^t_{m_2,m_2})\\ &\geq \mathcal{E}(S^t_{n-6,n-6} \cup S^t_{6,6}) = \mathcal{E}(S^1_{n-6,n-6} \cup S^1_{6,6}).\end{aligned}$$

We see that $\mathcal{E}(S^1_{6,6}) > 6$. Consider the functions, $\alpha_9(x) = x^4 - (n-6)x^2 - 2x + (n-9)$ and $\beta_9(x) = x^4 - (n+1)x^2 + (5n-21)$. It is easy to see that $\alpha_9(\frac{1}{2}) > 0$, $\alpha_9(1) < 0$, $\alpha_9(\sqrt{n-7}) < 0$ and $\alpha_9(\sqrt{n-4}) > 0$. Similarly, $\beta_9(2) > 0$, $\beta_9(\sqrt{5}) < 0$, $\beta_9(\sqrt{n-4}) < 0$ and $\beta_9(\sqrt{n-3}) > 0$. By Descartes' rule of signs, both $\alpha_9(x)$ and $\beta_9(x)$ have two positive and two negative zeros. Let the positive zeros of $\alpha_9(x)$ and $\beta_9(x)$ be x_1, x_2 and y_1, y_2, respectively. Therefore, we have $\mathcal{E}(S^1_{n-6,n-6} \cup S^1_{6,6}.) > 2(x_1+x_2)+6 > 2(\frac{1}{2}+\sqrt{n-7})+6 = 7+2\sqrt{n-7} > 2(\sqrt{5}+\sqrt{n-3}) > 2(y_1+y_2) = \mathcal{E}(B^{2,2}_{n,n+1})$ for all $n \geq 12$. This completes the proof of case(i).
Case (ii). Proceeding similarly as in case (i), we can prove that $\mathcal{E}(S) > \mathcal{E}(S^1_{n-5,n-5}) + \mathcal{E}(S)$, where S is the signed graph shown in Figure 5. Note that $\mathcal{E}(S) > 5.5$. Therefore, we have
$\mathcal{E}(S) > \mathcal{E}(S^1_{n-5,n-5}) + \mathcal{E}(S) > 2(\frac{1}{2} + \sqrt{n-6}) + 5.5 = 6.5 + 2\sqrt{n-6} > 2(\sqrt{5}+\sqrt{n-3}) > \mathcal{E}(B^{2,2}_{n,n+1})$ for all ≥ 12.
Case (iii). Again, for $n = 12$, we proved the result directly. For $n \geq 13$, we have

$$\begin{aligned}\mathcal{E}(S) > \mathcal{E}(S^1_{n-4,n-4}) + \mathcal{E}(C^+_4) > 2(\frac{1}{2} + \sqrt{n-5}) + 4 = 5 + 2\sqrt{n-5}\\ > 2(\sqrt{5}+\sqrt{n-3}) > \mathcal{E}(B^{2,2}_{n,n+1}).\end{aligned}$$

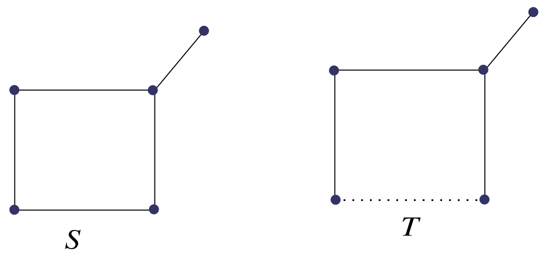


Figure 5: Signed graphs S and T

Case (iv). Finally, we have

$$\begin{aligned}\mathcal{E}(S) &> \mathcal{E}(S^1_{n-3,n-3}) + \mathcal{E}(C_3) > 2(\frac{1}{2} + \sqrt{n-4}) + 4 = 5 + 2\sqrt{n-4} \\ &> 2(\sqrt{5} + \sqrt{n-3}) > \mathcal{E}(B^{2,2}_{n,n+1}),\end{aligned}$$

for all $n \geq 12$. This completes the proof. □

Recall that $\infty(n,p,q)$ is the class of bicyclic signed graphs on n vertices, which have exactly two edge-disjoint cycles sharing a common vertex, which we call the meet vertex. According to the sign and order of cycles in $\infty(n,p,q)$, we divide the class $\infty(n,p,q)$ into three main subclasses.

Subclass 1. According to the sign of C_p(p is even) and C_q(q is odd).This class is further divided into four subclasses:
(1.1) $\infty^{11}(n,p,q) : \sigma(C_p) = \sigma(C_q) = +$; (1.2) $\infty^{12}(n,p,q) : \sigma(C_p) = +$ and $\sigma(C_q) = -$;
(1.3) $\infty^{13}(n,p,q) : \sigma(C_p) = -$ and $\sigma(C_q) = +$; (1.4) $\infty^{14}(n,p,q) : \sigma(C_p) = \sigma(C_q) = -$;
Subclass 2. According to the sign of C_p(p is odd) and C_q(q is odd). Also, this class is further divided into four subclasses:
(2.1) $\infty^{21}(n,p,q) : \sigma(C_p) = \sigma(C_q) = +$; (2.2) $\infty^{22}(n,p,q) : \sigma(C_p) = +$ and $\sigma(C_q) = -$;
(2.3) $\infty^{23}(n,p,q) : \sigma(C_p) = -$ and $\sigma(C_q) = +$; (2.4) $\infty^{24}(n,p,q) : \sigma(C_p) = \sigma(C_q) = -$;
Subclass 3. According to the sign of C_p(p is even) and C_q(q is even). This class is further divided into four subclasses:
(3.1) $\infty^{31}(n,p,q) : \sigma(C_p) = \sigma(C_q) = +$; (3.2) $\infty^{32}(n,p,q) : \sigma(C_p) = +$ and $\sigma(C_q) = -$;

(3.3) $\infty^{33}(n,p,q) : \sigma(C_p) = -$ and $\sigma(C_q) = +$; (3.4) $\infty^{34}(n,p,q) : \sigma(C_p) = \sigma(C_q) = -$;

For an underlying graph G and the corresponding signed graph $S_{ij} = (G, \sigma_{ij}) \in \infty^{ij}(n,p,q)$ $(i = 1,2,3$ and $j = 1,2,3,4)$, we can easily obtain $\lambda_k(A_{\sigma_{1.1}}) = -\lambda_k(A_{\sigma_{1.2}})$, $\lambda_k(A_{\sigma_{1.3}}) = -\lambda_k(A_{\sigma_{1.4}})$, $\lambda_k(A_{\sigma_{2.1}}) = -\lambda_k(A_{\sigma_{2.4}})$ and $\lambda_k(A_{\sigma_{2.2}}) = -\lambda_k(A_{\sigma_{2.3}})$ for $k = 1,2,\dots,n$. So $\mathcal{E}(G,\sigma_{1.1}) = \mathcal{E}(G,\sigma_{1.2})$, $\mathcal{E}(G,\sigma_{1.3}) = \mathcal{E}(G,\sigma_{1.4})$, $\mathcal{E}(G,\sigma_{2.1}) = \mathcal{E}(G,\sigma_{2.4})$ and $\mathcal{E}(G,\sigma_{2.2}) = \mathcal{E}(G,\sigma_{2.3})$. Thus we can regard (1.1) and (1.2) as identical, (1.3) and (1.4) as identical, (2.1) and (2.4) as identical and (2.2) and (2.3) as identical. Let $\infty_*(n,p,q)$ denote the collection of signed graphs in $\infty(n,p,q)$ having all $(n-p-q+1)$ pendent vertices adjacent to the meet vertex in $\infty_*(n,p,q)$. Let $\infty_*^{ij}(n,p,q)$ $(i = 1,2,3$ and $j = 1,2,3,4)$ be the corresponding switching class, as shown in Figure 6, in $\infty_*(n,p,q)$, where p and q are not equal to 3 simultaneously, that is according to sign and order of cycles as defined above in subclasses. Also let $\infty_2^*(n,3,3), \infty_2^{**}(n,3,3) \in \infty(n,3,3)$ be signed graphs having both cycles of length 3, $(n-6)$ pendent vertices are adjacent to meet vertex, remaining one pendent vertex is adjacent to any vertex of either cycle other than the meet vertex in $\infty_2^*(n,3,3)$ and $(n{-}5)$ pendent vertices are adjacent to a single vertex in either of the cycles other than the meet vertex in $\infty_2^{**}(n,3,3)$, respectively. We use $\infty_{2r}^*(n,3,3)$,$\infty_{2r}^{**}(n,3,3)$ $r = 1,2,3,4$ as the representative of these switching classes, as shown in Figure 7, in $\infty_2^*(n,3,3)$ and $\infty_2^{**}(n,3,3)$, respectively corresponding to subclass 2. We know that the necessary condition to use quasi-order method is that the coefficients of the characteristic polynomials of signed graphs must have uniform sign. We next have the following result.

Lemma 13 *(i) If* $S \in \infty^{1j}(n,p,q)$ $(j = 1,3)$, *contains an even cycle* C_p *and an odd cycle* C_q, $q = 2t+1$, *then, for all* $i \geq 0$, *we have*
(a) $(-1)^i a_{2i}(S) \geq 0$, (b) $(-1)^i a_{2i+1}(S) \geq 0$ $(\text{resp.} \leq 0)$ *if* t *is odd (resp,even)*
(ii) *If* $S \in \infty^{3j}(n,p,q)$ $(j = 1,2,3,4)$, *containing both even cycles, then for all* $i \geq 0$, *we have*
(a) $(-1)^i a_{2i}(S) \geq 0$, (b) $(-1)^i a_{2i+1}(S) = 0$
(iii) (a) *If* $S \in \infty^{2j}(n,p,q)$ $(j = 1,2,3,4)$, *containing both odd cycles,then for all* $i \geq 0$, *we have* $(-1)^i a_{2i}(S) \geq 0$
(b) *If* $S \in \infty^{2j}(n,p,p)$ $(j = 1,2)$, *containing both odd cycles of equal length* $p = 2t+1$,*then for all* $i \geq 0$, *we have* $(-1)^i a_{2i+1}(S) \geq 0$ $(\text{or} \leq 0)$.

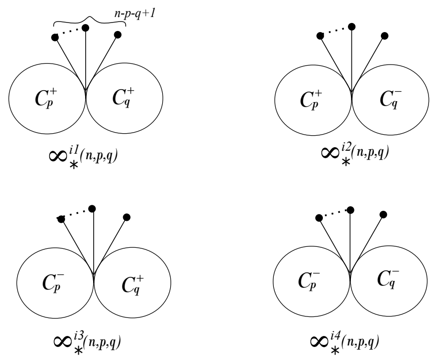


Figure 6: Switching classes corresponding to $\infty_*(n,p,q)$

Proof. (i). If $S \in \infty^{11}(n,p,q)$, then the proof follows from Lemma 1.8 in [8] and if $S \in \infty^{13}(n,p,q)$, then the proof follows from Lemma 4.3 in [12] .
(ii). If $S \in \infty^{3j}(n,p,q)$ $(j = 1,2,3,4)$,then the proof follows from Theorem 2.1 in [3]
(iii). Let $\mathscr{L}_{2i}$, $\mathscr{L}_{2i+1}^{(1)}$ and $\mathscr{L}_{2i+1}^{(2)}$ denote the basic figures of $S \in \infty^{2j}(n,p,q)$ $(j = 1,2,3,4)$ containing only edges, an odd cycle C_p and an odd cycle C_q, respectively. Then

(a). Since $S \in \infty^{2j}(n,p,q)$ $(j = 1,2,3,4)$, therefore the odd cycles share a common vertex in S and hence the basic figure on even vertices does not contain any odd cycle. Therefore, from Theorem 1, we have

$$\begin{aligned}(-1)^i a_{2i}(S) &= (-1)^i\Bigg(\sum_{L\in\mathscr{L}_{2i}} (-1)^{P(L)} 2^{|c(L)|} \prod_{X\in c(L)} \sigma(X)\Bigg)\\ &= (-1)^i\Bigg(\sum_{L\in\mathscr{L}_{2i}} (-1)^i\Bigg) = m(S,i).\end{aligned}$$

Thus, $(-1)^i a_{2i}(S) = m(S,i) \geq 0$ for all i. This proves part (a).
(b). There are two cases to be executed as follows.
Case 1. If $S \in \infty^{21}(n,p,p)$, that is, containing both positive cycles of equal odd lengths $p = 2t+1$. If $2i+1 < p = 2t+1$, then $(-1)^i a_{2i+1}(S) = 0$ and if

$2i+1 \geq p = 2t+1$, then

$$\begin{aligned}(-1)^i a_{2i+1}(S) &= (-1)^i\Big(2\sum_{L\in\mathscr{L}^{(1)}_{2i+1}} (-1)^{\frac{2i+1-(2t+1)}{2}+1} + 2\sum_{L\in\mathscr{L}^{(2)}_{2i+1}} (-1)^{\frac{2i+1-(2t+1)}{2}+1}\Big)\\ &= \Big(2\sum_{L\in\mathscr{L}^{(1)}_{2i+1}} (-1)^{-t+1} + 2\sum_{L\in\mathscr{L}^{(2)}_{2i+1}} (-1)^{-t+1}\Big)\end{aligned}$$

Thus $(-1)^i a_{2i+1}(S) \geq 0$ if $t = 2k+1$, and $(-1)^i a_{2i+1}(S) \leq 0$ if $t = 2k$ for all i.

Case 2. $S \in \infty^{22}(n,p,p)$, that is, containing one positive cycle and one negative cycle of equal odd lengths $p = 2t+1$, respectively. If $2i+1 < p = 2t+1$, then $(-1)^i a_{2i+1}(S) = 0$ and if $2i+1 \geq p = 2t+1$, then

$$\begin{aligned}(-1)^i a_{2i+1}(S) &= (-1)^i\Big(2\sum_{L\in\mathscr{L}^{(1)}_{2i+1}} (-1)^{\frac{2i+1-(2t+1)}{2}+1} - 2\sum_{L\in\mathscr{L}^{(2)}_{2i+1}} (-1)^{\frac{2i+1-(2t+1)}{2}+1}\Big)\\ &= \Big(2\sum_{L\in\mathscr{L}^{(1)}_{2i+1}} (-1)^{-t+1} - 2\sum_{L\in\mathscr{L}^{(2)}_{2i+1}} (-1)^{-t+1}\Big).\end{aligned}$$

Thus, $(-1)^i a_{2i+1}(S) \geq 0$ if $t = 2k+1$ and $|\mathscr{L}^{(1)}_{2i+1}| \geq |\mathscr{L}^{(2)}_{2i+1}|$; $(-1)^i a_{2i+1}(S) \leq 0$ if $t = 2k+1$ and $|\mathscr{L}^{(1)}_{2i+1}| \leq |\mathscr{L}^{(2)}_{2i+1}|$; $(-1)^i a_{2i+1}(S) \geq 0$ if $t = 2k$ and $|\mathscr{L}^{(1)}_{2i+1}| \leq |\mathscr{L}^{(2)}_{2i+1}|$; and $(-1)^i a_{2i+1}(S) \leq 0$ if $t = 2k$ and $|\mathscr{L}^{(1)}_{2i+1}| \geq |\mathscr{L}^{(2)}_{2i+1}|$ for all i, where $|Z|$ denotes the cardinality of a set Z. This completes the proof. □

The following two lemmas can be easily established.

Lemma 14 *For positive integers* $m_1, m_2 \geq 5$, $m_1+m_2 = n \geq 17$ *and* $t = 1,2$,

$$\mathcal{E}(S_{m_1} \cup S^t_{m_2,m_2}) \geq \mathcal{E}(S_5 \cup S^t_{n-5,n-5}),$$

with equality if and only if $m_1, m_2 \in \{5, n-5\}$.

Lemma 15 (i) $\mathcal{E}(S_{n-4} \cup S^1_{4,4}) > \mathcal{E}(B^{2,2}_{n,n+1})$ *for all* $n \geq 12$.
(ii) $\mathcal{E}(S_{n-4} \cup C^-_4) > \mathcal{E}(B^{2,2}_{n,n+1})$ *for all* $n \geq 12$.
(iii) $\mathcal{E}(S_5 \cup S^1_{n-5,n-5}) > \mathcal{E}(B^{2,2}_{n,n+1})$ *for all* $n \geq 12$.
(iv) $\mathcal{E}(S_{n-5} \cup S) > \mathcal{E}(B^{2,2}_{n,n+1})$ *for all* $n \geq 12$, *where* S *is the signed graph shown in Figure* 5.
(v) $\mathcal{E}(S_{n-5} \cup T) > \mathcal{E}(B^{2,2}_{n,n+1})$ *for all* $n \geq 12$, *where* T *is the signed graph shown in Figure* 5.

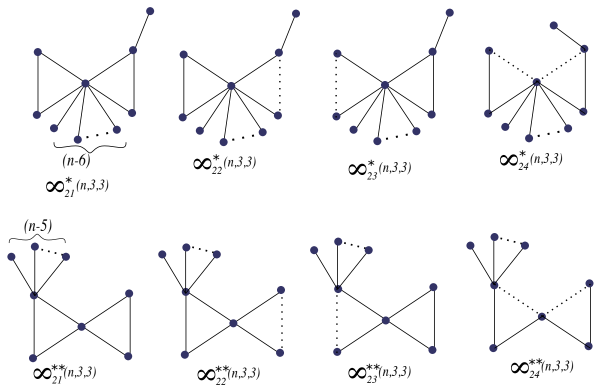


Figure 7: Switching classes $\infty^{*}_{2r}(n,3,3)$ and $\infty^{**}_{2r}(n,3,3)$, $r = 1,2,3,4$

Now, as the proof of the following result is similar as in Lemma 9. So we skip the proof here.

Lemma 16 (i) *For all* $n \geq 12$, $\mathcal{E}[\infty^{*}_{21}(n,3,3)] > \mathcal{E}[\infty^{*}_{22}(n,3,3)] > \mathcal{E}(B^{2,2}_{n,n+1})$.

(ii) *For all* $n \geq 12$, $\mathcal{E}[\infty^{**}_{21}(n,3,3)] > \mathcal{E}[\infty^{**}_{22}(n,3,3)] > \mathcal{E}(B^{2,2}_{n,n+1})$.

(iii) *For all* $n \geq 12$, $\mathcal{E}[\infty^{13}_{*}(n,4,3)] > \mathcal{E}[\infty^{11}_{*}(n,4,3)] > \mathcal{E}(B^{2,2}_{n,n+1})$.

(iv) *For all* $n \geq 12$, $\mathcal{E}[\infty^{34}_{*}(n,4,4)] > \mathcal{E}[\infty^{32}_{*}(n,4,4)] > \mathcal{E}[\infty^{31}_{*}(n,4,4)] > \mathcal{E}(B^{2,2}_{n,n+1})$.

(v) *For all* $n \geq 12$, $\mathcal{E}[\infty^{22}_{*}(n,5,3)] > \mathcal{E}[\infty^{21}_{*}(n,5,3)] > \mathcal{E}[\infty^{*}_{22}(n,3,3)]$.

Now, we proceed to prove the following theorem.

Theorem 17 *If* $S \in \infty(n,p,q)$, $n \geq 17$, $p,q \geq 3$ *and* $S \neq H^{r}_{n,n+1}$ $(r = 1,2,3)$, *then* $\mathcal{E}(S) > \mathcal{E}(B^{2,2}_{n,n+1})$.

Proof. Let v be the meet vertex of the two cycles C_p and C_q. The following cases arise.
Case 1. Let $S \in \infty(n,3,3)$. We have the following claim.
Claim. If $S \in \infty^{2r}(n,3,3)$ $(r = 1,2)$ and $S \neq H^{s}_{n,n+1}(s = 1,3)$, $\infty^{**}_{2t}(n,3,3)(t = 1,2)$, $\infty^{*}_{2r}(n,3,3)(r = 1,2)$, then for all $n \geq 7$, $S \succ \infty^{*}_{22}(n,3,3)$.

We shall prove the claim by induction on n. Assume that the claim holds for smaller values of n. Let v_1 be a pendent vertex which is adjacent to the meet vertex v in $\infty^*_{22}(n,3,3)$. For $n \geq 7$, $S \in \infty^{2r}(n,3,3)$, $r = 1,2$, has at least one pendent vertex say v_2 (such that $S - v_2$ is again different from signed graphs forbidden in this claim), which is adjacent to u (say) in S. Then from Lemmas 4 and 13, we obtain

$$b_i(S) = b_i(S - v_2) + b_{i-2}(S - v_2 - u)$$

and

$$b_i(\infty^*_{22}(n,3,3)) = b_i(\infty^*_{22}(n,3,3) - v_1) + b_{i-2}(P_3 \cup P_2).$$

By induction assumption, $S - v_2 \succ \infty^*_{22}(n,3,3) - v_1$.
Since $S \neq H^s_{n,n+1}(s = 1,3), \infty^{**}_{2t}(n,3,3)(t = 1,2)$, $\infty^*_{2r}(n,3,3)$ $(r = 1,2)$, therefore $S - v_2 - u$ has $P_3 \cup P_2$ as a subgraph and hence we have $S - v_2 - u \succeq P_3 \cup P_2$. This proves the claim.
By Lemma 13, the even and odd coefficients of S alternate in sign. Thus, by the above claim, for $S \neq H^s_{n,n+1}(s = 1,3), \infty^{**}_{2t}(n,3,3)(t = 1,2)$, $\infty^*_{2r}(n,3,3)$ $(r = 1,2)$, we have $\mathcal{E}(S) > \mathcal{E}[\infty^*_{22}(n,3,3)]$. Also for the same underlined graph , the energy of signed graphs $S \in \infty^{21}(n,3,3)$ and $T \in \infty^{24}(n,3,3)$ is same, $S \in \infty^{22}(n,3,3)$ and $T \in \infty^{23}(n,3,3)$ is same and therefore the result follows by Lemma 16 in this case.
Case 2. Let $S \in \infty(n,4,3)$. We have the following claim.
Claim. If $S \in \infty^{1r}(n,4,3)$ $(r = 1,3)$ and $S \neq \infty^{1r}_*(n,4,3)$ $(r = 1,3)$, then for all $n \geq 7$, $S \succ \infty^{11}_*(n,4,3)$.
We shall prove the claim by induction on n. Assume that the claim holds for smaller values of n. Let v_1 be a pendent vertex which is adjacent to the meet vertex v in $\infty^{11}_*(n,4,3)$. For $n \geq 7$, $S \in \infty^{1r}(n,4,3)$, $r = 1,2$, has at least one pendent vertex say v_2, which is adjacent to u (say) in S. Then from Lemmas 4 and 13, we obtain

$$b_i(S) = b_i(S - v_2) + b_{i-2}(S - v_2 - u).$$

and

$$b_i(\infty^{11}_*(n,4,3)) = b_i(\infty^{11}_*(n,4,3) - v_1) + b_{i-2}(P_3 \cup P_2).$$

By induction assumption, $S - v_2 \succ \infty^{11}_*(n,4,3) - v_1$. Since $S \neq \infty^{1r}_*(n,4,3)$ $(r = 1,3)$ and therefore $S - v_2 - u$ has $P_3 \cup P_2$ as a subgraph and thus $S - v_2 - u \succeq$

$P_3 \cup P_2$. This proves the claim.
By Lemma 13, the even and odd coefficients of S alternate in sign. Thus, by the above claim, we have $\mathcal{E}(S) > \mathcal{E}[\infty_*^{11}(n,3,3)]$. Also, for the same underlined graph , the energy of signed graphs $S \in \infty^{11}(n,4,3)$ and $T \in \infty^{12}(n,4,3)$ is same, $S \in \infty^{13}(n,4,3)$ and $T \in \infty^{14}(n,4,3)$ is same and therefore the result follows by Lemma 16 in this case.
Case 3. If $S \in \infty(n,4,4)$ and $S \neq \infty_*^{3r}(n,4,4)(r = 1,2,4)$, then proceeding similarly as in case 2, one can easily prove that, $\mathcal{E}(S) > \mathcal{E}[\infty_*^{31}(n,4,4)]$ and hence the result follows by Lemma 16.
Case 4. If $S \in \infty(n,5,3), \infty(n,5,4), \infty(n,5,5)$ and $S \neq \infty_*^{2r}(n,5,3), r = 1,2,3,4$ then it easy to see that $b_4(S) > b_4(B^{2,2}_{n,n+1}) = 5n - 21$. Since by Lemma 13 even coefficients S alternate in sign and therefore $\mathcal{E}(S) > \mathcal{E}(B^{2,2}_{n,n+1})$, and so the result follows in this case. If $S = \infty_*^{2r}(n,5,3), r = 1,2,3,4$, then the result follows by Lemma 16, since for the same underlined graph, the energy of signed graphs $S \in \infty^{21}(n,5,3)$ and $T \in \infty^{24}(n,5,3)$ is same, $S \in \infty^{22}(n,5,3)$ and $T \in \infty^{23}(n,5,3)$ is same.
Case 5. Let $S \in \infty(n,p,q)$ and at least one of p and q is greater or equal to 6. Without loss of generality, we assume C_p is such a cycle. Then the following four subcases arise.
Subcase 5.1. Let $C_q \neq C_3^+, C_3^-, C_4^+, C_4^-$, if there be at most a single noncyclic signed edge incident to any vertex of C_3^+ or C_3^- and no noncyclic signed edge incident to the vertices of C_4^+, C_4^-. Then choose the cut set $Z = \{e_1, e_2\}$, where e_1 and e_2 are the edges on the cycle C_p adjacent to v. Then $S - Z$ has two components, say S' and S'', where S' is a signed tree on $m_1 \geq 5$ vertices and S'' is unicyclic signed graph with $m_2 \geq 5$ vertices such that $m_1 + m_2 = n \geq 17$. Then the result follows by Lemmas 14 and 15.
Subcase 5.2. If $C_q = C_4^-, C_3^+, C_3^-$ such that there is exactly single noncyclic signed edge incident to any vertex of C_3^+, C_3^- and there is no noncyclic signed edge incident to any vertex of C_4^-, then choose the cut set $Z = \{e_1, e_2\}$, where e_1 and e_2 are the edges on the cycle C_p adjacent to v. Then $S - Z$ has two components, say S' and S'', where S' is a signed tree on $n-4$ vertices and S'' is unicyclic signed graph with 4 vertices such that $m_1 + m_2 = n \geq 17$. Since S'' is either $S^t_{4,4}, t = 1,2$ or C_4^- and $\mathcal{E}(S^1_{4,4}) = \mathcal{E}(S^2_{4,4})$, then the result follows by Lemma 15.
Subcase 5.3. Let $C_q = C_4^+$ and there be no noncyclic signed edge incident to

any vertex of C_4^+. Let $\{e_1, e_2, \ldots, e_{p-1}, e_p\}$ be the edges of cycle C_p. Without loss of generality, suppose that the edges e_1 and e_p are incident to meet vertex v such that the edges e_r and e_s are adjacent in C_p if $|r-s| = 1$ and e_1 , e_p are incident to meet vertex v. Then choose the cut set either $\{e_1, e_{p-1}\}$ or $\{e_2, e_p\}$ such that $S - \{e_1, e_{p-1}\}$ or $S - \{e_2, e_p\}$ has two components, say S' and S'', where S' is a signed tree on $m_1 \geq 5$ vertices and S'' is unicyclic signed graph with $m_2 \geq 5$ vertices such that $m_1 + m_2 = n \geq 17$. Then the result follows by Lemmas 14 and 15.

Subcase 5.4. Let $C_q = C_3^+, C_3^-$ and there is no noncyclic signed edge incident to any vertex of C_3^+, C_3^-. Then there exists a cut set Z consisting of the two edges of C_p such that $S - Z$ has two components, say S' and S'', where S' is a signed tree on $m_1 \geq 5$ vertices and S'' is unicyclic signed graph with $m_2 \geq 5$ vertices such that $m_1 + m_2 = n \geq 17$. Then the result follows by Lemmas 14 and 15. This completes the proof. □

Let L_q^- and L_q^+ respectively denote the number of negative and positive 4-cycles in a signed graph. Then, by Theorem 1, we have

$$b_4(S) = m(S,2) - 2(L_q^+ - L_q^-). \tag{1}$$

Where $m(S,k)$ denote the number of matchings of size k. Let $\mathscr{L}_k$ denote the set of basic figures of order k of a signed graph S and let $\mathscr{L}_k^1$ denote the set of basic figures which do not contain any cycle and $\mathscr{L}_k^2 = \mathscr{L}_k - \mathscr{L}_k^1$. It is clear that for a signed graph $S \in \theta(n,p,q,r)$, $\left|\mathscr{L}_{2k}^1\right| \geq 2\left|\mathscr{L}_{2k}^2\right|$. From this, we can easily see that if $S \in \theta(n,p,q,r)$, then $b_{2k}(S) \geq 0$ for all $k \geq 0$. Also, if $b_4(S) > 5n - 21$, then it is easy to see that $\mathcal{E}(S) > \mathcal{E}(B_{n,n+1}^{2,2})$. Let $Q_k^{1,1}$, $k = 1,2,3,...,52$ be the graphs as shown in Figure 8. Then it is easy to see that there are three switching classes on the signings of $Q_k^{1,1}$ $k = 1,2,3,\ldots,52$. Let $Q_k^{1,1}$, $Q_k^{1,2}$ and $Q_k^{1,3}$ $k = 1,2,3,\ldots,52$, respectively be the representative for these three switching classes, where $Q_k^{1,2}$ are the signed graphs obtained from $Q_k^{1,1}$, by making both triangles negative in $Q_k^{1,1}$ and $Q_k^{1,3}$ be the signed graphs obtained from $Q_k^{1,1}$ by making one triangle negative(left sided triangle) and other triangle positive(right sided triangle) in $Q_k^{1,1}$. It is easy to see that $Q_k^{1,2}$ is switching equivalent to $-Q_k^{1,1}$ and therefore $\mathcal{E}(Q_k^{1,1}) = \mathcal{E}(Q_k^{1,2})$ for all $k = 1,2,3,\ldots,52$. Thus, we can regard $Q_k^{1,1}$ and $Q_k^{1,2}$ as identical. Also, $b_4(Q_k^{1,3}) > 5n - 21$ for all $k = 15,16,19,20,21,22,23,24,25,26,27,28,29,31,32,\ldots,52$ and therefore we omit these signed graphs here, as these signed graphs will be considered later (Theorem 22). With these notations, we have the following observation.

Lemma 18 *For all* $n \geq 10$*, we have*
(i) $\mathcal{E}(B^{2,2}_{n,n+1}) < \mathcal{E}(Q^{1,1}_1) < \mathcal{E}(Q^{1,1}_k)$ *for all* $k = 2, 3, \ldots, 52$.
(ii) $\mathcal{E}(B^{2,2}_{n,n+1}) < \mathcal{E}(Q^{1,3}_1) < \mathcal{E}(Q^{1,3}_k)$ *for all* $k = 3, 4, 6, 8, 13, 14, 17, 39, 40, \ldots, 44$.
(iii) $\mathcal{E}(B^{2,2}_{n,n+1}) < \mathcal{E}(Q^{1,3}_2) < \mathcal{E}(Q^{1,3}_k)$ *for all* $k = 5, 9, 10, 11, 12, 18, 30$.
(iv) $\mathcal{E}(B^{2,2}_{n,n+1}) < \mathcal{E}(Q^{1,3}_7)$.

Proof. (i). We have

$$\psi(Q^{1,1}_1, x) = x^{n-6}\{x^6 - (n+1)x^4 - 4x^3 + (3n-12)x^2 + 2x - (n-5)\},$$

$$\psi(Q^{1,1}_2, x) = x^{n-6}\{x^6 - (n+1)x^4 - 4x^3 + (3n-11)x^2 + 4x - 2(n-6)\},$$

$$\psi(Q^{1,1}_3, x) = x^{n-6}\{x^6 - (n+1)x^4 - 4x^3 + (4n-18)x^2 + 4x - 2(n-6)\},$$

$$\psi(Q^{1,1}_4, x) = x^{n-6}\{x^6 - (n+1)x^4 - 4x^3 + (4n-18)x^2 + 2x - (2n-11)\},$$

$$\psi(Q^{1,1}_5, x) = x^{n-6}\{x^6 - (n+1)x^4 - 4x^3 + (4n-17)x^2 + 4x - (3n-17)\},$$

$$\psi(Q^{1,1}_6, x) = x^{n-8}\{x^8 - (n+1)x^6 - 4x^5 + (4n-16)x^4 + 6x^3 - (4n-24)x^2 - 2x + (n-7)\},$$

$$\psi(Q^{1,1}_7, x) = x^{n-6}\{x^6 - (n+1)x^4 - 4x^3 + (4n-17)x^2 + 4x - (2n-14)\},$$

$$\psi(Q^{1,1}_8, x) = x^{n-6}\{x^6 - (n+1)x^4 - 4x^3 + (4n-16)x^2 + 6x - 3(n-6)\},$$

$$\psi(Q^{1,1}_9, x) = x^{n-6}\{x^6 - (n+1)x^4 - 4x^3 + (4n-21)x^2 + 4x - (3n-22)\},$$

$$\psi(Q^{1,1}_{10}, x) = x^{n-6}\{x^6 - (n+1)x^4 - 4x^3 + (4n-19)x^2 + 8x - (4n-30)\},$$

$$\psi(Q^{1,1}_{11}, x) = x^{n-8}\{x^8 - (n+1)x^6 - 4x^5 + (4n-15)x^4 + 8x^3 - (5n-31)x^2 - 4x + 2(n-8)\},$$

$$\psi(Q^{1,1}_{12}, x) = x^{n-6}\{x^6 - (n+1)x^4 - 4x^3 + (4n-16)x^2 + 8x - 4(n-7)\},$$

$$\psi(Q^{1,1}_{13}, x) = x^{n-6}\{x^6 - (n+1)x^4 - 4x^3 + (5n-26)x^2 + 6x - 3(n-7)\},$$

$$\psi(Q^{1,1}_{14}, x) = x^{n-6}\{x^6 - (n+1)x^4 - 4x^3 + (5n-25)x^2 + 4x - (4n-26)\},$$

$$\psi(Q^{1,1}_{15}, x) = x^{n-6}\{x^6 - (n+1)x^4 - 4x^3 + (5n-24)x^2 + 6x - (5n-33)\},$$

$$\psi(Q^{1,1}_{16}, x) = x^{n-8}\{x^8 - (n+1)x^6 - 4x^5 + (5n-23)x^4 + 8x^3 - (6n-40)x^2 - 4x + 2(n-8)\},$$

$$\psi(Q^{1,1}_{17}, x) = x^{n-6}\{x^6 - (n+1)x^4 - 4x^3 + (5n-26)x^2 + 2x - (3n-19)\},$$

$$\psi(Q^{1,1}_{18}, x) = x^{n-6}\{x^6 - (n+1)x^4 - 4x^3 + (5n-25)x^2 + 4x - (4n-28)\},$$

$$\psi(Q_{19}^{1,1},x) = x^{n-8}\{x^8-(n+1)x^6-4x^5+(5n-24)x^4+4x^3-4(n-7)x^2+(n-7)\},$$

$$\psi(Q_{20}^{1,1},x)=x^{n-8}\{x^8-(n+1)x^6-4x^5+(5n-23)x^4+6x^3-(6n-39)x^2-2x+(2n-15)\},$$

$$\psi(Q_{21}^{1,1},x)=x^{n-8}\{x^8-(n+1)x^6-4x^5+(5n-22)x^4+8x^3-(7n-45)x^2-4x+(3n-23)\},$$

$$\psi(Q_{22}^{1,1},x) = x^{n-10}\{x^{10}-(n+1)x^8-4x^7+(5n-21)x^6+10x^5-(8n-52)x^4 \\ -8x^3+(5n-40)x^2+2x-(n-9)\},$$

$$\psi(Q_{23}^{1,1},x)=x^{n-8}\{x^8-(n+1)x^6-4x^5+(5n-22)x^4+8x^3-(7n-47)x^2-4x+(3n-25)\},$$

$$\psi(Q_{24}^{1,1},x)=x^{n-8}\{x^8-(n+1)x^6-4x^5+(5n-21)x^4+10x^3-(6n-39)x^2-2x+(n-7)\},$$

$$\psi(Q_{25}^{1,1},x)=x^{n-8}\{x^8-(n+1)x^6-4x^5+(5n-23)x^4+8x^3-(5n-32)x^2-2x+(n-7)\},$$

$$\psi(Q_{26}^{1,1},x) = x^{n-6}\{x^6-(n+1)x^4-4x^3+(5n-23)x^2+6x-(5n-32)\},$$

$$\psi(Q_{27}^{1,1},x)=x^{n-8}\{x^8-(n+1)x^6-4x^5+(5n-22)x^4+8x^3-(5n-28)x^2-2x+(n-7)\},$$

$$\psi(Q_{28}^{1,1},x)=x^{n-8}\{x^8-(n+1)x^6-4x^5+(5n-21)x^4+10x^3-(7n-43)x^2-6x+3(n-8)\},$$

$$\psi(Q_{29}^{1,1},x) = x^{n-6}\{x^6-(n+1)x^4-4x^3+(5n-24)x^2+8x-(5n-33)\},$$

$$\psi(Q_{30}^{1,1},x) = x^{n-6}\{x^6-(n+1)x^4-4x^3+(5n-25)x^2+4x-(3n-19)\},$$

$$\psi(Q_{31}^{1,1},x)=x^{n-8}\{x^8-(n+1)x^6-4x^5+(5n-23)x^4+6x^3-(5n-31)x^2-2x+(n-7)\},$$

$$\psi(Q_{32}^{1,1},x)=x^{n-8}\{x^8-(n+1)x^6-4x^5+(5n-22)x^4+8x^3-(6n-39)x^2-4x+2(n-8)\},$$

$$\psi(Q_{33}^{1,1},x) = x^{n-6}\{x^6-(n+1)x^4-4x^3+(5n-22)x^2+10x-5(n-7)\},$$

$$\psi(Q_{34}^{1,1},x) = x^{n-6}\{x^6-(n+1)x^4-4x^3+(5n-23)x^2+8x-4(n-7)\},$$

$$\psi(Q_{35}^{1,1},x)=x^{n-8}\{x^8-(n+1)x^6-4x^5+(5n-21)x^4+10x^3-(7n-46)x^2-4x+(2n-15)\},$$

$$\psi(Q_{36}^{1,1},x) = x^{n-6}\{x^6-(n+1)x^4-4x^3+(5n-22)x^2+8x-(6n-41)\},$$

$$\psi(Q_{37}^{1,1},x) = x^{n-6}\{x^6-(n+1)x^4-4x^3+(5n-21)x^2+12x-(6n-44)\},$$

$$\psi(Q_{38}^{1,1},x) = x^{n-6}\{x^6-(n+1)x^4-4x^3+(5n-23)x^2+12x-6(n-8)\},$$

$$\psi(Q_{39}^{1,1},x) = x^{n-6}\{x^6-(n+1)x^4-4x^3+(4n-18)x^2+2(n-5)x-(n-5)\},$$

$$\psi(Q_{40}^{1,1},x) = x^{n-6}\{x^6-(n+1)x^4-4x^3+(4n-17)x^2+(2n-8)x-(n-5)\},$$

$$\psi(Q_{41}^{1,1},x)=x^{n-7}\{x^7-(n+1)x^5-4x^4+(4n-16)x^3+(2n-6)x^2-3(n-6)x-2(n-6)\},$$

$$\psi(Q_{42}^{1,1},x)=x^{n-6}\{x^6-(n+1)x^4-4x^3+(5n-26)x^2+2(n-6)x-2(n-6)\},$$

$$\psi(Q_{43}^{1,1},x) = x^{n-6}\{x^6-(n+1)x^4-4x^3+(5n-26)x^2+(2n-10)x-3(n-6)\},$$

$$\psi(Q_{44}^{1,1},x) = x^{n-6}\{x^6-(n+1)x^4-4x^3+(5n-25)x^2+2(n-6)x-3(n-6)\},$$

$$\begin{aligned}\psi(Q_{45}^{1,1},x) &= x^{n-8}\{x^8-(n+1)x^6-4x^5+(5n-23)x^4+(2n-8)x^3\\ &\quad -(5n-32)x^2-2(n-7)x+(n-7)\},\end{aligned}$$

$$\psi(Q_{46}^{1,1},x) = x^{n-6}\{x^6-(n+1)x^4-4x^3+(5n-24)x^2+(2n-8)x-2(n-6)\},$$

$$\begin{aligned}\psi(Q_{47}^{1,1},x) &= x^{n-8}\{x^8-(n+1)x^6-4x^5+(5n-22)x^4+(2n-6)x^3\\ &\quad -(5n-31)x^2-(2n-12)x+(n-7)\},\end{aligned}$$

$$\begin{aligned}\psi(Q_{48}^{1,1},x) &= x^{n-7}\{x^7-(n+1)x^5-4x^4+(5n-21)x^3+(2n-4)x^2\\ &\quad -(6n-39)x-(4n-26)\},\end{aligned}$$

$$\psi(Q_{49}^{1,1},x)=x^{n-7}\{x^7-(n+1)x^5-4x^4+(5n-23)x^3+(2n-8)x^2-4(n-6)x-2(n-6)\},$$

$$\psi(Q_{50}^{1,1},x)=x^{n-7}\{x^7-(n+1)x^5-4x^4+(5n-22)x^3+(2n-6)x^2-4(n-6)x-2(n-6)\},$$

$$\begin{aligned}\psi(Q_{51}^{1,1},x) &= x^{n-9}\{x^9-(n+1)x^7-4x^6+(5n-21)x^5+(2n-4)x^4\\ &\quad -(7n-45)x^3-(4n-24)x^2+3(n-8)+2(n-8)\},\end{aligned}$$

$$\psi(Q_{52}^{1,1},x)=x^{n-7}\{x^7-(n+1)x^5-4x^4+(5n-22)x^3+(2n-4)x^2-6(n-7)x-4(n-7)\}.$$

First we will show that $\mathcal{E}(B_{n,n+1}^{2,2}) < \mathcal{E}(Q_1^{1,1})$. Now, consider the function

$$\alpha_{10}(x) = x^6-(n+1)x^4-4x^3+(3n-12)x^2+2x-(n-5)$$

and proceeding similarly as in part (ii), Lemma 9, we can prove that $\mathcal{E}(Q_1^{1,1}) > E(B_{n,n+1}^{2,2})$ for all $n \geq 10$. Also, it is easy to see that even and odd coefficients of $Q_k^{1,1}$, $k = 1,2,3,\dots,52$, alternate in sign and clearly $Q_1^{1,1} \prec Q_k^{1,1}$ for all $k = 2,3,\dots,52$. Therefore, $\mathcal{E}(Q_1^{1,1}) < \mathcal{E}(Q_k^{1,1})$ for all $k = 2,3,\dots,52$. This completes the proof.

(ii). We have

$$\psi(Q_1^{1,3},x) = x^{n-6}\{x^6-(n+1)x^4+(3n-8)x^2+2x-(n-5)\},$$

$$\psi(Q_3^{1,3},x) = x^{n-6}\{x^6-(n+1)x^4+(4n-14)x^2-4x-2(n-6)\},$$

$$\psi(Q_4^{1,3},x) = x^{n-6}\{x^6-(n+1)x^4+(4n-14)x^2-2x-(2n-11)\},$$

$$\psi(Q_6^{1,3},x) = x^{n-8}\{x^8-(n+1)x^6+(4n-12)x^4+2x^3-(4n-20)x^2-2x+(n-7)\},$$

$$\psi(Q_8^{1,3},x) = x^{n-6}\{x^6-(n+1)x^4+(4n-12)x^2+2x-(3n-14)\},$$

$$\psi(Q_{13}^{1,3},x) = x^{n-6}\{x^6-(n+1)x^4+(5n-22)x^2+6x-3(n-7)\},$$

$$\psi(Q_{14}^{1,3},x) = x^{n-6}\{x^6-(n+1)x^4+(5n-21)x^2+4x-(4n-26)\},$$

$$\psi(Q_{17}^{1,3},x) = x^{n-6}\{x^6-(n+1)x^4+(5n-22)x^2-2x-(3n-19)\},$$

$$\psi(Q_{39}^{1,3},x) = x^{n-6}\{x^6-(n+1)x^4+(4n-14)x^2-2(n-5)x-(n-5)\}$$

$$\psi(Q_{40}^{1,3},x) = x^{n-6}\{x^6-(n+1)x^4+(4n-13)x^2-(2n-8)x-(n-5)\},$$

$$\psi(Q_{41}^{1,3},x) = x^{n-7}\{x^7-(n+1)x^5+(4n-12)x^3-(2n-10)x^2-(3n-14)x+2(n-6)\},$$

$$\psi(Q_{42}^{1,3},x) = x^{n-6}\{x^6-(n+1)x^4+(5n-22)x^2-2(n-6)x-2(n-6)\},$$

$$\psi(Q_{43}^{1,3},x) = x^{n-6}\{x^6-(n+1)x^4+(5n-22)x^2-(2n-14)x-3(n-6)\},$$

$$\psi(Q_{44}^{1,3},x) = x^{n-6}\{x^6-(n+1)x^4+(5n-21)x^2-2(n-6)x-3(n-6)\}.$$

First we will show that $\mathcal{E}(B_{n,n+1}^{2,2}) < \mathcal{E}(Q_1^{1,3})$. Consider the function

$$\alpha_{11}(x) = x^6-(n+1)x^4+(3n-8)x^2+2x-(n-5)$$

and proceeding similarly as in part (ii), Lemma 9, we can prove that $\mathcal{E}Q_1^{1,3}) > \mathcal{E}(B_{n,n+1}^{2,2})$ for all $n \geq 10$. Also, the even and odd coefficients of $Q_k^{1,3}$,

$$k = 2,3,4,6,8,13,14,17,39,40,41,42,43,44,$$

alternate in sign and clearly $Q_1^{1,3} \prec Q_k^{1,1}$, for all

$$k = 3,4,6,8,13,14,17,39,40,41,42,43,44.$$

Therefore, $\mathcal{E}(Q_1^{1,1}) < \mathcal{E}(Q_k^{1,1})$ for all

$$k = 3,4,6,8,13,14,17,39,40,41,42,43,44.$$

This completes the proof.
(iii, iv). The proof is similar to (1). □

Let $Q_k^{2,1}$, $k = 1,2,3,\ldots,34$ be the graphs as shown in Figure 9. Then it is easy to see that there are four switching classes on the signings of $Q_k^{2,1}$, $k = 1,2,3,\ldots,34$. Let $Q_k^{2,1}$, $Q_k^{2,2}$, $Q_k^{2,3}$, $Q_k^{2,4}$, $k = 1,2,3,\ldots,34$, respectively be the representative for these four switching classes, where $Q_k^{2,1}$ contains C_3^+, C_4^+ and C_5^+;$Q_k^{2,2}$ contains C_3^-, C_4^- and C_5^+; $Q_k^{2,3}$ contains C_3^-, C_4^+ and C_5^-; and $Q_k^{2,4}$ contains C_3^+, C_4^- and C_5^-. It is easy to see that $Q_k^{2,1}$ are switching equivalent to $-Q_k^{2,3}$ and $Q_k^{2,2}$ are switching equivalent to $-Q_k^{2,4}$, therefore $\mathcal{E}(Q_k^{2,1}) = \mathcal{E}(Q_k^{2,3})$ and $\mathcal{E}(Q_k^{2,2}) = \mathcal{E}(Q_k^{2,4})$ for all $k = 1, 2, 3, \dots, 34$. Thus, we can regard $Q_k^{2,1}$ and $Q_k^{2,3}$ as identical, $Q_k^{2,2}$ and $Q_k^{2,4}$ as identical, respectively. Also, $b_4(Q_k^{2,r}) > 5n - 21$ for all $k = 7, 8, \dots, 34$, $k \neq 26$ and $r = 2, 4$, therefore we omit these signed graphs here, as these signed graphs will be considered later. With these notations, we have the following result, whose proof is similar as in Lemma 18. So we skip the proof here.

Lemma 19 *For all* $n \geq 10$*, we have*
(i) $\mathcal{E}(B_{n,n+1}^{2,2}) < \mathcal{E}(Q_1^{2,1}) < \mathcal{E}(Q_k^{2,1})$ *for all* $k = 2, 3, \dots, 34$.
(ii) $\mathcal{E}(Q_1^{2,1}) < \mathcal{E}(Q_1^{2,2}) < \mathcal{E}(Q_k^{2,2})$ *for all* $k = 2, 3, 4, 5, 6, 26$.

Let $Q_1^{3,1}$ be the graph as shown in Figure 10. It is easy to see that there are four switching classes on the signings of $Q_1^{3,1}$. Let $Q_1^{3,1}$, $Q_1^{3,2}$, $Q_1^{3,3}$, $Q_1^{3,3}$ be the representative for these four switching classes, where $Q_1^{3,1}$ contains C_3^+, C_5^+ and C_6^+; $Q_1^{3,2}$ contains C_3^-, C_5^- and C_6^+; $Q_1^{3,3}$ contains C_3^-, C_5^+ and C_6^-; and $Q_1^{3,4}$ contains C_3^+, C_5^- and C_6^-. It is easy to see that $Q_1^{3,1}$ is switching equivalent to $-Q_1^{3,2}$ and $Q_1^{3,3}$ is switching equivalent to $-Q_1^{3,4}$, therefore $\mathcal{E}(Q_1^{3,1}) = \mathcal{E}(Q_1^{3,2})$ and $\mathcal{E}(Q_1^{3,3}) = \mathcal{E}(Q_1^{3,4})$. Thus, we can regard $Q_1^{3,1}$ and $Q_1^{3,2}$ as identical, $Q_1^{3,3}$ and $Q_1^{3,4}$ as identical. Also, let $Q_k^{4,1}$, $k = 1, 2, 3, 4, 5, 6$ be the graphs as shown in Figure 10. Then it is easy to see that there are three switching classes on the signings of $Q_k^{4,1}$, $k = 1, 2, 3, 4, 5$. Let $Q_k^{4,1}$, $Q_k^{4,2}$, $Q_k^{4,3}$, $k = 1, 2, 3, 4, 5, 6$, respectively be the representative for these three switching classes, where $Q_k^{4,1}$ contains C_4^+, C_4^+ and C_6^+; $Q_k^{4,2}$ contains C_4^-, C_4^+ and C_6^-; $Q_k^{4,3}$ contains C_4^-, C_4^- and C_6^+. It is easy to see that $b_4(Q_k^{4,r}) > 5n - 21$ for all $k = 2, 3, 4, 5, 6$, $r = 2, 3$ and therefore we omit these signed graphs here, as these signed graphs will be considered later. With these notations, we have the following result, whose proof is similar as in Lemma 18. So we skip the proof here.

Lemma 20 *For all* $n \geq 10$*, we have* (i) $\mathcal{E}(B_{n,n+1}^{2,2}) < \mathcal{E}(Q_1^{3,1})$.
(ii) $\mathcal{E}(B_{n,n+1}^{2,2}) < \mathcal{E}(Q_1^{3,3})$.
(iii) $\mathcal{E}(B_{n,n+1}^{2,2}) < \mathcal{E}(Q_1^{4,1}) < \mathcal{E}(Q_k^{4,t})$ *for all* $k = 1$ *when* $t = 2, 3$ *and* $k = 2, 3, 4, 5$ *when* $t = 1$.
(iv) $\mathcal{E}(B_{n,n+1}^{2,2}) < \mathcal{E}(Q_6^{4,1})$.

Let $Q_k^{5,1}$, $k = 1, 2, 3, \dots, 15$, be the graphs as shown in Figure 11. Clearly, there are two switching classes on the signings of $Q_k^{5,1}$ $k = 1, 2, 3, \dots, 15$. Let $Q_k^{5,1}$ and $Q_k^{5,2}$, $k = 1, 2, 3, \dots, 15$, be the representative for these two switching classes, where $Q_k^{5,1}$ contains C_4^+, C_4^+, C_4^+; and $Q_k^{5,2}$ contains C_4^-, C_4^- and C_4^+ respectively. Also, let $Q_k^{6,1}$, $k = 1, 2, \dots, 7$, be the graphs as shown in Figure 11. Then it is easy to see that there are three switching classes on the signings of $Q_k^{6,1}$, $k = 1, 2, \dots, 7$. Let $Q_k^{6,1}$, $Q_k^{6,2}$, $Q_k^{6,3}$,$k = 1, 2, \dots, 7$ respectively, be the representative for these three switching classes, where $Q_k^{6,1}$ contains C_4^+, C_5^+ and C_5^+; $Q_k^{3,2}$ contains C_4^+, C_5^- and C_5^-; $Q_k^{6,3}$ contains C_4^-, C_5^+ and C_5^-. It is easy to see that $Q_k^{6,1}$ is switching equivalent to $-Q_k^{6,2}$. Therefore, $\mathcal{E}(Q_k^{6,1}) = \mathcal{E}(Q_k^{6,2})$ for all $k = 1, 2, \dots, 7$, thus we can regard $Q_k^{6,1}$ and $Q_k^{6,2}$ as identical. Also, $b_4(Q_k^{5,2}) > 5n - 21$ for all $k = 3, 4, \dots, 15$ and $b_4(Q_k^{6,3}) > 5n - 21$ for all $k = 2, 3, \dots, 7$. Therefore, we omit these signed graphs here, as these signed graphs will be considered later. With these notations, we have the following result, whose proof is similar as in Lemma 18. So we skip the proof here.

Lemma 21 *For all* $n \geq 10$, *we have* (i) $\mathcal{E}(B_{n,n+1}^{2,2}) < \mathcal{E}(Q_1^{5,1}) < \mathcal{E}(Q_k^{5,t})$ *for all* $k = 1, 2$ *when* $t = 2$ *and* $k = 2, 3, \dots, 15$ *when* $t = 1$.
(ii) $\mathcal{E}(Q_1^{5,1}) < \mathcal{E}(Q_k^{6,t})$ *for all* $k = 1$ *when* $t = 3$ *and* $k = 1, 2, 3, \dots, 7$ *when* $t = 1$.

Now, we have the following theorem.

Theorem 22 *Let* $S \in \theta(n, p, q, r)$, $S \neq S_{n,n+1}^{k,t}$ $(k = 0, 1, 2, 3$ *and* $t = 1, 2, 3)$, $B_{n,n+1}^{k,t}$ $(k = 0, 1, 2,$ *and* $t = 1, 2)$, $Q_{n,n+1}^{1,1}$, $Q_{n,n+1}^{2,t}$ $(t = 1, 3)$, $Q_{n,n+1}^{3,t}$ $(t = 1, 2, 3)$, *where* $n \geq 17$, $p \geq 3$, $q \geq 3$ *and* $r \geq 1$. *Then* $\mathcal{E}(S) > \mathcal{E}(B_{n,n+1}^{2,2})$.

Proof. As C_p and C_q have $r \geq 1$ common edges, we first assume that $r \geq 6$, and let $P_{r+1} = e_1 e_2 \dots e_r$ be the path formed by these r edges. Choose the cut set as $Z = \{e_1, e_r\}$, so that $S - \{e_1, e_r\}$ has two components say S' and S'', where S' is a signed tree on $m_1 \geq 5$ vertices and S'' is a unicyclic signed graph on $m_2 \geq 5$ vertices. Therefore the result follows by Lemmas 14 and 15. This proves the result for $r \geq 6$ and $n \geq 17$. For $r \leq 5$, the above technique still applies but we need to consider several cases while choosing the cut set. For $r \leq 5$, we prove the result by induction on $n - p - q + r$. Clearly, $n - p - q + r \geq -1$. If $n - p - q + r = -1$, then S has no pendant edge. We consider the following cases, (i) $r = 1$, (ii) $r = 2$ and (iii) $3 \leq r \leq 5$.

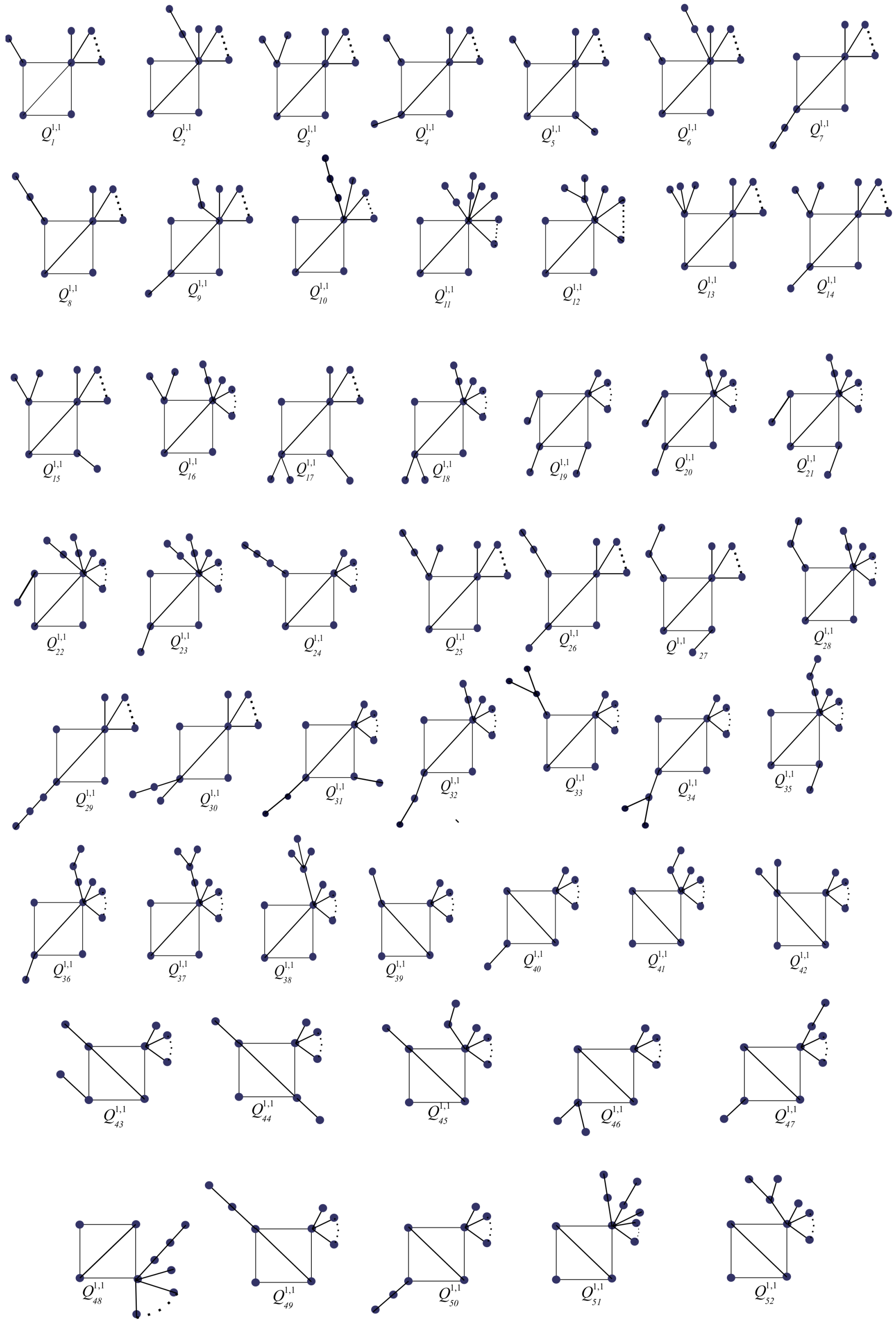


Figure 8: Signed graphs $Q_k^{1,1},\ k = 1, 2, \dots, 52$

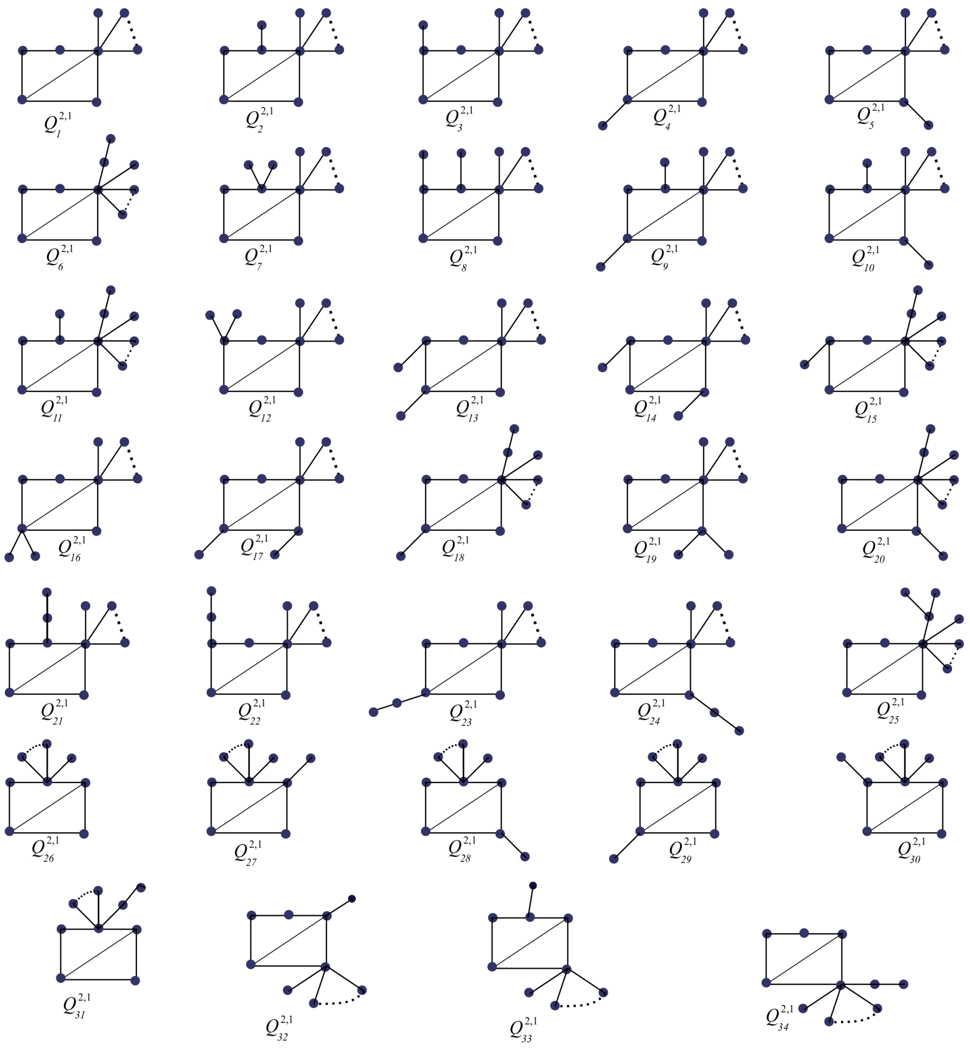


Figure 9: Signed graphs $Q_k^{2,1}$, $k = 1, 2, \ldots, 34$

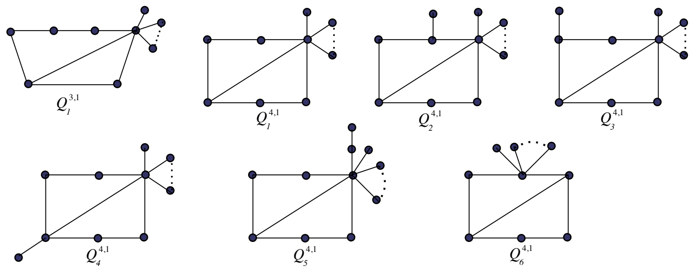


Figure 10: Signed graphs $Q_1^{3,1}$ and $Q_k^{4,1}$, $k = 1, 2, 3, 4, 5, 6$

Case (i). If $r = 1$, then $n - p - q = -2$ or $n = p + q - 2$. As $n \geq 17$, S can have at most one 4-cycle. If $p = 4$, then $n = q + 2$. Now, by (3.1), we have

$$b_4(S) = q - 2 + m(C_{q+2}, 2) \pm 2 = \begin{cases} \frac{1}{2}(q^2 + 3q - 10), & \text{if 4-cycle is positive} \\ \frac{1}{2}(q^2 + 3q - 2), & \text{if 4-cycle is negative.} \end{cases}$$

As $b_4(B^{2,2}_{n,n+1}) = 5n - 21$, so $b_4(B^{2,2}_{q+2,q+3}) = 5q - 11$ and therefore

$$b_4(S) - b_4(B^{2,2}_{q+2,q+3}) = b_4(S) - (5n - 11) = \begin{cases} \frac{1}{2}(q^2 - 7q + 12), & \text{if 4-cycle is positive} \\ \frac{1}{2}(q^2 - 7q + 20), & \text{if 4-cycle is negative.} \end{cases}$$

Since $n \geq 17$, so $q \geq 15$ and thus $b_4(S) - b_4(B^{2,2}_{q+2,q+3}) > 0$. If there is no 4-cycle in S, then since $n = p + 1 - 2$, we have $b_4(B^{2,2}_{n,n+1}) = 5n - 11 = 5(p+q) - 31$. Also,

$$b_4(S) = m(S, 2) = m(C_{p+q-2}, 2) + p + q - 6 = \frac{1}{2}[(p+q)^2 - 5(p+q) - 2]$$

and so

$$b_4(S) - b_4(B^{2,2}_{n,n+1}) = \frac{1}{2}[(p+q)^2 - 15(p+q) + 60] > 0$$

because $n \geq 17$ and so $p + q \geq 19$.

Case (ii). Let $r = 2$, so that in this case $n = p + q - 3$. Again, S can have at most one 4-cycle. Suppose S contains a 4-cycle and let $p = 4$. Then $n = q + 1$ and so $b_4(B^{2,2}_{n,n+1}) = b_4(B^{2,2}_{q+1,q+2}) = 5q - 16$. Also,

$$b_4(S) = 2(q-2) + m(C_q, 2) \pm 2 = 2(q-2) + \frac{q(q-3)}{2} \pm 2.$$

Therefore,

$$b_4(S)-b_4(B^{2,2}_{q+1,q+2})=b_4(S)-(5q-16)=\begin{cases}\frac{1}{2}(q^2-9q+20), & \text{if 4 cycle is positive}\\ \frac{1}{2}(q^2-9q+28), & \text{if 4 cycle is negative.}\end{cases}$$

As $n \geq 17$, so $q \geq 16$ and therefore $b_4(S) - b_4(B^{2,2}_{q+1,q+2}) > 0$. Suppose S does not contain a 4-cycle. Then

$$\begin{aligned} b_4(S) &= m(S,2) = 2(p+q-6) + m(C_{p+q-4}, 2)\\ &= \frac{1}{2}\{4(p+q-6)(p+q-4)(p+q-7)\}\\ &= \frac{1}{2}\{(p+q)^2 - 7(p+q) + 4\}. \end{aligned}$$

Therefore

$$b_4(S) - b_4(B^{2,2}_{n,n+1}) = \frac{1}{2}\{(p+q)^2 - 17(p+q) + 76\} > 0,$$

since $p + q \geq 20$.

Case (iii). If $3 \leq r \leq 5$, then $n = p + q - r - 1$. So S does not contain a 4-cycle. Then proceeding similarly as above, we can prove that

$$b_4(S) - b_4(B^{2,2}_{n,n+1}) = \frac{1}{2}\{(p+q)^2 - 13(p+q) + r^2 + 13r - 2(p+q)r + 46\} > 0.$$

This proves that the result is true for $n - p - q + r = -1$. Assume the result to be true for $n - p - q + r < p'$, where $p' \geq 0$. Let $n - p - q + r = p'$. Then S has a pendent edge, say $e = uv$, with v as a pendant vertex. Apply Lemma 4, we have

$$b_4(S) = b_4(S - \{v\}) + b_2(S - \{u, v\})$$

and

$$b_4(B^{2,2}_{n,n+1}) = b_4(B^{2,2}_{n-1,n}) + b_2(S_5).$$

By induction, it is easy to see that $b_4(S-\{v\}) > b_4(B^{2,2}_{n-1,n})$ and $b_2(S-\{u,v\}) \geq 5 = b_2(S_5)$ if $S \neq S^{k,t}_{n,n+1}$ $(k = 0,1,2,3$ and $t = 1,2,3)$, $B^{k,t}_{n,n+1}$ $(k = 0,1,2,$ and $t = 1,2)$, $Q^{1,t}_{n,n+1}$ $(t = 1,2)$, $Q^{2,t}_{n,n+1}$ $(t = 1,2,3,4)$, $Q^{3,t}_{n,n+1}$ $(t = 1,2,3)$, $Q^{1,t}_{k}$ $(k = 1,2,\ldots,52$ and $t = 1,2)$, $Q^{1,3}_{k}$ $(k = 1,2,\ldots,14,17,18,30,39,40,\ldots,44)$, $Q^{2,t}_{k}$ $(k = 1,2,3,\ldots,34$ and $t = 1,3)$, $Q^{2,t}_{k}$ $(k = 1,2,3,4,5,6,26$ and $t = 2,4)$, $Q^{3,t}_{1}$ $(t = 1,2,3,4)$, $Q^{4,1}_{k}$ $(k = 1,2,\ldots,6)$, $Q^{4,t}_{1}$ $(t = 2,3)$, $Q^{5,1}_{k}$ $(k =$

$1, 2, \dots, 15)$, $Q_1^{5,2}$, $Q_2^{5,2}$, $Q_k^{6,t}$ $(k = 1, 2, \dots, 7$ and $t = 1, 2)$ and $Q_1^{6,3}$. Thus $b_4(S) > b_4(B_{n,n+1}^{2,2})$. Further as $S \in \theta(n, p, q, r)$, so $b_{2j}(S) \geq 0$ for all $j \geq 3$. Also, $b_{2j}(B_{n,n+1}^{2,2}) = 0$ for all $j \geq 3$ and $b_{2j+1}(B_{n,n+1}^{2,2}) = 0$ for all $j \geq 0$. The second summand of logarithm in integral formula given in Theorem 2 is non negative for the signed graph $S \in \theta(n, p, q, r)$ (in fact for every signed graph). Hence, $S \succ B_{n,n+1}^{2,2}$. By integral formula given in Theorem 2, we see that $\mathcal{E}(S) > \mathcal{E}(B_{n,n+1}^{2,2})$. If $S = Q_{n,n+1}^{1,2}$, $Q_{n,n+1}^{2,2}$, $Q_{n,n+1}^{2,4}$, $Q_k^{1,t}$ $(k = 1, 2, \dots, 52$ and $t = 1, 2)$, $Q_k^{1,3}$ $(k = 1, 2, \dots, 14, 17, 18, 30, 39, 40, \dots, 44)$, $Q_k^{2,t}$ $(k = 1, 2, 3, \dots, 34$ and $t = 1, 3)$, $Q_k^{2,t}$ $(k = 1, 2, 3, 4, 5, 6, 26$ and $t = 2, 4)$, $Q_1^{3,t}$ $(t = 1, 2, 3, 4)$, $Q_k^{4,1}$ $(k = 1, 2, \dots, 6)$, $Q_1^{4,t}$ $(t = 2, 3)$, $Q_k^{5,1}$ $(k = 1, 2, \dots, 15)$, $Q_1^{5,2}$, $Q_2^{5,2}$, $Q_k^{6,t}$ $(k = 1, 2, \dots, 7$ and $t = 1, 2)$ and $Q_1^{6,3}$, then the result follows by Lemmas 9, 18, 19, 20 and 21. This completes the proof. □

Theorem 23 *Among all bicyclic signed graphs with* $n \geq 17$ *vertices,* $B_{n,n+1}^{2,2}$ *is the signed graph with* 20th *minimal energy for all* $n \geq 30$ *and with* 16th *minimal energy for all* $17 \leq n \leq 29$. *Also, we have ordering of energies in ascending order as follows.*

(i) *For all* $n \geq 30$*, we have*

$$\mathcal{E}(S_{n,n+1}^{0,1}) = \mathcal{E}(S_{n,n+1}^{0,2}) < \mathcal{E}(S_{n,n+1}^{0,3}) < \mathcal{E}(B_{n,n+1}^{0,1}) < \mathcal{E}(S_{n,n+1}^{1,1}) = \mathcal{E}(S_{n,n+1}^{1,2})$$
$$< \mathcal{E}(S_{n,n+1}^{1,3}) < \mathcal{E}(B_{n,n+1}^{0,2}) < \mathcal{E}(B_{n,n+1}^{1,1}) < \mathcal{E}(Q_{n,n+1}^{1,1}) < \mathcal{E}(S_{n,n+1}^{2,1}) = \mathcal{E}(S_{n,n+1}^{2,2})$$
$$< \mathcal{E}(S_{n,n+1}^{2,3})$$
$$< \mathcal{E}(Q_{n,n+1}^{2,1}) = \mathcal{E}(Q_{n,n+1}^{2,3}) < \mathcal{E}(B_{n,n+1}^{1,2}) < \mathcal{E}(Q_{n,n+1}^{3,1}) = \mathcal{E}(Q_{n,n+1}^{3,2})$$
$$< \mathcal{E}(Q_{n,n+1}^{3,3}) < \mathcal{E}(H_{n,n+1}^{3}) < \mathcal{E}(H_{n,n+1}^{1}) = \mathcal{E}(H_{n,n+1}^{2}) < \mathcal{E}(B_{n,n+1}^{2,1}) < \mathcal{E}(S_{n,n+1}^{3,1})$$
$$= \mathcal{E}(S_{n,n+1}^{3,2}) < \mathcal{E}(S_{n,n+1}^{3,3}) < \mathcal{E}(B_{n,n+1}^{2,2}).$$

(ii) *For all* $17 \leq n \leq 29$*, we have*

$$\mathcal{E}(S_{n,n+1}^{0,1}) = \mathcal{E}(S_{n,n+1}^{0,2}) < \mathcal{E}(S_{n,n+1}^{0,3}) < \mathcal{E}(B_{n,n+1}^{0,1}) < \mathcal{E}(S_{n,n+1}^{1,1}) = \mathcal{E}(S_{n,n+1}^{1,2})$$
$$< \mathcal{E}(S_{n,n+1}^{1,3}) < \mathcal{E}(B_{n,n+1}^{0,2}) < \mathcal{E}(B_{n,n+1}^{1,1}) < \mathcal{E}(Q_{n,n+1}^{1,1}) < \mathcal{E}(S_{n,n+1}^{2,1}) = \mathcal{E}(S_{n,n+1}^{2,2}) <$$
$$\mathcal{E}(S_{n,n+1}^{2,3}) < \mathcal{E}(Q_{n,n+1}^{2,1}) = \mathcal{E}(Q_{n,n+1}^{2,3})$$
$$< \mathcal{E}(B_{n,n+1}^{1,2}) < \mathcal{E}(B_{n,n+1}^{2,1}) < \mathcal{E}(S_{n,n+1}^{3,1}) = \mathcal{E}(S_{n,n+1}^{3,2}) < \mathcal{E}(S_{n,n+1}^{3,3}) < \mathcal{E}(B_{n,n+1}^{2,2}).$$

Proof. This follows by Corollary 10 and Theorems 12, 17 and 22. □

Acknowledgements. This research is supported by SERB-DST research project number CRG/2020/000109. The research of Tahir Shamsher is supported by JRF financial assistance by Council of Scientific and Industrial Research (CSIR), New Delhi, India.

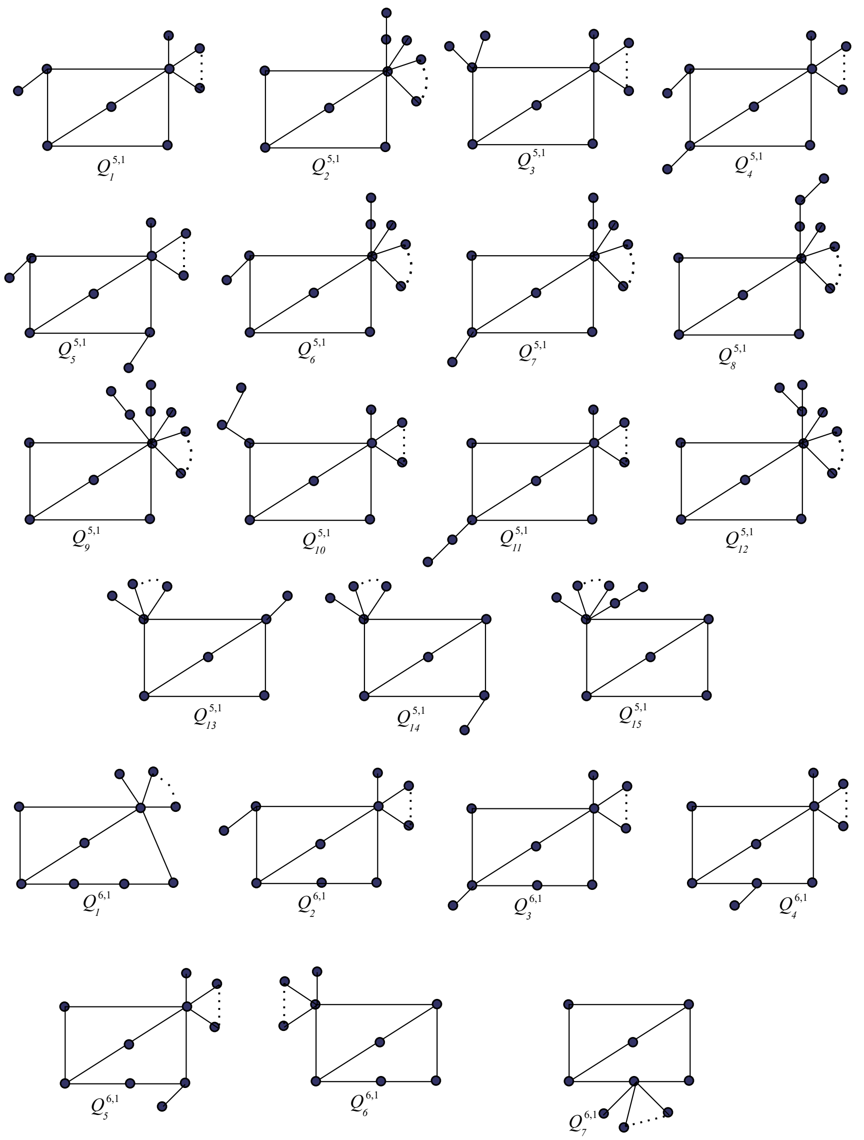


Figure 11: Signed raphs $Q_k^{5,1}$, $k = 1, 2, \ldots 15$ and $Q_k^{6,1}$, $k = 1, 2, \ldots, 7$

References

[1] B. D. Acharya, Spectral criterion for the cycle balance in networks, *J. Graph Theory* **4** (1980) 1–11. ⇒88

[2] M. A. Bhat, S. Pirzada, Unicyclic signed graphs with minimal energy, *Discrete Appl. Math.* **226** (2017) 32–39. ⇒87, 88, 90

[3] M. A. Bhat, S. Pirzada, On equienergetic signed graphs, *Discrete Appl. Math.* **189** (2015) 1–7. ⇒87, 103

[4] M. A. Bhat, U. Samee, S. Pirzada, Bicyclic signed graphs with minimal and second minimal energy, *Linear Algebra Appl.* **551** (2018) 18–35. ⇒87, 89, 90, 91, 99

[5] X. Yang L. Wang, On the ordering of bicyclic digraphs with respect to energy and iota energy, *Applied Math. Comput.* **339** (2018) 768–778. ⇒87

[6] K. A. Germina, S. Hameed, T. Zaslavsky, On products and line graphs of signed graphs,their eigenvalues and energy, *Linear Algebra Appl.* **435** (2010) 2432–2450. ⇒87

[7] I. Gutman, The energy of a graph, *Ber. Math. Statist. Forschungszenturm Graz.* **103** (1978) 1–22. ⇒87

[8] S. Ji, J. Li, An approach to the problem of the maximal energy of bicyclic graphs, *MATCH Commun. Math. Comput. Chem.* **68** (2012) 741–762. ⇒103

[9] D. Wang, Y. Hou, Bicyclic signed graphs with at most one odd cycle and maximal energy, *Discrete Applied Math.* **260** (2019) 244–255. ⇒87

[10] S. Hafeez, R. Farooq and M. Khan, Bicyclic signed digraphs with maximal energy, *Applied Math. Comput.* **347** (2019) 702-=711. ⇒87

[11] Y. Hou, Bicyclic graphs with minimum energy, *Linear Multilinear Algbera* **49** (2001) 347–354. ⇒87

[12] S. Pirzada, *An Introduction to Graph Theory*, Universities Press, Hyderabad, India, 2012. ⇒103

[13] S. Pirzada, M. A. Bhat, Energy of signed digraphs, *Discrete Applied Math.* **169** (2014) 195–205. ⇒87

[14] J. Rada, Energy ordering of catacondensed hexagonal systems, *Discrete Appl. Math.* **145** (2005) 437–443. ⇒87

[15] J. Zhu, Unicyclic signed graphs with first $\lfloor \frac{n+1}{2} \rfloor$ largest energies, *Discrete Appl. Math.* **285** (2020) 350–363. ⇒87

[16] J. Zhang, B. Zhou, On bicyclic graphs with minimal energy, *J. Math. Chem.* **37** (2005) 423—431. ⇒87

[17] J. Zhang, H. Kan, On the minimal energy of graphs, *Linear Algebra Appl.* **153** (2014) 141–153. ⇒87

Received: March 22, 2021 • Revised: April 5, 2021

Acta Univ. Sapientiae, Informatica **13,** 1 (2021) 122–133

DOI: 10.2478/ausi-2021-0006

Estimating the fractional chromatic number of a graph

Sándor SZABÓ
University of Pécs
email: `sszabo7@hotmail.com`

Abstract. The fractional chromatic number of a graph is defined as the optimum of a rather unwieldy linear program. (Setting up the program requires generating all independent sets of the given graph.) Using combinatorial arguments we construct a more manageable linear program whose optimum value provides an upper estimate for the fractional chromatic number. In order to assess the feasibility of the proposal and in order to check the accuracy of the estimates we carry out numerical experiments.

1 Introduction and preliminaries

A loop and double edge free graph with finitely many vertices and edges is referred as a finite simple graph. Let V and E be the set of vertices and edges of a finite simple graph G, respectively. Clearly, the ordered pair (V, E) determines G uniquely.

Let $G = (V, E)$ be a finite simple graph. A subset I of V is called an independent set if any two distinct vertices of I are non-adjacent in G. We say that a subset C of V is a clique in G if two distinct vertices of C are always adjacent in G. If C has k elements we call C a k-clique. We would like to point out that a one element subset of V is both a clique and an independent set

Computing Classification System 1998: G.2.2
Mathematics Subject Classification 2010: 11A05, 11R04
Key words and phrases: clique and chromatic numbers, fractional chromatic number, integer programming.

of G. Similarly, the empty set is both a clique and an independent set of G. Sometimes the subgraph induced by C in G is called a clique. This ambiguity in the terminology is not going to cause any problem.

For each finite simple graph G there is an integer k such that G contains a k-clique but G does not contain any $(k+1)$-clique. This well defined k is called the clique number of G and it is denoted by $\omega(G)$. We color the nodes of G choosing colors from a palette of k colors such that each vertex of G receives exactly one color and the end points of adjacent vertices never receive the same color. Such a coloring of the nodes of G is called a legal k-coloring. For each finite simple graph G there is an integer k such that the nodes of G can be legally colored with k colors but they cannot be legally colored using $(k-1)$ colors. This well defined k is called the chromatic number of G and it is denoted by $\chi(G)$.

There are important problems in applied and theoretical discrete optimization which are essentially about determining the clique or chromatic numbers of a given graph. (Many such instances can be found in [1].) It is known from the complexity theory of the computations that the optimization problems of computing the clique or chromatic numbers belong to the NP hard complexity class. (For more details see [2], [4].) A commonly held interpretation of this fact is that computing the clique or chromatic numbers are computationally demanding tasks.

Let $G = (V, E)$ be a finite simple graph. If a vertex v of G is adjacent to each vertex of G distinct from v, then we call v a full degree node. Deleting a full degree node from G is reducing both the clique and chromatic number of G by one. From this reason we may restrict our attention to graphs without full degree nodes when we are looking for the clique or chromatic numbers.

Let $G = (V, E)$ be a full degree free finite simple graph and let $I(1), \ldots, I(r)$ be all the independent sets of G. To each vertex v of G we assign a non-negative variable $x(v)$. The optimum value of the linear program P

$$\sum_{v\in V} x(v) \to \max$$

$$\sum_{v\in I(s)} x(v) \leq 1, \quad 1 \leq s \leq r$$

is called the fractional clique number of G and it is denoted by $\omega_f(G)$. The optimum value of the dual of the program is referred to as fractional chromatic number of the graph G and $\chi_f(G)$ denotes it. By the duality theorem of linear programming, $\omega_f(G) = \chi_f(G)$.

	1	2	3	4	5	6	7	8
1	×	●	●	●	●			
2	●	×	●			●	●	
3	●	●	×	●		●	●	
4	●		●	×				●
5	●				×	●		●
6		●	●		●	×	●	●
7		●	●			●	×	●
8				●	●	●	●	×

	1	2	3	4	5	6	7	8
1	×					●	●	●
2		×		●	●			●
3			×		●			●
4		●		×	●	●	●	
5		●	●	●	×		●	
6	●			●		×		
7	●			●	●		×	
8	●	●	●					×

Table 1: The adjacency matrices of the graph G and its complement $\overline{G}$ in Example 2.

Note that the linear program P is a relaxed version of the linear program Q

$$\sum_{v \in V} x(v) \to \max$$

$$x(u) + x(v) \leq 1, \quad \{u, v\} \notin E.$$

Solving Q in zero-one variables gives that $\omega(G) \leq \omega_f(G)$. The main reason behind restricting our attention to full degree free graphs is that in this situation the inequality $x(v) \leq 1$ holds for each $v \in V$.

Consider a legal coloring of the vertices of G and let $C(1), \ldots, C(k)$ be the color classes of the nodes. Note that the linear program R

$$\sum_{v \in V} x(v) \to \max$$

$$\sum_{v \in C(s)} x(v) \leq 1, \quad 1 \leq s \leq k$$

is a relaxed version of the linear program P. The optimum value of the dual of R is k even if we solve it in zero-one variables. It follows that $\chi_f(G) \leq \chi(G)$. Therefore

$$\omega(G) \leq \omega_f(G) = \chi_f(G) \leq \chi(G).$$

	1	1	1	2	2	2	3	3	4	4	4	5
	6	7	8	4	5	8	5	8	5	6	7	7
1,6	×											
1,7		×										
1,8			×									
2,4				×	•				•			
2,5				•	×				•			
2,8						×						
3,5							×					
3,8								×				
4,5				•	•				×		•	•
4,6										×		
4,7									•		×	•
5,7									•		•	×

	1	1	1	2	2	2	3	3	4	4	4	5
	6	7	8	4	5	8	5	8	5	6	7	7
1,6	×	•	•	•	•	•	•	•	•	•	•	•
1,7	•	×	•	•	•	•	•	•	•	•	•	•
1,8	•	•	×	•	•	•	•	•	•	•	•	•
2,4	•	•	•	×		•	•	•		•	•	•
2,5	•	•	•		×	•	•	•		•	•	•
2,8	•	•	•	•	•	×	•	•	•	•	•	•
3,5	•	•	•	•	•	•	×	•	•	•	•	•
3,8	•	•	•	•	•	•	•	×	•	•	•	•
4,5	•	•	•			•	•	•	×	•		
4,6	•	•	•	•	•	•	•	•	•	×	•	•
4,7	•	•	•	•	•	•	•	•		•	×	
5,7	•	•	•	•	•	•	•	•		•		×

Table 2: The adjacency matrices of the edge auxiliary graph Γ of the graph $\overline{G}$ and its complement $\overline{\Gamma}$ in Example 2.

{1,6}	[1	[1	[1	[1	[1	[1	[1	[1	[1	[1	[1	1
{1,7}	←	[2	[2	[2	[2	[2	[2	[2	[2	[2	[2	2
{1,8}		←	[3	[3	[3	[3	[3	[3	[3	[3	[3	3
{2,4}			←	4	[4	[4	[4	4	[4	[4	[4	4
{2,5}				←	[4	[4	[4	4	[4	[4	[4	4
{2,8}					←	[5	[5	[5	[5	[5	[5	5
{3,5}						←	[6	[6	[6	[6	[6	6
{3,8}							←	[7	[7	[7	[7	7
{4,5}								←	[4	4	4	4
{4,6}									←	[8	[8	8
{4,7}										←	9	9
{5,7}											←	9

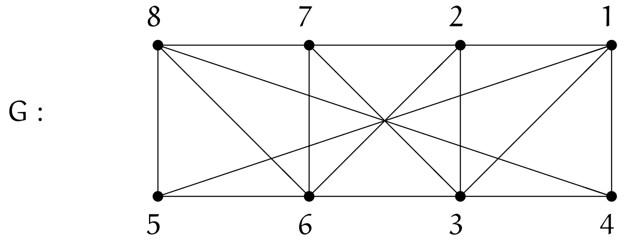
Table 3: Greedy sequential coloring of the nodes of the complement of the edge auxiliary graph $\overline{\Gamma}$ in Example 2.

C(1):	{1,6}	I(1):	1,6
C(2):	{1,7}	I(2):	1,7
C(3):	{1,8}	I(3):	1,8
C(4):	{2,4},{2,5},{4,5}	I(4):	2,4,5
C(5):	{2,8}	I(5):	2,8
C(6):	{3,5}	I(6):	3,5
C(7):	{3,8}	I(7):	3,8
C(8):	{4,6}	I(8):	4,6
C(9):	{4,7},{5,7}	I(9):	4,5,7

Table 4: The cliques $C(1), \dots, C(9)$ of the graph Γ and the corresponding cliques $I(1), \dots, I(9)$ of the graph $\overline{G}$ in Example 2.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8		
	1	1	1	1	1	1	1	1	$\rightarrow$	max
(1)	1					1			$\leq$	1
(2)	1						1		$\leq$	1
(3)	1							1	$\leq$	1
(4)		1		1	1				$\leq$	1
(5)		1						1	$\leq$	1
(6)			1		1				$\leq$	1
(7)			1					1	$\leq$	1
(8)				1		1			$\leq$	1
(9)				1	1		1		$\leq$	1

Table 5: The linear program constructed using the independent sets $I(1), \dots, I(9)$ of the graph G in Example 2.



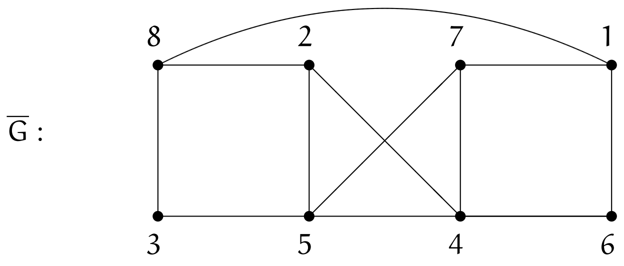


Figure 1: Graphical representations of the graph G and its complement $\overline{G}$ in Example 2.

2 The edge auxiliary graph

To a given finite simple graph $G = (V, E)$ we assign a new graph $\Gamma = (W, F)$. We call Γ the edge auxiliary graph of G. The nodes of Γ are the edges of G, that is, $W = E$. Let us consider two distinct nodes

$$w_1 = \{u_1, v_1\}, \quad w_2 = \{u_2, v_2\} \tag{1}$$

of Γ. Here the unordered pairs $\{u_1, v_1\}$, $\{u_2, v_2\}$ are edges of G. We construct the subset $X = \{u_1, v_1, u_2, v_2\}$ of V. Since the nodes w_1, w_2 are not equal, the set X has either 3 or 4 elements. If the subgraph induced by X in G is a clique in G, then we say that X is a qualifying subset of V. The nodes (1) are adjacent in Γ if the associated subset X is qualifying.

We are interested in the interplay between the cliques in the graphs G and Γ. If a subset U of V is the set of nodes of a k-clique in G, then the unordered pairs $\{u, v\}$, $u, v \in U$ are the nodes of an s-clique in Γ, where $s = k(k-1)/2$. In other words cliques in G give rise to cliques in Γ. Cliques in Γ also give rise to cliques in G. Let

$$w_1 = \{u_1, v_1\}, \ldots, w_s = \{u_s, v_s\} \tag{2}$$

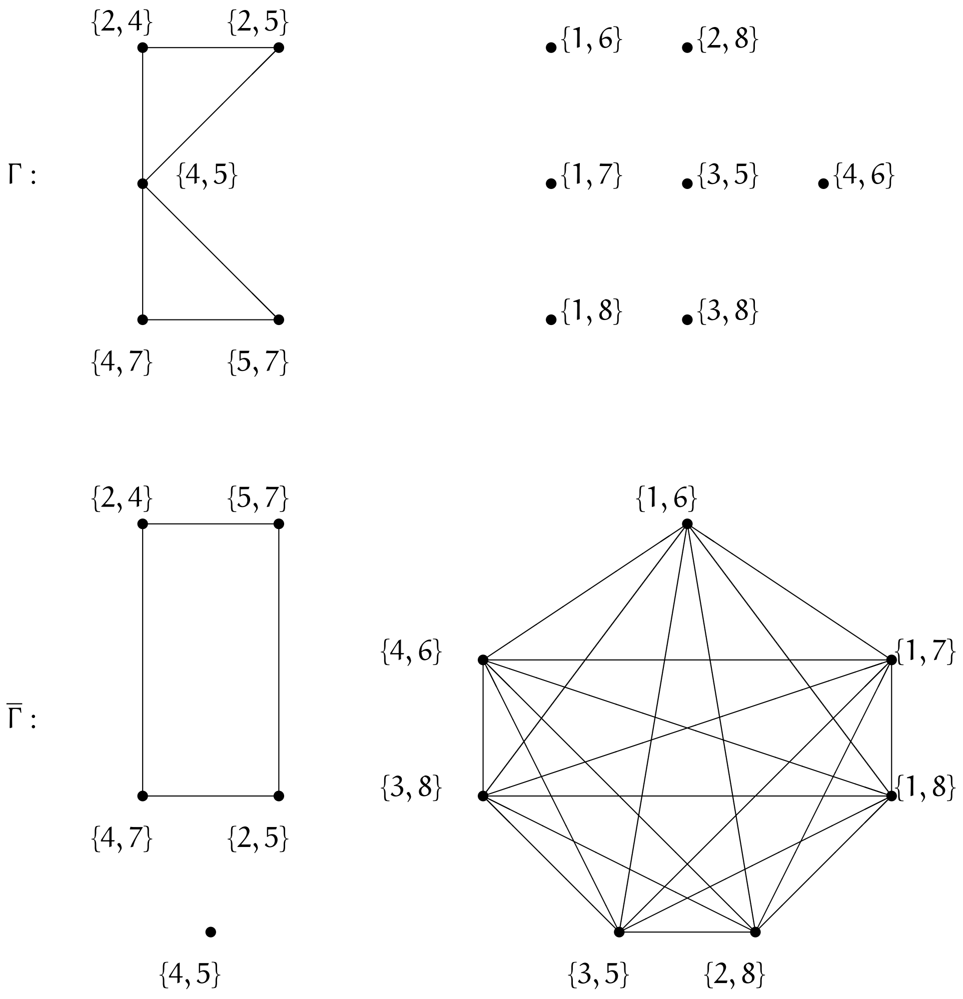


Figure 2: Graphical representations of the graph Γ and its complement $\overline{\Gamma}$ in Example 2. Each node of the graph on the right in $\overline{\Gamma}$ is adjacent to each node of the graph on the left. In order to avoid an overly cluttered picture we omitted these 35 edges.

parameter of G	number of nodes of G	number of nodes of $\overline{\Gamma}$	number of colors	estimate for $\omega(G)$
3	27	162	17	5
4	64	720	44	9
5	125	2 250	91	14
6	216	5 670	165	21
7	343	12 348	272	29
8	512	24 192	417	38
9	729	43 740	608	48
10	1 000	74 250	850	59
11	1 331	119 790	1 148	72
12	1 728	185 328	1 510	85
13	2 197	276 822	1 940	100

Table 6: Monoton matrices

be vertices of an s-clique in Γ and let

$$x_1, \ldots, x_k \tag{3}$$

be all the distinct elements among $u_1, v_1, \ldots, u_s, v_s$.

Lemma 1 *Using the notation introduced above the nodes* $x_1, \ldots, x_k$ *of* G *are the nodes of a* k*-clique in* G.

Proof. We show that the unordered pair $\{x_i, x_j\}$ is a edge of the graph G for each i, j, $1 \leq i < j \leq k$. As nodes of Γ on the list (2) are nodes of a clique in Γ it follows that there is an x_p on the list (3) such that the unordered pair $w'_1 = \{x_i, x_p\}$ is an edge of G. Similarly, there is an x_q on the list (3) such that the unordered pair $w'_2 = \{x_j, x_q\}$ is an edge of G. Using the fact that the nodes of Γ on the list (2) are the nodes of a clique in Γ we can draw the conclusion that the subset $X = \{x_i, x_p, x_j, x_q\}$ associated with w'_1 and w'_2 is qualifying. Consequently, the unordered pair $\{x_i, x_j\}$ is an edge of a clique in the graph G. (It is an edge of either a 3-clique or a 4-clique.) In particular, the unordered pair $\{x_i, x_j\}$ is an edge of the graph G. □

In the next lines we summarize the work flow of computing an upper bound for the clique number of a given finite simple graph.

(1) Using the given graph G we construct its complement graph $\overline{G}$.

parameter of G	number of nodes of G	number of nodes of $\overline{\Gamma}$	number of colors	estimate for $\omega(G)$
3	8	19	5	2
4	16	63	9	4
5	32	191	18	6
6	64	543	34	10
7	128	1 471	66	17
8	256	3 839	130	30
9	512	9 724	258	53
10	1024	24 063	514	96
11	2048	58 367	1 026	175

Table 7: Deletion error detecting codes

parameter of G	number of nodes of G	number of nodes of $\overline{\Gamma}$	number of colors	estimate for $\omega(G)$
6	15	60	20	5
7	35	210	35	8
8	70	560	56	11
9	126	1 260	84	21
10	210	2 520	120	30
11	330	4 620	165	41
12	495	7 920	220	55
13	715	12 870	286	71
14	1 001	20 020	364	91
15	1 365	30 030	455	113
16	1 820	43 680	560	140
17	2 380	61 880	680	170

Table 8: Johnson codes

(2) Using $\overline{G}$ we construct its edge auxiliary graph Γ.

(3) From Γ we construct its complement graph $\overline{\Gamma}$.

(4) We color the nodes of $\overline{\Gamma}$ legally. The color classes $C(1), \ldots, C(s)$ of the nodes of $\overline{\Gamma}$ are cliques in Γ.

(5) Using the cliques $C(1), \ldots, C(s)$ in Γ we construct cliques $I(1), \ldots, I(s)$ in $\overline{G}$. (We apply Lemma 1.) The cliques $I(1), \ldots, I(s)$ in $\overline{G}$ are independent sets in G.

(6) Using the independent sets $I(1), \ldots, I(s)$ in G we construct a linear program P.

(7) Solving the linear program P gives an upper bound for $\omega(G)$.

3 A small size toy example

In this section we work out a small example in details. Our intension is to illustrate the concepts and arguments we have seen earlier.

Example 2 *Let us consider the finite simple graph* $G = (V, E)$ *defined by its adjacency matrix in Table 1. The graph* G *has* 8 *vertices and* 16 *edges. The vertices are denoted by* $1, \ldots, 8$*, that is,* $V = \{1, \ldots, 8\}$*. A possible geometric representation of* G *can be seen in Figure 1.*

An inspection shows that $\omega(G) = 4$. We pretend that we are not aware of this fact and with full seriousness we carry out a procedure to establish an upper estimate for $\omega(G)$.

Table 1 shows the adjacency matrix of the given graph G. This is adjacency matrix is on the left. We constructed the adjacency matrix of the graph $\overline{G}$. This can be seen on the right. Possible geometric representations of these graphs are in Figure 1. Using $\overline{G}$ we constructed its edge auxiliary graph Γ. Table 2 shows the adjacency matrices of the edge auxiliary graph Γ of the graph $\overline{G}$ and its complement $\overline{\Gamma}$. Possible geometric representations of these graphs are depicted by Figure 2.

We used the simplest greedy coloring procedure to construct a legal coloring of the nodes of $\overline{\Gamma}$. Table 3 summarizes the procedure. The first column contains the nodes of $\overline{\Gamma}$ and the last column holds the colors of the nodes. The color classes $C(1), \ldots, C(9)$ of the nodes of $\overline{\Gamma}$ are nodes of cliques in Γ and are listed in

Table 4. This table also lists the independent sets $\mathtt{I}(1), \ldots, \mathtt{I}(9)$ corresponding to the cliques $\mathtt{C}(1), \ldots, \mathtt{C}(9)$ in the given graph $\mathtt{G}$.

We used these independent sets to set up a linear program. The linear program is given by Table 5.

4 Numerical experiments

The edge auxiliary graph Γ, we construct from the complement $\overline{\mathtt{G}}$ of the given graph $\mathtt{G}$, has as many vertices as many edges $\overline{\mathtt{G}}$ has. Then we locate a legal coloring of the nodes of $\overline{\Gamma}$ the complement of the auxiliary graph Γ. We carry out numerical experiments to see whether greedy coloring of such large graphs gives results that are useful in practical setting.

We use three infinite families of graphs as test cases. All of them are related to coding theory. The graphs associated with monotonic matrices are described in details in [6] and [7]. The graphs connected to deletion error detecting codes are presented in [5]. Finally the reader can find further material about the so-called Johnson codes in [3].

Table 6 contains the result of our computation related to the graphs associated with monotonic matrices. The first column contains the parameter of the graph $\mathtt{G}$ we work with. In the second column one can find the number of the nodes of $\mathtt{G}$. The third column holds the number vertices of $\overline{\Gamma}$ which is equal to the number of edges of $\overline{\mathtt{G}}$. The fourth column lists the number of colors used for legally coloring the nodes of $\overline{\Gamma}$. Finally, the last column shows the upper estimate we get for $\omega(\mathtt{G})$.

Tables 7 and 8 can be interpreted in an analogous way. We may conclude that the procedure we proposed to set up a linear program to estimate the clique number of a given graph can be carried out safely in connection with relatively large graphs that occur in practical clique search problems.

Acknowledgements

I would like to express my thanks for the anonymous referees.

References

[1] I. M. Bomze, M. Budinich, P. M. Pardalos, M. Pelillo, The Maximum Clique Problem, Handbook of Combinatorial Optimization Vol. 4, Kluwer Academic Publisher, 1999. ⇒123

[2] M. R. Garey, D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman, New York, 2003. ⇒123

[3] J. Hasselberg, P. M. Pardalos, G. Vairaktarakis, Test case generators and computational results for the maximum clique problem, *Journal of Global Optimization* **3** (1993) 463–482. ⇒132

[4] C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley Publishing Company, Inc., Reading, MA 1994. ⇒123

[5] N. J. A. Sloane, *Challenge Problems: Independent Sets in Graphs.* `https://oeis.org/A265032/a265032.html` ⇒132

[6] S. Szabó, Monotonic matrices and clique search in graphs, *Annales Univ. Sci. Budapest., Sect. Computatorica* **41** (2013), 307–322. ⇒132

[7] E. W. Weisstein, Monotonic Matrix, In: *MathWorld–A Wolfram Web Resource.* `http://mathworld.wolfram.com/MonotonicMatrix.html` ⇒132

Received: April 1, 2021 • Revised: May 16, 2021

Acta Univ. Sapientiae, Informatica **13,** 1 (2021) 134–179

DOI: 10.2478/ausi-2021-0007

Improving productivity in large scale testing at the compiler level by changing the intermediate language from C++ to Java

Izabella Ingrid FARKAS
Eötvös Loránd University,
Budapest, Hungary
email: `Ingrid.Farkas@inf.elte.hu`

Kristóf SZABADOS
Ericsson Hungary Ltd.,
Budapest, Hungary
email:
`Kristof.Szabados@ericsson.com`

Attila KOVÁCS
Eötvös Loránd University,
Budapest, Hungary
email: `Attila.Kovacs@inf.elte.hu`

Abstract. This paper is based on research results achieved by a collaboration between Ericsson Hungary Ltd. and the Large Scale Testing Research Lab of Eötvös Loránd University, Budapest. We present design issues and empirical observations on extending an existing industrial toolset with a new intermediate language[1].

Context: The industry partner's toolset is using C/C++ as an intermediate language, providing good execution performance, but "somewhat long" build times, offering a sub-optimal experience for users.

Objective: In cooperation with our industry partner our task was to perform an experiment with Java as a different intermediate language and evaluate results, to see if this could improve build times.

Computing Classification System 1998: D.2.2, D.2.3, D.2.6, D.2.9, D.3.4
Mathematics Subject Classification 2010: 68N20
Key words and phrases: C/C++, Java, IDE, compiler, Titan, developer productivity, performance, efficiency, incremental compilation, industrial experience, TTCN-3

[1]An intermediate language is a language which the input program is translated to, by using the already existing compilation toolchain, reaching the executable state.

Method: We extended the mentioned toolset to use Java as an intermediate language.

Results: Our measurements show that using Java as an intermediate language improves build times significantly. We also found that, while the runtime performance of C/C++ is better in some situations, Java, at least in our testing scenarios, can be a viable alternative to improve developer productivity.

Our contribution is unique in the sense that both ways of building and execution can use the same source code as input, written in the same language, generate intermediate codes with the same high-level structure, compile into executables that are configured using the same files, run on the same machine, show the same behaviour and generate the same logs.

Conclusions: We created an alternative build pipeline that might enhance the productivity of our industry partner's test developers by reducing the length of builds during their daily work.

1 Introduction

Nowadays, the usage of software – developed by 11 million professional software developers ([4]) – belongs to the everyday life of our society. Software helps in navigating to destinations, communicating with other people, driving the production, distribution and consumption of energy resources. Software drives companies, trades on the markets, takes care of people‘s health. In order to support the growing demand for assuring quality ETSI[2] designed the TTCN-3[3] ([16]) standardised notation specifically for testing. TTCN-3 is important in many industrial domains. It is used for testing telecommunication systems ([13, 14]), IoT[4] systems ([45]), ITS[5] systems ([15, 34]), oneM2M systems ([25]), security in the industry ([3]), smart grids ([35]), etc.

As the products to be tested grow, so did their test systems written in TTCN-3, growing to millions of lines of code in size ([5]), having complex architectures ([36, 39]), showing code quality patterns ([27, 28, 37]) and evolution trends ([38]) very similar to those present in other programming languages. The size and complexity of these huge test systems lead to long lasting build and development iterations (also seen for various projects ([23]) using other programming languages). At our industry partner in a full build scenario – Figure 1(a) – building some of these large-scale test systems could require

[2]European Telecommunications Standards Institute
[3]Test and Test Control Notation 3
[4]Internet of Things
[5]Intelligent Transportation Systems

approximately 20 minutes from zero even on server-grade, 28 core machines allocated for such tasks only. In an incremental, iterative scenario – Figure 1(b) – even if every developer worked on such a dedicated server, it would still take a few minutes to check the effect of a code change. Clearly, neither scenario means "fast feedback" for the developers and allocating a separate build server for each developer is not cost-efficient.

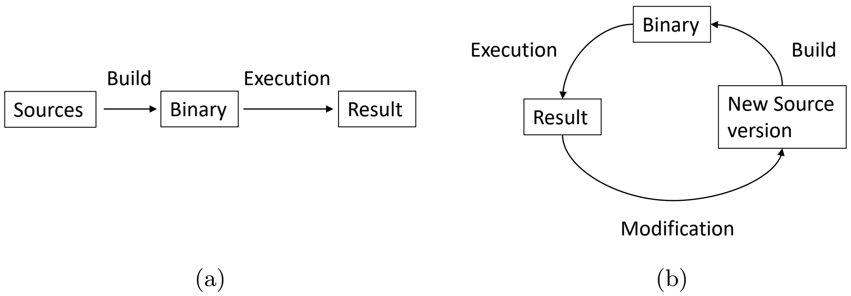


Figure 1: Building from source code. In the full build scenario 1(a) the full code base is built from zero so that the developer is able to execute the produced binary to see how it works. In the incremental build scenario 1(b) developers perform some modifications to the code, re-build the binary (incrementally, using the already built parts), execute the produced binary to see how it works and based on the result produce further modifications to the source starting the loop again. The length of these loops (build and execution time together) highly impact the development speed. Please note that applying a Continuous Integration machinery the full build can also be a loop, determining how fast/often developers can get feedback on their changes.

In this paper we present how a build process was created for our industry partner's tool that uses Java as the intermediate language. This new process is based on the existing one that used C/C++. We were tasked by our industry partner for performing and experimenting on the effects of adding a new intermediate language to our industry partner's toolset. We found that our solution is a good way to support the process of building large-scale TTCN-3 test systems by enabling the developers to get faster feedback after a code change.

Some of the reasons why our industry partner (at around 2000 as shown in [40]) chooses to create a build process that uses C/C++ as an intermediate language:

- TTCN-3 defines itself to be a language operating on an abstract level, creating abstract test suites, offering standardised mapping of its runtime ([12]) and control ([17]) interfaces to Java, ANSI C, C++[6], C#.
- Translating from TTCN-3 to C/C++ and lets platform dependent C/C++ compilers generate the binary, reduces development cost by supporting several platforms (operating system) handled by the C/C++ compilers.
- In case of a new C/C++ compiler version, used in the build pipeline on a platform, improves its code optimisation mechanisms (as part of the build process) then users of our industry partner's tool will also benefit from it on that platform.
- Translating from TTCN-3 to C/C++ not just offers high execution speeds, but as the code generated is statically typed, the type correctness of the operations is checked by the C/C++ compiler as well.

While for the purpose of the experiment presented in this article Java is just another programming language, in practice, our industry partner chooses Java for its added benefits:

- As the part of the toolset running on Eclipse was already written in Java (syntax and semantic analysis, code quality checking, refactoring, architecture visualisation), generating Java code keeps their systems consistent, and builds on already existing expertise.
- Eclipse users have access to a free and mature Java Development Environment that can be used to compile the generated Java code to an executable format, supports users in executing these projects, offers extensive support for developing code (for example developing test ports), etc...
- Opposed to C/C++ codes, that require to build separate binaries for the different operating systems, Java is running on virtual machines available on most platforms, and only one build is needed during the release, and CI procedures to support those systems (and maybe even ones not yet supported) automatically.

Our contribution is unique in the sense that with our extension there are two different build and runtime environments (see Figure 2) available for the

[6]Our industry partner uses a proprietary mapping better suited for their specific needs

developers. The input to both can be the same TTCN-3 source code. In order to unburden the conversion of the already existing platform dependent code parts, we kept the new runtime architecture, generated code and conversion methods as close as possible to the original one. The execution of the built "binaries" can be configured with exactly the same configuration files, performed on the same machine, creates same log messages, and in parallel mode, connects to and is directed by the same main controller via the same internal protocols. This puts us in the position of being able to study the effects of both environments on the development of the same user product.

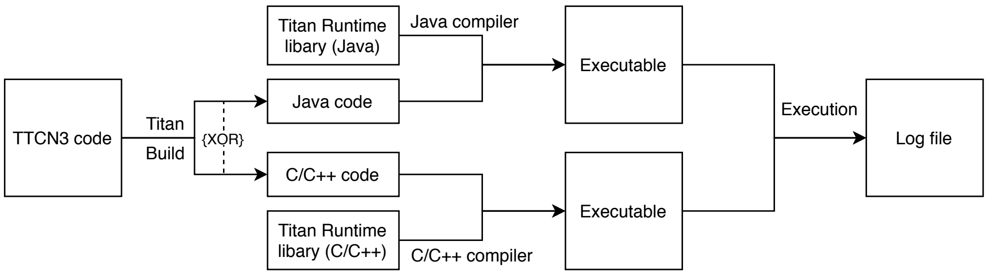


Figure 2: The figure presents the relationship of the 2 build and runtime environments. The input is the same for both the Java and C/C++ build pipeline and the execution of the binaries produce the same log files. The runtime libraries are the implementation of the base types, additional functions, etc. of the TTCN-3 language.

Our findings indicate that while C/C++ still might have performance advantages over Java in some situations, Java enables faster development iterations, and at least in our case, provides acceptable execution performance.

This paper is organised as follows. In Section 2 we present earlier work related to our subject. Section 3 presents our design and Section 4 some general implementation details. Section 5 shows our measurements and comparisons on build times and execution performance on both the C/C++ and Java sides. Section 6 deals with the limitations and Section 7 deals with the validity of our results. Finally, Section 8 summarises our findings.

In Section A in the Appendix we provide a short introduction to TTCN-3 and in Section B we show more details to our measurements.

2 Related Work

The examined toolset, Titan[7], has actively been developed and used in the last 20 years. During this time it has already gone through the usual steps of build time improvements: optimisation, experimenting with Continuous Integration configurations, better hardware, incremental build support, caching, etc. Our industry partner's experience showed that in the case of Titan's build process the full length of the compilation mostly came from using C/C++ as the intermediate language (in the projects shown in sections 5.2 and 5.5 this is measured to be approx. 96-97%). In order to improve the build process our university laboratory were asked to perform an experiment for compiling to a different intermediate language and for creating an alternative build process for the users of our industry partner's tool.

Although it could be said that we had to create a cross-compiler translating from TTCN-3 to Java and the runtime library it links to, this is only a part of the build process that creates value for the users. The users, in this case, write automated tests and execute them. From their point of view all of the operations required to make their tests run are "necessary technical limitations" that should be done in the background as fast as possible, as it engrosses time from the productive work. Also the term "cross-compilation" might induce misunderstandings: In the everyday usage of the term, a cross-compiler is a compiler that is compiling to a different hardware/software architecture that is not available, or does not support direct compilation on itself (like embedded devices). Compared to this, our solution takes the same input source code and creates an output executable that will be executed on the same machine (see figure 2). As such, we also find it better to look at our work as creating an alternative side of the toolset (instead of cross-compilation) as that better captures the scope and goal of the work.

When we tried to elicit information on the build time of different programming languages, we found that plenty of people engage in rather unhealthy discourses on how one programming language is "better" than the other, without any measurements for practical use cases. The general internet community does not seem to have easily available and thorough information about that. There also seems to be little academic interest on this topic. Maybe this is coming from the lack of deep connection between industry and academia. Maybe it is hard to find funding to run a 3 year-long experiment purely on

[7]https://projects.eclipse.org/projects/tools.titan

the academic side to reach functional equivalence in the implementation, while keeping industrial level code quality and coding practices.

At the same time long compilation times seem to be a widespread problem. While studying GitHub repositories of Java and Ruby projects Ghaleb et al. found ([23]), that 40% of the builds lasted more than 30 minutes. They suggest that developers should properly configure their Continuous Integration systems and use tools to identify cacheable spots. Others, like Abdalkareem et al. [1], tried to combat long-running builds by trying to identify changes for which the CI build might be skipped. While their results might make it possible to skip on average about 5% of the CI builds, in the iterative working scenarios we are investigating, this would still mean too many, too long-running builds, that hold back the productivity of the developers.

Reinholtz predicted ([33]) that Java not only can, but also will have better general performance than C++. He reasoned that Java can compile and re-compile the program as it executes and has access to runtime information not available to a traditional C++ compiler allowing to achieve better execution performance. Our measurements show that C++ still has an edge during the initial executions, however, for longer executions, Java can reach a measurably better performance in some situations.

Although they directly migrated a software from COBOL to Java, the closest to our task we could find was the experience of De Marco et al. ([8]). They explained that an earlier attempt at redeveloping the same application from scratch has failed. So they choose to migrate the application to a new platform and programming language using code and data translation approaches. They reported delivering an application functionally equivalent to the original and provided some useful remarks: (1) the new Java application still had to follow COBOL idioms and mainframe concepts, (2) they observed performance bottlenecks during the conversion, and (3) mentioned that even with the translation they needed to develop deep application understanding.

Batyuk et al. found ([6]) that native C applications running on the Linux layer of Android can be up to 30 times faster than identical Java applications running on the Java Virtual Machine of Android. For their measurement they sorted arrays of random integers with different sorting algorithms. Although this can be a way for performance measurement it is not something most people use their phones for.

Some researchers have already observed that even if Java is not always “better” than C++, it has other features which might make it as a reasonable choice. Amedro et al. ([2] compared the performance of a Fortran + MPI[8] and a Java implementation of the NPBs[9]. They found that “the overhead of Java is now acceptable when performing computationally intensive tasks”, but they also found that (in 2008) Java had scalability problems in communication-intensive benchmarks.

Gherardi et al. found ([24]) that using the server compiler option, Java was only 1.09 to 1.91 times slower in their tests aiming at robotics-related scenarios. They concluded that for their use cases, also taking into account the additional benefits offered by the language, “Java can be considered a valid alternative to C++”.

Taboada et al. analysed ([41]) the applicability of Java for High-Performance Computing. They also started with the assumption that Java lacks thorough evaluation on its performance, therefore they provided their implementation. They concluded that Java may achieve similar performance to natively compiled languages. Their most favourite features of the language (platform independence, portability, type safety, etc.) seem to them reasonable for the trade-off in the performance overhead.

Cook found ([7]) that a Java implementation of OpenMP[10] can be faster than a C implementation. They hypothesised that maybe the GCC implementation of OpenMP had deficiencies.

Nanz and Furia analysed ([31]) programs from the Rosetta Wiki, a site that collects solutions to programming tasks, implemented in various programming languages. This repository allowed them to compare several solutions, written in different programming languages, to the same programming tasks. Using statistical analysis (p-values, effect sizes) they found that “C is the king on computing-intensive workloads”, but in the case of “everyday” workloads “languages may be able to compete successfully regardless of their programming paradigm”. Later Furia et al. reanalysed ([21]) the same data with Bayesian techniques allowing them to draw more detailed conclusions.

While their works ([31, 21]) are consistent for their purposes, from our point of view their performance measurement had a bias for languages that compile into machine code: a) the compilation times were not measured/disclosed, which effectively gives an edge to pre-compiled languages over Just-In-time languages; b) as each execution was a standalone execution, this still might have penalised the Just-In-Time compiled languages where optimisations might only happen after a code part was executed several times (to have

[8]Message Passing Interface

[9]Numerical Aerodynamic Simulation Parallel Benchmarks

[10]http://www.openmp.org

profiling information for the optimisation). This way, pre-compiled languages could spend as much time as they wish on compilation and code optimisation at no cost in this setup, while Just-In-Time compiled languages might have been penalised for it if this starts during execution (because their compilation happens during execution), or have optimisations effectively disabled in some cases.

The papers described above presented some advantageous features of Java over C++, and in special cases showed that they have similar runtime performance. However, we could not find any recent academic studies which would present us the experiments on full build and execution times.

While working on this article it came to our attention that some hardware testing youtube channels like Gamers Nexus ([29]) started to publish code compilation test results for their hardware tests, in order to be able to measure the performance differences in massively multi-core modern CPUs. Their work offers useful, publicly available and detailed information for developers on what hardware to buy as their development equipment to improve their productivity. At the same time, by the very nature of their tests (building the same code with the same build system on several different CPUs), they will miss the opportunities presented in this article (where we reduce the build time on the same hardware, by using a different intermediate language, to reach a functionally equivalent executable).

3 Design

In this section we go into the design of our solution. We introduce the general context in which the design had to fit in. We explain the general design rules that we had to follow to be able to handle the long-lasting development process.

We don't detail the mapping of TTCN-3 elements to C/C++ or Java for two reasons: 1) as we tried to follow the mapping already existing on the C side, we dare claim as original ideas/observation only those cases where we had to differ in some way, 2) to save space in this article, as the mapping of TTCN-3 elements, runtime elements and API are already described in detail in the reference guide of the C side ([11]) and the reference guide of the Java side ([10]). We also made an example package available ([20]) containing a "Hello World" and some simple types in TTCN-3, together with the C/C++ and Java codes they compile into.

3.1 Context

To the best of our knowledge, at present TTCN-3 is the only internationally standardised notation designed specifically for testing which is also used frequently in the industry. Originally, Titan was able to translate TTCN-3 using C/C++ as the intermediate language ([40]). We were tasked to extend the part of Titan that runs on Eclipse to be able to generate Java code and write the necessary runtime libraries. As the examined tool stands behind many applications our solution might have a high impact, and at the same time, is also limited in how far we can drift from the current setup incentivising us to keep the architecture as close as possible to the original one.

Compared to De Marco et al. ([8]), we already had a deep understanding of the behaviour of the already existing code. Our team had a profound experience with Titan, either being its system architect, or being the member of the Large Scale Testing Research Lab.

Almost all of the needed tests had already been available for us before starting the development. Titan uses TTCN-3 code to test how well it can build and execute TTCN-3 code. As the new extension had to support the same input language, resulting in the same behaviour and output, and needed to conform to the same tests, our industry partner selected a set of their already existing tests to test our extension. We had to be able to successfully build and execute these tests, generate the same runtime logs during execution to show that the result of our experiment can be functionally equivalent to the existing solution.

We would like to highlight that we consider functionally equivalent the C/C++ and Java side if the logs, of the executions of the same tests, are the same since the logs contain detailed information together with the result (none, pass, inconc, fail, error) of the test cases. Keeping the architecture and the Java implementation as close as possible to the C/C++ architecture and implementation also contributed to guarantee this functional equivalence.

3.2 General design approach

It was clear from the beginning, that this experiment was going to take years, so we had to establish some general design rules that would govern our work and accept that certain limitations were reasonable to keep the timeframe.

Instead of inventing a new concept, we decided to stick to the already existing build and runtime structures. The following design decisions were made: (1) maximize the speed of work, (2) minimize the risk of running into major blocking issues, (3) provide the users of our industry partner with the option to choose between tools that are as much as possible identical in their functionality, (4) minimize the cost of converting the already existing platform dependent code (testports and external functions). We also decided to leave possible Java-specific optimisations (that were not necessary to provide the same functionality) for later.

3.3 Performance optimisation shifted to the end

It is important to understand that when building a compiler, most of the infrastructure must be in place and language support must be implemented, before any kind of relevant performance measurement, needed for optimisation, can be done. For example:

- The concept of the compiled code, the runtime libraries and how they relate to each other must exist.
- The compiler needs to be able to generate the classes that represent the TTCN-3 modules.
- The compiler must also be able to generate code for functions and function calls.
- The runtime must have a decent implementation of the classes representing the used types, so that they could be used in the generated code.
- The compiler must be able to generate code for variables of these types, for their instantiation, for checking their values, etc...
- The compiler must also be able to support statements that represent loops, assignments, variable definitions, etc...

Already, reaching the minimum level to do some performance measurement (e.g. simple operation in a loop) requires substantial work, and this might still not be indicative of the performance characteristics of the final tool, as that will need to support much more features with possibly different structures. Hence, as the first step, we decided to "just rewrite" the existing C/C++ code to a functionally equivalent Java code, and optimise it further only after thorough testing.

3.4 Handling long development timeframe

It took us 3 years to reach a stage where we could claim to have reached functional equivalence on the tests we received from our industry partner. This is of course not complete functional equivalence to the industrial tool, but a transition point when our experiment can be turned into industrial product to reach complete functional equivalence.

To make sure we are on the right track we used parts of the tests we received from our industrial partner. Throughout the development we have seen that the build times were showing promising results (for the limited set of elements supported at those times) and that we did not run into problems that could have blocked this project.

To handle the challenge of not yet supported features and missing code parts, we used FIXME and TODO tags in the compiler and Dynamic Testcase Errors (DTE) in the runtime. Initially, we generated a syntactically erroneous FIXME string into the code for every statement, type and definition telling what is missing. Also, functions in the runtime libraries reported DTE when executed. Putting them together:

1. When a new branch was implemented in the code generator or the runtime, the FIXME tag was moved into the still missing branch. In this way, we could effectively reduce the scope of the FIXME tags and allow testing the executable paths.

2. When a feature was first implemented in the compiler to generate valid code, the library function still reported a DTE during execution reducing the scope of the FIXME tag effectively.

3. When a design element for a type, statement or a feature was fully implemented, all tests had to be run to pass.

This strategy enabled us precise tracking of what was still missing, since those tests didn't compile or run into errors.

The TODO tags were used similarly as the FIXME tags, marking improvement ideas that will be implemented later.

4 Implementation

In this section we go into the technical implementation details of which features/concepts of the original C/C++ conversion could be reused, and which needed to be done differently in Java to succeed. Finally, we show a way how a compiler in an IDE might offer better performance than a traditional command-line compiler.

4.1 Concepts that could be reused from the C/C++ side

As we tried to stay as close to the original architecture as possible, there were several points/concepts that could be reused on the Java side without any problems.

- On the C side, each module was generated as a separate `.hh` and `.cc` file with a name generated from the module's name. The Java side kept the concept, generating a separate `.java` file from each module, using the same name conversion procedure.
- On both the C and Java sides non-synonym types were generated as a class representing the value version of the type and a class (with `_template` postfix in its name) representing the template version of the type.
- The runtime libraries of Titan on both sides contained the implementation of the same built-in types, runtime functionalities and additional functions.
- The implementation of each type in the runtime library of Titan on both sides followed the same abstract logic, had similar abstract interfaces, access patterns, etc...

4.2 Concepts of the C side that needed to be adapted

While many of the concepts could be reused, we had to make some changes.

- In the generated `.hh` and `.cc` files of the modules the code was located in a namespace generated from the module's name. On the Java side, we generated a class from each module, and the definitions of the module would be members and static nested classes of this class.
- Synonym types were mapped on the C side onto `typedef` -s. On the Java side we either used the referenced type's name instead of the new type's in the generated code or generated new classes to extend the referenced ones (if the user requested it or additional properties made it necessary).

- While the runtime libraries of Titan contained the same abstract contents on both sides, there were some minor differences: all code had to be placed into a package, the additional functions into a class of their own (on the C side they were outside namespaces), etc...
- While the implementations of each type in the runtime library of Titan on both sides were similar on an abstract level, in the actual implementations they differed slightly: Operator overloadings from the C side were translated as functions on the Java side (`operator[]` $\rightarrow$ `get_at()`, `operator==` $\rightarrow$ `operator_equals()`, etc...). Also, const and non-const access operators are important in the runtime, so they have been mapped to functions following the naming convention `get()` / `const_get()`, `get_at()` / `const_get_at()`, etc...
- On the C side each TTCN-3 `import` statement was translated to be a C/C++ `include` statement. As this statement is transitive in C/C++, it automatically ensures that all needed types are present. However, imports are not recursive in Java. As such, we decided not to generate code for TTCN-3 `import` statements, but whenever the code generator found that it needed to use a definition that was not defined in the TTCN-3 module (we are currently generating code for) it would generate the necessary Java `import` instead for the class that represents the module of that definition and generate the "usage" prefix with the name of its encompassing class (that represents the module it is located in). Please note, that this way the dependency hierarchy in the generated Java code can be different from that present in the generated C code. When the code compiled does not have unnecessary imports, all directly used modules will be referenced in both generated codes.
- In the generated code the C side used implicit type conversions[11] heavily. In the Java generated code, we had to use two different methods: (1) in the general case we generated the type conversions as static functions, or calls to constructors, or (2) we create several versions of the functions in the runtime library with both native and generated/runtime represented types, allowing to save the costs of the conversion.
- The Java side had a unique challenge not present on the C side. Nested classes in the .java files are translated into separate .class files. On MS-

[11]E.g. assignments between different types when the receiving type has a constructor with a single parameter of the type to be assigned to it or the assigning type has an implicit cast operator to the target type.

Windows however, where the file system is not case sensitive by default, this created a problem when the names of two or more embedded classes differ only in small/capital letters, as the file system will not allow generating the class files properly. To solve this problem an extra translation step checking the names of TTCN-3 types in the modules, before code generation, and the problematic names were postfixed with the starting offset of the definition (to make them unique). As this might create inconveniences for the users accessing these definitions from Java, a code smell checker was implemented in Titanium[12] that warns the user during semantic checking about similar type definition names in the same module.

- While C/C++ compilers can choose to ignore unreachable codes (and so it was also allowed in TTCN-3 in Titan), this is a semantic error in Java. Since we already had a code smell checker for such situations, we decided that this issue can be fixed best by the users: they can remove (the already reported) unnecessary statements from the TTCN-3 code.

- On the C side, the internal representation of string types used reference counting. This enabled efficient TTCN-3 assignment operations since the left-hand side could be set to reference the same internal structure used on the right-hand side (for example unsigned char array for hexstrings) and delay the actual copy till the variable of a string type needs to be modified and its internal structure has a reference count > 1. When we implemented this feature on the Java side we did not measure any significant performance benefit (outside of artificial use-cases) and decided to leave it out to keep the code as simple as possible.

- On the C side, static values of string types were generated into the code only once, as static objects, that are unique, initialised at startup time and only referenced during execution. On the Java side, each occurrence of such a string created a new object in the code. Once implemented, such an optimisation should improve the runtime speed of the Java generated code.

- The internal representation of some types on the C side used `unsigned char` array to store the elements (for example, `hexstring`, `octetstring`). As Java does not support such an unsigned type we had to use other

[12]The code quality analyser of Titan for checking for code smells, measuring code metrics and having support for extracting and visualizing architecture.

signed types, for example `byte` arrays instead. This also meant using additional code (for example using `& 0xFF` to make sure the value was understood correctly as an unsigned value) in certain operations.

- Java also has the limitation that functions cannot contain more than 65535 bytes of code ([43]). We run into this in two situations: (1) Some of the functions generated for a TTCN-3 union type with more than approx. 1670 alternative fields (in which case we had to generate helper functions each of which could handle 200 alternatives), or (2) we also ran into a 2.800+ lines long TTCN-3 function (where we had to ask the users to partition it into smaller parts as we could not find an easy way to automatically generate helper functions to it so far).

- When a new PTC[13] was created on a HC[14] the C side does a fork, both resulting in a parallel process for the new PTC and also copying the memory contents[15]. Java does not support this behaviour, so we decided to use threads instead. This way, while every PTC was a new process on the C side, they were new threads on the Java side. Plus, every static variable inside the runtime library on the C side was mapped to be a thread-local variable on the Java side. Thus, we do not need to start a new JVM and configure it for every PTC created, as the C side algorithms had already been designed to run in their own process, and the TTCN-3 language also does not allow direct access between two PTCs. This introduced minor slowdowns during access. As thread-local variables are currently implemented inside the JVM using map data structures with the threads as keys, accessing the value that should be seen in the current thread has the overhead of an extra data access.

For a detailed description of the mapping of TTCN-3 elements, runtime elements and API turn to the reference guide of the C side ([11]) and the reference guide of the Java side ([10]). To demonstrate the mappings we made a short example package available ([20]). This package contains a "Hello World" module and some simple types in TTCN-3, together with the C/C++ and Java codes they compile into, to illustrate the mappings between these language elements. For a more complete example turn to the regression tests ([19]) we

[13]Parallel Test Components are TTCN-3 concept for components created by the Main Test Component and running in parallel to each other.

[14]Host Controllers are instances of the executable test program.

[15]Actual implementations might use a copy-on-write solutions for performance.

used for our measurements, which can also be used to check the mappings of all supported language elements.

4.3 The benefits of being an IDE

In this subsection, we show one more element of our work, that goes beyond simple cross-compilation, to improve the performance of the build process. As the Java side was built into an IDE, that was able to track changes on a fine grain level to optimise semantic checking, we could use this information to improve the build times even before the compilation to Java happens.

On both the C and the Java sides the codes were generated on a per-module base. That is, first the code generator compiled the `string char*` and `StringBuilder` that represents the module in the target language (called S further), then compared it to the content of the already existing file. The string S was only written out if the target file did not exist, or had a different content. The mentioned process created a potentially large saving in compilation time: since most of the TTCN-3 modules do not change from one compilation to the next, their generated code will also not change, not even be touched. During the build process the C/C++ and Java compilers might notice that their input file has not been changed, and skip its compilation. It can be a large improvement, as comparing the contents of strings and files is negligible compared to compiling the file again (in larger systems the generated code can be several hundreds of thousands or millions of lines of code).

On the Java side, the compilation process could further be improved since it was part of the TTCN-3 IDE of Titan. As presented in the work of Oláh ([32]), the Designer part of Titan could analyze the code of TTCN-3 projects incrementally. Once the project was analyzed, and when the user edited something, the IDE was able to discover easily (and on a fine grain) the code parts whose status might have changed (being correct or erroneous in some way, or referring to a different target, etc...) and used this information to minimize the amount of code whose semantic information needed to be re-checked.

There is no need to generate the string S for modules which had already been compiled and their status has not been changed. Adding this technique (further referred to as IDE mode) to the new compiler, it had a substantial effect on the speed characteristics of compilation: as most daily changes can be reasonably assumed to affect the status of only a few code parts, the (same) file overwriting process can be saved for most of the dependent modules. For example, while a full compilation might have compiled all code, changing the signature of a function required recompiling only that module and the modules from where that function is called directly. While a change inside a function only required compiling the module it is located in.

Please note that while in this article we analyse this feature as something that can be turned on/off, in the product it will be always active.

5 Performance measurements

In the first part of this section we describe the input on which we performed the measurements together with the testing machines. In the second and third part, we present the build and execution performance. In the last part, we show the information we have on the scalability of our implementation.

5.1 Performance measurement environment

To measure and compare the performance of the C and Java sides we used the subset of the regression tests of Titan provided to us by our industry partner ([19]). These tests included:

- Tests for handling the values of all
  - Base types (`boolean`, `integer`, `float`, `objid`, `bitstring`, `hexstring`, `octetstring`, `charstring`, `universal charstring`);
  - Complex types (`records`, `sets`, `record ofs`, `set ofs`, `enumerations`, `unions`, the `anytype` type).
- Tests for template matching for all base types and complex types.
- Tests for basic statements.
- Two ASN.1 modules and two TTCN-3 modules that import them (to test importing from ASN.1).
- Tests for the predefined functions of TTCN-3.
- Tests for timers within testcases and used as a testcase timer to limit their execution duration.
- Tests for the `all from` language construct (for its special handling of values).
- Tests for the “RAW” encoding supported by the Java code generator.

We note that although the verdict tests were excluded from our measurements, they could also be compiled and executed. The reason was that when test verdicts other than pass are reported, the replication studies can be confusing.

Altogether 90 source files were used in the measurement of compilation and execution speed: 88 TTCN-3 files and 2 ASN.1 files. The package also contained configuration files, set to execute all control parts in the modules at a given amount of times. This package can be downloaded from [19].

From our point of view, these tests were input. Examining the output we could conclude that the implementation was functionally equivalent with the original compiler.

For the measurements, we used two different architectures (laptops).

1. Laptop 1 used for measurements was a HP EliteBook 840 G5, with an Intel® Core™ i5-8350U CPU, 16 GB RAM, using a SSD, running Windows 10 and the 3.0.7-1 version of Cygwin ([44]), GCC 7.4.0, Eclipse 4.4.2, Java SDK 1.8.

2. Laptop 2 used for measurements was a Lenovo ideapad Y700, with an Intel® Core™ i7-6700HQ CPU, 8 GB RAM, using a HDD, running Windows 10 and the 3.0.7-1 version of Cygwin, GCC 7.4.0, Eclipse 4.10.0, Java SDK 1.8.

Laptop 1 was expected to be the target architecture at our industry partner, Laptop 2 was used as a control platform to check our observations in a different setup ([22]). We had to use a control platform to account for the potential issues coming from Laptop 1 being a machine used at our industry partner. Users (test automation developers in this case) have limited control over it (the firewall software could not be stopped/deactivated/reconfigured for the duration of measurements, operators might trigger minor refreshes remotely, that would run in the background, etc.).

The source code used for the measurements is part of the ”CRL 113 200/6 R6A (6.6 pl0)” version of Titan ([46]), with the ”6.6.0” tag on github.

5.2 Build speed

To get a picture of the speed of the build processes for both the C and Java sides we set up and measured 4 uses cases.

1. Full build. In order to measure the speed of a full build we cleaned all generated files and ran the build process from scratch.

2. Incremental build with minor change. In order to measure how the system might perform in daily operations we also measured the build times after a small change in an already built project. For this reason, we changed in the 22nd line of the `TboolOper.ttcn` file the `true` value to `false` and back. This modification triggered the re-generation of the code for this module but did not re-generate code for any other modules. Also it made the "boolAssign" test report a fail verdict, but this was not an issue for measuring the compilation time.

3. Incremental build with inserting/deleting a testcase. In this use case we simulate a scenario when users write or delete a small testcase in a single file resulting in only one incremental build after this operation. For this purpose in the `TboolOper.ttcn` file we commented out the execution call of the `boolAssign` testcase from the control part, deleted the code of the testcase for one change and reverted this modification as another change.

4. Incremental build, refactoring in 5 files at the same time. In this use case we simulate a scenario, when users perform refactoring operations that create changes in 5 files at the same time. For this, we extended the `TcharOper.ttcn`, `TbitstrOper.ttcn`, `ThexstrOper.ttcn`, `ToctetstrOper.ttcn` and `TucharstrOper.ttcn` files with a new testcase called "dummy_test()" having an empty body and using the search & replace features of Eclipse we renamed it "dummy2_test" for one change and reverted it for another.

Full builds are expected to happen rarely (as they take a long time), but when developers wish to make sure that everything works from zero, they have to do full builds and it gives an upper bound for our measurements.

Incremental builds are expected to happen more frequent in practice. In fact, in Eclipse one of the default settings for builds is to "Build Automatically". That is, whenever a developer saves a file after a change, Eclipse starts an incremental build in the background. For builds on the C side users had to turn this off, as the duration of incremental builds was a lengthy and resource intensive process. Based on the numbers we have measured for incremental builds, on the Java side users should be able to leave this feature on and work interactively in the environment.

As the extended tool was an IDE running on Eclipse (supporting both C/C++ and Java code generation), we measured the compilation speed via the Eclipse's interface[16]. This method allowed us measuring the entire build duration, as the user would experience it. At the same time, it applied a common interface for both the build process generating the C/C++ code and the build process generating the Java code.

Internally, the build on the Java side used the mechanisms provided by Eclipse to build the generated Java files. Here we assumed that the process used all available processing power for a fast compilation. We considered the following settings as a form of configuration that might affect the build:

- The Eclipse in which the builds were done was launched with the "-Xms5178m" and "-Xmx5178m" options. In this way we could minimise the chance for garbage collections.

- Inside the Eclipse instance where the builds were done, we set the "Compiler compliance level" of the "JDK Compliance" options to 1.6 (compiling for Java version 1.6), as that is the minimal version of Java supported by the generated Java code.

On the C side, we used the processes and tools Titan offers for its users. The "-O2" flag was used to control the optimisation used by g++, make was called with "-j8" to enable parallel compilation, and inside the generated makefile the following rule was used to call g++ to translate the generated .c and .cc files into .o files[17]:

```
.cc.o .c.o:
          g++ -c $(CPPFLAGS) -Wall -O2 -o $@ $<
```

Where CPPFLAGS listed the files to be included from the runtime libraries of Titan.

To do statistically rigorous performance evaluation ([22]) we measured 50 compilations for all scenarios.

We could make several observations from the results presented in Tables 1(a) and 1(b).

The full build (case 1) was measured to be much faster on the Java side than on the C side, 14$\ast$ faster on laptop 1 and 8$\ast$ faster on laptop 2. Considering case 2 from the table the Java side was 35$\ast$ faster on Laptop 1 and 21$\ast$ faster

[16]To run our plugins from source we already had "Eclipse application" style launch configurations set up. We selected the "build/invoke" and "debug" options for the measurements on the launch configuration's "Tracing" tab. This way, for every build invoked in the launched Eclipse instance, we got debug logs on the duration of the build in the Eclipse instance the source code was located in.

[17]Please note that these are the default and recommended settings of our industry partner

	Tool	case 1	case 2	case 3	case 4
C side					
	avg	153.40	34.51	35.15	35.20
	min	150.25	29.06	33.44	32.63
	max	159.49	41.64	37.60	36.99
Java side					
	avg	10.95	0.98	0.86	0.87
	min	9.51	0.88	0.82	0.83
	max	18.31	1.30	1.16	0.97
Java side + IDE					
	avg		0.15	0.18	0.34
	min		0.08	0.15	0.31
	max		0.24	0.22	0.39

(a) Laptop 1

	Tool	case 1	case 2	case 3	case 4
C side					
	avg	224.52	25.73	25.74	26.38
	min	221.29	25.09	24.65	25.52
	max	234.08	26.72	27.47	27.26
Java side					
	avg	28.12	1.23	1.29	1.64
	min	26.43	1.09	1.18	1.59
	max	29.94	1.52	1.44	1.74
Java side + IDE					
	avg		0.3	0.32	0.74
	min		0.25	0.29	0.7
	max		0.38	0.44	0.82

(b) Laptop 2

Table 1: Build speeds measured on Laptop 1 1(a) and Laptop 2 1(b) in seconds. Case 1 is Full build, Case 2 is Incremental build with a minor change, Case 3 is Incremental build with inserting/deleting a testcase, Case 4 is Incremental build with refactoring in 5 files at the same time

on Laptop 2. When the IDE mode was used, the speedup increased to $230*$ on Laptop 1 and $85*$ on Laptop 2.

The speedups were different, but were present on both laptops in all scenarios used for measurements. It would be reasonable to assume that the HDD in Laptop 2 might have created the overhead in both cases (access time and read/write speed), compared to the SSD in Laptop 1.

Our observations shows that the effort for reducing the build time was successful. We could also see how users benefit from the IDE mode: knowing what the user has edited it is possible to calculate the set of code pieces needed for the re-analysis.

In order to understand better the importance of the measurements it is beneficial to look at the development activity from the viewpoint of the user ([26, 42]):

- 0.1 second is the limit where the user still feels the interface to be reacting instantaneously.

- $0.1 - 1$ second is a "stammer", when the user's flow of thought is not interrupted, but the delay is noticeable. User's do loose the feeling of operating directly with the interface.

- >10 seconds is “disruption”, this is the limit of keeping the user’s attention focused. User’s might wish to do something else while the computer is working, leading to even longer non-productive periods of time.

Based on the above classification we found that the user experience of builds on the C side fall into the “disruption” category, on the Java side into the “stammer” category, keeping user’s attention un-interrupted. In best case scenarios the user experience can even be instantaneous on the Java side.

5.2.1 Some interesting facts and observations

1. On the C side for this project, in full build, the translation from TTCN-3 to C/C++ files took approximately 6 seconds, the translation of C/C++ files into .o files approximately 132 seconds, linking the object files took approximately 15 seconds. Altogether, approximately 96% of the build on the C side was coming from using C/C++ as the intermediate language.

2. On the Java side for this project, in full build, the translation from TTCN-3 to Java files took approximately 1.2 seconds. Here, approximately 89% of the build on the Java side was coming from using Java as the intermediate language.

Although we lack the necessary skills to analyse the internal procedures done during build in GCC and Eclipse’s Java toolset, we can still offer some intuitive reason for this difference in build speeds:

- In C/C++ the platform specifics of the target system has to be taken into account during build time. In Java this is not done during build, as the actual execution platform is only known during execution.

- In C/C++ all optimisations have to take place during build, before execution. As Java is a Just In Time compiled language, most of the optimisation work happens during execution.

5.3 Execution speed

The execution speed of the generated C/C++ binary was measured on both laptops in a Cygwin bash shell instance using the `time` command.

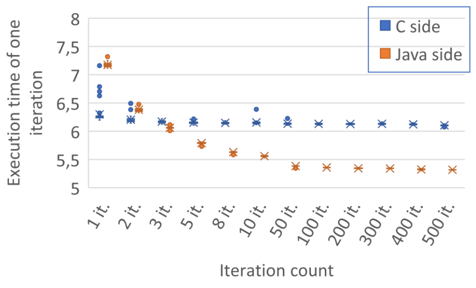
To measure the execution speed of the Java code, as it was executed from within Eclipse, we had to extend the generated code of the main class used for single-mode execution `Single_main`. The very first Java line to be encountered was `long absoluteStart = System.nanoTime()` (this was executed before any test-related code) and the last line calculates the elapsed time as `(System.nanoTime - absoluteStart) * (1e-9)` and outputs to the standard output (this was executed after all test-related code is properly terminated).

To see how each side performed on a longer run we decided to run and measure different amount of iterations of the same test set. A single iteration means that in the configuration file each module's control part was executed. Each control part in a module executed the testcases in that module, resulting in each testcase was executed exactly once. The 500 iteration means that we had a configuration file which listed the contents of the single iteration 500 times. This was not a good simulation for complex tests running for days, simulating complex network components and user, while sending millions of messages per seconds, but the best option we had to understand how the Java side we created might perform in such situations (the effects of Just In Time compilation, garbage collection, whether we had memory leaks, etc...).

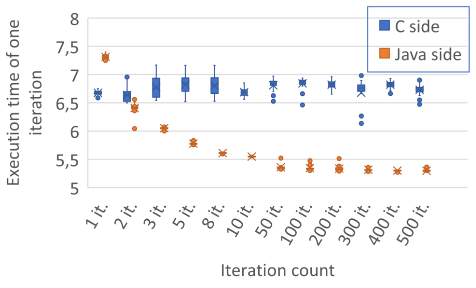
Since these tests also contained tests on how Titan was handling timers, we decided to perform two different measurements. The "with timers" means that we were executing the tests as they were. The "without timers" means that the testcases testing timers were commented out from their module's control parts. More precisely, in the `TcontrolTimer.ttcn` and `TlostTimer.ttcn` files we put in comment the content of the control part. This eliminated the part of the execution that was not beneficial for performance testing and also reduced the number of features to be tested. For each iteration size, both with and without timers, we measured the execution time 50 times.

Performance measurements (see Figures 5 and 6 in the appendix) showed similar results on both machines for high iteration counts, in the same measurement modes. For low iteration counts Laptop 1 executed the C side code and Laptop 2 the Java side code somewhat faster at identical settings (see data in [18]).

Figures 3 and 4 (showing the per iteration times on Laptop 1 and 2) help emphasise some interesting observations. In the first iteration, we could see that the C side is usually faster in both measured ways. The C side also had an approximately constant per iteration execution time for all iteration scenarios. The Java side seemed to "warm-up" as the number of iterations increased reaching a constant per iteration execution time after approximately 50 iterations. On Laptop 1, without timers, the iteration time (initially 2.37s) decreased to approx. 0.42s on average which is 83% reduction. Java finished the

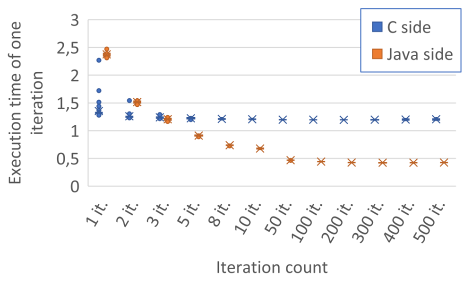


(a) Laptop 1

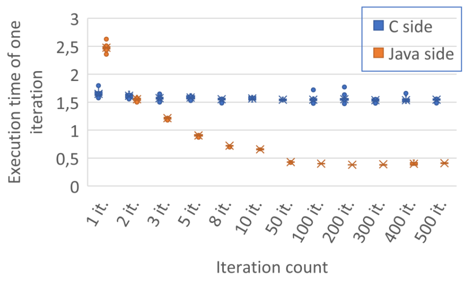


(b) Laptop 2

Figure 3: Box & Whisker chart of the Per Iteration Execution times with Timers.



(a) Laptop 1



(b) Laptop 2

Figure 4: Box & Whisker chart of the Per Iteration Execution times without Timers.

500 iterations in 390 seconds which was 65% faster than the C side. On Laptop 1, the Java side was running faster than the C side from the 3rd iteration, on Laptop 2 from already the second iteration.

We analysed our measurements with the statistical package R (see table 2). We applied t-tests (Welch two-sample) and Bayesian analysis (BESTmcmc described in [30]). Both showed a clear difference between the C and Java sides for the same iteration counts. BESTmcmc also showed how the C side was faster in one iteration, but fell behind the Java side for iteration counts larger than two for both laptops (see tables 2(a), 2(b)). For the 2-iteration case there was a difference between Laptop 1 and Laptop 2: the Java side was faster on Laptop 2 but slower on Laptop 1, compared to the C side.

Measurement \ Iteration	1, 2	3	5 - 500
with timers			
p	$< 2.2e-16$	$< 2.2e-16$	$< 2.2e-16$
muDiff	0.0	100.0	100.0
without timer			
p	$< 2.2e-16$	$< 1.523e-10$	$< 2.2e-16$
muDiff	0.0	100.0	100.0

(a) Laptop 1

Measurement \ Iteration	1	2	3 - 500
with timers			
p	$< 2.2e-16$	$< 2.2e-16$	$< 2.2e-16$
muDiff	0.0	100.0	100.0
without timer			
p	$< 2.2e-16$	$< 2.756e-14$	$< 2.2e-16$
muDiff	0.0	100.0	100.0

(b) Laptop 2

Table 2: t-test results on execution times (p value is reported by t.test in R doing Welch Two Sample t-test, muDiff is the $>compval value of muDiff reported by the summary of BESTmcmc result)

We note that, although so far we have only aimed at providing the necessary functionality, not execution performance, according to our measurements our tool might be also useful in long-running testing scenarios to improve the runtime performance of tests.

Recall that both Java and C sides were built from the same TTCN-3 code, had to parse the same configuration file, run on the same machine and generated the same amount of log files, so the measured difference in speed could not be explained by a different workload on the abstraction level of executing the testcases. However, while the static compilation of the C side might have resulted in faster code loading-up, the Java side might have also needed to start the JVM and/or perform the first iteration in interpreted mode (adding the JIT compilation as an extra overhead) [18].

5.3.1 Some interesting facts and observations

1. We could see the largest standard deviation of the measured execution times in case of the 1-iteration executions. In case of Laptop 1, the worst 1-iteration execution times of the C side were similar to the best Java execution time for the same scenario.

2. In the case of 1-iteration on Laptop 2, there were no huge extremal values. The deviation increased having multiple iterations compared to Laptop 1 in case of "with timers" (Figure 3).

[18]One more technical difference was that the C side used flex+bison for parsing the configuration files, the Java side ANTLRv4. The startup time could also be explained by the different performance characteristics of LALR(1) parsing used by Bison and LL* used by ANTLRv4, but as the input in our case was very simple, this was unlikely.

3. We could see that the per iteration time on Laptop 2 was larger than on Laptop 1 for the C side, however, on Java side, they were very close to each other.
4. A 500-iteration long execution was executing $882,500$ testcases.
5. A 500-iteration long execution generated a log file of 1.7GB[19].
6. The execution with timers was executing 1765, while the execution without timers executed 1757 testcases.

While in our research we did not do a deep analysis of why we observed such different performance curves we can still offer some intuitive reason that can explain them:

- In C/C++ the optimisations happen during build time, the built executable does not undergo changes during execution. As Java is a Just In Time compiled language, the performance of the application can be optimised during execution time, possibly leading to better performance over time.
- Parallel hardware architectures might also play a role. As the software architecture used by our industry partner is single threaded, on the C side memory deallocation has to be done by the same thread doing the business logic. In Java, even though the business logic is still single threaded, garbage collection can happen at any time after a resource is no longer in use delaying the cost of deallocation and the garbage collector itself might run on a CPU core different from the business logic, eliminating all work that CPU core would have to do to free up memory.

5.4 Build + Execution iterations

We have already presented our observations separately on the improvements in build time and the execution speeds on our test project. It is also interesting to take a look at these observations from the point of developer iterations, where we are interested in how long a single build + execute iteration takes. This is the duration developers need to wait from changing the source code, to being able to analyse the effects of the change.

[19]This cost was present on both sides during the execution.

For this we decided to check how long it would take for a build and execution to last, for 4 different build scenarios[20] and all iteration numbers measured so far, by adding together the average build time of the given build type and the average execution time of the given execution with a given iteration number, presented in the Appendix in Table B.

Please note that these are only theoretical durations, the real world durations could be much bigger. In the case of the incremental build and single iteration execution combination, on the Java side this operation is so fast ($2,49$ seconds) that simply the act of moving the mouse pointer after invoking the build to the toolbar of Eclipse to start the execution could substantially increase it. At the other end of the spectrum, full build and 500 iterations on the C side, we see approximately 53 minutes where the developer is bound to start doing something else either leaving their computer, or slowing it down with some other activities.

5.4.1 Some interesting facts and observations

1. For all build scenarios and iteration number combinations (and on both laptops) we observe that the Java side performs better. Having the shortest theoretical duration between a change by the developer and having the execution results ready to be analysed.

2. In case of full builds and executions with timers the improvement is at least approximately 140 seconds, increasing to approximately 535 seconds for the largest iteration number on Laptop 1 (respectively 88% to 16% improvement).

3. In the case of incremental builds and executions with timers, the improvement is at least approximately 33 seconds, increasing to approximately 424 seconds for the largest iteration number on Laptop 1 (respectively 82% to 13% improvement).

5.5 Scaling up for larger projects

In order to see if our solution would scale up for larger projects we asked for some help from our industry partner. They have provided us with another large scale project and one of their system architects could spend a day on providing us with some measurements on this project on a build server used by

[20]We leave out the incremental build on the Java side without the IDE mode, as the users will not be able to turn off IDE mode.

our industry partner. The aim in this exercise was purely to see if our solution would be able to scale up. We disclose both our numbers and the numbers we got from our industry partner, but note that at that point in time and implementation completeness, this was not a core part of our observations, only a look ahead. The 3 years of development was not enough for our group to reach full functional equivalence and for many of these numbers, we had to rely on the information given to us by our industry partner, where we had little control on the way the measurements were done.

The project used had 1275 modules (1143 TTCN-3 and 132 ASN.1), 1.24 million LOC at the time of measurements[21]. Our Java code generator generated 930 MBs of .java files from it. Since it was a large codebase, this project seemed to be appropriate to see if our method would work on large scale projects. There were 2 issues unique to this measurement, that we could not overcome for the amount of work needed:

1. At that time, our implemented compiler supported only the "RAW" encoding, while the examined project on the C side used several others. This means, that on the C side somewhat more code was generated. Approx. 18.93% of the code was used to support the other encodings.

2. To compile the project, we created dummy implementations (empty function bodies) of all required platform-dependent testports and external functions (proper implementation would have required a solid background on protocols and a large amount of time). The creation was made by hand and constitutes only a small amount of code compared to the 930MB generated codes. Hence, we didn't expect them to greatly impact the scaling of the build performance.

We performed 3 measurements for full build and 10 for incremental on Laptop 1, with the Java side only, as this project did not compile on Cygwin on the C side due to missing libraries.

The full build of the project took approximately 741 seconds (see Table 3). The time needed to incrementally re-build the project was approximately one to five seconds, depending on how many Java files had to be re-generated for the change. All 8 logical processors were heavily used during the builds.

To be able to compare build times of the C and Java sides of this project, a system architect of our industry partner (Eduard Czimbalmos) was asked to perform the measurements on one of their industrial build servers. The build

[21]More information is available on this project at [38, 39].

Scenario	build time
Laptop 1 full build	741.70
Laptop 1 incremental build (4 files are re-generated)	3.27
Laptop 1 incremental build (1 file is re-generated)	1.28

Table 3: Large industrial project build times on Laptop 1 (Java only, in seconds)

server characteristics used for the measurements were: Intel® Xeon® E5-2450v2 CPU @ 2.50Ghz, 64 GB RAM, running GCC version 4.8.5 (Ubuntu 4.8.5-4Ubuntu2) and Java HotSpot™ 64-Bit VM (build 25.201-B09) in server mode.

We received the numbers presented in Table 4 from our industry partner, who also told us that they were measuring a typical working scenario and there was no other task running on the build server during the measurements. Please note as these measurements were not done by our group, we had no direct control on the measurements and don't have more detail about them. These numbers should only be regarded as showing the possibility of scaling, not as exact figures.

Scenario	build time
Build Server full C build	1150
Build Server full Java build	640
Build Server incremental C build	61-130
Build Server incremental Java build	12

Table 4: Large industrial project build times on the Build Server (in seconds)

The full builds on the build server took 1150s on the C side and 640s on the Java side (see Table 4). The incremental build took approximately 61-130 seconds on the C side and 12 seconds on the Java side.

5.5.1 Some interesting facts and observations during the measurements

1. During the C build all 28 cores were used to almost 100% for the whole duration of the build (maximum or close to the maximum potential parallelisation of build).

2. During the Java build for most of the build time 1-3 cores were used and jumped to 4-8 cores only for a few seconds (as they could observe it using the output of `top` command).

3. On this project, doing a full build, the translation from TTCN-3 to C/C++ files took approx. 33 seconds, leaving 97% of the build on the C side coming from using C/C++ as the intermediate language.

5.5.2 Consequences

(1) While the C side required a 28 core build server for the development and build scenarios, the Java side might have made it possible to develop this project on a laptop; (2) the full build time of the Java side was half, the incremental build was 1/5th to 1/10th to the C side; (3) the scaling of the Java build in terms of build duration seemed to break in the many-core situations, as it was not able to take advantage of all cores to build faster; (4) at the same time, from the resource usage point of view, the Java side had about 50 times more efficient resource usage. While the C side put the build server under heavy load for approximately 20 minutes to build this project, using Java it might have been possible for approximately 20-28 developers to work on the same machine at the same time and still finish the full build in half the time.

6 Limitations

This paper presents a work that was an experiment with limited scope. We had a list of features not yet supported by our extension, compared to the industrial tool, but they were not used in the examples provided by our industry partner. It is also possible that there might be programming issues in the generated Java code. The C side had been in production and use for about 20 years, long enough for most issues to manifest. Although our extension was new, we were able to show functional equivalence (as far as the tests provided by our industry partner go). This makes us confident to say that we don't expect issues resulting in larger architectural changes in our solution.

As a limitation, there might be problems with scaling. The Java class file format is known to have some strict limitations ([43]) that might cause further problems in the generated Java code for larger projects.

Due to the very specific context of our work, our results might not generalise to other systems. Compiling from TTCN-3 to Java in itself might be a limited context and measuring build times and execution times of test systems might also not be a priority for other companies with smaller test systems.

We followed the approach presented by Georges et al. ([22]) to make our performance evaluation statistically rigorous for the compilation and execution measurements done on Laptop 1 and 2, but not for the build server as we had limited and indirect access to it.

7 Threats to validity

During our measurements we had to turn on some debug printouts, which might have added some overhead, that will not be present during the normal usage of the tool.

We had limited control over Laptop 1, as it was used at our industry partner (the firewall software could not be stopped/deactivated/reconfigured for the duration of measurements, etc.). We tried to make sure that no resource intensive operation was running in the background while performing our measurements, but there is still a possibility of some background tasks specific to our industry partner's systems affecting our measurements in ways that might not be present when reproducing this research elsewhere.

There is also a question of fairness in our measurements: we do not have intimate understanding of the internal implementations of either the C/C++ compiler or the Java builder used. That is, in our experiments we tried to reach the best possible performance on both sides. On the C side we used "-O2" to get optimised binaries and "-j8" to use all processing power available in our laptops. On the Java side, in Eclipse, we could not find such options, but we made sure to give enough memory for the build process to minimise the need for garbage collection.

Also at the time our work was accepted by our industrial partner we had limited access to their internal test systems: information on how certain equipment is being tested could raise potential security concerns.

As we had no control over the measurements our industry partner did, checking whether our solution is able to scale might not be perfectly valid and reproducible. However, our industry partner used the information gained from those measurements, and decided that our solution can be turned into an industrial product, we believe that it was in their best interest to gain valid and reliable information.

8 Summary

In this paper we presented our work and empirical observations on extending an industrial compiler to support a new intermediate language.

We have shown how we created an alternative of the existing build procedures of our industry partner by creating an extension that uses Java as the intermediate language besides the already supported C/C++. We described how much of the architecture could be kept to proceed fast and how we overcame the difficulties coming from the differences of the languages.

Our contribution is unique in the sense that both the build and the execution used the same source code (as input) written in the same input language, generated intermediate codes with the same abstract hierarchy, built into executables that were configured using the same files, performed the same behaviour on the same machine and generated the same logs. Evidenced by the test files provided to us by our industry partner, selected from their regression tests, to decide if our solution could be functionally equivalent to their existing system.

Our measurements showed that using Java as an intermediate language might improve build times significantly, providing users with better development iteration times, that might lead to improved productivity. While C/C++ still has better performance in some situations, at least in longer testing scenarios, Java can be a viable alternative as an intermediate language.

In contrast to De Marco et al. ([8]), our development from scratch can be considered a successful migration as we couldn't identify any performance bottleneck and is more maintainable since the knowledge was transferred by programmers using Java idioms where it was possible without changing the architecture. Our observations both supported and contradicted the findings of Nanz and Furia ([31]) and Furia et al. ([21]). Our 1-iteration measurements supported that the same input compiled to C/C++ could usually execute faster compared to Java (although at the cost of much longer compilation). But our measurements with larger iteration counts showed a reversing situation, where compiling to Java might lead to executables that finish faster. Adding the overhead of the build to the execution time (to measure build-execute workflows) would, in our situation, lead to Java finishing earlier in all investigated situations. As such in contrast to Taboada et al. ([41]), we found that for us, Java was a good alternative also from a performance point of view.

Thanks to the help of our industry partner we could check (using one of the largest test systems of our industry partner) that our method scaled up well and kept its benefits, even on large scale test systems. This will enable our industry partner to provide a unique solution for their customers. The developers of these large industrial test systems will be able to use the Java-based build system to work from an IDE with fast feedback cycles (eliminating the negative effect on productivity coming from the C sides build times). At the same time, they will also be able to build the same test systems using the C/C++ based build system for situations where the Java-based executable would not be able to provide the necessary runtime performance.

To support the reproduction of the results of this study all information was made publicly available. The code of the toolset is part of the open-source Titan ([9]) toolset (the Eclipse plugins), and the tests used for testing compilation and execution speed are available at [19].

9 Further work

Even with so limited time we could reach a point where we can confidently say that it is desirable to continue this work.

First of all, there were several features still missing from our solution, that were present in Titan already. We hope they will be added by our industry partner while turning our solution into an industrial product. So that their users can enjoy the benefit of this research.

As described in section 3.2, the 3 years of development was only enough to reach the level of functional equivalence our industry partner required from us. As a follow-up research, it would be desirable to investigate opportunities for performance optimisations in both the runtime and build time. For example: we generated the Java files from TTCN-3 files sequentially. In theory, all of these files could be generated at the same time, in parallel, leading to further build time improvements.

This analysis could be extended to involve more and different hardware into the research and go into more detail on how different hardware features impact performance. It would also be interesting to see how different compiler versions (GCC, Java, etc...) perform on the same source codes in build time and execution performance. For us, the target platforms we were aiming to support were given by our industry partner.

It would be interesting to look into how to improve CPU utilisation on massively multi-core CPUs. According to our industry partner, the build on the Java side did not seem to use all cores to provide maximum build performance.

It might also be a good idea to think of our solution as a platform for further research. A side effect of our solution is that we are transforming large scale industrial and standardised test codes, that are in active use, into a large amount of Java code. This could allow researchers to experiment with how different Java features and usage patterns impact both compilation and runtime performance. Such an experiment would just need to change how our solution translates TTCN-3 code into Java code to have a large amount of source code, from complex and in use systems, for measurements.

Acknowledgements

The authors would like to thank the Test Competence Center of Ericsson Hungary for the financial support of this research and for providing access to their in-house tools. These proved to be invaluable to our measurements.

We would also like to thank Andrea Pálfi, Gergő Újhelyi, Árpad Lovassy, Attila Balaskó and the Titan team for their help in the implementation and measurements. We would also like to thank Eduárd Czimbalmos for his help in our measurements and his feedbacks on the tool.

The research of the first author was supported by the ELTE-Ericsson Laboratory, for the third author by the Project no. TKP2020-NKA-06 (Application domain specific highly reliable IT solutions) with the support from the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme funding scheme.

References

[1] R. Abdalkareem, S. Mujahid, E. Shihab, A machine learning approach to improve the detection of ci skip commits, *IEEE Transactions on Software Engineering*, pp. 1–1, 2020. ⇒140

[2] B. Amedro, V. Bodnartchouk, D. Caromel, C. Delbe, F. Huet, G. L. Taboada, Current State of Java for HPC, Technical Report RT-0353, *INRIA*, 2008. [accessed Apr-2020] ⇒141

[3] ARMOUR, Test generation strategies for large-scale IoT security testing – v1, 2016. [accessed Apr-2020] ⇒135

[4] A. Avram, IDC Study: How Many Software Developers Are Out There?, 2014. ⇒135

[5] N. Bartha, *Scalability on IT projects*, Master's thesis, 2016. ⇒135

[6] L. Batyuk, A.-D. Schmidt, H.-G. Schmidt, A. Camtepe, S. Albayrak, Developing and Benchmarking Native Linux Applications on Android, in J.-M. Bonnin, C. Giannelli, T. Magedanz, *MobileWireless Middleware, Operating Systems, and Applications*, volume **7**, pages 381–392, Berlin, Heidelberg, 2009. Springer. ⇒ 140

[7] R. P. Cook, An OpenMP library for Java, In *2013 Proceedings of IEEE Southeastcon*, pp. 1–6, April 2013. ⇒141

[8] A.De Marco, V. Iancu, I. Asinofsky, COBOL to Java and Newspapers Still Get Delivered, in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 583–586, Sep. 2018. ⇒140, 143, 166

[9] Ericsson Telecom AB., Source code of titan, version 6.6.0. `https://github.com/eclipse/titan.EclipsePlug-ins/releases/tag/6.6.0`, 2019. [accessed Apr-2020]. ⇒167

[10] Ericsson Telecom AB., Programmers' Technical Reference Guide for the Java side of the TITAN TTCN-3 Toolset, `https://github.com/eclipse/titan.core/blob/master/usrguide/java_referenceguide/JavaReferenceGuide.adoc`, 2020. [accessed Apr-2020]. ⇒142, 149

[11] Ericsson Telecom AB., Programmers' Technical Reference Guide for the TITAN TTCN-3 Toolset, `https://github.com/eclipse/titan.core/blob/master/usrguide/referenceguide/ReferenceGuide.adoc`, 2020. [accessed Apr-2020]. ⇒142, 149

[12] ETSI, Methods for Testing and Specification (MTS);The Testing and Test Control Notation version 3; Part 5: TTCN-3 Runtime Interface (TRI), `https://www.etsi.org/deliver/etsi_es/201800_201899/20187305/04.08.01_60/es_20187305v040801p.pdf`, 2017. [accessed Apr-2020.] ⇒137

[13] ETSI, 3GPP test suites, `http://www.ttcn-3.org/index.php/downloads/publicts/publicts-3gpp`, 2020. [accessed Apr-2020]. ⇒135

[14] ETSI, 5G;5GS; User Equipment (UE) conformance specification; Part 3: Protocol Test Suites, `https://www.etsi.org/deliver/etsi_ts/138500_138599/13852303/15.00.00_60/ts_13852303v150000p.pdf`, 2020. [accessed Apr-2020]. ⇒135

[15] ETSI, Intelligent Transport Systems (ITS) Test Suites, `http://www.ttcn-3.org/index.php/downloads/publicts/publicts-etsi/65-publicts-its`, 2020. [accessed Apr-2020]. ⇒135

[16] ETSI, Methods for Testing and Specification (MTS);The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language, `https://www.etsi.org/deliver/etsi_es/201800_201899/20187301/04.12.01_60/es_20187301v041201p.pdf`, 2020, [accessed Oct-2020]. ⇒135, 172

[17] ETSI, Methods for Testing and Specification (MTS);The Testing and

Test Control Notation version 3; Part 6: TTCN-3 Control Interface (TCI), `https://www.etsi.org/deliver/etsi_es/201800_201899/20187306/04.12.01_60/es_20187306v041201p.pdf`, 2020, [accessed Oct-2020]. ⇒137

[18] I. I. Farkas, K. Szabados, A. Kovács, Measurement data with configuration, `http://compalg.inf.elte.hu/~attila/materials/Measurements_Laptop1.xlsx`, `http://compalg.inf.elte.hu/~attila/materials/Measurements_Laptop2.xlsx`, 2019. ⇒157

[19] I. I. Farkas, K. Szabados, A. Kovács, *Regression test data*, `http://compalg.inf.elte.hu/~attila/materials/RegressionTestSmall_20190724.zip`, 2019. ⇒ 149, 151, 152, 167

[20] I. I. Farkas, K. Szabados, A. Kovács, An example containing a "Hello World", some simple types in TTCN-3, and the compiled C/C++ and Java codes, `http://compalg.inf.elte.hu/~attila/materials/Example_package.zip`, 2020. ⇒142, 149

[21] C. A. Furia, R. Feldt, R. Torkar, Bayesian Data Analysis in Empirical Software Engineering Research, *IEEE Transactions on Software Engineering*, pp. 1–1, 2019. ⇒141, 166

[22] A. Georges, D.Buytaert, L. Eeckhout, Statistically Rigorous Java Performance Evaluation, *Proceedings of the 22Nd Annual ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications*, OOPSLA '07, pp. 57–76, New York, NY, USA, 2007. ACM. ⇒152, 154, 165

[23] T. A. Ghaleb, D. A. da Costa, Y. Zou, An empirical study of the long duration of continuous integration builds, *Empirical Software Engineering*, 24(4):2102–2139, Aug 2019. ⇒135, 140

[24] L. Gherardi, D. Brugali, D. Comotti, A Java vs. C++ Performance Evaluation: A 3D Modeling Benchmark, I. Noda, N. Ando, D. Brugali, J. J. Kuffner, editors, Simulation, Modeling, and Programming for Autonomous Robots, pp. 161–172, Berlin, Heidelberg, 2012. Springer. ⇒141

[25] IoTKETI, oneM2MTester, `https://github.com/IoTKETI/oneM2MTester`, 2016. Last visited: April, 2020. ⇒135

[26] J. Nielsen, Response times: The 3 important limits `https://www.nngroup.com/articles/response-times-3-important-limits/`, 1993. Last visited: October, 2020. ⇒155

[27] A. Kovács, K. Szabados, Test software quality issues and connections to international standards, *Acta Universitatis Sapientiae, Informatica*, **5**, pp. 77–102, 05 2013. ⇒135

[28] A. Kovács, K. Szabados, Advanced TTCN-3 Test Suite validation with Titan, *In Proceedings of the 9th International Conference on Applied Informatics*, volume **2**, pp. 273–281, 02 2014. ⇒135

[29] P. Lathan, S. Burke, K. Gallick, A. Coleman, New cpu test methodology 2020: Code compile, updated gaming, transcoding, & more `https://www.youtube.com/watch?v=sg9WgwIkhvU`, 2020, [accessed May-2020]. ⇒142

[30] M. Meredith, J. Kruschke, Bayesian Estimation Supersedes the t-Test, `https://cran.r-project.org/web/packages/BEST/vignettes/BEST.pdf`, 2018, [accessed Apr-2020]. ⇒158

[31] S. Nanz, C. A. Furia, A Comparative Study of Programming Languages in Rosetta Code, *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume **1**, pp. 778–788, May 2015. ⇒141, 166

[32] P. Oláh. *Improving the semantic analysis in TTCN-3 environment*, Master's thesis, *Eötvös Loránd University, Budapest*, Hungary, 2016. ⇒150

[33] K. Reinholtz, *Java will be faster than C++*, SIGPLAN Not., **35**(2):25–28, Feb. 2000. ⇒140

[34] A. Ruano, G. Réthy, Developing an Open Source conformance testing environment for ITS communications, UCAAT, 2016. ⇒135

[35] S. Salmons, M. Arnaud, X. Zeitoun, C. Bouattour, *Model-based platform for smart grid interoperability testing using TTCN-3* In UCAAT, 2018. ⇒135

[36] K. Szabados, *Structural Analysis of Large TTCN-3 Projects* In M. Núñez, P. Baker, M. G. Merayo, editors, Testing of Software and Communication Systems, pages 241–246, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ⇒135

[37] K. Szabados, A. Kovács. *Technical debt of standardized test software* In 2015 IEEE 7th International Workshop on Managing Technical Debt (MTD), pages 57–60, Bremen, Oct 2015. ⇒135

[38] K. Szabados, A. Kovács, *Internal quality evolution of a large test system–an industrial study* In Acta Universitatis Sapientiae, Informatica, 8(2):216–240, 12 2016. ⇒135, 162

[39] K. Szabados, A. Kovács, G. Jenei, D. Góbor, *Titanium: Visualization of TTCN-3 system architecture* In 2016 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), pages 7–11, May 2016. ⇒135, 162

[40] J. Szabó, T. Csöndes, *TITAN, TTCN-3 test execution environment*, `https://www.hiradastechnika.hu/data/upload/file/2007/2007_1a/HT_0701a-6.pdf`, 2007. Last visited: October, 2020. ⇒136, 143

[41] G. L. Taboada, S. Ramos, R. R. Expósito, J. Touriño, R. Doallo. *Java in the High Performance Computing arena: Research, practice and experience*, Science of Computer Programming, 78(5):425 – 444, 2013. Special section: Principles and Practice of Programming in Java 2009/2010 & Special section: Self-Organizing Coordination. ⇒141, 167

[42] U.S. General Services Administration (GSA) Technology Transformation Service, *Interaction Design Basics*, `https://www.usability.gov/what-and-why/interaction-design.html`, 2020. Last visited: October, 2020. ⇒155

[43] ∗ ∗ ∗. *Chapter 4. The class File Format*, `https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-4.html`, 2020. last visited: April, 2020. ⇒149, 165

[44] ∗ ∗ ∗. Cygwin, 2020. last visited: January, 2020. ⇒152

[45] ∗ ∗ ∗. Eclipse IoT-Testware, 2020. Last visited: April, 2020. ⇒135

[46] ∗ ∗ ∗. Titan, 2020. last visited: January, 2020. ⇒152

Appendices

A Short introduction to TTCN-3

TTCN-3[22] is a high level standardised language ([16]) designed for testing. Mostly used for functional testing (conformance testing, function testing, integration, verification, end-to-end and network integration testing) and performance testing. TTCN-3 can be used (1) to test reactive systems via: message based communication, (2) API based and analog interfaces and systems.

The language is governed by a strict, internationally accepted specification. Each language construct, allowed by the syntax and semantics of the standard, has a well specified behaviour. Tests written in TTCN-3 can be transferred to other vendor's tools without modification. Some standards of reactive systems (for example communication protocols) offer their specifications together with a set of tests written in TTCN-3. This provides an easy and automated way for tool vendors and users to check the conformance of the implementation.

TTCN-3 offers platform independent abstract data types (see listing 1). There is no value range restriction for integers, no precision restriction for floats, and no length restriction for string types. String types are differentiated based on their contents (bitstring, hexstring, octetstring, charstring, universal charstring). Creating new types is supported by building structured types with fields (record, set) or by list of an element type (record of, set of). It is also possible to create new types with restriction (for example length restriction on strings). This rich type / data constructs can easily be extended by importing other data types / schema (ASN.1[23], IDL[24], XSD[25] and JSON[26]) without need for manual conversion.

The templates of TTCN-3 merge the notions of test data and test data matching into one concept (see listing 2). This enables the specification of expected responses in a concise way. Matching rules can be for example: single value ("Budapest"), list of alternatives ("Monday", "Tuesday"), range (1 .. 5), ordered and unordered lists of values, sub- and supersets of unordered values, string patterns (pattern"* chapter"), permutations of values. When declaring templates for structured data types, these matching rules can be declared for each field and element individually or for the whole tem-

[22]Test and Test Control Notation 3
[23]Abstract Syntax Notation One
[24]Interface Definition Language
[25]XML Schema Definition
[26]JavaScript Object Notation

Listing 1: data types example

```
var boolean v_boolean:=true;
const integer c_i:= 123456789101112131415;
const float c_f1:=1E2;
const float c_f2:=100.0;
var bitstring v_bits:='01101'B;
var charstring v_chars:="ABCD";
var hexstring v_hexs := '01A'H;
var octetstring v_octs:='0BF2'O;
var universal charstring v_uchars := "F" & char(0, 0, 0, 65)

type record recordOper_trecord {
        integer x1 optional,
        float x2 };
type record of octetstring recordOper_trecof;
type set recordOper_tset {
        integer x1,
        float x2 optional };
type set of charstring recordOper_tsetof;

type integer templateInt_subtype (0..1457664);
type record length (3)
  of record length (3)
  of record length (3) of integer threeD;
```

Listing 2: templates example

```
template integer t_i := 123456789101112131415
var template float vt_f := (1.0 .. 2.0);
template mycstr t_mycstr := pattern "ab" & "cd";

template templateCharstr_rec templateCharstr_tList :={
        x1:="00AA", //specific value
        x2:=("01AA","01AB","11AC"), // value list
        x3:=complement ("11","0A","1BC0"), // complement list
        x4:=? length(2..4), //any string with a length of 2 to 4
        x5:= pattern "10*" //any string matching the pattern
};
```

Listing 3: Example for receiving message

```
testcase tc_HelloWorld() runs on MTCType system MTCType
{
  timer TL_T := 15.0;
  map(mtc:MyPCO_PT, system:MyPCO_PT);
  MyPCO_PT.send("Hello, world!");
  TL_T.start;
  alt { //branching based on events
    [] MyPCO_PT.receive("Hello, TTCN-3!") {
        TL_T.stop;
        setverdict(pass);//receiving the right message
       }
    [] TL_T.timeout {
        setverdict(inconc); // the test timed out
       }
    [] MyPCO_PT.receive {
        TL_T.stop; // some other message was received
        setverdict(fail);
       }
  }
}
```

plate. Checking whether a data value matches to the template is as easy as "match(value, templateValue)". Other constructs offer additional functionality, e.g. "∗.receive(templateValue) → value" activates only if a value matching to the provided template is received, in which case the value of the message is saved in "value" for further processing.

TTCN-3 can also be viewed as a "C -like" procedural language with testing specific extensions. The usual programming language features (function, if, while, for, etc.) are extended with other constructs needed for testing: test cases as standalone constructs, sending/receiving messages, invoking remote procedures and checking the content of the received data structures (messages/results/exceptions), alternative behaviors depending on the response of the tested entity, handling timers and timeouts, verdict assignment and tracking, logging of events (see listing 3) are all built in.

Creating distributed test cases and test execution logic is easy as well. A TTCN- 3 test may consist of several parallel test components which are distributed on a set of physical machines, able to work in tandem to test all interfaces of the tested system, or able to create high load. Test components, communication ports to the tested entity and to other test components are defined in TTCN-3. The number of test component instances and their connections are controlled from the code of the test case dynamically using various language features (see listing 4). Deploying and controlling the test component also happens in an abstract and platform independent way. The user does not need to work with the implementation details. It is the tools responsibility to utilize the available pool of machines, possibly running on different operating systems.

Listing 4: multiple components example

```
testcase commMessageValue() runs on commMessage_comp2 {
var commMessage_comp1 comp[5];
var integer xxint;
for (var integer i:=0; i<5; i:=i+1)
{ log(i);
  comp[i]:=commMessage_comp1.create;//creating component
  comp[i].start(commMessage_behav1(i));//start remote behavior
  connect(self:Port2[i],comp[i]:Port1);//connect to component
  xxint:=5;
  Port2[i].send(xxint);//send message on port
  Port2[i].receive(integer:?) -> value xxint;//receive response
  if (xxint==5+i) {setverdict(pass)}
    else {setverdict(fail)};
}
for (i:=0; i<5; i:=i+1) {comp[i].stop};//stop the components
};
```

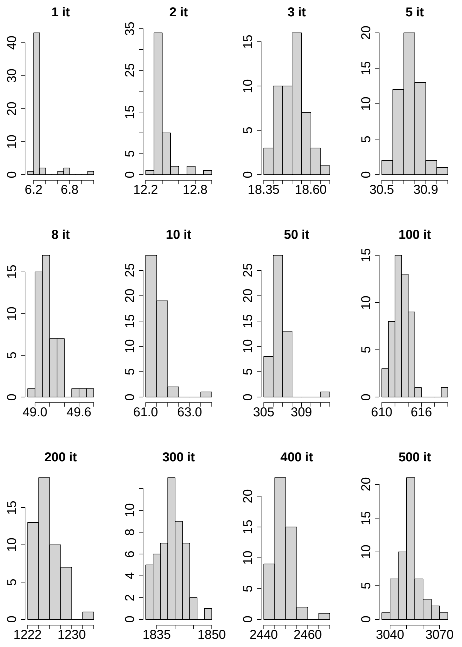
TTCN-3 is also independent from the test environment. The user needs only to define abstract messages exchanged between the test system and test tested entity. Message encoding (serialization), decoding (de-serialization), handling of connections and transport layers are done by the tools.

TTCN-3 also offers to control the test case execution logic and dynamic test selection from within the the TTCN-3 code itself (see listing 5). Module parameters allow for the user to leave data open in the source code and provide the actual values at execution time (IP addresses, IDs, passwords, etc...)

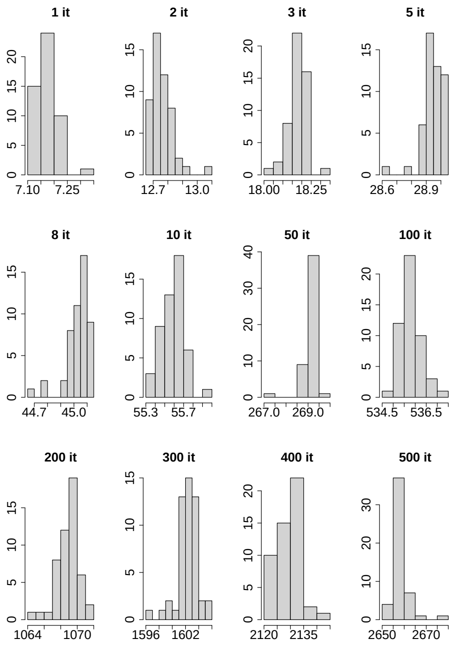
Listing 5: execution control example

```
control {
  for(var integer i := 0; i < 10; i := i+1)
  {
    execute(parameterised_testcase(i));
  }
  execute(transferTest());
  execute(tc_runsonself());
 }
```

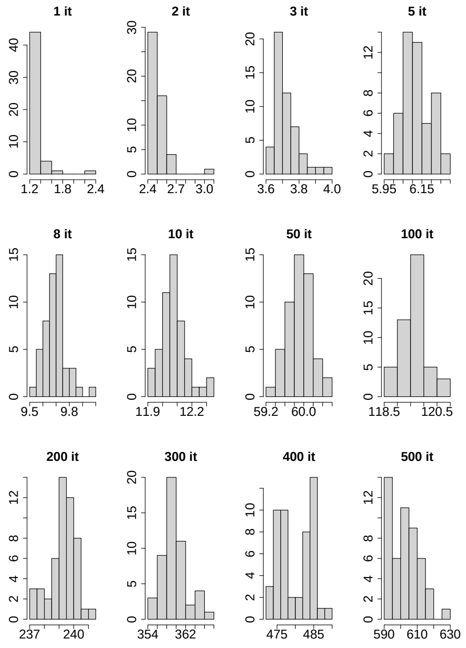
B Measurement details and histograms



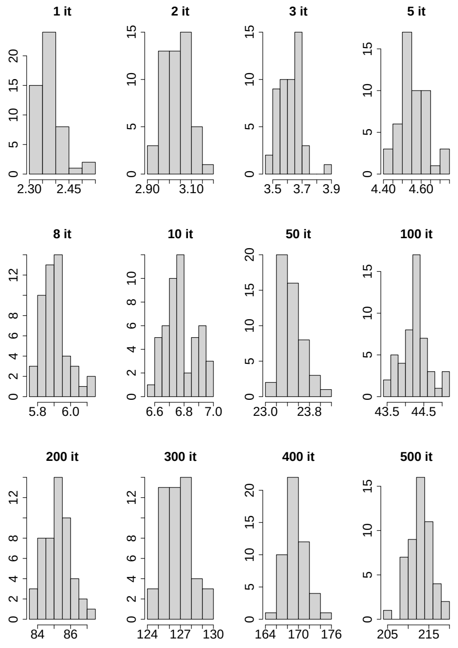
(a) C side with timers



(b) Java side with timers

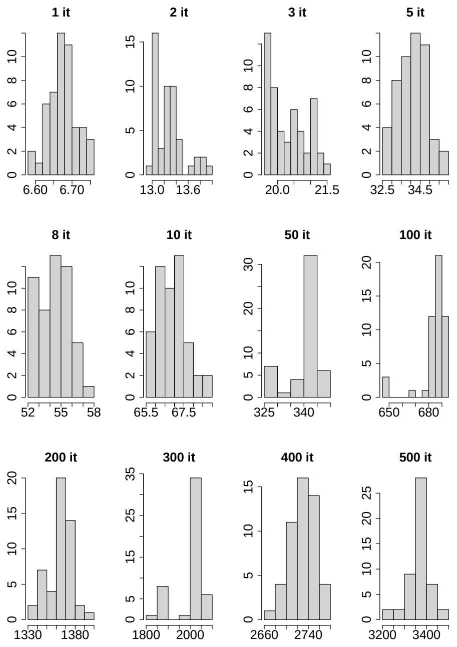


(c) C side without timers

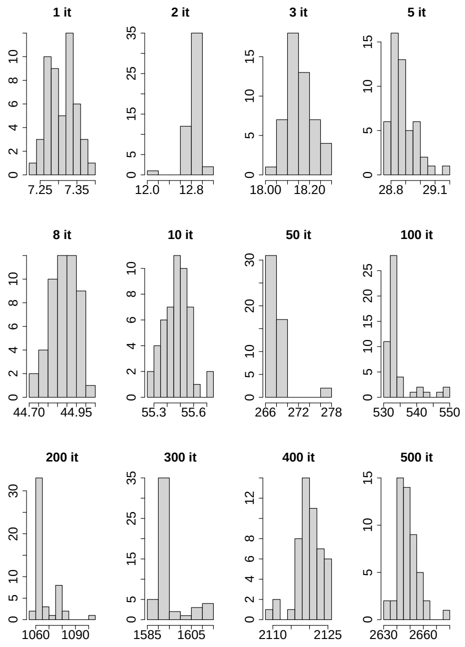


(d) Java side without timers

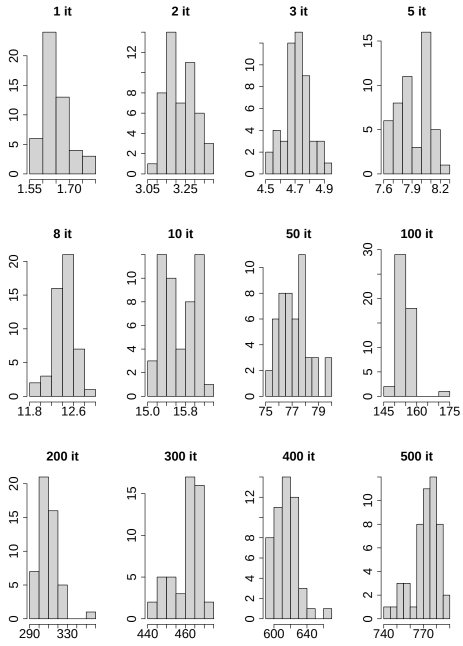
Figure 5: Histogram of the measured execution times on Laptop 1. C (5(a)) and Java (5(b)) side with timers; C (5(c)) and Java (5(d)) side without timers.



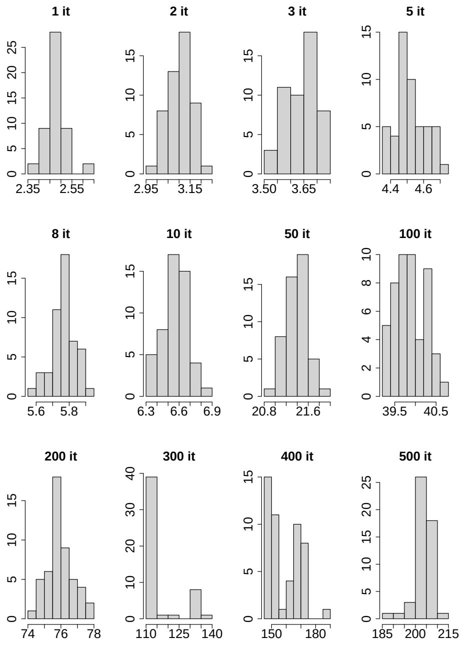
(a) C side with timers



(b) Java side with timers



(c) C side without timers



(d) Java side without timers

Figure 6: Histogram of the measured execution times on Laptop 2. C (6(a)) and Java (6(b)) side with timers; C (6(c)) and Java (6(d)) side without timers.

it. nr	C + full	Java + full	C + inc	Java + IDE m.
1	159.69	18.13	40.80	7.29
2	165.81	23.71	46.92	12.88
3	171.90	29.13	53.02	18.30
5	184.16	39.90	65.27	29.07
8	202.58	55.99	83.69	45.15
10	214.95	66.55	96.06	55.71
50	460.13	280.08	342.24	269.24
100	766.49	546.76	647.60	535.93
200	1378.82	1079.79	1259.93	1068.95
300	1992.45	1613.31	1873.56	1602.47
400	2602.08	2140.54	2483.19	2129.70
500	3205.31	2669.37	3086.42	2658.53

(a) Laptop 1 with timers

it. nr	C + full	Java + full	C + inc	Java + IDE m.
1	154.75	13.33	35.86	2.49
2	155.92	13.99	37.03	3.15
3	157.12	14.57	38.23	3.74
5	159.52	15.51	40.63	4.67
8	163.09	16.86	44.20	6.02
10	165.47	17.73	46.58	6.90
50	213.30	34.43	94.41	23.60
100	273.09	55.22	154.21	44.38
200	392.69	96.13	273.80	85.29
300	513.03	137.67	394.14	126.83
400	633.35	180.51	514.46	169.67
500	756.39	223.90	637.50	213.06

(b) Laptop 1 without timers

it. nr	C + full	Java + full	C + inc	Java + IDE m.
1	231.20	35.44	32.40	7.61
2	237.80	40.97	39.00	13.14
3	244.84	46.28	46.04	18.46
5	258.65	57.01	59.85	29.19
8	278.94	73.01	80.14	45.19
10	291.41	83.60	92.61	55.78
50	565.01	296.07	366.21	268.24
100	684.41	562.35	710.14	534.53
200	1364.92	1094.89	1390.65	1067.07
300	2228.21	1623.23	2029.41	1595.41
400	2953.54	2147.90	2754.74	2120.07
500	3590.27	2676.44	3391.47	2648.62

(c) Laptop 2 with timers

it. nr	C + full	Java + full	C + inc	Java + IDE m.
1	226.18	30.66	27.38	2.78
2	227.75	31.29	28.95	3.41
3	229.24	31.83	30.44	3.94
5	232.45	32.70	33.65	4.82
8	236.94	33.95	38.14	6.07
10	240.20	34.75	41.40	6.87
50	301.68	49.57	102.88	21.69
100	379.07	68.06	180.27	40.18
200	534.88	104.07	336.08	76.19
300	685.23	144.76	486.43	116.87
400	838.43	186.96	639.63	159.08
500	996.47	231.90	797.67	204.02

(d) Laptop 2 without timers

Table 5: The sums of the average build times of the build kinds (already presented in Table 1) and the averages of execution times for different iteration numbers on Laptop 1 with 5(a) and without timers 5(b), on Laptop 2 with 5(c) and without timers 5(d) in seconds (also already presented on Figures 3 and 4). The full postfix of the table means a full build, the inc postfix means the incremental build and the inc means the incremental build in IDE mode.

Received: April 18, 2021 • Revised: June 1, 2021

Acta Univ. Sapientiae, Informatica **13,** 1 (2021) 180–194

DOI: 10.2478/ausi-2021-0008

Fog-LAEEBA: Fog-assisted Link aware and energy efficient protocol for wireless body area network

Kifayat ULLAH
Department of Computer Science,
CECOS University of IT and Emerging Sciences, Peshawar, Pakistan
email: `kifayat@cecos.edu.pk`

Haris KHAN
Department of Computer Science,
CECOS University of IT and Emerging Sciences, Peshawar, Pakistan
email: `hariskhan5848@gmail.com`

Abstract. The integration of Wireless Sensor Networks (WSN) and cloud computing brings several advantages. However, one of the main problems with the existing cloud solutions is the latency involved in accessing, storing, and processing data. This limits the use of cloud computing for various types of applications (for instance, patient health monitoring) that require real-time access and processing of data. To address the latency problem, we proposed a fog-assisted Link Aware and Energy Efficient Protocol for Wireless Body Area Networks (Fog-LAEEBA). The proposed solution is based on the already developed state-of-the-art protocol called LAEEBA. We implement, test, evaluate and compare the results of Fog-LAEEBA in terms of stability period, end-to-end delay, throughput, residual energy, and path-loss. For the stability period all nodes in the LAEEBA protocol die after 7445 rounds, while in our case the last node dies after 9000 rounds. For the same number of rounds, the end-to-end delay is 2 seconds for LAEEBA and 1.25 seconds for Fog-LAEEBA. In terms of throughput, our proposed solution increases the number of packets received by the sink node from 1.5 packets to 1.8 packets. The residual energy of the nodes in Fog-LAEEBA is also less than

Computing Classification System 1998: G.2.2
Mathematics Subject Classification 2010: 68R15
Key words and phrases: wireless sensor network, wireless body area network, fog computing, fog-laeeba, energy efficiency

the LAEEBA protocol. Finally, our proposed solution improves the path loss by 24 percent.

1 Introduction

The advancement in emerging technologies is reshaping our world and living styles. Some of these emerging technologies are Wireless Sensor Network (WSN), Wireless Body Area Network (WBAN), and cloud computing. A large number of cloud-based solutions have been developed. One of the major drawbacks of cloud computing is the delay involved in accessing and processing data. This delay can significantly influence network performance [1]. Cloud computing does not fulfill the real-time requirements of delay-sensitive applications [14]. To address this limitation of cloud computing the idea of fog computing was proposed. Fog computing is a new paradigm shift in providing cloud services at the edge of the network. It provides tools for managing, distributing, and securing resources and facilities across networks and between devices that exist at the edge of the network [4].

WBAN is a special type of WSN in which several small lightweight wireless sensors are placed or implanted in human bodies for health monitoring [8]. The main objective is to monitor, collect, and report medical data from the patient body [12, 3]. Different applications have been proposed using WBAN. For example, patient monitoring, sudden fall of a patient, informing an emergency response unit, call, sudden falls of patients, and providing guidance to the patients, etc.

Current solutions for monitoring patient health focuses on the collection of medical data and sending it towards a centralized server for making it available to the medical staff. One such solution is the so-called Link Aware and Energy Efficient Scheme for Body Area Networks (LAEEBA). It is a well-known, reliable, and efficient protocol designed for WBAN. The design of LAEEBA is based on cloud architecture. The major issue with the LAEEBA protocol is the delay involved in the data processing, which is not tolerable in the case of health-related applications [10].

We extend the operations of the LAEEBA protocol to fog computing. In this regard, we proposed the design, and implementation of fog assisted efficient solution called Fog-LAEEBA. The main focus is to address the problem of delay involved in the LAEEBA protocol. On one side, this will help in storing, managing, computing, and analyzing the sensory data. On the other side, it will minimize the delay involved in the LAEEBA protocol. In the proposed solution, we deployed different sensors on the human body to monitor the patient's health-related data. Such data includes blood pressure, pulse rate, and temperate, etc. This information will be transferred to a sink node, placed on the body. The sink node will then send the data to the fog node.

The general objective of this research is to extend the operations of the LAEEBA protocol to support the fog computing paradigm. To this end, the more specific objectives are:

- To design a fog-assisted solution for patient health monitoring;
- To implement, test, and evaluate the performance of the proposed solution in terms of stability period, delay, throughput, path-loss, and energy utilization;
- To compare the result of the proposed solution with the LAEEBA protocol.

2 Literature review

In the literature different algorithms, and protocols have been proposed for WBAN using cloud and fog computing. These solutions include different applications for real-time health monitoring, improving network performance, enhancing energy utilization, and minimizing the delay involved in processing. In this section, we provide a literature review of the existing solutions.

A detailed overview of WBAN is provided in [3]. This work highlights both the medical and non-medical applications. It also specifies the requirements for different types of sensors, which are used in WBAN. Furthermore, this work outlines the communication technologies, like, Global System for Mobile communication (GSM), Wireless Personal Area Network (WPAN), WiFi, cellular networks, ZigBee, and Bluetooth for WBAN.

As the sensor nodes in WBAN are battery-powered, hence, it is important to efficiently utilize the energy of these nodes. To increase the network performance of WBAN, an Identification Key Scheme (IKS) was proposed in [2]. The IKS improves the performance of WBAN in terms of minimizing the packet loss, and end-to-end delay. The authors used the OMNET++ simulator to implement and evaluate the performance of IKS. Our proposed scheme also considered these evaluation metrics, however, we implement our scenario in the presence of a fog unit.

An analytical model, to improve the network performance of healthcare applications, is proposed in [13]. The proposed solution recommends the use of fuzzy logic and fog computing to reduce the latency involved in the WBAN. The proposed solution was implemented in the iFogSim simulator. The authors only consider network latency metric for the evaluation of the proposed model.

A Long Range (LoRa) based, fog-assisted health monitoring application is proposed in [7]. The proposed work highlight the importance of a health monitoring system. To address the need for a proper health system in practice, the authors emphasize the use of fog computing, IoT, and LoRa communication technologies. They also proposed an architecture for the proposed monitoring system. Finally, the authors developed a Raspberry Pi-based test-bed to perform their experiments. However, they focus on the communication aspect (e.g., RSS, and SNR) of the LoRa. One of the main drawbacks of the proposed solution is the placement of fog nodes. In this work, the fog nodes are placed health centers and hospitals, which are far away from the patient homes.

As the aging population of the world is increasing. There is a social need to address the health issues of this large community. To address this problem, a fog computing-based elderly health monitoring system is proposed in [5]. The proposed system utilizes the Mysignals HW kit for measuring health parameters. They also developed a mobile application to analyze the collected information.

A fog computing and K-Nearest Neighbor (KNN) based health monitoring system for heart patient monitoring is proposed in [11]. This work makes use of fog computing to minimize the delay problem and also enhance the accessibility of the system. The main focus of this work is to monitor the heart disease patient and alert the doctor, in case of an emergency. The authors implement their solution by using an Arduino board, Raspberry pi device, and heart rate sensor module. They evaluated the performance in terms of accuracy only.

The authors in [9] proposed a scheme called AnyCasting In Dual Sink (ACIDS) for enhancing the performance of WBAN. This work focuses on enhancing the performance of the WBAN in terms of throughput and stability period. Like our proposed solution, they compared the results with the LAEEBA protocol. This work does not give attention to energy efficiency, which is important to consider to evaluate the performance of WBAN. Furthermore, their solution does not consider the use of the fog computing paradigm.

The major inspiration behind our proposed solution is the protocol presented in [10]. This protocol is called Link Aware and Energy Efficient Scheme for BAN (LAEEBA). The LAEEBA is a well-recognized, reliable, and efficient routing protocol designed for WBAN. The LAEEBA protocol tries to improve the energy efficiency of the network by maximizing the network throughput and minimizing delay. For efficient path selection, it utilizes the number of hops while for energy consumption a cost function is calculated. The idea of cooperative Link-Aware and Energy Efficient protocol for WBAN (Co-LAEEBA) is provided in [?]. This work focuses on the impact of single-hop and multi-hop communication techniques. It also highlights the importance of cooperation among sensor nodes. The main conclusion was that such cooperation can maximize the throughput of a network.

3 Research methodology

The main objective of this research work is to propose a fog-assisted protocol for WBAN. To achieve this goal, we followed a step-by-step research methodology. This section presents the most important steps of our methodology.

3.1 Conduct literature survey

To conduct this research, we performed a detailed literature review by consulting standard libraries, e.g., Google Scholar, IEEE, ACM, SpringerLink, and ScienceDirect, etc. We studied different approaches for the provision of health services using WBAN. This study enables us to identify the main limitations of the existing approaches and identify a research gap.

3.2 Simulation tools

For WBAN and fog computing, performing real-world experiments is a costly, laborious, and time-consuming task. Similarly, it is difficult to repeat the same experiments for different parameters. An alternative solution is to perform simulations. Simulation tools are used to implement, test, validate, compare, and evaluate the performance of the proposed solution. For this purpose, a wide range of simulation tools, ranging from open source to proprietary solutions, are available. However, the selection of the most appropriate simulation tools is an important step in any research. We used a well-known simulation tool called MATLAB. It is a matured and popular platform for scientific computing and is widely used for conducting various simulation-based studies. MATLAB has a rich set of modules that allows the user to perform different tasks.

3.3 Simulation scenario

After the selection of simulation tool, an important step to perform any simulation-based experiment is to select a suitable scenario and set up the simulation environment. This step is also valuable for achieving reliable simulation results. In our proposed solution, we deployed different sensor nodes on the human body. In addition, we also placed a sink node. Fig. 1 shows the placement of these sensor nodes. All the sensor nodes are homogeneous having the same computational capabilities. These sensor nodes are responsible for sensing body temperature, heart rate, and oxygen level, etc. On the other side, the sink node forwards the data collected from these sensor nodes toward the fog node for further processing.

3.4 Simulation parameters

To evaluate the performance of the proposed solution, it is important to define a set of parameters and metrics. The parameters that we choose for our experiments are stability period, end-to-end delay, throughput, residual energy, and path loss. These parameters are important for WBAN. In addition, the selection of this standard set of parameters would help other researchers to compare their solution with our approach.

3.5 Performance evaluation

To evaluate the performance of our proposed system, we performed different experiments. We studied how the selected parameters influence the performance of the proposed solution. Furthermore, we also compared the result with the LAEEBA protocol.

3.6 Results analysis

The last step of our proposed research methodology is to analyze the collected results of the experiments and draw some useful conclusions. We discuss our results in more detail in section 5.

4 Fog-laeeba

In this section we present the details about our proposed solution. Fog-LAEEBA is an extension to the LAEEBA protocol. It addresses the limitations, e.g., delay, energy consumption, throughput, and path-loss of the LAEEBA protocol. The design of our protocol is based on fog computing paradigm. Compared to the cloud computing, the fog node is much closer to the sensor nodes and hence is considered to be an efficient solution, especially for health related applications.

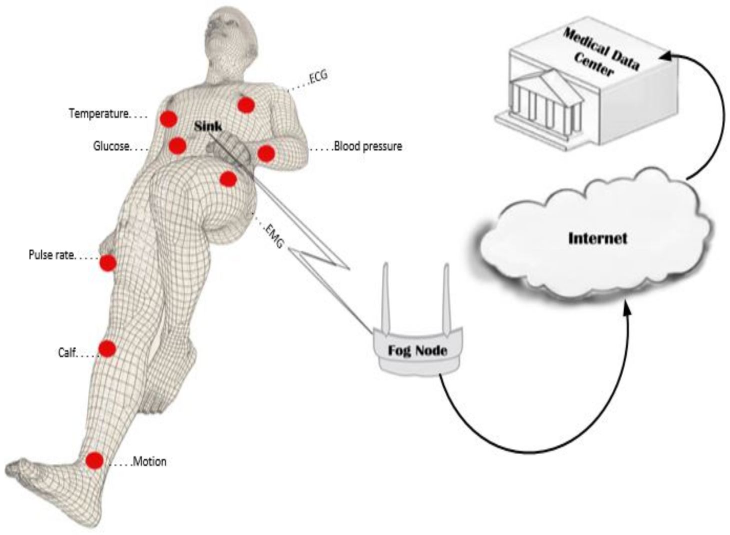


Figure 1: Proposed system model.

4.1 System model

In our proposed model we deployed eight sensors and one sink node. All nodes have equivalent power and computation abilities. Fig. 1 shows the placement of various sensors on the body. Each node is responsible for sensing different health-related parameters. From the sink node, the collected data is transferred to the fog Node. The flowchart of our system is also depicted in Fig. 2.

To implement our proposed solution, we used MATLAB. We perform extensive simulation experiments to evaluate and compare the performance. Table 1 indicates the various parameters used for experiments.

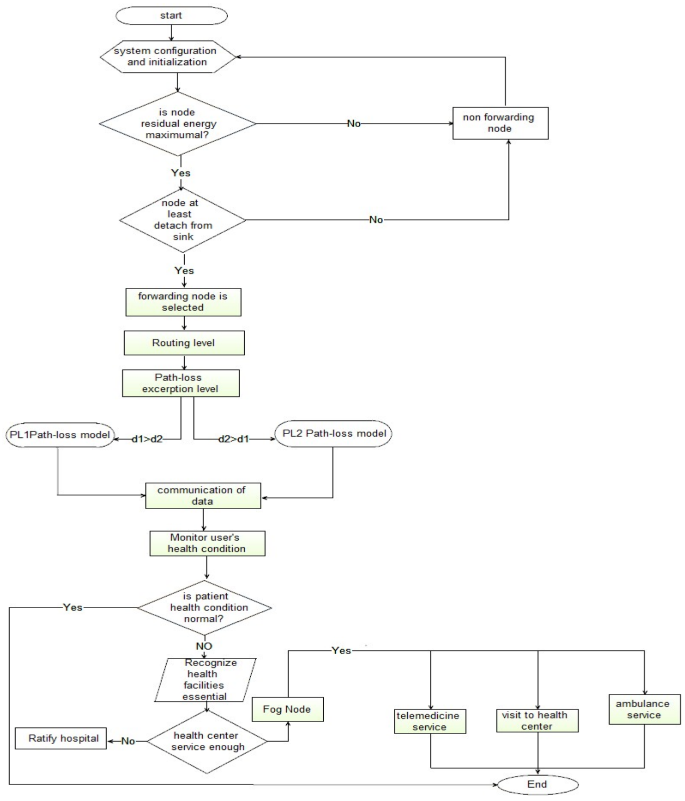

start
system configuration and initialization
is node residual energy maximumal?
No
non forwarding node
Yes
node at least detach from sink
No
Yes
forwarding node is selected
Routing level
Path-loss excerption level
PL1Path-loss model
d1>d2
d2>d1
PL2 Path-loss model
communication of data
Monitor user's health condition
is patient health condition normal?
Yes
NO
Recognize health facilities essential
Yes
Fog Node
telemedicine service
visit to health center
ambulance service
Ratify hospital
No
health center service enough
End

Figure 2: Flow diagram of proposed solution.

5 Results and discussion

This section presents the results of our simulation experiments. We also provide a comparison of the results with the existing LAEEBA protocol. It is important to note that we repeat each set of experiments five times.

5.1 Stability period

The first metric that we consider to evaluate the performance of Fog-LAEEBA is the stability period. The stability period is the time in which all the nodes in a network stay alive. It is considered to be the key performance evaluation metric. We present the obtained results of the stability metric in Fig. 3. The results reveal that in the LAEEBA protocol the first node dies at around 2147, while in our protocol the first node dies after 4437 rounds. After round number

Parameter	Value
Number of nodes	8
Position of nodes	On human body
Mobility model	First order radio Model
Size of packet	1000 bits
Initial Energy	0.4J
Eamp	0.0013 PJ/bit/m^4
Eelec	50nj/bit
Efs	100 pj/bit//m^2
Eda	5 NJ/bit
Emp	1.97 NJ/bit
Initial Energy	0.5 J
Frequency	2.4 GHz
Wavelength	0.125m
Standard	IEEE 802.15.6

Table 1: Simulation parameters and their values.

7445, all nodes in the LAEEBA protocol died, whereas in our proposed solution the last node dies after round number 9000. From the result, it is clear that our proposed solution performed better than the LAEEBA protocol. We can conclude that the stability period of our protocol is high, which will enhance the lifetime of the network.

5.2 End-to-end delay

An important parameter to consider in performance evaluation is an end-to-end delay. It is the time taken by a packet to reach the destination. One of the main goals of WBAN is to decrease the end-to-end delay. We evaluate the performance of the Fog-LAEEBA with the objectives to minimize this delay.

Our simulation results are shown in Fig. 4. The results demonstrate that in terms of end-to-end delay our solution performs better as compared to LAEEBA Protocol. It can be observed that in our case, the end-to-end delay is high (i.e., 2.243 milliseconds) during the early stages. However, after some cycles, this delay is minimized.

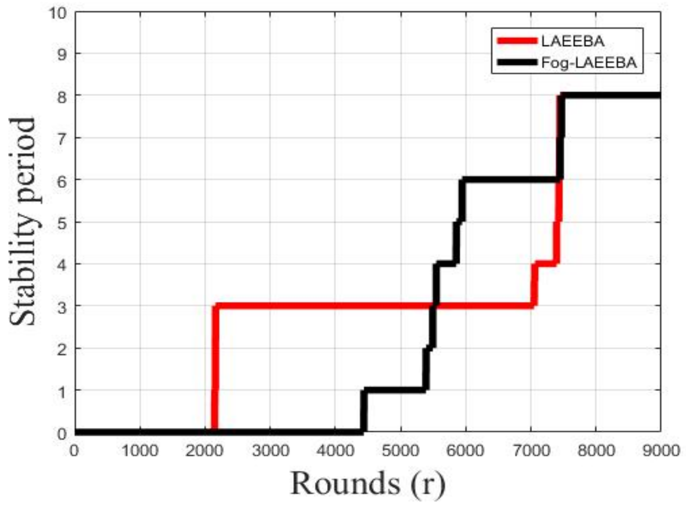


Figure 3: Stability period.

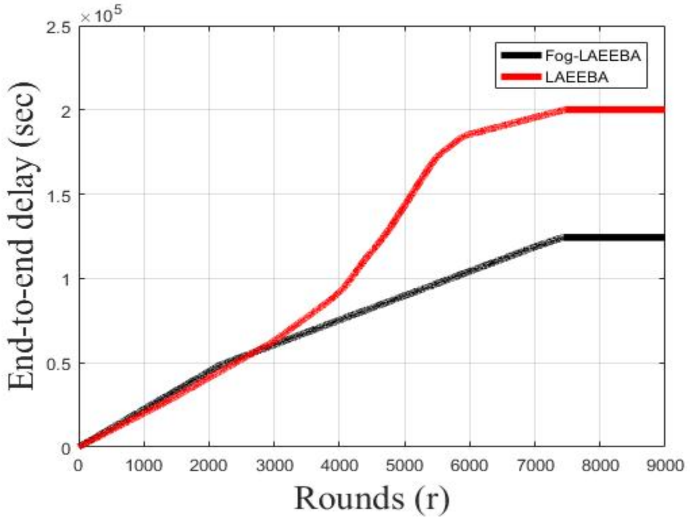


Figure 4: End-to-end delay.

5.3 Throughput

Throughput means the number of packets being transferred from source to destination in unit time. It is also an important factor to be considered for performance evaluation. The design goal of any solution should be to improve network performance by maximizing throughput. We also considered throughput for evaluating the performance of Fog-LAEEBA.

The results collected are shown in Fig. 5. At the initial stages of the simulations, the performance of both protocols is the same. However, at later stages, our proposed solution achieved higher throughput as compared to the LAEEBA protocol. In the case of the LAEEBA protocol, about 1.5 packets (on average) were received by the sink node. However, in the case of Fog-LAEEBA, around 1.8 packets were successfully received. From the results, we concluded that Fog-LAEEBA improved the performance by about 13 percent.

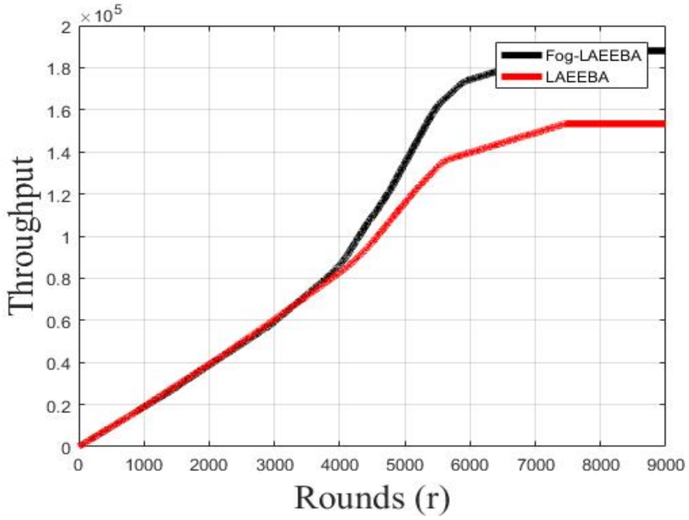


Figure 5: Throughput.

5.4 Residual energy

As the sensor nodes rely on batteries, hence, energy is the main concern in such types of networks. The term residual energy means the current energy of a sensor node, after performing basic operations,e .g., sending or receiving data packets. To perform the experiments, we kept the initial energy of sensor nodes to four joules. The obtained simulation results are shown in Fig. 6. From the results, it is clear that our proposed solution consumes less energy at the start of the network operation. Up to round number 4000, the residual energy of our proposed solution is less than that of the LAEEBA protocol. However, after that point, there is a slight variation in the energy consumption by both solutions.

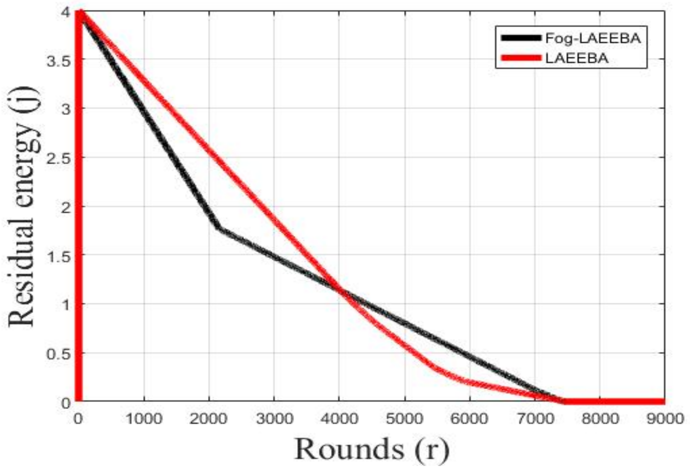


Figure 6: Residual energy.

5.5 Path-loss

Path loss is the reduction in the power of an electromagnetic wave. Several factors can affect the quality of the transmitted signal. Usually, in WBAN, the sensors are placed on or implanted inside the human body. Hence losses between these devices could reduce the performance of health monitoring activities. We performed different experiments to evaluate the performance of our proposed solution in terms of path-loss. The results of path-lose experiments are shown in Fig. 7. From the result, it is clear that the proposed solution reduces the path loss at various rounds. Only between 2150 and 3900, the path loss in our protocol is high. Overall, our proposed solution achieves 24 percent improvements over the LAEEBA protocol by reducing the average path loss from 316.01 dB to 239.95 dB.

6 Conclusion and future work

WBAN and fog computing will pave the way for the deployment of several applications. One such application area is the provision of health-related services. Although the integration of WBAN and cloud computing brings several

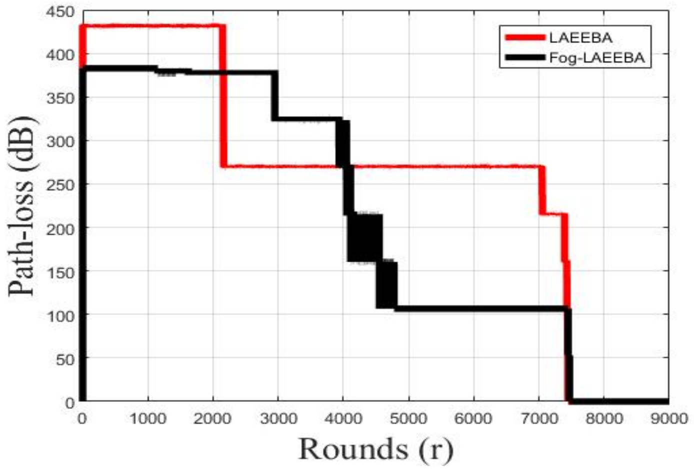


Figure 7: Path-loss.

advantages, one of the main problems with the existing cloud-based solutions is the latency involved in accessing, storing, and processing data. This limits the use of cloud computing for applications that require real-time access and processing of data. To address this challenge, we proposed Fog-LAEEBA, which follows the fog computing paradigm. Instead of sending the data to the cloud for processing, which involved delay and complexities, the sensory data would be processed locally on a fog unit. We implement, test, and evaluate the performance of Fog-LAEEBA in a simulation environment. For this purpose, we used a MATLAB simulator. Different parameters were used to evaluate the performance of our proposed solution, such as stability period, end-to-end delay, and throughput, etc. From the simulation results, we conclude that Fog-LAEEBA performs better in several ways. Fog-LAEEBA minimizes the end-to-end delay, involved in the LAEEBA protocol. In terms of energy efficiency, our proposed solution enhances the lifetime of the network. Moreover, we observed that Fog-LAEEBA brings a 24 percent improvement to the existing LAEEBA protocol by reducing the average path-loss from 316.01 dB to 239.95 dB.

In the future, we plan to test and deploy Fog-LAEEBA in real-world scenarios. Instead of using MATLAB simulations, we can deploy several medical sensors on the human body. Furthermore, we can also compare the results of our simulations with the results collected from real-world scenarios. This will help us to understand the gap between simulation and practical implementation setup. Security, privacy, safety, and trust cannot be eliminated from any research work. Another future direction would be the consideration of these aspects for safe and secure operations of the Fog-LAEEBA.

Finally, we will enhance the performance of the proposed solution by taking into consideration several other parameters, e.g., sending the collected information to a nearby hospital, or medical response teams.

References

[1] A. A. Mutlag, M. K. A. Ghani, N. Arunkumar, M. A. Mohammed, O. Mohd, Enabling technologies for fog computing in healthcare IoT systems, *Future Generation Computer Systems* **90,** (2019) 62–78. ⇒181

[2] A. Israa, A. Haider, S. Shihab, S. Siti, Identification key scheme to enhance network performance in wireless body area network, *Periodicals of Engineering and Natural Sciences (PEN)* **7,** 2 (2019) 895–906. ⇒182

[3] A. Md. Taslim, H. M. Haque, A.K.M Fazlul, Wireless Body Area Network: An Overview and Various Applications, *Journal of Computer and Communications* **5,** (2017) 53–64. ⇒181, 182

[4] B. Flavio, M. Rodolfo, Z. Jiang, A. Sateesh, Fog Computing and Its Role in the Internet of Things, *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, Helsinki, Finland, 2012, pp. 13–16. ⇒181

[5] E. Moghadas, J. Rezazadeh, R. Farahbakhsh, An IoT patient monitoring based on fog computing and data mining: Cardiac arrhythmia usecase, *Internet of Things* **11,** (2020) 2542–6605. ⇒183

[6] H. B. Hassen, W. Dghais, B. Hamdi, An E-health system for monitoring elderly health based on Internet of Things and Fog computing, *Health Information Science and Systems* **7,** 1 (2019) 1–9. ⇒183

[7] J. Kharel, H. T. Reda, S. Y. Shin, Fog Computing-Based Smart Health Monitoring System Deploying LoRa Wireless Communication, *IETE Technical Review* **36,** 1 (2019) 69–82. ⇒183

[8] K. Rani, N. Parma, Performance comparison of various routing protocols in WSN and WBAN, *2016 International Conference on Computing, Communication and Automation (ICCCA)*, Greater Noida, India, 2016, pp. 427–431. ⇒181

[9] M. R. Baig, N. Ullah, F. Hadi, S. Ahmed, A. Hanan, I. Ahmed, AnyCasting In Dual Sink Approach (ACIDS) for WBASNs, *International Journal of Advanced Computer Science and Applications* **8,** 3 (2017) 257–263. ⇒183

[10] S. Ahmed, N. Javaid, M. Akbar, A. Iqbal, Z. A. Khan, U. Qasim, LAEEBA: Link Aware and Energy Efficient Scheme for Body Area Networks, *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*, Victoria, BC, Canada, 2014, pp. 435–440. ⇒181, 183

[11] S. Ahmed, N. Javaid, S. Yousaf, A. Ahmad, M.M. Sandhu, M. Imran, Z.A. Khan, N. Alrajeh, Co-LAEEBA: Cooperative link aware and energy efficient protocol

for wireless body area networks, *Computers in Human Behavior* **51,** B (2019) 1205–1215. ⇒183

[12] S. Ahmed, N. Sadiq, K. Sadiq, N. Javaid, M. A. Taqi, *Node Density Analysis for WBAN Schemes in Terms of Stability and Throughput*, *Recent Trends and Advances in Wireless and IoT-enabled Networks*, Springer, Cham, 2019. ⇒181

[13] S. Saurabh, H. M. Fadzil, J. L. Tang, A. Azlan, K. M. Khalid, 3-Tier Architecture for Network Latency Reduction in Healthcare Internet-of-Things Using Fog Computing and Machine Learning, *Proceedings of the 2019 8th International Conference on Software and Computer Applications*, VPenang, Malaysia, 2019, pp. 522–528. ⇒182

[14] V. Prabal, S. Sandeep, Fog Assisted-IoT Enabled Patient Health Monitoring in Smart Homes, *IEEE Internet of Things Journal* **5,** 3 (2018) 1789–1796. ⇒181

Received: April 24, 2021 • Revised: June 1, 2021

Acta Universitatis Sapientiae

The scientific journal of Sapientia Hungarian University of Transylvania publishes original papers and surveys in several areas of sciences written in English. Information about each series can be found at http://www.acta.sapientia.ro.

Each volume contains two issues.

Sapientia University   Sciendo by De Gruyter   Scientia Publishing House

ISSN 1844-6086

http://www.acta.sapientia.ro

Information for authors

Acta Universitatis Sapientiae, Informatica publishes original papers and surveys in various fields of Computer Science. All papers are peer-reviewed.

Papers published in current and previous volumes can be found in Portable Document Format (pdf) form at the address: `http://www.acta.sapientia.ro`.

The submitted papers should not be considered for publication by other journals. The corresponding author is responsible for obtaining the permission of coauthors and of the authorities of institutes, if needed, for publication, the Editorial Board is disclaiming any responsibility.

Submission must be made by email (`acta-inf@acta.sapientia.ro`) only, using the LaTeX style and sample file at the address `http://www.acta.sapientia.ro`. Beside the LaTeX source a pdf format of the paper is necessary too.

Prepare your paper carefully, including keywords, ACM Computing Classification System codes (`http://www.acm.org/about/class/1998`) and AMS Mathematics Subject Classification codes (`http://www.ams.org/msc/`).

References should be listed alphabetically based on the Intructions for Authors given at the address `http://www.acta.sapientia.ro`.

Illustrations should be given in Encapsulated Postscript (eps) format.

Contact address and subscription:
Acta Universitatis Sapientiae, Informatica
RO 400112 Cluj-Napoca
Str. Matei Corvin nr. 4.
Email: acta-inf@acta.sapientia.ro

Printed by F&F INTERNATIONAL
Director: Enikő Ambrus

ISSN 1844-6086
http://www.acta.sapientia.ro